Global Constraint Catalog

2nd Edition (revision a)

Nicolas Beldiceanu[1]
TASC (CNRS/INRIA) Mines Nantes, FR-44307 Nantes, France

Mats Carlsson
SICS, Box 1263, SE-16 429 Kista, Sweden

Jean-Xavier Rampon
LINA, FR-44322 Nantes, France

Working version of SICS Technical Report T2012:03

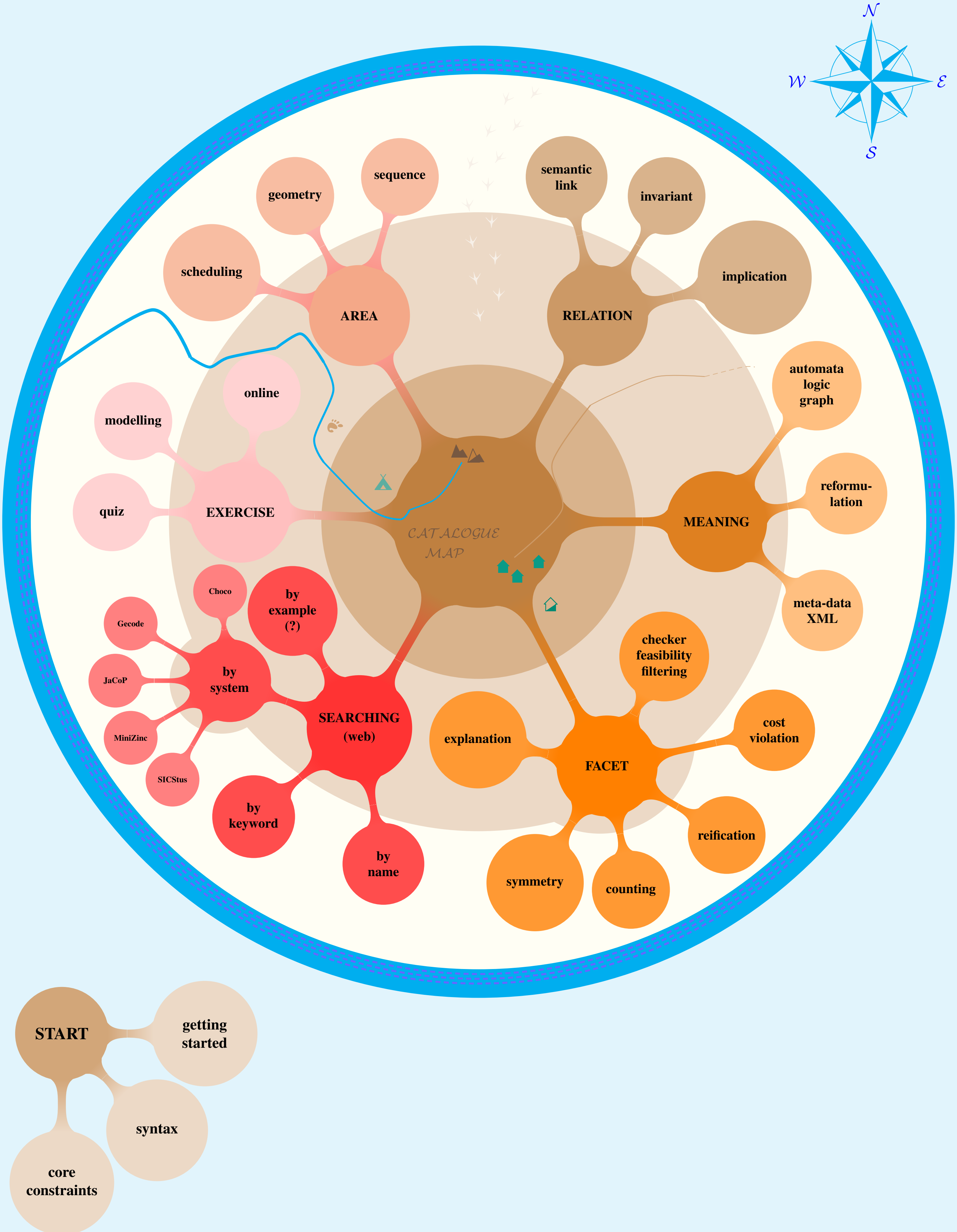
ISSN: 1100-3154

Abstract: This report presents a catalogue of global constraints where each constraint is explicitly described in terms of graph properties and/or automata and/or first order logical formulae with arithmetic. When available, it also presents some typical usage as well as some pointers to existing filtering algorithms.

Keywords: constraint programming, global constraint, catalogue, graph, automaton, first order formula, meta-data, ontology, symmetry, counting. ☞

December 14, 2014

[1]Corresponding author, Email: Nicolas.Beldiceanu@mines-nantes.fr


N
E
S
W
CATALOGUE MAP
AREA
sequence
geometry
scheduling
RELATION
semantic link
invariant
implication
MEANING
automata logic graph
reformu-lation
meta-data XML
FACET
checker feasibility filtering
explanation
cost violation
reification
symmetry
counting
SEARCHING (web)
by example (?)
by system
Choco
Gecode
JaCoP
MiniZinc
SICStus
by keyword
by name
EXERCISE
online
modelling
quiz
START
getting started
syntax
core constraints

Contents

Preface

> *"Although dictionaries give the impression of analyzing words all the way down to their very atoms, all they do in fact is graze their surfaces."*
> – Douglas Hofstadter, Emmanuel Sander, *Surfaces and Essences*

This catalogue presents a list of global constraints. Within this catalogue the term *global constraint* should be understood as an *expressive and concise condition involving a non-fixed number of variables*. This informal definition does not make any assumption neither about the potential use of a global constraint nor about the techniques[1] associated with the development of global constraints. It contains about 431 constraints, which are explicitly described in terms of graph properties and/or automata and/or first order logic formulae and/or conjunction of other constraints.

This *Global Constraint Catalogue* is an expanded version of the list of global constraints presented in [28] and an updated version of [43]. The principle used for describing global constraints has been slightly modified in order to deal with a larger number of global constraints. Since 2003, we try to provide an automaton that recognises the solutions associated with a global constraint. Since 2009, we also try to provide a first order logic formula for defining the solutions accepted by a geometrical constraint.

Writing a dictionary is a long process, especially in a field where new words are defined every year. In this context, one difficulty is to express explicitly the meaning of global constraints in terms of meta-data. Finding an appropriate and concise description that both easily captures the meaning of most global constraints and allows fast inferences[2] involving global constraints seems to be a tricky task.

One may wonder how so many constraints can be used at all in practice? However many fields produce a number of articles containing partial and specific results. Within the area of global constraints, we fill that trying extracting and classifying such knowledge, as well as providing meta-data for encoding it, may be a help, both for humans and machines, to exploit systematically ongoing research results and to put these results in perspective. Work about the *constraint seeker* [61] and the *model seeker* [62] relies on these meta-data to identify global constraints and to automatically come up with a constraint model from a set of positive samples.

[1] As quoted by J. N. Hooker in [225], "*identifying a field with its techniques is an intellectually as well as practically unsatisfying*" and has a lot of drawbacks.

[2] E.g., in the context of the constraint and model seeker [61, 62] we have gradually identified a number of properties for inferring that a constraint/conjunction of constraints is implied by another constraint/conjunction of constraints.

Goal of the catalogue. This catalogue has four main goals. First, it provides an overview of most of the different global constraints that were gradually introduced in the area of constraint programming since the work of J.-L. Laurière on ALICE [267]. It also identifies new global constraints for which no existing published work exists. The global constraints are arranged in alphabetic order, and for all of them a description and an example are systematically provided. When available, it also presents some typical usage as well as some pointers to existing filtering algorithms.

Second, the global constraints described in this catalogue are not only accessible to humans, who can read the catalogue for searching for some information. It is also available to machines, which can read and interpret it. This is why there exists an electronic version of this catalogue where one can get, for most global constraints, a complete description in terms of meta-data. In fact, most of this catalogue and its figures were automatically generated from this electronic version by a computer program. This description is based on three complementary ways to look at a global constraint. The first one defines a global constraint as searching for a graph with specific properties [27], the second one characterises a global constraint in terms of an automaton that only recognises the solutions associated with that global constraint [39, 317][3], while the third one defines in the context of geometric constraints a global constraint as a restricted first order logic formula [107]. The key point of these descriptions is their ability to define explicitly in a concise way the meaning of most global constraints. In addition these descriptions can also be systematically turned into polynomial filtering algorithms.

Third, we hope that this unified description of apparently diverse global constraints will allow for establishing a systematic link between the properties of basic concepts used for describing global constraints and the properties of the global constraints as a whole.

Finally, we also hope that it will attract more people from the algorithmic community into the area of constraints. To a certain extent this has already started in places like CWI in Amsterdam, the Max-Planck für Informatik (Saarbrücken) or the university of Waterloo. We also hope that it will attract people from combinatorics in order to produce theories and knowledge that could nicely unify and/or put in perspective different aspects of constraints (i.e., breaking symmetries, counting the number of solutions). Identifying bijections [408, 407] relating global constraints to well known combinatorial objects would be a step in this direction.[4]

Use of the catalogue. The catalogue is organised into five chapters:

- Chapter 1 provide a short overview of the main entries you may first consult when you are not familiar with the catalogue.

[3] Automata were first used in the 90ies by N. R. Vempaty [442] and J. Amilhastre [8] in the context of constraint networks. Later on in 2007, they were also used by M.-C. Coté *et al.* [134] in the context of linear programming.

[4] This is currently the case only for a few constraints like ALLDIFFERENT, CYCLE or TREE. The information about the number of solutions to a global constraint in the **Counting** slot could certainly help identifying relevant combinatorial objects.

- Chapter 2 explains how the meaning of global constraints is described in terms of graph-properties or in terms of automata. On the one hand, if one wants to consult the catalogue for getting the informal definition of a global constraint, examples of use of that constraint or pointers to filtering algorithms, then one only needs to read the first section of Chapter 2: describing the arguments of a global constraint, page 14. On the other hand, if one wants to understand those entries describing explicitly the meaning of a constraint then all the material of Chapter 2 is required.

- Chapter 3 describes the content of the catalogue as well as different ways for searching through the catalogue. This material is essential.

- Chapter 4 covers additional topics, such as the differences from the 2000 report [28] on global constraints, the generation of implied constraints that are systematically linked to the graph-based description of a global constraint, and the electronic version of the catalogue. The material describing the format of the entries of a global constraint is mandatory for those who want to exploit the electronic version in order to write pre-processors for performing various tasks for a global constraint.

- Finally, Chapter 5 corresponds to the catalogue itself, which gives the global constraints in alphabetical order.

Acknowledgments. Nicolas Beldiceanu was the principal investigator and main architect of the constraint catalogue, provided the main ideas, and wrote a checker for the constraint descriptions, a figure generation program for the constraint descriptions and an evaluator for most constraints. Jean-Xavier Rampon provided the proofs for the graph invariants. Mats Carlsson contributed to the design of the meta-data format, generated some of the automata together with their negated forms, provide some constraints evaluators, and wrote the program that created the LaTeX version of this catalogue from the constraint descriptions.

The idea of describing explicitly the meaning of global constraints in a declarative way has been inspired by the work on meta-knowledge of Jacques Pitrat [327, 328, 329].

We are grateful to Magnus Ågren, Abderrahmane Aggoun, Ernst Althaus, María Andreína Francisco Rodríguez, Gregor Baues, Christian Bessière, Mohammed Haykel Boukadida, Éric Bourreau, Sebastian Brand, Pascal Brisset, Hadrien Cambazard, Gilles Chabert, Peter Chan, Philippe Charlier, François Clautiaux, Evelyne Contejean, Radoslaw Cymer, Romuald Debruyne, Frédéric Deces, Sophie Demassey, Alban Derrien, Mehmet Dincbas, Grégoire Dooms, François Fages, Jean-Guillaume Fages, Pierre Flener, Xavier Gandibleux, Yan Georget, Dávid Hanák, Emmanuel Hebrard, Pascal Van Hentenryck, Fabien Hermenier, Han J. A. Hoogeveen, Stefan Hougardy, Giuseppe F. Italiano, Antoine Jouglet, Narendra Jussien, Irit Katriel, Waldemar Kocjan, Per Kreuger, Krzysztof Kuchcinski, Mikael Zayenz Lagerkvist, Michel Leconte, Christophe Lecoutre, Arnaud Letort, Xavier Lorca, Michael J. Maher, Michael Marte, Julien Martin, Julien Menana, Dominique Michelucci, Per Mildner, Nicolas Museux, Justin Pearson, Gilles Pesant, Thierry Petit, Emmanuel Poder, Charles Prud'homme, Luis Quesada, Claude Guy Quimper, Jean-Charles Régin, Florian Richoux, Guillaume Rochart, Xavier Savalle, Pierre Schaus, Thomas Schiex, Christian Schulte, Helmut Simonis, Péter Szeredi, Radoslaw Szymanek, Guido Tack, Sven Thiel, Charlotte Truchet, Eliane Vacheret, Willem-Jan van Hoeve, Richard J. Wallace, Toby Walsh and Stéphane Zampelli for correction, discussion, information exchange or common work about specific global constraints.

Furthermore, we are grateful to Irit Katriel who contributed by updating the description of some filtering algorithms related to flow and matching of the catalogue, to Luis Quesada and Stéphane Zampelli who provide inputs for the DOM_REACHABILITY, the SUBGRAPH_ISOMORPHISM and GRAPH_ISOMORPHISM constraints, and to Radoslaw Szymanek and Guido Tack for providing the correspondence of global constraints of the catalogue with the constraints of **JaCoP** and **Gecode**. We are also especially grateful to Sophie Demassey both, for creating the on-line version of the catalogue (`http://sofdem.github.io/gccat/`) and for writing down the entry related to the cumulative longest hole problems, to Helmut Simonis both, for designing the XML schema (see Section 4.5.2) for the global constraints and their arguments, for providing the corresponding generation programs, for providing data for several rectangle placement problems, and for providing the code for generating the figures that display the number of solutions to a constraint wrt some of their parameters, and to Pierre Flener and Justin Pearson for providing feedback with respect to the **Symmetry** slot of global constraints. Moreover Eliane Vacheret helps for structuring many exercises of the **Quiz** slot of the core constraints and did a first version of the online version constraint course, and Pierre Flener provides additional feedback on the exercises on the core constraints.

The geometric constraints GEOST and VISIBLE as well as the constraints related to the Region Connection Calculus where integrated within the catalogue while working on the European Union Sixth Framework Programme Contract FP6-034691 "Net-WMS".

The glue matrices of the automata were first worked out by María Andreína Francisco Rodríguez and Nicolas Beldiceanu, and latter on systematically tested with a program by Mats Carlsson.

Finally, we want to acknowledge the continuing support of SICS and EMN for providing excellent working conditions over the years. The part of this work related to graph properties in Chapter 5 was done while the corresponding author was working at SICS.

Readers may send their suggestions via email to the corresponding author with `catalogue` as subject.

Uppsala, Sweden, August 2003 – Nantes, France, December 2014 — NB, MC, JXR

1

Getting started

If you are using the pdf version of the catalogue use Adobe Reader if you want to be sure to see PDF annotations.[1] If you do not see on your screen a small yellow bullet at the beginning of this paragraph, you are using a PDF viewer that does not fully support PDF annotations. Within keywords and constraints, the icons

indicates a point of interest (e.g., a necessary condition, a typical use),

denotes a typical error or a common misunderstanding.

The main entries you may consult if you want to have a first look to the catalogue are:

- To get an idea of the multiple facets of global constraints look at Section 2.1.
- To get an idea of how global **constraint arguments** are described look at Section 2.2.
- To **search in the catalogue** look at Section 3.3.
- To **search a constraint from a keyword** look at Section 3.7.
- To get an idea **how keywords are structured** look at Section 3.6.
- To know **available semantic links between constraints** look at Section 2.6.
- To get through the **core global constraints** look at the keyword core. Most core global constraints have a small set of exercises with solutions for checking that different facets of a core constraint are well understood. These exercises are located in the **Quiz** slot at the end of a constraint catalog entry, e.g. for ALLDIFFERENT see page 557.
- To see how **constraints symmetries** are described look at Section 2.2.5.

[1] Since we are using the LaTeX package pdfcomment and since most PDF viewers do not support PDF annotations.

- To get an idea of **general filtering techniques** look at the meta-keyword filtering and more specifically to the entries Berge-acyclic constraint network, constructive disjunction, flow and sweep. To get the notion of **consistency** achieved by a filtering algorithm look at the keywords arc-consistency and bound-consistency.

- To get an idea of **modelling techniques** and of **modelling exercises** look at the meta-keywords modelling and modelling exercises.

- To get and idea of **reformulations** of global constraints look at Section 2.5.

- To get an idea of general ways to **explicitly represent the meaning of global constraints** look at (a) Section 2.3 for the *graph property based description*, (b) Section 2.4 for the *automaton based description*, (c) the reference [107]) for the *logical based description* (e.g., see the **Logic** slot of MEET_SBOXES).

- To get an idea of the **meta-data** used for describing various aspects of a constraint look at Section 4.5.1 for *the facts* and Section 4.5.2 for the *XML schema*.

- To get the **correspondence of global constraints** of the catalogue **with concrete constraint systems or modelling languages**, such as **Choco**, **Gecode**, **JaCoP**, **MiniZinc**, or **SICStus** look at Appendix C.

Some material of the global constraint catalogue may be used for teaching global constraints, namely:

- Section 2.1 to get a first idea about global constraints,

- The definition and use of the core global constraints starting with ALLDIFFERENT and ELEMENT, continuing with CUMULATIVE, DIFFN and CYCLE, and finishing with GLOBAL_CARDINALITY, NVALUE, and SORT. Exercises for these constraints are located in the **Quiz** slot at the end of a constraint catalog entry. Exercises involving more than one constraint are available from the keyword Modelling exercises. All exercises are listed at the end of the catalogue page 3930.

Moreover additional on-line material that could also be used for teaching is:

- An online version constraint course focussing on core global constraints is available where exercises can be done interactively.

- The constraint seeker allows users searching for global constraints, given positive and negative, fully instantiated examples without knowing neither the names of the constraints nor the way they arguments are organised (see the on-line help for using that tool).

2

Describing Global Constraints

Contents

We first introduce the notion of global constraint as well as the different facets attached to global constraints. We then motivate the need for an explicit description of global constraints and then present the *graph-based* as well as the *automaton-based* descriptions used throughout the catalogue. On the one hand, the graph-based representation considers a global constraint as a subgraph of an initial given graph. This subgraph has to satisfy a set of required graph properties. On the other hand, the automaton-based representation denotes a global constraint as a hypergraph constructed from a given constraint checker.[1] Both, the initial graph of the graph-based representation, as well as the hypergraph of the automaton-based representation have a very

[1] A *constraint checker* is a program that takes an instance of a constraint for which all variables are fixed and tests whether the constraint is satisfied or not.

regular structure, which should give the opportunity for efficient filtering algorithms taking advantage of this structure.

We now present our motivations for an explicit description of the meaning of global constraints. The current trend[2] is to first use natural language for describing the meaning of a global constraint and second to work out a specialised filtering algorithm. Since we have a huge number of potential global constraints that can be combined in a lot of ways, this is an immense task. Since we are also interested in providing other services, such as *visualisation* [463, 397, 400], *explanations* [371], *cuts for linear programming* [227], *moves for local search* [88], *generation of clauses for SAT solvers* [306], *generation of multivalued decision diagrams that represent compact relaxations of global constraints* [224], *soft global constraints* [325, 57, 435], *learning implied global constraints* [66], simplifying away fixed variables from global constraints when they have the same effect on the remaining unfixed variables in order to automatically identify equivalent subproblems during search [124], and *specialised heuristics* for each global constraint this is even worse. One could argue that a candidate for describing explicitly the meaning of global constraints would be second order predicate calculus. This could perhaps solve our description problem but would, at least currently, not be useful for deriving any filtering algorithm.[3] For a similar reason Prolog was restricted to Horn clauses for which one had a reasonable solving mechanism. What we want to stress through this example is the fact that a declarative description is really useful only if it also provides some hints about how to deal with that description. Our first choice of a graph-based representation has been influenced by the following observations:

- The concept of graph has its roots in the area of mathematical recreations (see, for example, L. Euler [164], H. E. Dudeney [155], E. Lucas [280] and T. P. Kirkman [249]), which was somehow the ancestor of combinatorial problems. In this perspective a graph-based description makes sense.

- In one of the first books introducing graph theory [63], C. Berge presents graph theory as a way of grouping apparently diverse problems and results. This was also the case for global constraints.

- The parameters associated with graphs are concrete and concise. Moreover a lot of results about graphs can be expressed in terms of graph invariants involving various graph parameters that are valid for specific graph classes. In essence, formulae are a kind of declarative statement that is much more compact than algorithms.

- Finally, it is well known that graph theory is an important tool [291] with respect to the development of efficient filtering algorithms [351, 353, 356, 364, 292, 244, 54, 433, 344].

Our second choice of an automaton-based representation has been motivated by the following observation. Writing a constraint checker is usually a straightforward

[2] This can be noted in all constraint manuals where the description of the meaning is always informal.

[3] One could perhaps use a system like MONA [221] or some ideas from [91] for getting a constraint checker in the context of the graph-based representation.

task. The corresponding program can usually be turned into an automaton. Of course an automaton is typically used on a fixed sequence of symbols. But, in the context of filtering algorithms, we have to deal with a sequence of variables. For this purpose we have shown [39] for some automata how to decompose them into a conjunction of smaller constraints. In this context, a global constraint can be seen as a hypergraph corresponding to its decomposition. The hypergraph has two types of hyperedges: the first type corresponds to *transition constraints* describing the behavior of the automaton, while the second type represents *signature constraints* encoding the mapping of arguments of the constraint to symbols of the alphabet of the automaton.

2.1 Global constraint: what it is, and what it is not

As said in the preface, the term global constraint should be understood as an *expressive* and *concise* condition involving a non-fixed number of partially known objects. The ideas conveyed by this informal definition are:

- *Context independence* points to the fact that the condition may be typically useful in more than one context. Like a word in the context of natural language, a global constraint encapsulates its own concept that is *independent from any specific use*. Like words, once you have the right concepts, you can directly reason at the appropriate level of abstraction.

- *Conciseness* means that the condition should be expressible in a compact way, typically by one or two sentences in natural language.[4] The condition directly mentions objects that match the *right level of abstraction* it considers, i.e., when appropriate it does not just refer to a flat list of variables. If necessary, the condition typically involves one or several collections of objects (e.g. tasks), where each object has a number of attributes (e.g., an origin, a duration and an end if we consider tasks). An attribute may correspond either to a constant (i.e., a fixed value), or to a variable.

The second key point conveyed by this informal definition is that, *it does not make any assumption about the techniques associated with the potential use of global constraints*. Consequently, a global constraint cannot just be reduced to:

1. A concept that is only linked to constraint programming.

2. A function that computes a result from a given set of input arguments.

3. An algorithm that, given a condition on a set of variables, removes some values to enforce that condition.

4. A way to express a condition as a set of clauses or as a set of linear constraints.

[4]In Sections 2.3 and 2.4 we will describe some possible ways of defining concisely the meaning of a significant number of global constraints in a formal way.

We now illustrate through the example of the ALLDIFFERENT constraint different facets of a global constraint. Given a collection of variables $\langle x_1, x_2, \ldots, x_n\rangle$, the ALLDIFFERENT$(\langle x_1, x_2, \ldots, x_n\rangle)$ constraint forces all variables $x_1, x_2, \ldots, x_n$ to be assigned distinct values. We successively consider the *checker* view, the *feasibility* view, the *filtering* view, the *explanation* view, the *cost violation* view, the *reification* view, the *counting* view and the *property* view.

2.1.1 Checker view

Considering a constraint for which all variables are fixed, the checker view is about finding an algorithm that checks whether a ground instance of that constraint holds or not. In the context of learning models, the usage of dedicated checkers rather than general filtering algorithms is crucial for performance issues.

For the ALLDIFFERENT$(\langle x_1, x_2, \ldots, x_n\rangle)$ constraint one can first sort the sequence $x_1, x_2, \ldots, x_n$ and then check that adjacent values are distinct, or alternatively, insert each value into a hash table in order to check that no value occurs more than once.

2.1.2 Feasibility view

Given a constraint where not all variables are fixed yet, a question is whether that constraint has at least one solution or not. We assume that all the not yet fixed variables must be assigned a value in a finite set of values. In this context we are looking for a necessary, and possibly sufficient, condition that can be evaluated in polynomial time with respect to the size of the arguments of that constraint.

For the ALLDIFFERENT constraint we associate a variable-value graph where (1) each vertex corresponds to its variables and to the values that can be assigned to these variables, and (2) each edge corresponds to the fact that a variable can be assigned a given value. A necessary and sufficient condition is that the cardinality of the maximum matching, i.e. the maximum number of edges such that no two edges have a vertex in common, in this variable-value graph is equal to the number of variables.

2.1.3 Filtering view

Once we know that a constraint for which not all variables are fixed yet is potentially feasible, the next question is to identify variable-value pairs of the form (var, val) such that, if value val is assigned to variable var, the constraint has no solution. Since removing such values reduces a priori the search space, filtering is strongly supported by the implicit motto of constraint programming that *the more you prune the better*. Assuming that you already have a necessary and sufficient condition that can be evaluated in polynomial time for checking whether a constraint has at least one solution or not, you can directly reuse it for checking whether a value can be assigned or not to a variable. Since the number of variable-value pairs to check may be quadratic with respect to the total number of variables and values one usually prefers developing a dedicated filtering algorithm that is less costly than checking the feasibility condition on each variable-value pair.

For the ALLDIFFERENT constraint a first filtering algorithm [351] is based on a characterisation of the edges of the variable-value graph that belong to a maximum matching but not to all. A *matching* on a graph is a set of edges of the graph such that no two edges have a vertex in common; it is *maximum* if its number of edges is maximum. We first introduce a digraph $\overrightarrow{G}_M$ that is associated with a matching M that matches all variables of the ALLDIFFERENT constraint. The vertices of $\overrightarrow{G}_M$ are defined as the variables and the values that can be assigned to the variables of the ALLDIFFERENT constraint. To each value val that can be assigned to a variable var corresponds an arc of $\overrightarrow{G}_M$ from var to val. Finally, if value val is matched to variable var in the matching M we add the reverse arc from val to var to the arcs of $\overrightarrow{G}_M$. Now a variable var can be assigned a value val if and only if var and val belong to the same strongly connected component of $\overrightarrow{G}_M$ or if there is a path consisting of distinct vertices and arcs of $\overrightarrow{G}_M$ that start with (var, val) and ends up in an unmatched value with respect to matching M, see [417, page 29].

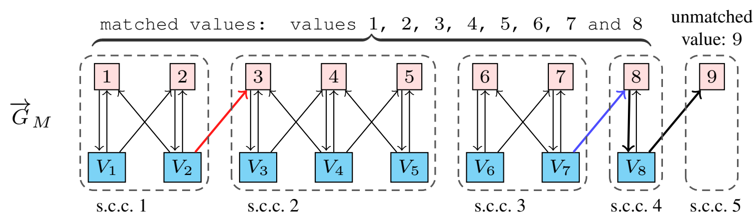


Figure 2.1: Illustration of the filtering for ALLDIFFERENT$(\langle V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8\rangle)$ with $V_1 \in [1,2]$, $V_2 \in [1,3]$, $V_3 \in [3,4]$, $V_4 \in [3,5]$, $V_5 \in [4,5]$, $V_6 \in [6,7]$, $V_7 \in [6,8]$, $V_8 \in [8,9]$ with respect to the maximum matching M defined by $V_i = i$ $(1 \leq i \leq 8)$ and the corresponding digraph and its five strongly connected components s.c.c. k $(1 \leq k \leq 5)$: 8 can be assigned to V_7 (blue arc) since the path $(V_7\ 8\ V_8\ 9)$ ends up in an unmatched value, but 3 cannot be assigned to V_2 (red arc) since 3 and V_2 do not belong to the same strongly connected component and since there is no path from V_2 to the unique unmatched value 9.

Another filtering algorithm for ALLDIFFERENT based on Hall intervals just focuses on adjusting the minimum and maximum values of the variables. Given a set of domain variables, a *Hall interval* is an interval of values $[low, up]$ denoted by $\mathcal{H}_{[low,up]}$ such that there are $up - low + 1$ variables whose domains are contained in $\mathcal{H}_{[low,up]}$. A *minimal Hall interval* $\mathcal{H}_{[low,up]}$ is a Hall interval that does not contain any Hall interval $\mathcal{H}_{[low',up']}$ such that either $low = low'$ or $up = up'$. Given a Hall interval $\mathcal{H}$ and a variable V whose domain is not included in $\mathcal{H}$ but intersects $\mathcal{H}$, the idea is to adjust the minimum (respectively maximum) value of variable V to the smallest (respectively largest) value that does not belong to $\mathcal{H}$. Figure 2.2 illustrates this idea on the constraint ALLDIFFERENT$(\langle V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9\rangle)$ with $V_1 \in [1,2]$, $V_2 \in [1,2]$, $V_3 \in [2,5]$, $V_4 \in [4,5]$, $V_5 \in [5,6]$, $V_6 \in [4,6]$, $V_7 \in [1,9]$, $V_8 \in [8,9]$, $V_9 \in [8,9]$.

- Part (A) of Figure 2.2 shows in light blue, in light pink and in light yellow the three minimal Hall intervals associated with the initial domains of variables $V_1, V_2, \ldots, V_9$.

  - $\mathcal{H}_{[1,2]}$ corresponds to interval $[1,2]$, which contains the domains of V_1 and V_2.
  - $\mathcal{H}_{[4,6]}$ corresponds to interval $[4,6]$, which contains the domains of V_4, V_5 and V_6.
  - $\mathcal{H}_{[8,9]}$ corresponds to interval $[8,9]$, which contains the domains of V_8 and V_9.

- Part (B) of Figure 2.2 shows the first propagation step with respect to the Hall intervals $\mathcal{H}_{[1,2]}$, $\mathcal{H}_{[4,6]}$ and $\mathcal{H}_{[8,9]}$. First note that variables V_3 and V_7 are the only variables whose domain is not included in a Hall interval. Consequently V_3 and V_7 are candidates for adjusting their minimum or maximum domain values.

  - Since the minimum value of V_3, that is i.e. value 2, belongs to the Hall interval $\mathcal{H}_{[1,2]}$ we adjust the minimum of V_3 to the smallest value that does not yet belong to any Hall interval, i.e. value 3.
  - Since the maximum value of V_3, that is i.e. value 5, belongs to the Hall interval $\mathcal{H}_{[4,6]}$ we adjust the maximum value of V_3 to the largest value that does not yet belong to any Hall interval, i.e. value 3.
  - Since the minimum value of V_7, that is i.e. value 1, belongs to the Hall interval $\mathcal{H}_{[1,2]}$ we adjust the minimum of V_7 to the smallest value that does not yet belong to any Hall interval, i.e. value 3.
  - Since the maximum value of V_7, that is i.e. value 9, belongs to the Hall interval $\mathcal{H}_{[8,9]}$ we adjust the maximum of V_7 to the largest value that does not yet belong to any Hall interval, that is i.e. value 7.

- Part (C) of Figure 2.2 shows the second propagation step with respect to the new Hall interval $\mathcal{H}_{[3,3]}$. Now V_7 is the only variable whose domain is not included in a Hall interval so it is a candidate for adjusting its minimum or maximum domain value.

  - Since the minimum value of V_7, i.e. value 3, belongs to the new Hall interval $\mathcal{H}_{[3,3]}$ we adjust the minimum of V_7 to the smallest value that does not yet belong to any Hall interval, i.e. value 7.

- Finally, part (D) of Figure 2.2 shows all the minimal Hall intervals after reaching the fix point of the filtering, where $\mathcal{H}_{[7,7]}$ is a new minimal Hall interval.

2.1.4 Explanation view

Given a constraint that cannot be satisfied the goal is to identify a smallest subset of variables and values for explaining that the constraint has no solution. Similarly, given a constraint that can be satisfied and a variable-value pair (var, val) such that, if value val is assigned to variable var the constraint has no solution, the same question arises.

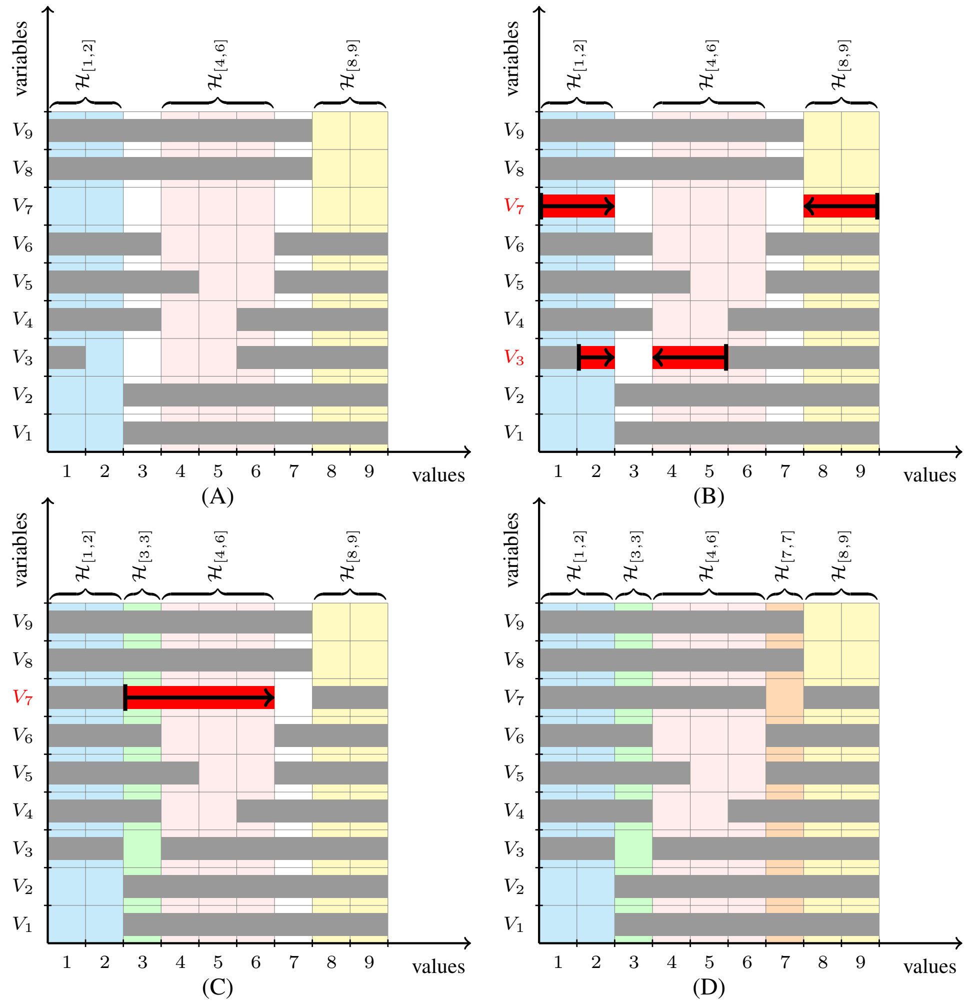


Figure 2.2: Steps (A), (B), (C) and (D) for filtering the minimum and maximum values wrt Hall intervals $\mathcal{H}_{[low,up]}$ for ALLDIFFERENT$(\langle V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9\rangle)$ with $V_1 \in [1,2]$, $V_2 \in [1,2]$, $V_3 \in [2,5]$, $V_4 \in [4,5]$, $V_5 \in [5,6]$, $V_6 \in [4,6]$, $V_7 \in [1,9]$, $V_8 \in [8,9]$, $V_9 \in [8,9]$. Each horizontal grey strip corresponds to a set of consecutive values that do not belong to the domain of a variable, while each horizontal red strip represents an adjustment of the minimum or the maximum value of the domain of a variable.

Explanations are expressed in term of values that should be added to the domains of some variables in order to prevent unsatisfiability or filtering.

For the ALLDIFFERENT constraint Figure 2.3 provides the explanation attached to the instance described in Figure 2.1, i.e. what arcs should be added to prevent value 3 from being removed from the domain of variable V_2.

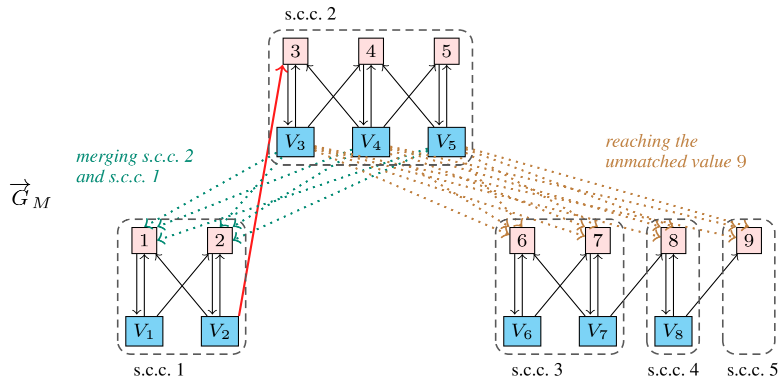


Figure 2.3: Illustration of the explanation why variable V_2 cannot be assigned value 3 for ALLDIFFERENT$(\langle V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8\rangle)$ with $V_1 \in [1,2]$, $V_2 \in [1,3]$, $V_3 \in [3,4]$, $V_4 \in [3,5]$, $V_5 \in [4,5]$, $V_6 \in [6,7]$, $V_7 \in [6,8]$, $V_8 \in [8,9]$ wrt the maximum matching M defined by $V_i = i$ $(1 \leq i \leq 8)$ and the corresponding digraph: (1) since adding any dotted green arc from V_3, V_4, V_5 to values 1 or 2 merges the strongly connected components containing V_2 and 3 this would prevent 3 from being removed from V_2; (2) since adding any dotted brown arc from V_3, V_4, V_5 to values 6, 7, 8 or 9 allows V_3 to reach the unmatched value 9 this would also prevent 3 from being removed from V_2.

2.1.5 Cost violation view

Considering a constraint for which all variables are fixed such that the constraint does not hold, a question is to evaluate the degree of violation of that constraint assuming that for a satisfied constraint the degree of violation is equal to zero.

For the ALLDIFFERENT constraint we can define its degree of violation by the minimum number of variables to reassign in order to get a solution. This is called the *variable-based degree of violation*. We can also define its degree of violation by the number of pairs of variables that do not satisfy the disequality constraint. This is called the *decomposition-based degree of violation*. Figure 2.4 illustrates these two degree of violation costs.

2.1.6 Reification view

Suppose we want to associate a 0-1 domain variable b to a constraint $\mathcal{C}$ and maintain the equivalence $b \equiv \mathcal{C}$. This is called the reification of $\mathcal{C}$. For most global constraints this can be achieved by reformulating the global constraint as a conjunction of pure functional dependency constraints together with constraints that can be easily reified, e.g. linear constraints involving at most two variables [37].

We can reify the ALLDIFFERENT$(\langle x_1, x_2, \ldots, x_n\rangle)$ constraint by using the idea of sorting its variables (i.e., the pure functional dependency part) and by stating that within the sorted list of variables adjacent variables are in strictly increasing order. This

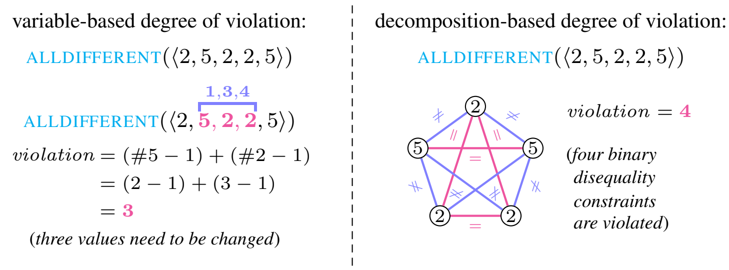


Figure 2.4: Illustration of the variable-based and the decomposition-based degrees of violation for ALLDIFFERENT$(\langle 2,5,2,2,5\rangle)$, where $\#v$ denotes the number of occurrences of value v in the argument of the ALLDIFFERENT constraint instance

leads to the following expression SORT$(\langle x_1, x_2, \dots, x_n\rangle, \langle y_1, y_2, \dots, y_n\rangle) \ \wedge \ (y_1 < y_2 \wedge y_2 < y_3 \wedge \dots \wedge y_{n-1} < y_n) \equiv b$.

2.1.7 Counting view

Considering a constraint for which not all variables are fixed yet, a question is to count, or estimate, its number of solutions. This is useful for writing heuristics that take into account the tightness of the constraints in order, for example, to select the next variable to assign. Considering a pure functional dependency constraint it is interesting to consider how the number of solutions to that constraint varies depending on the value of the pure functional dependency parameter (e.g., in the context of the NVALUE(`N`, `VARIABLES`) constraint, its number of solutions if extremely low when `N` $= 1$, then increase as `N` increases up to a point where it decreases again and ends up with `N` $= |$`VARIABLES`$|$ like an ALLDIFFERENT). This is useful, for example, for ranking pure functional dependency constraints in the context of the constraint seeker [61].

Counting the number of solutions to an ALLDIFFERENT constraint is equivalent to counting the number of maximum matchings in a bipartite graph, which is #*P*-complete [424]. Consequently faster approximations for estimating the number of solutions are used in practice [459].

2.1.8 Property view

If we want to reason about constraints we need to know their properties. Examples of properties of a constraint with respect to one of its arguments are:

- A constraint $c(\langle x_1, x_2, \dots, x_n\rangle)$ is *contractible* if each ground satisfied instance $c(\langle val_1, val_2, \dots, val_n\rangle)$ is still satisfied if we remove any of its val_i $(1 \leq i \leq n)$.

- A constraint $c(\langle x_1, x_2, \dots, x_n\rangle)$ is *extensible* if each ground satisfied instance $c(\langle val_1, val_2, \dots, val_n\rangle)$ is still satisfied if we insert any value into the sequence $val_1, val_2, \dots, val_n$.

Examples of properties of a constraint with respect to another constraint are:

- The fact that a constraint implies another constraint under the assumption that both constraints have exactly the same arguments.
- The fact that if a constraint holds then another constraint does not hold, under the assumption that both constraints have exactly the same arguments.

Implications between two different constraints are provided by the **See also** slot (i.e., see *implies*, *implied by* and *negation*) as well as by the **Conditional implication** slot.

Regarding the ALLDIFFERENT constraint we have:

- It is contractible since removing a value from a ground satisfied instance leaves the remaining constraint satisfied.
- It is not extensible since adding a value into a ground satisfied instance may not necessarily lead to a satisfied constraint.
- The ALLDIFFERENT constraint is implied by more specific constraints, such as the ALLDIFFERENT_CONSECUTIVE_VALUES constraint, which in addition forces all used values to be consecutive.

We now illustrate how to use such knowledge as a method for trying to check that a given conjunction of identical constraints, i.e. constraints with the same name, is implied by another conjunction of also identical constraints. This subproblem originates from learning models where we want to keep only the more general conjunctions of identical constraints. Consider the conjunction $\mathcal{C}_1$

$$\left\{\begin{array}{l}\text{ALLDIFFERENT\_CONSECUTIVE\_VALUES}(\langle v_1, v_2, v_3, v_4\rangle) \wedge \\ \text{ALLDIFFERENT\_CONSECUTIVE\_VALUES}(\langle v_5, v_6, v_7, v_8\rangle)\end{array}\right.$$

where ALLDIFFERENT_CONSECUTIVE_VALUES forces variables to take consecutive distinct values, and the conjunction $\mathcal{C}_2$

$$\left\{\begin{array}{l}\text{ALLDIFFERENT}(\langle v_1, v_2\rangle) \;\wedge\; \text{ALLDIFFERENT}(\langle v_3, v_4\rangle) \wedge \\ \text{ALLDIFFERENT}(\langle v_5, v_6\rangle) \;\wedge\; \text{ALLDIFFERENT}(\langle v_7, v_8\rangle)\end{array}\right.$$

Conjunction $\mathcal{C}_1$ implies conjunction $\mathcal{C}_2$ since every constraint of $\mathcal{C}_2$ is implied by at least one constraint of $\mathcal{C}_1$. For example, ALLDIFFERENT$(\langle v_1, v_2\rangle)$ is implied by ALLDIFFERENT_CONSECUTIVE_VALUES$(\langle v_1, v_2, v_3, v_4\rangle)$ since:

- ALLDIFFERENT_CONSECUTIVE_VALUES$(\langle v_1, v_2, v_3, v_4\rangle)$ $\Rightarrow$ ALLDIFFERENT$(\langle v_1, v_2, v_3, v_4\rangle)$.
- ALLDIFFERENT$(\langle v_1, v_2, v_3, v_4\rangle)$ is contractible, it implies ALLDIFFERENT$(\langle v_1, v_2\rangle)$.

2.2 Describing the arguments of a global constraint

Since global constraints have to receive their arguments in some form, no matter whether we use the graph-based or the automaton-based description, we start by describing the abstract data types that we use in order to specify the arguments of a global constraint. These abstract data types are not related to any specific programming language like Caml, C, C++, Java or Prolog. If one wants to focus on a specific language, then one has to map these abstract data types to the data types that are available within the considered programming language. In a second phase we describe all the restrictions that one can impose on the arguments of a global constraint. Finally, in a third phase we show how to use these ingredients in order to declare the arguments of a global constraint.

2.2.1 Basic data types

We provide the following *basic data types*:

- `atom` corresponds to an atom. Predefined atoms are `MININT` and `MAXINT`, which respectively correspond to the *smallest* and to the *largest integer*.
- `int` corresponds to an *integer value*.
- `dvar` corresponds to a *domain variable*. A *domain variable* V is a variable that will be assigned an *integer* value taken from an initial finite set of integer values denoted by $dom(V)$. $\underline{V}$ and $\overline{V}$ respectively denote the minimum and the maximum values of $dom(V)$.
- `fdvar` corresponds to a *possibly unbounded domain variable*. A *possibly unbounded domain variable* is a variable that will be assigned an *integer* value from an initial finite set of integer values denoted by $dom(V)$ or from interval minus infinity, plus infinity. This type is required for declaring the domain of a variable. It is also required by some systems in the context of specific constraints like arithmetic or ELEMENT constraints.
- `sint` corresponds to a *finite set of integer values*.
- `svar` corresponds to a *set variable*. A *set variable* V is a variable that will be assigned to a *finite set* of integer values. Its *lower bound* $\underline{V}$ denotes the set of integer values that for sure belong to V, while its *upper bound* $\overline{V}$ denotes the set of integer values that may belong to V. $dom(V) = \{\mathbf{v_1}, \ldots, \mathbf{v_n}, v_{n+1}, \ldots, v_m\}$ is a shortcut for combining the lower and upper bounds of V in a single notation:
  - Bold values designate those values that only belong to $\underline{V}$.
  - Plain values indicate those values that belong to $\overline{V}$ and not to $\underline{V}$.
- `mint` corresponds to a *multiset of integer values*.
- `mvar` corresponds to a *multiset variable*. A *multiset variable* is a variable that will be assigned to a *multiset of integer values*.

- `real` corresponds to a *real number*.
- `rvar` corresponds to a *real variable*. A *real variable* is a variable that will be assigned a *real number* taken from an initial finite set of intervals. A real number is usually represented by an interval of two floating point numbers.

Beside domain, set, multiset and float variables we have not yet introduced *graph variables* [151]. A graph variable is currently simulated by using one set variable for each vertex of the graph (see the third example of type declaration of 2.2.2).

2.2.2 Compound data types

We provide the following *compound data types*:

- $\texttt{list}(T)$ corresponds to a list of elements of type T, where T is a basic or a compound data type.

- $\texttt{collection}(A_1, A_2, \ldots, A_n)$ corresponds to a collection of ordered items, where each item consists of $n > 0$ attributes $A_1, A_2, \ldots, A_n$. Each attribute is an expression of the form $\texttt{a} - T$, where $\texttt{a}$ is the *name* of the attribute and T the *type* of the attribute (a basic or a compound data type). All names of the attributes of a given collection should be distinct and different from the keyword `key`, which corresponds to an implicit[5] attribute. Its value is the position of an item within the collection. The first item of a collection is associated with position 1.

The following notations are used for instantiated arguments:

- A *list of elements* $e_1, e_2, \ldots, e_n$ is denoted $[e_1, e_2, \ldots, e_n]$.

- A *finite set of integers* $i_1, i_2, \ldots, i_n$ is denoted $\{i_1, i_2, \ldots, i_n\}$.

- A *multiset of integers* $i_1, i_2, \ldots, i_n$ is denoted $\{\{i_1, i_2, \ldots, i_n\}\}$.

- A *collection of n items*, each item having m attributes, is denoted by $\langle \texttt{a}_1 - v_{11} \ldots \texttt{a}_m - v_{1m},\ \texttt{a}_1 - v_{21} \ldots \texttt{a}_m - v_{2m},\ \ldots\ ,\texttt{a}_1 - v_{n1} \ldots \texttt{a}_m - v_{nm}\rangle$. Each item is separated from the previous item by a comma. When the items of the collection involve a single attribute $\texttt{a}_1$, $\langle v_{11}, v_{21}, \ldots, v_{n1}\rangle$ can possibly be used as a shortcut for $\langle \texttt{a}_1 - v_{11},\ \texttt{a}_1 - v_{21},\ \ldots\ ,\texttt{a}_1 - v_{n1}\rangle$.

- The *i^{th} item of a collection* $\texttt{c}$ is denoted $\texttt{c}[i]$.

- The *value of the attribute* $\texttt{a}$ *of the i^{th} item of a collection* $\texttt{c}$ is denoted $\texttt{c}[i].\texttt{a}$. Note that, within an arithmetic expression, we can use the shortcut $\texttt{c}[i]$ when the collection $\texttt{c}$ involves a single attribute.

- The *number of items* of a collection $\texttt{c}$ is denoted $|\texttt{c}|$.

[5] This attribute is not explicitly defined.

EXAMPLE: Let us illustrate with four examples, the types one can create. These examples concern the creation of a collection of variables, a collection of tasks, a graph variable [151] and a collection of orthotopes.[a]

- In the first example we define VARIABLES so that it corresponds to a collection of variables. VARIABLES is used, for example, in the ALLDIFFERENT constraint. The declaration $\texttt{VARIABLES} : \texttt{collection}(\texttt{var} - \texttt{dvar})$ defines a collection of items, each of which having one attribute `var` that is a domain variable.

- In the second example we define TASKS so that it corresponds to a collection of tasks, each task being defined by its origin, its duration, its end and its resource consumption. Such a collection is used, for example, in the CUMULATIVE constraint. The declaration $\texttt{TASKS} : \texttt{collection}(\texttt{origin} - \texttt{dvar}, \texttt{duration} - \texttt{dvar}, \texttt{end} - \texttt{dvar}, \texttt{height} - \texttt{dvar})$ defines a collection of items, each of which having the four attributes `origin`, `duration`, `end` and `height` which all are domain variables.

- In the third example we define a graph as a collection of nodes NODES, each node being defined by its index (i.e., identifier) and its successors. Such a collection is used, for example, in the DAG constraint. The declaration $\texttt{NODES} : \texttt{collection}(\texttt{index} - \texttt{int}, \texttt{succ} - \texttt{svar})$ defines a collection of items, each of which having the two attributes `index` and `succ` which respectively are integers and set variables.

- In the last example we define ORTHOTOPES so that is corresponds to a collection of orthotopes. Each orthotope is described by an attribute `orth`. Unlike the previous examples, the type of this attribute does not correspond any more to a basic data type but rather to a collection of n items, where n is the number of dimensions of the orthotope.[b] This collection, named ORTHOTOPE, defines for a given dimension the origin, the size and the end of the object in this dimension. This leads to the two declarations:

  - $\texttt{ORTHOTOPE} - \texttt{collection}(\texttt{ori} - \texttt{dvar}, \texttt{siz} - \texttt{dvar}, \texttt{end} - \texttt{dvar})$,
  - $\texttt{ORTHOTOPES} - \texttt{collection}(\texttt{orth} - \texttt{ORTHOTOPE})$.

  ORTHOTOPES is used, for example, in the DIFFN constraint.

[a] An *orthotope* corresponds to the generalisation of a segment, a rectangle and a box to the n-dimensional case.

[b] 1 for a segment, 2 for a rectangle, 3 for a box, ...

2.2.3 Restrictions

When defining the arguments of a global constraint, it is often the case that one needs to express additional conditions that refine the type declarations of its arguments. For this purpose we provide *restrictions* that allow for specifying these additional conditions. Each restriction has a name and a set of arguments and is described by the following items:

- A small paragraph first describes the effect of the restriction,
- An example points to a constraint using the restriction,
- Finally, a ground instance, preceded by the symbol ▷, which satisfies the restriction is given. Similarly, a ground instance, preceded by the symbol ▶, which

violates the restriction is proposed. In this latter case, a bold font may be used for pointing to the source of the problem.

Currently the list of restrictions is:

- `in_list(Arg, ListAtoms)`

  - `Arg` is an argument of type `atom`,
  - `ListAtoms` is a non-empty list of distinct atoms.

  This restriction forces `Arg` to be one of the atoms specified in the list `ListAtoms`.

  EXAMPLE: An example of use of such restriction can be found in the CHANGE(NCHANGE, VARIABLES, CTR) constraint: in_list(CTR, $[=, \neq, <, \geq, >, \leq]$) forces the last argument CTR of the CHANGE constraint to take its value in the list of atoms $[=, \neq, <, \geq, >, \leq]$.
  ▷ CHANGE(1, ⟨var − 4, var − 4, var − 4, var − 6⟩, $\neq$)
  ▶ CHANGE(1, ⟨var − 4, var − 4, var − 4, var − 6⟩, **3**)

- `in_list(Arg, Attr, ListIntOrAtom)`

  - `Arg` is an argument of type `collection`,
  - `Attr` is an attribute of type `int` or of type `atom` of the collection denoted by `Arg`,
  - When `Attr` is an attribute of type `int`, `ListIntOrAtom` is a non-empty list of distinct integers; otherwise, when `Attr` is an attribute of type `atom`, `ListIntOrAtom` is a non-empty list of distinct atoms.

  This restriction forces for all items of the collection `Arg`, the attribute `Attr` to take its value within the list `ListIntOrAtom`.

- `in_attr(Arg1, Attr1, Arg2, Attr2)`

  - `Arg1` is an argument of type `collection`,
  - `Attr1` is an attribute of type `dvar` or of type `int` of the collection denoted by `Arg1`,
  - `Arg2` is an argument of type `collection`,
  - `Attr2` is an attribute of type `int` of the collection denoted by `Arg2`.

  Let $\mathcal{S}_2$ denotes the set of values assigned to the `Attr2` attributes of the items of the collection `Arg2`. This restriction forces the following condition: for all items of the collection `Arg1`, the attribute `Attr1` takes its value in the set $\mathcal{S}_2$.

EXAMPLE: An example of use of such restriction can be found in the CUMULATIVES(TASKS, MACHINES, CTR) constraint: in_attr(TASKS, machine, MACHINES, id) forces that the machine attribute of each task of the TASKS collection correspond to a machine identifier (i.e., an id attribute of the MACHINES collection).

▷CUMULATIVES(⟨ machine − 1 origin − 2 duration − 2 end − 4 height − 2,
machine − 1 origin − 2 duration − 2 end − 4 height − 2,
machine − 2 origin − 1 duration − 4 end − 5 height − 5,
machine − 1 origin − 4 duration − 2 end − 6 height − 1⟩,
⟨id − 1 capacity − 9, id − 2 capacity − 8⟩, ≤)

▶CUMULATIVES(⟨ machine − **5** origin − 2 duration − 2 end − 4 height − 2,
machine − 1 origin − 2 duration − 2 end − 4 height − 2,
machine − 2 origin − 1 duration − 4 end − 5 height − 5,
machine − 1 origin − 4 duration − 2 end − 6 height − 1⟩,
⟨id − **1** capacity − 9, id − **2** capacity − 8⟩, ≤)

- distinct(Arg, Attrs)
  - Arg is an argument of type collection,
  - Attrs is an attribute of type int or dvar, or a list (possibly empty) of distinct attributes of type int or dvar of the collection denoted by Arg.

For all pairs of distinct items of the collection Arg this restriction forces that there be at least one attribute specified by Attrs with two distinct values. When Attrs is the empty list all items of the collection Arg should be distinct.

EXAMPLE: An example of use of such restriction can be found in the CYCLE(NCYCLE, NODES) constraint: distinct(NODES, index) forces that all index attributes of the NODES collection take distinct values.

▷CYCLE(2, ⟨index − 1 succ − 2, index − 2 succ − 1, index − 3 succ − 3⟩)

▶CYCLE(2, ⟨index − **1** succ − 2, index − **1** succ − 1, index − 3 succ − 3⟩)

- increasing_seq(Arg, Attrs)
  - Arg is an argument of type collection,
  - Attrs is an attribute of type int or a list of distinct attributes of type int of the collection denoted by Arg.

Let n and m respectively denote the number of items of the collection Arg, and the number of attributes of Attrs. For item i of the collection Arg let t_i denotes the tuple of values $\langle v_{i,1}, v_{i,2}, \ldots, v_{i,m}\rangle$ where $v_{i,j}$ is the value of attribute j of Attrs of item i of Arg. The restriction forces a strict lexicographical ordering on the tuples $t_1, t_2, \ldots, t_n$.

EXAMPLE: An example of use of such restriction can be found in the ELEMENT_MATRIX(MAX_I, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE) constraint: increasing_seq(MATRIX, [i, j]) forces that all items of the MATRIX collection be sorted in strictly increasing lexicographic order on the pair (i, j).
▷ ELEMENT_MATRIX(2, 2, 1, 2, ⟨i − 1 j − 1 v − 4, i − 1 j − 2 v − 7,
i − 2 j − 1 v − 1, i − 2 j − 2 v − 1⟩, 7)
▶ ELEMENT_MATRIX(2, 2, 1, 2, ⟨i − **1** j − **2** v − 4, i − **1** j − **1** v − 7,
i − 2 j − 1 v − 1, i − 2 j − 2 v − 1⟩, 7)

- non_increasing_size(Arg, Attr)
  - Arg is an argument of type collection,
  - Attr is an attribute of the collection denoted by Arg. This attribute should be of type collection.

This restriction forces for each pair of consecutive items Arg[i], Arg[$i+1$] that the number of items of the collection Arg[i].Attr is greater than or equal to the number of items of the collection Arg[$i+1$].Attr.

EXAMPLE: An example of use of such restriction can be found in the K_USED_BY(SETS) constraint: non_increasing_size(SETS, set) forces for all consecutive pairs of items SETS[i], SETS[$i+1$] that the number of items of the collection SETS[i].set is not greater than or equal to the number of items of the collection SETS[$i+1$].set.
▷K_USED_BY(⟨ set − ⟨var − 5, var − 1, var − 1⟩,
set − ⟨var − 5, var − 1, var − 1⟩,
set − ⟨var − 5, var − 1⟩ ⟩)
▶K_USED_BY(⟨ set − ⟨var − 5, var − 1, var − 1⟩,
set − ⟨var − 5, var − 1⟩,
set − ⟨var − 5, var − 1, var − 1⟩ ⟩)

- required(Arg, Attrs)
  - Arg is an argument of type collection,
  - Attrs is an attribute or a list of distinct attributes of the collection denoted by Arg.

This restriction forces that all attributes denoted by Attrs be explicitly used within all items of the collection Arg.

EXAMPLE: An example of use of such restriction can be found in the CUMULATIVE(TASKS, LIMIT) constraint: required(TASKS, height) forces that all items of the TASKS collection mention the height attribute.
▷CUMULATIVE(⟨ origin − 2 duration − 2 end − 4 height − 2,
origin − 2 duration − 2 end − 4 height − 2,
origin − 1 duration − 4 end − 5 height − 5,
origin − 4 duration − 2 end − 6 height − 1⟩, 12)
▶CUMULATIVE(⟨ **origin** − 2 **duration** − 2 **end** − 4,
origin − 2 duration − 2 end − 4 height − 2,
origin − 1 duration − 4 end − 5 height − 5,
origin − 4 duration − 2 end − 6 height − 1⟩, 12)

The `required` restriction is usually systematically used for every attribute of a collection. It is not used when some attributes may be implicitly defined according to other attributes. In this context, we use the `require_at_least` restriction, which we now introduce.

- $\texttt{require\_at\_least}(\texttt{Atleast}, \texttt{Arg}, \texttt{Attrs})$

  - `Atleast` is a positive integer,
  - `Arg` is an argument of type `collection`,
  - `Attrs` is a non-empty list of distinct attributes of the collection denoted by `Arg`. The length of this list should be strictly greater than `Atleast`.

This restriction forces that at least `Atleast` attributes of the list `Attrs` be explicitly used within all items of the collection `Arg`.

EXAMPLE: An example of use of such restriction can be found in the CUMULATIVE($\texttt{TASKS}, \texttt{LIMIT}$) constraint:
$\texttt{require\_at\_least}(2, \texttt{TASKS}, [\texttt{origin}, \texttt{duration}, \texttt{end}])$ forces that all items of the `TASKS` collection mention at least two attributes from the list of attributes $[\texttt{origin}, \texttt{duration}, \texttt{end}]$. In this context, this stems from the equality $\texttt{origin} + \texttt{duration} = \texttt{end}$. This allows for retrieving the third attribute from the values of the two others.

$\triangleright$CUMULATIVE($\langle$ $\texttt{origin} - 2\ \texttt{duration} - 2\ \texttt{height} - 2,$
$\texttt{origin} - 2\ \texttt{end} - 4\ \texttt{height} - 2,$
$\texttt{duration} - 4\ \texttt{end} - 5\ \texttt{height} - 5,$
$\texttt{origin} - 4\ \texttt{duration} - 2\ \texttt{end} - 6\ \texttt{height} - 1\rangle,\ 12)$

$\blacktriangleright$CUMULATIVE($\langle$ $\textbf{origin} - 2\ \texttt{height} - 2,$
$\texttt{origin} - 2\ \texttt{duration} - 2\ \texttt{end} - 4\ \texttt{height} - 2,$
$\texttt{origin} - 1\ \texttt{duration} - 4\ \texttt{end} - 5\ \texttt{height} - 5,$
$\texttt{origin} - 4\ \texttt{duration} - 2\ \texttt{end} - 6\ \texttt{height} - 1\rangle,\ 12)$

- $\texttt{same\_size}(\texttt{Arg}, \texttt{Attr})$

  - `Arg` is an argument of type `collection`,
  - `Attr` is an attribute of the collection denoted by `Arg`. This attribute should be of type `collection`.

This restriction forces that all collections denoted by `Attr` have the same number of items.

EXAMPLE: An example of use of such restriction can be found in the DIFFN(ORTHOTOPES) constraint[a]: same_size(ORTHOTOPES, orth) forces all the items of the ORTHOTOPES collection to be constituted from the same number of items (of type ORTHOTOPE). From a practical point of view, this forces the DIFFN constraint to take as its argument a set of points, a set of rectangles, ... , a set of orthotopes.

▷DIFFN($\langle$ orth $-$ $\langle$ori $-$ 2 siz $-$ 2 end $-$ 4, ori $-$ 1 siz $-$ 3 end $-$ 4$\rangle$,
orth $-$ $\langle$ori $-$ 4 siz $-$ 4 end $-$ 8, ori $-$ 3 siz $-$ 3 end $-$ 3$\rangle$,
orth $-$ $\langle$ori $-$ 9 siz $-$ 2 end $-$ 11, ori $-$ 4 siz $-$ 3 end $-$ 7$\rangle$ $\rangle$)

▶DIFFN($\langle$ orth $-$ $\langle$ori $-$ 2 siz $-$ 2 end $-$ 4$\rangle$,
orth $-$ $\langle$ori $-$ 4 siz $-$ 4 end $-$ 8, ori $-$ 3 siz $-$ 3 end $-$ 3$\rangle$,
orth $-$ $\langle$ori $-$ 9 siz $-$ 2 end $-$ 11, ori $-$ 4 siz $-$ 3 end $-$ 7$\rangle$ $\rangle$)

[a] ORTHOTOPES corresponds to the third item of the example presented at page 17.

- Term$_1$ Comparison Term$_2$
  - Term$_1$ is a *term*. A *term* is an expression that can be evaluated to one or possibly several integer values. The expressions we allow for a *term* are defined in the next paragraph.
  - Comparison is one of the following comparison operators $\leq$, $\geq$, $<$, $>$, $=$, $\neq$.
  - Term$_2$ is a *term*.

Let $v_{1,1}, v_{1,2}, \dots, v_{1,n_1}$ and $v_{2,1}, v_{2,2}, \dots, v_{2,n_2}$ be the values respectively associated with Term$_1$ and with Term$_2$. The restriction Term$_1$ Comparison Term$_2$ forces $v_{1,i}$ Comparison $v_{2,j}$ to hold for every $i \in [1, n_1]$ and every $j \in [1, n_2]$.

A *term* is one of the following expressions:

- e, where e is an integer. The corresponding value is e.
- $|c|$, where c is an argument of type collection. The value of $|c|$ is the number of items of the collection denoted by c.

  EXAMPLE: This kind of expression is used, for example, in the restrictions of the ATLEAST(N, VARIABLES, VALUE) constraint: N $\leq$ |VARIABLES| restricts N to be less than or equal to the number of items of the VARIABLES collection.

  ▷ATLEAST(2, $\langle$var $-$ 5, var $-$ 8, var $-$ 5$\rangle$, 5)

  ▶ATLEAST(**4**, $\langle$var $-$ 5, var $-$ 8, var $-$ 5$\rangle$, 5)

- first($c.a$) : first($c.a$) denotes the value assigned to the attribute a of the first item of the collection denoted by c. It is equal to 0 if the collection is empty.
- last($c.a$) : last($c.a$) denotes the value assigned to the attribute a of the last item of the collection denoted by c. It is equal to 0 if the collection is empty.
- sum($c.a$), sum(ℓ) : sum($c.a$) denotes the sum of the values assigned to the attribute a of the collection denoted by c, it is equal to 0 if the collection

is empty; $\texttt{sum}(\ell)$ where ℓ is a list of collections attributes, each of them of the form $c_i.a_i$ (with $i \in [1, n]$), is the sum of the values assigned to the attributes a_i of the collections denoted by c_i ($i \in [1, n]$).

- $\texttt{range}(c.a), \texttt{range}(\ell)$: $\texttt{range}(c.a)$ denotes the difference between the maximum value and the minimum value plus one of the values assigned to the attribute a of the collection denoted by c, it is equal to 0 if the collection is empty; $\texttt{range}(\ell)$ where ℓ is a list of collections attributes, each of them of the form $c_i.a_i$ (with $i \in [1, n]$), is the difference between the maximum value and the minimum value plus one of the values assigned to the attributes a_i of the collections denoted by c_i ($i \in [1, n]$).
- $\texttt{minval}(c.a), \texttt{minval}(\ell)$: $\texttt{minval}(c.a)$ denotes the minimum over the values assigned to the attribute a of the collection denoted by c, it is equal to 0 if the collection is empty; $\texttt{minval}(\ell)$ where ℓ is a list of collections attributes, each of them of the form $c_i.a_i$ (with $i \in [1, n]$), is the minimum over the values assigned to the attributes a_i of the collections denoted by c_i ($i \in [1, n]$).
- $\texttt{maxval}(c.a), \texttt{maxval}(\ell)$: $\texttt{maxval}(c.a)$ denotes the maximum over the values assigned to the attribute a of the collection denoted by c, it is equal to 0 if the collection is empty; $\texttt{maxval}(\ell)$ where ℓ is a list of collections attributes, each of them of the form $c_i.a_i$ (with $i \in [1, n]$), is the maximum over the values assigned to the attributes a_i of the collections denoted by c_i ($i \in [1, n]$).
- $\texttt{nval}(c.a), \texttt{nval}(\ell)$: $\texttt{nval}(c.a)$ denotes the number of distinct values over the values assigned to the attribute a of the collection denoted by c, it is equal to 0 if the collection is empty; $\texttt{nval}(\ell)$ where ℓ is a list of collections attributes, each of them of the form $c_i.a_i$ (with $i \in [1, n]$), is the number of distinct values over the values assigned to the attributes a_i of the collections denoted by c_i ($i \in [1, n]$).
- $\texttt{prod}(c.a), \texttt{prod}(\ell)$: $\texttt{prod}(c.a)$ denotes the product of the values assigned to the attribute a of the collection denoted by c, it is equal to 1 if the collection is empty; $\texttt{prod}(\ell)$ where ℓ is a list of collections attributes, each of them of the form $c_i.a_i$ (with $i \in [1, n]$), is the product of the values assigned to the attributes a_i of the collections denoted by c_i ($i \in [1, n]$).
- $\texttt{pfdc}(c.a), \texttt{pfdc}(\ell)$: $\texttt{pfdc}(c.a)$, where pfdc is a pure functional dependency constraint of the form $\texttt{pfdc}(v, col)$ (e.g. AMONG_DIFF_0, PEAK, VALLEY) that computes a value v from a collection of variables col, and where $c.a$ is a collection with attribute a denotes the pfdc of the values assigned to the attribute a of the collection denoted by c, it is equal to 0 if the collection is empty; $\texttt{pfdc}(\ell)$ where ℓ is a list of collections attributes, each of them of the form $c_i.a_i$ (with $i \in [1, n]$), is the pfdc of the values assigned to the attributes a_i of the collections denoted by c_i ($i \in [1, n]$).
- t , where t is an argument of type int. The value of t is the value of the corresponding argument.

EXAMPLE: This kind of expression is used, for example, in the restrictions of the ATLEAST($\mathtt{N}, \mathtt{VARIABLES}, \mathtt{VALUE}$) constraint: $\mathtt{N} \geq 0$ forces the first argument of the ATLEAST constraint to be greater than or equal to 0.
▷ATLEAST$(2, \langle \mathtt{var}-5, \mathtt{var}-8, \mathtt{var}-5\rangle, 5)$
▶ATLEAST$(\mathbf{-1}, \langle \mathtt{var}-5, \mathtt{var}-8, \mathtt{var}-5\rangle, 5)$

– $\boxed{v}$, where v is an argument of type `dvar`. The value of v will be the value assigned to variable v.[6]

EXAMPLE: This kind of expression is used, for example, in the restrictions of the AMONG($\mathtt{NVAR}, \mathtt{VARIABLES}, \mathtt{VALUES}$) constraint: $\mathtt{NVAR} \geq 0$ forces the first argument of the AMONG constraint to be greater than or equal to 0.
▷AMONG$(2, \langle \mathtt{var}-5, \mathtt{var}-8, \mathtt{var}-5\rangle, \langle \mathtt{val}-1, \mathtt{val}-5\rangle)$
▶AMONG$(\mathbf{-9}, \langle \mathtt{var}-5, \mathtt{var}-8, \mathtt{var}-5\rangle, \langle \mathtt{val}-1, \mathtt{val}-5\rangle)$

– $\boxed{s}$, where s is an argument of type `sint` or `svar`. The values denoted by s are all the values of the corresponding set.

EXAMPLE: This kind of expression is used, for example, in the restrictions of the OPEN_ALLDIFFERENT($\mathtt{S}, \mathtt{VARIABLES}$) constraint: $\mathtt{S} \geq 1$ forces all elements of the set corresponding to the first argument of the OPEN_ALLDIFFERENT constraint to be greater than or equal to 1.
▷OPEN_ALLDIFFERENT$(\{1, 2, 3\}, \langle \mathtt{var}-5, \mathtt{var}-8, \mathtt{var}-3, \mathtt{var}-8, \mathtt{var}-9\rangle)$
▶OPEN_ALLDIFFERENT$(\{\mathbf{0}, 1, 2, 3\}, \langle \mathtt{var}-5, \mathtt{var}-8, \mathtt{var}-3, \mathtt{var}-8, \mathtt{var}-9\rangle)$

– $\boxed{c.a}$, where c is an argument of type `collection` and a an attribute of c of type `int` or `dvar`. The values denoted by $c.a$ are all the values corresponding to attribute a for the different items of c. When $c.a$ designates a domain variable we consider the value assigned to that variable.

EXAMPLE: This kind of expression is used, for example, in the restrictions of the CUMULATIVE($\mathtt{TASKS}, \mathtt{LIMIT}$) constraint: $\mathtt{TASKS.duration} \geq 0$ forces for all items of the `TASKS` collection that the `duration` attribute be greater than or equal to 0.

$$\triangleright\text{CUMULATIVE}\left(\left\langle\begin{array}{l}\mathtt{origin}-2\ \mathtt{duration}-2\ \mathtt{end}-4\ \mathtt{height}-2,\\ \mathtt{origin}-2\ \mathtt{duration}-2\ \mathtt{end}-4\ \mathtt{height}-2,\\ \mathtt{origin}-1\ \mathtt{duration}-4\ \mathtt{end}-5\ \mathtt{height}-5,\\ \mathtt{origin}-4\ \mathtt{duration}-2\ \mathtt{end}-6\ \mathtt{height}-1\end{array}\right\rangle, 12\right)$$

$$\blacktriangleright\text{CUMULATIVE}\left(\left\langle\begin{array}{l}\mathtt{origin}-2\ \mathtt{duration}-\mathbf{-2}\ \mathtt{end}-4\ \mathtt{height}-2,\\ \mathtt{origin}-2\ \mathtt{duration}-2\ \mathtt{end}-4\ \mathtt{height}-2,\\ \mathtt{origin}-1\ \mathtt{duration}-4\ \mathtt{end}-5\ \mathtt{height}-5,\\ \mathtt{origin}-4\ \mathtt{duration}-2\ \mathtt{end}-6\ \mathtt{height}-1\end{array}\right\rangle, 12\right)$$

– $\boxed{c.a}$, where c is an argument of type `collection` and a an attribute of c of type `sint` or `svar`. The values denoted by $c.a$ are all the values belonging to the sets corresponding to attribute a for the different items of c. When $c.a$ designates a set variable we consider the values that finally belong to that set.

[6] Restrictions are defined on the ground instance of a global constraint.

EXAMPLE: This kind of expression is used, for example, in the restrictions of the INVERSE_SET$(\mathtt{X},\mathtt{Y})$ constraint: $\mathtt{X.x} \geq 1$ forces for all items of the X collection that all the potential elements of the set variable associated with the x attribute be greater than or equal to 1.

$\triangleright$INVERSE_SET$(\langle\,\mathtt{index}-1\;\mathtt{x}-\{2,4\},$
$\mathtt{index}-2\;\mathtt{x}-\{4\},$
$\mathtt{index}-3\;\mathtt{x}-\{1\},$
$\mathtt{index}-4\;\mathtt{x}-\{4\}\,\rangle,$
$\langle\,\mathtt{index}-1\;\mathtt{y}-\{3\},$
$\mathtt{index}-2\;\mathtt{y}-\{1\},$
$\mathtt{index}-3\;\mathtt{y}-\{\},$
$\mathtt{index}-4\;\mathtt{y}-\{1,2,4\},$
$\mathtt{index}-5\;\mathtt{y}-\{\}\,\rangle)$

$\blacktriangleright$INVERSE_SET$(\langle\,\mathtt{index}-1\;\mathtt{x}-\{\mathbf{0},2,4\},$
$\mathtt{index}-2\;\mathtt{x}-\{4\},$
$\mathtt{index}-3\;\mathtt{x}-\{1\},$
$\mathtt{index}-4\;\mathtt{x}-\{4\}\,\rangle,$
$\langle\,\mathtt{index}-1\;\mathtt{y}-\{3\},$
$\mathtt{index}-2\;\mathtt{y}-\{1\},$
$\mathtt{index}-3\;\mathtt{y}-\{\},$
$\mathtt{index}-4\;\mathtt{y}-\{1,2,4\},$
$\mathtt{index}-5\;\mathtt{y}-\{\}\,\rangle)$

– $\min(t_1, t_2)$ or $\max(t_1, t_2)$, where t_1 and t_2 are *terms*. Let $\mathcal{V}_1$ and $\mathcal{V}_2$ denote the sets of values respectively associated with the terms t_1 and t_2. Let $\min(\mathcal{V}_1)$, $\max(\mathcal{V}_1)$ and $\min(\mathcal{V}_2)$, $\max(\mathcal{V}_2)$ denote the minimum and maximum values of $\mathcal{V}_1$ and $\mathcal{V}_2$. The value associated with $\min(t_1, t_2)$ is $\min(\min(\mathcal{V}_1), \min(\mathcal{V}_2))$, while the value associated with $\max(t_1, t_2)$ is $\max(\max(\mathcal{V}_1), \max(\mathcal{V}_2))$.

EXAMPLE: This kind of expression is used, for example, in the restrictions of the NINTERVAL$(\mathtt{NVAL}, \mathtt{VARIABLES}, \mathtt{SIZE_INTERVAL})$ constraint: $\mathtt{NVAL} \geq \min(1, |\mathtt{VARIABLES}|)$ forces NVAL to be greater than or equal to the minimum of 1 and the number of items of the VARIABLES collection.

$\triangleright$ NINTERVAL$(2, \langle \mathtt{var}-3, \mathtt{var}-1, \mathtt{var}-9, \mathtt{var}-1, \mathtt{var}-9\rangle, 4)$
$\blacktriangleright$ NINTERVAL$(\mathbf{0}, \langle \mathtt{var}-3, \mathtt{var}-1, \mathtt{var}-9, \mathtt{var}-1, \mathtt{var}-9\rangle, 4)$

– t_1 op t_2, where t_1 and t_2 are *terms* and op one of the operators $+$, $-$, $*$ or $/$.[7] Let $\mathcal{V}_1$ and $\mathcal{V}_2$ denote the sets of values respectively associated with the terms t_1 and t_2. The set of values associated with t_1 op t_2 is $\mathcal{V}_{12} = \{v : v = v_1 \;\mathtt{op}\; v_2, v_1 \in \mathcal{V}_1, v_2 \in \mathcal{V}_2\}$.

[7] / denotes an integer division, a division in which the fractional part is discarded.

EXAMPLE: This kind of expression is used, for example, in the restrictions of the RELAXED_SLIDING_SUM(ATLEAST, ATMOST, LOW, UP, SEQ, VARIABLES) constraint: $\texttt{ATMOST} \leq |\texttt{VARIABLES}| - \texttt{SEQ} + 1$ forces ATMOST to be less than or equal to an arithmetic expression that corresponds to the number of sequences of SEQ consecutive variables in a sequence of |VARIABLES| variables.

▷ RELAXED_SLIDING_SUM$(3, 4, 3, 7, 4, \langle \texttt{var} - 2, \texttt{var} - 4, \texttt{var} - 2, \texttt{var} - 0, \texttt{var} - 0, \texttt{var} - 3, \texttt{var} - 4 \rangle)$

▶ RELAXED_SLIDING_SUM$(3, \mathbf{9}, 3, 7, 4, \langle \texttt{var} - 2, \texttt{var} - 4, \texttt{var} - 2, \texttt{var} - 0, \texttt{var} - 0, \texttt{var} - 3, \texttt{var} - 4 \rangle)$

- We can use a disjunction between two restrictions.

EXAMPLE: This kind of expression is used, for example, in the **Typical** slot of the AMONG_LOW_UP(LOW, UP, VARIABLES, VALUES) constraint: $\texttt{LOW} > 0 \lor \texttt{UP} < |\texttt{VARIABLES}|$ forces the pair LOW, UP to impose a restriction on the variables of the VARIABLES collection.[a]

▷ AMONG_LOW_UP$(1, 2, \langle 9, 2, 4, 5 \rangle, \langle 0, 2, 4, 6, 9 \rangle)$
▷ AMONG_LOW_UP$(0, 3, \langle 9, 2, 4, 5 \rangle, \langle 0, 2, 4, 6, 9 \rangle)$
▷ AMONG_LOW_UP$(1, 4, \langle 9, 2, 4, 5 \rangle, \langle 0, 2, 4, 6, 9 \rangle)$
▶ AMONG_LOW_UP$(\mathbf{0}, \mathbf{4}, \langle 9, 2, 4, 5 \rangle, \langle 0, 2, 4, 6, 9 \rangle)$

[a] Since when both, $\texttt{LOW} \leq 0$ and $\texttt{UP} \geq |\texttt{VARIABLES}|$, the corresponding AMONG_LOW_UP constraint always holds.

- Finally, we can also use a constraint C of the catalogue for expressing a restriction as long as that constraint is not defined according to the constraint under consideration. The constraint C should have a graph-based or an automaton-based description so that its meaning is explicitly defined.

EXAMPLE: An example of use of such restriction can be found in the SORT_PERMUTATION(FROM, PERMUTATION, TO) constraint: ALLDIFFERENT(PERMUTATION) is used to express that the variables of the second argument of the SORT_PERMUTATION constraint should take distinct values.

2.2.4 Declaring a global constraint

Declaring a global constraint consists of providing the following information:

- A *term* $\texttt{constraint}(A_1, A_2, \ldots, A_n)$, where `constraint` corresponds to the *name* of the global constraint and $A_1, A_2, \ldots, A_n$ to its *arguments*.

- A possibly empty *list of type declarations*, where each declaration has the form `type:type_declaration`; `type` is the *name* of the new type we define and `type_declaration` is a basic data type, a compound data type or a type previously defined.

- An *argument declaration* $A_1{:}T_1, A_2{:}T_2, \ldots, A_n{:}T_n$ giving for each argument $A_1, A_2, \ldots, A_n$ of the global constraint `constraint` its type. Each type is a basic data type, a compound data type, or a type that was declared in the list of type declarations.

- A possibly empty *list of restrictions*, where each restriction is one of the restrictions described in Section 2.2.3 on page 17.

EXAMPLE: The arguments of the ALL_DIFFER_FROM_AT_LEAST_K_POS constraint are described by:

Constraint	ALL_DIFFER_FROM_AT_LEAST_K_POS(K, VECTORS)
Type(s)	VECTOR − collection(var − dvar)
Argument(s)	K − int
	VECTORS − collection(vec − VECTOR)
Restriction(s)	required(VECTOR, var)
	$\texttt{K} \geq 0$
	required(VECTORS, vec)
	same_size(VECTORS, vec)

The first line indicates that the ALL_DIFFER_FROM_AT_LEAST_K_POS constraint has two arguments: K and VECTORS. The second line declares a new type VECTOR, which corresponds to a collection of variables. The third line indicates that the first argument K is an integer, while the fourth line tells that the second argument VECTORS corresponds to a collection of vectors of type VECTOR. Finally the four restrictions respectively enforce that:

- All the items of the VECTOR collection mention the var attribute,
- K be greater than or equal to 0,
- All the items of the VECTORS collection mention the vec attribute,
- All the vectors have the same number of components.

2.2.5 Describing symmetries between arguments

Given a satisfied ground instance of a global constraint constraint, it is often the case that the constraint is still satisfied [129, 180] if we permute:

- *Some of its* **arguments**.

  E.g., consider the disequality constraint NEQ(X, Y), which forces X being assigned an integer value that is different from Y. Given the solution NEQ$(3, 5)$ we can swap both arguments and still get a solution (i.e., NEQ$(5, 3)$).

- **Items** *of some collections that are passed as one of its arguments.*

  E.g., consider the ALLDIFFERENT(VARIABLES) constraint, which imposes all variables of the collection VARIABLES being assigned a distinct integer value. Given the solution ALLDIFFERENT$(\langle 5, 1, 9, 3\rangle)$ we can swap any pair of items and still get a solution. For example, if we swap the first and fourth items we still get a solution (i.e., ALLDIFFERENT$(\langle 3, 1, 9, 5\rangle)$).

- **Attributes** *of some items of some of its collections.*

  E.g., given a collection of pairs PAIRS, where each pair has two attributes x and y, the NPAIR(N, PAIRS) constraint forces N being the number of distinct pairs in

PAIRS. Given the solution NPAIR$(3, \langle \mathtt{x}-3\ \mathtt{y}-1, \mathtt{x}-1\ \mathtt{y}-5, \mathtt{x}-3\ \mathtt{y}-1, \mathtt{x}-1\ \mathtt{y}-5, \mathtt{x}-1\ \mathtt{y}-3\rangle)$ we can interchange attributes x and y and still get a solution (i.e., NPAIR$(3, \langle \mathtt{x}-1\ \mathtt{y}-3, \mathtt{x}-5\ \mathtt{y}-1, \mathtt{x}-1\ \mathtt{y}-3, \mathtt{x}-5\ \mathtt{y}-1, \mathtt{x}-3\ \mathtt{y}-1\rangle)$).

- **A pair of values** *with respect to an attribute of some of its collections.*

  E.g., consider the BIN_PACKING constraint, which assigns items to bins in such a way that the total weight of the items in each bin does not exceed an overall fixed capacity. Each item has a bin and a weight attributes, which respectively give the bin to which the item will be assigned, and the weight of the item. Given the solution BIN_PACKING$(5, \langle \mathtt{bin}-3\ \mathtt{weight}-4, \mathtt{bin}-1\ \mathtt{weight}-3, \mathtt{bin}-3\ \mathtt{weight}-1\rangle)$, we can interchange all occurrences of value 3 with all occurrences of value 1 with respect to the bin attribute. After this swap of values we get the new solution BIN_PACKING$(5, \langle \mathtt{bin}-1\ \mathtt{weight}-4, \mathtt{bin}-3\ \mathtt{weight}-3, \mathtt{bin}-1\ \mathtt{weight}-1\rangle)$. This simply consists of swapping the content of two bins. Since all bins have the same capacity we still get a solution.

We provide the following *moves*, where each move is described by (1) an explicit fact (i.e., a meta-data), (2) a textual explanation, and (3) several concrete examples:

- args(PERMUTATION) denotes the fact that we swap the arguments of a constraint with respect to a given permutation. Arguments which are exchanged must have the same type under the hypothesis that they are ground (for example, the basic data types int and dvar, which respectively denote an integer value and a domain variable can be exchanged since a ground domain variable corresponds to an integer value). The permutation PERMUTATION is described by using standard notation, that is by providing the different cycles of the permutation.

EXAMPLE 1: As a first example where we can swap two arguments, consider the EQ_CST(VAR1, VAR2, CST2) constraint which, given two domain variables VAR1, VAR2 and an integer value CST2, forces the condition VAR1 = VAR2 + CST2. Within the electronic catalogue this is represented by the following meta-data, args([[VAR1], [VAR2, CST2]]), to which corresponds the following textual form:

arguments are permutable w.r.t. permutation (VAR1) (VAR2, CST2).

Note that, even though arguments VAR2 and CST2 do not have the same type (i.e., VAR2 is a domain variable, while CST2 is an integer value), both arguments can be exchanged since we consider the ground case. For example, since EQ_CST$(8, \mathbf{2}, \mathbf{6})$ is satisfied, EQ_CST$(8, \mathbf{6}, \mathbf{2})$ is also satisfied.

EXAMPLE 2: As a second example where we can swap several arguments, consider the COMMON(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2) constraint which, given two domain variables NCOMMON1, NCOMMON2 and two collections of domain variables VARIABLES1, VARIABLES2, forces the following two conditions:

- NCOMMON1 is the number of variables of VARIABLES1 assigned a value in VARIABLES2.
- NCOMMON2 is the number of variables of VARIABLES2 assigned a value in VARIABLES1.

Within the electronic catalogue this is represented by the following meta-data, args([[NCOMMON1, NCOMMON2], [VARIABLES1, VARIABLES2]]), to which corresponds the following textual form:

arguments are permutable w.r.t. permutation (NCOMMON1, NCOMMON2) (VARIABLES1, VARIABLES2).

For example, since COMMON($\mathbf{3}, \mathbf{4}, \langle 1, 9, 1, 5\rangle, \langle 2, 1, 9, 9, 6, 9\rangle$) is satisfied, COMMON($\mathbf{4}, \mathbf{3}, \langle 2, 1, 9, 9, 6, 9\rangle, \langle 1, 9, 1, 5\rangle$) is also satisfied.

- items(COLLECTION, PERMUTATIONS) denotes the fact that we can permute the items of the collection COLLECTION with respect to a permutation belonging to a given set of permutations PERMUTATIONS:

  - COLLECTION stands for one of the following:

    1. An argument ARG of the global constraint that corresponds to a collection of items.
    2. A term ARG.attr, where attr is an attribute of a collection of items that is an argument ARG of the global constraint; in addition, the type of attr is itself a collection. Given a collection ARG of m items $\langle$ARG$[1]$, ARG$[2], \dots,$ ARG$[m]\rangle$, a permutation of PERMUTATIONS, *not necessarily the same*, is applied on the items of a subset of the set of collections $\{$ARG$[1]$.attr, ARG$[2]$.attr$, \dots,$ ARG$[m]$.attr$\}$.

  - PERMUTATIONS represents a set of permutations. It can take one of the following values:

    1. all stands for *all possible permutations*. Note that this case is a little artificial since it does not really correspond to a symmetry of the constraint, but rather to the use of a collection for representing a set of variables. But, to our best knowledge in 2010, concrete solvers do also not use sets of variables but rather collections, lists or arrays of variables.
    2. reverse stands for the set that only contains the permutation that maps the sequence $e_1, e_2, \dots, e_n$ to $e_n, e_{n-1}, \dots, e_1$.
    3. shift stands for the set that only contains the permutation that maps the sequence $e_1, e_2, \dots, e_n$ to $e_n, e_1, \dots, e_{n-1}$.

EXAMPLE 1: As a first example, consider the ALLDIFFERENT(VARIABLES) constraint, which has a single argument corresponding to a collection of variables which must all be assigned distinct values. Within the electronic catalogue this is represented by the following meta-data, `items`(VARIABLES, `all`), to which corresponds the following textual form:

items of VARIABLES *are permutable*.

For example, since ALLDIFFERENT($\langle \mathbf{1}, \mathbf{4}, \mathbf{9} \rangle$) is satisfied, all permutations of $\langle 1, 4, 9 \rangle$ (i.e., $\langle \mathbf{1}, \mathbf{4}, \mathbf{9} \rangle$, $\langle \mathbf{1}, \mathbf{9}, \mathbf{4} \rangle$, $\langle \mathbf{4}, \mathbf{1}, \mathbf{9} \rangle$, $\langle \mathbf{4}, \mathbf{9}, \mathbf{1} \rangle$, $\langle \mathbf{9}, \mathbf{1}, \mathbf{4} \rangle$, $\langle \mathbf{9}, \mathbf{4}, \mathbf{1} \rangle$) correspond to valid solutions to the ALLDIFFERENT constraint.

EXAMPLE 2: As a second example, consider the K_SAME(SETS) constraint, which has a single argument corresponding to a collection of sets, where each set is a collection of domain variables that must be assigned the same set of values (i.e., K_SAME forces an equality between multisets). The argument SETS is a collection, where each item consists of a single `set` attribute. The type of a `set` attribute is a collection of domain variables. Within the electronic catalogue this is represented by the following meta-data, `items`(SETS.`set`, `all`), to which corresponds the following textual form:

items of SETS.`set` *are permutable*.

For example, since K_SAME($\langle$`set` $- \langle 1, 4, 4 \rangle$, `set` $- \langle \mathbf{4}, \mathbf{4}, \mathbf{1} \rangle$, `set` $- \langle 1, 4, 4 \rangle \rangle$) is satisfied, it is also satisfied for all permutations of the elements of its second set $\langle 4, 4, 1 \rangle$, i.e.:

- K_SAME($\langle$`set` $- \langle 1, 4, 4 \rangle$, `set` $- \langle \mathbf{1}, \mathbf{4}, \mathbf{4} \rangle$, `set` $- \langle 1, 4, 4 \rangle \rangle$),
- K_SAME($\langle$`set` $- \langle 1, 4, 4 \rangle$, `set` $- \langle \mathbf{4}, \mathbf{1}, \mathbf{4} \rangle$, `set` $- \langle 1, 4, 4 \rangle \rangle$),
- K_SAME($\langle$`set` $- \langle 1, 4, 4 \rangle$, `set` $- \langle \mathbf{4}, \mathbf{4}, \mathbf{1} \rangle$, `set` $- \langle 1, 4, 4 \rangle \rangle$).

- `items_sync`(COLLECTIONS, PERMUTATIONS) denotes the fact that we can permute the items of several collections COLLECTIONS with respect to a permutation belonging to a given set of permutations PERMUTATIONS in such a way that *one and the same permutation is used on all collections* (i.e., therefore the keyword `items_sync` which stands for *items synchronisation*):

  - COLLECTIONS stands for a non-empty list of terms of the form ARG or ARG.`attr`, where ARG is an argument of the global constraint that corresponds to a collection, and `attr` is an attribute of ARG such that its type is itself a collection. In addition, we also have the following restrictions:

    1. If COLLECTIONS contains a single element then this element has the form ARG.`attr`. This is done to allow to designate more than a single collection.
    2. All collections designated by COLLECTIONS have the same type as well as the same number of items.

    The *same permutation* of PERMUTATIONS is applied on the items of the different collections referenced by COLLECTIONS.

  - As for the symmetry keyword `items`, PERMUTATIONS represents a set of permutations. It can take the same set of values as before, namely:

    1. `all` stands for *all possible permutations*.

2. `reverse` stands for the set that only contains the permutation that maps the sequence $e_1, e_2, \ldots, e_n$ to $e_n, e_{n-1}, \ldots, e_1$.
3. `shift` stands for the set that only contains the permutation that maps the sequence $e_1, e_2, \ldots, e_n$ to $e_n, e_1, \ldots, e_{n-1}$.

EXAMPLE 1: As a first example, consider the CONSECUTIVE_GROUPS_OF_ONES(GROUP_SIZES, VARIABLES) constraint, which has two arguments GROUP_SIZES and VARIABLES respectively corresponding to a collection of positive integers and to a collection of 0-1 domain variables. The constraint imposes that the m successive maximum groups of consecutive ones of VARIABLES have sizes GROUP_SIZES[1].nb, GROUP_SIZES[2].nb, ..., GROUP_SIZES[m].nb. Note that, if we reverse the items of both GROUP_SIZES and VARIABLES, we still have a solution. Within the electronic catalogue this is represented by the following meta-data, items_sync([GROUP_SIZES, VARIABLES], reverse), to which corresponds the following textual form:

items of GROUP_SIZES *and* VARIABLES *are simultaneously reversible*.

For example, since CONSECUTIVE_GROUPS_OF_ONES($\langle \mathbf{2}, \mathbf{1} \rangle$, $\langle 1, 1, 0, 0, 0, 1, 0 \rangle$) is a solution, CONSECUTIVE_GROUPS_OF_ONES($\langle \mathbf{1}, \mathbf{2} \rangle$, $\langle 0, 1, 0, 0, 0, 1, 1 \rangle$) is also a valid solution.

EXAMPLE 2: As a second example, consider the NVECTOR(NVEC, VECTORS) constraint, which has two arguments NVEC and VECTORS respectively corresponding to a domain variable and to a collection of collections of domain variables, where all collections have the same number of items. The unique attribute of VECTORS is denoted by vec and its type is a collection of domain variables. Each collection is interpreted as a vector and two vectors are distinct if and only if they differ in at least one component. The NVECTOR constraint forces NVEC to be equal to the number of distinct vectors within VECTORS. If we permute the components of all vectors with respect to a same permutation we still have the same number of distinct vectors. Within the electronic catalogue this is represented by the following meta-data, items_sync([VECTORS.vec], all), to which corresponds the following textual form:

items of VECTORS.vec *are permutable (*same permutation used*)*.

For example, since NVECTOR($2, \langle \mathtt{vec} - \langle \mathbf{1}, \mathbf{1}, \mathbf{8} \rangle, \mathtt{vec} - \langle \mathbf{5}, \mathbf{1}, \mathbf{6} \rangle, \mathtt{vec} - \langle \mathbf{1}, \mathbf{1}, \mathbf{8} \rangle \rangle$) is a solution, any permutation applied simultaneously to the three components of each vector leads to a solution, i.e.:

- NVECTOR($2, \langle \mathtt{vec} - \langle \mathbf{1}, \mathbf{1}, \mathbf{8} \rangle, \mathtt{vec} - \langle \mathbf{5}, \mathbf{1}, \mathbf{6} \rangle, \mathtt{vec} - \langle \mathbf{1}, \mathbf{1}, \mathbf{8} \rangle \rangle$),
- NVECTOR($2, \langle \mathtt{vec} - \langle \mathbf{1}, \mathbf{8}, \mathbf{1} \rangle, \mathtt{vec} - \langle \mathbf{5}, \mathbf{6}, \mathbf{1} \rangle, \mathtt{vec} - \langle \mathbf{1}, \mathbf{8}, \mathbf{1} \rangle \rangle$),
- NVECTOR($2, \langle \mathtt{vec} - \langle \mathbf{1}, \mathbf{1}, \mathbf{8} \rangle, \mathtt{vec} - \langle \mathbf{1}, \mathbf{5}, \mathbf{6} \rangle, \mathtt{vec} - \langle \mathbf{1}, \mathbf{1}, \mathbf{8} \rangle \rangle$),
- NVECTOR($2, \langle \mathtt{vec} - \langle \mathbf{1}, \mathbf{8}, \mathbf{1} \rangle, \mathtt{vec} - \langle \mathbf{1}, \mathbf{6}, \mathbf{5} \rangle, \mathtt{vec} - \langle \mathbf{1}, \mathbf{8}, \mathbf{1} \rangle \rangle$),
- NVECTOR($2, \langle \mathtt{vec} - \langle \mathbf{8}, \mathbf{1}, \mathbf{1} \rangle, \mathtt{vec} - \langle \mathbf{6}, \mathbf{1}, \mathbf{5} \rangle, \mathtt{vec} - \langle \mathbf{8}, \mathbf{1}, \mathbf{1} \rangle \rangle$),
- NVECTOR($2, \langle \mathtt{vec} - \langle \mathbf{8}, \mathbf{1}, \mathbf{1} \rangle, \mathtt{vec} - \langle \mathbf{6}, \mathbf{5}, \mathbf{1} \rangle, \mathtt{vec} - \langle \mathbf{8}, \mathbf{1}, \mathbf{1} \rangle \rangle$).

- attrs(COLLECTION, PERMUTATION) denotes the fact that we can permute the attributes of the collection COLLECTION, *not necessarily all items*, with respect to a permutation PERMUTATION. Attributes that are exchanged must have the same type under the hypothesis that they are ground (e.g., an attribute attr1 of type int can be exchanged with an attribute attr2 of type dvar.

EXAMPLE: As an example, consider the SCALAR_PRODUCT(LINEARTERM, CTR, VAL) constraint, which forces a linear term, represented by a collection with two attributes `coeff` and `var`, to be *equal*, *different*, *less*, *greater than or equal*, *greater*, or *less than or equal* (i.e., depending on the value of CTR) to VAL. In the ground case we can exchange attributes `coeff` and `var` without affecting the fact the constraint is satisfied. Within the electronic catalogue this is represented by the following meta-data, attrs(LINEARTERM, [[coeff, var]]), to which corresponds the following textual form:

attributes of LINEARTERM *are permutable w.r.t. permutation* (coeff, var) *(*permutation not necessarily applied to all items*)*.

For example, since SCALAR_PRODUCT($\langle$coeff $-$ 1 var $-$ 1, coeff $-$ **3** var $-$ **1**, coeff $-$ 1 var $-$ 4, $\rangle$, =, 8) is a solution, SCALAR_PRODUCT($\langle$coeff $-$ 1 var $-$ 1, coeff $-$ **1** var $-$ **3**, coeff $-$ 1 var $-$ 4, $\rangle$, =, 8) is also a valid solution (i.e., the attributes `coeff` and `var` of the second item were permuted).

- attrs_sync(COLLECTION, PERMUTATION) denotes the fact that we can permute the attributes of the collection COLLECTION, *necessarily all items*, with respect to a permutation PERMUTATION. As before, attributes that are exchanged must have the same type under the hypothesis that they are ground.

EXAMPLE: As an example, consider the CROSSING(NCROSS, SEGMENTS) constraint, which forces NCROSS to be equal to the number of line segments intersections between the line segments defined by the SEGMENTS collection. Each line segment is defined by the coordinates (ox, oy) and (ex, ey) of its two extremities. Note that we can exchange the role of the x and y axes without affecting the number of line segments intersections. Within the electronic catalogue this is represented by the following meta-data, attrs_sync(SEGMENTS, [[ox, oy], [ex, ey]]), to which corresponds the following textual form:

attributes of SEGMENTS *are permutable w.r.t. permutation* (ox, oy) (ex, ey) *(*permutation applied to all items*)*.

For example, since CROSSING(3, $\langle$ox $-$ **1** oy $-$ **4** ex $-$ *9* ey $-$ *2*, ox $-$ **1** oy $-$ **1** ex $-$ *3* ey $-$ *5*, ox $-$ **3** oy $-$ **2** ex $-$ *7* ey $-$ *4*, ox $-$ **9** oy $-$ **1** ex $-$ *9* ey $-$ *4*$\rangle$) is a solution, CROSSING(3, $\langle$ox $-$ **4** oy $-$ **1** ex $-$ *2* ey $-$ *9*, ox $-$ **1** oy $-$ **1** ex $-$ *5* ey $-$ *3*, ox $-$ **2** oy $-$ **3** ex $-$ *4* ey $-$ *7*, ox $-$ **1** oy $-$ **9** ex $-$ *4* ey $-$ *9*$\rangle$) is also a valid solution.

- vals(ATTRIBUTES, PARTITION, PAIRS, SOURCE, TARGET) denotes the fact that we can permute some source value with some distinct target value. The kind of value permutation we can perform is parameterised by five parameters:
  - ATTRIBUTES is a list of paths of the form ARG0 or ARG1. $\cdots$ .ARGn.attr ($n \geq 1$), where:
    - ARG0 is an argument of the global constraint of type domain variable, integer, or collection of domain variables or integers.
    - ARG1. $\cdots$ .ARGn.attr is a path to an integer attribute or to a collection of integers attribute of the global constraint. ARG1, ARG2, . . . , ARGn are collections and `attr` is an attribute of ARGn of type domain variable, integer, or collection of domain variables or integers. In this last context, all collections have the same number of items since we can only

exchange tuples of values that have the same number of components. The path does not necessarily start from a top level collection.

Its purpose is to define the scope where the exchange of values, or tuples of values, will take place. Note that:

* The case corresponding to ARG0 allows to express the fact the value of an integer argument can be changed in such a way that we still have a solution.
* The case when ARG1 is not a top level collection allows to express the fact the exchange of value takes place within a nested collection. In this context this implicitly defines several scopes for the exchange of values.
* The case where ARG1. ··· .ARGn.attr is a path to a collection of variables or integers allows expressing swap between tuples of values (i.e., the exchange of values is generalised to the exchange of tuples of values).

– PARTITION usually defines a partition $\mathcal{P}$ of integer values. Only when ARG1. ··· .ARGn.attr is a path to a collection of variables or integers, PARTITION defines a partition of tuples of integer values. For the time being we focus on the first case, i.e., a partition of integer values. Its aim is to define classes of values from which the source and target values will be selected. In order to define a partition $\mathcal{P}$ we first introduce the notion of *set of values generator*. Within these definitions, u and v both denote (1) an integer value, or (2) an argument of the constraint of type integer or domain variable, or (3) a term of the form |ARG| where ARG is an argument of type collection denoting the number of items of the collection, (4) a sum or difference of elements of the form (1), (2) or (3). We have two kinds of generators, namely:

  * A *basic set of values generator* is defined by one of those:
    · ARG.attr, where ARG is an argument of type collection and attr is an attribute of ARG of type integer or domain variable, denotes the set of all values assigned to ARG.attr.
    · notin(ARG.attr), where ARG is an argument of type collection and attr is an attribute of ARG of type integer or domain variable, denotes the set of all elements of $\mathbb{Z}$ that are not assigned to ARG.attr.
    · diff(ARG1.attr1, ARG2.attr2), where ARG1 (respectively ARG2) is an argument of type collection and attr1 (respectively attr2) is an attribute of ARG1 (respectively ARG2) of type integer or domain variable, denotes the set of all elements of $\mathbb{Z}$ that are assigned to ARG1.attr1 but not to ARG2.attr2.
    · u, denotes the set $\{u\}$.
    · cmp(u), ($cmp \in \{=, \neq, <, \geq, >, \leq\}$), denotes the set of all integers e such that the comparison e cmp u holds.

- $\cdot$ $\texttt{in}(u, v)$, $(u \leq v)$, denotes the set of all integers located in interval $[u, v]$.
- $\cdot$ $\texttt{notin}(u, v)$, $(u \leq v)$, denotes the set of all integers not located in interval $[u, v]$.
- $\cdot$ $\texttt{mod}(u, v)$, $(0 < v < u, u, v \in \mathbb{N}^+)$, denotes all integer values in $\mathbb{Z}$ that have v as remainder when divided by u.[8]

* Given set of values generators $\mathcal{S}_1$, $\mathcal{S}_2$, $\ldots$, $\mathcal{S}_n$ $(n \geq 2)$, a *compound set of values generator* is defined by:
  - $\cdot$ $[\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n]$ denotes all values that are in at least one of the sets $\mathcal{S}_1$, $\mathcal{S}_2$, $\ldots$, $\mathcal{S}_n$.
  - $\cdot$ $\texttt{notin}([\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n])$ denotes all values of $\mathbb{Z}$ that are not in any set $\mathcal{S}_1$, $\mathcal{S}_2$, $\ldots$, $\mathcal{S}_n$.

We now describe the different partition generators. Within the description, $\mathcal{S}$ and $\mathcal{D}$ denote set of values generators. Classes of a partition are ordered. Unless explicitly specified, classes are ordered with respect to the smallest element they contain.

* $\texttt{int}$ denotes a partition $\mathcal{P}$ where, to each element of $\mathbb{Z}$ corresponds a specific class of $\mathcal{P}$ containing just that element.
* $\texttt{int}(\mathcal{S})$ denotes a partition $\mathcal{P}$ where, to each element of $\mathcal{S}$ corresponds a specific class of $\mathcal{P}$ containing just that element.
* $\texttt{all}$ denotes a partition $\mathcal{P}$ containing a single class of values corresponding to all integer values in $\mathbb{Z}$.
* $\texttt{all}(\mathcal{S})$ denotes a partition $\mathcal{P}$ containing a single class of values corresponding to the elements of $\mathcal{S}$.
* $\texttt{comp}(\mathcal{S})$ denotes of partition $\mathcal{P}$ containing two classes of values: a first class corresponding to the elements of $\mathcal{S}$, and a second class consisting of all elements of $\mathbb{Z}$ that are not in $\mathcal{S}$.
* $\texttt{comp\_diff}(\mathcal{S}, \mathcal{D})$ denotes of partition $\mathcal{P}$ containing two classes of values: a first class corresponding to the elements of $\mathcal{S}$ but not in $\mathcal{D}$, and a second class consisting of all elements of $\mathbb{Z}$ that are neither in $\mathcal{S}$ nor in $\mathcal{D}$.
* $\texttt{intervals}(u)$, $(u > 0)$, denotes a partition $\mathcal{P}$ containing intervals of the form $[k \cdot u, k \cdot u + u - 1]$, $k \in \mathbb{Z}$.
* $\texttt{mod}(u)$, $(u > 0)$, denotes a partition $\mathcal{P}$ such that each class of $\mathcal{P}$ is made up from all integers in $\mathbb{Z}$ that have the same remainder when divided by u.[9]
* $\texttt{part}(P)$, where P is a collection of collections of integers passed as one of the arguments of the constraint, where each integer occurs once, denotes a partition $\mathcal{P}$ such that each class corresponds to the elements of one of the collections of P. Classes are ordered with respect to their occurrences in P.

[8] remainder$(a, n) = a - n\lfloor \frac{a}{n} \rfloor$.
[9] remainder$(a, n) = a - n\lfloor \frac{a}{n} \rfloor$.

When PARTITION defines a partition of tuples, where each tuple consists of k integers, PARTITION can only be set to int. In this context int denotes a partition $\mathcal{P}$ where, to each element of $\mathbb{Z}^k$ corresponds a specific class of $\mathcal{P}$ containing just that element.

- PAIRS is one of the symbols $\neq$, $=$, $<$, $\geq$, $>$, $\leq$, or dontcare. It specifies a set of pairs $\{(p_{i_1}, p_{j_1}), (p_{i_2}, p_{j_2}), \ldots, (p_{i_n}, p_{j_n})\}$ of elements of the partition $\mathcal{P}$ such that, when PAIRS is different from dontcare,[10] the condition i_k PAIRS j_k holds for all $k \in [1, n]$. The aim of the PAIRS parameter is to allow to specify which partitions of $\mathcal{P}$ the source value u and the target value v should belong to. In fact there should exist a pair (p_{i_k}, p_{j_k}), $(k \in [1, n])$, such that $u \in p_{i_k}$ and $v \in p_{j_k}$.
- SOURCE is one of the options all or dontcare:
  - When set to all it indicates that all occurrences of the source value should be replaced by the target value. All occurrences of the target value, if it is used, should also be replaced by the source value.
  - When set to dontcare it tells that not necessarily all occurrences of the source value should be replaced. The target value is left unchanged.
- TARGET is one of the options in or dontcare:
  - When set to in it indicates that the target value should correspond to an already existing value of ARG.attr.
  - When set to dontcare it tells that the target value can either correspond to an already existing value of ARG.attr, or designate a new value.

We now define the set of conditions we must have in order to exchange a source and a target values. Consider,

1. a ground instance of a global constraint C,
2. a path PATH that designates either an argument of type integer, or an integer attribute of a collection that occurs, possibly in a nested way, as one of the arguments of C,
3. the sets of values $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_h$ that are assigned to PATH in the ground instance of C,[11]
4. a partition of integer values $\mathcal{P}$ derived from PARTITION,
5. a set of pairs $\{(p_{i_1}, p_{j_1}), (p_{i_2}, p_{j_2}), \ldots, (p_{i_n}, p_{j_n})\}$ of elements of the partition $\mathcal{P}$ such that the condition PAIRS $=$ dontcare $\vee\, i_k$ PAIRS j_k holds for all $k \in [1, n]$,
6. a TARGET option.

Given one of the sets of values $\mathcal{V}_\alpha$, $(1 \leq \alpha \leq h)$, a source value u can be permuted with a target value v if and only if the following conditions are all satisfied:

[10] When PAIRS is equal to dontcare we just consider all possible pairs.
[11] We may have more than one set when the path does not start from a top level collection.

1. $u \neq v$ (*source and target values should be distinct*),
2. $u \in \mathcal{V}_\alpha$ (*source value, i.e., value that is replaced, should be part of the solution*),
3. $\exists k | u \in p_{i_k} \wedge v \in p_{j_k}$ (*source and target values should be located in the appropriate partition classes*),
4. TARGET $=$ in $\Rightarrow v \in \mathcal{V}_\alpha$ (*if* TARGET $=$ in *then the target value should also be part of the solution*).

If SOURCE is equal to all we replace each occurrence of u by v, and conversely each occurrence of v by u. Otherwise we replace at least one occurrence of u by v.

Without loss of generality, when PATH designates a collection of integer values or domain variables, the exchange of tuples of values is defined in a similar way.

We now provide a number of examples of value symmetry and illustrate how to encode them with the five parameters we just introduced. We start from the most common value symmetry, namely exchanging all occurrences of two distinct values or replacing all occurrences of a value by an unused value.

EXAMPLE 1: As a first example, consider the ALLDIFFERENT(VARIABLES) constraint, which forces all variables of the collection VARIABLES to take distinct values. Note that we can exchange two assigned values of VARIABLES, or replace an assigned value of VARIABLES by a new value, i.e., a value that is not yet assigned to any variable of VARIABLES. Within the electronic catalogue this is represented by the following meta-data, vals([VARIABLES.var], int, $\neq$, all, dontcare), to which corresponds the following textual form:

Two distinct values of VARIABLES.var *can be swapped; a value of* VARIABLES.var *can be renamed to any unused value.*

For example, since ALLDIFFERENT($\langle 5, 1, \mathbf{9}, 3 \rangle$) is a solution, we can replace value 9 by a not yet assigned value 0, for instance, and get another valid solution ALLDIFFERENT($\langle 5, 1, \mathbf{0}, 3 \rangle$).

The five parameters of vals([VARIABLES.var], int, $\neq$, all, dontcare) have the following meaning:

- [VARIABLES.var] indicates that the modification takes place within the values assigned to the var attribute of the VARIABLES collection.
- int defines the partition of values $\mathcal{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots$.
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- all specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- dontcare tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in VARIABLES.var.

EXAMPLE 2: As a second example, consider the NVALUE(NVAL, VARIABLES) constraint, which forces NVAL to be equal to the number of distinct values assigned to the variables of the collection VARIABLES. Note that we can exchange all occurrences of two distinct values of VARIABLES, or replace all occurrences of an assigned value of VARIABLES by a new value, i.e., a value that is not yet assigned to any variable of VARIABLES. Within the electronic catalogue this is represented by the following meta-data, vals([VARIABLES.var], int, $\neq$, all, dontcare), to which corresponds the following textual form:

All occurrences of two distinct values of VARIABLES.var *can be swapped; all occurrences of a value of* VARIABLES.var *can be renamed to any unused value.*

For example, since NVALUE$(4, \langle 3, \mathbf{1}, 7, \mathbf{1}, 6\rangle)$ is a solution, we can replace all occurrences of value 1 by a not yet assigned value 8, for instance, and get another valid solution NVALUE$(4, \langle 3, \mathbf{8}, 7, \mathbf{8}, 6\rangle)$. We can also swap all occurrences of value 1 and value 3, and get another valid solution NVALUE$(4, \langle \mathbf{1}, \mathbf{3}, 7, \mathbf{3}, 6\rangle)$.

The five parameters of vals([VARIABLES.var], int, $\neq$, all, dontcare) have the following meaning:

- [VARIABLES.var] indicates that the modification takes place within the values assigned to the var attribute of the VARIABLES collection.
- int defines the partition of values $\mathcal{P} = \dots, \{-1\}, \{0\}, \{1\}, \dots$.
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- all specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- dontcare tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in VARIABLES.var.

We now introduce a third and a fourth example where the meta-data used for describing value symmetry, vals([VARIABLES.var], int, $\neq$, all, dontcare), is replaced by vals([VARIABLES.var], int, $\neq$, all, in), i.e., we are not allowed to introduce an unused value.

EXAMPLE 3: As a third example, consider the ALL_MIN_DIST(MINDIST, VARIABLES) constraint, which forces for each pair $(\mathtt{var}_i, \mathtt{var}_j)$ of distinct variables of the collection VARIABLES that $|\mathtt{var}_i - \mathtt{var}_j| \geq \mathtt{MINDIST}$. Note that we can exchange two occurrences of distinct values of VARIABLES, but we cannot replace an existing value u by a new value v (since the new value v may be too close from another existing value w, i.e., $|v - w| < \mathtt{MINDIST}$). Within the electronic catalogue this is represented by the following meta-data, $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{int}, \neq, \mathtt{all}, \mathtt{in})$, to which corresponds the following textual form:

Two distinct values of VARIABLES.var *can be swapped.*

For example, since ALL_MIN_DIST$(2, \langle \mathbf{5}, 1, \mathbf{9}, 3 \rangle)$ is a solution, we can swap values 5 and 9, and get another valid solution ALL_MIN_DIST$(2, \langle \mathbf{9}, 1, \mathbf{5}, 3 \rangle)$.
The five parameters of $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{int}, \neq, \mathtt{all}, \mathtt{in})$ have the following meaning:

- [VARIABLES.var] indicates that the modification takes place within the values assigned to the var attribute of the VARIABLES collection.
- int defines the partition of values $\mathcal{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots$.
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- all specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- in tells that the source value has to be replaced by an already existing value in VARIABLES.var.

EXAMPLE 4: As a fourth example, consider the MINIMUM(MIN, VARIABLES) constraint, which forces MIN to be equal to the minimum value of the collection VARIABLES. Note that we can exchange all occurrences of two distinct values of VARIABLES, but we cannot replace an existing value u by a new value v (since the new value v may be smaller than MIN). Within the electronic catalogue this is represented by the following meta-data, $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{int}, \neq, \mathtt{all}, \mathtt{in})$, to which corresponds the following textual form:

All occurrences of two distinct values of VARIABLES.var *can be swapped.*

For example, since MINIMUM$(2, \langle 3, 2, 7, 2, 6 \rangle)$ is a solution, we can swap values 2 and 6, and get another valid solution MINIMUM$(2, \langle 3, 6, 7, 6, 2 \rangle)$.
The five parameters of $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{int}, \neq, \mathtt{all}, \mathtt{in})$ have the following meaning:

- [VARIABLES.var] indicates that the modification takes place within the values assigned to the var attribute of the VARIABLES collection.
- int defines the partition of values $\mathcal{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots$.
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- all specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- in tells that the source value has to be replaced by an already existing value in VARIABLES.var.

We now present three examples where, using the partition generator $\mathtt{comp}(\mathcal{S})$, we consider two classes of values: a first class consisting of elements of $\mathcal{S}$ and a second class of elements of $\mathbb{Z}$ not in $\mathcal{S}$. The first example corresponds to a value symmetry where values from the same class are exchanged, while the two other examples consider permutation of values between distinct classes with respect to a given class ordering.

EXAMPLE 5: As a fifth example, consider the AMONG($\mathtt{NVAR}$, $\mathtt{VARIABLES}$, $\mathtt{VALUES}$) constraint, which forces `NVAR` to be equal to the number of variables of the collection `VARIABLES` that are assigned a value in `VALUES`. We focus on exchanges of values that take place within `VARIABLES`. Note that, given a value that both occurs in `VARIABLES` and in `VALUES`, we can replace it by any value in `VALUES`. But we can also replace a value that occurs in `VARIABLES`, but not in `VALUES`, by any value not in `VALUES`. Within the electronic catalogue this is represented by the following meta-data, $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{comp}(\mathtt{VALUES.val}), =, \mathtt{dontcare}, \mathtt{dontcare})$, to which corresponds the following textual form:

An occurrence of a value of `VARIABLES.var` *that belongs to* `VALUES.val` *(resp. does not belong to* `VALUES.val`*) can be replaced by any other value in* `VALUES.val` *(resp. not in* `VALUES.val`*).*

For example, since AMONG$(3, \langle 4, \mathbf{5}, 5, \mathbf{4}, 1\rangle, \langle 1, 5, 8\rangle)$ is a solution, we can swap the first occurrence of value 5 with the second occurrence of value 4 in `VARIABLES.var`, and get another valid solution AMONG$(3, \langle 4, \mathbf{4}, 5, \mathbf{5}, 1\rangle, \langle 1, 5, 8\rangle)$.

The five parameters of $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{comp}(\mathtt{VALUES.val}), =, \mathtt{dontcare}, \mathtt{dontcare})$ have the following meaning:

- $[\mathtt{VARIABLES.var}]$ indicates that the modification takes place within the values assigned to the `var` attribute of the `VARIABLES` collection.
- $\mathtt{comp}(\mathtt{VALUES.val})$ defines two set of values, a first set $\mathcal{S}_1$ corresponding to all values in `VALUES.val`, and a second set $\mathcal{S}_2$ corresponding to all values not in `VALUES.val`.
- $=$ indicates that the exchange of values takes place within the same set, i.e., within $\mathcal{S}_1$ or within $\mathcal{S}_2$.
- `dontcare` specifies that one occurrence of the source value has to be replaced by the target value.
- `dontcare` tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in `VARIABLES.var`.

EXAMPLE 6: As a sixth example, consider the ATLEAST($\mathtt{N}, \mathtt{VARIABLES}, \mathtt{VALUE}$) constraint, which forces at least `N` variables of the collection `VARIABLES` to be assigned value `VALUE`. Note that, given an occurrence of value that belongs to `VARIABLES` that is different from `VALUE`, we can replace it by any other value that is also different from `VALUE`.[a] But we can also replace it by value `VALUE` since this does not decrease the number of variables that are assigned value `VALUE`. Within the electronic catalogue this is represented by the following meta-data, $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{comp}(\mathtt{VALUE}), \geq, \mathtt{dontcare}, \mathtt{dontcare})$, to which corresponds the following textual form:

An occurrence of a value of `VARIABLES.var` *that is different from* `VALUE` *can be replaced by any other value.*

For example, since ATLEAST$(2, \langle 4, 2, 4, 5, \mathbf{2}\rangle, 4)$ is a solution, we can replace the second occurrence of value 2 with a value that is different from value 4, e.g., value 8, and get another valid solution ATLEAST$(2, \langle 4, 2, 4, 5, \mathbf{8}\rangle, 4)$. We can also replace the second occurrence of value 2 with value 4 and get another valid solution ATLEAST$(2, \langle 4, 2, 4, 5, \mathbf{4}\rangle, 4)$.

The five parameters of $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{comp}(\mathtt{VALUE}), \geq, \mathtt{dontcare}, \mathtt{dontcare})$ have the following meaning:

- [`VARIABLES.var`] indicates that the modification takes place within the values assigned to the `var` attribute of the `VARIABLES` collection.
- `comp(VALUE)` defines two set of values, a first set $\mathcal{S}_1$ containing only value `VALUE`, and a second set $\mathcal{S}_2$ corresponding to all values different from `VALUE`.
- $\geq$ indicates that the the source and target values should respectively belong to sets $\mathcal{S}_i$ and $\mathcal{S}_j$ where $i \geq j$:
  1. If the source value is different from `VALUE` (i.e., the source value belongs to $\mathcal{S}_2$), then the target value can indifferently be equal or not equal to `VALUE` (i.e., the target value belongs to $\mathcal{S}_1$ or $\mathcal{S}_2$).
  2. If the source value is equal to `VALUE` (i.e., the source value belongs to $\mathcal{S}_1$), then the target value is equal to `VALUE` (i.e., the target value also belongs to $\mathcal{S}_1$). But in this case no exchange can take place since the source and target values are identical.
- `dontcare` specifies that one occurrence of the source value has to be replaced by the target value.
- `dontcare` tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in `VARIABLES.var`.

[a] Within the collection `VARIABLES`, this swap does not change the number of variables that are assigned value `VALUE`.

EXAMPLE 7: As a seventh example, consider the ATMOST(N, VARIABLES, VALUE) constraint, which forces at most N variables of the collection VARIABLES to be assigned value VALUE. Note that, given an occurrence of value that belongs to VARIABLES, and that is different from VALUE, we can replace it by any other value that is also different from VALUE.[a] But we can also replace an occurrence of value VALUE by a value that is different from VALUE, since this does not increase the number of variables that are assigned value VALUE. Within the electronic catalogue this is represented by the following meta-data, vals([VARIABLES.var], comp(VALUE), $\leq$, dontcare, dontcare), to which corresponds the following textual form:

An occurrence of a value of VARIABLES.var *can be replaced by any other value that is different from* VALUE.

For example, since ATMOST$(1, \langle 4, 2, \mathbf{4}, 5\rangle, 2)$ is a solution, we can replace the second occurrence of value 4 with a value that is different from value 2, e.g., value 8, and get another valid solution ATMOST$(1, \langle 4, 2, \mathbf{8}, 5\rangle, 2)$. But, within ATMOST$(1, \langle 4, \mathbf{2}, 4, 5\rangle, 2)$, we can also replace value 2 with any other value, e.g. value 4 and get another valid solution ATMOST$(1, \langle 4, \mathbf{4}, 4, 5\rangle, 2)$.

The five parameters of vals([VARIABLES.var], comp(VALUE), $\leq$, dontcare, dontcare) have the following meaning:

- [VARIABLES.var] indicates that the modification takes place within the values assigned to the var attribute of the VARIABLES collection.
- comp(VALUE) defines two set of values, a first set $\mathcal{S}_1$ containing only value VALUE, and a second set $\mathcal{S}_2$ corresponding to all values different from VALUE.
- $\leq$ indicates that the the source and target values should respectively belong to sets $\mathcal{S}_i$ and $\mathcal{S}_j$ where $i \leq j$:
  1. If the source value is different from VALUE (i.e., the source value belongs to $\mathcal{S}_2$), then the target value is also different from VALUE (i.e., the target value belongs to $\mathcal{S}_2$). This supports the fact that we do not want to increase the number of occurrences of value VALUE.
  2. If the source value is equal to VALUE (i.e., the source value belongs to $\mathcal{S}_1$), then there is no restriction on the target value (i.e., the target value belongs to $\mathcal{S}_1$ or to $\mathcal{S}_2$). But the set $\mathcal{S}_1$ is not relevant since the target value would also be fixed to VALUE, and, in this context, no exchange can take place.
- dontcare specifies that one occurrence of the source value has to be replaced by the target value.
- dontcare tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in VARIABLES.var.

[a] Within the collection VARIABLES, this swap does not change the number of variables that are assigned value VALUE.

We now illustrate the fact the scope of value symmetry can sometimes be extended to several collections of variables. For this purpose we consider the COMMON constraint.

EXAMPLE 8: Consider the COMMON(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2) constraint, which forces the two following conditions:

- NCOMMON1 is the number of variables of the collection VARIABLES1 taking a value in VARIABLES2.
- NCOMMON2 is the number of variables of the collection VARIABLES2 taking a value in VARIABLES1.

Note that we can exchange all occurrences of two distinct values of VARIABLES1 or VARIABLES2, or replace all occurrences of an assigned value of VARIABLES1 or VARIABLES2 by a new value, i.e., a value that is not yet assigned to any variable of VARIABLES1 and VARIABLES2. Within the electronic catalogue this is represented by the following meta-data, vals([VARIABLES1.var, VARIABLES2.var], int, $\neq$, all, dontcare), to which corresponds the following textual form:

All occurrences of two distinct values in VARIABLES1.var *or* VARIABLES2.var *can be swapped; all occurrences of a value in* VARIABLES1.var *or* VARIABLES2.var *can be renamed to any unused value.*

For example, since COMMON$(3, 4, \langle \mathbf{1}, 9, \mathbf{1}, 5\rangle, \langle 2, \mathbf{1}, 9, 9, 6, 9\rangle)$ is a solution, we can replace all occurrences of value 1 by a not yet assigned value 7, for instance, and get another valid solution COMMON$(3, 4, \langle \mathbf{7}, 9, \mathbf{7}, 5\rangle, \langle 2, \mathbf{7}, 9, 9, 6, 9\rangle)$.

The five parameters of vals([VARIABLES1.var, VARIABLES2.var], int, $\neq$, all, dontcare) have the following meaning:

- [VARIABLES1.var, VARIABLES2.var] indicates that the modification takes place within the values assigned to the var attribute of the VARIABLES1 and VARIABLES2 collections.
- int defines the partition of values $\mathcal{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots$.
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- all specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- dontcare tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in VARIABLES1 or VARIABLES2.

We now present an example that illustrates the fact that value symmetry can also occur between two arguments that both correspond to a domain variable, i.e., not just between the variables of a collection of variables. For this purpose we consider the LEQ constraint.

EXAMPLE 9: Consider the LEQ(VAR1, VAR2) constraint, which forces VAR1 to be less than or equal to VAR2. Note that VAR1 can be decreased to any value, and that VAR1 can be increased up to VAR2. Similarly, VAR2 can be increased to any value, and VAR2 can be decreased down to VAR1. Within the electronic catalogue this is respectively represented by the following meta-data, vals([VAR1], int($\leq$ (VAR2)), $\neq$, all, dontcare) and vals([VAR2], int($\geq$ (VAR1)), $\neq$, all, dontcare), to which corresponds the following textual form:

VAR1 *can be replaced by any value* $\leq$ VAR2*;*

VAR2 *can be replaced by any value* $\geq$ VAR1.

For example, since LEQ($\mathbf{2}, 9$) is a solution, we can replace value 2 by any value less than or equal to 9, e.g. value 5 and get another valid solution LEQ($\mathbf{5}, 9$). But, within LEQ($2, \mathbf{9}$), we can also replace value 9 with any other value greater than or equal to 2, e.g. value 4 and get another valid solution LEQ($2, \mathbf{4}$).

The five parameters of vals([VAR1], int($\leq$ (VAR2)), $\neq$, all, dontcare) have the following meaning:

- [VAR1] indicates that the modification takes place within the value assigned to the argument VAR1 of the constraint LEQ.
- int($\leq$ (VAR2)) defines the partition of values $\mathcal{P} = \ldots, \{\mathtt{VAR2} - 2\}, \{\mathtt{VAR2} - 1\}, \{\mathtt{VAR2}\}$ (i.e., we only consider values that are less than or equal to VAR2).
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- all specifies that all occurrences of the source value have to be replaced by the target value. Note that, since the scope of the change is reduced to a single variable, we have one occurrence of the source value and no occurrence of the target value.
- dontcare tells that the source value will be replaced by a new value.

The meta-data vals([VAR2], int($\geq$ (VAR1)), $\neq$, all, dontcare) has a similar explanation.

We now present two examples related to the K_DISJOINT constraint. The first example illustrates the fact the path specifying the scope of the exchange can contain more than one collection. The second example exemplifies the fact the path specifying the scope of the exchange does not necessarily start with a top level collection.

EXAMPLE 10: Consider the K_DISJOINT(SETS) constraint which, given |SETS| sets of domain variables, forces that no value is assigned to more than one set. Note that we can swap all the occurrences of two values, or replace all occurrences of a value by a value that is not yet used. Within the electronic catalogue this is represented by the following meta-data, vals([SETS.set.var], int, $\neq$, all, dontcare), to which corresponds the following textual form:

All occurrences of two distinct values of SETS.set.var *can be swapped; all occurrences of a value of* SETS.set.var *can be renamed to any unused value.*

For example, since K_DISJOINT($\langle$set $-$ $\langle \mathbf{1}, 9, \mathbf{1}, 5\rangle$, set $-$ $\langle 7, 2, 7\rangle\rangle$) is a solution, we can replace value 1 by any value that is different from the already used values 2, 5, 7, and 9, e.g. value 3, and get another valid solution K_DISJOINT($\langle$set $-$ $\langle \mathbf{3}, 9, \mathbf{3}, 5\rangle$, set $-$ $\langle 7, 2, 7\rangle\rangle$). From the solution K_DISJOINT($\langle$set $-$ $\langle \mathbf{1}, 9, \mathbf{1}, 5\rangle$, set $-$ $\langle 7, \mathbf{2}, 7\rangle\rangle$), we can also swap all occurrences of two values, e.g. values 1 and 2, and get another valid solution K_DISJOINT($\langle$set $-$ $\langle \mathbf{2}, 9, \mathbf{2}, 5\rangle$, set $-$ $\langle 7, \mathbf{1}, 7\rangle\rangle$).

The five parameters of vals([SETS.set.var], int, $\neq$, all, dontcare) have the following meaning:

- [SETS.set.var] indicates that the modification takes place within the values assigned to the var attribute of the SETS.set collections.
- int defines the partition of values $\mathcal{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots$.
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- all specifies that all occurrences of the source value have to be exchanged with all occurrences of the target value.
- dontcare tells that the source value can be replaced by an already existing value or by a new value, i.e., a value not already used in SETS.set.var.

EXAMPLE 11: Consider the K_DISJOINT(SETS) constraint which, given |SETS| sets of domain variables, forces that no value is assigned to more than one set. Note that, within any set, we can replace any occurrence of a value by another value that is already used in the same set. Within the electronic catalogue this is represented by the following meta-data, $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{int}, \neq, \mathtt{dontcare}, \mathtt{in})$, to which corresponds the following textual form:

An occurrence of a value of VARIABLES.var *can be replaced by any value of* VARIABLES.var.

For example, since K_DISJOINT($\langle \mathtt{set} - \langle \mathbf{1}, 9, 1, 5\rangle, \mathtt{set} - \langle 7, 2, 7\rangle\rangle$) is a solution, we can replace within the first set the first occurrence of value 1 by the already used value 5, and get another valid solution K_DISJOINT($\langle \mathtt{set} - \langle \mathbf{5}, 9, 1, 5\rangle, \mathtt{set} - \langle 7, 2, 7\rangle\rangle$).

The five parameters of $\mathtt{vals}([\mathtt{VARIABLES.var}], \mathtt{int}, \neq, \mathtt{dontcare}, \mathtt{in})$ have the following meaning:

- [VARIABLES.var] indicates that the modification takes place within the values assigned to the var attribute of the VARIABLES.var collections. Note that since the corresponding path does not start from a top level collection (i.e., VARIABLES does not correspond to an argument of the K_DISJOINT constraint), this represents one set of values for each set: the scope of value symmetry is located within a single set.
- int defines the partition of values $\mathcal{P} = \ldots, \{-1\}, \{0\}, \{1\}, \ldots$.
- $\neq$ indicates that the exchange of values takes place between two distinct elements of $\mathcal{P}$.
- dontcare specifies that one occurrence of the source value has to be replaced by the target value.
- in tells that the source value has to be replaced by an already existing value in VARIABLES.var.

We present a last example where the path specifying the scope of the exchange does not end with an attribute but rather with a collection. This can be seen as a generalisation of value symmetry where, instead of exchanging values, we exchange tuples of values. This kind of value symmetry occurs in constraints like COND_LEX_COST, IN_RELATION, NPAIR, NVECTOR, NVECTORS, or PATTERN.

EXAMPLE 12: Consider the NVECTOR(NVEC, VECTORS) constraint which forces an equality between NVEC and the number of distinct tuples of values taken by the vectors of the collection VECTORS. Note that we can swap all the occurrences of two tuples of values, or replace all occurrences of a tuple of values by a tuple of values that is not yet used. Within the electronic catalogue this is represented by the following meta-data, vals([VECTORS.vec], int, $\neq$, all, dontcare), to which corresponds the following textual form:

All occurrences of two distinct tuples of values of VECTORS.vec *can be swapped; all occurrences of a tuple of values of* VECTORS.vec *can be renamed to any unused tuple of values.*

For example, since NVECTOR(2, ⟨vec − ⟨**5**, **6**⟩, vec − ⟨9, 2⟩, vec − ⟨**5**, **6**⟩⟩) is a solution, we can replace all the occurrences of the tuple of values ⟨5, 6⟩ by any unused tuple of values, e.g. the tuple of values ⟨1, 2⟩, and get another valid solution NVECTOR(2, ⟨vec − ⟨**1**, **2**⟩, vec − ⟨9, 2⟩, vec − ⟨**1**, **2**⟩⟩).

The five parameters of vals([VECTORS.vec], int, $\neq$, all, dontcare) have the following meaning:

- [VECTORS.vec] indicates that the modification takes place within the tuples of values assigned to the vec attribute of the VECTORS collections.
- int defines the partition of values $\mathcal{P} = \mathbb{Z}^{|\mathtt{VECTORS}|}$.
- $\neq$ indicates that the exchange of tuple of values takes place between two distinct elements of $\mathcal{P}$.
- all specifies that all occurrences of the source tuple of values have to be exchanged with all occurrences of the target tuple of values.
- dontcare tells that the source tuple of values can be replaced by an already existing tuple of values or by a new tuple of values, i.e., a tuple of values not already used in VECTORS.vec.

- translate(ATTRIBUTES) denotes the fact that we add a constant to some collection attributes (i.e., we express the fact that solutions are preserved under some specific translation). ATTRIBUTES is a list of terms of the form ARG1, or ARG2.attr, or ARG3.attr$_\mathtt{i}$.attr$_\mathtt{j}$, where:

  - ARG1 is an argument of the global constraint of type domain variable or integer.
  - ARG2 is an argument of the global constraint that corresponds to a collection, and attr is an attribute of ARG2 of type domain variable or integer.
  - ARG3 is an argument of the global constraint that corresponds to a collection, and attr$_\mathtt{i}$ is an attribute of ARG3 of type collection, and attr$_\mathtt{j}$ is an attribute of ARG3.attr$_\mathtt{i}$ of type domain variable or integer.

  Its purpose is to define all the elements that have to be simultaneously incremented by one and the same constant.

  - The case corresponding to ARG1 is motivated by the fact that we sometimes want to increment an argument that is a domain variable or an integer.

– The case corresponding to $\texttt{ARG2.attr}$ is the standard case where we want to express that we increment attribute $\texttt{attr}$ of all items of a collection that is passed as an argument of the global constraint.

– Finally, the last case $\texttt{ARG3.attr}_\texttt{i}\texttt{.attr}_\texttt{j}$ corresponds to the fact that we want to increment attribute $\texttt{attr}_\texttt{j}$ of all items corresponding to $\texttt{ARG3.attr}_\texttt{i}$.

We now provide two examples, where the translation is respectively applied on a single attribute and on two attributes of a collection.

EXAMPLE 1: Consider the ALL_MIN_DIST($\texttt{MINDIST}, \texttt{VARIABLES}$) constraint which forces for each pair $(\texttt{var}_i, \texttt{var}_j)$ of distinct variables of the collection $\texttt{VARIABLES}$ that $|\texttt{var}_i - \texttt{var}_j| \geq \texttt{MINDIST}$. Note that we can add one and the same constant to all variables of the collection $\texttt{VARIABLES}$ since this does not change the difference between any pair of variables. Within the electronic catalogue this is represented by the following meta-data, $\texttt{translate}([\texttt{VARIABLES.var}])$, to which corresponds the following textual form:

One and the same constant can be added to the $\texttt{var}$ *attribute of all items of* $\texttt{VARIABLES}$.

For example, since ALL_MIN_DIST$(2, \langle \mathbf{5}, \mathbf{1}, \mathbf{9}, \mathbf{3} \rangle)$ is a solution, we can add the constant 6 to all items of the collection $\langle 5, 1, 9, 3 \rangle$, and get another valid solution ALL_MIN_DIST$(2, \langle \mathbf{11}, \mathbf{7}, \mathbf{15}, \mathbf{9} \rangle)$.

EXAMPLE 2: Consider the CUMULATIVE($\texttt{TASKS}, \texttt{LIMIT}$) constraint which forces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. Note that we can add one and the same constant to all $\texttt{origin}$ and $\texttt{end}$ attributes of the different tasks of the $\texttt{TASKS}$ collection. This operation simply shifts the overall schedule by a given constant without affecting the maximum resource consumption. Within the electronic catalogue this is represented by the following meta-data, $\texttt{translate}([\texttt{TASKS.origin}, \texttt{TASKS.end}])$, to which corresponds the following textual form:

One and the same constant can be added to the $\texttt{origin}$ *and* $\texttt{end}$ *attributes of all items of* $\texttt{TASKS}$.

For example, since

$$\left(\left\langle \begin{array}{llll} \texttt{origin} - \mathbf{1} & \texttt{duration} - 3 & \texttt{end} - \mathbf{4} & \texttt{height} - 1, \\ \texttt{origin} - \mathbf{2} & \texttt{duration} - 9 & \texttt{end} - \mathbf{11} & \texttt{height} - 2, \\ \texttt{origin} - \mathbf{3} & \texttt{duration} - 10 & \texttt{end} - \mathbf{13} & \texttt{height} - 1, \\ \texttt{origin} - \mathbf{6} & \texttt{duration} - 6 & \texttt{end} - \mathbf{12} & \texttt{height} - 1, \\ \texttt{origin} - \mathbf{7} & \texttt{duration} - 2 & \texttt{end} - \mathbf{9} & \texttt{height} - 3 \end{array} \right\rangle, 8 \right)$$

is a solution, we can add the constant 2 to all $\texttt{origin}$ and $\texttt{end}$ attributes, and get another valid solution

$$\left(\left\langle \begin{array}{llll} \texttt{origin} - \mathbf{3} & \texttt{duration} - 3 & \texttt{end} - \mathbf{6} & \texttt{height} - 1, \\ \texttt{origin} - \mathbf{4} & \texttt{duration} - 9 & \texttt{end} - \mathbf{13} & \texttt{height} - 2, \\ \texttt{origin} - \mathbf{5} & \texttt{duration} - 10 & \texttt{end} - \mathbf{15} & \texttt{height} - 1, \\ \texttt{origin} - \mathbf{8} & \texttt{duration} - 6 & \texttt{end} - \mathbf{14} & \texttt{height} - 1, \\ \texttt{origin} - \mathbf{9} & \texttt{duration} - 2 & \texttt{end} - \mathbf{11} & \texttt{height} - 3 \end{array} \right\rangle, 8 \right).$$

We conclude by listing other types of symmetries that we may also consider in the future, namely:

- In the context of graph constraints we can usually relabel the vertices of the corresponding graph. This is the case, for example, of the CIRCUIT constraint where the `index` attribute corresponds to the name of a vertex.

- In the context of constraints on a matrix we can have symmetries on both the rows and the columns of the matrix. On the one hand, since a row corresponds to a collection this can be currently expressed. On the other hand, since a column corresponds to all the i^{th} items of the collections corresponding to the rows, this currently cannot be expressed.

- Given a collection of items, we want to express a symmetry on different subsets of items: more precisely, on all items for which a given attribute is assigned the same value. As an illustrative example consider the CUMULATIVES constraint. We would like to express the possibility of translating the origin of all tasks that are assigned the same machine.

- Given a collection of items we can sometimes multiply by -1 all occurrences of one of its attributes. This usually corresponds to a mirror symmetry. This is the case, for example, for the `origin` attribute of the CUMULATIVE constraint.

2.3 Describing global constraints in terms of graph properties

Through a practical example, we first present in a simplified form the basic principles used for describing the meaning of global constraints in terms of graph properties. We then give the full details about the different features used in the description process.

2.3.1 Basic ideas and illustrative example

Within the graph-based representation, a global constraint is represented as a digraph where each vertex corresponds to a variable and each arc to a binary arc constraint between the variables associated with the extremities of the corresponding arc. The main difference from classical constraint networks [142], stems from the fact that we do not force any more all arc constraints to hold. We rather consider this graph from which we discard all the arc constraints that do not hold as well as all isolated vertices (i.e, vertices not involved any more in any arc) and impose one or several graph properties on this remaining graph. These properties can be, for example, a restriction on the number of connected components, on the size of the smallest connected component or on the size of the largest connected component.

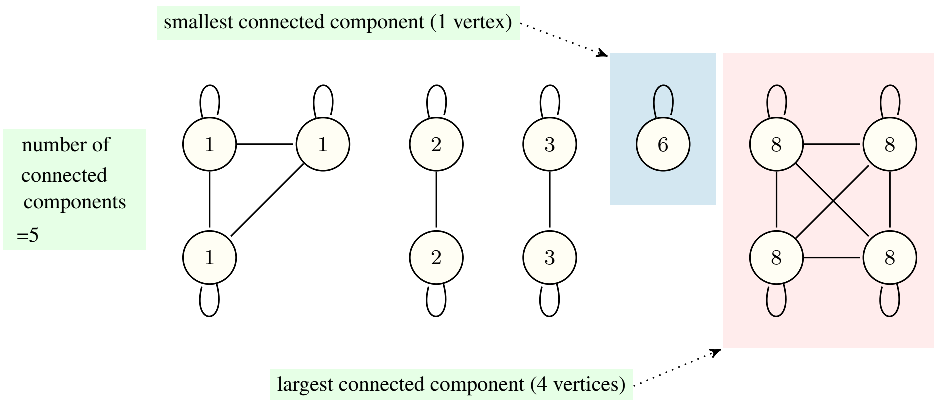


Figure 2.5: Illustration of the link between graph-properties and global constraints

EXAMPLE: We give an example of interpretation of such graph properties in terms of global constraints. For this purpose we consider the sequence s of values 1 3 1 1 2 8 8 2 3 6 8 8 3 from which we construct the following graph G:

- To each value associated with a position in s corresponds a vertex of G,
- There is an arc from a vertex v_1 to a vertex v_2 if these vertices correspond to the same value.

Figure 2.5 depicts graph G. Since G is symmetric, we omit the directions of the arcs. We have the following correspondence between graph properties and constraints on the sequence s:

- The number of connected components of G corresponds to the number of distinct values of s.
- The size of the smallest connected component of G is the smallest number of occurrences of the same value in s.
- The size of the largest connected component of G is the largest number of occurrences of the same value in s.

As a result, in this context, putting a restriction on the number of connected components of G can been seen as a global constraint on the number of distinct values of a sequence of variables. Similar global constraints can be associated with the two other graph properties.

We now explain how to generate the initial graph associated with a global constraint. A global constraint has one or more arguments, which usually correspond to an integer value, to one variable or to a collection of variables. Therefore we have to describe the process that allows for generating the vertices and the arcs of the initial graph from the arguments of a global constraint under consideration. For this purpose we will take a concrete example.

Consider the constraint NVALUE(NVAL, VARIABLES) where NVAL and VARIABLES respectively correspond to a domain variable and to a collection of domain variables

$\langle \mathtt{var} - V_1, \mathtt{var} - V_2, \ldots, \mathtt{var} - V_m\rangle$.[12] This constraint holds if NVAL is equal to the number of distinct values assigned to the variables $V_1, V_2, \ldots, V_m$. We first show how to generate the initial graph associated with the NVALUE constraint. We then describe the arc constraint associated with each arc of this graph. Finally, we give the graph property we impose on the final graph.

To each variable of the collection VARIABLES corresponds a vertex of the initial graph. We generate an arc between each pair of vertices. To each arc, we associate an equality constraint between the variables corresponding to the extremities of that arc. We impose that NVAL, the variable corresponding to the first argument of NVALUE, be equal to the number of strongly connected components of the final graph. This final graph consists of the initial graph from which we discard all arcs such that the corresponding equality constraint does not hold.

Part (A) of Figure 2.6 shows the graph initially generated for the constraint NVALUE $(\mathtt{NVAL}, \langle \mathtt{var} - V_1, \mathtt{var} - V_2, \mathtt{var} - V_3, \mathtt{var} - V_4\rangle)$, where NVAL, V_1, V_2, V_3 and V_4 are domain variables. Part (B) presents the final graph associated with the ground instance NVALUE$(3, \langle \mathtt{var}-5, \mathtt{var}-5, \mathtt{var}-1, \mathtt{var}-8\rangle)$. For each vertex of the initial and final graph we respectively indicate the corresponding variable and the value assigned to that variable. We have removed from the final graph all the arcs associated with equalities that do not hold. The constraint NVALUE$(3, \langle \mathtt{var} - 5, \mathtt{var} - 5, \mathtt{var} - 1, \mathtt{var} - 8\rangle)$ holds since the final graph contains three strongly connected components, which in the context of the definition of the NVALUE constraint, can be reinterpreted as the fact that NVAL is the number of distinct values assigned to variables V_1, V_2, V_3, V_4.

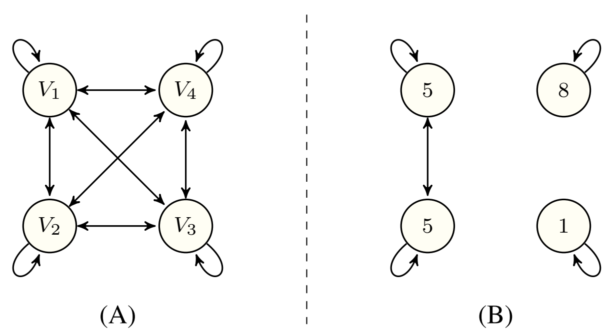


Figure 2.6: (A) Initial and (B) final graph associated with the constraint NVALUE$(3, \langle \mathtt{var} - 5, \mathtt{var} - 5, \mathtt{var} - 1, \mathtt{var} - 8\rangle)$

Now that we have illustrated the basic ideas for describing a global constraint in terms of graph properties, we go into more details.

2.3.2 Ingredients used for describing global constraints

We first introduce the basic ingredients used for describing a global constraint and illustrate them shortly on the example of the NVALUE constraint introduced in the previous

[12] var corresponds to the name of the attribute used in the collection of variables.

section on page 49. We then go through each basic ingredient in more detail. The graph-based description is founded on the following basic ingredients:

- *Data types* and *restrictions* used in order to describe the arguments of a global constraint. Data types and restrictions were already described in the previous section (from page 14 to page 27).

- *Collection generators* used in order to derive new collections from the arguments of a global constraint for one of the following reasons:

  – Collection generators are sometimes required since the initial graph of a global constraint cannot always be directly generated from the arguments of the global constraint. The NVALUE(NVAL, VARIABLES) constraint did not require any collection generator since the vertices of its initial graph were directly generated from the VARIABLES collection.

  – A second use of collection generators is for deriving a collection of items for different set of vertices of the final graph. This is sometimes required when we use *set generators* (see the last item of the enumeration).

- *Elementary constraints* associated with the arcs of the initial and final graph of a global constraint. The NVALUE constraint was using an *equality* constraint, but other constraints are usually required.

- *Graph generators* employed for constructing the initial graph of a global constraint. In the context of the NVALUE constraint the initial graph was a *clique*. As we will see later, other patterns are needed for generating an initial graph.

- *Graph properties* and *graph classes* used for constraining the final graph we want to obtain. In the context of the NVALUE constraint we were using the *number of strongly connected components* for counting the number of distinct values.

- *Set generators* that may be used for generating specific sets of vertices of the final graph on which we want to enforce a given constraint. Since the NVALUE constraint forces a graph property on the final graph (and not on subparts of the final graph) we did not use this feature.

We first start to explain each ingredient separately and then show how one can describe most global constraints in terms of these basic ingredients.

Collection generators

The vertices of the initial graph are usually directly generated from collections of items that are arguments of the global constraint G under consideration. However, it sometimes happens that we would like to derive a new collection from existing arguments of G in order to produce the vertices of the initial graph.

EXAMPLE: This is the case, for example, of the ELEMENT(INDEX, TABLE, VALUE) constraint, where INDEX and VALUE are domain variables that we would like to group as a single item $\mathcal{I}$ (with two attributes) of a new derived collection. This is in fact done in order to generate the following initial graph:

- The item $\mathcal{I}$ as well as all items of TABLE constitute the vertices,
- There is an arc from $\mathcal{I}$ to each item of the TABLE collection.

We provide the following mechanism for deriving new collections:

- In a first phase we declare the name of the new collection as well as the names of its attributes and their respective types. This is achieved exactly in the same way as those collections that are used in the arguments of a global constraint (see page 16).

  EXAMPLE: Consider again the example of the ELEMENT(INDEX, TABLE, VALUE) constraint. The declaration $\texttt{ITEM} - \texttt{collection}(\texttt{index} - \texttt{dvar}, \texttt{value} - \texttt{dvar})$ introduces a new collection called ITEM where each item has an index and a value attribute. Both attributes correspond to domain variables.

- In a second phase we give a list of patterns that are used for generating the items of the new collection. A pattern $o - \texttt{item}(\texttt{a}_1 - v_1, \texttt{a}_2 - v_2, \ldots, \texttt{a}_n - v_n)$ or $\texttt{item}(\texttt{a}_1 - v_1, \texttt{a}_2 - v_2, \ldots, \texttt{a}_n - v_n)$ specifies for each attribute $\texttt{a}_i (1 \leq i \leq n)$ of the new collection how to fill it.[13] This is done by providing for each attribute $\texttt{a}_i$ one of the following expression v_i:

  - A constant.
  - An argument of the global constraint G.
  - An expression c.a, where a is an attribute of a collection c, such that c is an argument of the global constraint G or a derived collection that was previously declared. An expression of this form is called a *direct reference to an attribute of a collection*.
  - An expression $\texttt{c}_1.\texttt{c}_2.\texttt{a}$, where a is an attribute of a collection $\texttt{c}_2$, and $\texttt{c}_2$ is an attribute of a collection $\texttt{c}_1$ such that $\texttt{c}_1$ is an argument of the global constraint G or a derived collection that was previously declared. An expression of this form is called an *indirect reference to an attribute of a collection*.

  This expression v_i must be compatible with the type declaration of the corresponding attribute of the new collection.

[13] o is one of the comparison operators $=, \neq, <, \geq, >, \leq$. When omitted its default value is $=$.

EXAMPLE: We continue the example of the ELEMENT(INDEX, TABLE, VALUE) constraint and the derived collection ITEM − collection(index − dvar, value − dvar). The pattern item(index − INDEX, value − VALUE) indicates that:

- The index attribute of the ITEM collection will be generated by using the INDEX argument of the ELEMENT constraint. Since INDEX is a domain variable, it is compatible with the declaration ITEM − collection(index − dvar, value − dvar) of the new collection.
- The value attribute of the ITEM collection will be generated by using the VALUE argument of the ELEMENT constraint. VALUE is also compatible with the declaration statement of the new collection.

We now describe how we use the pattern for generating the items of a derived collection. We have the following two cases:

- If the pattern $o - \mathtt{item}(\mathtt{a}_1 - v_1, \mathtt{a}_2 - v_2, \ldots, \mathtt{a}_n - v_n)$ does not contain any direct or indirect reference to an attribute of a collection then we generate a single item for such pattern.[14] In this context the value v_i of the attribute $\mathtt{a}_i$ $(1 \leq i \leq n)$ corresponds to a constant, to an argument of the global constraint or to a new derived collection.
- If the pattern $o - \mathtt{item}(\mathtt{a}_1 - v_1, \mathtt{a}_2 - v_2, \ldots, \mathtt{a}_n - v_n)$, where o is one of the comparison operators $=, \neq, <, \geq, >, \leq$, contains one or several direct or indirect references to an attribute of a collection[15] we denote by:
  - $\mathcal{D}$ the set of indices of the positions corresponding to a direct reference to an attribute of a collection within $\mathtt{item}(\mathtt{a}_1 - v_1, \mathtt{a}_2 - v_2, \ldots, \mathtt{a}_n - v_n)$. In this context, let $\mathtt{c}_{\alpha_1}, \mathtt{c}_{\alpha_2}, \ldots, \mathtt{c}_{\alpha_m}$ and $\mathtt{a}_{\alpha_1}, \mathtt{a}_{\alpha_2}, \ldots, \mathtt{a}_{\alpha_m}$ respectively denote the corresponding collections and attributes.
  - $\mathcal{I}$ the set of indices of the positions corresponding to an indirect reference to an attribute of a collection within $\mathtt{item}(\mathtt{a}_1 - v_1, \mathtt{a}_2 - v_2, \ldots, \mathtt{a}_n - v_n)$. In this context, let $\mathtt{c}^1_{\beta_1}, \mathtt{c}^1_{\beta_2}, \ldots, \mathtt{c}^1_{\beta_p}$, $\mathtt{c}^2_{\beta_1}, \mathtt{c}^2_{\beta_2}, \ldots, \mathtt{c}^2_{\beta_p}$ and $\mathtt{a}_{\beta_1}, \mathtt{a}_{\beta_2}, \ldots, \mathtt{a}_{\beta_p}$ respectively denote the corresponding collections, attributes of type collection and attributes.
  - Let $dir_1, dir_2, \ldots, dir_m$, $ind_1, ind_2, \ldots, ind_p$ and $id_1, id_2, \ldots, id_{m+p}$ respectively denote the indices sorted in increasing order of $\mathcal{D}$, $\mathcal{I}$ and $\mathcal{D} \cup \mathcal{I}$.

  For each combination of items $\mathtt{c}_{\alpha_1}[i_1], \mathtt{c}_{\alpha_2}[i_2], \ldots, \mathtt{c}_{\alpha_m}[i_m]$, $\mathtt{c}^1_{\beta_1}[j_1].\mathtt{c}^2_{\beta_1}[k_1]$, $\mathtt{c}^1_{\beta_2}[j_2].\mathtt{c}^2_{\beta_2}[k_2]$, $\ldots$, $\mathtt{c}^1_{\beta_p}[j_p].\mathtt{c}^2_{\beta_p}[k_p]$ such that:

$$\begin{cases} i_1 \in [1, |\mathtt{c}_{\alpha_1}|], & i_2 \in [1, |\mathtt{c}_{\alpha_2}|], & \ldots, & i_m \in [1, |\mathtt{c}_{\alpha_m}|] \\ j_1 \in [1, |\mathtt{c}^1_{\beta_1}|], & j_2 \in [1, |\mathtt{c}^1_{\beta_2}|], & \ldots, & j_p \in [1, |\mathtt{c}^1_{\beta_p}|] \\ k_1 \in [1, |\mathtt{c}^1_{\beta_1}[j_1].\mathtt{c}^2_{\beta_1}|], & k_2 \in [1, |\mathtt{c}^1_{\beta_2}[j_2].\mathtt{c}^2_{\beta_2}|], & \ldots, & k_p \in [1, |\mathtt{c}^1_{\beta_p}[j_p].\mathtt{c}^2_{\beta_p}|] \\ id_1 \ o \ id_2 \ o \ \ldots \ o \ id_{m+p} \end{cases}$$

  we generate an item of the new derived collection $\langle \mathtt{a}_1 - w_1 \ \mathtt{a}_2 - w_2 \ \ldots \ \mathtt{a}_n - w_n \rangle$ defined by:

[14] In this first case the value of o is irrelevant.

[15] This collection is an argument of the global constraint or corresponds to a newly derived collection.

$$w_j(1 \le j \le n) = \begin{cases} \mathtt{c}_{\alpha_r}[i_r].\mathtt{a}_{\alpha_r} & \text{if} j \in \mathcal{D}, j = dir_r \\ \mathtt{c}^1_{\beta_r}[j_r].\mathtt{c}^2_{\beta_r}[k_r].\mathtt{a}_{\beta_r} & \text{if} j \in \mathcal{I}, j = ind_r \\ v_j & \text{if} j \notin \mathcal{D} \cup \mathcal{I} \end{cases}.$$

We illustrate this generation process on a set of examples. Each example is described by providing:

- The global constraint and its arguments,
- The declaration of the new derived collection,
- The pattern used for creating an item of the new collection,
- The items generated by applying this pattern to the global constraint,
- A comment about the generation process.

We first start with four examples that do not mention any references to an attribute of a collection. A box surrounds an argument of a global constraint that is mentioned in a generated item.

EXAMPLE

CONSTRAINT : ELEMENT($\boxed{\mathtt{INDEX}}$, TABLE, $\boxed{\mathtt{VALUE}}$)

DERIVED COLLECTION: ITEM − collection(index − dvar, value − dvar)

PATTERN(S) : item(index − INDEX, value − VALUE)

GENERATED ITEM(S) : ⟨index−$\boxed{\mathtt{INDEX}}$ value−$\boxed{\mathtt{VALUE}}$⟩

We generate a single item where the two attributes index and value respectively take the first argument INDEX and the third argument VALUE of the ELEMENT constraint.

EXAMPLE

CONSTRAINT : LEX_LESSEQ(VECTOR1, VECTOR2)

DERIVED COLLECTION: DESTINATION − collection(index − int, x − int, y − int)

PATTERN(S) : item(index − 0, x − 0, y − 0)

GENERATED ITEM(S) : ⟨index − 0 x − 0 y − 0⟩

We generate a single item where the three attributes index, x and y take value 0.

EXAMPLE

CONSTRAINT : IN_RELATION($\boxed{\mathtt{VARIABLES}}$, TUPLES_OF_VALS)

DERIVED COLLECTION: TUPLES_OF_VARS − collection(vec − TUPLE_OF_VARS)

PATTERN(S) : item(vec − VARIABLES)

GENERATED ITEM(S) : ⟨vec−$\boxed{\mathtt{VARIABLES}}$⟩

We generate a single item where the unique attribute vec takes the first argument of the IN_RELATION constraint as its value.

EXAMPLE

CONSTRAINT : DOMAIN_CONSTRAINT($\boxed{\texttt{VAR}}$, VALUES)

DERIVED COLLECTION: VALUE − collection(var01 − int, value − dvar)

PATTERN(S) : item(var01 − 1, value − VAR)

GENERATED ITEM(S) : ⟨var01 − 1 value−$\boxed{\texttt{VAR}}$⟩

We generate a single item where the two attributes `var01` and `value` respectively take value 1 and the first argument of the DOMAIN_CONSTRAINT constraint.

We continue with three examples that mention one or several direct references to an attribute of some collections. We now need to explicitly give the items of these collections in order to generate the items of the derived collection.

EXAMPLE

CONSTRAINT : LEX_LESSEQ($\boxed{\texttt{VECTOR1}}$, $\boxed{\texttt{VECTOR2}}$)

VECTOR1 : ⟨var − 5, var − 2, var − 3, var − 1⟩

VECTOR2 : ⟨var − 5, var − 2, var − 6, var − 2⟩

DERIVED COLLECTION: COMPONENTS − collection(index − int, x − dvar, y − dvar)

PATTERN(S) : item(index − VECTOR1.key[a], x − VECTOR1.var, y − VECTOR2.var)

GENERATED ITEM(S) : ⟨index − 1 x − 5 y − 5, index − 2 x − 2 y − 2, index − 3 x − 3 y − 6, index − 4 x − 1 y − 2⟩

The pattern mentions three references `VECTOR1.key`, `VECTOR1.var` and `VECTOR2.var` to the collections `VECTOR1` and `VECTOR2` used in the arguments of the LEX_LESSEQ constraint. $\forall i_1 \in [1, |\texttt{VECTOR1}|], \forall i_2 \in [1, |\texttt{VECTOR2}|]$ such that $i_1 = i_2$[b] we generate an item index − v_1 x − v_2 y − v_3 where:

$$v_1 = i_1, \qquad v_2 = \texttt{VECTOR1}[i_1].\texttt{var}, \qquad v_3 = \texttt{VECTOR2}[i_1].\texttt{var}.$$

This leads to the four items listed in the `GENERATED ITEM(S)` field.

[a] As defined in Section 2.2.2 on page 16, `key` is an implicit attribute corresponding to the position of an item within a collection.

[b] We use an equality since this is the default value of the comparison operator o when we do not use a pattern of the form o − item(...).

EXAMPLE

CONSTRAINT : CUMULATIVES($\boxed{\texttt{TASKS}}$, MACHINES, CTR)

TASKS : $\langle$machine − 1 origin − 1 duration − 4 end − 5 height − 1,
machine − 1 origin − 4 duration − 2 end − 6 height − 3,
machine − 1 origin − 2 duration − 3 end − 5 height − 2,
machine − 2 origin − 5 duration − 2 end − 7 height − 2$\rangle$

DERIVED COLLECTION: TIME_POINTS − collection(idm − int,
duration − dvar, point − dvar)

PATTERN(S) : item(idm − TASKS.machine,
duration − TASKS.duration, point − TASKS.origin)
item(idm − TASKS.machine,
duration − TASKS.duration, point − TASKS.end)

GENERATED ITEM(S) : $\langle$idm − 1 duration − 4 point − 1,
idm − 1 duration − 2 point − 4,
idm − 1 duration − 3 point − 2,
idm − 2 duration − 2 point − 5,
idm − 1 duration − 4 point − 5,
idm − 1 duration − 2 point − 6,
idm − 1 duration − 3 point − 5,
idm − 2 duration − 2 point − 7$\rangle$

The two patterns mention the references TASKS.machine, TASKS.duration, TASKS.origin and TASKS.end of the TASKS collection used in the arguments of the CUMULATIVES constraint. $\forall i \in [1, |\texttt{TASKS}|]$, we generate two items idm − u_1 duration − u_2 point − u_3 , idm − v_1 duration − v_2 point − v_3 where:

$u_1 = \texttt{TASKS}[i]\texttt{.machine}$, $u_2 = \texttt{TASKS}[i]\texttt{.duration}$, $u_3 = \texttt{TASKS}[i]\texttt{.origin}$,
$v_1 = \texttt{TASKS}[i]\texttt{.machine}$, $v_2 = \texttt{TASKS}[i]\texttt{.duration}$, $v_3 = \texttt{TASKS}[i]\texttt{.end}$.

This leads to the eight items listed in the GENERATED ITEM(S) field.

EXAMPLE

CONSTRAINT : GOLOMB($\boxed{\texttt{VARIABLES}}$)

VARIABLES : $\langle$var − 0, var − 1, var − 4, var − 6$\rangle$

DERIVED COLLECTION: PAIRS − collection(x − dvar, y − dvar)

PATTERN(S) : > −item(x − VARIABLES.var, y − VARIABLES.var)

GENERATED ITEM(S) : $\langle$x − 1 y − 0,
x − 4 y − 0, x − 4 y − 1,
x − 6 y − 0, x − 6 y − 1, x − 6 y − 4$\rangle$

The pattern mentions two references VARIABLES.var and VARIABLES.var to the VARIABLES collection used in the arguments of the GOLOMB constraint. $\forall i_1 \in [1, |\texttt{VARIABLES}|]$, $\forall i_2 \in [1, |\texttt{VARIABLES}|]$ such that $i_1 > i_2$[a] we generate the item x − u_1 y − u_2 where:

$u_1 = \texttt{VARIABLES}[i_1]\texttt{.var}$, $u_2 = \texttt{VARIABLES}[i_2]\texttt{.var}$.

This leads to the six items listed in the GENERATED ITEM(S) field.

[a] We use the comparison operator > since we have a pattern of the form > −item(. . .).

We finish with an example that mentions an indirect reference to an attribute of a collection.

EXAMPLE

CONSTRAINT	: CUMULATIVE_CONVEX($\boxed{\texttt{TASKS}}$, LIMIT)
TASKS	: $\langle$points $-$ $\langle$var $-$ 2, var $-$ 1, var $-$ 5$\rangle$ height $-$ 1, points $-$ $\langle$var $-$ 4, var $-$ 5, var $-$ 7$\rangle$ height $-$ 2, points $-$ $\langle$var $-$ 14, var $-$ 15$\rangle$ height $-$ 2$\rangle$
DERIVED COLLECTION:	INSTANTS $-$ collection(instant $-$ int)
PATTERN(S)	: item(instant $-$ TASKS.points.var)
GENERATED ITEM(S)	: $\langle$instant $-$ 2, instant $-$ 1, instant $-$ 5, instant $-$ 4, instant $-$ 5, instant $-$ 7, instant $-$ 14, instant $-$ 15$\rangle$

The pattern mentions the indirect reference TASKS.points.var of the TASKS collection used in the arguments of the CUMULATIVE_CONVEX constraint. $\forall i \in [1, |\texttt{TASKS}|]$, $\forall j \in [1, |\texttt{TASKS}[i].\texttt{points}|]$ we generate the item instant $- u_{ij}$ where:
$u_{ij} = \texttt{TASKS}[i].\texttt{points}[j]$.
This leads to the eight items listed in the GENERATED ITEM(S) field.

Elementary constraints attached to the arcs

This section describes the constraints that are associated with the arcs of the initial graph of a global constraint. These constraints are called *arc constraints*. To each arc one can associate one or several arc constraints. An arc will belong to the final graph if and only if all its arc constraints hold. An arc constraint from a vertex v_1 to a vertex v_2 mentions variables and/or values associated with v_1 and v_2. Before defining an *arc constraint*, we first need to introduce *simple arithmetic expressions* as well as *arithmetic expressions*. Simple arithmetic expressions and arithmetic expressions are defined recursively.

Simple arithmetic expressions A *simple arithmetic expression* is defined by one of the five following expressions.

- I : I is an integer.
- Arg : Arg is an argument of the global constraint of type int or dvar.
- Arg : Arg is a formal parameter provided by the arc generator[16] of the graph-constraint.
- Col.Attr : Col is a formal parameter provided by the arc generator or the collection used in the For all items of iterator.[17] Attr is an attribute of the collection referenced by Col.

[16] Arc generators are described in Section 2.3.2 on page 61.
[17] The For all items of iterator is described in Section 2.3.3 on page 80.

EXAMPLE: As an example consider the first graph-constraint associated with the GLOBAL_CARDINALITY_WITH_COSTS(VARIABLES, VALUES, MATRIX, COST) constraint and its arc constraint variables.var = VALUES.val. Both, variables.var as well as VALUES.val are *simple arithmetic expressions* of the form Col.Attr:

- In variables.var, variables corresponds to the formal parameter provided by the arc generator $SELF \mapsto$ collection(variables), while var is an attribute of the VARIABLES collection.
- In VALUES.val, VALUES corresponds to the collection denoted by the For all items of iterator, while val is an attribute of the VALUES collection.

- Col[Expr].Attr : Col is an argument of type collection, Attr one attribute of Col and Expr an *arithmetic expression*.

  Col[Expr].Attr denotes the value of attribute Attr of the Exprth item of the collection denoted by Col.

EXAMPLE: As an example consider the GLOBAL_CARDINALITY_WITH_COSTS(VARIABLES, VALUES, MATRIX, COST) constraint and its second graph-constraint, which defines the COST variable. The expression MATRIX[(variables.key − 1) ∗ |VALUES|+ values.key].c is a *simple arithmetic expression* of the form Col[Expr].Attr:

- MATRIX is a collection of items collection(i − int, j − int, c − int) where all items are sorted in increasing order on attributes i, j (because of the restriction increasing_seq(MATRIX, [i, j])).
- MATRIX[(variables.key − 1) ∗ |VALUES| + values.key].c denotes the value of attribute c of an item of the MATRIX collection. The position of this item within the MATRIX collection depends on the position of a variable of the VARIABLES collection[a] as well as on the position of a value of the VALUES collection.[b]

[a]This position is denoted by the expression variables.key. As defined in Section 2.2.2 on page 16, key is an implicit attribute corresponding to the position of an item within a collection.
[b]This position is denoted by the expression values.key.

Arithmetic expressions An *arithmetic expression* is recursively defined by one of the following expressions:

- A *simple arithmetic expression*.
- Exp_1 Op Exp_2
  - Exp_1 is an *arithmetic expression*,
  - Op is one of the following symbols $+$, $-$, $*$, $/$[18],
  - Exp_2 is an *arithmetic expression*.
- |Collection|
  - Collection is an argument of type collection and |Collection| denotes the number of items of that collection.

[18] / denotes an integer division, a division in which the fractional part is discarded.

- $|\texttt{Exp}|$
  - Exp is an *arithmetic expression*, and $|\texttt{Exp}|$ denotes the absolute value of this expression.

- sign(Exp)
  - Exp is an *arithmetic expression*, and sign(Exp) the sign of Exp (-1 if Exp is negative, 0 if Exp is equal to 0, 1 if Exp is positive).

EXAMPLE: An example of use of sign can be found in the last part of the arc constraint of the CROSSING constraint:
$\texttt{sign}((\texttt{s2.ox} - \texttt{s1.ex}) * (\texttt{s1.ey} - \texttt{s1.oy}) - (\texttt{s1.ex} - \texttt{s1.ox}) * (\texttt{s2.oy} - \texttt{s1.ey})) \neq$
$\texttt{sign}((\texttt{s2.ex} - \texttt{s1.ex}) * (\texttt{s2.oy} - \texttt{s1.oy}) - (\texttt{s2.ox} - \texttt{s1.ox}) * (\texttt{s2.ey} - \texttt{s1.ey}))$

- card_set(Set) :
  - Set is a reference to a set of integers or to a set variable. card_set(Set) denotes the number of elements of that set.

EXAMPLE: An example of use of card_set can be found in the SYMMETRIC_GCC constraint: vars.nocc = card_set(vars.var).

- $\texttt{SimpleExp}_1 \bmod \texttt{SimpleExp}_2$,

  $\min(\texttt{SimpleExp}_1, \texttt{SimpleExp}_2)$ or $\max(\texttt{SimpleExp}_1, \texttt{SimpleExp}_2)$
  - $\texttt{SimpleExp}_1$ is a *simple arithmetic expression*,
  - $\texttt{SimpleExp}_2$ is a *simple arithmetic expression*.

Arc constraints Now that we have introduced *simple arithmetic expressions* as well as *arithmetic expressions* we define an *arc constraint*. An *arc constraint* is recursively defined by one of the following expressions:

- TRUE

  This stands for an arc constraint that always holds. As a result, the corresponding arc always belongs to the final graph.

EXAMPLE: An example of use of TRUE can be found in the SUM_CTR(VARIABLES, CTR, VAR) constraint, where it is used in order to enforce keeping all items of the VARIABLES collection in the final graph.

- $\texttt{Exp}_1$ Comparison $\texttt{Exp}_2$
  - $\texttt{Exp}_1$ is an *arithmetic expression*,
  - Comparison is one of the comparison operators $\leq$, $\geq$, $<$, $>$, $=$, $\neq$,
  - $\texttt{Exp}_2$ is an *arithmetic expression*.

EXAMPLE: As an example of such arc constraint, the second graph-constraint of the CUMULATIVE(TASKS, LIMIT) constraint uses the following arc constraints:

- tasks1.duration > 0,
- tasks2.origin $\leq$ tasks1.origin,
- tasks1.origin $<$ tasks2.end.

The conjunction of these three arc constraints can be interpreted in the following way: an arc from a task tasks1 to a task tasks2 will belong to the final graph if and only if tasks2 overlaps the origin of tasks1.

- $\texttt{Exp}_1$ SimpleCtr $\texttt{Exp}_2$
  - $\texttt{Exp}_1$ is an *arithmetic expression*,
  - SimpleCtr is an argument of type atom that can only take one of the values $\leq, \geq, <, >, =, \neq$,
  - $\texttt{Exp}_2$ is an *arithmetic expression*.

EXAMPLE: An example of use of such an arc constraint can be found in the CHANGE(NCHANGE, VARIABLES, CTR) constraint: variables1.var CTR variables2.var. Within this expression, variables1 and variables2 correspond to consecutive items of the VARIABLES collection.

- $\texttt{Exp}_1$ $\neg$SimpleCtr $\texttt{Exp}_2$
  - $\texttt{Exp}_1$ is an *arithmetic expression*,
  - SimpleCtr is an argument of type atom that can only take one of the values $\leq, \geq, <, >, =, \neq$,
  - $\texttt{Exp}_2$ is an *arithmetic expression*.

EXAMPLE: An example of use of such an arc constraint can be found in the CHANGE_CONTINUITY(NB_PERIOD_CHANGE, NB_PERIOD_CONTINUITY, MIN_SIZE_CHANGE, MAX_SIZE_CHANGE, MIN_SIZE_CONTINUITY, MAX_SIZE_CONTINUITY, NB_CHANGE, NB_CONTINUITY, VARIABLES, CTR) constraint: variables1.var $\neg$CTR variables2.var. Within this expression, variables1 and variables2 correspond to consecutive items of the VARIABLES collection.

- constraint($\texttt{Exp}_1, \ldots, \texttt{Exp}_n$)
  - constraint is a global constraint defined in the catalogue for which there exists a graph-based and/or an automaton-based representation,
  - $\texttt{Exp}_1, \ldots, \texttt{Exp}_n$ correspond to the arguments of the global constraint constraint. Each argument should be a *simple arithmetic expression* that is compatible with the type declaration of the argument of constraint.

EXAMPLE: An example of such arc constraint can be found in the definition of DIFFN: DIFFN(ORTHOTOPES) uses the TWO_ORTH_DO_NOT_OVERLAP(ORTHOTOPE1, ORTHOTOPE2) global constraint for defining its arc constraint. Since ORTHOTOPES is a collection of type collection(ori − dvar, siz − dvar, end − dvar) and since both ORTHOTOPE1 and ORTHOTOPE2 correspond to items of ORTHOTOPES there is no type compatibility problem between the call to TWO_ORTH_DO_NOT_OVERLAP and its definition.

- $\texttt{ArcCtr}_1$ LogicalConnector $\texttt{ArcCtr}_2$
  - $\texttt{ArcCtr}_1$ is an *arc constraint*,
  - LogicalConnector is one of the logical connectors $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$,
  - $\texttt{ArcCtr}_2$ is an *arc constraint*.

EXAMPLE: As shown by the following example, MINIMUM(MIN, VARIABLES) uses this kind of arc constraint: variables1 = variables2 $\vee$ variables1.var $<$ variables2.var, where variables1 and variables2 correspond to items of the VARIABLES collection, holds if and only if one of the following conditions holds:

- variables1 and variables2 correspond to the same item of the VARIABLES collection,
- The var attribute of variables1 is strictly less than the var attribute of variables2.

Graph generators

This section describes how to generate the initial graph associated with a global constraint. Initial graphs correspond to directed hypergraphs [64], which have a very regular structure. They are defined in the following way:

- The vertices of the directed hypergraph are generated from collections of items such that each item corresponds to one vertex of the directed hypergraph. These collections are either collections that arise as arguments of the global constraint, or collections that are derived from one or several arguments of the global constraint. In this latter case these *derived collections* are computed by using the *collection generators* previously introduced (see Section 2.3.2 on page 51).

- To all arcs of the directed hypergraph corresponds the same arc constraint that involves vertices in a given order.[19] These arc constraints, which are mainly unary and binary constraints, were described in the previous section (see Section 2.3.2 on page 57). We describe all the arcs of an initial graph with a set of predefined *arc generators*, which correspond to classical regular structures one can find in the graph literature [402, pages 140–153]. An *arc generator* of arity a takes n collections of items, denoted $\mathtt{c}_i (1 \leq i \leq n)$, as input and returns the corresponding hypergraph where the vertices are the items of the input collections

[19] Usually the edges of a hypergraph are not oriented [64, pages 1–2]. However for our purpose we need to define an order on the vertices of an edge since the corresponding arc constraint takes its arguments in a given order.

$\mathtt{c}_i(1 \leq i \leq n)$ and where all arcs involve a vertices. Specific arc generators allow for giving an a-ary constraint for which a is not fixed, which means that the corresponding hypergraph contains arcs involving various number of vertices.

Each arc generator has a name and takes one or several collections of items as input and generates a set of arcs. Each arc is made from a sequence of items $i_1\ i_2\ \ldots\ i_a$ and is denoted by $(i_1, i_2, \ldots, i_a)$. a is called the *arity* of the arc generator. We have the following types of arc generators:

- Arc generators with a fixed predefined arity. In fact most arc generators have a fixed predefined arity of 2. The graphs they generate correspond to digraphs.
- Arc generators that can be used with any arity a greater than or equal to 1. These arc generators generate directed hypergraphs where all arcs consist of a items.
- Arc generators that generate arcs that do not involve the same number of items.

We now give the list of arc generators, listed in alphabetic order, and the arcs they generate. For each arc generator we point to a global constraint where it is used in practice. Finally, Figure 2.8 illustrates the different arc generators. At present the following arc generators are in use:

- *CHAIN* has a predefined arity of 2. It takes one collection $\mathtt{c}$ and generates the following arcs[20]:

  - $\forall i \in [1, |\mathtt{c}| - 1]$: $(\mathtt{c}[i], \mathtt{c}[i+1])$,
  - $\forall i \in [1, |\mathtt{c}| - 1]$: $(\mathtt{c}[i+1], \mathtt{c}[i])$.

  EXAMPLE: The arc generator *CHAIN* is used, for example, in the GROUP_SKIP_ISOLATED_ITEM constraint.

- *CIRCUIT* has a predefined arity of 2. It takes one collection $\mathtt{c}$ and generates the following arcs:

  - $\forall i \in [1, |\mathtt{c}| - 1]$: $(\mathtt{c}[i], \mathtt{c}[i+1])$,
  - $(\mathtt{c}[|\mathtt{c}|], \mathtt{c}[1])$.

  EXAMPLE: The arc generator *CIRCUIT* is used, for example, in the CIRCULAR_CHANGE constraint.

- *CLIQUE* can be used with any arity a greater than or equal to 2. It takes one collection $\mathtt{c}$ and generates the arcs: $\forall i_1 \in [1, |\mathtt{c}|], \forall i_2 \in [1, |\mathtt{c}|], \ldots, \forall i_a \in [1, |\mathtt{c}|] : (\mathtt{c}[i_1], \mathtt{c}[i_2], \ldots, \mathtt{c}[i_a])$.

  EXAMPLE: The arc generator *CLIQUE* is usually used with an arity $a = 2$. For example, this is the case of the ALLDIFFERENT constraint.

[20]As defined in Section 2.2.2 on page 16 we use the following notation: for a given collection $\mathtt{c}$, $|\mathtt{c}|$ and $\mathtt{c}[i]$ respectively denote the number of items of $\mathtt{c}$ and the i^{th} item of $\mathtt{c}$.

- $CLIQUE$(`Comparison`), where `Comparison` is one of the comparison operators $\leq, \geq, <, >, =, \neq$, can be used with any arity a greater than or equal to 2. It takes one collection `c` and generates the arcs:

  $\forall i_1 \in [1, |\mathtt{c}|]$,

  $\forall i_2 \in [1, |\mathtt{c}|]$ such that i_1 `Comparison` i_2,

  . . . . ,

  $\forall i_a \in [1, |\mathtt{c}|]$ such that i_{a-1} `Comparison` i_a : $(\mathtt{c}[i_1], \mathtt{c}[i_2], \ldots, \mathtt{c}[i_a])$.

  EXAMPLE: The ORCHARD(`TREES`) constraint is an example of constraint that uses the $CLIQUE(<)$ arc generator with an arity $a = 3$. It generates an arc for each set of three trees.

- $CYCLE$ has a predefined arity of 2. It takes one collection `c` and generates the following arcs:

  - $\forall i \in [1, |\mathtt{c}| - 1]$ $(\mathtt{c}[i], \mathtt{c}[i+1])$ and $(\mathtt{c}[i+1], \mathtt{c}[i])$,
  - $(\mathtt{c}[|\mathtt{c}|], \mathtt{c}[1])$ and $(\mathtt{c}[1], \mathtt{c}[|\mathtt{c}|])$.

  The arc generator $CYCLE$ is currently not used.

- $GRID([d_1, d_2, \ldots, d_n])$ takes a collection `c` consisting of $d_1 \cdot d_2 \cdot \ \cdots \ \cdot d_n$ items and generates the arcs $(\mathtt{c}[i], \mathtt{c}[j])$ where i and j satisfy the following condition. There exists an integer α $(0 \leq \alpha \leq n-1)$ such that (1) and (2) hold:

  (1) $|i - j| = \prod_{1 \leq k \leq \alpha} d_k$ (when $\alpha = 0$ we have $\prod_{1 \leq k \leq \alpha} = 1$),

  (2) $\lfloor \frac{i}{\prod_{1 \leq k \leq \alpha+1} d_k} \rfloor = \lfloor \frac{j}{\prod_{1 \leq k \leq \alpha+1} d_k} \rfloor$.

  EXAMPLE: The CONNECT_POINTS constraint uses the $GRID$ arc generator.

- $LOOP$ has a predefined arity of 2. It takes one collection `c` and generates the arcs: $\forall i \in [1, |\mathtt{c}|]$: $(\mathtt{c}[i], \mathtt{c}[i])$. $LOOP$ is usually used in order to generate a loop on some vertices, so that they do not disappear from the final graph.

  EXAMPLE: The GLOBAL_CONTIGUITY(`VARIABLES`) constraint is an example of constraint that uses the $LOOP$ arc generator so that each variable of the `VARIABLES` collection belongs to the final graph.

- $PATH$ can be used with any arity a greater than or equal to 1. It takes one collection `c`, and generates the following arcs: $\forall i \in [1, |\mathtt{c}| - a + 1] : (\mathtt{c}[i], \mathtt{c}[i+1], \ldots, \mathtt{c}[i+a-1])$.

  EXAMPLE: $PATH$ is used, for example, in the SLIDING_SUM(`LOW`, `UP`, `SEQ`, `VARIABLES`) constraint with an arity `SEQ`, where `SEQ` is an argument of the SLIDING_SUM constraint.

- *PATH_1* generates arcs that do not involve the same number of items. It takes one collection c, and generates the following arcs: (c[1]), (c[1], c[2]), ... , (c[1], c[2], ... , c[|c|]).

> **EXAMPLE:** *PATH_1* is used in the SIZE_MAX_STARTING_SEQ_ALLDIFFERENT constraint.

- *PATH_N* generates arcs that do not involve the same number of items. It takes one collection c, and generates the following arcs: $\forall i \in [1, |c|], \forall j \in [i, |c|]$: (c[i], c[$i+1$], ... , c[j]).

> **EXAMPLE:** *PATH_N* is used, for example, in the SIZE_MAX_SEQ_ALLDIFFERENT constraint.

- *PRODUCT* has a predefined arity of 2. It takes two collections c_1, c_2 and generates the arcs: $\forall i \in [1, |c_1|], \forall j \in [1, |c_2|]$: ($c_1[i]$, $c_2[j]$).

> **EXAMPLE:** *PRODUCT* is used, for example, in the SAME(VARIABLES1, VARIABLES2) constraint for generating an arc from every item of the VARIABLES1 collection to every item of the VARIABLES2 collection.

- *PRODUCT*(Comparison), where Comparison is one of the comparison operators $\leq$, $\geq$, $<$, $>$, $=$, $\neq$, has a predefined arity of 2. It takes two collections c_1, c_2 and generates the arcs: $\forall i \in [1, |c_1|], \forall j \in [1, |c_2|]$ such that i Comparison j : ($c_1[i]$, $c_2[j]$).

> **EXAMPLE:** *PRODUCT*(=) is used, for example, in the DIFFER_FROM_AT_LEAST_K_POS(K, VECTOR1, VECTOR2) constraint in order to generate an arc between the i^{th} component of VECTOR1 and the i^{th} component of VECTOR2.

- *SELF* has a predefined arity of 1. It takes one collection c and generates the arcs: $\forall i \in [1, |c|]$: (c[i]).

> **EXAMPLE:** *SELF* is used, for example, in the AMONG(NVAR, VARIABLES, VALUES) constraint in order to generate a unary arc constraint IN(variables.var, VALUES) for each variable of the VARIABLES collection.

- *SYMMETRIC_PRODUCT* has a predefined arity of 2. It takes two collections c_1, c_2 and generates the following arcs: $\forall i \in [1, |c_1|], \forall j \in [1, |c_2|]$: ($c_1[i]$, $c_2[j]$) and ($c_2[j]$, $c_1[i]$).

> **EXAMPLE:** *SYMMETRIC_PRODUCT* is used, for example, in the INVERSE_WITHIN_RANGE constraint.

- *SYMMETRIC_PRODUCT*(Comparison), where Comparison is one of the comparison operators $\leq$, $\geq$, $<$, $>$, $=$, $\neq$, has a predefined arity of 2. It takes two collections c_1, c_2 and generates the arcs: $\forall i \in [1, |c_1|], \forall j \in [1, |c_2|]$ such that i Comparison j : ($c_1[i]$, $c_2[j]$) and ($c_2[j]$, $c_1[i]$).

EXAMPLE: The TWO_ORTH_DO_NOT_OVERLAP constraint is an example of constraint that uses the $SYMMETRIC\_PRODUCT(=)$ arc generator.

- $VOID$ takes one collection and does not generate any arc.

EXAMPLE: $VOID$ is used, for example, in the LEX_LESSEQ constraint.

Finally, we can combine the $PRODUCT$ arc generator with the arc generators from the following set $\mathcal{G}enerator = \{CIRCUIT, CHAIN, CLIQUE, LOOP, PATH, VOID\}$. This is achieved by using the construction $PRODUCT(\mathtt{G}_1, \mathtt{G}_2)$ where $\mathtt{G}_1$ and $\mathtt{G}_2$ belong to $\mathcal{G}enerator$. It applies $\mathtt{G}_1$ to the first collection $\mathtt{c}_1$ passed to $PRODUCT$ and $\mathtt{G}_2$ to the second collection $\mathtt{c}_2$ passed to $PRODUCT$. Finally, it applies $PRODUCT$ on $\mathtt{c}_1$ and $\mathtt{c}_2$. In a similar way the $PRODUCT(\mathtt{Comparison})$ arc generator is extended to $PRODUCT(\mathtt{G}_1, \mathtt{G}_2, \mathtt{Comparison})$.

EXAMPLE: As an illustrative example, consider the ALLDIFFERENT_SAME_VALUE(NSAME, VARIABLES1, VARIABLES2) constraint, which uses the arc generator $PRODUCT(CLIQUE, LOOP, =)$ on the collections VARIABLES1 and VARIABLES2. It generates the following arcs:

- Since the first argument of $PRODUCT$ is $CLIQUE$ it generates an arc between each pair of items of the VARIABLES1 collection.
- Since the second argument of $PRODUCT$ is $LOOP$ it generates a loop for each item of the VARIABLES2 collection.
- Since the third argument is the comparison operator $=$ it finally generates an arc between an item of the VARIABLES1 collection and an item of the VARIABLES2 collection when the two items have the same position.

Figure 2.7 shows the generated graph under the hypothesis that VARIABLES1 and VARIABLES2 have respectively 3 and 3 items.

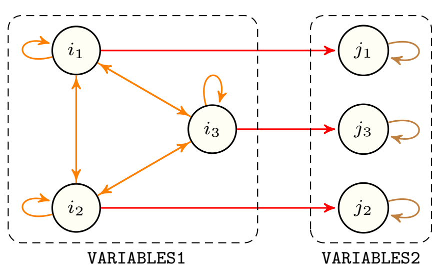


Figure 2.7: Example of initial graph generated by PRODUCT(CLIQUE, LOOP, $=$) when applied to collections VARIABLES1 and VARIABLES2

Figure 2.8 illustrates the different arc generators. On the one hand, for those arc generators that take a single collection, we apply them on the collection of items $\langle i-1, i-2, i-3, i-4\rangle$. On the other hand, for those arc generators that take two collections, we apply them on $\langle i-1, i-2\rangle$ and $\langle i-3, i-4\rangle$. We use the following pictogram for the graphical representation of a constraint network:

- A line for an arc constraint of arity 1,
- An arrow for an arc constraint of arity 2,
- A closed line for an arc constraint with an arity strictly greater than 2. In this last case, since the vertices of an arc are ordered, a black circle at one of the extremities indicates the direction of the closed line. For example, consider the example of *PATH_1* in Figure 2.8. The closed line that contains vertices 1, 2 and 3 means that a 3-ary arc constraint involves items 1, 2, and 3 in this specific order.

Dotted circles represent vertices that do not belong to the graph. This stems from the fact the arc generator did not produce any arc involving these vertices. The leftmost lowest corner indicates the arity of the corresponding arc generator:

- An integer if it has a fixed predefined arity,
- n if it can be used with any arity greater than or equal to 1,
- $*$ if it generates arcs that do not necessarily involve the same number of items.

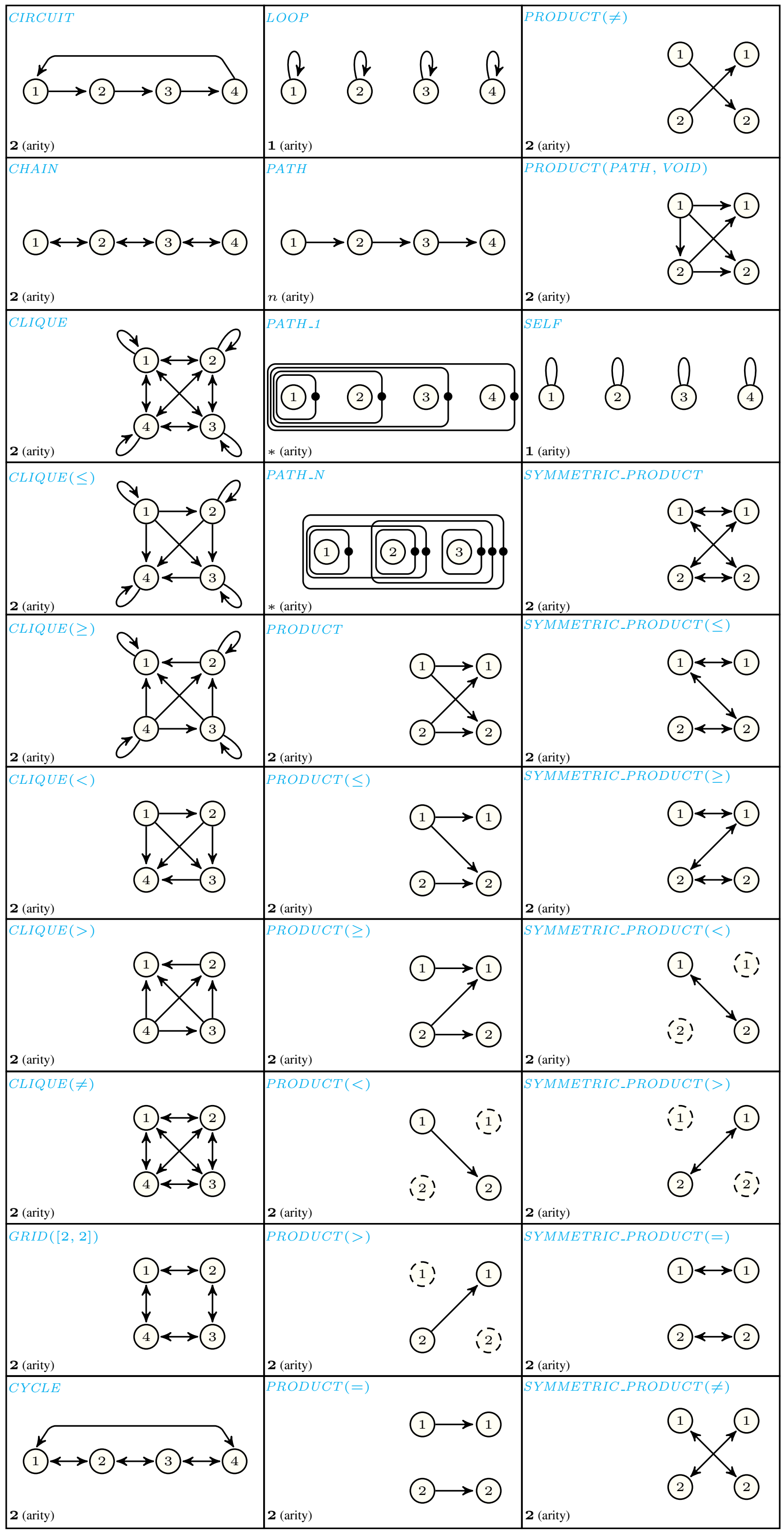


Figure 2.8: Examples of arc generators

Graph properties

We represent a global constraint as the search of a subgraph (i.e., a final graph) of a known initial graph, so that this final graph satisfies a given set of graph properties and possibly belongs to a specific graph class. Most graph properties have the form `Parameter Comparison Exp` or the form `Parameter` $\notin$ $[\mathtt{Exp}_1, \mathtt{Exp}_2]$, where `Parameter` is a graph parameter [63], [208], `Comparison` is one of the comparison operators $=$, $<$, $\geq$, $>$, $\leq$, $\neq$, and `Exp`, $\mathtt{Exp}_1$, $\mathtt{Exp}_2$ are expressions that can be evaluated to an integer. Before defining each graph parameter, let's first introduce some basic vocabulary on graphs.

Graph terminology and notations A *digraph* $G = (V(G), E(G))$ is a pair where $V(G)$ is a finite set, called the set of *vertices*, and where $E(G)$ is a set of ordered pairs of vertices, called the set of *arcs*. The *arc*, *path*, *circuit* and *strongly connected component* of a graph G correspond to oriented concepts, while the *edge*, *chain*, *cycle* and *connected component* are non-oriented concepts. However, as reported in [63, page 6] an undirected graph can be seen as a digraph where to each edge we associate the corresponding two arcs. Parts (A) and (B) of Figure 2.9 respectively illustrate the terms for undirected graphs and digraphs.

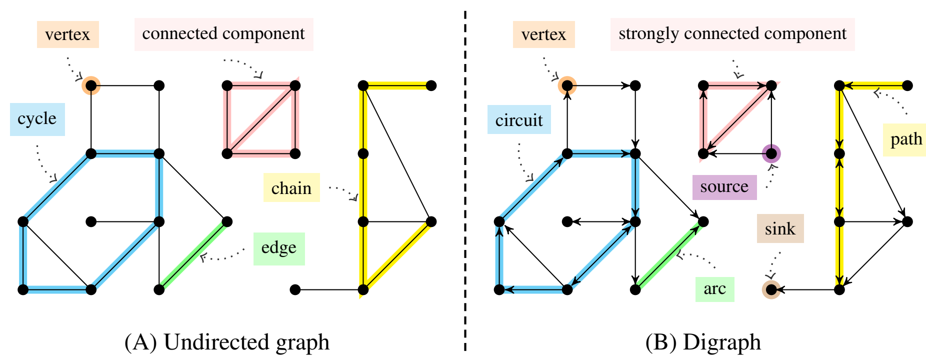


Figure 2.9: Graph terminology for an undirected graph and a digraph (similar concepts are outlined with the same colour)

- We say that e_2 is a *successor* of e_1 if there exists an arc that starts from e_1 and ends at e_2. In the same way, we say that e_2 is a *predecessor* of e_1 if there exists an arc that starts from e_2 and ends at e_1.

- A vertex of G that does not have any predecessor is called a *source*. A vertex of G that does not have any successor is called a *sink*.

- A sequence $(e_1, e_2, \ldots, e_k)$ of edges of G such that each edge has a common vertex with the previous edge, and the other vertex common to the next edge is called a *chain* of length k. A chain where all vertices are distinct is called an *elementary chain*. Each equivalence class of the relation "e_i is equal to e_j or there exists a chain between e_i and e_j" is a *connected component* of the graph G.

- A sequence $(e_1, e_2, \ldots, e_k)$ of arcs of G such that, for each arc e_i $(1 \leq i < k)$ the end of e_i is equal to the start of the arc e_{i+1}, is called a *path* of length k. A path where all vertices are distinct is called an *elementary path*. Each equivalence class of the relation "e_i is equal to e_j or there exists a path between e_i and e_j" is a *strongly connected component* of the graph G.

- A chain $(e_1, e_2, \ldots, e_k)$ of G is called a *cycle* if the same edge does not occur more than once in the chain and if the two extremities of the chain coincide. A cycle $(e_1, e_2, \ldots, e_k)$ of G is called a *circuit* if for each edge e_i $(1 \leq i < k)$, the end of e_i is equal to the start of the edge e_{i+1}.

- Given a graph G, we define the *reduced graph* $R(G)$ of G as follows: to each strongly connected component of G corresponds a vertex of $R(G)$; to each arc of G that connects different strongly connected components corresponds an arc in $R(G)$ (multiple arcs between the same pair of vertices are merged).

- The *rank* function associated with the vertices $V(G)$ of a graph G that does not contain any circuit is defined in the following way:

  – The rank of the vertices that do not have any predecessor (i.e., the sources) is equal to 0,

  – The rank r of a vertex v that is not a source is the length of longest path $(e_1,\ e_2,\ \ldots,\ e_r)$ such that the start of the arc e_1 is a source and the end of arc e_r is the vertex v.

We now present the different notations used in the catalogue:

- $[k]$ corresponds to $\{1, \cdots, k\}$ for k any positive integer.

- Given a set X, $|X|$ is the number of its elements.

- Given two sets X and Y, $X \biguplus Y$ denotes the union of the two sets when they are disjoint.

- Given a digraph G and $x \in V(G)$, $d_G^+(x) = |\{y : y \in V(G) : (x, y) \in E(G)\}|$ and $d_G^-(x) = |\{y : y \in V(G) : (y, x) \in E(G)\}|$.

- Given a digraph G and X a subset of $V(G)$, the sub-digraph of G induced by X is the digraph $G[X]$ where $V(G[X]) = X$ and $E(G[X]) = X^2 \cap E(G)$. By aim of simplicity, we denote $G[V(G) - X]$ by $G - X$. Moreover, if $X = \{x\}$, we use $G - x$ instead of $G - \{x\}$.

- Given two digraph G_1 and G_2 such that $V(G_1) \cap V(G_2) = \emptyset$, $G_1 \oplus G_2$ denotes the graph whose vertices set is $V(G_1) \cup V(G_2)$ and whose arcs set is $E(G_1) \cup E(G_2)$.

- Given a graph parameter $\mathbf{P} \in \{\mathbf{NCC}, \mathbf{NSCC}\}$, a digraph G and an integer k, $\mathbf{CH}(G, k)$ is the number of connected components (respectively strongly connected components) of G with cardinal k.

Given a graph parameter, for example, the number of connected components, $\mathbf{NCC}_{\texttt{INITIAL}}$ denotes the number of connected components of the initial graph (i.e., the graph induced by the constraint under consideration), **NCC** denotes the number of connected components of the final graph (i.e., a subgraph of the initial graph). $\mathbf{NCC}(G)$ denotes the number of connected components of the digraph G.

Given a global constraint C, and a graph parameter $\mathbf{P}$ used in the description of C, $\underline{\mathbf{P}}$ (respectively $\overline{\mathbf{P}}$) denotes a lower bound (respectively upper bound) of $\mathbf{P}$ among all possible final graphs compatible with the current status of C.

Graph parameters We list in alphabetic order the different *graph parameters* we consider for a final graph $G_f = (V(G_f), E(G_f))$ associated with a global constraint and give an example of constraint where they are used:

- **MAX_DRG** : largest distance between sources and sinks in the reduced graph associated with G_f (adjacent vertices are at a distance of 1).

  EXAMPLE: We do not provide any example since **MAX_DRG** is currently not used.

- **MAX_ID** : number of predecessors of the vertex of G_f that has the maximum number of predecessors without counting an arc from a vertex to itself.

  EXAMPLE: The CIRCUIT constraint uses the graph property $\mathbf{MAX\_ID} = 1$ in order to force each vertex of the final graph to have at most one predecessor.

- **MAX_NCC** : number of vertices of the largest connected component of G_f.

  EXAMPLE: The LONGEST_CHANGE(`SIZE`, `VARIABLES`, `CTR`) constraint uses the graph property $\mathbf{MAX\_NCC} = \texttt{SIZE}$ in order to catch in `SIZE` the maximum number of consecutive variables of the `VARIABLES` collection for which constraint `CTR` holds.

- **MAX_NSCC** : number of vertices of the largest strongly connected component of G_f.

  EXAMPLE: The TREE constraint covers a digraph by a set of trees in such a way that each vertex belongs to a distinct tree. It uses the graph-property $\mathbf{MAX\_NSCC} \leq 1$ in order to avoid to have any circuit involving more than one vertex.

- **MAX_OD** : number of successors of the vertex of G_f that has the maximum number of successors without counting an arc from a vertex to itself.

> **EXAMPLE:** The TOUR constraint forces to cover a graph with a Hamiltonian cycle. It uses the graph-property $\mathbf{MAX\_OD} = 2$ to enforce that each vertex of G_f have at most two[a] successors.
>
> [a] Since the TOUR constraint uses the $CLIQUE(\neq)$ arc generator the vertices of G_f do not have any loop.

- **MIN_DRG** : smallest distance between sources and sinks in the reduced graph associated with G_f (adjacent vertices are at a distance of 1).

> **EXAMPLE:** We do not provide any example since **MIN_DRG** is currently not used by any constraint.

- **MIN_ID** : number of predecessors of the vertex of G_f that has the minimum number of predecessors without counting an arc from a vertex to itself.

> **EXAMPLE:** The TOUR constraint forces to cover a graph with a Hamiltonian cycle. It uses the graph-property $\mathbf{MIN\_ID} = 2$ to enforce that each vertex of G_f have at most two[a] predecessors.
>
> [a] Since the TOUR constraint uses the $CLIQUE(\neq)$ arc generator the vertices of G_f do not have any loop.

- **MIN_NCC** : number of vertices of the smallest connected component of G_f.

> **EXAMPLE:** Within the GROUP constraint, each connected component of G_f corresponds to a maximum sequence of consecutive variables that take their values in a given set of values. Therefore, the graph-property $\mathbf{MIN\_NCC} =$ `MIN_SIZE` forces that the smallest sequence of such variables consist of `MIN_SIZE` variables.

- **MIN_NSCC** : number of vertices of the smallest strongly connected component of G_f.

> **EXAMPLE:** The CIRCUIT$(NODES)$ constraint forces covering a digraph with one circuit visiting once all its vertices. The graph-property $\mathbf{MIN\_NSCC} = |\texttt{NODES}|$ forces that the smallest strongly connected component of G_f contain $|\texttt{NODES}|$ vertices. Since $|\texttt{NODES}|$ also corresponds to the number of vertices of the initial graph this means that G_f is a strongly connected component involving all the vertices. This is clearly a necessary condition[a] for having a circuit visiting once all vertices.
>
> [a]Of course, this is not enough, and the description of the CIRCUIT constraint asks for some other properties.

- **MIN_OD** : number of successors of the vertex of G_f that has the minimum number of successors without counting an arc from a vertex to itself.

> **EXAMPLE:** The TOUR constraint forces to cover a graph with a Hamiltonian cycle. It uses the graph-property $\mathbf{MIN\_OD} = 2$ to enforce that each vertex of G_f have at most two[a] successors.
>
> [a]Since the TOUR constraint uses the $CLIQUE(\neq)$ arc generator the vertices of G_f do not have any loop.

- **NARC** : cardinality of the set $E(G_f)$.

> **EXAMPLE:** The DISJOINT(`VARIABLES1`, `VARIABLES2`) constraint forces that each variable of the collection `VARIABLES1` take a value that is distinct from all the values assigned to the variables of the collection `VARIABLES2`.
> This is imposed by creating an arc from each variable of `VARIABLES1` to each variable of `VARIABLES2`. To each arc corresponds an equality constraint involving the variables associated with the extremities of the arc. Finally, the graph property $\mathbf{NARC} = 0$ forces G_f to be empty so that no value is both assigned to a variable of `VARIABLES1` as well as to a variable of `VARIABLES2`.

- **NARC_NO_LOOP** : cardinality of the set $E(G_f)$ without considering the arcs linking the same vertex (i.e., a loop).

> **EXAMPLE:** The constraint ALLDIFFERENT_SAME_VALUE uses the **NARC_NO_LOOP** graph-property.

- **NCC** : number of connected components of G_f.

EXAMPLE: The TREE constraint covers a digraph by NTREES trees in such a way that each vertex belongs to a distinct tree. It uses the graph-property **NCC** = NTREES in order to state that G_f is made up from NTREES connected components.

- **NSCC** : number of strongly connected components of G_f.

EXAMPLE: The constraint NVALUE(NVAL, VARIABLES) forces NVAL to be equal to the number of distinct values assigned to the variables of the collection VARIABLES. This is enforced by using the graph-property **NSCC** = NVAL. Each strongly connected component of the final graph corresponds to the variables that are assigned to the same value.

- **NSINK** : number of vertices of G_f that do not have any successor.

EXAMPLE: The SAME(VARIABLES1, VARIABLES2) forces that the variables of the VARIABLES1 collection correspond to the variables of the VARIABLES2 collection according to a permutation.
We first create an arc from each variable of VARIABLES1 to each variable of VARIABLES2. To each arc corresponds an equality constraint involving the variables associated with the extremities of the arc. We use the graph-property **NSINK** = |VARIABLES2| in order to express the fact that each value assigned to a variable of VARIABLES2 should also be assigned to a variable of VARIABLES1.

- **NSINK_NSOURCE** : sum over the different connected components of G_f of the minimum of the number of sinks and the number of sources of a connected component.

EXAMPLE: The SOFT_SAME_VAR(C, VARIABLES1, VARIABLES2) constraint forces C to be the minimum number of values to change in the VARIABLES1 and the VARIABLES2 collections of variables[a], so that the variables of VARIABLES2 correspond to the variables of VARIABLES1 according to a permutation.
A connected component $\mathcal{C}_{val}$ of the final graph G_f corresponds to all variables that are assigned to the same value val: the sources and the sinks of $\mathcal{C}_{val}$ respectively correspond to the variables of VARIABLES1 and to the variables of VARIABLES2 that are assigned to val. For a connected component, the minimum of the number of sources and sinks expresses the number of variables for which we do not need to make any change. Therefore we use the graph-property **NSINK_NSOURCE** = |VARIABLES1| − C for encoding the meaning of the SOFT_SAME_VAR constraint.

[a] Both collections have the same number of variables.

- **NSOURCE** : number of vertices of G_f that do not have any predecessor.

> **EXAMPLE:** The SAME(VARIABLES1, VARIABLES2) forces that the variables of the VARIABLES1 collection correspond to the variables of the VARIABLES2 collection according to a permutation.
> We first create an arc from each variable of VARIABLES1 to each variable of VARIABLES2. To each arc corresponds an equality constraint involving the variables associated with the extremities of the arc. We use the graph-property $\mathbf{NSOURCE} = |\mathtt{VARIABLES1}|$ in order to express the fact that each value assigned to a variable of VARIABLES1 should also be assigned to a variable of VARIABLES2.

- **NTREE** : number of vertices of G_f that do not belong to any circuit and for which at least one successor belongs to a circuit. Such vertices can be interpreted as root nodes of a tree.

> **EXAMPLE:** The CYCLE(NCYCLE, NODES) forces that NCYCLE equal the number of circuits for covering an initial graph in such a way that each vertex belongs to a single circuit.
> The graph-property $\mathbf{NTREE} = 0$ forces that all vertices of the final graph belong to a circuit.

- **NVERTEX** : cardinality of the set $V(G_f)$.

> **EXAMPLE:** The CUTSET(SIZE_CUTSET, NODES) constraint considers a digraph with n vertices described by the NODES collection. It forces that the subset of kept vertices of cardinality $n -$ SIZE_CUTSET and their corresponding arcs form a graph without a circuit. It uses the graph-property $\mathbf{NVERTEX} = n - \mathtt{SIZE\_CUTSET}$ for enforcing that the final graph G_f contain the required number of vertices.

- **RANGE_DRG** : difference between the largest distance between sources and sinks in the reduced graph associated with G_f and the smallest distance between sources and sinks in the reduced graph associated with G_f.

> **EXAMPLE:** The TREE_RANGE constraint forces to cover a digraph in such a way that each vertex belongs to a distinct tree. In addition it forces the difference between the longest and the shortest paths of G_f to be equal to the variable R. For this purpose it uses the graph-property $\mathbf{RANGE\_DRG} = \mathtt{R}$.

- **RANGE_NCC** : difference between the number of vertices of the largest connected component of G_f and the number of vertices of the smallest connected component of G_f.

EXAMPLE: We do not provide any example since **RANGE_NCC** is currently not used by any constraint.

- **RANGE_NSCC** : difference between the number of vertices of the largest strongly connected component of G_f and the number of vertices of the smallest strongly connected component of G_f.

EXAMPLE: The BALANCE($\texttt{BALANCE}, \texttt{VARIABLES}$) constraint forces BALANCE to be equal to the difference between the number of occurrences of the value that occurs the most and the value that occurs the least within the collection of variables VARIABLES. Each strongly connected component of G_f corresponds to the variables that are assigned to the same value. The graph property $\textbf{RANGE\_NSCC} = \texttt{BALANCE}$ allows for expressing this definition.

- $\textbf{ORDER}(\texttt{rank}, \texttt{default}, \texttt{attr})$

  - `rank` is an integer or an argument of type integer of the global constraint,
  - `default` is an integer,
  - `attr` is an attribute corresponding to an integer or to a domain variable that occurs in all the collections that were used for generating the vertices of the initial graph.

We explain what is the value associated with $\textbf{ORDER}(\texttt{rank}, \texttt{default}, \texttt{attr})$. Let $\mathcal{V}$ denotes the vertices of rank `rank` of G_f from which we remove any loops.

  - When $\mathcal{V}$ is not empty, it corresponds to the values of attribute `attr` of the items associated with the vertices of $\mathcal{V}$,
  - Otherwise, when $\mathcal{V}$ is empty, it corresponds to the default value `default`.

EXAMPLE: The MINIMUM($\texttt{MIN}, \texttt{VARIABLES}$) forces MIN to be the minimum value of the collection of domain variables VARIABLES. There is an arc from a variable $\texttt{var}_1$ to a variable $\texttt{var}_2$ if and only if $\texttt{var}_1 < \texttt{var}_2$. The graph-property $\textbf{ORDER}(0, \texttt{MAXINT}, \texttt{var}) = \texttt{MIN}$ expresses the fact that MIN is equal to the value of the source of G_f (since $\texttt{rank} = 0$).

- $\textbf{PATH\_FROM\_TO}(\texttt{attr}, \texttt{from}, \texttt{to})$

  - * `attr` is an attribute corresponding to an integer that occurs in all the collections that were used for generating the vertices of the initial graph,

    * `from` is an integer or an argument of type integer of the global constraint,
    * `to` is an integer or an argument of type integer of the global constraint.

  Let $\mathcal{F}$ (respectively $\mathcal{T}$) denotes the vertices of G_f such that `attr` is equal to `from` (respectively `to`). $\mathbf{PATH\_FROM\_TO}(\texttt{attr}, \texttt{from}, \texttt{to})$ is equal to 1 if there exists a path between each vertex of $\mathcal{F}$ and each vertex of $\mathcal{T}$, and 0 if there exists no path between a vertex of $\mathcal{F}$ and a vertex of $\mathcal{T}$.

- * `attr` is an attribute corresponding to an integer that occurs in all the collections that were used for generating the vertices of the initial graph,
  * `from` is an attribute corresponding to an integer or to a set of integers that occurs in all the collections that were used for generating the vertices of the initial graph,
  * `to` is an attribute corresponding to an integer or to a set of integers that occurs in all the collections that were used for generating the vertices of the initial graph,

  For each vertex v of G_f let:

  * $\mathcal{F}_v$ the set of vertices for which the value of the attribute `attr` is equal to the `from` attribute (or is included within the `from` attribute when it corresponds to a set of integers).
  * $\mathcal{T}_v$ the set of vertices for which the value of the attribute `attr` is equal to the `to` attribute (or is included within the `to` attribute when it corresponds to a set of integers).

  $\mathbf{PATH\_FROM\_TO}(\texttt{attr}, \texttt{from}, \texttt{to})$ is equal to

  * 1 if for each vertex of G_f there exists a path between each vertex of $\mathcal{F}_v$ and each vertex of $\mathcal{T}_v$.
  * 0 if for a vertex of G_f there is no path between a vertex of $\mathcal{F}_v$ and a vertex of $\mathcal{T}_v$.

EXAMPLE: The constraints LEX_LESSEQ and STABLE_COMPATIBILITY use the **PATH_FROM_TO** graph-property.

- $\mathbf{PROD}(\texttt{col}, \texttt{attr})$
  - `col` is a collection that was used for generating the vertices of the initial graph,
  - `attr` is an attribute corresponding to an integer or to a domain variable of the collection `col`.

Let $\mathcal{V}$ be the set of vertices of G_f that were generated from the items of the collection `col`.

- If $\mathcal{V}$ is not empty, $\mathbf{PROD}(\mathtt{col},\mathtt{attr})$ corresponds to the product of the values of attribute `attr` associated with the vertices of $\mathcal{V}$,
- Otherwise, if $\mathcal{V}$ is empty, $\mathbf{PROD}(\mathtt{col},\mathtt{attr})$ is equal to 1.

EXAMPLE: The constraint PRODUCT_CTR(VARIABLES, CTR, VAR) forces the product of the variables of the VARIABLES collection to be equal, less than or equal, ... to a given domain variable VAR.
To each variable of VARIABLES corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the *SELF* arc generator together with the TRUE arc constraint. Finally, $\mathbf{PROD}(\mathtt{VARIABLES},\mathtt{var})$ CTR VAR expresses the required condition. In this expression var and CTR respectively corresponds to the attribute of the collection VARIABLES (a domain variable) and to the condition we want to enforce. Since the final graph G_f contains all the vertices of the initial graph, the expression $\mathbf{PROD}(\mathtt{VARIABLES},\mathtt{var})$ corresponds to the product of the variables of the VARIABLES collection.

- $\mathbf{RANGE}(\mathtt{col},\mathtt{attr})$
  - col is a collection that was used for generating the vertices of the initial graph,
  - attr is an attribute corresponding to an integer or to a domain variable of the collection col.

Let $\mathcal{V}$ be the set of vertices of G_f that were generated from the items of the collection col.

- If $\mathcal{V}$ is not empty, $\mathbf{RANGE}(\mathtt{col},\mathtt{attr})$ corresponds to the difference between the maximum and the minimum values of attribute attr associated with the vertices of $\mathcal{V}$,
- Otherwise, if $\mathcal{V}$ is empty, $\mathbf{RANGE}(\mathtt{col},\mathtt{attr})$ is equal to 0.

EXAMPLE: The constraint RANGE_CTR(VARIABLES, CTR, VAR) forces the difference between the maximum value and the minimum value of the variables of the VARIABLES collection to be equal, less than or equal, ... to a given domain variable VAR.
To each variable of VARIABLES corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the *SELF* arc generator together with the TRUE arc constraint. Finally, $\mathbf{RANGE}(\mathtt{VARIABLES},\mathtt{var})$ CTR VAR expresses the required condition. In this expression var and CTR respectively corresponds to the attribute of the collection VARIABLES (a domain variable) and to the condition we want to enforce. Since the final graph G_f contains all the vertices of the initial graph, the expression $\mathbf{RANGE}(\mathtt{VARIABLES},\mathtt{var})$ corresponds to the difference between the maximum value and the minimum value of the variables of the VARIABLES collection.

- **SUM**(col, attr)
  - col is a collection that was used for generating the vertices of the initial graph,
  - attr is an attribute corresponding to an integer or to a domain variable of the collection col.

  Let $\mathcal{V}$ be the set of vertices of G_f that were generated from the items of the collection col.

  - If $\mathcal{V}$ is not empty, **SUM**(col, attr) corresponds to the sum of the values of attribute attr associated with the vertices of $\mathcal{V}$,
  - Otherwise, if $\mathcal{V}$ is empty, **SUM**(col, attr) is equal to 0.

> **EXAMPLE:** The constraint SUM_CTR(VARIABLES, CTR, VAR) forces the sum of the variables of the VARIABLES collection to be equal, less than or equal, ... to a given domain variable VAR.
> To each variable of VARIABLES corresponds a vertex of the initial graph. Since we want to keep all the vertices of the initial graph we use the *SELF* arc generator together with the TRUE arc constraint. Finally, **SUM**(VARIABLES, var) CTR VAR expresses the required condition. In this expression var and CTR respectively correspond to the attribute of the collection VARIABLES (a domain variable) and to the condition we want to enforce. Since the final graph G_f contains all the vertices of the initial graph, the expression **SUM**(VARIABLES, var) corresponds to the sum of the variables of the VARIABLES collection.

- **SUM_WEIGHT_ARC**(Expr) Expr is an *arithmetic expression*. For each arc a of $E(G_f)$, let $f(a)$ denotes the value of Expr. **SUM_WEIGHT_ARC**(Expr) is equal to $\sum_{a \in E(G_f)} f(a)$. The value of Expr usually depends on the attributes of the items located at the extremities of an arc.

> **EXAMPLE:** The constraint GLOBAL_CARDINALITY_WITH_COSTS(VARIABLES, VALUES, MATRIX, COST) forces that each value VALUES$[i]$.val be assigned to exactly VALUES$[i]$.noccurrence variables of the VARIABLES collection. In addition the COST of an assignment is equal to the sum of the elementary costs associated with the fact that we assign the i^{th} variable of the VARIABLES collection to the j^{th} value of the VALUES collection. These elementary costs are given by the MATRIX collection.
> The graph-property SUM_WEIGHT_ARC(MATRIX[(variables.key-1)$*$size(VALUES)$+$ values.key].c) $=$ COST expresses that the COST variable is equal to the sum of the elementary costs associated with each variable-value assignment. All these elementary costs are recorded in the MATRIX collection. More precisely, the cost c_{ij} is recorded in the attribute c of the $((i-1) * |\text{VALUES}| + j)^{th}$ entry of the MATRIX collection.

A last graph parameter, **DISTANCE** , is computed on two final graphs G_1 and G_2 that have the same set V of vertices and the sets $E(G_1)$ and $E(G_2)$ of arcs. This graph parameter is the cardinality of the set $(E(G_1) - E(G_2)) \cup (E(G_2) - E(G_1))$. This corresponds to the number of arcs that belong to $E(G_1)$ but not to $E(G_2)$, plus the number of arcs that are in $E(G_2)$ but not in $E(G_1)$.

Graph class For a given global constraint, a graph class specifies a general property that holds on its final digraph. We list the different graph classes and, for each of them, we point to some global constraints that fit in that class. Finding all the global constraints corresponding to a given graph class can be done by looking into the list of keywords (see Section 3.7 on page 161).

- `ACYCLIC` : the final graph does not have any circuit.

- `BIPARTITE` : the final graph is bipartite.

- `CONSECUTIVE_LOOPS_ARE_CONNECTED` : denotes that the graph constraint of a global constraint uses only the *PATH* and the *LOOP* arc generators and that the final graph does not contain consecutive vertices that have a loop and that are not connected together by an arc.

- `EQUIVALENCE` : the final graph is reflexive, symmetric and transitive.

- `NO_LOOP` : the final graph does not have any loop.

- `ONE_SUCC` : the vertices of the initial graph belong to the final graph and all vertices of the final graph have exactly one successor.

- `SYMMETRIC` : the final graph is symmetric. A digraph is *symmetric* if and only if, if there is an arc from a vertex u to a vertex v, there is also an arc from v to u.

2.3.3 Graph constraint

A global constraint can be defined as a conjunction of several *simple* or *dynamic graph constraints*[21] that all share the same name, the same arguments and the same argument restrictions.[22] This section first describes *simple graph constraints* and then *dynamic graph constraints*, which are an extension of *simple graph constraints*.

Simple graph constraint

To a *simple graph constraint* correspond several initial graphs, usually one, where all the initial graphs have the same vertices and arcs. Specifying more than one initial graph is usually[23] achieved by using the `FOR ALL ITEMS OF` iterator (e.g., see the definition of the GLOBAL_CARDINALITY constraint), which takes a collection C and generates an initial graph $G_i(t)$ for each item t of C. In this context, the arc constraints and/or graph properties of an initial graph may depend of the attributes of the item t of C from which they were generated. All arc constraints attached to a given arc[24] have to be pairwise mutually incompatible.[25]

The graphs of a *simple graph constraint* are defined by the following slots:

- An **Arc input(s)** slot, which consists of:
  - Either a sequence of collections $C_1, C_2, \ldots, C_d$ $(d \geq 1)$. To each item of these collections corresponds a vertex of the initial graph (i.e., in this context we generate a single initial graph).
  - Either a list of sequences of collections. To each item of the collections of a given sequence corresponds a vertex of one of the initial graphs (i.e., in this context we generate one initial graph for each sequence [26]).
- An **Arc generator** slot, which can be one or several expressions[27] of the following forms:
  - $ARC\_GENERATOR \mapsto \texttt{collection}(\texttt{item}_1, \texttt{item}_2, \ldots, \texttt{item}_a)$, where $ARC\_GENERATOR$ is one of the arc generators with a fixed arity[28] defined in Section 2.3.2 on page 61, and $\texttt{item}_i$ $(1 \leq i \leq a)$ denotes the i^{th} item associated with the i^{th} vertex of an arc. These items correspond to formal parameters[29] which can be used within an arc constraint.

[21] For an example of a global constraint that is defined by more than one graph constraint see, for instance, the SORT constraint and its two graph constraints.

[22] The arguments and the argument restrictions were described in Section 2.2.4 on page 26.

[23] Another way of generating several initial graphs will be explained later on in the **Arc input(s)** slot.

[24] As we previously said, even though we have more than one initial graph, all vertices and arcs of the different initial graphs are identical.

[25] Two arc constraints $\texttt{constraint}_1(X_1, X_2, \ldots, X_n)$ and $\texttt{constraint}_2(X_1, X_2, \ldots, X_n)$ are *incompatible* if there does not exist any tuple of values $\langle v_1, v_2, \ldots, v_n \rangle$ such that both $\texttt{constraint}_1(X_1, X_2, \ldots, X_n)$ and $\texttt{constraint}_2(X_1, X_2, \ldots, X_n)$ hold.

[26] This is the case, for example, for the DISTANCE_BETWEEN constraint.

[27] Usually a single expression.

[28] Any arc generator different from $PATH\_1$ and $PATH\_N$.

[29] See the description of *simple arithmetic expressions* page 57.

When the **Arc input(s)** slot consists of a single collection $(d = 1)$, $\texttt{item}_i$ $(1 \leq i \leq a)$ represents an item of the collection C_1. Otherwise, when $d > 1$, we must have $a = d$ and, in this context, $\texttt{item}_i$ $(1 \leq i \leq a)$ represents an item of C_i.

> **EXAMPLE:** The ALLDIFFERENT(`VARIABLES`) constraint has the following **Arc input(s)** and **Arc generator** slots:
>
> * Its **Arc input(s)** slot refers only to the collection `VARIABLES` (i.e., $d = 1$).
> * Its **Arc generator** slot consists of $CLIQUE \mapsto$ `collection` (`variables1`, `variables2`) (i.e., $a = 2$).
>
> In this context, where $d = 1$, both `variables1` and `variables1` are items of the `VARIABLES` collection.

> **EXAMPLE:** The SAME(`VARIABLES1`, `VARIABLES2`) constraint has the following **Arc input(s)** and **Arc generator** slots:
>
> * Its **Arc input(s)** slot refers to the collections `VARIABLES1` and `VARIABLES2` (i.e., $d = 2$).
> * Its **Arc generator** slot consists of $PRODUCT \mapsto$ `collection`(`variables1`, `variables2`) (i.e., $a = 2$).
>
> In this context, where $d > 1$, `variables1` and `variables1` respectively correspond to items of the `VARIABLES1` and the `VARIABLES2` collections.

– $ARC\_GENERATOR \mapsto$ `collection`, where $ARC\_GENERATOR$ is one of the arc generators $PATH\_1$ or $PATH\_N$. In this context, `collection` denotes a collection of items corresponding to the vertices of an arc of the initial graph. An arc constraint forces a restriction on the items of this collection.

> **EXAMPLE:**
> The SIZE_MAX_SEQ_ALLDIFFERENT (`SIZE`, `VARIABLES`) constraint has the following **Arc input(s)** and **Arc generator** slots:
>
> * Its **Arc input(s)** slot refers to the `VARIABLES` collection.
> * Its **Arc generator** slot consists of $PRODUCT \mapsto$ `collection`.
>
> In this context, `collection` is a collection of the same type as the `VARIABLES` collection. It corresponds to the variables associated with an arc of the initial graph.

When the **Arc generator** slot consists of n $(n > 1)$ expressions then these expressions have the form:

$$ARC\_GENERATOR_1 \mapsto \texttt{collection}(\texttt{item}_1, \texttt{item}_2, \ldots, \texttt{item}_a)$$
$$ARC\_GENERATOR_2 \mapsto \texttt{collection}(\texttt{item}_1, \texttt{item}_2, \ldots, \texttt{item}_a)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$ARC\_GENERATOR_n \mapsto \texttt{collection}(\texttt{item}_1, \texttt{item}_2, \ldots, \texttt{item}_a)$$

All leftmost part of the expressions must be the same since they will be involved in a single **Arc constraint(s)** slot. The GLOBAL_CONTIGUITY constraint is an example of global constraint where more than one arc generator is used.

- An **Arc arity** slot, which corresponds to the number of vertices a of each arc of the initial graph. a is either a strictly positive integer, an argument of the global constraint of type `int`, or the character *. In this last case, this is used for denoting that all the arc constraints do not involve the same number of vertices. This is the case, for example, when we use the arc generators $PATH\_1$ or $PATH\_N$ as in the ARITH_SLIDING or the SIZE_MAX_SEQ_ALLDIFFERENT constraints.

- An **Arc constraint(s)** slot, which corresponds to a conjunction of *arc constraints*[30] those were introduced in Section 2.3.2 on page 57.

- A **Graph property(ies)** slot, which corresponds to one or several *graph properties* (see Section 2.3.2 on page 68) to be satisfied on the final graphs associated with an instantiated solution to the global constraint. To each initial graph corresponds one final graph obtained by removing all arcs for which the corresponding arc constraints do not hold as well as all vertices that do not have any arc.

We now give several examples of descriptions of *simple graph constraints*, starting from the NVALUE constraint, which was introduced as a first example of global constraint that can be modelled by a graph property in Section 2.3.1 on page 48.

EXAMPLE: The constraint NVALUE(`NVAL`, `VARIABLES`) restricts `NVAL` to be the number of distinct values taken by the variables of the collection `VARIABLES`. Its meaning is described by a *simple graph constraint* corresponding to the following items:

Arc input(s)	:	`VARIABLES`
Arc generator	:	$CLIQUE \mapsto$ `collection(variables1, variables2)`
Arc arity	:	2
Arc constraint(s)	:	`variables1.var` $=$ `variables2.var`
Graph property(ies):		$\mathbf{NSCC} =$ `NVAL`

Since this description does not use the `FOR ALL ITEMS OF` iterator we generate a single initial graph. Each vertex of this graph corresponds to one item of the `VARIABLES` collection. Since we use the $CLIQUE$ arc generator we have an arc between each pair of vertices. An arc constraint corresponds to an equality constraint between the two variables that are associated with the extremities of the arc. Finally, the **Graph property(ies)** slot forces the final graph to have `NVAL` strongly connected components.

[30] Usually this conjunction consists of a single *arc constraint*.

EXAMPLE: The constraint GLOBAL_CONTIGUITY(VARIABLES) forces all variables of the VARIABLES collection to be assigned to 0 or 1. In addition, all variables assigned to value 1 appear contiguously. Its meaning is described by a *simple graph constraint* corresponding to the following items:

Arc input(s) : VARIABLES

Arc generator : $PATH \mapsto$ collection(variables1, variables2)

$LOOP \mapsto$ collection(variables1, variables2)

Arc arity : 2

Arc constraint(s) : variables1.var = variables2.var

variables1.var = 1

Graph property(ies): $\mathbf{NCC} \leq 1$

Since this description does not use the FOR ALL ITEMS OF iterator we generate a single initial graph. Each vertex of this graph corresponds to one item of the VARIABLES collection. Since we use the $PATH$ arc generator we generate an arc from item VARIABLES$[i]$ to item VARIABLES$[i+1]$ $(1 \leq i < |$VARIABLES$|)$. In addition, since we use the $LOOP$ arc generator, we generate also an arc from each item of the VARIABLES collection to itself.[a] The effect of the arc constraint is to keep in the final graph those vertices for which the corresponding variable is assigned to 1. Adjacent variables assigned to 1 form a connected component of the final graph and the graph property $\mathbf{NCC} \leq 1$ forces to have at most one such group of adjacent variables assigned to 1.

[a] We use the $LOOP$ arc generator in order to keep in the final graph those isolated variables assigned to 1. This is because isolated vertices with no arcs are always removed from the final graph.

EXAMPLE:
The GLOBAL_CARDINALITY(VARIABLES, VALUES) constraint forces that each value VALUES$[i]$.val $(1 \leq i \leq |$VALUES$|)$ be taken by exactly VALUES$[i]$.noccurrence variables of the VARIABLES collection. Its meaning is described by a *simple graph constraint* corresponding to the following items:

For all items of VALUES:

Arc input(s) : VARIABLES

Arc generator : $SELF \mapsto$ collection(variables)

Arc arity : 1

Arc constraint(s) : variables.var = VALUES.val

Graph property(ies): $\mathbf{NVERTEX}$ = VALUES.noccurrence

Since this description uses the **For all items of** VALUES iterator on the VALUES collection we generate an initial graph for each item of the VALUES collection (i.e., one graph for each value). Each vertex of an initial graph corresponds to one item of the VARIABLES collection. Since we use the $SELF$ arc generator we have an arc for each vertex. For an initial graph associated with a value val an arc constraint on a vertex v corresponds to an equality constraint between the variable associated with v and the value val. Finally, the **Graph property(ies)** slot forces the final graph to have a given number of vertices (i.e., associated with the attribute val).

Dynamic graph constraint

The purpose of a *dynamic graph constraint* is to enforce a condition on different subsets of variables, not known in advance. This situation occurs frequently in practice and is hard to express since one cannot use a classical constraint for which it is required to provide all variables right from the beginning. One good example of such global constraint is the CUMULATIVE constraint where one wants to force the sum of some variables to be less than or equal to a given limit. In the context of the CUMULATIVE constraint, each set of variables is defined by the height of the different tasks that overlap a given instant i. Since the origins of the tasks are not initially fixed, we do not know in advance which task will overlap a given instant and so, we cannot state any sum constraint initially.

A *dynamic graph constraint* is defined in exactly the same way as a *simple graph constraint*, except that we may omit the **Graph property(ies)** slot, and that we have to provide the two following additional slots:

- The **Set** slot denotes a generator of sets of vertices. Such a generator takes as argument a final graph and produces different sets of vertices. In order to have something tractable, we force the total number of generated sets to be polynomial in the number of vertices.

  In practice each set of vertices is represented by a collection of items. The type of this collection corresponds either to the type of the items associated with the vertices, or to the type of a new derived collection. This is achieved by providing an expression of the form `name` or `name-derived_collection`, where `name` represents a formal parameter, and `derived_collection` a declaration of a new derived collection (as specified in Section 2.3.2 on page 51).

- The **Constraint(s) on sets** slot provides a global constraint defined in the catalogue that has to hold for each set created by the previous generator.

We now describe the different generators of sets of vertices currently available:

- ALL_VERTICES generates a single set containing all the vertices of the final graph. It is specified by a declaration of the form

  ALL_VERTICES>> [`vertices`]

  where `vertices` represents all the vertices of the final graph.

- CC generates one set of vertices for each connected component of the final graph. These sets correspond to all the vertices of a given connected component. It is specified by a declaration of the form

  CC>> [`connected_component`]

  where `connected_component` represents the vertices of a connected component of the final graph.

- PATH_LENGTH(L) generates all elementary paths[31] of L vertices of the final graph such that, discarding loops, all vertices of a path (except the last one) have

[31] A path where all vertices are distinct is called an *elementary path*.

no more than one successor in the final graph. It is specified by a declaration of the form

$\mathsf{PATH\_LENGTH}(L)>>$ [`path`]

where `path` represents the vertices of an elementary path, ordered according to their occurrences in the path.

- PRED generates the non-empty sets corresponding to the predecessors of each vertex of the final graph. It is specified by a declaration of the form

  PRED$>>$ [`predecessor`, `destination`]

  where `destination` represents a vertex of the final graph and `predecessor` its predecessors.

- SUCC generates the non-empty sets corresponding to the successors of each vertex of the final graph. It is specified by a declaration of the form

  SUCC$>>$ [`source`, `successor`]

  where `source` represents a vertex of the final graph and `successor` its successors.

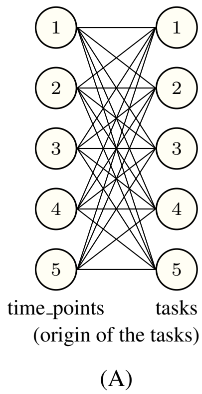
As an illustrative example of *dynamic graph constraint* we now consider the CUMULATIVE constraint.

EXAMPLE: The CUMULATIVE(TASKS, LIMIT) constraint, where TASKS is a collection of the form collection(origin − dvar, duration − dvar, end − dvar, height − dvar), and where LIMIT is a non-negative integer, holds if, for any point the cumulated height of the set of tasks that overlap that point, does not exceed LIMIT.
The first graph constraint of CUMULATIVE forces for each task of the TASKS collection the equality origin + duration = end. We focus on the second graph constraint, which uses a *dynamic graph constraint* described by the following items:

Arc input(s)	:	TASKS TASKS
Arc generator	:	$PRODUCT$ $\mapsto$ collection(tasks1, tasks2)
Arc arity	:	2
Arc constraint(s)	:	tasks1.duration > 0
		tasks2.origin ≤ tasks1.origin
		tasks1.origin ≤ tasks2.end
Sets	:	SUCC>> [source, variables − col(VARIABLES − collection(var − dvar), [item(var − TASKS.height)])]
Constraint(s) on sets:		SUM_CTR(variables, ≤, LIMIT)

The second graph constraint is defined by:

- To each item of the TASKS collection correspond two vertices of the initial graph.
- The arity of the arc constraint is 2.
- The arcs of the initial graph are constructed with the $PRODUCT$ arc generator between the TASKS collection and the TASKS collection. Therefore, each vertex associated with a task is linked to all the vertices related to the different tasks.
- The arc constraint that is associated with an arc between a task tasks1 and a task tasks2 is an overlapping constraint that holds if both, the duration of tasks1 is strictly greater than zero, and if the origin of tasks1 is overlapped by task tasks2.
- The set generator is SUCC. The final graph will consist of those tasks for which the origin is covered by at least one task and of those corresponding tasks.
- The dynamic constraint on a set forces the sum of the heights of the tasks that belong to a successor set to not exceed LIMIT.



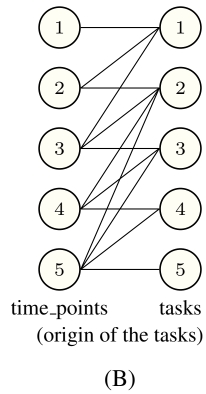


Figure 2.10: Initial and final graph of an instance of the CUMULATIVE constraint

Parts (A) and (B) of Figure 2.10 respectively show the initial and the final graph corresponding to the following instance:

CUMULATIVE(⟨origin − 1 duration − 3 height − 1,
origin − 2 duration − 9 height − 2,
origin − 3 duration − 10 height − 1,
origin − 6 duration − 6 height − 1,
origin − 7 duration − 2 height − 3⟩, 8).

We label the vertices of the initial and final graph by giving the key[a] of the corresponding task. On both graphs the edges are oriented from left to right. On the final graph we consider the sets that consist of the successors of the different vertices; those are the sets of tasks $\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$, $\{2, 3, 4\}$ and $\{2, 3, 4, 5\}$. Since the SUCC set generator uses a derived collection that only considers the height attribute of a task, these sets respectively correspond to the following collection of items:

- ⟨var − 1⟩,
- ⟨var − 1, var − 2⟩,
- ⟨var − 1, var − 2, var − 1⟩,
- ⟨var − 2, var − 1, var − 1⟩,
- ⟨var − 2, var − 1, var − 1, var − 3⟩.

The CUMULATIVE constraint holds since, for each successors set, the corresponding constraint holds:

- SUM_CTR(⟨var − 1⟩, $\leq$, 8),
- SUM_CTR(⟨var − 1, var − 2⟩, $\leq$, 8),
- SUM_CTR(⟨var − 1, var − 2, var − 1⟩, $\leq$, 8),
- SUM_CTR(⟨var − 2, var − 1, var − 1⟩, $\leq$, 8),
- SUM_CTR(⟨var − 2, var − 1, var − 1, var − 3⟩, $\leq$, 8).

The SUM_CTR(VARIABLES, CTR, VAR) constraint holds if the sum S of the variables of the VARIABLES collection satisfies S CTR VARIABLES, where CTR is a comparison operator.

[a] key is an implicit attribute corresponding to the position of an item within a collection that was introduced in Section 2.2.2 on page 16.

2.4 Describing global constraints in terms of automata

This section is based on the article describing global constraint in terms of automata [39]. The main difference from the original article is the introduction of array of counters within the description of an automaton. We consider global constraints for which any ground instance can be checked in linear time by scanning once through their variables without using any data structure, except counters or arrays of counters. In order to concretely illustrate this point we first select a set of global constraints and write down a checker for each of them. Finally, we give for each checker a sketch of the corresponding automaton. Based on these observations, we define the type of automaton we use in the catalogue.

2.4.1 Selecting an appropriate description

As we previously said, we focus on those global constraints that can be checked by scanning once through their variables. This is the case, for example, of:

- ELEMENT [429],
- MINIMUM [29],
- PATTERN [92],
- GLOBAL_CONTIGUITY [282],
- LEX_LESSEQ [184],
- AMONG [47],
- INFLEXION [27],
- ALLDIFFERENT [351].

Since they illustrate key points needed for characterising the set of solutions associated with a global constraint, our discussion will be based on the last five constraints for which we now recall the definition:

- The GLOBAL_CONTIGUITY(`vars`) constraint forces the sequence of 0-1 variables `vars` to have at most one group of consecutive 1. For example, the constraint GLOBAL_CONTIGUITY($\langle 0, 1, 1, 0\rangle$) holds since we have only one group of consecutive 1.

- The lexicographic ordering constraint $\vec{x} \leq_{\text{lex}} \vec{y}$ (see LEX_LESSEQ) over two vectors of variables $\vec{x} = \langle x_0, \ldots, x_{n-1}\rangle$ and $\vec{y} = \langle y_0, \ldots, y_{n-1}\rangle$ holds if and only if $n = 0$ or $x_0 < y_0$ or $x_0 = y_0$ and $\langle x_1, \ldots, x_{n-1}\rangle \leq_{\text{lex}} \langle y_1, \ldots, y_{n-1}\rangle$.

- The AMONG(`nvar`, `vars`, `values`) constraint restricts the number of variables of the sequence of variables `vars` that take their values in a given set `values`, to be equal to the variable `nvar`. For example, AMONG($3, \langle 4, 5, 5, 4, 1\rangle, \langle 1, 5, 8\rangle$) holds since exactly 3 values of the sequence 45541 are located in the set of values $\{1, 5, 8\}$.

- The INFLEXION(`ninf`, `vars`) constraint forces the number of inflexions of the sequence of variables `vars` to be equal to the variable `ninf`. An *inflexion* is described by one of the following patterns: a strict increase followed by a strict decrease or, conversely, a strict decrease followed by a strict increase. For example, INFLEXION($4, \langle 3, 3, 1, 4, 5, 5, 6, 5, 5, 6, 3\rangle$) holds since we can extract from the sequence 33145565563 the four subsequences 314, 565, 6556 and 563, which all follow one of these two patterns.

- The ALLDIFFERENT($\texttt{vars}$) constraint forces all pairs of distinct variables of the collection $\texttt{vars}$ to take distinct values. For example, ALLDIFFERENT($\langle 6, 1, 5, 9\rangle$) holds since we have four distinct values.

Parts (A1), (B1), (C1), (D1) and (E1) of Figure 2.11 depict the five checkers respectively associated with GLOBAL_CONTIGUITY, with LEX_LESSEQ, with AMONG, with INFLEXION and with ALLDIFFERENT. Within the corresponding automata an initial state is indicated by an arc coming from no state and an accepting state is denoted graphically by a double circle. For each checker we note the following facts:

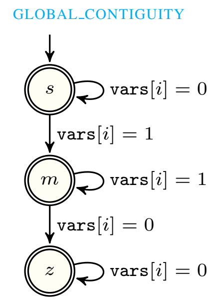
- Within the checker depicted by part (A1) of Figure 2.11, the values of the sequence $\texttt{vars}[0], \dots, \texttt{vars}[n-1]$ are successively compared against 0 and 1 in order to check that we have at most one group of consecutive 1. This can be translated to the automaton depicted by part (A2) of Figure 2.11. The automaton takes as input the sequence $\texttt{vars}[0], \dots, \texttt{vars}[n-1]$, and triggers successively a transition for each term of this sequence. Transitions labelled by 0 and 1 are respectively associated with the conditions $\texttt{vars}[i] = 0$ and $\texttt{vars}[i] = 1$.Transitions leading to failure are systematically skipped. This is why no transition labelled with a 1 starts from state z.

- Within the checker given by part (B1) of Figure 2.11, the components of vectors $\overrightarrow{x}$ and $\overrightarrow{y}$ are scanned in parallel. We first skip all the components that are equal and then perform a final check. This is represented by the automaton depicted by part (B2) of Figure 2.11. The automaton takes as input the sequence $\langle x[0], y[0]\rangle, \dots, \langle x[n-1], y[n-1]\rangle$ and triggers a transition for each term of this sequence. Unlike the GLOBAL_CONTIGUITY constraint, some transitions now correspond to a condition (e.g., $x[i] = y[i]$, $x[i] < y[i]$) between two variables of the LEX_LESSEQ constraint.

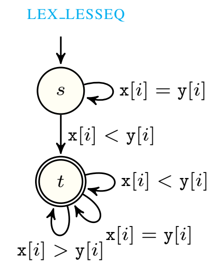
- Note that the AMONG($\texttt{nvar}, \texttt{vars}, \texttt{values}$) constraint involves a variable $\texttt{nvar}$ whose value is computed from a given collection of variables $\texttt{vars}$. The checker depicted by part (C1) of Figure 2.11 counts the number of variables of $\texttt{vars}[0], \dots, \texttt{vars}[n-1]$ that take their values in $\texttt{values}$. For this purpose it uses a counter c, which is possibly tested against the value of $\texttt{nvar}$. This convinced us to allow the use of counters in an automaton. Each counter has an initial value, which can be updated while triggering certain transitions. The accepting states of an automaton can force a variable of the constraint to be equal to a given counter. Part (C2) of Figure 2.11 describes the automaton corresponding to the code given in part (C1) of the same figure. The automaton uses the counter variable c initially set to 0 and takes as input the sequence $\texttt{vars}[0], \dots, \texttt{vars}[n-1]$. It triggers a transition for each variable of this sequence and increments c when the corresponding variable takes its value in $\texttt{values}$. The accepting state returns a success when the value of c is equal to $\texttt{nvar}$. At this point we want to stress the following fact: it would have been

```
GLOBAL_CONTIGUITY(vars[0..n − 1]):boolean
begin
  i ← 0
  while i < n ∧ vars[i] = 0 do i++
  while i < n ∧ vars[i] = 1 do i++
  while i < n ∧ vars[i] = 0 do i++
  return (i = n)
end
```

(A1)



(A2)



(B2)

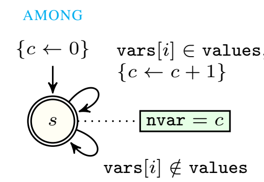
```
LEX_LESSEQ(x[0..n − 1], y[0..n − 1]):boolean
begin
  i ← 0
  while i < n ∧ x[i] = y[i] do i++
  return (i = n ∨ x[i] < y[i])
end
```

(B1)

```
AMONG(nvar, vars[0..n − 1], values):boolean
begin
  i, c ← 0
  while i < n do
    if vars[i] ∈ values then c++
    i++
  return (nvar = c)
end
```

(C1)

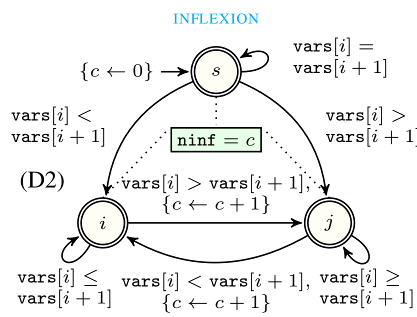
(C2)




```
INFLEXION(ninf, vars[0..n − 1]):boolean
begin
  i, c ← 0
  while i < n − 1 ∧ vars[i] = vars[i + 1] do i++
  if i < n − 1 then less ← (vars[i] < vars[i + 1])
  while i < n − 1 do
    if less then
      if vars[i] > vars[i + 1] then {c++; less ← false;}
    else
      if vars[i] < vars[i + 1] then {c++; less ← true;}
    i++
  return (ninf = c)
end
```

(D1)

(D2)




```
ALLDIFFERENT(vars[0..n − 1]):boolean
begin
  u, v ← vars[0]; i ← 1;
  while i < n do
    if vars[i] < u then u ← vars[i]
    else if vars[i] > v then v ← vars[i]
    i++
  for i ← u to v do c[i] ← 0
  for i ← 0 to n − 1 do
    c[vars[i]] ← c[vars[i]] + 1
  for i ← u to v do
    if c[i] > 1 then return false
  return true
end
```

(E1)

(E2)

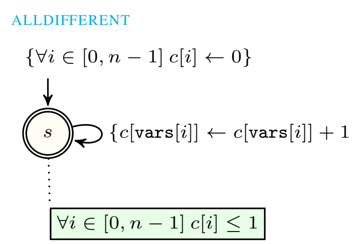


Figure 2.11: Five checkers and their corresponding automata

possible to use an automaton that avoids the use of counters. However, this automaton would depend on the effective value of the argument `nvar`. In addition, it would require more states than the automaton of part (C2) of Figure 2.11. This is typically a problem if we want to have a fixed number of states in order to save memory as well as time.

- As the AMONG constraint, the INFLEXION($\mathtt{ninf}, \mathtt{vars}$) constraint involves a variable ninf whose value is computed from a given sequence of variables $\mathtt{vars}[0], \ldots, \mathtt{vars}[n-1]$. Therefore, the checker depicted in part (D1) of Figure 2.11 uses also a counter c for counting the number of inflexions, and compares its final value to the ninf argument. The automaton depicted by part (D2) of Figure 2.11 represents this program. It takes as input the sequence of pairs $\langle \mathtt{vars}[0], \mathtt{vars}[1]\rangle$, $\langle \mathtt{vars}[1], \mathtt{vars}[2]\rangle$, ..., $\langle \mathtt{vars}[n-2], \mathtt{vars}[n-1]\rangle$ and triggers a transition for each pair. Note that a given variable may occur in more than one pair. Each transition compares the respective values of two consecutive variables of $\mathtt{vars}[0..n-1]$ and increments the counter c when a new inflexion is detected. The accepting state returns a success when the value of c is equal to ninf.

- The checker associated with ALLDIFFERENT is depicted by part (E1) of Figure 2.11. It first initialises an array of counters to 0. The entries of the array correspond to the potential values of the sequence $\mathtt{vars}[0], \ldots, \mathtt{vars}[n-1]$. In a second phase the checker computes for each potential value its number of occurrences in the sequence $\mathtt{vars}[0], \ldots, \mathtt{vars}[n-1]$. This is done by scanning this sequence. Finally in a third phase the checker verifies that no value is used more than once. These three phases are represented by the automaton depicted by part (E2) of Figure 2.11. The automaton depicted by part (E2) takes as input the sequence $\mathtt{vars}[0], \ldots, \mathtt{vars}[n-1]$. Its initial state initialises an array of counters to 0. Then it triggers successively a transition for each element $\mathtt{vars}[i]$ of the input sequence and increments by 1 the entry corresponding to $\mathtt{vars}[i]$. The accepting state checks that all entries of the array of counters are strictly less than 2, which means that no value occurs more than once in the sequence $\mathtt{vars}[0], \ldots, \mathtt{vars}[n-1]$.

Synthesising all the observations we got from these examples leads to the following remarks and definitions for a given global constraint $\mathcal{C}$:

- For a given state, no transition can be triggered indicates that the constraint $\mathcal{C}$ does not hold.

- Since all transitions starting from a given state are mutually incompatible all automata are deterministic. Let $\mathcal{M}$ denotes the set of mutually incompatible conditions associated with the different transitions of an automaton.

- Let $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ denotes the sequence of subsets of variables of $\mathcal{C}$ on which the transitions are successively triggered. All these subsets contain the same number of elements and refer to some variables of $\mathcal{C}$. Since these subsets typically depend on the constraint, we leave the computation of $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ outside the automaton. To each subset $\mathcal{S}_i$ of this sequence corresponds a variable S_i with an initial domain ranging over $[min, min + |\mathcal{M}| - 1]$, where min is a fixed integer. To each integer of this range corresponds one of the mutually incompatible conditions of $\mathcal{M}$. The sequences $S_0, \ldots, S_{m-1}$ and $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ are respectively called the *signature* and the *signature argument* of the constraint. The constraint between S_i and the variables of $\mathcal{S}_i$ is called the *signature constraint* and is denoted by $\Psi_{\mathcal{C}}(S_i, \mathcal{S}_i)$.

- From a pragmatic point the view, the task of writing a constraint checker is naturally done by writing down an imperative program where local variables, arrays, assignment statements and control structures are used. This suggested us to consider deterministic finite automata augmented with local variables and assignment statements on these variables. Regarding control structures, we did not introduce any extra feature since the deterministic choice of which transition to trigger next seemed to be good enough.
- Many global constraints involve a variable whose value is computed from a given collection of variables. This convinced us to allow the accepting state of an automaton to optionally return a result. In practice, this result corresponds to the value of a local variable of the automaton in the accepting state.

2.4.2 Defining an automaton

An automaton $\mathcal{A}$ of a global constraint $\mathcal{C}$ is defined by

$$\langle \mathcal{S}ignature, \mathcal{S}ignature\mathcal{D}omain, \mathcal{S}ignature\mathcal{A}rg, \mathcal{S}ignature\mathcal{A}rg\mathcal{P}attern, \\ \mathcal{C}ounters, \mathcal{A}rrays, \mathcal{S}tates, \mathcal{T}ransitions\rangle$$

where:

- $\mathcal{S}ignature$ is the sequence of variables $S_0, \ldots, S_{m-1}$ corresponding to the signature of the constraint $\mathcal{C}$.
- $\mathcal{S}ignature\mathcal{D}omain$ is an interval that defines the range of possible values of the variables of $\mathcal{S}ignature$.
- $\mathcal{S}ignature\mathcal{A}rg$ is the signature argument $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ of the constraint $\mathcal{C}$. The link between the variables of $\mathcal{S}_i$ and the variable S_i $(0 \leq i < m)$ is done by writing down the signature constraint $\Psi_{\mathcal{C}}(S_i, \mathcal{S}_i)$.
- When used, $\mathcal{S}ignature\mathcal{A}rg\mathcal{P}attern$ defines a symbolic name for each term of $\mathcal{S}ignature\mathcal{A}rg$. These names can be used within the description of a transition for expressing an additional condition for triggering the corresponding transition.
- $\mathcal{C}ounters$ is the, possibly empty, list of all counters used in the automaton $\mathcal{A}$. Each counter is described by a term $\mathtt{t}(Counter, InitialValue, FinalVariable)$ where $Counter$ is a symbolic name representing the counter, $InitialValue$ is an integer giving the value of the counter in the initial state of $\mathcal{A}$, and $FinalVariable$ gives the variable that should be unified with the value of the counter in the accepting state of $\mathcal{A}$.
- $\mathcal{A}rrays$ is the, possibly empty, list of all arrays used in the automaton $\mathcal{A}$. Each array is described by a term $\mathtt{t}(Array, InitialValue, FinalConstraint)$

where $Array$ is a symbolic name representing the array, $InitialValue$ is an integer giving the value of all the entries of the array in the initial state of $\mathcal{A}$. $FinalConstraint$ denotes an existing constraint of the catalogue that should hold in the accepting state of $\mathcal{A}$. Arguments of this constraint correspond to collections of variables that are bound to array of counters, or to variables that are bound to counters declared in $\mathcal{C}ounters$. For an array of counters we only consider those entries that are located between the first and the last entries that were modified while triggering a transition of $\mathcal{A}$.

- $\mathcal{S}tates$ is the list of states of $\mathcal{A}$, where each state has the form $\mathrm{source}(id)$, $\mathrm{sink}(id)$ or $\mathrm{node}(id)$. id is a unique identifier associated with each state. Finally, $\mathrm{source}(id)$ and $\mathrm{sink}(id)$ respectively denote the initial and the accepting state of $\mathcal{A}$. An automaton has a single initial state and at least one accepting state. The initial and accepting states may coincide.

- $\mathcal{T}ransitions$ is the list of transitions of $\mathcal{A}$. Each transition t has the form $\mathrm{arc}(id_1,$ $label,$ $id_2)$ or $\mathrm{arc}(id_1,$ $label,$ $id_2,$ $counters)$. id_1 and id_2 respectively correspond to the state just before and just after t, while $label$ denotes the value that the signature variable should have in order to trigger t. When used, $counters$ gives for each counter of $\mathcal{C}ounters$ its value after firing the corresponding transition. This value is specified by an arithmetic expression involving counters, constants, as well as usual arithmetic functions, such as $+$, $-$, $\min$, or $\max$. The order used in the $counters$ list is identical to the order used in $\mathcal{C}ounters$.

EXAMPLE: As an illustrative example we give the description of the automaton associated with the INFLEXION$(ninf, vars)$ constraint. We have:

- $\mathcal{S}ignature = S_0, S_1, \ldots, S_{n-2}$,
- $\mathcal{S}ignature\mathcal{D}omain = 0..2$,
- $\mathcal{S}ignature\mathcal{A}rg = \langle vars[0], vars[1]\rangle, \ldots, \langle vars[n-2], vars[n-1]\rangle$,
- $\mathcal{S}ignature\mathcal{A}rg\mathcal{P}attern$ is not used,
- $\mathcal{C}ounters = \mathrm{t}(c, 0, ninf)$,
- $\mathcal{S}tates = [source(s), sink(s), sink(i), sink(j)]$,
- $\mathcal{T}ransitions = [\mathrm{arc}(s,1,s),\ \mathrm{arc}(s,2,i),\ \mathrm{arc}(s,0,j),\ \mathrm{arc}(i,1,i),\ \mathrm{arc}(i,2,i),$ $\mathrm{arc}(i,0,j,[c+1]), \mathrm{arc}(j,1,j), \mathrm{arc}(j,0,j), \mathrm{arc}(j,2,i,[c+1])]$.

The signature constraint relating each pair of variables $\langle vars[i], vars[i+1]\rangle$ to the signature variable S_i is defined as follows: $\Psi_{\texttt{inflexion}}(S_i, vars[i], vars[i+1]) \equiv vars[i] > vars[i+1] \Leftrightarrow S_i = 0 \ \wedge vars[i] = vars[i+1] \Leftrightarrow S_i = 1 \ \wedge vars[i] < vars[i+1] \Leftrightarrow S_i = 2$. The sequence of transitions triggered on the ground instance INFLEXION$(4,\ [3,3,1,4,5,5,6,5,5,6,3])$ is $\frac{s}{c=0} \xrightarrow{3=3\Leftrightarrow S_0=1} s \xrightarrow{3>1\Leftrightarrow S_1=0} j \xrightarrow[c=1]{1<4\Leftrightarrow S_2=2} i \xrightarrow{4<5\Leftrightarrow S_3=2} i \xrightarrow{5=5\Leftrightarrow S_4=1} i \xrightarrow{5<6\Leftrightarrow S_5=2} i \xrightarrow[c=2]{6>5\Leftrightarrow S_6=0} j \xrightarrow{5=5\Leftrightarrow S_7=1} j \xrightarrow[c=3]{5<6\Leftrightarrow S_8=2} i \xrightarrow[c=4]{6>3\Leftrightarrow S_9=0} j$. Each transition gives the corresponding condition and, possibly, the value of the counter c just after firing that transition. After the last encountered state j the first argument $ninf$ of the INFLEXION constraint is fixed to the current value of the counter c, i.e. $ninf = 4$.

2.5 Reformulating global constraints as a conjunction

Many global constraints can be reformulated as a conjunction of global or reified constraints. The slot **Reformulation** provides for some global constraints such reformulations (see, for example, the reformulation slots respectively associated with the COLOURED_CUMULATIVE or the TREE constraints). When it exists, the corresponding code is available in the ".pl file" attached to a constraint. The initial concrete motivation for providing reformulations was triggered by the fact that it is usually an easy way to have a first implementation of a constraint, which is a feature we want to have in the context of the catalogue. However many reformulations (e.g., ALLDIFFERENT, NVALUE, TREE) involve a quadratic (or even more) number of variables and/or constraints, which does not scale in practice when one wants to handle constraints with thousands of variables. This is why many filtering algorithms compute again and again common quantities that would require too much memory if stored explicitly.

2.6 Semantic links between global constraints

For each global constraint entry of the catalogue, the slot **See also** provides links to other global constraints. Rather than just pointing to a set of constraints, we prefer to explicitly indicate the reason why we point to a given constraint. A link $link(C_{entry}, C_{also})$ from a constraint C_{entry} (i.e., the constraint associated with a catalogue entry) to another constraint C_{also} (i.e., the constraint of the **See also** slot located in the catalogue entry of constraint C_{entry}) has a given semantics and this section describes the kind of *semantic links* that are currently used. Before introducing each semantic link and its meaning, let us first quote that some of them are related by one of the following relations:

- A link $link$ is *symmetric* if and only if $link(C_1, C_2) \Leftrightarrow link(C_2, C_1)$.
- A link $link$ is *asymmetric* if and only if $link(C_1, C_2) \Rightarrow \neg link(C_2, C_1)$ ($\neg link(C_2, C_1)$ is a shortcut for denoting that the link $link(C_2, C_1)$ does not occur in the catalogue).
- A link $link_j$ is the *converse* of a link $link_i$ if and only if $link_i(C_1, C_2) \Leftrightarrow link_j(C_2, C_1)$.

Table 2.1 lists each semantic link and the relation it has.[32] Then one section describes the meaning of each semantic link.

2.6.1 Assignment dimension added

Constraint C_{also} corresponds to constraint C_{entry} where an assignment dimension is added to C_{entry}.

[32] All links are automatically checked with respect to their relations each time the catalogue is generated.

semantic links	relation between semantic links
assignment dimension added	converse: *assignment dimension removed*
assignment dimension removed	converse: *assignment dimension added*
attached to cost variant	converse: *cost variant*
common keyword	symmetric
comparison swapped	symmetric
cost variant	converse: *attached to cost variant*
generalisation	converse: *specialisation*
hard version	converse: *soft variant*
implied by	converse: *implies*
implies	converse: *implied by*
implies (if swap arguments)	symmetric
implies (items to collection)	asymmetric
negation	symmetric
part of system of constraints	converse: *system of constraints*
related	symmetric
related to a common problem	symmetric
root concept	converse: *shift of concept*
shift of concept	converse: *root concept*
soft variant	converse: *hard version*
specialisation	converse: *generalisation*
system of constraints	converse: *part of system of constraints*
used in graph description	asymmetric
used in reformulation	converse: *uses in its reformulation*
uses in its reformulation	converse: *used in reformulation*

Table 2.1: Available semantic links between constraints

EXAMPLE: As an example, constraint C_{also} = CUMULATIVES corresponds to constraint C_{entry} = CUMULATIVE where an assignment dimension corresponding to the `machine` attribute is added (i.e., the constraint CUMULATIVES enforces a CUMULATIVE constraint for each maximum set of tasks that are assigned the same machine).

2.6.2 Assignment dimension removed

Constraint C_{also} corresponds to constraint C_{entry} where an assignment dimension is removed from C_{entry}.

EXAMPLE: As an example, constraint C_{also} = AMONG_LOW_UP corresponds to constraint C_{entry} = INTERVAL_AND_COUNT where an assignment dimension corresponding to the `origin` attribute is removed from C_{entry} = INTERVAL_AND_COUNT (i.e., the constraint INTERVAL_AND_COUNT enforces a AMONG_LOW_UP constraint for each maximum set of tasks for which the origin is assigned the same interval $[k \cdot \mathtt{SIZE\_INTERVAL}, k \cdot \mathtt{SIZE\_INTERVAL} + \mathtt{SIZE\_INTERVAL} - 1]$) (`SIZE_INTERVAL` is the last argument of INTERVAL_AND_COUNT).

2.6.3 Attached to cost variant

Constraint C_{also} is the original version attached to the cost variant constraint C_{entry}.

> **EXAMPLE:** As an example, constraint C_{also} = ALLDIFFERENT is the original version attached to the cost variant constraint C_{entry} = MINIMUM_WEIGHT_ALLDIFFERENT, where the total cost of a solution is the sum of the costs associated with the fact that we assign a given value to a specific variable.

2.6.4 Common keyword

Constraints C_{entry} and C_{also} share one or more common keywords with a strong semantic connotation.

> **EXAMPLE:** As an example, constraints C_{entry} = TREE and C_{also} = CYCLE are both graph partitioning constraints (i.e., constraints that partition the vertices of a given initial digraph so that each partition corresponds to a specific pattern, a *tree* and a *circuit* in this example).

2.6.5 Comparison swapped

Constraint C_{also} corresponds to constraint C_{entry} where one of the following conditions holds:

- The comparison operator $\geq$ is swapped to $\leq$ or, conversely, $\leq$ is swapped to $\geq$.
- The comparison operator $>$ is swapped to $<$ or, conversely, $<$ is swapped to $>$.

> **EXAMPLE:** Constraint C_{also} = ATMOST corresponds to constraint C_{entry} = ATLEAST where the comparison $\leq$ `N` for expressing that we should not exceed a given threshold (i.e., restricts the maximum number of occurrences for a given value) is replaced by $\geq$ `N` for expressing that we should reach a given threshold (i.e., forces a minimum number of occurrences for a given value).

2.6.6 Cost variant

Constraint C_{also} is a cost variant of constraint C_{entry}.

> **EXAMPLE:** As an example, constraint C_{also} = SUM_OF_WEIGHTS_OF_DISTINCT_VALUES is the cost variant of constraint C_{entry} = NVALUE, where we introduce a weight for each value and we replace the number of distinct values by the sum of weights associated with distinct values.

2.6.7 Generalisation

Denotes that constraint C_{also} is a generalisation of constraint C_{entry}.

> **EXAMPLE:** As an example, constraint C_{also} = ALL_MIN_DIST is a generalisation of constraint C_{entry} = ALLDIFFERENT where we replace a disequality between two variables by the fact that two line segments of same length do not overlap.

2.6.8 Hard version

Constraint C_{also} is a hard version of constraint C_{entry} (i.e., constraint C_{entry} is a soft variant of constraint C_{also}).

> **EXAMPLE:** As an example, constraint C_{also} = ALLDIFFERENT is a hard version of constraint C_{entry} = SOFT_ALLDIFFERENT, which restricts the minimum number of variables that should be assigned differently in order that all variables take a distinct value.

2.6.9 Implied by

If constraint C_{also} holds and if all restrictions of constraint C_{entry} hold then constraint C_{entry} also holds. Note that we try to restrict ourselves to the transitive reduction of the implication graph between constraints.

> **EXAMPLE:** As an example, constraint C_{entry} = MINIMUM is implied by constraint C_{also} = AND.

2.6.10 Implies

If constraint C_{entry} holds and if all restrictions of constraint C_{also} hold then constraint C_{also} also holds. Note that we also consider all the implications depicted in the implication graphs mentioned in the tables associated with the normalised signature tree of global constraints arguments. For an example of such table see Table 3.1.

> **EXAMPLE:** As an example, constraint C_{entry} = ALLDIFFERENT implies constraint C_{also} = NOT_ALL_EQUAL. Note that the case of an ALLDIFFERENT constraint with a single variable does not imply a NOT_ALL_EQUAL constraint since its restriction (i.e., the number of variables of a NOT_ALL_EQUAL constraint should be strictly greater than one) does not hold.

2.6.11 Implies (if swap arguments)

Given two constraints C_{entry} and C_{also} that both have two arguments, if constraint $C_{entry}(\texttt{arg}_1, \texttt{arg}_2)$ holds then constraint $C_{also}(\texttt{arg}_2, \texttt{arg}_1)$ also holds.

> **EXAMPLE:** As an example, we can go from constraint C_{entry} = LEX_LESSEQ to constraint LEX_GREATEREQ if we swap the two arguments of constraint LEX_LESSEQ.

2.6.12 Implies (items to collection)

Given two constraints C_{entry} and C_{also} where:

- C_{entry} has a single argument $\texttt{arg}_1$ corresponding to a collection of k items, each attribute of type `int` or `dvar`.
- C_{also} has a single argument $\texttt{arg}_2$ corresponding to a collection of collections of `dvar`, each of them having the same number of items k.

If constraint $C_{entry}(\texttt{arg}_1)$ holds then constraint $C_{also}(\texttt{arg}_2)$ also holds.

EXAMPLE: As an example, we can go from constraint C_{entry} = CIRCUIT to constraint LEX_ALLDIFFERENT if we create for each item "$\texttt{index} - i$ $\texttt{succ} - s$" of the CIRCUIT constraint a collection $\langle \texttt{var} - i, \texttt{var} - s \rangle$.

2.6.13 Negation

If constraint C_{entry} holds then constraint C_{also} does not hold. Reciprocally, if constraint C_{also} holds then constraint C_{entry} does not hold. Note that constraints C_{entry} and C_{also} must also have exactly the same parameters, but not necessarily the same parameters restrictions.

EXAMPLE: As an example, the constraint C_{also} = NOT_ALL_EQUAL (i.e., prevent all variables to be assigned the same value) is the negation of constraint C_{entry} = ALL_EQUAL (i.e., enforce all variables to be assigned the same value).

Note that negation is also directly available for constraints which are defined by:

- A single counter free automaton, see keyword automaton without counters.
- A single automaton with counter, see keyword automaton with counters.
- A set of functional dependencies, see keyword pure functional dependency.

2.6.14 Part of system of constraints

Denotes that a constraint C_{entry} is a conjunction of constraints C_{also} (i.e., see the keyword system of constraints).

EXAMPLE: As an example, the constraint C_{also} = NEQ (i.e., prevent two variables to be assigned the same value) can be used to reformulate the constraint C_{entry} = ALLDIFFERENT (i.e., enforce a set of variables to take distinct values) as a conjunction of NEQ constraints.

2.6.15 Related

Denotes that a constraint C_{entry} and a constraint C_{also} are related by a specific reason that is not covered by an existing link.

EXAMPLE: As an example, the constraint C_{also} = TREE_RANGE (i.e., given a digraph, partition it so that each vertex belongs to one tree for which the difference between the longest and the shortest paths – from a leaf to the root – is restricted) is related to the constraint C_{entry} = BALANCE (i.e., given a set of variables, restrict the difference between the number of occurrence of the value that occurs the most and the value that occurs the least) by the fact that, on the one hand the constraint TREE_RANGE can express a *balanced tree*, on the other side the constraint BALANCE can express a *balanced assignment*.

2.6.16 Related to a common problem

Denotes that a constraint C_{entry} and a constraint C_{also} are related to a same problem (i.e., they can both be used for modelling that problem).

EXAMPLE: As an example, the constraints C_{entry} = COLORED_MATRIX and C_{also} = SAME can both be used for modelling the *matrix reconstruction problem*.

2.6.17 Root concept

Constraint C_{entry} is derived from constraint C_{also}.

EXAMPLE: As an example, the constraint C_{entry} = TREE_RESOURCE is derived from the constraint C_{also} = TREE. Given a digraph G, the TREE constraint forces a partitioning of G by a set of trees in such a way that each vertex of G belongs to one distinct tree. In addition, the TREE_RESOURCE constraint distinguishes *resource* and *task* vertices, and forces each tree to contain exactly one resource vertex.

2.6.18 Shift of concept

Constraint C_{also} is derived from constraint C_{entry}.

EXAMPLE: As an example, constraint C_{also} = GLOBAL_CARDINALITY_NO_LOOP(`NLOOP`, `VARIABLES`, `VALUES`) is derived from constraint C_{entry} = GLOBAL_CARDINALITY(`VARIABLES`, `VALUES`) (i.e., each value `VALUES`$[i]$`.val` should be taken by exactly `VALUES`$[i]$`.val` variables of the `VARIABLES` collection) by discarding all variables such that `VARIABLES`$[i]$`.var` $= i$.

2.6.19 Soft variant

Constraint C_{also} is a soft variant of constraint C_{entry}. Note that, from an academic point of view, a soft constraint C_{also} = is usually defined with a cost variable that quantifies how much the constraint C_{entry} = is violated. We exceptionally breaks this rule when it seems to make sense from an application point of view. For example, within the ALLDIFFERENT constraint, we refer to the ALLDIFFERENT_EXCEPT_0 since it can be seen as a kind of relaxation of the ALLDIFFERENT constraint where we allow to use value 0 several times.

EXAMPLE: As an example, one of the possible soft variants of constraint C_{entry} = ALLDIFFERENT (i.e., the ALLDIFFERENT constraint forces all variables of a collection to take distinct values) is the constraint C_{also} = SOFT_ALLDIFFERENT_VAR, where the cost is the minimum number of variables that need to be assigned differently to satisfy the ALLDIFFERENT constraint.

2.6.20 Specialisation

Denotes that constraint C_{also} is a specialisation of constraint C_{entry}.

EXAMPLE: As an example, constraint C_{also} = PATH is a specialisation of constraint C_{entry} = TREE. Given a digraph G, the TREE constraint forces a covering of G by a set of trees in such a way that each vertex of G belongs to one distinct tree. If, in addition, we restrict each vertex to have at most one child we get the PATH constraint.

2.6.21 System of constraints

Denotes that a constraint C_{also} is a conjunction of constraints C_{entry} (see the keyword system of constraints).

EXAMPLE: As an example, the constraint C_{also} = COLORED_MATRIX corresponds to a conjunction of constraints of the form C_{entry} = GLOBAL_CARDINALITY: Given a matrix $\mathcal{M}$ of variables, the COLORED_MATRIX constraint forces a GLOBAL_CARDINALITY on each row and each column of $\mathcal{M}$.

2.6.22 Used in graph description

Constraint C_{also} is used within a graph based description of constraint C_{entry}.

EXAMPLE: As an example, the constraint C_{also} = TWO_ORTH_DO_NOT_OVERLAP, a constraint enforcing two orthotopes[a] to not overlap, is used in the graph based description of the constraint C_{entry} = DIFFN. Given a collection of orthotopes, the DIFFN constraint forces for each pair of orthotopes (O_1, O_2) that O_1 and O_2 do not overlap.

[a] An *orthotope* corresponds to the generalisation of a segment, a rectangle and a box to the n-dimensional case.

2.6.23 Used in reformulation

Constraint C_{also} is used within a reformulation of constraint C_{entry}. Since it is already handled by the link part of system of constraints, we do not consider the case where constraint C_{entry} can be expressed as a conjunction of constraints C_{also}.

EXAMPLE: As an example, the constraint C_{also} = OPEN_MINIMUM is used within the reformulation slot of the constraint C_{entry} = TREE_RANGE.

2.6.24 Uses in its reformulation

Constraint C_{also} uses constraint C_{entry} in its reformulation. Since it is already handled by the link system of constraints, we do not consider the case where constraint C_{also} can be expressed as a conjunction of constraints C_{entry}.

EXAMPLE: As an example, the reformulation slot of constraint C_{also} = TREE_RANGE uses the constraint C_{entry} = OPEN_MINIMUM.

3

Description of the Catalogue

Contents

3.1 Which global constraints are included?

The global constraints of this catalogue come from the following sources:

- Existing constraint systems like:
  - ALICE [267],

- CHARME in C [309],
- **CHIP** [150] in Prolog, C and C++ (`http://www.cosytec.com`),
- **Choco** [256] in Java (`http://choco.emn.fr/`),
- ECLAIR [416] in Claire,
- **ECLiPSe** [121, 11] in Prolog (`http://eclipseclp.org/`),
- **FaCile** in OCaml (`http://www.recherche.enac.fr/opti/facile/`),
- **Gecode** in C++ [385] (`http://www.gecode.org/`),
- **IF/PROLOG** in Prolog (`http://www.ifcomputer.com/IFProlog/Constraints/home_en.html`),
- **Ilog Solver** [334] in C++ and later in Java (`http://www.ilog.com`),
- **JaCoP** in Java (`http://www.jacop.eu/`),
- Koalog in Java,
- **Minion** [199] (`http://minion.sourceforge.net/index.html`),
- **Mozart** [405, 130] in Oz (`http://www.mozart-oz.org/`),
- **SICStus** [109] in Prolog (`http://www.sics.se/sicstus/`).

When available, the **Systems** slot of a global constraint entry of the catalogue provides the name of the corresponding global constraint in the context of the **Choco**, **Gecode**, **JaCoP**, **MiniZinc**, and **SICStus** systems.

- Constraint programming articles mostly from conferences like:
  - The Principles and Practice of Constraint Programming (CP) (`http://www.informatik.uni-trier.de/~ley/db/conf/cp/index.html`),
  - The International Joint Conference on Artificial Intelligence (IJCAI) (`http://www.informatik.uni-trier.de/~ley/db/conf/ijcai/index.html`),
  - The National Conference on Artificial Intelligence (AAAI) (`http://www.informatik.uni-trier.de/~ley/db/conf/aaai/index.html`),
  - The International Conference on Logic Programming (ICLP) (`http://www.informatik.uni-trier.de/~ley/db/conf/iclp/index.html`),
  - The International Conference of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR) (`http://www.informatik.uni-trier.de/~ley/db/conf/cpaior/`).
- Graph constraints from the CP(Graph) computation domain [151].
- New constraints inspired by variations of existing constraints, practical applications, combinatorial problems, puzzles or discussions with colleagues.

3.2 Which global constraints are missing?

Constraints with too many arguments like, for example, the original CYCLE [133] constraint with 16 arguments, which are in fact a combination of several constraints, were not directly put into the catalogue. Constraints that have complex arguments were also omitted. Beside this, the following constraints should be added in some future version of the catalogue: ALLDIFFERENT_ON_MULTISETS [346] [347], CASE [108, 103], [122, 123], CHOQUET [231], COST_REGULAR [145], CUMULATIVE_TRAPEZE [331, 59], DEVIATION [382, 380], INEQUALITY_SUM [365, 366], MINIMUM_SPANNING_TREE [152, 360], NO_CYCLE [113], RANGE [69, 71], REGULAR [317] [134], SOFT_GCC_VAL [435, 436, 457, 383], SOFT_GCC_VAR [435, 436, 457], SOFT_REGULAR [435], SPREAD [318, 381], MULTICOST_REGULAR [293], PREF_ALLDIFFERENT_VAR (i.e., variable-based relaxation of ALLDIFFERENT with preferences) [295, page 100], PREF_ALLDIFFERENT_CTR (i.e., decomposition-based relaxation of ALLDIFFERENT with preferences) [295, page 103], PREF_GLOBAL_CARDINALITY_LOW_UP_VAR (i.e., variable-based relaxation of GLOBAL_CARDINALITY_LOW_UP with preferences) [295, page 123], PREF_GLOBAL_CARDINALITY_LOW_UP_CTR (i.e., decomposition-based relaxation of GLOBAL_CARDINALITY_LOW_UP with preferences) [295, page 126]. Finally we only consider a restricted number of constraints involving set variables since this is a relatively new area, which is currently growing rapidly since 2003.

3.3 Searching in the catalogue

3.3.1 How to see if a global constraint is in the catalogue?

Searching a given global constraint through the catalogue can be achieved in the following ways:

- If you have an idea of the name of the global constraint you are looking for, then put all its letters in lower case, separate distinct words by an underscore and search the resulting name in the index. Within the pdf document, the entry of the catalogue where the constraint is defined is shown in **bold**. Common abbreviations, synonyms and usual names found in articles have also been put in the index in ***bold and italic***.

- If you do not know the name of the global constraint you are looking for, but you know the types of its arguments then Section 3.5 lists the different argument patterns and the corresponding global constraints.

- You can also search a global constraint through the list of keywords that is attached to each global constraint. All available keywords are listed alphabetically in Section 3.7 on page 161. For each keyword we give the list of global constraints using the corresponding keyword as well as the definition of the keyword.

- In order to make it possible to search for all keywords related to a specific area, we have also attached to each keyword one, or exceptionally two, meta-keywords. For example, if you are searching for global constraints that are mentioning puzzles, you first look to the meta-keyword *Puzzles* where you find the keywords corresponding to puzzles (i.e., *Autoref*, *Conway packing problem*, ..., *Sudoku*, *Zebra puzzle*). Then as previously described, for each keyword you can access to the corresponding global constraints. All available meta-keywords are listed alphabetically in Section 3.6 on page 150. For each meta-keyword it first gives the list of keywords using the corresponding meta-keyword and then defines the meta-keyword.

3.3.2 How to search for all global constraints sharing the same structure

Since we have three ways of defining global constraints (e.g., searching for a graph with specific properties, coming up with an automaton that only recognises the solutions associated with the global constraint or using a first order logic formula) we can look to the global constraints from these three perspectives.

Searching from a graph property perspective

The index contains all the arc generators as well as all the graph properties and the pages where they are mentioned.[1] This allows finding all global constraints that use a given arc generator or a given graph property in their definitions. You can further restrict your search to those global constraints using a specific combination of arc generators and graph properties. All these combinations are listed at the "signature" entry of the index. Within these combinations, a graph property with an underline means that the constraint should be evaluated each time the minimum of this graph property increases. Similarly a graph property with an overline indicates that the constraint should be evaluated each time the maximum of this graph property decreases. For example, if we look for those constraints that both use the $CLIQUE$ arc generator as well as the $\mathbf{NARC}$ graph-property we find the INVERSE and PLACE_IN_PYRAMID constraints. Since $\mathbf{NARC}$ is underlined and overlined these constraints will have to be woken each time the minimum or the maximum of $\mathbf{NARC}$ changes. The signature associated with a global constraint is also shown in the header of the even pages corresponding to the description of the global constraint.

Searching from an automaton perspective

We have created the following list of keywords, which allow finding all global constraints defined by a specific type of automaton that recognises its solutions[2]:

- Automaton indicates that the catalogue provides a deterministic automaton,

[1] Arc generators and graph properties are introduced in the section "Describing Explicitly Global Constraints".

[2] Automata that recognise the solutions to a global constraint were introduced in the section "Describing Explicitly Global Constraints".

- Automaton without counters indicates that the catalogue provides a deterministic automaton without counters as well as without array of counters,
- Automaton with counters indicates that the catalogue provides a deterministic automaton with counters but without array of counters,
- Automaton with array of counters indicates that the catalogue provides a deterministic automaton with array of counters and possibly with counters.

In addition, we also provide a list of keywords that characterise the structure of the hypergraph associated with the decomposition of the automaton of a global constraints (i.e., see the meta-keyword constraint network structure). Note that, when a global constraint is defined by several graph properties it is also defined by several automata (usually one automata for each graph property). This is the case, for example, of the CHANGE_CONTINUITY constraint. Currently we have these keywords:

- Berge-acyclic constraint network,
- Alpha-acyclic constraint network(2),
- Alpha-acyclic constraint network(3),
- Sliding cyclic(1) constraint network(1),
- Sliding cyclic(1) constraint network(2),
- Sliding cyclic(1) constraint network(3),
- Sliding cyclic(2) constraint network(2),
- Circular sliding cyclic(1) constraint network(2),
- Centered cyclic(1) constraint network(1),
- Centered cyclic(2) constraint network(1),
- Centered cyclic(3) constraint network(1),

When a global constraint is only defined by one or several automaton its signature is set to the keyword `AUTOMATON`.

Searching from a first order logic perspective

The keyword logic provides the list of constraints that are described within the catalogue in term of a first order logic formula where predicates are replaced by arithmetic constraints.

3.3.3 Searching all places where a global constraint is referenced

Beside the page where a global constraint is defined (in bold), the index also gives all the pages where a global constraint is referenced.

Last, since a global constraint can also be used for defining another global constraint the slot **Used in** of the description of a global constraint provides this information.

3.3.4 Searching the mapping with a constraint of a concrete system

Two distinct ways are provided for making the correspondence between a constraint of the catalogue and a constraint of a concrete existing system:

1. Appendix C provides, when it exists, the direct correspondence[3] between the constraints of the catalogue and the constraints of a given concrete system. For the time being we have considered, with the help of their respective authors, the following systems:

   - **Choco** in Java [256] (`http://choco.emn.fr/`),
   - **Gecode** in C++ [385] (`http://www.gecode.org/`),
   - **JaCoP** in Java (`http://www.jacop.eu/`),
   - **MiniZinc** (`http://www.minizinc.org/`),
   - **SICStus** [109] in Prolog (`http://www.sics.se/sicstus/`).

   Since not all constraints of a given system always have their counterparts in the current version of the catalogue, and since systems are always enriched, this is the reason why this mapping is not complete.

2. Within the entry of the catalogue the slot **Systems** provides the correspondence between the constraint associated with that entry and the name of the constraint in a given concrete system or modelling language. For example, the **Systems** slot of the entry of the catalogue corresponding to the ELEMENT constraint indicates that ELEMENT is called NTH in **Choco** and ELEMENT in **Gecode**, **JaCoP** **MiniZinc** and **SICStus**.

3.4 Figures of the catalogue

The catalogue contains the following types of figures:

- Figures that give the normalised signature tree of the arguments of a global constraint These figures are located in Section 3.5.

- Figures that provide the implication graph between global constraints that have the same normalised signature tree for their arguments (e.g., see the figure embedded in the lower part of Table 3.1).

- Figures that illustrate a global constraint or a keyword (e.g., see Figure 3.38 that illustrates the keyword limited discrepancy search).

- Figures that depict the initial as well as the final graphs associated with a global constraint (e.g., see Figure 5.159 that provides the initial and final graphs of the CHANGE constraint).

[3] We do not consider that a given constraint of the catalogue can be reformulated in terms of a conjunction of constraints of a given concrete system.

- Figures that provide an automaton that only recognises the solutions associated with a given global constraint (e.g., see Figure 5.387 that gives the automaton of the GLOBAL_CONTIGUITY constraint).

- Figures that give the hypergraph associated with the decomposition of an automaton in terms of signature and transition constraints (e.g., see Figure 5.388 that gives the hypergraph of the automaton-based reformulation of the GLOBAL_CONTIGUITY constraint).

- Figures for the graph structure of the XML schema of the parameters of a global constraint. They are only available in the on-line version of the catalogue.

- Figures for visualising different views (i.e., compulsory part and cumulative profile) of two-dimensional placement of constraints. These figures are only available in the on-line version of the catalogue. They are accessible from the table containing the squared squares problem instances.

Most of the graph figures that depict the initial and final graph of a global constraint of this catalogue as well as the graph structure of the `XML` schema of the parameters of a global constraint were automatically generated by using the open source graph drawing software `Graphviz` [193] available from AT&T.[4] Since late 2012 Ti*k*Z [415] is used for generating all new figures and for converting the old `Xfig`, `PSTricks` [447] and `Graphviz` figures so that all figures are done with Ti*k*Z.

[4] http://www.research.att.com/sw/tools/graphviz

3.5 Constraints argument patterns

If you do not know the name of the global constraint you are looking for, but you know the types of its arguments this section allows to find out all global constraints which have similar arguments. For this purpose we associate to each global constraint of the catalogue a unique normalised signature tree derived from the types of its arguments.[5] The purpose of this normalised signature tree is to get a concise normal form of the arguments of a global constraint that does not depend of the order in which these arguments are defined.

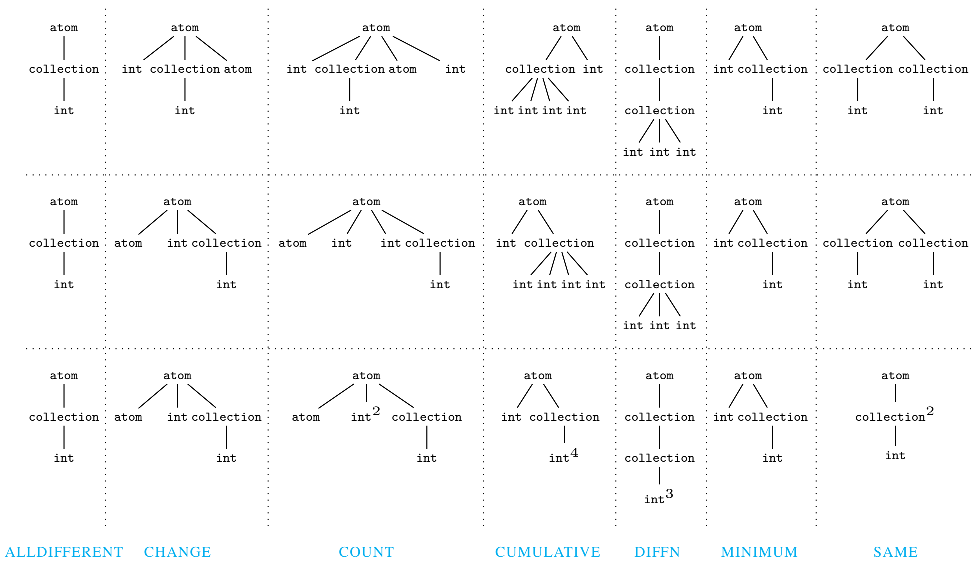


Figure 3.1: Illustrating steps (2), (3) and (4) for computing the normalised signature tree

The normalisation takes as input the slots **Type(s)** and **Argument(s)** of the description of a global constraint[6] and computes the normalised signature tree in four steps:

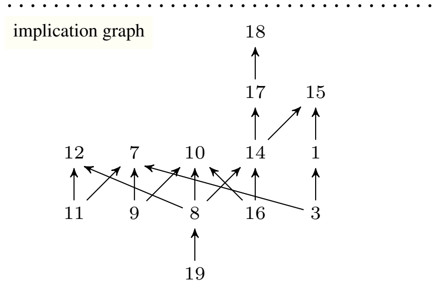
1. The first step converts all types related to variables to their corresponding ground counterparts: the types `dvar`, `svar`, `mvar` and `rvar` are respectively transformed to `int`, `sint`, `mint` and `real`.
2. The second step builds a tree of types $\mathcal{T}$ by exploring the slot **Argument(s)** and by developing the compound data types possibly used. The root of this tree is the type `atom` and represents the name of the global constraint.
3. The third step normalises the tree of types $\mathcal{T}$ by first normalising each subtree of $\mathcal{T}$ and then by sorting the children of $\mathcal{T}$. We assume the following ordering

[5] An informal rule used in the catalogue about the order of the arguments of a constraint is that we usually first mention a domain variable which represents a result computed from one or several collections that occur just after. Finally, eventual parameters are put as the last arguments of the constraint.

[6] See Section 2.2.4 for the description of these slots.

on the different types: `atom` $\prec$ `int` $\prec$ `sint` $\prec$ `mint` $\prec$ `real` $\prec$ `list` $\prec$ `collection`. Let $\mathcal{T}_n$ denotes the normalised tree obtained at this third step.

4. Finally the last step tries to reduce the size of the normalised tree $\mathcal{T}_n$ by identifying $k(k > 1)$ children of a vertex v of $\mathcal{T}_n$ for which the k subtrees are identical. When such a configuration is identified the k subtrees of v are replaced by a single subtree and the integer k is put as an exponent of v.

<table>
<tr>
<td>atom
col^2
int</td>
<td>
1. ALLDIFFERENT_ON_INTERSECTION
2. CONSECUTIVE_GROUPS_OF_ONES
3. DISJOINT
4. INCOMPARABLE
5. INT_VALUE_PRECEDE_CHAIN
6. INVERSE_WITHIN_RANGE
7. LEX_DIFFERENT
8. LEX_EQUAL
9. LEX_GREATER
10. LEX_GREATEREQ
11. LEX_LESS
12. LEX_LESSEQ
13. LEX_LESSEQ_ALLPERM
14. SAME
15. SAME_INTERSECTION
16. SORT
17. USED_BY
18. USES
19. VEC_EQ_TUPLE


</td>
</tr>
</table>

Table 3.1: Example of information associated with a normalised signature tree (within the signature tree `col` is a shortcut for `collection`)

The three rows of Figure 3.1 illustrate respectively the second, third and fourth steps for computing the normalised signature tree associated with the arguments of the constraints ALLDIFFERENT, CHANGE, COUNT, CUMULATIVE, DIFFN, MINIMUM and SAME.

The next sections provide for each possible constraints arity all existing normalised signature trees together with the corresponding list of global constraints of the catalogue. The leftmost part of an entry corresponds to a normalised signature tree, while the rightmost upper part gives the corresponding list of global constraints. Finally the rightmost lower part describes the dependency between the constraints of the list: there is an edge from a constraint $\mathtt{ctr}_1$ to a constraint $\mathtt{ctr}_2$ if and only if the fact that $\mathtt{ctr}_1$ holds implies that $\mathtt{ctr}_2$ also holds. For example, consider the constraints associated with the normalised signature tree corresponding to two collections of integers depicted by Table 3.1. There is an edge from 16 (i.e., SORT) to 14 (i.e., SAME) since the fact that a SORT constraint holds implies that a SAME constraint also holds.

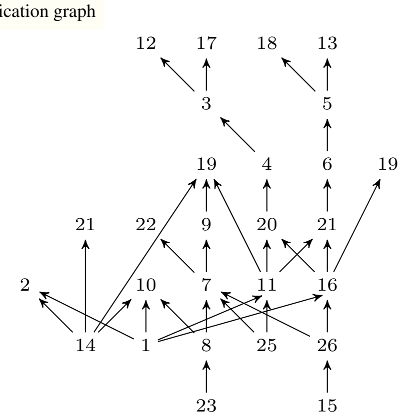
3.5.1 Constraints with 1 argument

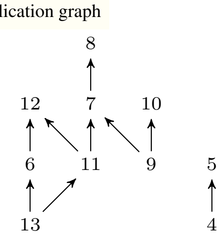
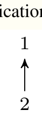
`atom` — `sint`	1. SUM_FREE

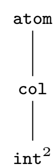
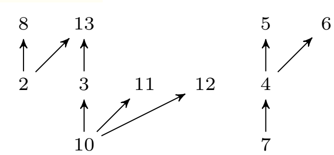
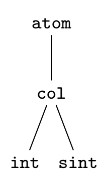
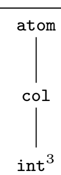
```
atom
 |
col
 |
int
```

1. ALL_EQUAL
2. ALL_EQUAL_EXCEPT_0
3. ALL_EQUAL_PEAK
4. ALL_EQUAL_PEAK_MAX
5. ALL_EQUAL_VALLEY
6. ALL_EQUAL_VALLEY_MIN
7. ALLDIFFERENT
8. ALLDIFFERENT_CONSECUTIVE_VALUES
9. ALLDIFFERENT_EXCEPT_0
10. CONSECUTIVE_VALUES
11. DECREASING
12. DECREASING_PEAK
13. DECREASING_VALLEY
14. GLOBAL_CONTIGUITY
15. GOLOMB
16. INCREASING
17. INCREASING_PEAK
18. INCREASING_VALLEY
19. MULTI_GLOBAL_CONTIGUITY
20. NO_PEAK
21. NO_VALLEY
22. NOT_ALL_EQUAL
23. PERMUTATION
24. SOME_EQUAL
25. STRICTLY_DECREASING
26. STRICTLY_INCREASING

implication graph



atom col sint	1. ALLDIFFERENT_BETWEEN_SETS
atom col col int	1. ALLPERM 2. K_ALLDIFFERENT 3. K_DISJOINT 4. K_SAME 5. K_USED_BY 6. LEX2 7. LEX_ALLDIFFERENT 8. LEX_ALLDIFFERENT_EXCEPT_0 9. LEX_CHAIN_GREATER 10. LEX_CHAIN_GREATEREQ 11. LEX_CHAIN_LESS 12. LEX_CHAIN_LESSEQ 13. STRICT_LEX2 14. ZERO_OR_NOT_ZERO_VECTORS implication graph 
atom col col int^3	1. DIFFN 2. ORTHS_ARE_CONNECTED implication graph 

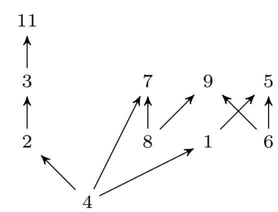
	1. ALLDIFFERENT_CST 2. CIRCUIT 3. DERANGEMENT 4. DISJUNCTIVE 5. DISJUNCTIVE_OR_SAME_END 6. DISJUNCTIVE_OR_SAME_START 7. PRECEDENCE 8. PROPER_CIRCUIT 9. SEQUENCE_FOLDING 10. SYMMETRIC_ALLDIFFERENT 11. SYMMETRIC_ALLDIFFERENT_EXCEPT_0 12. SYMMETRIC_ALLDIFFERENT_LOOP 13. TWIN implication graph 
	1. ATMOST1 2. BIPARTITE 3. CONNECTED 4. DAG 5. STRONGLY_CONNECTED 6. SYMMETRIC 7. TOUR
	1. INVERSE 2. INVERSE_EXCEPT_LOOP 3. ORTH_LINK_ORI_SIZ_END

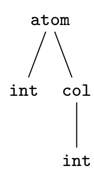
`atom`
|
`col`
/ \\
`int`^{3} `sint`
	1. DISJ

`atom`
|
`col`
/ \\
`int`^{2} `sint`^{2}
	1. STABLE_COMPATIBILITY

`atom`
|
`col`
|
`int`^{5}
	1. POLYOMINO

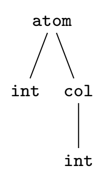
3.5.2 Constraints with 2 arguments

atom | int^2	1. ABS_VALUE 2. DIVISIBLE 3. DIVISIBLE_OR 4. EQ 5. GEQ 6. GT 7. LEQ 8. LT 9. NEQ 10. OPPOSITE_SIGN 11. SAME_SIGN 12. SIGN_OF 13. ZERO_OR_NOT_ZERO implication graph 
atom / \\ int sint	1. IN_SET



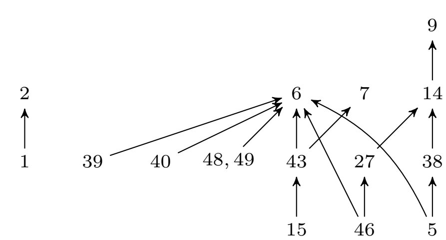
1. ALL_MIN_DIST
2. ALLDIFFERENT_INTERVAL
3. ALLDIFFERENT_MODULO
4. AMONG_DIFF_0
5. AND
6. ATLEAST_NVALUE
7. ATMOST_NVALUE
8. BALANCE
9. BETWEEN_MIN_MAX
10. DEEPEST_VALLEY
11. EQUIVALENT
12. FIRST_VALUE_DIFF_0
13. HIGHEST_PEAK
14. IN
15. INCREASING_NVALUE
16. INCREASING_SUM
17. INFLEXION
18. IMPLY
19. LENGTH_FIRST_SEQUENCE
20. LENGTH_LAST_SEQUENCE
21. LONGEST_DECREASING_SEQUENCE
22. LONGEST_INCREASING_SEQUENCE
23. MAX_DECREASING_SLOPE
24. MAX_INCREASING_SLOPE
25. MAX_NVALUE
26. MAX_SIZE_SET_OF_CONSECUTIVE_VAR
27. MAXIMUM
28. MIN_DECREASING_SLOPE
29. MIN_DIST_BETWEEN_INFLEXION

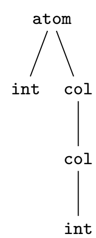
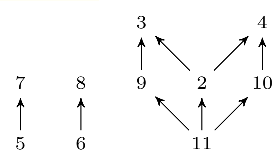
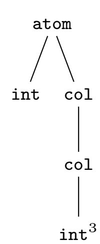
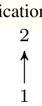
continued ☛

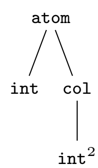


30. MIN_INCREASING_SLOPE
31. MIN_NVALUE
32. MIN_SIZE_FULL_ZERO_STRETCH
33. MIN_SIZE_SET_OF_CONSECUTIVE_VAR
34. MIN_SURF_PEAK
35. MIN_WIDTH_PEAK
36. MIN_WIDTH_PLATEAU
37. MIN_WIDTH_VALLEY
38. MINIMUM
39. NAND
40. NOR
41. NOT_IN
42. NSET_OF_CONSECUTIVE_VALUES
43. NVALUE
44. NVISIBLE_FROM_END
45. NVISIBLE_FROM_START
46. OR
47. PEAK
48. SIZE_MAX_SEQ_ALLDIFFERENT
49. SIZE_MAX_STARTING_SEQ_ALLDIFFERENT
50. SOFT_ALLDIFFERENT_CTR
51. SOFT_ALLDIFFERENT_VAR
52. SOFT_ALL_EQUAL_MAX_VAR
53. SOFT_ALL_EQUAL_MIN_CTR
54. SOFT_ALL_EQUAL_MIN_VAR
55. SUM_OF_INCREMENTS
56. VALLEY
57. XOR

implication graph



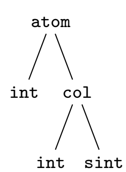
	1. ALL_DIFFER_FROM_AT_LEAST_K_POS 2. NVECTOR 3. ATLEAST_NVECTOR 4. ATMOST_NVECTOR 5. K_SAME_INTERVAL 6. K_SAME_MODULO 7. K_USED_BY_INTERVAL 8. K_USED_BY_MODULO 9. ORDERED_ATLEAST_NVECTOR 10. ORDERED_ATMOST_NVECTOR 11. ORDERED_NVECTOR implication graph 
	1. DIFFN_COLUMN 2. DIFFN_INCLUDE 3. PLACE_IN_PYRAMID implication graph 

<table>
<tr><td>
</td><td>

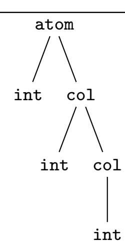
1. BALANCE_CYCLE
2. BALANCE_PATH
3. BALANCE_TREE
4. BIN_PACKING
5. BINARY_TREE
6. CYCLE
7. DOMAIN_CONSTRAINT
8. IN_INTERVALS
9. INCREASING_NVALUE_CHAIN
10. MAX_INDEX
11. MIN_INDEX
12. NPAIR
13. OPEN_MAXIMUM
14. OPEN_MINIMUM
15. ORDERED_GLOBAL_CARDINALITY
16. PATH
17. TREE

implication graph

$16 \rightarrow 5 \rightarrow 17$

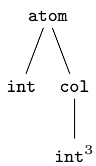
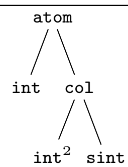
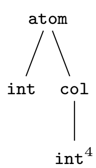
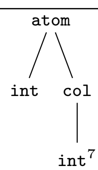
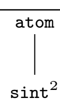
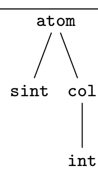
</td></tr>
<tr><td>
</td><td>

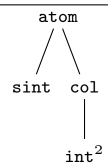
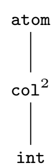
1. CLIQUE
2. DISCREPANCY
3. K_CUT
4. PROPER_FOREST

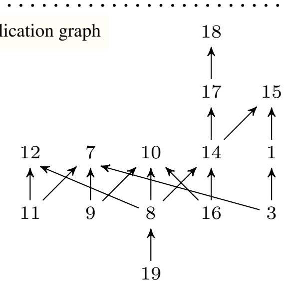
</td></tr>
<tr><td>
</td><td>

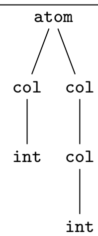
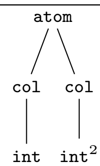
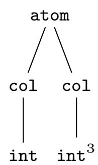
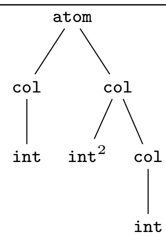
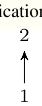
1. CUMULATIVE_CONVEX

</td></tr>
</table>

	1. CIRCUIT_CLUSTER 2. GRAPH_CROSSING 3. ORCHARD 4. ORTH_ON_THE_GROUND 5. TRACK
	1. CUTSET
	1. COLOURED_CUMULATIVE 2. CROSSING 3. CUMULATIVE 4. CUMULATIVE_PRODUCT 5. TEMPORAL_PATH . . . . implication graph $3 \rightarrow 1$
	1. CUMULATIVE_TWO_D
	1. EQ_SET
	1. OPEN_ALLDIFFERENT

<table>
<tr>
<td>
</td>
<td>1. LINK_SET_TO_BOOLEANS</td>
</tr>
<tr>
<td>
</td>
<td>1. ALLDIFFERENT_ON_INTERSECTION
2. CONSECUTIVE_GROUPS_OF_ONES
3. DISJOINT
4. INCOMPARABLE
5. INT_VALUE_PRECEDE_CHAIN
6. INVERSE_WITHIN_RANGE
7. LEX_DIFFERENT
8. LEX_EQUAL
9. LEX_GREATER
10. LEX_GREATEREQ
11. LEX_LESS
12. LEX_LESSEQ
13. LEX_LESSEQ_ALLPERM
14. SAME
15. SAME_INTERSECTION
16. SORT
17. USED_BY
18. USES
19. VEC_EQ_TUPLE

implication graph

</td>
</tr>
</table>

	1. ALLDIFFERENT_PARTITION 2. IN_RELATION 3. ORDER 4. PATTERN
	1. GLOBAL_CARDINALITY
	1. INCREASING_GLOBAL_CARDINALITY 2. GLOBAL_CARDINALITY_LOW_UP 3. STRETCH_CIRCUIT 4. STRETCH_PATH
	1. STRETCH_PATH_PARTITION
	1. K_SAME_PARTITION 2. K_USED_BY_PARTITION . . . . implication graph 

$\begin{array}{c}\texttt{atom}\\ \mid\\ \texttt{col}^2\\ \mid\\ \texttt{int}^2\end{array}$	1. BIN_PACKING_CAPA 2. ELEM 3. ELEMENT_GREATEREQ 4. ELEMENT_LESSEQ 5. ELEMENTS 6. ELEMENTS_ALLDIFFERENT 7. INDEXED_SUM implication graph $\begin{array}{cccc}3 & & 4 & 5\\ & \nwarrow & \uparrow & \uparrow\\ & & 2 & 6\end{array}$
$\begin{array}{cc}\multicolumn{2}{c}{\texttt{atom}}\\ / & \backslash\\ \texttt{col} & \texttt{col}\\ \mid & \mid\\ \texttt{int}^2 & \texttt{int}^3\end{array}$	1. STAGE_ELEMENT 2. TREE_RESOURCE
$\begin{array}{cc}\multicolumn{2}{c}{\texttt{atom}}\\ \multicolumn{2}{c}{\mid}\\ \multicolumn{2}{c}{\texttt{col}^2}\\ / & \backslash\\ \texttt{int} & \texttt{sint}\end{array}$	1. INVERSE_SET
$\begin{array}{cc}\multicolumn{2}{c}{\texttt{atom}}\\ / & \backslash\\ \texttt{col} & \texttt{col}\\ \mid & \mid\\ \texttt{int}^2 & \texttt{int}^5\end{array}$	1. COLOURED_CUMULATIVES 2. CUMULATIVE_WITH_LEVEL_OF_PRIORITY 3. ELEM_FROM_TO
$\begin{array}{c}\texttt{atom}\\ \mid\\ \texttt{col}^2\\ \mid\\ \texttt{int}^3\end{array}$	1. CYCLE_RESOURCE 2. DISJOINT_TASKS 3. TWO_ORTH_ARE_IN_CONTACT 4. TWO_ORTH_DO_NOT_OVERLAP implication graph $\begin{array}{c}4\\ \uparrow\\ 3\end{array}$

atom col^2 int^2 sint	1. SYMMETRIC_GCC
atom col^2 int^3 sint	1. SYMMETRIC_CARDINALITY

3.5.3 Constraints with 3 arguments

<table>
<tr>
<td>

`atom`
|
`int`^{3}

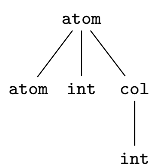
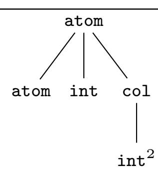
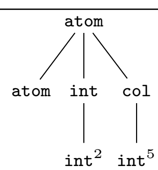
</td>
<td>

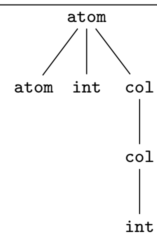
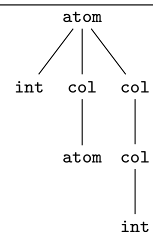
1. DISTANCE
2. EQ_CST
3. GCD
4. GEQ_CST
5. IN_INTERVAL
6. LEQ_CST
7. MULTIPLE
8. NEQ_CST
9. POWER
10. REMAINDER

implication graph

$$\begin{tikzcd} 4 & 6 \\ 2 \arrow[u] \arrow[ur] & 1 \arrow[u] \end{tikzcd}$$

</td>
</tr>
</table>

	1. ARITH 2. ARITH_SLIDING 3. CHANGE 4. CIRCULAR_CHANGE 5. LONGEST_CHANGE 6. NVALUES 7. NVALUES_EXCEPT_0 8. PERIOD 9. PERIOD_EXCEPT_0 10. PRODUCT_CTR 11. RANGE_CTR 12. SUM_CTR 13. SUM_CUBES_CTR 14. SUM_POWERS4_CTR 15. SUM_POWERS5_CTR 16. SUM_POWERS6_CTR 17. SUM_SQUARES_CTR . . . . implication graph 9 ↑ 8
	1. ASSIGN_AND_NVALUES 2. SCALAR_PRODUCT
	1. CUMULATIVES

	1. NVECTORS
	1. CHANGE_VECTORS 2. PERIOD_VECTORS

<table>
<tr>
<td>atom
int^2 col
int</td>
<td>

1. ALL_BALANCE
2. ATLEAST
3. ATMOST
4. BALANCE_INTERVAL
5. BALANCE_MODULO
6. DOMAIN
7. ELEMENT
8. EXACTLY
9. INT_VALUE_PRECEDE
10. ITH_POS_DIFFERENT_FROM_0
11. MAX_N
12. MAXIMUM_MODULO
13. MIN_N
14. MINIMUM_EXCEPT_0
15. MINIMUM_GREATER_THAN
16. MINIMUM_MODULO
17. MULTI_INTER_DISTANCE
18. NEQUIVALENCE
19. NEXT_GREATER_ELEMENT
20. NINTERVAL
21. NUMBER_DIGIT
22. SAME_REMAINDER
23. SMOOTH

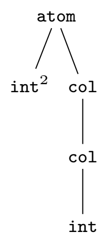
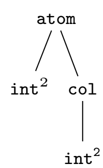
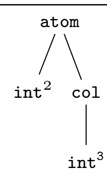
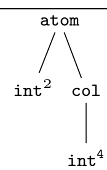
implication graph

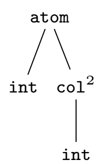
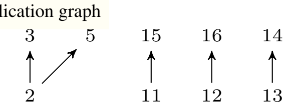
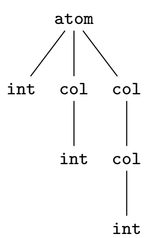
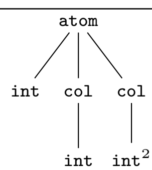
$$\begin{tikzcd} 15 & 2 & 3 \\ 19 \arrow[u] & 8 \arrow[u] \arrow[ur] & \end{tikzcd}$$

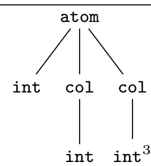
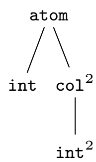
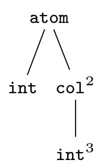
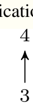
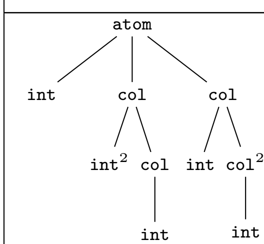
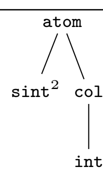
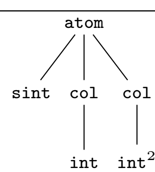
</td>
</tr>
<tr>
<td>atom
int^2 col
sint</td>
<td>

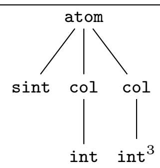
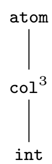
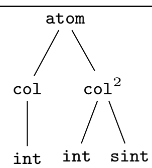
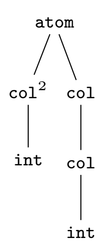
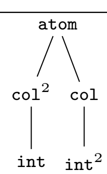
1. SET_VALUE_PRECEDE

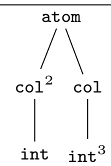
</td>
</tr>
</table>

	1. IN_SAME_PARTITION 2. MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES 3. MAX_OCC_OF_SORTED_TUPLES_OF_VALUES 4. MAX_OCC_OF_TUPLES_OF_VALUES implication graph 
	1. INTERVAL_AND_SUM 2. MAP 3. PATH_FROM_TO 4. TREE_RANGE
	1. INVERSE_OFFSET 2. SHIFT 3. SLIDING_TIME_WINDOW
	1. CYCLE_OR_ACCESSIBILITY 2. SLIDING_TIME_WINDOW_SUM

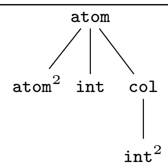
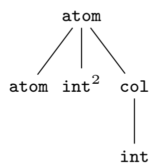
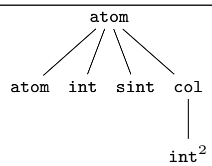
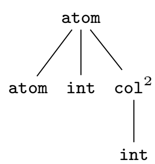
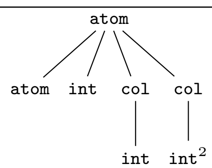
	1. ALLDIFFERENT_SAME_VALUE 2. AMONG 3. AMONG_VAR 4. CARDINALITY_ATLEAST 5. CARDINALITY_ATMOST 6. CLAUSE_AND 7. CLAUSE_OR 8. DIFFER_FROM_AT_LEAST_K_POS 9. ELEMENTN 10. NVALUE_ON_INTERSECTION 11. SAME_INTERVAL 12. SAME_MODULO 13. SOFT_SAME_VAR 14. SOFT_USED_BY_VAR 15. USED_BY_INTERVAL 16. USED_BY_MODULO implication graph 
	1. BALANCE_PARTITION 2. CARDINALITY_ATMOST_PARTITION 3. CHANGE_PARTITION 4. COND_LEX_COST 5. NCLASS
	1. GLOBAL_CARDINALITY_NO_LOOP 2. SUM_OF_WEIGHTS_OF_DISTINCT_VALUES

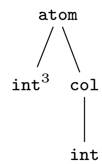
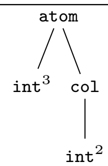
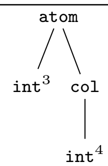
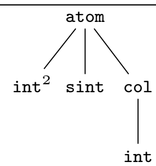
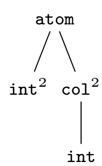
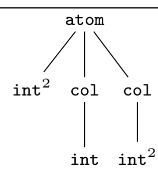
	1. MINIMUM_WEIGHT_ALLDIFFERENT 2. SLIDING_DISTRIBUTION
	1. ELEMENT_SPARSE 2. ELEMENTS_SPARSE implication graph
	1. ORTH_ON_TOP_OF_ORTH 2. TASKS_INTERSECTION 3. TWO_ORTH_COLUMN 4. TWO_ORTH_INCLUDE implication graph 
	1. GEOST
	1. ROOTS
	1. OPEN_GLOBAL_CARDINALITY

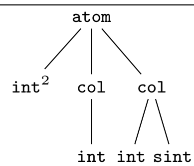
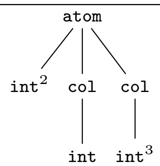
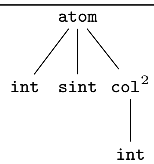
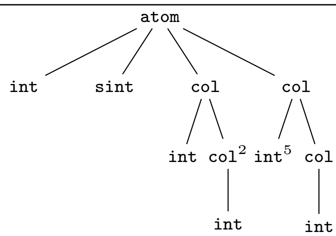
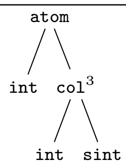
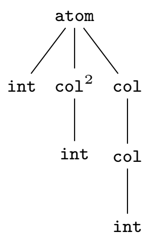
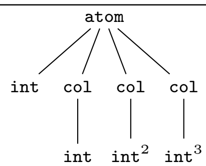
<table>
<tr><td></td><td>1. OPEN_GLOBAL_CARDINALITY_LOW_UP</td></tr>
<tr><td></td><td>1. CORRESPONDENCE
2. LEX_BETWEEN
3. SORT_PERMUTATION
implication graph
$3 \rightarrow 1$</td></tr>
<tr><td></td><td>1. SUBGRAPH_ISOMORPHISM
2. GRAPH_ISOMORPHISM</td></tr>
<tr><td></td><td>1. COND_LEX_GREATER
2. COND_LEX_GREATEREQ
3. COND_LEX_LESS
4. COND_LEX_LESSEQ
5. SAME_PARTITION
6. USED_BY_PARTITION
implication graph
$1 \rightarrow 2$, $3 \rightarrow 4$, $5 \rightarrow 6$</td></tr>
<tr><td></td><td>1. SAME_AND_GLOBAL_CARDINALITY</td></tr>
</table>

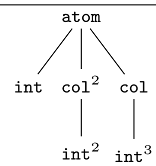
	1. SAME_AND_GLOBAL_CARDINALITY_LOW_UP

3.5.4 Constraints with 4 arguments

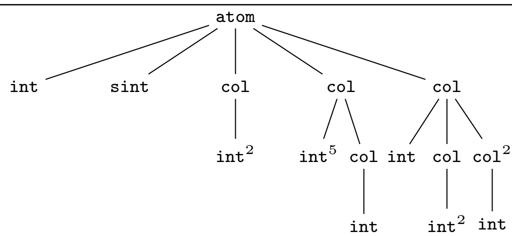
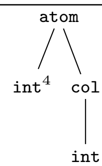
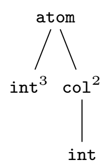
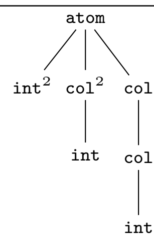
	1. CHANGE_PAIR
	1. COUNT 2. CYCLIC_CHANGE 3. CYCLIC_CHANGE_JOKER implication graph 3 ↑ 2
	1. SUM_SET
	1. ARITH_OR 2. COUNTS 3. DISTANCE_BETWEEN 4. DISTANCE_CHANGE
	1. ASSIGN_AND_COUNTS
	1. IN_INTERVAL_REIFIED

	1. AMONG_INTERVAL 2. AMONG_MODULO 3. ELEMENT_PRODUCT 4. SLIDING_SUM
	1. NEXT_ELEMENT 2. SLIDING_TIME_WINDOW_FROM_START
	1. SOFT_CUMULATIVE
	1. OPEN_ATLEAST 2. OPEN_ATMOST
	1. AMONG_LOW_UP 2. COMMON 3. SLIDING_CARD_SKIP0 4. SOFT_SAME_INTERVAL_VAR 5. SOFT_SAME_MODULO_VAR 6. SOFT_USED_BY_INTERVAL_VAR 7. SOFT_USED_BY_MODULO_VAR . . . . implication graph 6 7 ↑ ↑ 4 5
	1. INTERVAL_AND_COUNT 2. WEIGHTED_PARTIAL_ALLDIFF

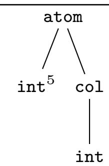
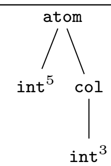
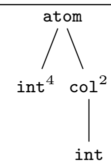
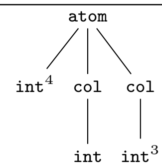
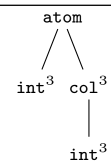
	1. SUM
	1. GLOBAL_CARDINALITY_LOW_UP_NO_LOOP
	1. OPEN_AMONG
	1. GEOST_TIME
	1. DOM_REACHABILITY
	1. SOFT_SAME_PARTITION_VAR 2. SOFT_USED_BY_PARTITION_VAR . . . . implication graph 2 ↑ 1
	1. GLOBAL_CARDINALITY_WITH_COSTS

	1. TWO_LAYER_EDGE_CROSSING

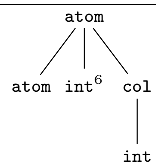
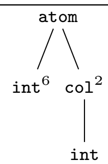
3.5.5 Constraints with 5 arguments

	1. VISIBLE
	1. CONNECT_POINTS
	1. AMONG_SEQ 2. COMMON_INTERVAL 3. COMMON_MODULO
	1. COMMON_PARTITION

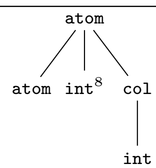
3.5.6 Constraints with 6 arguments

<table>
<tr><td>
</td><td>1. RELAXED_SLIDING_SUM</td></tr>
<tr><td>
</td><td>1. ELEMENT_MATRIX</td></tr>
<tr><td>
</td><td>1. GROUP_SKIP_ISOLATED_ITEM</td></tr>
<tr><td>
</td><td>1. CYCLE_CARD_ON_PATH</td></tr>
<tr><td>
</td><td>1. COLORED_MATRIX</td></tr>
</table>

3.5.7 Constraints with 8 arguments

	1. EQUILIBRIUM
	1. FULL_GROUP 2. GROUP

3.5.8 Constraints with 10 arguments

	1. CHANGE_CONTINUITY

3.6 Meta-keywords attached to the keywords

This section explains the meaning of the meta-keywords attached to the keywords of the catalogue. Keywords are usually associated with a single meta-keyword, except some that are linked to the meta-keyword modelling exercises and to one other meta-keyword like modelling or puzzles (e.g., see the keywords magic series or degree of diversity of a set of solutions). For each meta-keyword it first gives the list of keywords using the corresponding meta-keyword and then defines the meta-keyword. At present the following meta-keywords are in use.

3.6.1 Application area

- Air traffic management,
- Assignment,
- Bioinformatics,
- Configuration problem,
- Deadlock breaking,
- Floor planning problem,
- Frequency allocation problem,
- Phylogeny,
- Program verification,
- SLAM problem,
- Sport timetabling,
- Workload covering.

Denotes that a keyword is related to an application area.

3.6.2 Characteristic of a constraint

- All different,
- Automaton,
- Automaton with array of counters,
- Automaton with counters,
- Automaton with same input symbol,
- Automaton without counters,
- Coloured,
- Consecutive values,

- Convex,
- Convex hull relaxation,
- Core,
- Cyclic,
- Derived collection,
- Difference,
- Disequality,
- Equality,
- Hypergraph,
- Joker value,
- Maximum,
- maxint,
- Minimum,
- Modulo,
- Non-deterministic automaton,
- Pair,
- Partition,
- Product,
- Range,
- Rank,
- Reified automaton constraint,
- Reified constraint,
- Sort,
- Sort based reformulation,
- Sum,
- Time window,
- Tuple,
- Undirected graph,
- Vector.

Denotes that a keyword is related to a characteristic of the description of a constraint.

3.6.3 Combinatorial object

- Involution,
- Latin square,
- Matching,
- Multiset,
- Path,
- Pentomino,
- Periodic,
- Permutation,
- Relation,
- Run of a permutation,
- Sequence.

Denotes that a keyword corresponds to a combinatorial object or to a characteristic of a combinatorial object.

3.6.4 Complexity

- 3-dimensional-matching,
- 3-SAT,
- Minimum hitting set cardinality,
- Rectangle clique partition,
- Sequencing with release times and deadlines,
- Set packing,
- Subset sum.

Denotes that a keyword corresponds to a problem used to recognise NP-hard problems attached to the feasibility of a constraint.

3.6.5 Constraint network structure

- Alpha-acyclic constraint network(2),
- Alpha-acyclic constraint network(3),
- Berge-acyclic constraint network,
- Centered cyclic(1) constraint network(1),
- Centered cyclic(2) constraint network(1),
- Centered cyclic(3) constraint network(1),
- Circular sliding cyclic(1) constraint network(2),
- Sliding cyclic(1) constraint network(1),
- Sliding cyclic(1) constraint network(2),
- Sliding cyclic(1) constraint network(3),
- Sliding cyclic(2) constraint network(2),

Denotes that a keyword designates a specific constraint network structure occurring repeatedly in several constraints.

3.6.6 Constraint type

- Arithmetic constraint,
- Boolean constraint,
- Conditional constraint,
- Constraint on the intersection,
- Counting constraint,
- Data constraint,
- Decomposition,
- Decomposition-based violation measure,
- Extension,
- Graph constraint,
- Graph partitioning constraint,
- Logic,
- Open automaton constraint,
- Open constraint,
- Order constraint,
- Overlapping alldifferent,
- Predefined constraint,
- Proximity constraint,
- Relaxation,
- Resource constraint,
- Scheduling constraint,
- Sliding sequence constraint,
- Soft constraint,
- System of constraints,
- Temporal constraint,
- Timetabling constraint,
- Value constraint,
- Value partitioning constraint,
- Variable-based violation measure.

Denotes that a keyword designates a constraint category.

3.6.7 Constraint arguments

- Aggregate,
- Binary constraint,
- Business rules,
- Constraint between three collections of variables,
- Constraint between two collections of variables,
- Constraint involving set variables,
- Contractible,
- Extensible,
- Pure functional dependency,
- Reverse of a constraint,
- Ternary constraint,
- Unary constraint.

Denotes that a keyword provides an information about the arguments of a constraint.

3.6.8 Filtering

- Abstract interpretation,
- Arc-consistency,
- Bipartite matching,
- Bipartite matching in convex bipartite graphs,
- Border,
- Bound-consistency,
- Compulsory part,
- Constructive disjunction,
- Convex bipartite graph,
- Cost filtering constraint,
- Cumulative longest hole problems,
- DFS-bottleneck,
- Duplicated variables,
- Dynamic programming,
- Entailment,
- Flow,
- Glue matrix,
- Hall interval,
- Hungarian method for the assignment problem,
- Hybrid-consistency,
- Klee measure problem,
- Linear programming,
- Minimum cost flow,
- Minimum task duration,
- Phi-tree,
- Planarity test,
- Quadtree,
- SAT,
- Strong articulation point,
- Strong bridge,
- Sweep.

Denotes that a keyword is related to an existing or a potential filtering algorithm of a constraint or to an algorithm checking a ground instance of a constraint.

3.6.9 Final graph structure

- Acyclic,
- Apartition,
- Bipartite,
- Circuit,
- Connected component,
- Consecutive loops are connected,
- Directed acyclic graph,
- Equivalence,
- No cycle,
- No loop,
- One succ,
- Strongly connected component,
- Symmetric,
- Tree,
- Vpartition.

Denotes that a keyword describes the structure of the final graph associated with a constraint.

3.6.10 Geometry

- Alignment,
- Contact,
- Geometrical constraint,
- Guillotine cut,
- Line segments intersection,
- Non-overlapping,
- Orthotope,
- Polygon,
- Positioning constraint,
- RCC8,
- Touch.

Denotes that a keyword is related to a geometrical constraint or to a geometrical object.

3.6.11 Heuristics

- Heuristics,
- Heuristics and Berge-acyclic constraint network,
- Heuristics and lexicographical ordering,
- Heuristics for two-dimensional rectangle placement problems,
- Labelling by increasing cost,
- Limited discrepancy search,
- Regret based heuristics,
- Regret based heuristics in matrix problems.

Denotes that a keyword is related to a search heuristic.

3.6.12 Miscellaneous

- Obscure.

Denotes that a keyword does not belong to any class.

3.6.13 Modelling

- Array constraint,
- Assigning and scheduling tasks that run in parallel,
- Assignment dimension,
- Assignment to the same set of values,
- At least,
- At most,
- Balanced assignment,
- Balanced tree,
- Boolean channel,
- Channelling constraint,
- Cluster,
- Cost matrix,
- Cycle,
- Degree of diversity of a set of solutions,
- Difference between pairs of variables,
- Disjunction,
- Domain channel,
- Domain definition,
- Dual model,
- Empty intersection,
- Equality between multisets,
- Excluded,
- Functional dependency,
- Included,
- Inclusion,
- Incompatible pairs of values,
- Interval,

- Matrix,
- Matrix model,
- Maximum number of occurrences,
- Minimum number of occurrences,
- Multi-site employee scheduling with calendar constraints,
- Number of changes,
- Number of distinct equivalence classes,
- Number of distinct values,
- Permutation channel,
- Preferences,
- Relaxation dimension,
- Scalar product,
- Scheduling with machine choice, calendars and preemption,
- Sequence dependent set-up,
- Set channel,
- Shared table,
- Sparse functional dependency,
- Sparse table,
- Statistics,
- Table,
- Variable indexing,
- Variable subscript,
- Zero-duration task.

Denotes that a keyword is related to a modelling issue.

3.6.14 Modelling exercises

- Assigning and scheduling tasks that run in parallel: inspired by a modelling question on the **Choco** mailing list about an assignment and scheduling problem involving nurses and surgeons, use one GEOST constraint as well as inequalities for breaking symmetries with respect to groups of identical persons. The keyword relaxation dimension shows how to extend the previous model in order to take into account over-constrained assignment and scheduling problems.
- Assignment to the same set of values: inspired by a presentation of F. Hermenier about a task assignment problem where subtasks have to be assigned a same group of machines, use several ELEMENT constraints and a single resource constraint that has an assignment dimension (e.g., BIN_PACKING, CUMULATIVES, DIFFN, GEOST).

- Degree of diversity of a set of solutions: inspired by a discussion with E. Hebrard, how to find out 9 *completely different solutions* for the 10-queens problem, use the ALLDIFFERENT, the SOFT_ALLDIFFERENT_CTR and the LEX_CHAIN_LESS constraints.
- Logigraphe: inspired by an instance from [328, page 36], use a conjunction of CONSECUTIVE_GROUPS_OF_ONES constraints.
- Magic series: a special case of Autoref, use a single GLOBAL_CARDINALITY constraint.
- Metro: a model from H. Simonis, use only LEQ_CST constraints and propagation (i.e., no enumeration) for modelling the *shortest path* problem in a network.
- Multi-site employee scheduling with calendar constraints: a timetabling problem, inspired by H. Simonis, where tasks have to be assigned groups of employes located in different countries subject to different calendars, use resource constraints as well as the CALENDAR constraint.
- n-Amazons: an extension of the n-queens problem, use one ALLDIFFERENT constraint, two ALLDIFFERENT_CST constraints and three SMOOTH constraints.
- relaxation dimension: illustrate how to model over-constrained placement problems by introducing an extra dimension in the context of the DIFFN and the GEOST constraints.
- Scheduling with machine choice, calendars and preemption: a scheduling problem with *crossable and non-crossable unavailability periods* as well as *resumable and non-resumable tasks*, illustrate the use of *two time coordinates systems* within the same model, use precedence and resource constraints as well as the CALENDAR constraint.
- Sequence dependent set-up: a classical scheduling problem, use the SUM_CTR, ELEMENT and TEMPORAL_PATH constraints.
- Zebra puzzle: illustrate the *duality of choice of what is a variable and what is a value* in a constraint model as well as the difficulty of stating the constraints in one of the two models, use the ALLDIFFERENT, the ELEMENT – with variables in the table – and the INVERSE constraints.

Denotes that a keyword describes a constraint modelling exercise.

3.6.15 Problems

- Channel routing,
- Demand profile,
- Domination,
- Facilities location problem,
- Graph colouring,
- Hamiltonian,
- Maximum clique,
- Minimum feedback vertex set,
- Pallet loading,
- Pattern sequencing,
- Pick-up delivery,
- Producer-consumer,
- Schur number,
- Strip packing,
- Two-dimensional orthogonal packing,
- Weighted assignment.

Denotes that a keyword is related to a problem from Operations Research.

3.6.16 Puzzles

- Autoref,
- Conway packing problem,
- Costas arrays,
- Dominating queens,
- Euler knight,
- Golomb ruler,
- Logigraphe,
- Magic hexagon,
- Magic series,
- Magic square,
- n-Amazons,
- n-queens,
- Packing almost squares,
- Partridge,
- pentomino,
- Shikaku,
- Smallest rectangle area,
- Smallest square for packing consecutive dominoes,
- Smallest square for packing rectangles with distinct sizes,
- Squared squares,

- Sudoku,
- Zebra puzzle.

Denotes that a keyword is related to a specific puzzle.

3.6.17 Symmetry

- Indistinguishable values,
- Lexicographic order,
- Matrix symmetry,
- Multiset ordering,
- Symmetry,
- Value precedence.

Denotes that a keyword is related to a symmetry breaking technique [129, 180].

3.7 Keywords attached to the global constraints

This section explains the meaning of the keywords attached to the global constraints of the catalogue. For each keyword it first gives the list of global constraints using the corresponding keyword and then defines the keyword. At present the following keywords are in use.

3.7.1 3-dimensional-matching [2 CONS]

- K_SAME,
- SOFT_ALL_EQUAL_MIN_CTR.

Denotes that, by reduction to *3-dimensional-matching*, deciding whether a constraint has a solution or not was shown to be NP-hard. The *3-dimensional-matching* problem can be described as follows: given a set $S \subseteq X \times Y \times Z$, where X, Y and Z are disjoint sets having the same number of elements m, does S contain a subset M of m elements such that no two elements of M agree in any coordinate?

3.7.2 3-SAT [5 CONS]

- ATMOST_NVALUE,
- COMMON,
- GLOBAL_CARDINALITY,
- NVALUE,
- USES.

Denotes that, by reduction to *3-SAT*, deciding whether a constraint has a solution or not was shown to be NP-hard. The *3-SAT* problem can be described as follows: given a collection C of clauses involving a set of variables V, where each clause has exactly 3 variables, is there a truth assignment for V that satisfies all the clauses of C?

3.7.3 Abstract interpretation [2 CONS]

- GCD,
- POWER.

Denotes that abstract interpretation was used for deriving a filtering algorithm for a constraint C from a polynomial algorithm describing a checker for a ground instance of C. *Abstract interpretation* [136] executes an algorithm on abstract values in order to deduce some information about that algorithm.

3.7.4 Acyclic [28 CONS]

- ALLDIFFERENT_ON_INTERSECTION,
- ALLPERM,
- AMONG_LOW_UP,
- AMONG_VAR,
- ARITH_OR,
- ASSIGN_AND_COUNTS,
- ASSIGN_AND_NVALUES,
- BIN_PACKING,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST,
- CARDINALITY_ATMOST_PARTITION,
- CHANGE,
- CHANGE_CONTINUITY,
- CHANGE_PAIR,
- CHANGE_PARTITION,
- COMMON,
- COMMON_INTERVAL,
- COMMON_MODULO,
- COMMON_PARTITION,
- CORRESPONDENCE,
- COUNTS,
- CROSSING,
- CUTSET,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- DECREASING,
- LEX_EQUAL,
- USES.

Denotes that a constraint is defined by a single graph constraint for which the final graph does not have any circuit.

3.7.5 Aggregate [36 CONS]

- AMONG (+, union, sunion),
- AMONG_DIFF_0 (+, union),
- AMONG_INTERVAL (+, union, id, id),
- AMONG_LOW_UP (+, +, union, sunion),
- AMONG_MODULO (+, union, id, id),
- AMONG_VAR (+, union, union),
- AND ($\wedge$, union),
- COUNT (id, union, id, +) when RELOP $\in [<, \leq, \geq, >]$,
- COUNTS (unions, union, id, +) when RELOP $\in [<, \leq, \geq, >]$,
- DISCREPANCY (union, +),
- EXACTLY (+, union, id),
- INT_VALUE_PRECEDE (id, id, union),
- INT_VALUE_PRECEDE_CHAIN (id, union),
- MAXIMUM (max, union),
- MINIMUM (min, union),
- MINIMUM_GREATER_THAN (min, id, union),
- NAND ($\vee$, union),
- NOR ($\wedge$, union),
- OR ($\vee$, union),
- PRODUCT_CTR (union, id, $*$) when CTR $\in [=]$,
- SAME (union, union),
- SAME_INTERVAL (union, union, id),
- SAME_MODULO (union, union, id),
- SAME_PARTITION (union, union, id),
- SCALAR_PRODUCT (union, id, +),
- SUM_CTR (union, id, +),
- SUM_CUBES_CTR (union, id, +),
- SUM_POWERS4_CTR (union, id, +),
- SUM_POWERS5_CTR (union, id, +),
- SUM_POWERS6_CTR (union, id, +),
- SUM_SQUARES_CTR (union, id, +),
- USED_BY (union, union),
- USED_BY_INTERVAL (union, union, id),
- USED_BY_MODULO (union, union, id),
- USED_BY_PARTITION (union, union, id),
- USES (union, union).

Denotes that, given two instances of a constraint, we can combine (i.e., aggregate) these two instances in order to obtain a third constraint, which has the same name as the first two constraints. The first two constraints are called the *source* constraints, while the implied constraint is called the *target* constraint. The i^{th} argument of the target constraint is obtained by combining the i^{th} arguments of the two source constraints. This is specified for each argument by one of the following options.

- id: check that the corresponding arguments of the two source constraints are *identical* and take it as the argument of the target constraint; this option if often used for specifying that an argument corresponding to a parameter has to be the same in the two source constraints, as well as in the target constraint (i.e., the source and the target constraints share the same parameter).
- $+$: *add* the corresponding arguments of the two source constraints.
- $*$: *multiply* the corresponding arguments of the two source constraints.
- $\wedge$: make an *and* between the corresponding 0-1 arguments of the two source constraints.
- $\vee$: make an *or* between the corresponding 0-1 arguments of the two source constraints.

- $\min$: take the *minimum* of the corresponding arguments of the two source constraints.
- $\max$: take the *maximum* of the corresponding arguments of the two source constraints.
- `union`: take the *union*, without removing duplicates, of the collections items of the corresponding arguments of the two source constraints.
- `sunion`: take the *union*, and remove duplicates, of the collections items of the corresponding arguments of the two source constraints, where collections correspond to collection of ground values (i.e., parameters).

Finally, the aggregation may me be conditioned by a list of restrictions, each restriction corresponding to one of the restrictions described in Section 2.2.3. We call this *conditional aggregation*.

Most constraints for which aggregation applies correspond to constraints where one of the arguments is functionally determined by the other arguments. For example, this is the case for the MAXIMUM(MAX, VARIABLES) constraint which forces MAX to be equal to the maximum value assigned to the variables of VARIABLES. However some constraints, like the SAME constraint, for which aggregation applies, do not have any argument that is functionally determined by the other arguments.

We now present three examples of deductions that can be obtained by aggregating two source constraints.

- AMONG$(1, \langle 4,5,5,4,1\rangle, \langle 0,1\rangle) \wedge$ AMONG$(3, \langle 1,1,9,0\rangle, \langle 0,1\rangle) \Rightarrow$ AMONG$(4, \langle 4,5,5,4,1,1,1,9,0\rangle, \langle 0,1\rangle)$, where:
  1. The first argument of the target constraint, i.e., 4, is equal to the sum of the first arguments of the two source constraints, i.e., $1+3$.
  2. The second argument of the target constraint, $\langle 4,5,5,4,1,1,1,9,0\rangle$, is equal to the union (without removing duplicates) of the second arguments $\langle 4,5,5,4,1\rangle$ and $\langle 1,1,9,0\rangle$ of the two source constraints.
  3. The third arguments of the two source constraints are identical, i.e., $\langle 0,1\rangle$, and the third argument of the target constraint.
- MAXIMUM$(5, \langle 3,0,5,2,5\rangle) \wedge$ MAXIMUM$(9, \langle 1,1,9,0\rangle) \Rightarrow$ MAXIMUM$(9, \langle 3,0,5,2,5,1,1,9,0\rangle)$, where:
  1. The first argument of the target constraint, i.e., 9, is equal to the maximum value of the first arguments of the two source constraints, i.e., $\max(5,9)$.
  2. The second argument of the target constraint, $\langle 3,0,5,2,5,1,1,9,0\rangle$, is equal to the union (without removing duplicates) of the second arguments $\langle 3,0,5,2,5\rangle$ and $\langle 1,1,9,0\rangle$ of the two source constraints.
- SAME$(\langle 3,3,1\rangle, \langle 3,1,3\rangle) \wedge$ SAME$(\langle 1,9,1,5,5\rangle, \langle 5,5,1,1,9\rangle) \Rightarrow$ SAME$(\langle 3,3,1,1,9,1,5,5\rangle, \langle 3,1,3,5,5,1,1,9\rangle)$, where:

1. The first argument of the target constraint, $\langle 3, 3, 1, 1, 9, 1, 5, 5\rangle$, is equal to the union (without removing duplicates) of the first arguments $\langle 3, 3, 1\rangle$ and $\langle 1, 9, 1, 5, 5\rangle$ of the two source constraints.
2. The second argument of the target constraint, $\langle 3, 1, 3, 5, 5, 1, 1, 9\rangle$, is equal to the union (without removing duplicates) of the second arguments $\langle 3, 1, 3\rangle$ and $\langle 5, 5, 1, 1, 9\rangle$ of the two source constraints.

3.7.6 ▾Air traffic management ➠ [4 CONS]

- ALL_MIN_DIST,
- K_ALLDIFFERENT,
- MULTI_INTER_DISTANCE,
- SORT.

Denotes that a constraint was used for solving a problem in the area of air traffic management.

3.7.7 ▾Alignment ➠ [1 CONS]

- ORCHARD.

Denotes that a constraint forces the alignment of different sets of points.

3.7.8 All different [20 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_BETWEEN_SETS,
- ALLDIFFERENT_CST,
- ALLDIFFERENT_CONSECUTIVE_VALUES,
- ALLDIFFERENT_EXCEPT_0,
- ALLDIFFERENT_INTERVAL,
- ALLDIFFERENT_MODULO,
- ALLDIFFERENT_ON_INTERSECTION,
- ALLDIFFERENT_PARTITION,
- GOLOMB,
- K_ALLDIFFERENT,
- OPEN_ALLDIFFERENT,
- PERMUTATION,
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT,
- SIZE_MAX_SEQ_ALLDIFFERENT,
- SOFT_ALLDIFFERENT_CTR,
- SOFT_ALLDIFFERENT_VAR,
- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_LOOP,
- WEIGHTED_PARTIAL_ALLDIFF.

Denotes that we have one or several cliques of disequalities or that a constraint is a variation of the ALLDIFFERENT constraint. Variations may be related to relaxation (see, e.g., the ALLDIFFERENT_EXCEPT_0, SOFT_ALLDIFFERENT_CTR, and SOFT_ALLDIFFERENT_VAR constraints), or to specialisation (see, e.g., the SYMMETRIC_ALLDIFFERENT constraint), of the ALLDIFFERENT constraint. Variations may also result from an extension of the notion of disequality (see, e.g., the ALLDIFFERENT_INTERVAL, ALLDIFFERENT_MODULO, ALLDIFFERENT_PARTITION and GOLOMB constraints).

3.7.9 Alpha-acyclic constraint network(2) [15 CONS]

- AMONG,
- AMONG_DIFF_0,
- AMONG_INTERVAL,
- AMONG_LOW_UP,
- AMONG_MODULO,
- ATLEAST,
- ATMOST,
- COUNT,
- COUNTS,
- DIFFER_FROM_AT_LEAST_K_POS,
- EXACTLY,
- FULL_GROUP,
- GROUP,
- GROUP_SKIP_ISOLATED_ITEM,
- SLIDING_CARD_SKIP0.

Before defining *alpha-acyclic constraint network(2)* we first need to introduce the following notions:

- The *dual graph* of a constraint network $\mathcal{N}$ is defined in the following way: to each constraint of $\mathcal{N}$ corresponds a vertex in the dual graph and if two constraints have a non-empty set S of shared variables, there is an edge labelled S between their corresponding vertices in the dual graph.
- An edge in the dual graph of a constraint network is *redundant* if its variables are shared by every edge along an alternative path between the two end points [144].
- If the subgraph resulting from the removal of the redundant edges of the dual graph is a tree the original constraint network is called α-acyclic [168].

Alpha-acyclic constraint network(2) denotes an α-acyclic constraint network such that, for any pair of constraints, the two sets of involved variables share at most two variables.

3.7.10 Alpha-acyclic constraint network(3) [5 CONS]

- FULL_GROUP,
- GROUP,
- GROUP_SKIP_ISOLATED_ITEM,
- ITH_POS_DIFFERENT_FROM_0,
- MIN_SIZE_FULL_ZERO_STRETCH.

Alpha-acyclic constraint network(3) denotes an α-acyclic constraint network (see *alpha-acyclic constraint network(2)*) such that, for any pair of constraints, the two sets of involved variables share at most three variables.

3.7.11 Apartition [1 CONS]

- CHANGE_CONTINUITY.

Denotes that a constraint is defined by two graph constraints having the same initial graph, where each arc of the initial graph belongs to one of the final graphs (but not to both).

3.7.12 Arc-consistency [117 CONS]

- ABS_VALUE,
- ALLDIFFERENT,
- ALLDIFFERENT_CST,
- ALLDIFFERENT_EXCEPT_0,
- ALLDIFFERENT_INTERVAL,
- ALLDIFFERENT_MODULO,
- ALLDIFFERENT_PARTITION,
- AMONG,
- AMONG_DIFF_0,
- AMONG_INTERVAL,
- AMONG_LOW_UP,
- AMONG_MODULO,
- AMONG_SEQ,
- AND,
- ARITH,
- ARITH_OR,
- ATLEAST,
- ATLEAST_NVALUE,
- ATMOST,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST,
- CARDINALITY_ATMOST_PARTITION,
- CLAUSE_AND,
- CLAUSE_OR,
- COND_LEX_COST,
- COND_LEX_GREATER,
- COND_LEX_GREATEREQ,
- COND_LEX_LESS,
- COND_LEX_LESSEQ,
- CONSECUTIVE_GROUPS_OF_ONES,
- COUNT,
- COUNTS,
- DECREASING,
- DERANGEMENT,
- DISCREPANCY,
- DIVISIBLE,
- DOMAIN_CONSTRAINT,
- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENTN,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_MATRIX,
- ELEMENT_SPARSE,
- ELEMENTS,
- ELEMENTS_SPARSE,
- EQ,
- EQ_CST
- EQUIVALENT,

- EXACTLY,
- GEQ,
- GEQ_CST,
- GLOBAL_CARDINALITY_LOW_UP,
- GLOBAL_CONTIGUITY,
- GT,
- IMPLY,
- IN,
- IN_INTERVAL,
- IN_INTERVAL_REIFIED,
- IN_INTERVALS,
- IN_RELATION,
- IN_SAME_PARTITION,
- INCREASING,
- INCREASING_GLOBAL_CARDINALITY,
- INCREASING_NVALUE,
- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- INVERSE,
- INVERSE_OFFSET,
- LEQ,
- LEQ_CST,
- LEX_ALLDIFFERENT,
- LEX_BETWEEN,
- LEX_CHAIN_GREATER,
- LEX_CHAIN_GREATEREQ,
- LEX_CHAIN_LESS,
- LEX_CHAIN_LESSEQ,
- LEX_DIFFERENT,
- LEX_EQUAL,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LT,
- MAXIMUM,
- MINIMUM,
- NAND,
- NEQ,
- NEQ_CST,
- NOR,
- NOT_ALL_EQUAL,
- NOT_IN,
- OPPOSITE_SIGN,
- OR,
- ORDERED_GLOBAL_CARDINALITY,
- PATTERN,
- PRECEDENCE,
- SAME,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP,
- SAME_SIGN,
- SIGN_OF,
- SOFT_ALL_EQUAL_MAX_VAR,
- SOFT_ALL_EQUAL_MIN_VAR,
- STAGE_ELEMENT,
- STRETCH_CIRCUIT,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION,
- STRICTLY_DECREASING,
- STRICTLY_INCREASING,
- SYMMETRIC_ALLDIFFERENT,
- TREE,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_DO_NOT_OVERLAP,
- USED_BY,
- VEC_EQ_TUPLE,
- XOR.

Denotes that, for a given constraint involving only domain variables, there is a filtering algorithm that ensures *arc-consistency*. A constraint *ctr* defined on the distinct domain variables $V_1, \ldots, V_n$ is *arc-consistent* if and only if for every pair (V, v) such that V is a domain variable of *ctr* and $v \in dom(V)$, there exists at least one solution to *ctr* in which V is assigned the value v. As quoted by C. Bessière in [65], "*a different name has often been used for arc-consistency on non-binary constraints*", like *domain consistency*, *generalised arc-consistency* or *hyper arc-consistency*.

There is also a weaker form of *arc-consistency* that also try to remove values from the middle of the domain of a variable V (i.e., unlike bound-consistency which focus on reducing the minimum and maximum value of a variable), called *range consistency* in [65], that is defined in the following way. A constraint *ctr* defined on the distinct domain variables $V_1, \ldots, V_n$ is *range-consistent* if and only if, for every pair (V, v) such that V is a domain variable of *ctr* and $v \in dom(V)$, there exists at least a solution to *ctr* in which, (1) V is assigned the value v, and (2) each variable $U \in \{V_1, \ldots, V_n\}$ distinct from V is assigned a value located in its range $[\underline{U}, \overline{U}]$.

3.7.13 Arithmetic constraint [37 CONS]

- ABS_VALUE,
- ARITH_SLIDING,
- DISTANCE,
- DIVISIBLE,
- DIVISIBLE_OR,
- EQ,
- EQ_CST,
- GCD,
- GEQ,
- GEQ_CST,
- GT,
- INCREASING_SUM,
- LEQ,
- LEQ_CST,
- LT,
- MULTIPLE,
- NEQ,
- NEQ_CST,
- NUMBER_DIGIT,
- OPPOSITE_SIGN,
- POWER,
- PRODUCT_CTR,
- RANGE_CTR,
- REMAINDER,
- SAME_REMAINDER,
- SAME_SIGN,
- SIGN_OF,
- SCALAR_PRODUCT,
- SUM_CTR,
- SUM_SET,
- SUM_CUBES_CTR,
- SUM_POWERS4_CTR,
- SUM_POWERS5_CTR,
- SUM_POWERS6_CTR,
- SUM_SQUARES_CTR,
- ZERO_OR_NOT_ZERO,
- ZERO_OR_NOT_ZERO_VECTORS.

An arithmetic constraint between two or three variables or an arithmetic constraint involving a sum, a product, or a difference between a maximum and a minimum value. The non binary constraints were introduced within the catalogue since they are required for defining a given global constraint. For example, the SUM_CTR constraint is used within the definition of the CUMULATIVE constraint.

3.7.14 Array constraint [9 CONS]

- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENTS_ALLDIFFERENT,
- ELEMENT_LESSEQ,
- ELEMENT_GREATEREQ,
- ELEMENT_MATRIX,
- ELEMENT_PRODUCT,
- ELEMENT_SPARSE.

A constraint that allows for expressing simple array equations.

3.7.15 Assigning and scheduling tasks that run in parallel [3 CONS]

- DIFFN,
- GEOST,
- GEOST_TIME.

Consider a set of tasks defined by a set of subtasks, where each subtask has the following attributes:

- A *start* telling when the subtask starts.
- A *duration* giving the duration of the subtask.
- A *deadline* indicating the date by which the subtask must be finished.
- A *person* indicating which person performs the subtask.

Both the start and the person correspond to decision variables, while the duration and the deadline are integers. Since all subtasks of a same task must run in parallel, their starts, durations and deadlines are identical. Since a person can perform at most one task at each timepoint, persons assigned to the subtasks of a same task must all be distinct. We also assume that a subtask cannot be preempted.

As an instance of this pattern, consider the problem of scheduling surgical operations in a hospital. Each surgery corresponds to a task that requires a number of persons with specific skills; these persons will all work together during the operation (e.g., typically an anaesthetist, a surgeon and one or several nurses). Moreover, each person has his own calendar defining his unavailability. On the one hand, let us assume we have two anaesthetists, two surgeons and four nurses, labelled from 1 to 8. Each of them has the following unavailability over the time horizon $[0, 24]$:

- The first anaesthetist is not available during the time periods $[0, 1]$, $[5, 6]$, and $[12, 16]$.
- The second anaesthetist is not available during the time periods $[0, 2]$, $[6, 6]$, $[15, 15]$, and $[22, 22]$.
- The first surgeon is not available during the time periods $[0, 1]$, $[8, 9]$, and $[13, 14]$.
- The second surgeon is not available during the time periods $[5, 5]$, and $[20, 21]$.
- The four nurses are all not available during the time periods $[0, 0]$, $[7, 7]$, $[12, 12]$, and $[22, 22]$.

On the other hand, let us suppose we have to schedule five surgery tasks, each of them requiring a specific team:

- Task t_1 needs one anaesthetist, one surgeon and two nurses during two consecutive time slots.
- Task t_2 needs one anaesthetist, one surgeon and one nurse during four consecutive time slots.
- Task t_3 needs one anaesthetist, two surgeons and two nurses during three consecutive time slots.
- Task t_4 needs one anaesthetist, one surgeon and three nurses during two consecutive time slots.
- Task t_5 needs one anaesthetist, one surgeon and one nurse during six consecutive time slots.

Moreover, tasks t_1, t_2, t_3, t_4 and t_5 must be respectively completed no later than timepoint 12, 15, 24, 24 and 24. The problem is modelled by using a two-dimensional GEOST constraint, where the first and second dimensions respectively correspond to the time and resource axes. For each person required by a task we create a rectangle of length corresponding to the necessary duration and of height 1 (since it requires one person). The coordinates of the lower left point of the rectangle correspond to the start of the corresponding task as well as to the person that will be assigned to the subtask (i.e., a value between 1 and 2 for an anaesthetist, a value between 3 and 4 for a surgeon, and a value between 5 and 8 for a nurse). Both the start and the person correspond to a domain variable. Each unavailability period of an anaesthetist, a surgeon and a nurse is modelled by introducing a fixed rectangle (i.e., its coordinates are set to the start of the unavailability period and to the person to which the unavailability belongs; its duration is set to the duration of the unavailability period) that prevent tasks overlapping the corresponding time period for a specific person. This leads to the following GEOST constraint, GEOST(2, ← - - - *number of dimensions of the placement space: time and resources*

objects — *subtasks of tasks* $t_1, t_2, \ldots, t_5$	⟨`oid` − 1	`sid` − 2	`x` − $\langle o_1, a_1\rangle$,	`oid` − 2	`sid` − 2	`x` − $\langle o_1, s_1\rangle$,
	`oid` − 3	`sid` − 2	`x` − $\langle o_1, n_{11}\rangle$,	`oid` − 4	`sid` − 2	`x` − $\langle o_1, n_{12}\rangle$,
	`oid` − 5	`sid` − 4	`x` − $\langle o_2, a_2\rangle$,	`oid` − 6	`sid` − 4	`x` − $\langle o_2, s_2\rangle$,
	`oid` − 7	`sid` − 4	`x` − $\langle o_2, n_2\rangle$,	`oid` − 8	`sid` − 3	`x` − $\langle o_3, a_3\rangle$,
	`oid` − 9	`sid` − 3	`x` − $\langle o_3, s_{31}\rangle$,	`oid` − 10	`sid` − 3	`x` − $\langle o_3, s_{32}\rangle$,
	`oid` − 11	`sid` − 3	`x` − $\langle o_3, n_{31}\rangle$,	`oid` − 12	`sid` − 3	`x` − $\langle o_3, n_{32}\rangle$,
	`oid` − 13	`sid` − 2	`x` − $\langle o_4, a_4\rangle$,	`oid` − 14	`sid` − 2	`x` − $\langle o_4, s_4\rangle$,
	`oid` − 15	`sid` − 2	`x` − $\langle o_4, n_{41}\rangle$,	`oid` − 16	`sid` − 2	`x` − $\langle o_4, n_{42}\rangle$,
	`oid` − 17	`sid` − 2	`x` − $\langle o_4, n_{43}\rangle$,	`oid` − 18	`sid` − 6	`x` − $\langle o_5, a_5\rangle$,
	`oid` − 19	`sid` − 6	`x` − $\langle o_5, s_5\rangle$,	`oid` − 20	`sid` − 6	`x` − $\langle o_5, n_5\rangle$,
objects — *unavailability periods*	`oid` − 21	`sid` − 2	`x` − $\langle 0, 1\rangle$,	`oid` − 22	`sid` − 2	`x` − $\langle 5, 1\rangle$,
	`oid` − 23	`sid` − 5	`x` − $\langle 12, 1\rangle$,	`oid` − 24	`sid` − 3	`x` − $\langle 0, 2\rangle$,
	`oid` − 25	`sid` − 1	`x` − $\langle 6, 2\rangle$,	`oid` − 26	`sid` − 1	`x` − $\langle 15, 2\rangle$,
	`oid` − 27	`sid` − 1	`x` − $\langle 22, 2\rangle$,	`oid` − 28	`sid` − 2	`x` − $\langle 0, 3\rangle$,
	`oid` − 29	`sid` − 2	`x` − $\langle 8, 3\rangle$,	`oid` − 30	`sid` − 2	`x` − $\langle 13, 3\rangle$,
	`oid` − 31	`sid` − 1	`x` − $\langle 5, 4\rangle$,	`oid` − 32	`sid` − 2	`x` − $\langle 20, 4\rangle$,
	`oid` − 33	`sid` − 1	`x` − $\langle 0, 5\rangle$,	`oid` − 34	`sid` − 1	`x` − $\langle 7, 5\rangle$,
	`oid` − 35	`sid` − 1	`x` − $\langle 12, 5\rangle$,	`oid` − 36	`sid` − 1	`x` − $\langle 22, 5\rangle$,
	`oid` − 37	`sid` − 1	`x` − $\langle 0, 6\rangle$,	`oid` − 38	`sid` − 1	`x` − $\langle 7, 6\rangle$,
	`oid` − 39	`sid` − 1	`x` − $\langle 12, 6\rangle$,	`oid` − 40	`sid` − 1	`x` − $\langle 22, 6\rangle$,
	`oid` − 41	`sid` − 1	`x` − $\langle 0, 7\rangle$,	`oid` − 42	`sid` − 1	`x` − $\langle 7, 7\rangle$,
	`oid` − 43	`sid` − 1	`x` − $\langle 12, 7\rangle$,	`oid` − 44	`sid` − 1	`x` − $\langle 22, 7\rangle$,
	`oid` − 45	`sid` − 1	`x` − $\langle 0, 8\rangle$,	`oid` − 46	`sid` − 1	`x` − $\langle 7, 8\rangle$,
	`oid` − 47	`sid` − 1	`x` − $\langle 12, 8\rangle$,	`oid` − 48	`sid` − 1	`x` − $\langle 22, 8\rangle\rangle$,
shapes	⟨`sid` − 1	`t` − $\langle 0, 0\rangle$	`l` − $\langle 1, 1\rangle$,	`sid` − 2	`t` − $\langle 0, 0\rangle$	`l` − $\langle 2, 1\rangle$,
	`sid` − 3	`t` − $\langle 0, 0\rangle$	`l` − $\langle 3, 1\rangle$,	`sid` − 4	`t` − $\langle 0, 0\rangle$	`l` − $\langle 4, 1\rangle$,
	`sid` − 5	`t` − $\langle 0, 0\rangle$	`l` − $\langle 5, 1\rangle$,	`sid` − 6	`t` − $\langle 0, 0\rangle$	`l` − $\langle 6, 1\rangle\rangle$).

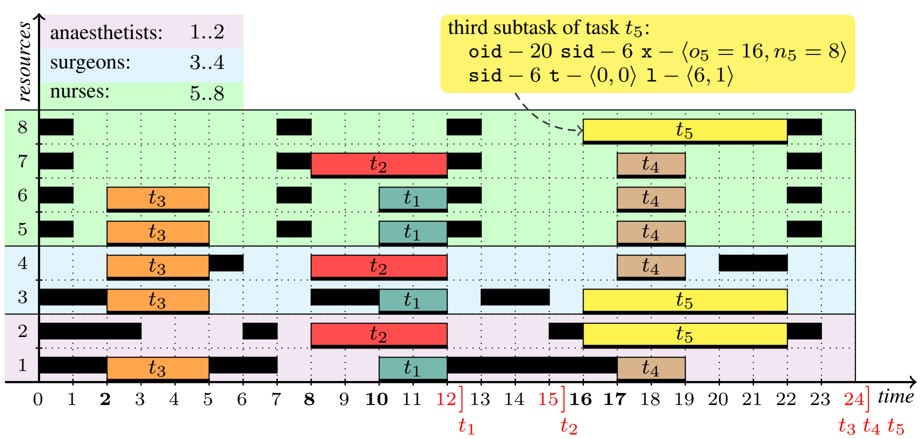


Figure 3.2: A solution to the surgery scheduling problem using four nurses where the start and the latest completion time of each task are respectively shown in bold and in red; a solution using only 3 nurses can be obtained by starting task t_4 at instant 13 and by assigning it to the second anaesthetist rather than to the first one.

A deadline constraint for a surgery starting at o and of duration d is modelled by a precedence constraint of the form $o + d \leq deadline$. This leads to the five constraints $o_1 + 2 \leq 12$, $o_2 + 4 \leq 15$, $o_3 + 3 \leq 24$, $o_4 + 2 \leq 24$, and $o_5 + 6 \leq 24$. Finally, we break symmetry on the assignment variables corresponding to a group of similar persons. In the example, the four nurses are similar since (1) they all have exactly the same unavailability periods, and since (2) no task requires a specific nurse. For each task using more than one nurse (i.e., tasks t_1, t_3, and t_4) this leads to a chain of strict inequalities, i.e., $n_{11} < n_{12}$, $n_{31} < n_{32}$, and $n_{41} < n_{42} < n_{43}$. Figure 3.2 depicts a solution to the problem corresponding to the assignment

tasks	origin	anaesthetist	surgeon	nurse
t_1	$o_1 = 10$	$a_1 = 1$	$s_1 = 3$	$n_{11} = 5, n_{12} = 6$
t_2	$o_2 = 8$	$a_2 = 2$	$s_2 = 4$	$n_2 = 7$
t_3	$o_3 = 2$	$a_3 = 1$	$s_{31} = 3, s_{32} = 4$	$n_{31} = 5, n_{32} = 6$
t_4	$o_4 = 17$	$a_4 = 1$	$s_4 = 4$	$n_{41} = 5, n_{42} = 6, n_{43} = 7$
t_5	$o_5 = 16$	$a_5 = 2$	$s_5 = 3$	$n_5 = 8$

The entry corresponding to the keyword relaxation dimension shows how to express relaxation in the context of over-constrained problems where we have too many surgeries to schedule with respect to the number of anaesthetists, surgeons and nurses and to their unavailability periods.

3.7.16 Assignment [32 CONS]

- ALL_BALANCE,
- ASSIGN_AND_COUNTS,
- ASSIGN_AND_NVALUES,
- BALANCE,
- BALANCE_INTERVAL,
- BALANCE_MODULO,
- BALANCE_PARTITION,
- BIN_PACKING,
- BIN_PACKING_CAPA,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST,
- GLOBAL_CARDINALITY,
- GLOBAL_CARDINALITY_LOW_UP,
- GLOBAL_CARDINALITY_WITH_COSTS,
- INCREASING_GLOBAL_CARDINALITY,
- INDEXED_SUM,
- INTERVAL_AND_COUNT,
- INTERVAL_AND_SUM,
- K_ALLDIFFERENT,
- MAX_NVALUE,
- MIN_NVALUE,
- MIN_SIZE_SET_OF_CONSECUTIVE_VAR,
- MINIMUM_WEIGHT_ALLDIFFERENT,
- OPEN_GLOBAL_CARDINALITY,
- OPEN_GLOBAL_CARDINALITY_LOW_UP,
- ORDERED_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP,
- SUM_OF_WEIGHTS_OF_DISTINCT_VALUES,
- SYMMETRIC_CARDINALITY,
- SYMMETRIC_GCC,
- WEIGHTED_PARTIAL_ALLDIFF.

A constraint related to assignment problems (i.e., K_ALLDIFFERENT), or a constraint putting a restriction on all items that are assigned to the same equivalence class or on all equivalence classes that are actually used. Usually an equivalence class corresponds to a single value (see, e.g., the BALANCE, BIN_PACKING, GLOBAL_CARDINALITY, and SUM_OF_WEIGHTS_OF_DISTINCT_VALUES constraints), to an interval of consecutive values (see, e.g., the BALANCE_INTERVAL, INTERVAL_AND_COUNT, and INTERVAL_AND_SUM constraints) or to all values that are congruent modulo a given number (see, e.g., the BALANCE_MODULO constraint). The restriction on all items that are assigned to the same equivalence class can be, for example, a constraint on the number of items (see, e.g., the CARDINALITY_ATLEAST, CARDINALITY_ATMOST, GLOBAL_CARDINALITY, and GLOBAL_CARDINALITY_LOW_UP constraints) or a constraint on the sum of a specific attribute (see, e.g., the BIN_PACKING, and INTERVAL_AND_SUM constraints).

3.7.17 Assignment dimension **[12 CONS]**

- ASSIGN_AND_COUNTS (attribute `bin` of `ITEMS` collection),
- ASSIGN_AND_NVALUES (attribute `bin` of `ITEMS` collection),
- BIN_PACKING (attribute `bin` of `ITEMS` collection),
- BIN_PACKING_CAPA (attribute `bin` of `ITEMS` collection),
- CALENDAR (attribute `machine` of `INSTANTS` collection),
- COLOURED_CUMULATIVES (attribute `machine` of `TASKS` collection),
- CUMULATIVES (attribute `machine` of `TASKS` collection),
- DIFFN (attribute `ori` of `ORTHOTOPE` collection for which $\mathtt{siz} = 1$),
- GEOST (attribute `x` of `OBJECTS` collection for which $\mathtt{l} = 1$),
- GEOST_TIME (attribute `x` of `OBJECTS` collection for which $\mathtt{l} = 1$),
- INTERVAL_AND_COUNT (attribute `origin` of `TASKS` collection),
- INTERVAL_AND_SUM (attribute `origin` of `TASKS` collection).

A constraint for handling placement problems involving orthotopes, where one of the dimensions of the placement space is so called an assignment dimension (i.e., one of the attributes of a collection passed as argument indicates the assignment dimension — the attribute is shown in parenthesis for each constraint). In order to illustrate the notion of *assignment dimension* let us first introduce three typical examples described in Figure 3.3:

- Part (A) of Figure 3.3 considers a scheduling problem where we have both to assign a task to a machine and to fix its start to a time-point, in such a way that two tasks that overlap in time are not assigned to the same machine. In this context the different potential machines where tasks can be assigned is called an assignment dimension. This problem can be directly modelled by a CUMULATIVES, a DIFFN or a GEOST constraint. The corresponding three ground instances encoding the example are (attributes related to the assignment dimension are shown in bold and red):

- CUMULATIVES(
  $\langle$machine $-$ **1** origin $-$ 2 duration $-$ 2 end $-$ 4 height $-$ 1,
  machine $-$ **3** origin $-$ 4 duration $-$ 3 end $-$ 7 height $-$ 1,
  machine $-$ **1** origin $-$ 7 duration $-$ 1 end $-$ 8 height $-$ 1$\rangle$,
  $\langle$id $-$ **1** capacity $-$ **1**,
  id $-$ **2** capacity $-$ **1**,
  id $-$ **3** capacity $-$ **1**$\rangle$)
- DIFFN(
  $\langle$orth $-$ $\langle$ori $-$ 2 siz $-$ 2 end $-$ 4, ori $-$ **1** siz $-$ **1** end $-$ **2**$\rangle$,
  orth $-$ $\langle$ori $-$ 4 siz $-$ 3 end $-$ 7, ori $-$ **3** siz $-$ **1** end $-$ **4**$\rangle$,
  orth $-$ $\langle$ori $-$ 7 siz $-$ 1 end $-$ 8, ori $-$ **1** siz $-$ **1** end $-$ **2**$\rangle\rangle$)
- GEOST(2, $\langle$oid $-$ 1 sid $-$ 1 x $-$ $\langle 2, \mathbf{1}\rangle$,
  oid $-$ 2 sid $-$ 2 x $-$ $\langle 4, \mathbf{3}\rangle$,
  oid $-$ 3 sid $-$ 3 x $-$ $\langle 7, \mathbf{1}\rangle\rangle$
  $\langle$sid $-$ 1 t $-$ $\langle 0, \mathbf{0}\rangle$ l $-$ $\langle 2, \mathbf{1}\rangle$,
  sid $-$ 2 t $-$ $\langle 0, \mathbf{0}\rangle$ l $-$ $\langle 3, \mathbf{1}\rangle$,
  sid $-$ 3 t $-$ $\langle 0, \mathbf{0}\rangle$ l $-$ $\langle 1, \mathbf{1}\rangle\rangle$)

- Part (B) of Figure 3.3 considers a placement problem where we have both to assign a rectangle to a rectangular piece and to locate it within the selected rectangular piece. In this context the different potential rectangular pieces where rectangles can be placed is also called an assignment dimension. Note that in such placement problems the size of an object in an assignment dimension is always equal to one. This problem can be directly modelled by a DIFFN or a GEOST constraint. The corresponding two ground instances encoding the example are (attributes related to the assignment dimension are shown in bold and red):

  - DIFFN($\langle$orth $-$ $\langle$**ori $-$ 2 siz $-$ 1 end $-$ 3**,
    ori $-$ 2 siz $-$ 2 end $-$ 4,
    ori $-$ 2 siz $-$ 2 end $-$ 4$\rangle$,
    orth $-$ $\langle$**ori $-$ 1 siz $-$ 1 end $-$ 2**,
    ori $-$ 3 siz $-$ 3 end $-$ 6,
    ori $-$ 1 siz $-$ 2 end $-$ 3$\rangle$,
    orth $-$ $\langle$**ori $-$ 2 siz $-$ 1 end $-$ 3**,
    ori $-$ 6 siz $-$ 1 end $-$ 7,
    ori $-$ 1 siz $-$ 3 end $-$ 4$\rangle\rangle$)
  - GEOST(3, $\langle$oid $-$ 1 sid $-$ 1 x $-$ $\langle \mathbf{2}, 2, 2\rangle$,
    oid $-$ 2 sid $-$ 2 x $-$ $\langle \mathbf{1}, 3, 1\rangle$,

$$
\begin{array}{l}
\quad\mathtt{oid}-3\ \mathtt{sid}-3\ \mathtt{x}-\langle\mathbf{2},6,1\rangle\rangle\\
\langle\mathtt{sid}-1\ \mathtt{t}-\langle\mathbf{0},0,0\rangle\ \mathtt{l}-\langle\mathbf{1},2,2\rangle,\\
\quad\mathtt{sid}-2\ \mathtt{t}-\langle\mathbf{0},0,0\rangle\ \mathtt{l}-\langle\mathbf{1},3,2\rangle,\\
\quad\mathtt{sid}-3\ \mathtt{t}-\langle\mathbf{0},0,0\rangle\ \mathtt{l}-\langle\mathbf{1},1,3\rangle\rangle)
\end{array}
$$

- Part (C) of Figure 3.3 considers a placement problem where we have both to assign a box to a container and to place it within the selected container. In this context the different potential containers where boxes can be packed is also called an assignment dimension. Note that in such placement problems the size of an object in an assignment dimension is always equal to one. This problem can be directly modelled by a DIFFN or a GEOST constraint. The corresponding two ground instances encoding the example are (attributes related to the assignment dimension are shown in bold and red):

  - DIFFN
$$
\begin{array}{l}
(\langle\mathtt{orth}-\langle\mathbf{ori}-\mathbf{1}\ \mathbf{siz}-\mathbf{1}\ \mathbf{end}-\mathbf{2},\\
\qquad\qquad\mathtt{ori}-1\ \mathtt{siz}-1\ \mathtt{end}-2,\\
\qquad\qquad\mathtt{ori}-1\ \mathtt{siz}-2\ \mathtt{end}-3,\\
\qquad\qquad\mathtt{ori}-1\ \mathtt{siz}-1\ \mathtt{end}-2\rangle,\\
\quad\mathtt{orth}-\langle\mathbf{ori}-\mathbf{1}\ \mathbf{siz}-\mathbf{1}\ \mathbf{end}-\mathbf{2},\\
\qquad\qquad\mathtt{ori}-1\ \mathtt{siz}-1\ \mathtt{end}-2,\\
\qquad\qquad\mathtt{ori}-1\ \mathtt{siz}-1\ \mathtt{end}-2,\\
\qquad\qquad\mathtt{ori}-2\ \mathtt{siz}-1\ \mathtt{end}-3\rangle,\\
\quad\mathtt{orth}-\langle\mathbf{ori}-\mathbf{2}\ \mathbf{siz}-\mathbf{1}\ \mathbf{end}-\mathbf{3},\\
\qquad\qquad\mathtt{ori}-1\ \mathtt{siz}-2\ \mathtt{end}-3,\\
\qquad\qquad\mathtt{ori}-1\ \mathtt{siz}-2\ \mathtt{end}-3,\\
\qquad\qquad\mathtt{ori}-1\ \mathtt{siz}-1\ \mathtt{end}-2\rangle\rangle)
\end{array}
$$
  - GEOST
$$
\begin{array}{l}
(4,\langle\mathtt{oid}-1\ \mathtt{sid}-1\ \mathtt{x}-\langle\mathbf{1},1,1,1\rangle,\\
\quad\mathtt{oid}-2\ \mathtt{sid}-2\ \mathtt{x}-\langle\mathbf{1},1,1,2\rangle,\\
\quad\mathtt{oid}-3\ \mathtt{sid}-3\ \mathtt{x}-\langle\mathbf{2},1,1,1\rangle\rangle\\
\langle\mathtt{sid}-1\ \mathtt{t}-\langle\mathbf{0},0,0,0\rangle\ \mathtt{l}-\langle\mathbf{1},1,2,1\rangle,\\
\quad\mathtt{sid}-2\ \mathtt{t}-\langle\mathbf{0},0,0,0\rangle\ \mathtt{l}-\langle\mathbf{1},1,1,1\rangle,\\
\quad\mathtt{sid}-3\ \mathtt{t}-\langle\mathbf{0},0,0,0\rangle\ \mathtt{l}-\langle\mathbf{1},2,2,1\rangle\rangle)
\end{array}
$$

In summary, within the context of placement problems that use a constraint like DIFFN or GEOST, the coordinate of an object in the assignment dimension corresponds to the resource to which the object is assigned. Note that *the size of an object in the assignment dimension is always set to* 1. This stems from the fact that an object is assigned to a single resource.

Using constraints like COLOURED_CUMULATIVES, CUMULATIVES, DIFFN, GEOST or GEOST_TIME allows to model directly with a single global constraint such problems without knowing in advance to which machine, to which rectangular piece, to which container, a task, a rectangle, a box will be assigned. For each object the potential values of its assignment variable provide the machines, the rectangular pieces, the containers to which the object can possibly be assigned. Note that this allows to avoid 0-1 variables for modelling such problems.

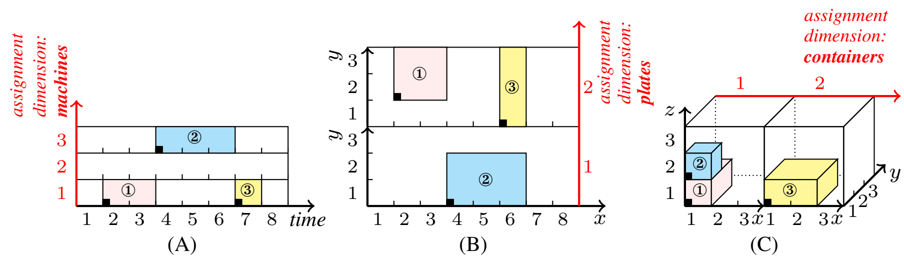


Figure 3.3: Three illustrations of the notion of *assignment dimension* where the assignment dimension is shown in red

Within constraints like INTERVAL_AND_COUNT or INTERVAL_AND_SUM the concept of assignment dimension is extended from the fact that *a variable is assigned a value* to the fact that *a variable is assigned an interval (i.e., a value in an interval)*.

3.7.18 Assignment to the same set of values [9 CONS]

- BIN_PACKING,
- BIN_PACKING_CAPA,
- COLOURED_CUMULATIVES,
- CUMULATIVES,
- DIFFN,
- ELEM,
- ELEMENT,
- GEOST,
- GEOST_TIME.

Given several mutually disjoint finite sets of values $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ $(m > 1)$ such that $S_1 \cup S_2 \cup \cdots \cup S_m = \{1, 2, \ldots, p\}$, as well as a set of variables $V_1, V_2, \ldots, V_n$, the *assignment to the same set of values subproblem* consists of assigning all variables $V_1, V_2, \ldots, V_n$ values that belong to the same set $\mathcal{S}_i$ $(1 \leq i \leq m)$. As we will see later on, this subproblem arises naturally in many resource assignment problems where an additional constraint between variables $V_1, V_2, \ldots, V_n$ also has to hold. The subproblem can be modelled as a conjunction of ELEMENT constraints of the form:

ELEMENT$(V_1, \langle set_of_val_1, set_of_val_2, \ldots, set_of_val_p\rangle, SET_INDEX) \wedge$
ELEMENT$(V_2, \langle set_of_val_1, set_of_val_2, \ldots, set_of_val_p\rangle, SET_INDEX) \wedge$
$\ldots$
ELEMENT$(V_n, \langle set_of_val_1, set_of_val_2, \ldots, set_of_val_p\rangle, SET_INDEX)$,
where $set\_of\_val_i = j$ if and only if $i \in \mathcal{S}_j$ (i.e., $set\_of\_val_i$ corresponds to the index

of the set that contains value i). The k-th ELEMENT constraint expresses that variable V_k is assigned a value in set $\mathcal{S}_{SET\_INDEX}$. Since all ELEMENT constraints share the same third argument this forces all variables $V_1, V_2, \ldots, V_n$ to be assigned a value within the same set. Note that this conjunction of ELEMENT constraints corresponds to a Berge-acyclic constraint network. Consequently, one can achieve arc-consistency on this subproblem provided that arc-consistency is enforced on each ELEMENT constraint.

As an example, consider the four sets of values $\mathcal{S}_1 = \{3, 4, 8\}$, $\mathcal{S}_2 = \{1, 5\}$, $\mathcal{S}_3 = \{6, 7\}$, and $\mathcal{S}_4 = \{2, 9\}$, as well as four variables w, x, y and z that all must be assigned values that belong to the same set $\mathcal{S}_s$ $(1 \le s \le 4)$. This leads to the following conjunction of ELEMENT constraints:

ELEMENT$(w, \langle 2, 4, 1, 1, 2, 3, 3, 1, 4\rangle, s) \wedge$
ELEMENT$(x, \langle 2, 4, 1, 1, 2, 3, 3, 1, 4\rangle, s) \wedge$
ELEMENT$(y, \langle 2, 4, 1, 1, 2, 3, 3, 1, 4\rangle, s) \wedge$
ELEMENT$(z, \langle 2, 4, 1, 1, 2, 3, 3, 1, 4\rangle, s)$.

The first entry of the table $\langle 2, 4, 1, 1, 2, 3, 3, 1, 4\rangle$ is set to 2 since value 1 belongs to $\mathcal{S}_2$. Similarly, the second entry of the table is set of 4 since value 2 belongs to $\mathcal{S}_4$. The same logic is used for building up the other entries of the table.

A generalisation of this subproblem consists in lifting the restriction that the sets of values $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ are mutually disjoint. The only change to adapt the previous model is to replace within each ELEMENT constraint each value val_i $(1 \le i \le p)$ by a *value variable* Val_i (i.e., each value of a value variable represents a set containing i), where $j \in dom(Val_i)$ if and only if $i \in \mathcal{S}_j$. Distinct ELEMENT constraints will get distinct value variables. As an example, consider the previous four sets of values where we add value 2 to $\mathcal{S}_1$ and value 5 to $\mathcal{S}_3$. We now have the sets $\mathcal{S}_1 = \{2, 3, 4, 8\}$, $\mathcal{S}_2 = \{1, 5\}$, $\mathcal{S}_3 = \{5, 6, 7\}$, and $\mathcal{S}_4 = \{2, 9\}$ where value 2 occurs both in $\mathcal{S}_1$ and $\mathcal{S}_4$, and value 5 appears both in $\mathcal{S}_2$ and $\mathcal{S}_3$. This leads to the following conjunction of constraints:

IN$(a_1, \langle 1, 4\rangle)$ $\wedge$ IN$(b_1, \langle 2, 3\rangle)$ $\wedge$ ELEMENT$(w, \langle 2, a_1, 1, 1, b_1, 3, 3, 1, 4\rangle, s) \wedge$
IN$(a_2, \langle 1, 4\rangle)$ $\wedge$ IN$(b_2, \langle 2, 3\rangle)$ $\wedge$ ELEMENT$(x, \langle 2, a_2, 1, 1, b_2, 3, 3, 1, 4\rangle, s) \wedge$
IN$(a_3, \langle 1, 4\rangle)$ $\wedge$ IN$(b_3, \langle 2, 3\rangle)$ $\wedge$ ELEMENT$(y, \langle 2, a_3, 1, 1, b_3, 3, 3, 1, 4\rangle, s) \wedge$
IN$(a_4, \langle 1, 4\rangle)$ $\wedge$ IN$(b_4, \langle 2, 3\rangle)$ $\wedge$ ELEMENT$(z, \langle 2, a_4, 1, 1, b_4, 3, 3, 1, 4\rangle, s)$.

The domains of the variables a_i $(1 \le i \le 4)$ associated with the second entry of the table[7] of the ELEMENT constraints is set to 1 and 4 since value 2 belongs to $\mathcal{S}_1$ and to $\mathcal{S}_4$. Similarly, the domain of variables b_i $(1 \le i \le 4)$ associated with the fifth entry is set to 2 and 3 since value 5 belongs to $\mathcal{S}_2$ and $\mathcal{S}_3$. Note that, since variables a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4 are distinct, the corresponding constraint network is still Berge-acyclic. We now provide an alternative model where the i^{th} entry of the table of the k^{th} $(1 \le k \le n)$ ELEMENT constraint corresponds to a variable S_{ki} for which the initial domain is the set of values that belong to $\mathcal{S}_i$ $(1 \le i \le m)$. We have a conjunction of ELEMENT constraints of the form:

ELEMENT$(SET_INDEX, \langle S_{11}, S_{12}, \ldots, S_{1m}\rangle, V_1) \wedge$
ELEMENT$(SET_INDEX, \langle S_{21}, S_{22}, \ldots, S_{2m}\rangle, V_2) \wedge$
$\ldots$

[7]The *table* corresponds to the second argument of the ELEMENT constraint.

ELEMENT$(SET_INDEX, \langle S_{n1}, S_{n2}, \ldots, S_{nm}\rangle, V_n)$,
where SET_INDEX is a variable ranging from 1 to m designating the selected set. This model perhaps seems more natural. However unlike the first model, when the sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ are mutually disjoint, it enforces using variables instead of integers in the table of each ELEMENT constraint. Like the first model, it is Berge-acyclic.

Now that we have presented two dual models for the assignment to the same set of values subproblem, we introduce the *resource assignment with groups* pattern, which uses several instances of the subproblem. We consider a set of tasks $t_1, t_2, \ldots, t_q$ ($q \geq 1$) tasks, where each task t_i ($1 \leq i \leq q$) is decomposed into s_i subtasks t_{ij} ($1 \leq j \leq s_i$). All subtasks that belong to one and the same task should be assigned the same group, where groups are defined by the finite sets of values $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ ($m > 1$) introduced early on. For this purpose an *assignment variable* and a *group variable* are respectively associated with each subtask and each task. In addition, we also have a resource constraint involving all subtasks. This resource constraint has an assignment dimension corresponding to the different resources where subtasks can potentially be assigned. To each resource corresponds a value of $S_1 \cup S_2 \cup \cdots \cup S_m = \{1, 2, \ldots, p\}$. Depending on the kind of resource constraint we have (e.g., BIN_PACKING, CUMULATIVES, DIFFN, GEOST), each subtask has additional attributes that characterise it. For example, if we have a BIN_PACKING constraint then, in addition to the assignment dimension that corresponds to the bin where a subtask will be assigned, we also have a weight attribute that describes how much space a subtask uses in a bin. Then the BIN_PACKING constraint expresses that the total weight of the subtasks in each bin does not exceed a given fixed capacity.

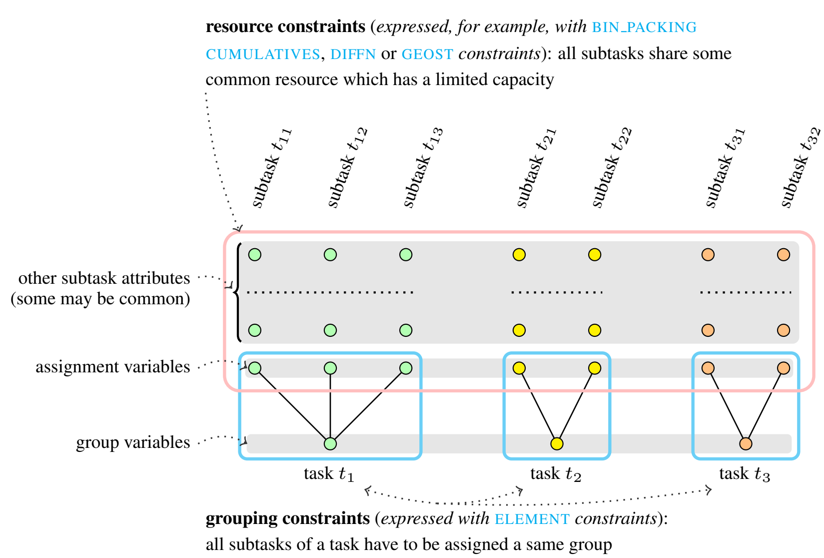


Figure 3.4: Illustration of the constraint network associated with the *resource assignment with groups* pattern

Figure 3.4 illustrates the constraint network associated with the *resource assign-*

ment with groups pattern. Lower circles represent the group variables associated with the different tasks (three tasks in the example), while all the other circles represent the attributes of the different subtasks (i.e., vertically aligned circles correspond to the attributes of a given subtask). All circles that are associated with the same task are coloured with the same colour. As said before, each subtask has an attribute that gives the resource to which the resource will be assigned (called assignment variables in Figure 3.4) and other attributes that depend of the resource constraint we are considering (called other subtask attributes in the Figure). Each blue rounded box corresponds to a group constraint which enforces all subtasks of a given task to be assigned the same group (i.e., within this blue box, each line segment represents an ELEMENT constraint of the assignment to the same set of values subproblem). Finally, the pink rounded box represents the resource constraint that involves all subtasks.

Before illustrating the resource assignment with groups pattern on a particular resource constraint, we first point out a potential weakness that is inherent to this constraint network, no matter what kind of resource constraint we use. When pruning the assignment variables, the resource constraint will ignore the groups (since the resource constraint is not aware of the ELEMENT constraints) and will therefore miss some filtering. Consequently one may complete the constraint network by some global necessary conditions. When fixing variables it may be a good idea to fix all variables that are attached to one task before considering the next task. While fixing the variables of a task one may first assign its group variable, and second fix the variables of its subtasks; again we may prefer to fix all variables of a subtask before considering the next subtask. The idea behind this heuristic is to try to avoid the creation of infeasible subproblems during search.

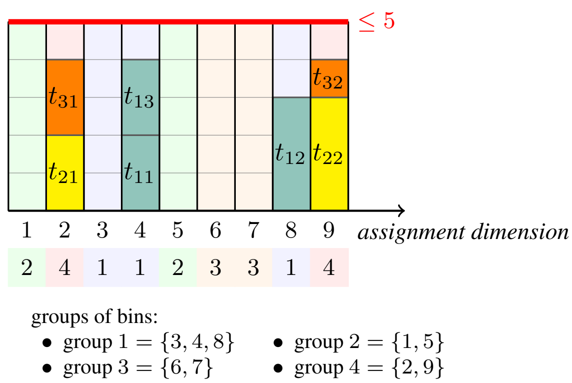


Figure 3.5: Illustration of the resource assignment with groups pattern in the context of a BIN_PACKING resource constraint (subtasks of the same colour are assigned to the same group of bins)

Figure 3.5 illustrates the resource assignment with groups pattern when the resource constraint corresponds to a BIN_PACKING constraint. As in Figure 3.4, we have three tasks t_1, t_2 and t_3 such that:

- Three subtasks t_{11}, t_{12} and t_{13} are associated with task t_1. They have a respective weight of 2, 3 and 2 and are coloured in green in Figure 3.5.

- Two subtasks t_{21} and t_{22} of respective weight 2 and 3 are associated with task t_2. They are coloured in yellow.
- Two subtasks t_{31} and t_{32} of respective weight 2 and 1 are associated with task t_3. They are coloured in orange.

We consider 9 bins that are partitioned into four groups of bins $\mathcal{S}_1 = \{3, 4, 8\}$ (coloured in light blue in Figure 3.5), $\mathcal{S}_2 = \{1, 5\}$ (coloured in light green), $\mathcal{S}_3 = \{6, 7\}$ (coloured in light brown), and $\mathcal{S}_4 = \{2, 9\}$ (coloured in light violet), and enforce that all subtasks that are associated with the same task are assigned the same group of bins. In addition, the sum of the weights of the subtasks that are assigned the same bin should not exceed the capacity of the bins, 5 in our example. Within the solution depicted by Figure 3.5, all constraints are satisfied since:

1. For each task, all its subtasks are assigned the same group of bins (i.e., all subtasks that have the same colour are assigned bins with the same colour).
2. The capacity constraint of each bin is respected (i.e., the overall capacity of five is never exceeded).

The conjunction of constraints corresponding to this solution is:

$\text{ELEMENT}(4, \langle 2,4,1,1,2,3,3,1,4\rangle, 1) \wedge$
$\text{ELEMENT}(8, \langle 2,4,1,1,2,3,3,1,4\rangle, 1) \wedge$
$\text{ELEMENT}(4, \langle 2,4,1,1,2,3,3,1,4\rangle, 1) \wedge$
$\text{ELEMENT}(2, \langle 2,4,1,1,2,3,3,1,4\rangle, 4) \wedge$
$\text{ELEMENT}(9, \langle 2,4,1,1,2,3,3,1,4\rangle, 4) \wedge$
$\text{ELEMENT}(2, \langle 2,4,1,1,2,3,3,1,4\rangle, 4) \wedge$
$\text{ELEMENT}(9, \langle 2,4,1,1,2,3,3,1,4\rangle, 4) \wedge$
$\text{BIN\_PACKING}(5, \langle \texttt{bin}-4\ \texttt{weight}-2,\ \texttt{bin}-8\ \texttt{weight}-3,\ \texttt{bin}-4\ \texttt{weight}-2,$
$\texttt{bin}-2\ \texttt{weight}-2,\ \texttt{bin}-9\ \texttt{weight}-3,$
$\texttt{bin}-2\ \texttt{weight}-2,\ \texttt{bin}-9\ \texttt{weight}-1\rangle).$

For each subtask we have one ELEMENT constraint expressing that all subtasks of a given task are assigned the same group of bins. Finally we have one BIN_PACKING constraint expressing the capacity condition.

We now quote two concrete examples of the resource assignment with groups pattern:

- Given, (1) a set of jobs where each job is decomposed into a set of tasks, each of them requiring an amount of memory for its execution, as well as (2) a set of potential machines, each of them having a given available memory, organised into clusters, the problem is to:
  - Assign all tasks to machines in such a way that tasks from the same job are assigned the same cluster.
  - Fulfil the available memory constraint of each machine (i.e., the sum of the required memory of all tasks that are assigned a given machine does not exceed the machine available memory).

This concrete problem corresponds to the example presented in Figure 3.5.

- Given, (1) a set of maintenance activities where each maintenance activity is decomposed into a set of subactivities, each of them requiring a specific skill and a given duration, as well as (2) a set of technicians, each of them having its own home base location and its own working time window, the problem is to:

  – Assign all maintenance subactivities to technicians in such a way that subactivities from the same activity are assigned technicians that have the same home base location (i.e., each subactivity should be assigned a single technician).

  – Fulfil both the working time window of each technician, and the fact that subactivities that are assigned the same technician should not overlap (i.e., subactivities must be assigned a starting time and preemption is not allowed).

  In this problem we replace the BIN_PACKING constraint by a CUMULATIVES(TASKS, MACHINES, $\leq$) constraint. To each item of the TASKS collection corresponds a subactivity, such that:

  – Its `machine` attribute designates the potential technicians that can take care of this subactivity.

  – Its `origin` attribute corresponds to the timepoint where the subactivity will actually start.

  – Its `duration` attribute is set to the duration of the corresponding subactivity.

  – Its `end` attribute is equal to `origin` + `duration`.

  – Its `height` attribute is set to one.

  In addition to the subactivities, we also introduce for each technician two fixed dummy tasks for preventing assigning subactivities outside its time window. To each item of the MACHINES collection corresponds a technician, such that:

  – Its `id` attribute is a fixed integer that uniquely identifies the technician.

  – Its `capacity` attribute is set to one since it cannot perform more than one subactivity at any timepoint.

3.7.19 At least [3 CONS]

- ATLEAST,
- CARDINALITY_ATLEAST,
- OPEN_ATLEAST.

A constraint enforcing that one or several values occur a minimum number of time within a given collection of domain variables.

3.7.20 At most [5 CONS]

- ATMOST,
- CARDINALITY_ATMOST,
- CARDINALITY_ATMOST_PARTITION,
- MULTI_INTER_DISTANCE,
- OPEN_ATMOST.

A constraint enforcing that one or several values occur a maximum number of time within a given collection of domain variables.

3.7.21 Automaton [147 CONS]

- ALL_EQUAL_EXCEPT_0,
- ALL_EQUAL_PEAK,
- ALL_EQUAL_PEAK_MAX,
- ALL_EQUAL_VALLEY,
- ALL_EQUAL_VALLEY_MIN,
- ALLDIFFERENT,
- ALLDIFFERENT_EXCEPT_0,
- ALLDIFFERENT_INTERVAL,
- ALLDIFFERENT_MODULO,
- ALLDIFFERENT_ON_INTERSECTION,
- ALLDIFFERENT_SAME_VALUE,
- AMONG,
- AMONG_DIFF_0,
- AMONG_INTERVAL,
- AMONG_LOW_UP,
- AMONG_MODULO,

- AND,
- ARITH,
- ARITH_OR,
- ARITH_SLIDING,
- ASSIGN_AND_COUNTS,
- ATLEAST,
- ATMOST,
- BALANCE,
- BALANCE_INTERVAL,
- BALANCE_MODULO,
- BETWEEN_MIN_MAX,
- BIG_PEAK,
- BIG_VALLEY,
- BIN_PACKING,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST,
- CHANGE,
- CHANGE_CONTINUITY,
- CHANGE_PAIR,
- CHANGE_VECTORS,
- CIRCULAR_CHANGE,
- CLAUSE_AND,
- CLAUSE_OR,
- COND_LEX_COST,
- COND_LEX_GREATER,
- COND_LEX_GREATEREQ,
- COND_LEX_LESS,
- COND_LEX_LESSEQ,
- CONSECUTIVE_GROUPS_OF_ONES,
- COUNT,
- COUNTS,
- CUMULATIVE,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- DECREASING,
- DECREASING_PEAK,
- DECREASING_VALLEY,
- DEEPEST_VALLEY,
- DIFFER_FROM_AT_LEAST_K_POS,
- DISJOINT,
- DISTANCE_CHANGE,
- DOMAIN_CONSTRAINT,
- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENTN,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_MATRIX,
- ELEMENT_SPARSE,
- EQUIVALENT,
- EXACTLY,
- FIRST_VALUE_DIFF_0,
- FULL_GROUP,
- GLOBAL_CARDINALITY,
- GLOBAL_CONTIGUITY,
- GROUP,
- GROUP_SKIP_ISOLATED_ITEM,
- HIGHEST_PEAK,
- IMPLY,
- IN,
- IN_INTERVAL,
- IN_SAME_PARTITION,
- INCREASING,
- INCREASING_GLOBAL_CARDINALITY,
- INCREASING_NVALUE,
- INCREASING_PEAK,
- INCREASING_VALLEY,
- INFLEXION,
- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- INTERVAL_AND_COUNT,
- INTERVAL_AND_SUM,
- INVERSE,
- ITH_POS_DIFFERENT_FROM_0,
- LENGTH_FIRST_SEQUENCE,
- LENGTH_LAST_SEQUENCE,
- LEX_BETWEEN,
- LEX_DIFFERENT,
- LEX_EQUAL,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LONGEST_CHANGE,

- LONGEST_DECREASING_SEQUENCE,
- LONGEST_INCREASING_SEQUENCE,
- MAX_DECREASING_SLOPE,
- MAX_INCREASING_SLOPE,
- MAX_NVALUE,
- MAXIMUM,
- MIN_DECREASING_SLOPE,
- MIN_DIST_BETWEEN_INFLEXION,
- MIN_INCREASING_SLOPE,
- MIN_N,
- MIN_NVALUE,
- MIN_SIZE_FULL_ZERO_STRETCH,
- MIN_SURF_PEAK,
- MIN_WIDTH_PEAK,
- MIN_WIDTH_PLATEAU,
- MIN_WIDTH_VALLEY,
- MINIMUM,
- MINIMUM_EXCEPT_0,
- MINIMUM_GREATER_THAN,
- NAND,
- NEXT_ELEMENT,
- NO_PEAK,
- NO_VALLEY,
- NOR,
- NOT_ALL_EQUAL,
- NOT_IN,
- NVALUE,
- OPEN_MAXIMUM,
- OPEN_MINIMUM,
- OR,
- PATTERN,
- PEAK,
- SAME,
- SEQUENCE_FOLDING,
- SLIDING_CARD_SKIP0,
- SMOOTH,
- STAGE_ELEMENT,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION,
- STRICTLY_DECREASING,
- STRICTLY_INCREASING,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_DO_NOT_OVERLAP,
- USED_BY,
- VALLEY,
- XOR.

A constraint for which the catalogue provides a deterministic automaton for the ground case. This automaton can usually be used for deriving mechanically a filtering algorithm for the general case. We have the following three types of deterministic automata:

- Deterministic automata without counters and without array of counters,
- Deterministic automata with counters but without array of counters,
- Deterministic automata with array of counters and possibly with counters.

Figure 3.6 shows three automata respectively associated with the GLOBAL_CONTIGUITY, the EXACTLY and the ALLDIFFERENT constraints. These automata correspond to the three types we described above.

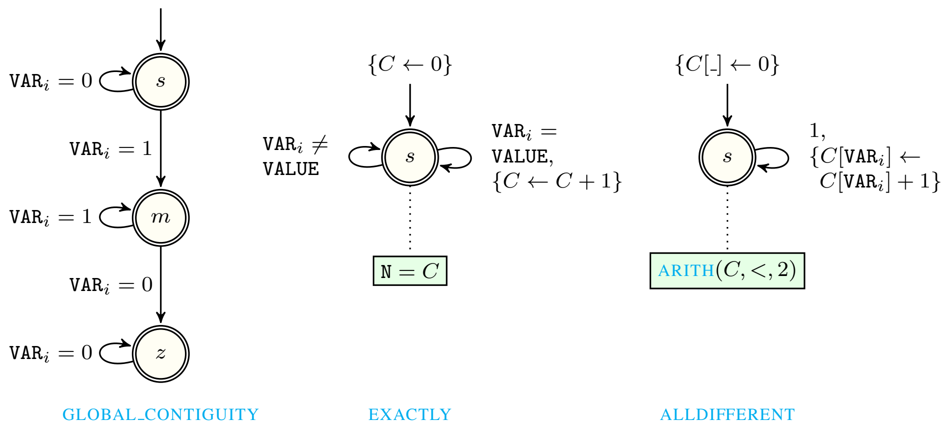


Figure 3.6: Examples of automata where an initial state is indicated by an arc coming from no state and an accepting state is denoted graphically by a double circle

3.7.22 Automaton with array of counters [25 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_EXCEPT_0,
- ALLDIFFERENT_INTERVAL,
- ALLDIFFERENT_MODULO,
- ALLDIFFERENT_ON_INTERSECTION,
- ALLDIFFERENT_SAME_VALUE,
- ASSIGN_AND_COUNTS,
- BALANCE,
- BALANCE_INTERVAL,
- BALANCE_MODULO,
- BIN_PACKING,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST,
- CUMULATIVE,
- DISJOINT,
- GLOBAL_CARDINALITY,
- INTERVAL_AND_COUNT,
- INTERVAL_AND_SUM,
- INVERSE,
- MAX_NVALUE,
- MIN_N,
- MIN_NVALUE,
- NVALUE,
- SAME,
- USED_BY.

A constraint for which the catalogue provides a deterministic automaton with array of counters and possibly with counters.

3.7.23 Automaton with counters [59 CONS]

- ALL_EQUAL_EXCEPT_0,
- ALL_EQUAL_PEAK,
- ALL_EQUAL_PEAK_MAX,
- ALL_EQUAL_VALLEY,
- ALL_EQUAL_VALLEY_MIN,
- AMONG,
- AMONG_DIFF_0,
- AMONG_INTERVAL,
- AMONG_LOW_UP,
- AMONG_MODULO,
- ARITH_SLIDING,
- ATLEAST,
- ATMOST,
- BIG_PEAK,
- BIG_VALLEY,
- CHANGE,
- CHANGE_CONTINUITY,
- CHANGE_PAIR,
- CHANGE_VECTORS,
- CIRCULAR_CHANGE,
- COUNT,
- COUNTS,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- DECREASING_PEAK,
- DECREASING_VALLEY,
- DEEPEST_VALLEY,
- DIFFER_FROM_AT_LEAST_K_POS,
- DISTANCE_CHANGE,
- EQUILIBRIUM,
- EXACTLY,
- FIRST_VALUE_DIFF_0,
- FULL_GROUP,
- GROUP,
- GROUP_SKIP_ISOLATED_ITEM,
- HIGHEST_PEAK,
- INCREASING_PEAK,
- INCREASING_VALLEY,
- INFLEXION,
- ITH_POS_DIFFERENT_FROM_0,
- LENGTH_FIRST_SEQUENCE,
- LENGTH_LAST_SEQUENCE,
- LONGEST_CHANGE,
- LONGEST_DECREASING_SEQUENCE,
- LONGEST_INCREASING_SEQUENCE,
- MAX_DECREASING_SLOPE,
- MAX_INCREASING_SLOPE,
- MIN_DECREASING_SLOPE,
- MIN_DIST_BETWEEN_INFLEXION,
- MIN_INCREASING_SLOPE,
- MIN_SIZE_FULL_ZERO_STRETCH,
- MIN_SURF_PEAK,
- MIN_WIDTH_PEAK,
- MIN_WIDTH_PLATEAU,
- MIN_WIDTH_VALLEY,
- PEAK,
- SLIDING_CARD_SKIP0,
- SMOOTH,
- VALLEY.

A constraint for which the catalogue provides a deterministic automaton with counters but without array of counters.

3.7.24 Automaton with same input symbol [39 CONS]

- ALL_EQUAL_PEAK,
- ALL_EQUAL_PEAK_MAX,
- ALL_EQUAL_VALLEY,
- ALL_EQUAL_VALLEY_MIN,
- CHANGE_CONTINUITY (NB_PERIOD_CHANGE),
- CHANGE_CONTINUITY (NB_PERIOD_CONTINUITY),
- CHANGE_CONTINUITY (MIN_SIZE_CHANGE),
- CHANGE_CONTINUITY (MIN_SIZE_CONTINUITY),
- DECREASING_PEAK,
- DECREASING_VALLEY,
- DEEPEST_VALLEY,
- FULL_GROUP (NGROUP),
- FULL_GROUP (MIN_SIZE),
- FULL_GROUP (MAX_SIZE),
- FULL_GROUP (MIN_DIST),
- FULL_GROUP (MAX_DIST),
- FULL_GROUP (NVAL),
- GLOBAL_CONTIGUITY,
- GROUP (NGROUP),
- GROUP (MIN_SIZE),
- GROUP (MIN_DIST),
- GROUP_SKIP_ISOLATED_ITEM (NGROUP),
- GROUP_SKIP_ISOLATED_ITEM (MIN_SIZE),
- GROUP_SKIP_ISOLATED_ITEM (MAX_SIZE),
- HIGHEST_PEAK,
- INCREASING_PEAK,
- INCREASING_VALLEY,
- INFLEXION,
- LONGEST_DECREASING_SEQUENCE,
- LONGEST_INCREASING_SEQUENCE,
- MIN_DIST_BETWEEN_INFLEXION,
- MIN_SIZE_FULL_ZERO_STRETCH,
- MIN_SURF_PEAK,
- MIN_WIDTH_PEAK,
- MIN_WIDTH_VALLEY,
- NO_PEAK,
- NO_VALLEY,
- PEAK,
- VALLEY.

A constraint for which the catalogue provides an automaton belonging to the following category:

- Symbols of the alphabet are split in two categories: *neutral* ones and *non-neutral* ones.
- *Non-neutral* symbols correspond to symbols occurring on transitions between two distinct states, while *neutral* symbols correspond to all the other symbols of the alphabet.
- Self-loops labelled by a neutral symbol do not modify any counter.
- Ignoring transitions labelled by neutral symbols, every state has its incoming transitions labelled by the same non-neutral symbol.
- Ignoring transitions labelled by neutral symbols, outgoing transitions of a state are not labelled by the symbol associated with its incoming non-loop transitions.

For such automata we define the *semantics of a state* s as the regular expression associated with the language fragment obtained from entering state s to just before leaving state s.

As an example, consider the VALLEY constraint and its automaton depicted by Figure 3.7. The alphabet $\mathcal{A}$ corresponds to the set of symbols $\{<, =, >\}$ from which $<$

STATE SEMANTICS

s	: stationary/increasing mode	$(\{<\mid=\}^*)$
u	: decreasing mode	$(>\{>\mid=\}^*)$

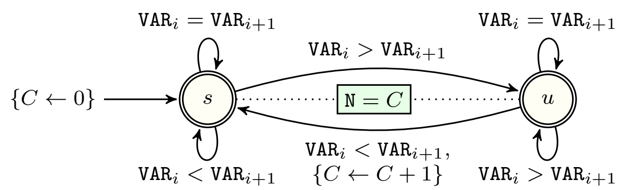


Figure 3.7: Semantics of the states of the automaton of the VALLEY constraint (an accepting state is denoted graphically by a double circle)

and $>$ are non-neutral symbols (i.e., the symbols associated with the transitions between states s and u), and $=$ is a neutral symbol. First there is no counter modification on all self-loops. If we remove the self-loops carrying the neutral symbol $=$ we have that:

- All incoming transitions in state s are labelled by the non-symbol $<$, and all outgoing transitions from state s are not labelled by $<$.
- All incoming transitions in state u are labelled by the non-symbol $>$, and all outgoing transitions from state u are not labelled by $>$.

The corresponding state semantics is given by the upper-leftmost box.

3.7.25 Automaton without counters [60 CONS]

- AND,
- ARITH,
- ARITH_OR,
- BETWEEN_MIN_MAX,
- CLAUSE_AND,
- CLAUSE_OR,
- COND_LEX_COST,
- CONSECUTIVE_GROUPS_OF_ONES,
- DECREASING,
- DOMAIN_CONSTRAINT,
- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_MATRIX,

- ELEMENT_SPARSE,
- ELEMENTN,
- EQUIVALENT,
- GLOBAL_CONTIGUITY,
- IMPLY,
- IN,
- IN_INTERVAL,
- IN_SAME_PARTITION,
- INCREASING,
- INCREASING_GLOBAL_CARDINALITY,
- INCREASING_NVALUE,
- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- LEX_BETWEEN,
- LEX_DIFFERENT,
- LEX_EQUAL,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- MAXIMUM,
- MINIMUM,
- MINIMUM_EXCEPT_0,
- MINIMUM_GREATER_THAN,
- NAND,
- NEXT_ELEMENT,
- NO_PEAK,
- NO_VALLEY,
- NOR,
- NOT_ALL_EQUAL,
- NOT_IN,
- OPEN_MAXIMUM,
- OPEN_MINIMUM,
- OR,
- PATTERN,
- SEQUENCE_FOLDING,
- STAGE_ELEMENT,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION,
- STRICTLY_DECREASING,
- STRICTLY_INCREASING,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_DO_NOT_OVERLAP,
- XOR.

A constraint for which the catalogue provides a deterministic automaton without counters and without array of counters. Note that the filtering algorithm [317] and the reformulation [39] that were initially done in the context of deterministic automata can also be used for non-deterministic automata. All these constraints are also annotated with the keyword reified automaton constraint.

3.7.26 Autoref [1 CONS]

- GLOBAL_CARDINALITY.

A constraint that allows for modelling the *autoref* problem with a single constraint. The *autoref* problem is a generalisation of the problem of finding a magic serie and can be defined in the following way. Given an integer $n > 0$ and an integer $m \geq 0$, the problem is to find a non-empty finite series $S = (s_0, s_1, \ldots, s_n, s_{n+1})$ such that (1) there are s_i occurrences of i in S for each integer i ranging from 0 to n, and (2) $s_{n+1} = m$. This leads to the following model:

$$\text{GLOBAL\_CARDINALITY}\left(\begin{array}{l}\langle\ \mathtt{var}-s_0, \mathtt{var}-s_1, \ldots, \mathtt{var}-s_n, \mathtt{var}-m\ \rangle, \\ \left\langle\begin{array}{ll}\mathtt{val}-0 & \mathtt{noccurrence}-s_0, \\ \mathtt{val}-1 & \mathtt{noccurrence}-s_1, \\ & \vdots \\ \mathtt{val}-n & \mathtt{noccurrence}-s_n\end{array}\right\rangle\end{array}\right)$$

23, 2, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 5 and 23, 3, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 5 are the two unique solutions for $n = 27$ and $m = 5$.

3.7.27 Balanced assignment [8 CONS]

- ALL_BALANCE,
- BALANCE,
- BALANCE_INTERVAL,
- BALANCE_MODULO,
- BALANCE_PARTITION,
- DEVIATION,
- MAXIMUM,
- SPREAD.

A constraint to obtain a balanced assignment over a set of domain variables. Given a set of domain variables $\{x_1, x_2, \ldots, x_n\}$, some classical balance criteria reported in [379] are:

- The *maximum value*, i.e., the maximum value over x_i $(i \in [1, n])$ can be modelled with a MAXIMUM constraint.
- The *maximum deviation*, i.e., the maximum value over $x_i - \frac{\sum_{j\in[1,n]} x_j}{n}$ $(i \in [1, n])$.

- The *total deviation*, i.e., $\sum_{i\in[1,n]}\left|x_i - \frac{\sum_{j\in[1,n]} x_j}{n}\right|$ can be modelled with a DEVIATION constraint [382, 380].
- The *total quadratic deviation*, i.e, $\sum_{i\in[1,n]}\left(x_i - \frac{\sum_{j\in[1,n]} x_j}{n}\right)^2$ can be modelled with a SPREAD constraint [318, 381].

3.7.28 Balanced tree [1 CONS]

- TREE_RANGE.

A constraint that allows for expressing that we want to cover a digraph by one (or more) *balanced tree*. A *balanced tree* is a tree where no leaf is much farther away than a given threshold from the root than any other leaf. The distance between a leaf and the root of a tree is the number of vertices on the path from the root to the leaf.

3.7.29 Berge-acyclic constraint network [40 CONS]

- AMONG,
- AND,
- ARITH,
- ARITH_OR,
- CHANGE,
- CHANGE_VECTORS,
- CLAUSE_AND,
- CLAUSE_OR,
- COND_LEX_COST,
- COND_LEX_GREATER,
- COND_LEX_GREATEREQ,
- COND_LEX_LESS,
- COND_LEX_LESSEQ,
- CONSECUTIVE_GROUPS_OF_ONES,
- ELEMENTN,
- EQUIVALENT,
- GLOBAL_CONTIGUITY,
- IMPLY,
- IN_INTERVAL,

- INCREASING_GLOBAL_CARDINALITY,
- INCREASING_NVALUE,
- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- LEX_BETWEEN,
- LEX_DIFFERENT,
- LEX_EQUAL,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- NAND,
- NOR,
- OR,
- PATTERN,
- SMOOTH,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_DO_NOT_OVERLAP,
- XOR.

A constraint for which the decomposition associated with its usually counter-free deterministic automaton[8] is *Berge-acyclic*. Arc-consistency for a *Berge-acyclic* constraint network is achieved by making each constraint of the corresponding network arc-consistent [25]. A constraint network for which the corresponding *intersection graph* does not contain any cycle and such that, for any pair of constraints, the two sets of involved variables share at most one variable is *Berge-acyclic*, where Berge-acyclic is defined by the following two conditions:

1. There is no more than one shared variable between any pair of constraints,

2. The hypergraph corresponding to the constraint network does not contain any cycle. Within [64, page 150] a cycle of an hypergraph H is defined as "*Let H be an hypergraph on a finite set X. A* cycle of length k *($k \geq 2$) is a sequence* $(x_1, E_1, x_2, E_2, x_3, \ldots, E_k, x_1)$ *such that* (1) $E_1, E_2, \ldots, E_k$ *are distinct edges of* H*,* (2) $x_1, x_2, \ldots, x_k$ *are distinct vertices of* H*,* (3) $x_i, x_{i+1} \in E_i$ *($i = 1, 2, \ldots, k-1$),* (4) $x_k, x_1 \in E_k$."

The *intersection graph* of a constraint network is built in the following way: to each vertex corresponds a constraint and there is an edge between two vertices if and only if the sets of variables involved in the two corresponding constraints intersect.

Parts (A), (B), (C) and (D) of Figure 3.8 provide four examples of constraint networks, while parts (E), (F), (G) and (F) give their corresponding intersection graphs.

1. The constraint network corresponding to part (A) is Berge-acyclic since its corresponding intersection graph (E) does not contain any cycle and since there is no more than one shared variable between any pair of constraints.

2. The constraint network corresponding to part (B) is not Berge-acyclic since its hypergraph (B) contains a cycle.

3. The constraint network corresponding to (C) is also not Berge-acyclic since its third and fourth constraints share more than one variable.

[8] All the above constraints, except AMONG, CHANGE, and SMOOTH have a deterministic counter-free automaton. The AMONG constraint has an automaton involving one counter and a single state, see Figure 5.61, while the CHANGE and the SMOOTH constraints have a counter-free non deterministic automaton, see Figures 5.162 and 5.738.

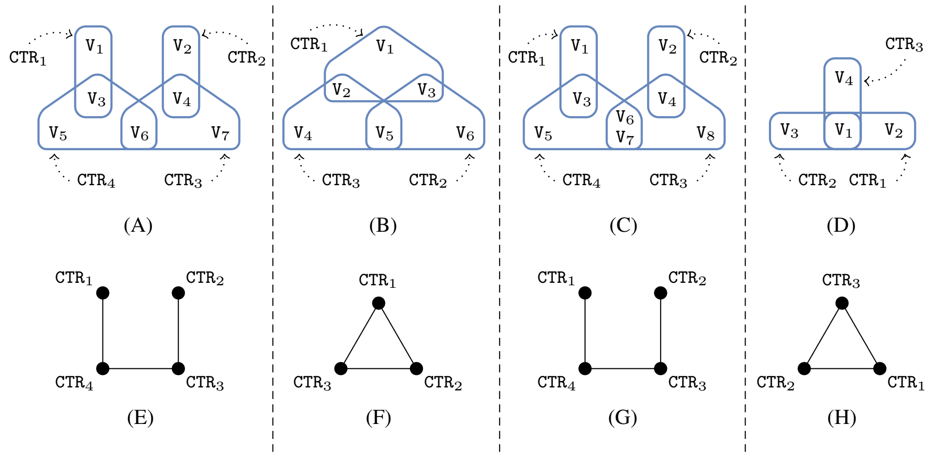


Figure 3.8: (A) and (D): Berge-acyclic constraint networks; (B) and (C): non Berge-acyclic constraint networks; (E), (F), (G), (H): corresponding intersection graphs.

4. Finally, the constraint network corresponding to (D) is Berge acyclic, even though its intersection graph (H) has a cycle, since its hypergraph (D) does not contain any cycle and since there is no more than one shared variable between any pair of constraints.

If we execute the filtering algorithm of each constraint of a Berge-acyclic constraint network $\mathcal{N}$ in an appropriate order then each constraint needs only to be waken twice in order to reach the fix-point. A static ordering for waking the constraints of $\mathcal{N}$ can be determined as follows:

- Consider the intersection graph $\mathcal{G}_{\mathcal{N}}$ associated with the constraint network $\mathcal{N}$. We perform a topological sort on $\mathcal{G}_{\mathcal{N}}$, which always first selects in the remaining part of $\mathcal{G}_{\mathcal{N}}$ a vertex (i.e., a constraint) which has only a single neighbour. Let $C_1, C_2, \ldots, C_n$ be the constraints successively removed by the topological sort.

- Then, the static ordering for reaching a fix-point is given by the sequence $C_1, C_2, \ldots, C_{n-1}, C_n, C_{n-1}, \ldots, C_2, C_1$, where each constraint is woken at most twice. This can be done by using the notion of *propagator group* [258]. This facility allows the user of a solver controlling the order of execution of a group of constraints. Propagator groups are useful, both to guaranty the theoretical worst case complexity of a decomposition, and for accelerating convergence to the fix-point in practice.

If we consider the Berge-acyclic constraint network given by Part (D) of Figure 3.8 an appropriate order for waking the constraints could be, for example, CTR_1, CTR_4, CTR_2, CTR_3, CTR_2, CTR_4, CTR_1.

For heuristics that try creating a Berge-acyclic constraint network see also the keyword heuristics and Berge-acyclic constraint network.

3.7.30 Binary constraint [26 CONS]

- ABS_VALUE,
- DIVISIBLE,
- DIVISIBLE_OR,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_SPARSE,
- EQ,
- EQ_CST,
- EQ_SET,
- GEQ,
- GEQ_CST,
- GT,
- IN_SAME_PARTITION,
- LEQ,
- LEQ_CST,
- LT,
- MULTIPLE,
- NEQ,
- NEQ_CST,
- OPPOSITE_SIGN,
- IN_INTERVAL_REIFIED,
- SAME_SIGN,
- SIGN_OF,
- STAGE_ELEMENT,
- SUM_SET,
- ZERO_OR_NOT_ZERO.

A constraint involving only two variables.

3.7.31 Bioinformatics [3 CONS]

- ALL_DIFFER_FROM_AT_LEAST_K_POS,
- SEQUENCE_FOLDING,
- STABLE_COMPATIBILITY.

Denotes that, for a given constraint, either there is a reference to its uses in Bioinformatics, or it was inspired by a problem from the area of Bioinformatics.

3.7.32 Bipartite [29 CONS]

- ALLDIFFERENT_ON_INTERSECTION,
- ALLPERM,
- AMONG_LOW_UP,
- AMONG_VAR,
- ARITH_OR,
- ASSIGN_AND_COUNTS,
- ASSIGN_AND_NVALUES,
- BIN_PACKING,
- BIPARTITE,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST,
- CARDINALITY_ATMOST_PARTITION,
- CHANGE,
- CHANGE_CONTINUITY,
- CHANGE_PAIR,
- CHANGE_PARTITION,
- COMMON,
- COMMON_INTERVAL,
- COMMON_MODULO,
- COMMON_PARTITION,
- CORRESPONDENCE,
- COUNTS,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- DECREASING,
- INVERSE_WITHIN_RANGE,
- LEX_EQUAL,
- TWO_ORTH_DO_NOT_OVERLAP,
- USES.

Denotes that a constraint is defined by one graph constraint for which the final graph is bipartite.

3.7.33 Bipartite matching [15 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_BETWEEN_SETS,
- ALLDIFFERENT_EXCEPT_0,
- ALLDIFFERENT_CST,
- ATLEAST_NVALUE,
- CORRESPONDENCE,
- DISJOINT,
- INVERSE,
- LEX_ALLDIFFERENT,
- SAME,
- SOFT_ALLDIFFERENT_VAR,
- SOFT_SAME_VAR,
- SOFT_USED_BY_VAR,
- SYMMETRIC_CARDINALITY,
- USED_BY.

Denotes that, for a given constraint, a *bipartite matching* algorithm can be used within its filtering algorithm. A *bipartite matching* is a subgraph that pairs every vertex of a *bipartite graph* with exactly one other vertex. A *bipartite graph* is a graph for

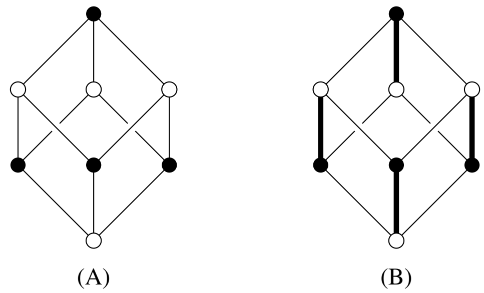


Figure 3.9: (A) A bipartite graph and (B) one of its bipartite matching

which the set of vertices can be partitioned in two parts such that no two vertices in the same part are joined by an edge. Part (A) of Figure 3.9 shows a bipartite graph with a possible division of the vertices in black and white, while part (B) depicts with a thick line a bipartite matching of this graph.

A used generalisation so called *degree-matching* of a graph is a spanning sugraph where every vertex is associated with the bound degree of the matched edges.

3.7.34 Bipartite matching in convex bipartite graphs [2 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_CST.

Denotes that, for a given constraint, a bipartite matching algorithm using *Glover's rule* for constructing a maximum matching of a convex bipartite graph can be used. Given a convex bipartite graph $G = (U, V, E)$ where $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$, Glover [205] showed how to efficiently compute a maximum matching in such a graph:

1. First start with the empty matching.
2. Second for each vertex v_j of V, $(j = 1, 2, \ldots, m)$, if v_j has still a free neighbour in U, then add to the current matching the edge (u_i, v_j) for which u_i is free and $\alpha_i = \max\{j : (x_i, y_j) \in E, y_j \in V\}$ is as small as possible.

3.7.35 Boolean channel [1 CONS]

- DOMAIN_CONSTRAINT.

A constraint that allows for making the link between a set of 0-1 variables $B_1, B_2, \ldots, B_n$ and a domain variable V. It forces a condition of the form $V = i \Leftrightarrow B_i = 1$.

3.7.36 Boolean constraint [9 CONS]

- AND,
- CLAUSE_AND,
- CLAUSE_OR,
- EQUIVALENT,
- IMPLY,
- NAND,
- NOR,
- OR,
- XOR.

A *Boolean constraint* is a constraint of the form $v = f(v_1, \ldots, v_n)$ $(n \geq 2)$ where $v, v_1, \ldots, v_n$ are 0-1 variables and where $f(v_1, \ldots, v_n)$ is a logical expression involving connectors, such as $\neg$, $\vee$, or $\wedge$.

3.7.37 Border [1 CONS]

- PERIOD.

A constraint that can be related to the notion of *border*, which we define now. Given a sequence $s = urv$, r is a *prefix* of s when u is empty, r is a *suffix* of s when v is empty, r is a *proper factor* of s when $r \neq s$. *A border* of a non-empty sequence s is a *proper factor* of s, which is both a *prefix* and a *suffix* of s. We have that the smallest period of a sequence s is equal to the size of s minus the length of the longest border of s.

3.7.38 Bound-consistency

[19 CONS]

- ALLDIFFERENT,
- ALL_MIN_DIST,
- ATMOST1,
- ATMOST_NVALUE,
- NVALUE,
- GLOBAL_CARDINALITY,
- GLOBAL_CARDINALITY_LOW_UP,
- INCREASING_SUM,
- K_ALLDIFFERENT,
- MULTI_INTER_DISTANCE,
- SAME,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP,
- SLIDING_SUM,
- SOFT_ALL_EQUAL_MAX_VAR,
- SOFT_ALL_EQUAL_MIN_CTR,
- SORT,
- SUM_FREE,
- SUM_OF_INCREMENTS,
- USED_BY.

Denotes that, for a given constraint, there is a filtering algorithm or a reformulation in term of other constraints that ensures *bound-consistency* for its domain variables.[9] A filtering algorithm or a reformulation ensure *bound-consistency* for a given constraint *ctr* using distinct domain variables if and only if for every domain variable V of *ctr*:

- There exists at least one solution to *ctr* such that $V = \underline{V}$ and every other domain variable W of *ctr* is assigned to a value located in its range $[\underline{W}, \overline{W}]$,
- There exists at least one solution to *ctr* such that $V = \overline{V}$ and every other domain variable W of *ctr* is assigned to a value located in its range $[\underline{W}, \overline{W}]$.

This consistency is called *bound(Z) consistency* in [65]. One of its interest is that it sometimes gives the opportunity to come up with a filtering algorithm that has a lower complexity than the algorithm that achieves arc-consistency. Discarding holes from the domain variables usually leads to graphs with a specific structure for which one can take advantage in order to derive more efficient graph algorithms. Filtering algorithms that achieve *bound-consistency* can also be used in a pre-processing phase before applying a more costly filtering algorithm that achieves arc-consistency.

Note that there is a second definition of *bound-consistency*, called *bound(D) consistency* in [65], where the range $[\underline{W}, \overline{W}]$ is replaced by the domain of the variable W. However within the context of global constraints most filtering algorithms do not refer to this second definition.

Finally, within the context of constraints involving only set variables, *bound-consistency* is defined in the following way. A constraint *ctr* defined on distinct set variables is *bound-consistent* if and only if for every pair (V, v) such that V is a set variable of *ctr* and v an integer value, if $v \in \underline{V}$ then v belongs to the set assigned to V in all solutions to *ctr* and if $v \in \overline{V} \setminus \underline{V}$ then v belongs to the set assigned to V in at least one solution and is excluded from this set in at least one solution.

[9] In the context of the NVALUE constraint, bound-consistency is only achieved if and only if, the minimum of the variable that denotes the number of distinct values is not constrained at all. In the context of the K_ALLDIFFERENT constraint, bound-consistency is only achieved when we have two overlapping ALLDIFFERENT constraints, see [80] for more details.

3.7.39 Business rules [3 CONS]

- CYCLE,
- DIFFN,
- GEOST.

Denotes that a dedicated language was introduced within an argument of a global constraint for directly specifying a specific type of business rules:

- The CYCLE constraint was extended in order to accept rules specifying *forbidden sequences of vertices* within each cycle [93].
- The DIFFN constraint was extended in order to accept *calendar rules* specifying the way tasks can be interrupted or not on each resource [26]. This was done since many real scheduling problems have not only to consider disjunctive and assignment constraints, but also operational rules expressing how tasks can be interrupted.
- The GEOST constraint was extended in order to directly accept a great variety of *packing and placement rules* [107].

3.7.40 Centered cyclic(1) constraint network(1) [9 CONS]

- BETWEEN_MIN_MAX,
- DOMAIN_CONSTRAINT,
- IN,
- MAXIMUM,
- MINIMUM,
- MINIMUM_EXCEPT_0,
- NOT_IN,
- OPEN_MAXIMUM,
- OPEN_MINIMUM.

A constraint network corresponding to the pattern depicted by Figure 3.10. Circles depict variables, while arcs are represented by a set of variables. Grey circles correspond to optional variables. All pairs of constraints have at most one variable in common.

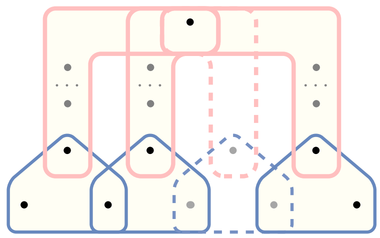


Figure 3.10: Hypergraph associated with a centered cyclic(1) constraint network(1)

3.7.41 Centered cyclic(2) constraint network(1) [8 CONS]

- ELEM,
- ELEMENT,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_SPARSE,
- IN_SAME_PARTITION,
- MINIMUM_GREATER_THAN,
- STAGE_ELEMENT.

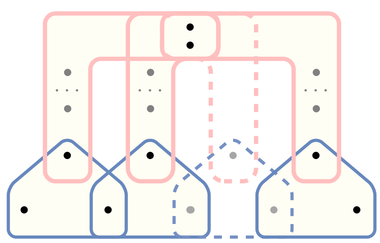


Figure 3.11: Hypergraph associated with a centered cyclic(2) constraint network(1)

A constraint network corresponding to the pattern depicted by Figure 3.11. Circles depict variables, while arcs are represented by a set of variables. Grey circles correspond to optional variables.

3.7.42 Centered cyclic(3) constraint network(1) [2 CONS]

- ELEMENT_MATRIX,
- NEXT_ELEMENT.

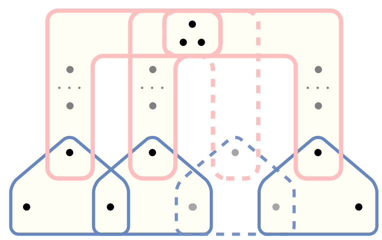


Figure 3.12: Hypergraph associated with a centered cyclic(3) constraint network(1)

A constraint network corresponding to the pattern depicted by Figure 3.12. Circles depict variables, while arcs are represented by a set of variables. Grey circles correspond to optional variables.

3.7.43 Channel routing [1 CONS]

- CONNECT_POINTS.

A constraint that can be used for modelling *channel routing* problems. *Channel routing* consists of creating a layout in a rectangular region of a VLSI chip in order to link together the terminals of different modules of the chip. Connections are usually made by wire segments on two different layers: horizontal wire segments on the first layer are placed along lines called tracks, while vertical wire segments on the second layer connect terminals to the horizontal wire segments, with vias at the intersection.

3.7.44 Channelling constraint [9 CONS]

- CALENDAR,
- DOMAIN_CONSTRAINT,
- INVERSE,
- INVERSE_EXCEPT_LOOP,
- INVERSE_OFFSET,
- INVERSE_SET,
- INVERSE_WITHIN_RANGE,
- LINK_SET_TO_BOOLEANS,
- SAME.

Constraints that allow for linking two models of the same problem [223]. Usually channelling constraints show up in the following context:

- When a problem can be modelled by using different types of variables (e.g., 0-1 variables, domain variables, set variables),
- When a problem can be modelled by using two distinct matrices of variables representing the same information redundantly,
- When, in a problem, the roles of the variables and the values can be interchanged. This is typically the case when we have a bijection between a set of variables and the values they can take.
- When, in a problem, we use two time coordinates systems (e.g., see CALENDAR).

3.7.45 Circuit [7 CONS]

- BALANCE_CYCLE,
- CIRCUIT,
- CUTSET,
- CYCLE,
- PROPER_CIRCUIT,
- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_LOOP.

A constraint such that its initial or its final graph corresponds to zero (e.g., CUTSET), one (e.g., CIRCUIT) or several (see, e.g., the CYCLE, and SYMMETRIC_ALLDIFFERENT constraints) vertex-disjoint circuits.

3.7.46 Circular sliding cyclic(1) constraint network(2) [1 CONS]

- CIRCULAR_CHANGE.

A constraint network corresponding to the pattern depicted by Figure 3.13. Circles depict variables, while arcs are represented by a set of variables.

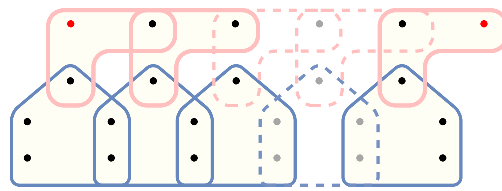


Figure 3.13: Hypergraph corresponding to a circular sliding cyclic(1) constraint network(2), where the two red circles correspond to the same variable

3.7.47 Cluster [1 CONS]

- CIRCUIT_CLUSTER.

A constraint that partitions the vertices of an initial graph into several clusters.

3.7.48 Coloured [5 CONS]

- ASSIGN_AND_COUNTS,
- COLOURED_CUMULATIVE,
- COLOURED_CUMULATIVES,
- CYCLE_CARD_ON_PATH,
- INTERVAL_AND_COUNT.

A constraint with a collection where one of the attributes is a colour.

3.7.49 Compulsory part [9 CONS]

- COLOURED_CUMULATIVE,
- COLOURED_CUMULATIVES,
- CUMULATIVE,
- CUMULATIVE_CONVEX,
- CUMULATIVE_PRODUCT,
- CUMULATIVE_TWO_D,
- CUMULATIVES,
- DIFFN,
- DISJUNCTIVE.

A constraint for which the filtering algorithm may use the notion of *compulsory part*. The notion of *compulsory part* was introduced by A. Lahrichi within the context of cumulative scheduling problems [259], [261], [260] as well as within the context of rectangles placement problems [262]. Within these two contexts, the *compulsory part* respectively corresponds to the intersection of all feasible instances of a task or to the intersection of all feasible instances of a rectangle.

Figure 3.14 illustrates the notion of *compulsory part* in the context of scheduling and placement problems. The first, second and third rows respectively corresponds to the CUMULATIVE [1], the CUMULATIVE_TRAPEZE [330, 331] and the DIFFN [50] constraints. The first, second and third columns respectively correspond to the shape of the object for which we compute the compulsory part, to the extreme positions of the object and to the corresponding compulsory part. When both, the shape of an object is convex and the domain of its origin is also convex, we do not need to consider all feasible instances of the object to compute its compulsory part. We only need to position the object to the extreme positions of its domain and to compute the intersection to get its compulsory part [50].

- This is the case of the CUMULATIVE constraint where a task is positioned to its earliest and latest starts s_{min} and s_{max} (see the first row and second column of Figure 3.14).

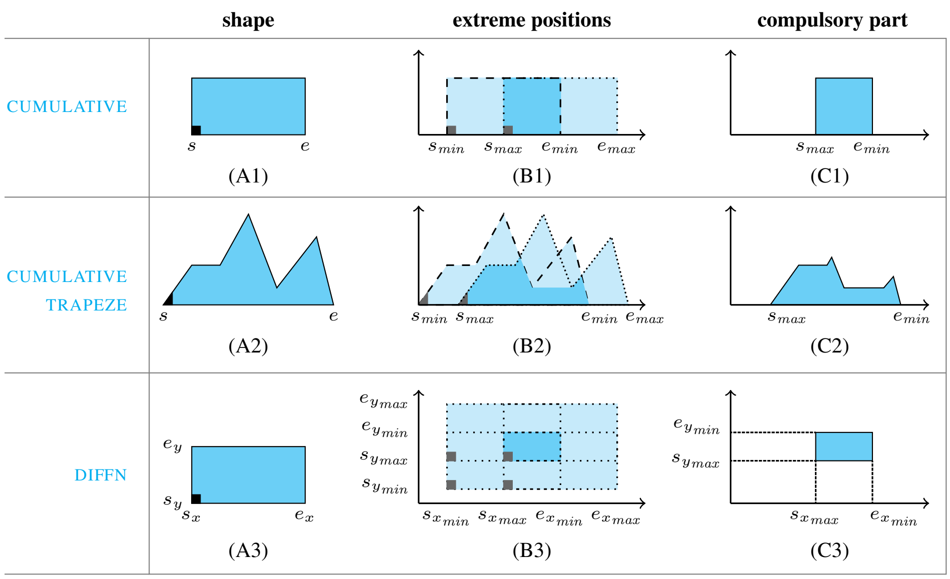


Figure 3.14: Illustration of the notion of compulsory part

- This is also the case of the DIFFN constraint where an orthotope is positioned to its $2 \cdot n$ extreme positions, where n is the number of dimensions of the placement space (see the third row and second column of Figure 3.14, where the origin of the rectangle is fixed to the extreme positions $(s_{x_{min}}, s_{y_{min}})$, $(s_{x_{max}}, s_{y_{min}})$, $(s_{x_{min}}, s_{y_{max}})$, and $(s_{x_{max}}, s_{y_{max}})$).

- But this is not the case of the CUMULATIVE_TRAPEZE constraint with a task that has a valley, i.e. a task for which a resource consumption decrease is followed by a resource consumption increase. In addition of computing the intersection between the two extreme positions $\mathrm{I}^{\mathrm{min}}$ and $\mathrm{I}^{\mathrm{max}}$ of a task, we must also consider the valleys to further reduce $\mathrm{I}^{\mathrm{min}} \cap \mathrm{I}^{\mathrm{max}}$ as explained now [331, page 250]. The *end of a valley* is the lowest rightmost point of a valley. We must remove from $\mathrm{I}^{\mathrm{min}} \cap \mathrm{I}^{\mathrm{max}}$ all parts that are located both (1) between the earliest start and latest end of the valley end, (2) and on top of the valley. Figure 3.15 illustrates this point for the task used in the second row and first column of Figure 3.14.

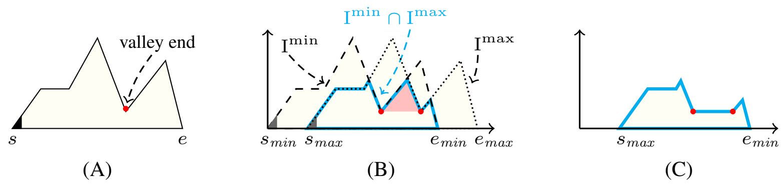


Figure 3.15: Illustrating the computation of the compulsory part of a task with a valley: (A) the task shape and its valley end in red, (B) in cyan the intersection between the task positioned at its earliest start (in dashed) and its latest start (in dotted); in pink the part located (1) between the earliest and latest positions of the valley end, and (2) on top of the valley, (C) the compulsory part of the task, i.e., $\mathrm{I}^{\mathrm{min}} \cap \mathrm{I}^{\mathrm{max}}$ from which we remove the pink part on top of the valley.

3.7.50 Conditional constraint [2 CONS]

- SIZE_MAX_SEQ_ALLDIFFERENT,
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT.

A constraint that allows for expressing that some constraints can be enforced during the enumeration phase.

3.7.51 Configuration problem [1 CONS]

- ELEMENT_PRODUCT.

A constraint that was used for modelling configuration problems. Within the context of configuration problems [411], it is crucial to identify all variable-value pairs which do not participate to any solution. This stems from the fact that one wants typically to avoid proposing invalid choices to the user of such configuration systems.

Note also that open constraints are also useful in the context of configuration problems.

3.7.52 Connected component [21 CONS]

- ALLDIFFERENT_ON_INTERSECTION,
- BALANCE_CYCLE,
- BALANCE_PATH,
- BALANCE_TREE,
- BINARY_TREE,
- CHANGE_CONTINUITY,
- CONNECTED,
- CYCLE,
- CYCLE_CARD_ON_PATH,
- CYCLE_RESOURCE,
- GLOBAL_CONTIGUITY,
- GROUP,
- K_CUT,
- MAP,
- NVALUE_ON_INTERSECTION,
- PATH,
- PROPER_FOREST,
- TEMPORAL_PATH,
- TREE,
- TREE_RANGE,
- TREE_RESOURCE.

Denotes that a constraint uses in its definition a graph property (e.g., **MAX_NCC**, **MIN_NCC**, **NCC**) constraining the connected components of its associated final graph.

3.7.53 Consecutive loops are connected [3 CONS]

- GROUP,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION.

Denotes that the graph constraints of a global constraint use only the $PATH$ and the $LOOP$ arc generators and that their final graphs do not contain consecutive vertices that are not connected together by an arc. Moreover all vertices of their final graphs have a loop.

3.7.54 Consecutive values [3 CONS]

- MAX_SIZE_SET_OF_CONSECUTIVE_VAR,
- MIN_SIZE_SET_OF_CONSECUTIVE_VAR,
- NSET_OF_CONSECUTIVE_VALUES.

A constraint for which the definition involves the notion of consecutive values assigned to the variables of a collection of domain variables.

3.7.55 Constraint between two collections of variables [36 CONS]

- ALLDIFFERENT_ON_INTERSECTION,
- COMMON,
- COMMON_INTERVAL,
- COMMON_MODULO,
- COMMON_PARTITION,
- DISJOINT,
- INCOMPARABLE,
- INVERSE_WITHIN_RANGE,
- INCOMPARABLE,
- LEX_DIFFERENT,
- LEX_EQUAL,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LEX_LESSEQ_ALLPERM,
- SAME,
- SAME_AND_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP,
- SAME_INTERSECTION,
- SAME_INTERVAL,
- SAME_MODULO,
- SAME_PARTITION,
- SOFT_SAME_INTERVAL_VAR,
- SOFT_SAME_MODULO_VAR,
- SOFT_SAME_PARTITION_VAR,
- SOFT_SAME_VAR,
- SOFT_USED_BY_INTERVAL_VAR,
- SOFT_USED_BY_MODULO_VAR,
- SOFT_USED_BY_PARTITION_VAR,
- SOFT_USED_BY_VAR,
- SORT,
- USES,
- USED_BY,
- USED_BY_INTERVAL,
- USED_BY_MODULO,
- USED_BY_PARTITION.

A constraint involving only two collections of domain variables in its arguments.

3.7.56 Constraint between three collections of variables [2 CONS]

- CORRESPONDENCE,
- SORT_PERMUTATION.

A constraint involving only three collections of domain variables in its arguments.

3.7.57 Constraint involving set variables [32 CONS]

- ALLDIFFERENT_BETWEEN_SETS,
- ATMOST1,
- BIPARTITE,
- CLIQUE,
- CONNECTED,
- DAG,
- DISJ,
- DOM_REACHABILITY,
- EQ_SET,
- GRAPH_ISOMORPHISM,
- IN_SET,
- INVERSE_SET,
- K_CUT,
- LINK_SET_TO_BOOLEANS,
- OPEN_ALLDIFFERENT,
- OPEN_AMONG,
- OPEN_ATLEAST,
- OPEN_ATMOST,
- OPEN_GLOBAL_CARDINALITY,
- OPEN_GLOBAL_CARDINALITY_LOW_UP,
- PATH_FROM_TO,
- PROPER_FOREST,
- ROOTS,
- SET_VALUE_PRECEDE,
- STRONGLY_CONNECTED,
- SUBGRAPH_ISOMORPHISM,
- SUM_FREE,
- SUM_SET,
- SYMMETRIC,
- SYMMETRIC_CARDINALITY,
- SYMMETRIC_GCC,
- TOUR.

A constraint involving set variables in its arguments.

3.7.58 Constraint on the intersection [4 CONS]

- COMMON,
- ALLDIFFERENT_ON_INTERSECTION,
- NVALUE_ON_INTERSECTION,
- SAME_INTERSECTION.

Denotes that a constraint involving two collections of variables imposes a restriction on the values that occur in both collections.

3.7.59 Constructive disjunction [5 CONS]

- CASE,
- DISJUNCTIVE,
- DIFFN,
- GEOST,
- TWO_ORTH_DO_NOT_OVERLAP.

A constraint for which a filtering algorithm uses *constructive disjunction*. *Constructive disjunction* [431, 454] is a technique for handling in an active way a set of disjunctive constraints. It consists to try out each alternative of a disjunction and then to remove values that were pruned in all alternatives. Table 3.10 illustrates this technique in the context of a non-overlapping constraint between two rectangles (i.e., a special case of the TWO_ORTH_DO_NOT_OVERLAP constraint). The first rectangle R_1 has a width of 3 and a height of 2, while the second rectangle R_2 has a width of 2 and a height of 5. The coordinates (x_1, y_1) of the lower leftmost corner of R_1 have to be respectively located within intervals $[3, 5]$ and $[6, 7]$. Similarly the coordinates (x_2, y_2) of the lower leftmost corner of R_2 have to be located within $[2, 4]$ and $[3, 4]$.

- In the context of the CASE constraint, constructive disjunction is applied on each sink node of the dag describing the set of solutions (i.e., we remove values that are removed in all the sink nodes).
- In the context of the DISJUNCTIVE (respectively DIFFN) constraint, constructive disjunction can be applied on each pair of tasks (respectively objects). However, as described in the **Algorithm** slots of these two constraints, more specific and efficient filtering algorithms exist for both constraints.
- In the context of the GEOST constraint, constructive disjunction is applied on the different potential values of the shape variable of an object in order to prune its coordinates.

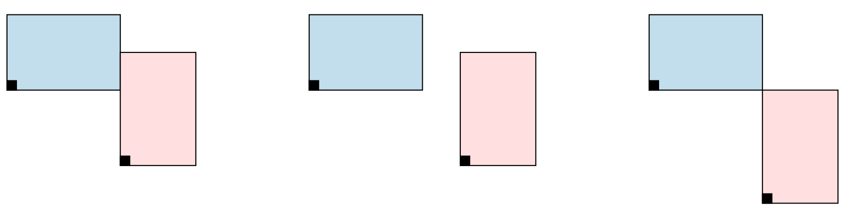
Table 3.10: Illustrating *constructive disjunction* in the context of a non-overlapping constraint between two rectangles.

Hypothesis regarding the respective position of R_1 and R_2			
R_2 before R_1: $X_2 + 2 \leq X_1$ $[2,4] + 2 \leq [3,5]$ $[2,\mathbf{3}] + 2 \leq [\mathbf{4},5]$	R_2 after R_1: $X_1 + 3 \leq X_2$ $[3,5] + 3 \leq [2,4]$ **contradiction**	R_2 below R_1: $Y_2 + 5 \leq Y_1$ $[3,4] + 5 \leq [6,7]$ **contradiction**	R_2 on top of R_1: $Y_1 + 2 \leq Y_2$ $[6,7] + 2 \leq [3,4]$ **contradiction**
Removed values from each variable according to each hypothesis			
$X_1 : \{3\}$ $X_2 : \{4\}$ $Y_1 : \emptyset$ $Y_2 : \emptyset$	$X_1 : \{3,4,5\}$ $X_2 : \{2,3,4\}$ $Y_1 : \{6,7\}$ $Y_2 : \{3,4\}$	$X_1 : \{3,4,5\}$ $X_2 : \{2,3,4\}$ $Y_1 : \{6,7\}$ $Y_2 : \{3,4\}$	$X_1 : \{3,4,5\}$ $X_2 : \{2,3,4\}$ $Y_1 : \{6,7\}$ $Y_2 : \{3,4\}$
Values finally removed: value 3 from X_1 and value 4 from X_2			

3.7.60 Contact [2 CONS]

- ORTHS_ARE_CONNECTED,
- TWO_ORTH_ARE_IN_CONTACT.

A constraint enforcing that some orthotopes touch each other. Part (A) of Figure 3.16 shows two orthotopes that are in contact while parts (B) and (C) give two examples of orthotopes that are not in contact.



(A) (B) (C)

Figure 3.16: Illustration of the notion of contact: (A) two rectangles in contact, (B), (C) two rectangles not in contact

3.7.61 Contractible [195 CONS]

- ALL_DIFFER_FROM_AT_LEAST_K_POS (contractible wrt VECTORS),
- ALL_DIFFER_FROM_EXACTLY_K_POS (contractible wrt VECTORS),
- ALL_EQUAL (contractible wrt VARIABLES),
- ALL_INCOMPARABLE (contractible wrt VECTORS),
- ALL_MIN_DIST (contractible wrt VARIABLES),
- ALLDIFFERENT (contractible wrt VARIABLES),
- ALLDIFFERENT_BETWEEN_SETS (contractible wrt VARIABLES),
- ALLDIFFERENT_CST (contractible wrt VARIABLES),
- ALLDIFFERENT_EXCEPT_0 (contractible wrt VARIABLES),
- ALLDIFFERENT_INTERVAL (contractible wrt VARIABLES),
- ALLDIFFERENT_MODULO (contractible wrt VARIABLES),
- ALLDIFFERENT_ON_INTERSECTION (contractible wrt VARIABLES1),
- ALLDIFFERENT_ON_INTERSECTION (contractible wrt VARIABLES2),
- ALLDIFFERENT_PARTITION (contractible wrt VARIABLES),
- ALLPERM (suffix-contractible wrt MATRIX.vec),
- AMONG (contractible wrt VARIABLES when NVAR $= 0$),
- AMONG (contractible wrt VARIABLES when NVAR $= |$VARIABLES$|$),
- AMONG_DIFF_0 (contractible wrt VARIABLES when NVAR $= 0$),
- AMONG_DIFF_0 (contractible wrt VARIABLES when NVAR $= |$VARIABLES$|$),
- AMONG_INTERVAL (contractible wrt VARIABLES when NVAR $= 0$),
- AMONG_INTERVAL (contractible wrt VARIABLES when NVAR $= |$VARIABLES$|$),
- AMONG_LOW_UP (contractible wrt VARIABLES when UP $= 0$),
- AMONG_LOW_UP (contractible wrt VARIABLES when UP $= |$VARIABLES$|$),
- AMONG_MODULO (contractible wrt VARIABLES when NVAR $= 0$),
- AMONG_MODULO (contractible wrt VARIABLES when NVAR $= |$VARIABLES$|$),
- AMONG_SEQ (contractible wrt VARIABLES when UP $= 0$),
- AMONG_SEQ (contractible wrt VARIABLES when SEQ $= 1$),
- AMONG_SEQ (prefix-contractible wrt VARIABLES),
- AMONG_SEQ (suffix-contractible wrt VARIABLES),
- AMONG_VAR (contractible wrt VARIABLES when NVAR $= 0$),
- AMONG_VAR (contractible wrt VARIABLES when NVAR $= |$VARIABLES$|$),
- ARITH (contractible wrt VARIABLES),
- ARITH_OR (contractible wrt [VARIABLES1, VARIABLES2]),
- ARITH_SLIDING (contractible wrt VARIABLES when RELOP $\in [<, \leq]$ and minval(VARIABLES.var) ≥ 0),
- ARITH_SLIDING (suffix-contractible wrt VARIABLES),
- ASSIGN_AND_COUNTS (contractible wrt ITEMS when RELOP $\in [<, \leq]$),
- ASSIGN_AND_NVALUES (contractible wrt ITEMS when RELOP $\in [<, \leq]$),
- ATMOST (contractible wrt VARIABLES),
- ATMOST1 (contractible wrt SETS),
- ATMOST_NVALUE (contractible wrt VARIABLES),
- ATMOST_NVECTOR (contractible wrt VECTORS),
- BIN_PACKING (contractible wrt ITEMS),
- BIN_PACKING_CAPA (contractible wrt ITEMS),

- CALENDAR (contractible wrt INSTANTS),
- CHANGE (contractible wrt VARIABLES when CTR $\in [\neq, <, \geq, >, \leq]$ and NCHANGE $= 0$),
- CHANGE (contractible wrt VARIABLES when CTR $\in [=, <, \geq, >, \leq]$ and NCHANGE $= |\texttt{VARIABLES} - 1|$),
- COLOURED_CUMULATIVE (contractible wrt TASKS),
- COLOURED_CUMULATIVES (contractible wrt TASKS),
- COMPARE_AND_COUNT (contractible wrt [VARIABLES1, VARIABLES2] when COUNT $\in [<, \leq]$),
- CONTAINS_SBOXES (suffix-contractible wrt OBJECTS),
- COUNT (contractible wrt VARIABLES when RELOP $\in [<, \leq]$),
- COUNTS (contractible wrt VARIABLES when RELOP $\in [<, \leq]$),
- COVERS_SBOXES (suffix-contractible wrt OBJECTS),
- CUMULATIVE (contractible wrt TASKS),
- CUMULATIVE_CONVEX (contractible wrt TASKS),
- CUMULATIVE_PRODUCT (contractible wrt TASKS),
- CUMULATIVE_TWO_D (contractible wrt RECTANGLES),
- CUMULATIVE_WITH_LEVEL_OF_PRIORITY (contractible wrt TASKS),
- CUMULATIVES (contractible wrt TASKS when RELOP $\in [\leq]$ and $\texttt{minval}(\texttt{TASKS.height}) \geq 0$),
- DECREASING (contractible wrt VARIABLES),
- DIFFN (contractible wrt ORTHOTOPES),
- DIFFN_COLUMN (contractible wrt ORTHOTOPES),
- DIFFN_INCLUDE (contractible wrt ORTHOTOPES),
- DISJOINT (contractible wrt VARIABLES1),
- DISJOINT (contractible wrt VARIABLES2),
- DISJOINT_SBOXES (suffix-contractible wrt OBJECTS),
- DISJOINT_TASKS (contractible wrt TASKS1),
- DISJOINT_TASKS (contractible wrt TASKS2),
- DISJUNCTIVE (contractible wrt TASKS),
- DISJUNCTIVE_OR_SAME_END (contractible wrt TASKS),
- DISJUNCTIVE_OR_SAME_START (contractible wrt TASKS),
- DOMAIN (contractible wrt VARIABLES),
- EQUAL_SBOXES (suffix-contractible wrt OBJECTS),
- GLOBAL_CARDINALITY (contractible wrt VALUES),
- GLOBAL_CARDINALITY_LOW_UP (contractible wrt VALUES),
- GLOBAL_CONTIGUITY (contractible wrt VARIABLES),
- GOLOMB (contractible wrt VARIABLES),
- INCREASING (contractible wrt VARIABLES),
- INSIDE_SBOXES (suffix-contractible wrt OBJECTS),
- INT_VALUE_PRECEDE (suffix-contractible wrt VARIABLES),
- INT_VALUE_PRECEDE_CHAIN (contractible wrt VALUES),
- INT_VALUE_PRECEDE_CHAIN (suffix-contractible wrt VARIABLES),
- INTERVAL_AND_COUNT (contractible wrt COLOURS),
- INTERVAL_AND_COUNT (contractible wrt TASKS),
- INTERVAL_AND_SUM (contractible wrt TASKS),
- K_ALLDIFFERENT (contractible wrt VARS),
- K_DISJOINT (contractible wrt SETS),
- K_SAME (contractible wrt SETS),

- K_SAME_INTERVAL (contractible wrt SETS),
- K_SAME_MODULO (contractible wrt SETS),
- K_SAME_PARTITION (contractible wrt SETS),
- K_USED_BY (contractible wrt SETS),
- K_USED_BY_INTERVAL (contractible wrt SETS),
- K_USED_BY_MODULO (contractible wrt SETS),
- K_USED_BY_PARTITION (contractible wrt SETS),
- LEX_ALLDIFFERENT (contractible wrt VECTORS),
- LEX_BETWEEN (suffix-contractible wrt [LOWER_BOUND, VECTOR, UPPER_BOUND_BOUND]),
- LEX_CHAIN_LESS (contractible wrt VECTORS),
- LEX_CHAIN_LESSEQ (contractible wrt VECTORS),
- LEX_CHAIN_LESSEQ (suffix-contractible wrt VECTORS.vec),
- LEX_EQUAL (contractible wrt [VECTOR1, VECTOR2]),
- LEX_GREATEREQ (suffix-contractible wrt [VECTOR1, VECTOR2]),
- LEX_LESSEQ (suffix-contractible wrt [VECTOR1, VECTOR2]),
- LEX_LESSEQ_ALLPERM (suffix-contractible wrt [VECTOR1, VECTOR2]),
- MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES (contractible wrt VECTORS when MAX $= 1$),
- MAX_OCC_OF_SORTED_TUPLES_OF_VALUES (contractible wrt VECTORS when MAX $= 1$),
- MAX_OCC_OF_TUPLES_OF_VALUES (contractible wrt VECTORS when MAX $= 1$),
- MEET_SBOXES (suffix-contractible wrt OBJECTS),
- MULTI_INTER_DISTANCE (contractible wrt VARIABLES),
- MULTI_GLOBAL_CONTIGUITY (contractible wrt VARIABLES),
- NAND (contractible wrt VARIABLES when VAR $= 0$),
- NEQUIVALENCE (contractible wrt VARIABLES when NEQUIV $= 1$ and $|$VARIABLES$| > 0$),
- NEQUIVALENCE (contractible wrt VARIABLES when NEQUIV $=$ $|$VARIABLES$|$),
- NINTERVAL (contractible wrt VARIABLES when NVAL $= 1$ and $|$VARIABLES$| > 0$),
- NINTERVAL (contractible wrt VARIABLES when NVAL $=$ $|$VARIABLES$|$),
- NO_PEAK (contractible wrt VARIABLES),
- NO_VALLEY (contractible wrt VARIABLES),
- NON_OVERLAP_SBOXES (suffix-contractible wrt OBJECTS),
- NOR (contractible wrt VARIABLES when VAR $= 1$),
- NOT_IN (contractible wrt VALUES),
- NPAIR (contractible wrt PAIRS when NPAIRS $= 1$ and $|$PAIRS$| > 0$),
- NPAIR (contractible wrt PAIRS when NPAIRS $=$ $|$PAIRS$|$),
- NVALUE (contractible wrt VARIABLES when NVAL $= 1$ and $|$VARIABLES$| > 0$),
- NVALUE (contractible wrt VARIABLES when NVAL $=$ $|$VARIABLES$|$),
- NVALUE_ON_INTERSECTION (contractible wrt VARIABLES1 when NVAL $= 0$),
- NVALUE_ON_INTERSECTION (contractible wrt VARIABLES2 when NVAL $= 0$),
- NVALUES (contractible wrt VARIABLES when RELOP $\in [<, \leq]$),
- NVALUES (contractible wrt VARIABLES when RELOP $\in [=]$ and LIMIT $= 1$ and $|$VARIABLES$| > 0$),
- NVALUES (contractible wrt VARIABLES when RELOP $\in [=]$ and LIMIT $=$ $|$VARIABLES$|$),
- NVALUES_EXCEPT_0 (contractible wrt VARIABLES when RELOP $\in [<, \leq]$),
- NVECTOR (contractible wrt VECTORS when NVEC $= 1$ and $|$VECTORS$| > 0$),
- NVECTOR (contractible wrt VECTORS when NVEC $=$ $|$VECTORS$|$,
- NVECTORS (contractible wrt VECTORS when RELOP $\in [<, \leq]$),
- OPEN_ALLDIFFERENT (suffix-contractible wrt VARIABLES),

- OPEN_AMONG (suffix-contractible wrt VARIABLES when $NVAR = 0$),
- OPEN_ATMOST (suffix-contractible wrt VARIABLES),
- OR (contractible wrt VARIABLES when $VAR = 0$),
- ORDERED_ATMOST_NVECTOR (contractible wrt VECTORS),
- ORDERED_GLOBAL_CARDINALITY (contractible wrt VALUES),
- ORDERED_NVECTOR (contractible wrt VECTORS when $NVEC = 1$ and $|VECTORS| > 0$),
- ORDERED_NVECTOR (contractible wrt VECTORS when $NVEC = |VECTORS|$),
- ORTH_LINK_ORI_SIZ_END (contractible wrt ORTHOTOPE),
- OVERLAP_SBOXES (suffix-contractible wrt OBJECTS),
- PATTERN (prefix-contractible wrt VARIABLES),
- PATTERN (suffix-contractible wrt VARIABLES),
- PEAK (contractible wrt VARIABLES when $N = 0$),
- PERIOD (contractible wrt VARIABLES when $CTR \in [=]$ and $PERIOD = 1$),
- PERIOD (prefix-contractible wrt VARIABLES),
- PERIOD (suffix-contractible wrt VARIABLES),
- PERIOD_EXCEPT_0 (contractible wrt VARIABLES when $CTR \in [=]$ and $PERIOD = 1$),
- PERIOD_EXCEPT_0 (prefix-contractible wrt VARIABLES),
- PERIOD_EXCEPT_0 (suffix-contractible wrt VARIABLES),
- PERIOD_VECTORS (prefix-contractible wrt VARIABLES),
- PERIOD_VECTORS (suffix-contractible wrt VARIABLES),
- PRODUCT_CTR (contractible wrt VARIABLES when $CTR \in [<, \leq]$ and $\mathtt{minval}(VARIABLES.var) > 0$),
- RANGE_CTR (contractible wrt VARIABLES when $CTR \in [<, \leq]$),
- SAME_AND_GLOBAL_CARDINALITY (contractible wrt VALUES),
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP (contractible wrt VALUES),
- SCALAR_PRODUCT (contractible wrt LINEARTERM when $CTR \in [<, \leq]$, $\mathtt{minval}(LINEARTERM.coeff) \geq 0$ and $\mathtt{minval}(LINEARTERM.var) \geq 0$),
- SET_VALUE_PRECEDE (suffix-contractible wrt VARIABLES),
- SLIDING_DISTRIBUTION (contractible wrt VARIABLES when $SEQ = 1$),
- SLIDING_DISTRIBUTION (prefix-contractible wrt VARIABLES),
- SLIDING_DISTRIBUTION (suffix-contractible wrt VARIABLES),
- SLIDING_DISTRIBUTION (contractible wrt VALUES),
- SLIDING_SUM (contractible wrt VARIABLES when $SEQ = 1$),
- SLIDING_SUM (prefix-contractible wrt VARIABLES),
- SLIDING_SUM (suffix-contractible wrt VARIABLES),
- SLIDING_TIME_WINDOW (contractible wrt TASKS),
- SLIDING_TIME_WINDOW_FROM_START (contractible wrt TASKS),
- SLIDING_TIME_WINDOW_SUM (contractible wrt TASKS),
- SMOOTH (contractible wrt VARIABLES when $NCHANGE = 0$),
- SMOOTH (contractible wrt VARIABLES when $NCHANGE = |VARIABLES| - 1$),
- SOFT_ALLDIFFERENT_CTR (contractible wrt VARIABLES),
- SOFT_ALLDIFFERENT_VAR (contractible wrt VARIABLES),
- STRICTLY_DECREASING (contractible wrt VARIABLES),
- STRICTLY_INCREASING (contractible wrt VARIABLES),
- SUM_CTR (contractible wrt VARIABLES when $CTR \in [<, \leq]$ and $\mathtt{minval}(VARIABLES.var) \geq 0$),

- SUM_CTR (contractible wrt VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) ≤ 0),
- SUM_CUBES_CTR (contractible wrt VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) ≥ 0),
- SUM_CUBES_CTR (contractible wrt VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) ≤ 0),
- SUM_POWERS4_CTR (VARIABLES) when CTR $\in [<, \leq]$,
- SUM_POWERS5_CTR (contractible wrt VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) ≥ 0),
- SUM_POWERS5_CTR (contractible wrt VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) ≤ 0),
- SUM_POWERS6_CTR (VARIABLES) when CTR $\in [<, \leq]$,
- SUM_OF_INCREMENTS (prefix-contractible wrt VARIABLES),
- SUM_OF_INCREMENTS (suffix-contractible wrt VARIABLES),
- SUM_SQUARES_CTR (VARIABLES) when CTR $\in [<, \leq]$,
- TWIN (contractible wrt PAIRS),
- USED_BY (VARIABLES2),
- USED_BY_INTERVAL (contractible wrt VARIABLES2),
- USED_BY_MODULO (contractible wrt VARIABLES2),
- USED_BY_PARTITION (contractible wrt VARIABLES2),
- USES (contractible wrt VARIABLES2),
- VALLEY (contractible wrt VARIABLES when N $= 0$),
- VEC_EQ_TUPLE (contractible wrt [VARIABLES, TUPLE]).

A *contractible* constraint is a constraint for which, given any satisfied ground instance, one can remove any item from one of its collection arguments, without affecting that the resulting constraint still holds, assuming all its restrictions hold. A typical example of a contractible constraint is the ALLDIFFERENT constraint: given any ground satisfied instance, e.g., ALLDIFFERENT($\langle 3, 8, 1\rangle$), we can remove any value from its unique argument without affecting that the resulting constraint still holds. We generalise slightly the original definition of contractibility introduced by [283] in the following ways:

- The sequence of variables is *replaced by a collection*. Consequently, variables are replaced by items. For example, in the context of the CUMULATIVE(TASKS, LIMIT) constraint, we can remove any task from TASKS from any satisfied instance without affecting that the resulting constraint still holds (e.g., if the resource limit LIMIT is not exceeded at any point in time, this still is the case if we remove any task, i.e., since task heights are restricted to be non negative).

- Since the constraint may have *more than one argument*, one has to explicitly specify the argument from which one may remove items.

- Items cannot only be removed from the end of a collection like in [283], but also *from the beginning or from any part*. Allowing to remove items from the beginning is called *prefix-contractibility*, while permitting to remove items from the end is called *suffix-contractibility*. Removing items from any part is just called *contractibility*. As an example, consider the AMONG_SEQ(LOW, UP, SEQ, VARIABLES, VALUES) constraint which forces all sequences of SEQ consecutive variables of the collection VARIABLES to be assigned at least LOW and at most UP values from VALUES. The constraint AMONG_SEQ is not contractible w.r.t. the collection VARIABLES, since removing an item in the middle of VARIABLES creates a new sequence for which the restriction with respect to LOW and UP may not hold. However, if we restrict ourselves to removing just a prefix or suffix from VARIABLES, then the corresponding AMONG_SEQ constraint still holds, since no new sequence is created.

- A constraint may be contractible only if certain restrictions apply to some of its arguments. This is done by explicitly providing a list of restrictions, each restriction corresponding to one of the restrictions described in Section 2.2.3. We call this *conditional contractibility*. Given a source and a target constraint (i.e., the target constraint corresponds to the source constraint from which we remove some items in some arguments) all arguments of the target constraint should be identical to the arguments of the source constraint, except:

  - Argument corresponding to a collection from which we remove items.
  - Argument arg occurring in the list of conditional restrictions with of restriction of the form arg $= f(|c|)$, where c is an argument corresponding to a collection from which we remove items and f a function.

  In addition, all restrictions from the list of restrictions should apply both to the source and target constraints.

  We now provide two examples of conditional contractibility with respect to the AMONG(NVAR, VARIABLES, VALUES) constraint, which forces NVAR to be the number of variables of the collection VARIABLES that are assigned a value in VALUES.

  - In general AMONG is not contractible since removing an item from VARIABLES may change the value of NVAR. However, given a ground satisfied instance for which NVAR is set to 0, we can remove any item from VARIABLES without affecting that the constraint still holds. In this context, the two arguments NVAR and VALUES are left unchanged within the source and the target constraint.

    As an illustration, consider the source constraint AMONG$(0, \langle 2, 4, 2\rangle, \langle 1, 5\rangle)$ and the target constraint AMONG$(0, \langle 2, 2\rangle, \langle 1, 5\rangle)$. Since NVAR is set to 0 both in the source and the target constraint and since VALUES is set to the same list of values both in the source and the target constraint, we have that AMONG$(0, \langle 2, 4, 2\rangle, \langle 1, 5\rangle)$ implies AMONG$(0, \langle 2, 2\rangle, \langle 1, 5\rangle)$.
  - Similarly, when NVAR is equal to |VARIABLES|, all variables are assigned a value in VALUES. In this context, we can remove any variable from

VARIABLES to get a new constraint that still holds, provided that the restriction NVAR $=$ |VARIABLES| still holds. In this example only the argument VALUES is left unchanged between the source and the target constraint. NVAR changes since it occurs in a restriction of the form NVAR $=$ |VARIABLES| in the list of conditional restrictions.

As an illustration, consider the source constraint AMONG$(3, \langle 2, 4, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$ and the target constraint AMONG$(2, \langle 4, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$. Since NVAR is set to the number of items of the VARIABLES collection both in the source and the target constraint, and since VALUES is set to the same list of values both in the source and the target constraint, we have that AMONG$(3, \langle 2, 4, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$ implies AMONG$(2, \langle 4, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$.

- Finally, a last extension corresponds to the fact the sequence of variables from which we remove elements may be replaced by several collections. In this context, items are *removed simultaneously from all collections from exactly the same set of positions*. A set of collections is either defined by a list of collections, or by a collection and one of its attributes, which is itself a collection.

  As a first example, consider the LEX_GREATEREQ(VECTOR1, VECTOR2) constraint, which given two vectors each defined by a collection of variables of the same length, forces that VECTOR1 is lexicographically greater than or equal to VECTOR2. We have that LEX_GREATEREQ is *suffix-contractible* with respect to VECTOR1 and VECTOR2. This means that we can remove the k $(1 \leq k \leq |\text{VECTOR1}|)$ last items from collections VECTOR1 and VECTOR2. Note that the k items should be removed from both collections *simultaneously*. As an illustration, consider the source constraint LEX_GREATEREQ$(\langle 5, 2, 8, 9\rangle, \langle 5, 2, 6, 2\rangle)$ and the target constraint LEX_GREATEREQ$(\langle 5, 2, 8\rangle, \langle 5, 2, 6\rangle)$. Since LEX_GREATEREQ is suffix-contractible with respect to the two collections VECTOR1 and VECTOR2, we have that LEX_GREATEREQ$(\langle 5, 2, 8, 9\rangle, \langle 5, 2, 6, 2\rangle)$ implies LEX_GREATEREQ$(\langle 5, 2, 8\rangle, \langle 5, 2, 6\rangle)$.

  As a second example, consider the LEX_CHAIN_LESSEQ(VECTORS) constraint, which given a collection of vectors each of them defined by a collection of variables of the same length, forces the i^{th} vector to be lexicographically less than or equal to the $(i + 1)^{th}$ vector $(1 \leq i < |\text{VECTORS}|)$. We have that LEX_CHAIN_LESSEQ is *suffix-contractible* with respect to VECTORS.vec. This means that we can remove the k last components of each vectors of the VECTORS collection. As in the previous example the k items should be removed from all collections *simultaneously*. As an illustration, consider the source constraint LEX_CHAIN_LESSEQ$(\langle \texttt{vec} - \langle 5, 2, 3, 9\rangle, \texttt{vec} - \langle 5, 2, 6, 2\rangle, \texttt{vec} - \langle 5, 2, 6, 2\rangle\rangle)$ and the target constraint LEX_CHAIN_LESSEQ$(\langle \texttt{vec} - \langle 5, 2, 3\rangle, \texttt{vec} - \langle 5, 2, 6\rangle, \texttt{vec} - \langle 5, 2, 6\rangle\rangle)$. Since LEX_CHAIN_LESSEQ is suffix-contractible with respect to VECTORS.vec, we have that LEX_CHAIN_LESSEQ$(\langle \texttt{vec} - \langle 5, 2, 3, 9\rangle, \texttt{vec} - \langle 5, 2, 6, 2\rangle, \texttt{vec} - \langle 5, 2, 6, 2\rangle\rangle)$ implies LEX_CHAIN_LESSEQ$(\langle \texttt{vec} - \langle 5, 2, 3\rangle, \texttt{vec} - \langle 5, 2, 6\rangle, \texttt{vec} -$

$\langle 5, 2, 6\rangle\rangle)$.

The keyword extensible introduces a dual notion, where items can be added to a collection that is passed as an argument of a satisfied global constraint without affecting the fact that the resulting constraint is satisfied. Contractibility is a more common property than extensibility.

3.7.62 Convex [2 CONS]

- CUMULATIVE_CONVEX,
- GLOBAL_CONTIGUITY.

A constraint involving the notion of *convexity*. A subset $\mathcal{S}$ of the plane is called *convex* if and only if for any pair of points p, q of this subset the corresponding line segment is contained in $\mathcal{S}$. Part (A) of Figure 3.17 gives an example of convex set, while part (B) depicts an example of non-convex set.

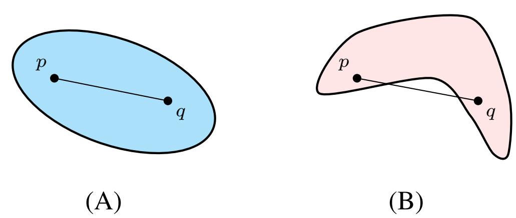


Figure 3.17: (A) A convex set and (B) a non-convex set

3.7.63 Convex bipartite graph [3 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_CST,
- NVALUE.

Denotes that, for a given constraint, its filtering algorithm can take advantage of having a *convex bipartite graph*. A bipartite graph $G = (U, V, E)$ is called *convex* according to its second set of vertices V if there is an ordering on V such that, for any vertex u of U, the neighbours of u form an interval in the previous ordering. Some graph algorithms or some problems become simpler in the context of a convex bipartite graph.

3.7.64 Convex hull relaxation [1 CONS]

- SUM.

Given a non-convex set $\mathcal{S}$, $\mathcal{R}$ is a *convex outer approximation* of $\mathcal{S}$ if:

- $\mathcal{R}$ is convex,
- If $s \in \mathcal{S}$, then $s \in \mathcal{R}$.

Given a non-convex set $\mathcal{S}$, $\mathcal{R}$ is the *convex hull* of $\mathcal{S}$ if:

- $\mathcal{R}$ is a convex outer approximation of $\mathcal{S}$,
- For every $\mathcal{T}$ where $\mathcal{T}$ is a convex outer approximation of $\mathcal{S}$, $\mathcal{R} \subseteq \mathcal{T}$.

Part (A) of Figure 3.18 depicts a non-convex set, while part (B) gives its corresponding convex hull.

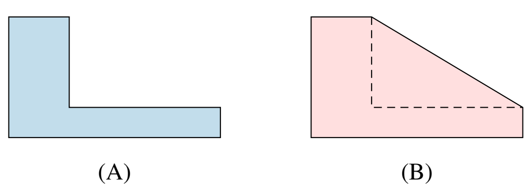


Figure 3.18: (B) Convex hull of a (A) non-convex set

Within the context of linear programming the *convex hull relaxation* of a non-convex set $\mathcal{S}$ corresponds to the set of linear constraints characterising the convex hull of $\mathcal{S}$.

3.7.65 Conway packing problem [2 CONS]

- DIFFN,
- GEOST.

Denotes that a constraint can be used for solving the *Conway packing* problem, which consists of placing 6 orthotopes of size $4 \times 2 \times 1$, 6 orthotopes of size $3 \times 2 \times 2$ and 5 unit cubes within a $5 \times 5 \times 5$ cube. Figure 3.19 shows a solution to the Conway packing problem.

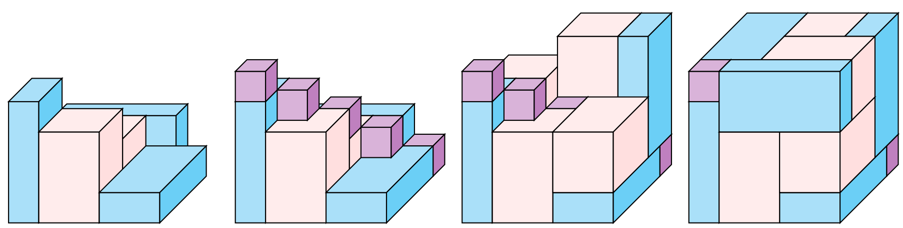


Figure 3.19: A solution to the Conway packing problem

3.7.66 Core [11 CONS]

- ALLDIFFERENT,
- CUMULATIVE,
- CYCLE,
- DIFFN,
- DISJUNCTIVE,
- ELEMENT (see also ELEM for the usage),
- GLOBAL_CARDINALITY,
- GLOBAL_CARDINALITY_WITH_COSTS,
- MINIMUM_WEIGHT_ALLDIFFERENT,
- NVALUE,
- SORT.

Denotes that a global constraint is an important constraint. In fact many constraints can been seen as variations or extensions around one of the following notions:

- The notion of all different forces a set of domain variables to be assigned distinct values. Given a set of domain variables $\{v_1, v_2, \ldots, v_n\}$, the ALLDIFFERENT$(\langle v_1, v_2, \ldots, v_n\rangle)$ imposes such a condition. For example, the ground instance ALLDIFFERENT$(\langle 3, 8, 2, 1\rangle)$ is satisfied, while ALLDIFFERENT$(\langle 1, 8, 2, 1\rangle)$ is not, since value 1 is assigned twice.

- The notion of functional dependency states that a domain variable depends directly of another domain variable. A functional dependency can either be defined in *intention* or in *extension*.

  - On the one hand, functional dependencies defined by intension are usually associated with numerical constraints such as, for example, ABS_VALUE(y, x) which forces the condition $y = |x|$. They can also be associated with global constraints that mention a characteristic that is computed from one or several collections of variables. This is the case, for example, for the NVALUE$(y, \langle x_1, x_2, \ldots, x_n\rangle)$ constraint which forces y to be equal to the number of distinct values assigned to $x_1, x_2, \ldots, x_n$.
  - On the other hand, functional dependencies defined by extension are more general since they allow representing any kind of functional dependency. The ELEMENT(x, t, y) constraint allows expressing that a variable y is determined by a variable x via a table of integers t, i.e., $y = t[x]$. For example, the ground instance ELEMENT$(2, \langle 3, 8, 3, 1\rangle, 8)$ is satisfied since 8 is equal to the second entry of the table $3, 8, 3, 1$. Typical usages of the ELEMENT constraint are, for example:
    * Representing a numerical constraint that is not available in a solver, e.g. a non-linear constraint like $y = x^3$ (see first item of the **Usage** slot of the ELEM constraint).
    * Expressing the link between a discrete choice and its corresponding choice (see second item of the **Usage** slot of the ELEM constraint).

Both, the ELEMENT and the ALLDIFFERENT constraints, are the most commonly used global constraints. Many core global constraints can be seen as an extension of the ALLDIFFERENT$(\langle x_1, x_2, \ldots, x_n\rangle)$ constraint along one of the two following lines:

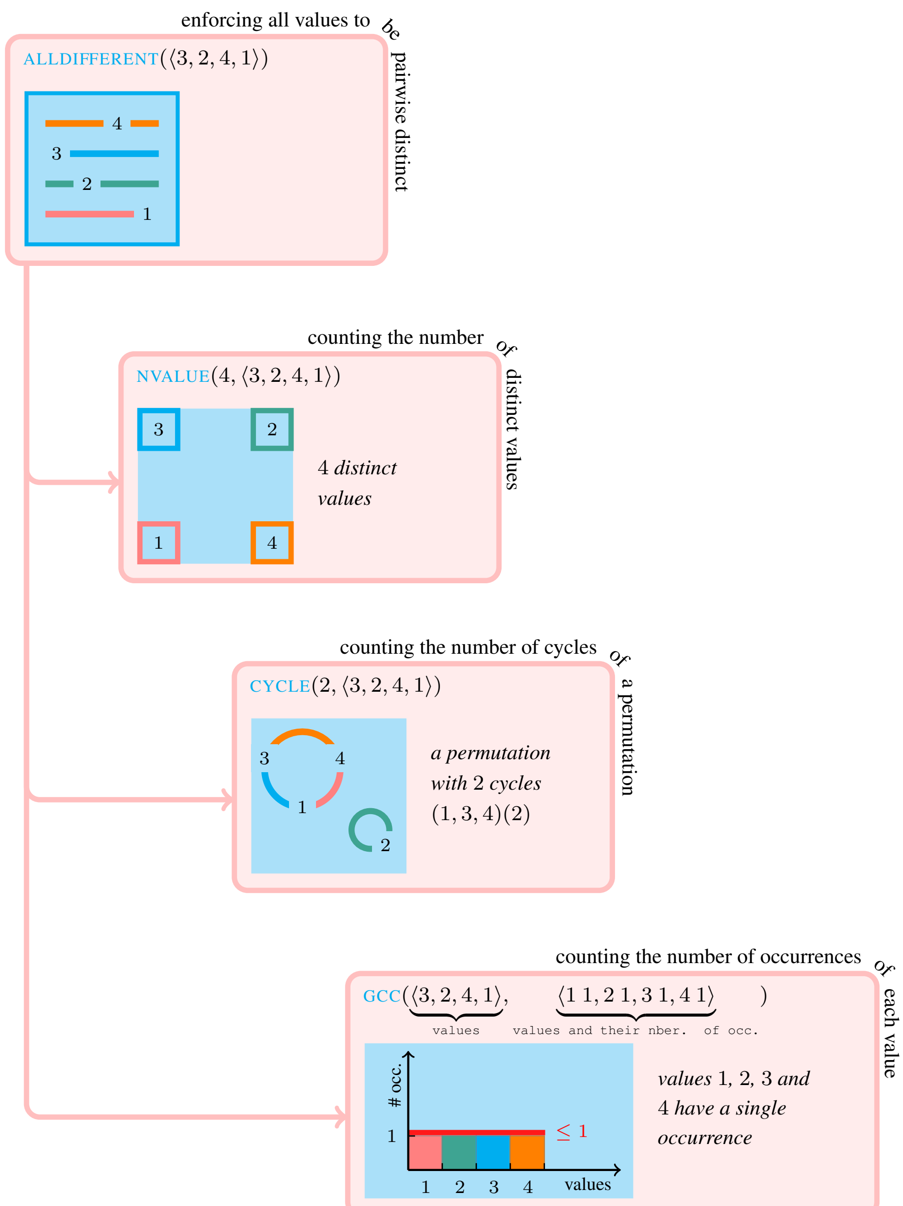


Figure 3.20: Three counting based generalisations of the ALLDIFFERENT constraint: the NVALUE, the CYCLE and the GLOBAL_CARDINALITY (i.e., GCC) constraints; the same example ALLDIFFERENT($\langle 3, 2, 4, 1\rangle$) is reinterpreted with respect to the three generalisations

- In the first line we replace the fact that each value should not be used more than once by some more involved *counting constraints* like:
  - Counting the *total number of actually used distinct values* like the NVALUE$(y, \langle x_1, x_2, \dots, x_n \rangle)$ constraint which forces y to be be equal to the number of distinct values assigned to $x_1, x_2, \dots, x_n$. When y is set to the total number of variables, i.e. $y = n$, NVALUE$(n, \langle x_1, x_2, \dots, x_n \rangle)$ and ALLDIFFERENT$(\langle x_1, x_2, \dots, x_n \rangle)$ are equivalent.
  - Counting the *number of cycles of a permutation*, i.e. we assume that the values assigned to variables $x_1, x_2, \dots, x_n$ belong to interval $[1, n]$, like the CYCLE$(y, \langle x_1, x_2, \dots, x_n \rangle)$ constraint. When (1) y is unconstrained, i.e. it can take any value in $[1, n]$, and when (2) all variables $x_1, x_2, \dots, x_n$ belong to $[1, n]$, CYCLE$(y, \langle x_1, x_2, \dots, x_n \rangle)$ and ALLDIFFERENT$(\langle x_1, x_2, \dots, x_n \rangle)$ are equivalent.
  - Counting the *number of occurrences of each assigned value* like the GLOBAL_CARDINALITY$(\langle x_1, x_2, \dots, x_n \rangle, \langle v_1\ o_1, v_2\ o_2, \dots, v_m\ o_m \rangle)$ constraint, which forces each value v_i $(1 \leq i \leq m)$ to be assigned to exactly o_i variables of $x_1, x_2, \dots, x_n$. When (1) all the occurrence variables $o_1, o_2, \dots, o_m$ are 0-1 variables, and when (2) all variables $x_1, x_2, \dots, x_n$ can only be assigned values in $\{v_1, v_2, \dots, v_m\}$, GLOBAL_CARDINALITY$(\langle x_1, x_2, \dots, x_n \rangle, \langle v_1\ o_1, v_2\ o_2, \dots, v_m\ o_m \rangle)$ and ALLDIFFERENT$(\langle x_1, x_2, \dots, x_n \rangle)$ are equivalent. When in addition (3) $m = n$ and $o_i = 1$ for all i we have a bijection between variables and values.
- In the second line we *generalise the disequality between two variables* in some way like:
  - We replace the disequality between two variables by a *non-overlapping condition between two tasks* where a task t_i is defined by its origin o_i and its duration d_i. The disequality between two variables is changed to a disjunction stating that a task ends before the start of another task or vice versa. This leads to the DISJUNCTIVE$(\langle o_1\ d_1, o_2\ d_2, \dots, o_n\ d_n \rangle)$ constraint.
  - We replace the disequality between two variables by a *non-overlapping condition between two orthotopes* where each orthotope $orth_i$ is defined by the coordinates of its origin and by its sizes. Two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap, i.e. the disequality between two variables is changed to a disjunction with a number of alternatives that is equal to two times the number of dimensions. This leads to the DIFFN constraint.
  - We replace
    - a variable of the ALLDIFFERENT constraint by a *task with a start, a duration, an end and a height attributes*, and
    - the disequality between two variables by *the condition that the sum of the heights of the tasks that overlap a given time point should not exceed a given limit*.

This leads to the CUMULATIVE constraint.

3.7.67 Costas arrays [1 CONS]

- ALLDIFFERENT.

A constraint that allows for expressing the *Costas arrays* problem. A *Costas array* is a permutation $p_1, p_2, \ldots, p_n$ of n integers $1, 2, \ldots, n$ such that $\forall \delta \in [1, n-2], \forall i \in [1, n-\delta-1], \forall j \in [i+1, n-\delta] : p_i - p_{i+\delta} \neq p_j - p_{j+\delta}$. A. Vellino compares in [441] three approaches respectively using Prolog, Pascal and CHIP for solving the *Costas arrays* problem. In fact the weaker formulation $\forall \delta \in [1, \lfloor \frac{n-1}{2} \rfloor], \forall i \in [1, n-\delta-1], \forall j \in [i+1, n-\delta] : p_i - p_{i+\delta} \neq p_j - p_{j+\delta}$ was shown to be equivalent to the original one in [119].

3.7.68 Cost filtering constraint [5 CONS]

- COND_LEX_COST,
- GLOBAL_CARDINALITY_WITH_COSTS,
- MINIMUM_WEIGHT_ALLDIFFERENT,
- SUM_OF_WEIGHTS_OF_DISTINCT_VALUES,
- WEIGHTED_PARTIAL_ALLDIFF.

A constraint that has a set of decision variables as well as a cost variable and for which there exists a filtering algorithm that restricts the state variables from the minimum or maximum value of the cost variable.

3.7.69 Cost matrix [2 CONS]

- GLOBAL_CARDINALITY_WITH_COSTS,
- MINIMUM_WEIGHT_ALLDIFFERENT.

A constraint for which a first argument corresponds to a collection of variables Vars, a second argument to a cost matrix M, and a third argument to a cost variable C. Let Vals denotes the set of values that can be assigned to the variables of Vars. The cost matrix defines for each pair v, u ($v \in$ Vars, $u \in$ Vals) an elementary cost, which is used for computing C when value u is assigned to variable v.

3.7.70 Counting constraint [39 CONS]

- AMONG,
- AMONG_DIFF_0,
- AMONG_INTERVAL,
- AMONG_LOW_UP,
- AMONG_MODULO,
- AMONG_VAR,
- ATLEAST_NVALUE,
- ATLEAST_NVECTOR,
- ATMOST_NVALUE,
- ATMOST_NVECTOR,
- COUNT,
- COUNTS,
- DISCREPANCY,
- EXACTLY,
- GLOBAL_CARDINALITY,
- GLOBAL_CARDINALITY_LOW_UP,
- INCREASING_NVALUE,
- INCREASING_NVALUE_CHAIN,
- LENGTH_FIRST_SEQUENCE,
- LENGTH_LAST_SEQUENCE,
- MAX_NVALUE,
- MIN_NVALUE,
- NCLASS,
- NEQUIVALENCE,
- NINTERVAL,
- NPAIR,
- NVALUE,
- NVALUE_ON_INTERSECTION,
- NVALUES,
- NVALUES_EXCEPT_0,
- NVECTOR,
- NVECTORS,
- OPEN_AMONG,
- OPEN_GLOBAL_CARDINALITY,
- OPEN_GLOBAL_CARDINALITY_LOW_UP,
- ORDERED_ATLEAST_NVECTOR,
- ORDERED_ATMOST_NVECTOR,
- ORDERED_NVECTOR,
- ROOTS.

A constraint restricting the number of occurrences of some values (respectively some pairs of values) within a given collection of domain variables (respectively pairs of domain variables).

3.7.71 Cumulative longest hole problems [1 CONS]

- CUMULATIVE.

A constraint that can use some filtering based on the *longest closed and open hole* problems [41]. We follow the presentation from the previous paper. Before presenting the longest closed open hole scheduling problems, let us first introduce some notation related to the CUMULATIVE$(\mathtt{TASKS}, \mathtt{LIMIT})$ constraint that will be used within the context of the longest closed and open hole problems.

Here, TASKS is a collection of tasks, and for a task $t \in \mathtt{TASKS}$, $t.\mathtt{origin}$, $t.\mathtt{duration}$ and $t.\mathtt{height}$ denote respectively its start, duration and height, while $\mathtt{LIMIT} \in \mathbb{Z}^+$ is the height of the resource. The constraint is equivalent to finding an assignment $s : \mathtt{TASKS.origin} \rightarrow \mathbb{Z}^+$[10] that solves the *cumulative placement* of TASKS of maximum height LIMIT, i.e.:

$$\forall i \in \mathbb{Z} : \sigma_s(i) = \mathtt{LIMIT} - P(\mathtt{TASKS}, i) \geq 0$$

where the *coverage* $P(\mathtt{TASKS}, i)$ by TASKS of instant $i \in \mathbb{Z}$ is:

$$P(\mathtt{TASKS}, i) = \sum_{t \in \mathtt{TASKS} | t.\mathtt{origin} \leq i < t.\mathtt{origin} + t.\mathtt{duration}} t.\mathtt{height}$$

We are now in position to define the longest closed and open hole problems. Given a quantity $\sigma \in \mathbb{Z}^+$ of *slack* (i.e. the difference between the available space and the total area of the tasks to place), the *longest closed hole problem* is to find the largest integer $lcmax_\sigma^{\mathtt{LIMIT}}(\mathtt{TASKS})$ for which there exists a cumulative placement s of a subset of tasks $\mathtt{TASKS}' \subseteq \mathtt{TASKS}$ of maximum height LIMIT, such that the resource area that is not occupied by s on interval $[0, lcmax_\sigma^{\mathtt{LIMIT}})$ does not exceed the maximum allowed slack value σ:

$$\sum_{i=0}^{lcmax_\sigma^{\mathtt{LIMIT}}-1} \sigma_s(i) \leq \sigma.$$

The *longest open hole problem* is to find the largest integer $lmax_\sigma^{\mathtt{LIMIT}}(\mathtt{TASKS})$ for which there exists a cumulative placement s of a subset of tasks $\mathtt{TASKS}' \subseteq \mathtt{TASKS}$ of

[10] Without loss of generality we assume the earliest start of each task to be greater than or equal to 0.

maximum height $\mathtt{LIMIT}$ and an interval $[i', i' + lmax_{\sigma}^{\mathtt{LIMIT}}) \subset \mathbb{Z}$ of length $lmax_{\sigma}^{\mathtt{LIMIT}}$, such that the resource area that is not occupied by s on $[i', i' + lmax_{\sigma}^{\mathtt{LIMIT}})$ does not exceed the maximum allowed slack value σ:

$$\sum_{i=i'}^{i'+lmax_{\sigma}^{\mathtt{LIMIT}}-1} \sigma_s(i) \leq \sigma.$$

As an example, consider seven tasks of respective size 11×11, 9×9, 8×8, 7×7, 6×6, 4×4, 2×2. Part (A) of Figure 3.21 provides a cumulative placement corresponding to the longest open hole problem according to $\mathtt{LIMIT} = 11$ and $\sigma = 0$. The longest open hole $lmax_0^{11}(\{11 \times 11, 9 \times 9, 8 \times 8, 7 \times 7, 6 \times 6, 4 \times 4, 2 \times 2\}) = 17$ since:

- The task 8×8 cannot contribute since a gap of 3 cannot be filled by the unique candidate the task 2×2.
- The task 6×6 can also not contribute since a gap of 5 cannot be completely filled by the candidates 4×4 and 2×2.

The longest close hole $lcmax_0^{11}(\{11 \times 11, 9 \times 9, 8 \times 8, 7 \times 7, 6 \times 6, 4 \times 4, 2 \times 2\}) = 15$: it corresponds to the longest time interval on which the resource is saturated by the illustrated placement and such that one bound of the interval does not intersect any tasks.

Second, consider a task of size 3×2. Part (B) of Figure 3.21 provides a cumulative placement corresponding to the longest open hole problem according to $\epsilon = 11$ and $\sigma = 20$. The longest open hole $lmax_{20}^{11}(\{3 \times 2\}) = 2$.

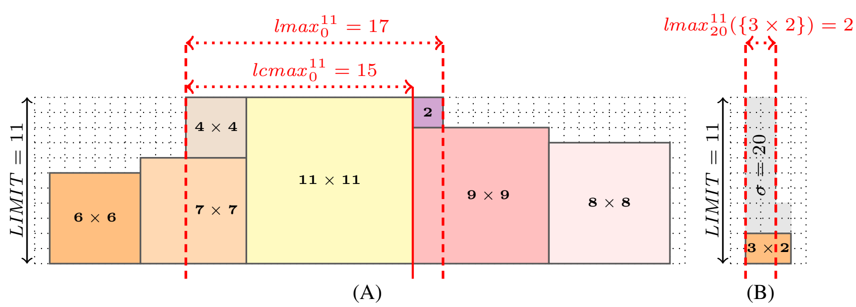


Figure 3.21: Two examples for illustrating the longest *open hole problem*: (A) a first instance with seven tasks of size 11×11, 9×9, 8×8, 7×7, 6×6, 4×4, 2×2 with a slack $\sigma = 0$ and a gap of 11, (B) a second instance with a single task of size 3×2 with a slack $\sigma = 20$ and a gap of 11.

Figure 3.22 provides examples of the longest closed hole when we have 15 squares of sizes $1, 2, \ldots, 15$ and a zero slack. Parts (A), (B),...,(O) respectively give a solution achieving the longest closed hole for a gap of $1, 2, \ldots, 15$. For comparison, Figure 3.23 provides the same examples of the longest open hole with zero slack.

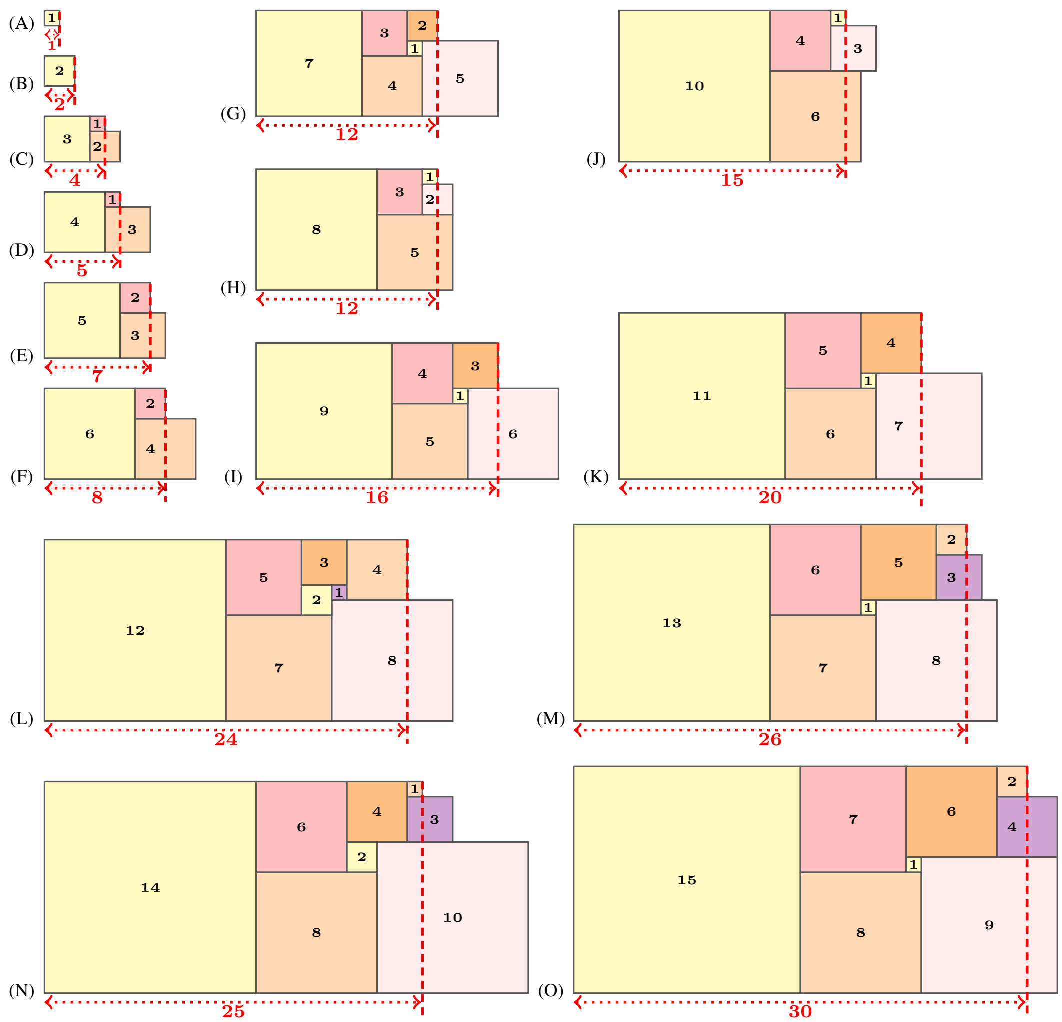


Figure 3.22: Given 15 tasks of sizes $1 \times 1, 2 \times 2, \ldots, 15 \times 15$ and a slack $\sigma = 0$, examples of *longest closed holes* (in red) for a gap of $1, 2, \ldots, 15$

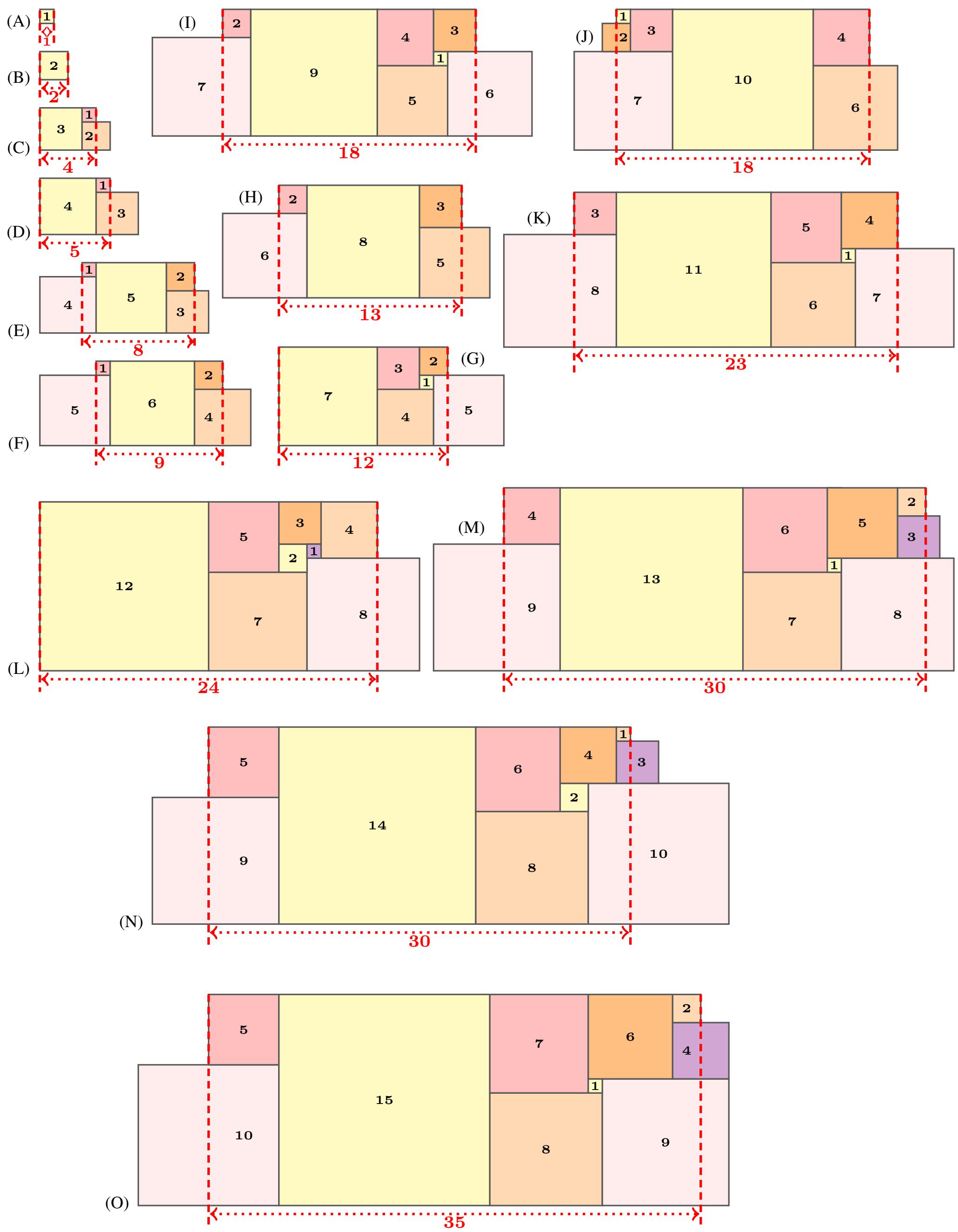


Figure 3.23: Given 15 tasks of sizes $1 \times 1, 2 \times 2, \ldots, 15 \times 15$ and a slack $\sigma = 0$, examples of *longest open holes* (in red) for a gap of $1, 2, \ldots, 15$

3.7.72 Cycle [4 CONS]

- BALANCE_CYCLE,
- CYCLE,
- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_LOOP.

A constraint that can be used for restricting the number of cycles of a permutation (i.e., CYCLE), or for restricting the size of the cycles of a permutation (i.e., SYMMETRIC_ALLDIFFERENT, SYMMETRIC_ALLDIFFERENT_LOOP), or for restricting the difference between the largest and the smallest cycle (i.e., BALANCE_CYCLE).

3.7.73 Cyclic [4 CONS]

- CIRCULAR_CHANGE,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- STRETCH_CIRCUIT.

A constraint that involves a kind of cyclicity in its definition. It either uses the arc generator *CIRCUIT* or an arc constraint involving $\bmod$.

3.7.74 Data constraint [18 CONS]

- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENTN,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_MATRIX,
- ELEMENT_PRODUCT,
- ELEMENT_SPARSE,
- ELEMENTS,
- ELEMENTS_ALLDIFFERENT,
- ELEMENTS_SPARSE,
- IN_RELATION,
- ITH_POS_DIFFERENT_FROM_0,

- NEXT_ELEMENT,
- NEXT_GREATER_ELEMENT,
- STAGE_ELEMENT,
- SUM.

In the literature also known as *ad-hoc constraints*. A constraint that allows for representing an access to an element of a data structure (e.g., a table, a matrix, a relation) or to compute a value from a given data structure.

3.7.75 Deadlock breaking [1 CONS]

- CUTSET.

A constraint that was used within the application area of *deadlock breaking*.

3.7.76 Decomposition [48 CONS]

- ALL_MIN_DIST,
- ALL_DIFFER_FROM_AT_LEAST_K_POS,
- ALL_DIFFER_FROM_AT_MOST_K_POS,
- ALL_DIFFER_FROM_EXACTLY_K_POS,
- ALL_INCOMPARABLE,
- AMONG_SEQ,
- ARITH,
- ARITH_OR,
- ARITH_SLIDING,
- DECREASING,
- DIFFN,
- DIFFN_COLUMN,
- DIFFN_INCLUDE,
- DISJ,
- DISJUNCTIVE,
- DISJUNCTIVE_OR_SAME_END,
- DISJUNCTIVE_OR_SAME_START,
- DOMAIN_CONSTRAINT,
- GEOST,
- GEOST_TIME,
- INCREASING,
- K_ALLDIFFERENT,
- K_DISJOINT,
- K_SAME,
- K_SAME_INTERVAL,

- K_SAME_MODULO,
- K_SAME_PARTITION,
- K_USED_BY,
- K_USED_BY_INTERVAL,
- K_USED_BY_MODULO,
- K_USED_BY_PARTITION,
- LEX_ALLDIFFERENT,
- LEX_CHAIN_GREATER,
- LEX_CHAIN_GREATEREQ,
- LEX_CHAIN_LESS,
- LEX_CHAIN_LESSEQ,
- LINK_SET_TO_BOOLEANS,
- ORTH_LINK_ORI_SIZ_END,
- PRECEDENCE,
- ROOTS,
- SEQUENCE_FOLDING,
- SLIDING_DISTRIBUTION,
- SLIDING_SUM,
- STRICTLY_DECREASING,
- STRICTLY_INCREASING,
- SYMMETRIC_CARDINALITY,
- SYMMETRIC_GCC,
- VISIBLE.

A constraint for which the catalogue provides a description in terms of a conjunction of more elementary constraints. This is the case when the constraint is described by one or several graph constraints that all satisfy the following property: the description uses the **NARC** graph property and forces all arcs of the initial graph to belong to the final graph. Most of the time we have only a single graph constraint. But some constraints (e.g., DIFFN) use more than one. Note that the arc constraint can sometimes be a logical expression involving several constraints (e.g., DOMAIN_CONSTRAINT).

3.7.77 Decomposition-based violation measure [2 CONS]

- SOFT_ALLDIFFERENT_CTR,
- SOFT_ALL_EQUAL_MIN_CTR.

A soft constraint associated with a constraint that can be described in terms of a conjunction of more elementary constraints for which the violation cost is the number of violated elementary constraints.

3.7.78 Demand profile [3 CONS]

- CUMULATIVES,
- SAME_AND_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP.

A constraint that allows for representing problems where one has to allocate resources in order to cover a given demand. A profile specifies for each instant the minimum, and possibly maximum, required demand.

3.7.79 Degree of diversity of a set of solutions [3 CONS]

- LEX_CHAIN_GREATER,
- LEX_CHAIN_LESS,
- SOFT_ALLDIFFERENT_CTR.

A constraint that allows finding a set of solutions with a certain degree of diversity. As an example, consider the problem of finding 9 diverse solutions for the 10-queens problem. For this purpose we create a 10 by 9 matrix $\mathcal{M}$ of domain variables taking their values in interval $[0, 9]$. Each row of $\mathcal{M}$ corresponds to a solution to the 10-queens problem. We assume that the variables of $\mathcal{M}$ are assigned row by row, and that within a given row, they are assigned from the first to the last column. Moreover values are tried in increasing order. We first post for each row of $\mathcal{M}$ the 3 ALLDIFFERENT constraints related to the 10-queens problem (see Figure 5.32 for an illustration of the 3 ALLDIFFERENT constraints). With a LEX_CHAIN_LESS constraint, we lexicographically order the first two variables of each row of $\mathcal{M}$ in order to enforce that the first two variables of any pair of solutions are always distinct. We then impose a SOFT_ALLDIFFERENT_CTR constraint on the variables of each column of $\mathcal{M}$. Let C_i denotes the corresponding cost variable associated with the SOFT_ALLDIFFERENT_CTR constraint of the i-th column of $\mathcal{M}$ (i.e., the first argument of the SOFT_ALLDIFFERENT_CTR constraint). We put a maximum limit (e.g., 3 in our example) on these cost variables. We also impose that the sum of these cost variables should not exceed a given maximum value (e.g., 8 in our example). Finally, in order to balance the diversity over consecutive variables we state that the sum of two consecutive cost variables should not exceed a given threshold (e.g., 2 in our example). As one of the possible results we get the following nine solutions depicted below.

- $S_1 = \langle 0, 2, 5, 7, 9, 4, 8, 1, 3, 6\rangle$,
- $S_2 = \langle 0, 3, 5, 8, 2, 9, 7, 1, 4, 6\rangle$,
- $S_3 = \langle 1, 3, 7, 2, 8, 5, 9, 0, 6, 4\rangle$,
- $S_4 = \langle 2, 4, 8, 3, 9, 6, 1, 5, 7, 0\rangle$,
- $S_5 = \langle 3, 6, 9, 1, 4, 7, 0, 2, 5, 8\rangle$,
- $S_6 = \langle 5, 9, 2, 6, 3, 1, 8, 4, 0, 7\rangle$,
- $S_7 = \langle 6, 8, 1, 5, 0, 2, 4, 7, 9, 3\rangle$,
- $S_8 = \langle 8, 1, 4, 9, 7, 0, 3, 6, 2, 5\rangle$,
- $S_9 = \langle 9, 5, 0, 4, 1, 8, 6, 3, 7, 2\rangle$.

The costs associated with the SOFT_ALLDIFFERENT_CTR constraints of columns $1, 2, \ldots, 10$ are respectively equal to 1, 1, 1, 0, 1, 0, 1, 1, 1, and 1. The different types of constraints between the previous 9 solutions are illustrated by Figure 3.24. The nine diverse solutions $S_1, S_2, \ldots, S_9$ are shown by Figure 3.25. Figure 3.26 depicts the distribution of all the queens of the nine solutions on a unique chessboard.

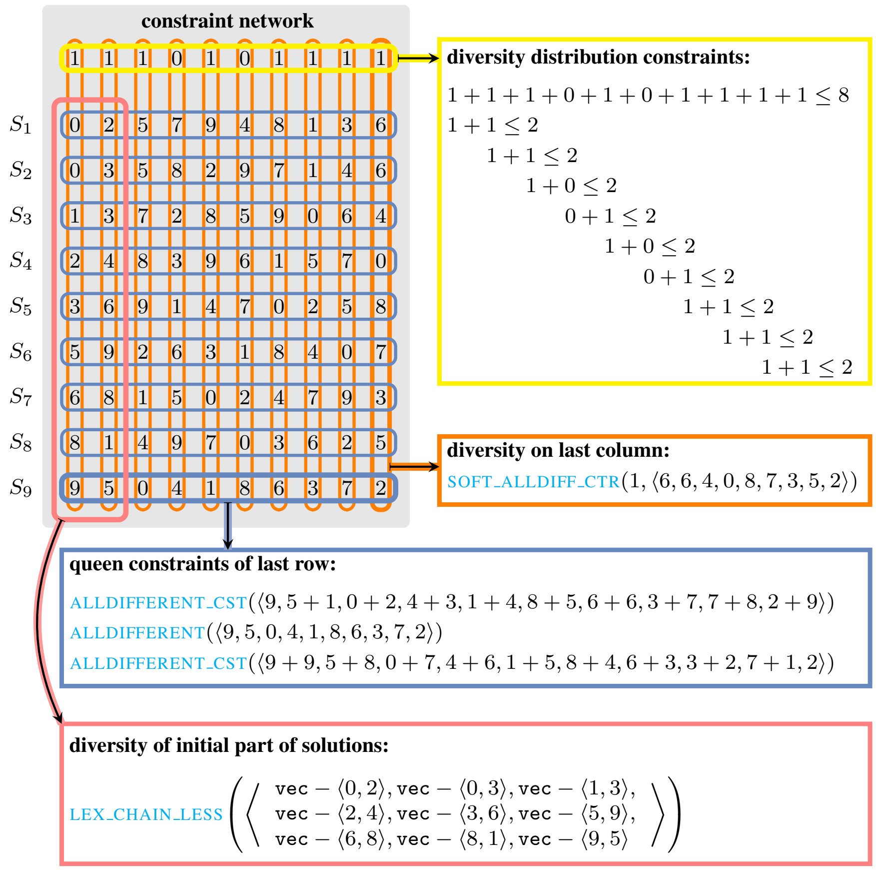


Figure 3.24: Constraint network associated with the problem of finding 9 diverse solutions for the 10-queens problem where variables are fixed to the solutions corresponding to $S_1 = \langle 0,2,5,7,9,4,8,1,3,6\rangle$, $S_2 = \langle 0,3,5,8,2,9,7,1,4,6\rangle$, $\dots$, $S_9 = \langle 9,5,0,4,1,8,6,3,7,2\rangle$, and where each type of constraint (hyperedge) is drawn with its own colour

Approaches for finding diverse and similar solutions based on the Hamming distances between each pair of solutions are presented by E. Hebrard *et al.* [215].

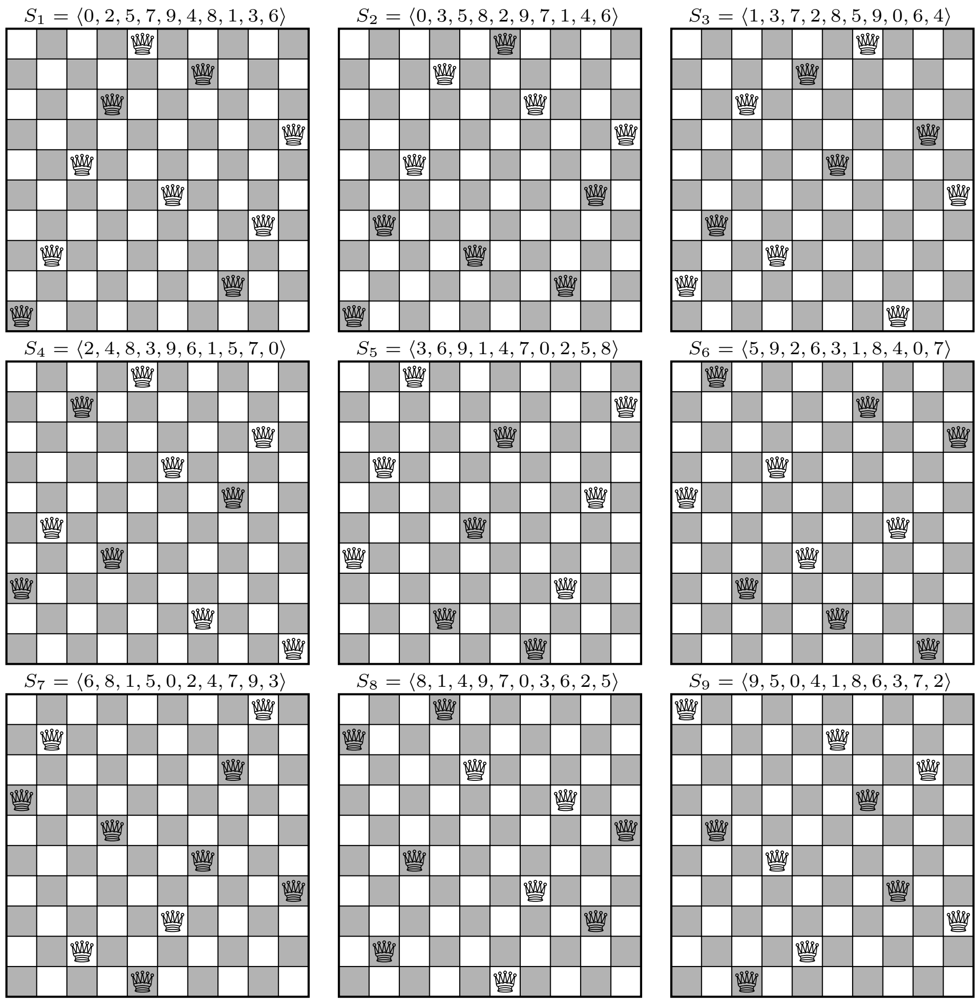
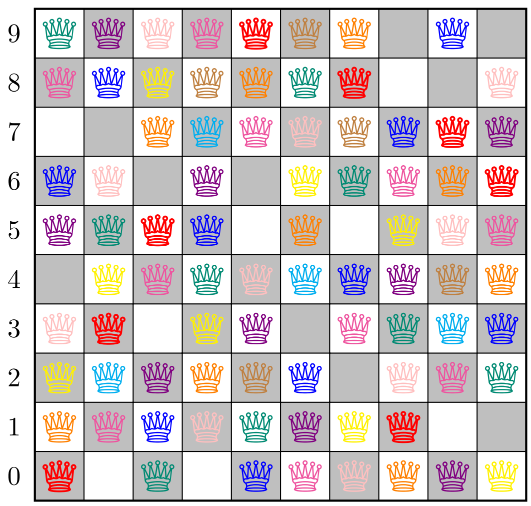


Figure 3.25: Nine diverse solutions to the 10-queens problem



$S_9 = \langle 9, 5, 0, 4, 1, 8, 6, 3, 7, 2 \rangle$

$S_8 = \langle 8, 1, 4, 9, 7, 0, 3, 6, 2, 5 \rangle$

$S_7 = \langle 6, 8, 1, 5, 0, 2, 4, 7, 9, 3 \rangle$

$S_6 = \langle 5, 9, 2, 6, 3, 1, 8, 4, 0, 7 \rangle$

$S_5 = \langle 3, 6, 9, 1, 4, 7, 0, 2, 5, 8 \rangle$

$S_4 = \langle 2, 4, 8, 3, 9, 6, 1, 5, 7, 0 \rangle$

$S_3 = \langle 1, 3, 7, 2, 8, 5, 9, 0, 6, 4 \rangle$

$S_2 = \langle 0, 3, 5, 8, 2, 9, 7, 1, 4, 6 \rangle$

$S_1 = \langle 0, 2, 5, 7, 9, 4, 8, 1, 3, 6 \rangle$

Figure 3.26: Distribution of the queens on the chessboard of the nine diverse solutions depicted by Figure 3.25 to the 10-queens problem: a red queen means two queens from two different solutions that are placed on a same cell, non-red queens of a same colour are queens that belong to a same solution; out of the $10 \cdot 10$ cells of the original chessboard, $9 \cdot 10 - 2 \cdot 8 = 74$ cells are occupied by a single queen, 8 by two queens, and 18 by no queen at all.

3.7.80 Derived collection [30 CONS]

- ASSIGN_AND_COUNTS,
- CORRESPONDENCE,
- CUMULATIVE_TWO_D,
- CUMULATIVE_WITH_LEVEL_OF_PRIORITY,
- CUMULATIVES,
- CYCLE_RESOURCE,
- DOMAIN_CONSTRAINT,
- ELEMENT,
- ELEMENT_MATRIX,
- ELEMENT_SPARSE,
- ELEMENTS_SPARSE,
- GOLOMB,
- IN,
- IN_INTERVAL,
- IN_RELATION,
- IN_SAME_PARTITION,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LINK_SET_TO_BOOLEANS,
- MINIMUM_GREATER_THAN,
- NEXT_ELEMENT,
- NEXT_GREATER_ELEMENT,
- NOT_IN,
- SLIDING_TIME_WINDOW_FROM_START,
- SORT_PERMUTATION,
- TRACK,
- TREE_RESOURCE,
- TWO_LAYER_EDGE_CROSSING.

A constraint that uses one or several derived collections. Derived collections were introduced in Section 2.3.2 on page 51.

3.7.81 DFS-bottleneck [18 CONS]

- ALLDIFFERENT (filtering DFS-bottleneck),
- BALANCE_CYCLE (filtering and reformulation DFS-bottleneck),
- BALANCE_PATH (filtering and reformulation DFS-bottleneck),
- BALANCE_TREE (filtering and reformulation DFS-bottleneck),
- BIPARTITE (filtering and reformulation DFS-bottleneck),
- CIRCUIT (filtering and reformulation DFS-bottleneck),
- CYCLE (filtering and reformulation DFS-bottleneck),
- CONNECTED (filtering and reformulation DFS-bottleneck),
- DERANGEMENT (filtering DFS-bottleneck),
- GLOBAL_CARDINALITY (filtering DFS-bottleneck),
- GLOBAL_CARDINALITY_LOW_UP (filtering DFS-bottleneck),
- MAP (filtering and reformulation DFS-bottleneck),

- PATH (filtering and reformulation DFS-bottleneck),
- PROPER_CIRCUIT (filtering and reformulation DFS-bottleneck),
- SAME (filtering DFS-bottleneck),
- TOUR (filtering and reformulation DFS-bottleneck),
- TREE (filtering and reformulation DFS-bottleneck),
- USED_BY (filtering DFS-bottleneck).

reformulation DFS-bottleneck A constraint on a graph for which a depth-first search based procedure is normally required for checking whether a ground instance is satisfied or not, e.g., a connectivity constraint. The reformulation of such a constraint as a conjunction of other constraints is usually not easy. A possibility, when each node has a single successor in the ground case, is to use an ELEMENT constraint to express the link between a node and its successor at the price of using a large number of ELEMENT constraints (e.g., see the **Reformulation** slot of the CYCLE constraint).

filtering DFS-bottleneck A constraint for which a depth-first search based algorithm usually constitutes a bottleneck of its filtering algorithm. This is a pity, especially on dense graphs[11], where most of the invocations of the filtering algorithm do not usually bring any new deductions. Motivated by this fact, randomised filtering algorithms were introduced in [243] and [246] in the context of the GLOBAL_CARDINALITY_LOW_UP and ALLDIFFERENT constraints. A second approach is to come up with a probabilistic analysis for predicting whether triggering a given filtering algorithm can produce new deductions. This was done for the bound-consistency algorithm of the ALLDIFFERENT constraint in [89].

[11] A common implementation trick relies on the fact that, quite often on dense graphs, a depth-first search algorithm develops a path (rather than a tree) visiting all vertices, such that one can directly reach (i.e., with a single arc) the first node of the path from the last one (i.e., we have a single strongly connected component). In this context the trick is to stop the depth-first search algorithm as soon as the last node of the path is reached, in order to avoid scanning through all remaining arcs of the graph. When this is the case the complexity of the DFS goes from $O(m)$ down to $O(n)$ where n is the number of vertices and where m is the number of arcs of the graph.

3.7.82 Difference [2 CONS]

- GOLOMB,
- SUM_OF_INCREMENTS.

Denotes that the definition of a constraint involves one or several differences between pairs of variables.

3.7.83 Difference between pairs of variables [2 CONS]

- LEX_ALLDIFFERENT,
- LEX_ALLDIFFERENT_EXCEPT_0.

A constraint that allows expressing that a set of pairs of variables are different. Two pairs of variables (X_1, Y_1) and (X_2, Y_2) are different if and only if $X_1 \neq X_2$ or $Y_1 \neq Y_2$. Constraint LEX_ALLDIFFERENT_EXCEPT_0 ignores pairs for which both components are assigned value 0.

3.7.84 Directed acyclic graph [1 CONS]

- CUTSET.

A constraint that forces the final graph to be a *directed acyclic graph*. A *directed acyclic graph* is a digraph with no path starting and ending at the same vertex.

3.7.85 Disequality [25 CONS]

- ALL_DIFFER_FROM_AT_LEAST_K_POS,
- ALL_DIFFER_FROM_AT_MOST_K_POS,
- ALL_DIFFER_FROM_EXACTLY_K_POS,
- ALLDIFFERENT,
- ALLDIFFERENT_BETWEEN_SETS,
- ALLDIFFERENT_CST,
- ALLDIFFERENT_CONSECUTIVE_VALUES,
- DISJOINT,
- ELEMENTS_ALLDIFFERENT,
- GOLOMB,
- K_ALLDIFFERENT,
- K_DISJOINT,
- LEX_DIFFERENT,
- NEQ_CST,
- NOT_ALL_EQUAL,
- NOT_IN,
- OPEN_ALLDIFFERENT,
- PERMUTATION,
- ROOTS,
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT,
- SIZE_MAX_SEQ_ALLDIFFERENT,
- SOFT_ALLDIFFERENT_CTR,
- SOFT_ALLDIFFERENT_VAR,
- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_LOOP.

Denotes that a disequality between two domain variables, one domain variable and a fixed value, or two set variables is used within the definition of a constraint. Denotes also that the notion of disequality can be used within the informal definition of a constraint. This is the case, for example, for the relaxation of the ALLDIFFERENT constraint (i.e., SOFT_ALLDIFFERENT_CTR, SOFT_ALLDIFFERENT_VAR), which do not strictly enforce a disequality.

3.7.86 Disjunction [12 CONS]

- CASE,
- ARITH_OR,
- CLAUSE_OR,
- DIFFN,
- DISJUNCTIVE,
- DISJUNCTIVE_OR_SAME_END,
- DISJUNCTIVE_OR_SAME_START,
- ELEMENT,
- ELEM,
- GEOST,
- GEOST_TIME,
- OR.

Denotes that a constraint can be used for modelling some kind of disjunction.

3.7.87 Domain channel [1 CONS]

- DOMAIN_CONSTRAINT.

A constraint that allows for making the link between a domain variable V and a set of 0-1 variables $B_1, B_2, \ldots, B_n$. It enforces a condition of the form $V = i \Leftrightarrow B_i = 1$.

3.7.88 Domain definition [6 CONS]

- ARITH,
- DOMAIN,
- IN,
- IN_INTERVAL,
- IN_INTERVALS,
- NOT_IN.

A constraint that is used for defining the initial domain of one or several domain variables or for removing some values from the domain of one or several domain variables.

3.7.89 Dominating queens [1 CONS]

- NVALUE.

A constraint that can be used for modelling the *dominating queens* problem. Place a number of queens on an n by n chessboard in such a way that all squares are either attacked by a queen or are occupied by a queen. A queen can attack all squares located on the same column, on the same row or on the same diagonal. Values of the minimum number of queens for n less than or equal to 120 are reported in [310]. Most of them are in fact either equal to $\lfloor\frac{n+1}{2}\rfloor$ or to $\lfloor\frac{n+1}{2}\rfloor + 1$. Values $n = 3$ and $n = 11$ are the only two values below 120 for which the previous assertion is not true since we only need in these two cases $\lfloor\frac{n}{2}\rfloor$ queens.

3.7.90 Domination [6 CONS]

- ATLEAST_NVECTOR,
- ATMOST_NVECTOR,
- NVALUE,
- NVECTOR,
- NVECTORS,
- SUM_OF_WEIGHTS_OF_DISTINCT_VALUES.

A constraint that can be used for expressing directly the fact that we search for a *dominating set* in an undirected graph. Given an undirected graph $G = (V, E)$ where V is a finite set of vertices and E a finite set of unordered pairs of distinct elements from V, a set S is a *dominating set* if for every vertex $u \in V - S$ there exists a vertex $v \in S$ such that u is adjacent to v. Part (A) of Figure 3.27 gives an undirected graph G, while part (B) depicts a dominating set $S = \{e, f, g\}$ in G.

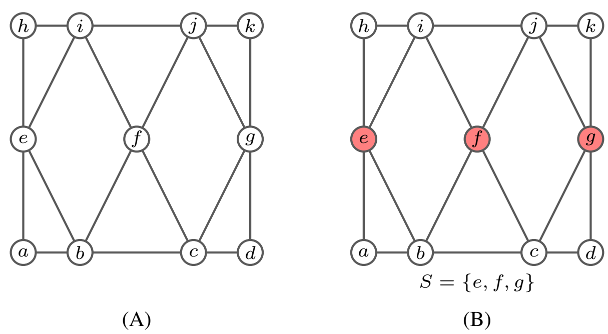


Figure 3.27: (A) A graph and (B) one of its dominating set $S = \{e, f, g\}$

3.7.91 Dual model [5 CONS]

- INVERSE,
- INVERSE_EXCEPT_LOOP,
- INVERSE_OFFSET,
- INVERSE_SET,
- INVERSE_WITHIN_RANGE.

A constraint that can be used as a channelling constraint in a problem where the roles of the variables and the values can be interchanged. This is the case, for example, when we have a bijection between a set of variables and the values they can take.

3.7.92 Duplicated variables [8 CONS]

- GLOBAL_CARDINALITY,
- K_ALLDIFFERENT,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- SCALAR_PRODUCT,
- STRETCH_CIRCUIT.

A constraint for which the situation where the same variable can occur more than once was considered in order to derive a better filtering algorithm or to prove a complexity result for achieving arc-consistency. Also in the case of the STRETCH_CIRCUIT constraint, a constraint for which the reformulation duplicates some variables.

3.7.93 Dynamic programming [5 CONS]

- AMONG_SEQ,
- CHANGE,
- CUMULATIVE,
- STRETCH_CIRCUIT,
- STRETCH_PATH.

A constraint for which a filtering algorithm uses *dynamic programming*. Note that dynamic programming was also used by M. A. Trick within the context of linear constraints [419].

3.7.94 Empty intersection [2 CONS]

- DISJOINT,
- K_DISJOINT.

A constraint which forces an empty intersection between two sets of variables.

3.7.95 Entailment [6 CONS]

- ALLDIFFERENT,
- AMONG_LOW_UP,
- GLOBAL_CARDINALITY_LOW_UP,
- MAXIMUM,
- MINIMUM,
- NOT_IN.

Denotes that the catalogue mentions a sufficient condition for the entailment of a constraint. Consider a constraint $\mathcal{C}(V_1, V_2, \ldots, V_n)$ and the potential sets of values $dom(V_1), dom(V_2), \ldots, dom(V_n)$ that can respectively be assigned to the distinct domain variables $V_1, V_2, \ldots, V_n$. The constraint $\mathcal{C}(V_1, V_2, \ldots, V_n)$ is *entailed* if and only if $\mathcal{C}(V_1, V_2, \ldots, V_n)$ holds whatever values $val_1 \in dom(V_1), val_2 \in dom(V_2), \ldots, val_n \in dom(V_n)$ will respectively be assigned variables $V_1, V_2, \ldots, V_n$. A satisfied constraint for which all variables are already fixed is trivially entailed.

Entailment is usually not considered as very important when designing a filtering algorithm, even though it can sometimes save waking again and again a constraint that will for sure be satisfied. Failure to detect entailment can leads to a memory leak if the constraint system is supposed to reclaim memory for entailed constraints for which it is no more possible to backtrack over the point where the constraint was posted. From a modelling point of view, entailment detection is mandatory for coming up with the reified version of a constraint (see also reified automaton constraint).

3.7.96 Equality [1 CONS]

- EQ_SET.

Denotes that the notion of equality can be used within the informal definition of a constraint.

3.7.97 Equality between multisets [4 CONS]

- K_SAME,
- SAME,
- SAME_AND_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP.

A constraint that can be used for modelling an equality constraint between two multisets.

3.7.98 Equivalence [21 CONS]

- ATLEAST_NVALUE,
- ATLEAST_NVECTOR,
- ATMOST_NVALUE,
- ATMOST_NVECTOR,
- BALANCE_INTERVAL,
- BALANCE_MODULO,
- BALANCE_PARTITION,
- BALANCE,
- INCREASING_NVALUE,
- MAX_NVALUE,
- MIN_NVALUE,
- NCLASS,
- NEQUIVALENCE,
- NINTERVAL,
- NOT_ALL_EQUAL,
- NPAIR,
- NVALUE,
- NVALUES,
- NVECTOR,
- NVECTORS,
- SOFT_ALLDIFFERENT_VAR.

Denotes that a constraint is defined by a graph constraint for which the final graph is reflexive, symmetric and transitive.

3.7.99 Euler knight [4 CONS]

- ALLDIFFERENT,
- CIRCUIT,
- CYCLE,
- TOUR.

Denotes that a constraint can be used for modelling some parts of the *Euler knight problem*. The *Euler knight problem* consists of finding a sequence of moves on a chess-board by a knight such that each square of the board is visited exactly once. While a natural model uses an undirected graph together with the TOUR constraint, the problem is usually modeled with a directed graph that does not require set variables.

3.7.100 Excluded [1 CONS]

- NOT_IN.

A constraint that prevents certain values to be taken by a variable.

3.7.101 Extensible [57 CONS]

- ALL_DIFFER_FROM_AT_LEAST_K_POS (extensible wrt VECTORS.vec),
- AND (extensible wrt VARIABLES when VAR $= 0$),
- ASSIGN_AND_COUNTS (extensible wrt ITEMS when RELOP $\in [\geq, >]$),
- ASSIGN_AND_NVALUES (extensible wrt ITEMS when RELOP $\in [\geq, >]$),
- ATLEAST (extensible wrt VARIABLES),
- ATLEAST_NVALUE (extensible wrt VARIABLES),
- ATLEAST_NVECTOR (extensible wrt VECTORS),
- BETWEEN_MIN_MAX (extensible wrt VARIABLES),
- CLAUSE_AND (extensible wrt POSVARS when VAR $= 0$),
- CLAUSE_AND (extensible wrt NEGVARS when VAR $= 0$),
- CLAUSE_OR (extensible wrt POSVARS when VAR $= 1$),
- CLAUSE_OR (extensible wrt NEGVARS when VAR $= 1$),
- COMPARE_AND_COUNT (extensible wrt [VARIABLES1, VARIABLES2] when COUNT $\in [\geq, >]$),
- COUNT (extensible wrt VARIABLES when RELOP $\in [\geq, >]$),
- COUNTS (extensible wrt VARIABLES when RELOP $\in [\geq, >]$),
- DIFFER_FROM_AT_LEAST_K_POS (extensible wrt [VARIABLES1, VARIABLES2]),
- ELEMENT (suffix-extensible wrt TABLE),
- ELEMENT_PRODUCT (suffix-extensible wrt TABLE),
- ELEMENTN (suffix-extensible wrt TABLE),
- IN (extensible wrt VALUES),
- IN_INTERVALS (extensible wrt INTERVALS),
- IN_RELATION (extensible wrt TUPLES_OF_VALS),
- IN_SAME_PARTITION (extensible wrt PARTITIONS),
- ITH_POS_DIFFERENT_FROM_0 (suffix-extensible wrt VARIABLES),
- LEX_ALLDIFFERENT (extensible wrt VECTORS.vec),
- LEX_CHAIN_LESS (suffix-extensible wrt VECTORS.vec),
- LEX_DIFFERENT (extensible wrt [VECTOR1, VECTOR2]),

- LEX_GREATER (suffix-extensible wrt [VECTOR1, VECTOR2]),
- LEX_LESS (suffix-extensible wrt [VECTOR1, VECTOR2]),
- NAND (extensible wrt VARIABLES when VAR $=1$),
- NCLASS (extensible wrt VARIABLES when NCLASS $=$ |PARTITIONS|),
- NEQUIVALENCE (extensible wrt VARIABLES when NEQUIV $=$ M),
- NOR (extensible wrt VARIABLES when VAR $=0$),
- NOT_ALL_EQUAL (extensible wrt VARIABLES),
- NVALUES (extensible wrt VARIABLES when RELOP $\in [\geq, >]$),
- NVALUES_EXCEPT_0 (extensible wrt VARIABLES when RELOP $\in [\geq, >]$),
- NVECTORS (extensible wrt VECTORS when RELOP $\in [\geq, >]$),
- OPEN_ATLEAST (suffix-extensible wrt VARIABLES),
- OR (extensible wrt VARIABLES when VAR $=1$),
- RANGE_CTR (extensible wrt VARIABLES when CTR $\in [\geq, >]$),
- SCALAR_PRODUCT (extensible wrt LINEARTERM when CTR $\in [\geq, >]$, minval(LINEARTERM.coeff) ≥ 0 and minval(LINEARTERM.var) ≥ 0),
- SOME_EQUAL (extensible wrt VARIABLES),
- STAGE_ELEMENT (suffix-extensible wrt TABLE),
- SUM_CTR (extensible wrt VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) ≥ 0),
- SUM_CTR (extensible wrt VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) ≤ 0),
- SUM_CUBES_CTR (extensible wrt VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) ≥ 0),
- SUM_CUBES_CTR (extensible wrt VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) ≤ 0),
- SUM_POWERS4_CTR (extensible wrt VARIABLES when CTR $\in [\geq, >]$,
- SUM_POWERS5_CTR (extensible wrt VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) ≥ 0),
- SUM_POWERS5_CTR (extensible wrt VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) ≤ 0),
- SUM_POWERS6_CTR (extensible wrt VARIABLES when CTR $\in [\geq, >]$,
- SUM_SQUARES_CTR (extensible wrt VARIABLES when CTR $\in [\geq, >]$,
- USED_BY (extensible wrt VARIABLES1),
- USED_BY_INTERVAL (extensible wrt VARIABLES1),
- USED_BY_MODULO (extensible wrt VARIABLES1),
- USED_BY_PARTITION (extensible wrt VARIABLES1),
- USES (extensible wrt VARIABLES1).

An *extensible* constraint is a constraint for which, given any satisfied ground instance (i.e., a *source* constraint), one can add any item without affecting that the resulting constraint (i.e., a *target* constraint) still holds, assuming all its restrictions holds. All the extensions of contractibility described at the corresponding keyword entry apply also for extensibility. In particular we also have the restricted notions of *prefix-extensible* and *suffix-extensible* constraints, which respectively means that items are added before the first item of a collection or after the last item. As for contractibility, extensibility may also be conditioned by a list of restrictions. Finally extensibility may involve more than one collection. In this context, items are *added simultaneously to all collections from exactly the same set of positions*. We now present different examples of extensible constraints, starting from a very simple one.

- As a first example, consider the ATLEAST(N, VARIABLES, VALUE) constraint, which forces at least N variables of the VARIABLES collection to be assigned value VALUE. We have that ATLEAST is *extensible* with respect to VARIABLES, since adding a variable to an already satisfied instance of ATLEAST preserves the fact the new constraint is satisfied.

  As an illustration consider the source constraint ATLEAST$(2, \langle 4, 2, 4, 5\rangle, 4)$ and the target constraint ATLEAST$(2, \langle 4, 2, 4, 5, 0, 4\rangle, 4)$. Since the first argument N is set to the same value, both in the source and the target constraint, and since the third VALUE is also set to the same value both in the source and the target constraint, we have that ATLEAST$(2, \langle 4, 2, 4, 5\rangle, 4)$ implies ATLEAST$(2, \langle 4, 2, 4, 5, 0, 4\rangle, 4)$.

- As a second example, consider the ELEMENT(INDEX, TABLE, VALUE) constraint, which forces VALUE to equal the INDEXth item of TABLE. We have that ELEMENT is *suffix-extensible* with respect to TABLE, since adding new elements at the end of TABLE for an already satisfied instance of ELEMENT preserves the fact the new constraint is satisfied.

  As an illustration consider the source constraint ELEMENT$(3, \langle 6, 9, 2, 9\rangle, 2)$ and the target constraint ELEMENT$(3, \langle 6, 9, 2, 9, 8, 0, 2\rangle, 2)$. Since the first argument INDEX is set to the same value, both in the source and the target constraint, and since the third argument VALUE is also set to the same value both in the source and the target constraint, we have that ELEMENT$(3, \langle 6, 9, 2, 9\rangle, 2)$ implies ELEMENT$(3, \langle 6, 9, 2, 9, 8, 0, 2\rangle, 2)$.

- As a third example, consider the AND(VAR, VARIABLES) constraint, which forces VAR to equal 1 if all variables of VARIABLES are set to 1, and 0 otherwise. We have that AND is *extensible* with respect to VARIABLES when VAR is equal to 0. This stems from the fact that, given a satisfied instance of AND where VAR $= 0$, adding any new variable to VARIABLES preserves the fact the new constraint is satisfied. As an illustration consider the source constraint AND$(0, \langle 1, 0, 1\rangle)$ and the target constraint AND$(0, \langle 1, 0, 0, 1\rangle)$. Since the first argument VAR is set to 0, both in the source and the target constraint, we have that AND$(0, \langle 1, 0, 1\rangle)$ implies AND$(0, \langle 1, 0, 0, 1\rangle)$.

- As a fourth example, consider the LEX_GREATER(VECTOR1, VECTOR2) constraint, which forces VECTOR1 to be lexicographically strictly greater than VECTOR2. We have that LEX_GREATER is *suffix-extensible* with respect to VECTOR1 and VECTOR2. This means that, given a satisfied instance of LEX_GREATER, adding k items at the end of its first argument VECTOR1 and adding k other items at the end of its second argument VECTOR2 preserves the fact the new constraint is satisfied.

  As an illustration consider the source constraint LEX_GREATER$(\langle 5, 2, 7, 1\rangle, \langle 5, 2, 6, 2\rangle)$ and the target constraint LEX_GREATER$(\langle 5, 2, 7, 1, 0\rangle, \langle 5, 2, 6, 2, 9\rangle)$. We have that LEX_GREATER$(\langle 5, 2, 7, 1\rangle, \langle 5, 2, 6, 2\rangle)$ implies LEX_GREATER$(\langle 5, 2, 7, 1, 0\rangle, \langle 5, 2, 6, 2, 9\rangle)$.

- As a fifth example, consider the LEX_CHAIN_LESS(VECTORS) constraint, which given a collection of vectors each of which defined by a collection of variables of the same length, forces the i^{th} vector to be lexicographically strictly less than the $(i+1)^{th}$ vector $(1 \leq i < |\texttt{VECTORS}|)$. We have that LEX_CHAIN_LESS is *suffix-extensible* with respect to VECTORS.vec. This means that, given a satisfied instance of LEX_CHAIN_LESS, adding k items at the end of all collections *simultaneously* preserves the fact the new constraint is satisfied.

  As an illustration consider the source constraint LEX_CHAIN_LESS$(\langle \texttt{vec} - \langle 5,2,3,9\rangle, \texttt{vec} - \langle 5,2,6,2\rangle, \texttt{vec} - \langle 5,2,6,3\rangle\rangle)$ and the target constraint LEX_CHAIN_LESS$(\langle \texttt{vec}-\langle 5,2,3,9,9\rangle, \texttt{vec}-\langle 5,2,6,2,8\rangle, \texttt{vec}-\langle 5,2,6,3,7\rangle\rangle)$. Since each vector of the source constraint is a prefix of the vector located at the same position in the target constraint the source constraint implies the target constraint.

The keyword contractible introduces a dual notion, where items can be removed from a collection that is passed as an argument of a satisfied global constraint without affecting the fact that the resulting constraint is satisfied. Contractibility is a more common property than extensibility.

3.7.102 Extension [1 CONS]

- IN_RELATION.

A constraint that is defined by explicitly providing all its solutions.

3.7.103 Facilities location problem [2 CONS]

- CYCLE_OR_ACCESSIBILITY,
- SUM_OF_WEIGHTS_OF_DISTINCT_VALUES.

A constraint that allows for modelling a facilities location problem. In a facilities location problem one has to select a subset of locations from a given initial set so that a given set of conditions holds.

3.7.104 Floor planning problem [4 CONS]

- DIFFN,
- GEOST,
- LEX_CHAIN_GREATER,
- LEX_CHAIN_LESS.

A constraint that can be used for the *floor planning problem*. The *floor planning problem* [326, 418, 281, 120, 289] involves various type of spaces, such as the *placement space* itself (i.e., the *floor*), the *rooms* to place within the placement space, and the *circulation* between the rooms. The placement space can be located on a single level or on several levels. Very often the placement space corresponds to a single rectangle and all rooms are rectangles with their borders parallel to the contour of the placement space. Circulation typically corresponds to *corridors* or *stairs* that respectively allow to access from one room to another room or from one level to another level. Within the context of floor planning three main classes of constraints have been identified [290], namely dimensional topological and implicit constraints:

- A *dimensional constraint* usually restricts the length, the width or the surface of *a single space*. Ratio constraints enforce aesthetic proportions between the length and the width of a single space or constraint the surfaces of two closely related spaces such as the toilets and the shower. Dimensional constraints can be expressed by reducing the domain of some variable or by stating some arithmetic constraints between two variables.
- A *topological constraint* imposes a condition between *two spaces*. Typical topological constraints are:
  - *Adjacency constraints with a minimum contact* between a room and a corridor or another room allow expressing that there must be enough place to put a door between two given spaces. In the context of staircases one has

to enforce that fact that the first and last stairs are completely accessible. When a corridor is made up from two parts, one also has to enforce that the two parts are fully in contact.

  - *Adjacency with the contour constraints* between a room and a specified (or not) side of the contour allow expressing the orientation of a room (or just that a room must have some window).
  - *Relative positioning constraints* between two specified rooms allow, for example, expressing the fact that a room is located to the north of another room.
  - *Minimum and maximum distance constraints* between two rooms allow expressing the proximity between two given rooms.

  Topological constraints occur naturally in the preliminary design phase in architecture and can typically be expressed by using reified or global constraints.

- An *implicit constraint* puts a global condition that is inherent to floor planning problems between *all the spaces* of the floor. We typically have:

  - *Inclusion* of each room and circulation within the contour.
  - *Partitioning* of the placement space (i.e., no wasted space is permitted). This is usually a hard constraint which requires specific propagation in order to prevent the creation of wasted space.
  - *Non-overlapping* between rooms.
  - *Symmetry breaking constraints* between identical rooms imposes, for example, a lexicographic order between their respective lower leftmost corners.

  Such constraints can typically be expressed by using global constraints, such as DIFFN, GEOST, or LEX_CHAIN_LESS.

Finally, in order to allocate as much surface as possible to the rooms, one wants sometimes to minimise the total circulation area between the different rooms.

In order to illustrate these constraints we now consider an example of floor planning problem taken from R. Maculet PhD thesis [281] involving 11 spaces. Constraints on the dimensions of these space are:

- The *floor* where to place everything has a size of 12 by 10 meters.
- The *living* has a surface between 33 and 42 square meters and a minimum size of 4 by 4.
- The *kitchen* has a surface between 9 and 15 square meters and a minimum size of 3 by 3.
- The *shower* has a surface between 6 and 9 square meters and a minimum size of 2 by 2.

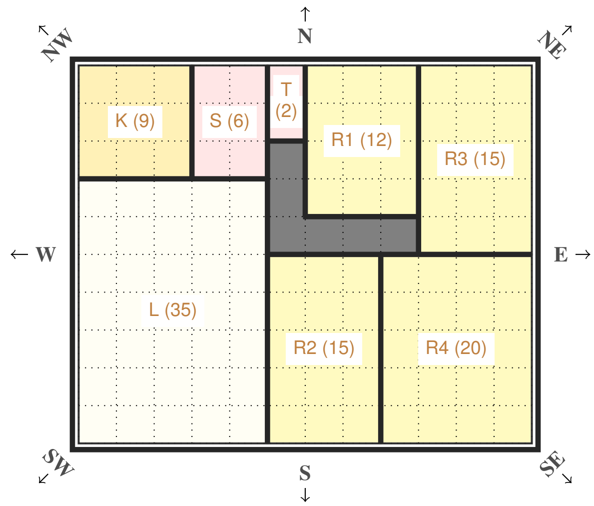


Figure 3.28: A solution to Maculet floor planning problem that minimises the total area of the corridors

- The *toilet* has a surface between 1 and 2 square meters and a minimum size of 1 by 1.
- The *first and second parts of the corridor* have both a surface between 1 and 12 square meters and a minimum size of 1 by 1.
- The *first, second and third rooms* have all a surface between 11 and 15 square meters and a minimum size of 3 by 3.
- The *fourth room* has a surface between 15 and 20 square meters and a minimum size of 3 by 3.

Topological constraints between spaces are:

- The *living* is located on the south-west contour. The *kitchen*, the *first*, *second* and *third rooms* are either located on the south or on the north contour. The *fourth room* is on the south contour.
- All spaces, except the *kitchen*, are adjacent to one of the *corridors* with at least 1 meter of full contact.
- The *kitchen* is adjacent to the *living* and to the *shower*.
- The *toilet* is adjacent to the *kitchen* or to the *shower*.
- The *first* and the *second parts* of the *corridor* are adjacent and fully in contact.

Finally no wasted space is permitted. Figure 3.28 presents a solution to the corresponding floor planning problem that minimises the area of the two corridors.

3.7.105 Flow [17 CONS]

- ALL_BALANCE,
- ALLDIFFERENT,
- AMONG_SEQ,
- GLOBAL_CARDINALITY,
- GLOBAL_CARDINALITY_LOW_UP,
- GLOBAL_CARDINALITY_LOW_UP_NO_LOOP,
- GLOBAL_CARDINALITY_NO_LOOP,
- OPEN_ALLDIFFERENT,
- OPEN_GLOBAL_CARDINALITY,
- OPEN_GLOBAL_CARDINALITY_LOW_UP,
- SAME,
- SAME_AND_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP,
- SLIDING_SUM,
- SYMMETRIC_CARDINALITY,
- SYMMETRIC_GCC,
- USED_BY.

A constraint for which there is a filtering algorithm based on an algorithm that finds a feasible flow in a graph. The graph is usually constructed[12] from the variables of the constraint as well as from their potential values. The usual game is to come up with a flow model such that there exists a one to one correspondence between feasible flows in the flow model and solutions to the constraint, so that detecting arcs that cannot carry any flow in any feasible flow will lead removing some values from the domains of some variables. The next sections provide standard flow models for the ALLDIFFERENT, the OPEN_ALLDIFFERENT, the GLOBAL_CARDINALITY_LOW_UP, the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP, the USED_BY, the SAME, and the SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraints.

A. Flow models for the ALLDIFFERENT and the OPEN_ALLDIFFERENT constraints

Figure 3.29 presents flow models for the ALLDIFFERENT and the OPEN_ALLDIFFERENT constraints. Blue arcs represent feasible flows respectively corresponding to the solutions ALLDIFFERENT($\langle x_1 = 1,\ x_2 = 2,\ x_3 = 3,\ x_4 = 4,\ x_5 = 5\rangle$) and OPEN_ALLDIFFERENT($\{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{5}\}, \langle x_1 = 1,\ x_2 = 2,\ x_3 = 3,\ x_4 = 3,\ x_5 = 4\rangle$), while red arcs correspond to arcs that cannot carry any flow if the constraint has a solution. Tables 3.11 and 3.12 respectively provide the initial domains of the variables we assume for the ALLDIFFERENT and OPEN_ALLDIFFERENT constraints.

- Within the context of the ALLDIFFERENT constraint the assignments $x_3 = 1$, $x_3 = 2$ and $x_4 = 2$ are forbidden since values 1 and 2 must already be assigned to variables x_1 and x_2. Finally the assignments $x_4 = 3$ and $x_5 = 3$ are also forbidden since values 1, 2 and 3 must be assigned to variables x_1, x_2 and x_3.

- Within the context of the OPEN_ALLDIFFERENT constraint, the assignment $x_4 = 3$ does not matter at all since the position of x_4 within $\langle x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 3, x_5 = 4\rangle$ (i.e., position 4) does not belong to the set of variables

[12] Sometimes it is also constructed from the reformulation of a global constraint in term of a conjunction of linear constraints. This is the case, for example, for the AMONG_SEQ and the SLIDING_SUM global constraints.

Table 3.11: Domains of the variables for the ALLDIFFERENT constraint of Figure 3.29.

i	$dom(x_i)$	i	$dom(x_i)$	i	$dom(x_i)$
1	$\{1,2\}$	3	$\{1,2,3\}$	5	$\{3,4,5,6\}$
2	$\{1,2\}$	4	$\{2,3,4,5\}$		

Table 3.12: Domains of the variables for the OPEN_ALLDIFFERENT constraint of Figure 3.29. The lower and upper bounds of the set variable corresponding to the first argument of the OPEN_ALLDIFFERENT constraint are respectively equal to set of variables positions $\{1,2,3,5\}$, and $\{1,2,3,4,5\}$, where $\{\mathbf{1},\mathbf{2},\mathbf{3},4,\mathbf{5}\}$ is a shortcut for denoting both bounds.

i	$dom(x_i)$	i	$dom(x_i)$	i	$dom(x_i)$
1	$\{1,2\}$	3	$\{1,2,3\}$	5	$\{3,4\}$
2	$\{1,2\}$	4	$\{2,3\}$		

positions $\{1,2,3,5\}$. We can only prune according to those variables that for sure should be assigned distinct values. Consequently $x_3 = 1$ and $x_3 = 2$ are forbidden since values 1 and 2 must already be assigned to x_1 and x_2. Finally the assignment $x_5 = 3$ is also forbidden since values 1, 2 and 3 must be assigned to x_1, x_2 and x_3.

B. Flow models for the GLOBAL_CARDINALITY_LOW_UP and the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraints

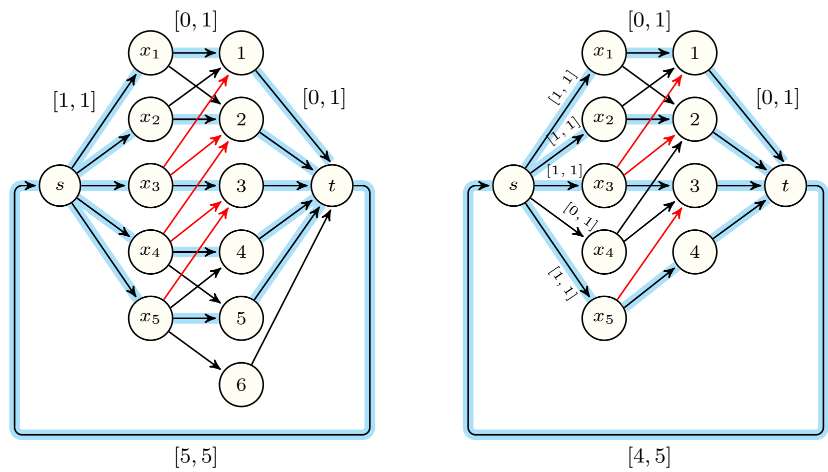
Figure 3.30 presents flow models for the GLOBAL_CARDINALITY_LOW_UP and the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraints. Blue arcs represent feasible

Table 3.13: Domains of the variables and minimum and maximum number of occurrences of each value for the GLOBAL_CARDINALITY_LOW_UP constraint of Figure 3.30.

i	$dom(x_i)$	i	$dom(x_i)$	i	$[omin_i, omax_i]$	i	$[omin_i, omax_i]$
1	$\{1,2\}$	5	$\{1,2,3\}$	1	$[1,2]$	5	$[0,2]$
2	$\{1,2\}$	6	$\{2,3,4,5\}$	2	$[1,2]$		
3	$\{1,2\}$	7	$\{3,5\}$	3	$[1,1]$		
4	$\{1,2\}$			4	$[0,2]$		

$x_1 \in [1,2], x_2 \in [1,2], x_3 \in [1,3],$
$x_4 \in [2,5], x_5 \in [3,6],$
ALLDIFFERENT$(\langle x_1, x_2, x_3, x_4, x_5 \rangle)$.

$x_1 \in [1,2], x_2 \in [1,2], x_3 \in [1,3],$
$x_4 \in [2,3], x_5 \in [3,4],$
$s = \{\mathbf{1}, \mathbf{2}, \mathbf{3}, 4, \mathbf{5}\},$
OPEN_ALLDIFFERENT $(s, \langle x_1, x_2, x_3, x_4, x_5 \rangle)$.



Blue arcs correspond to the feasible flow associated with the solution ALLDIFFERENT$(\langle 1, 2, 3, 4, 5 \rangle)$, *while red arcs cannot carry any flow in a solution.*

Blue arcs correspond to the feasible flow associated with the solution OPEN_ALLDIFFERENT$(\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{5}\}, \langle 1, 2, 3, 3, 4 \rangle)$, *while red arcs cannot carry any flow in a solution.*

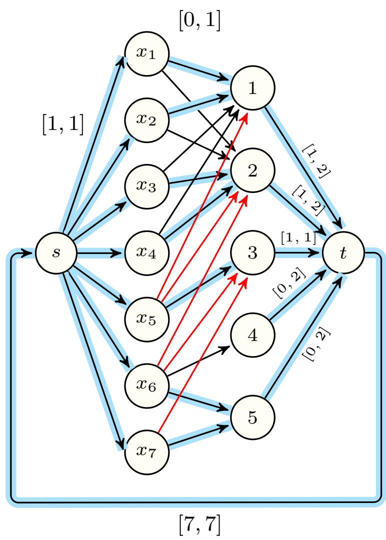
Figure 3.29: Flow models for the ALLDIFFERENT and the OPEN_ALLDIFFERENT constraints described in Tables 3.11 and 3.12: in both cases a first layer consists of the variables of the constraint and a second layer corresponds to the values that can be assigned to the variables; each arc has a lower and upper capacity regarding the flow it can carry; all variables x_i (with $i \in [1,5]$) such that the minimum capacity of the arc from s to x_i is equal to 1 must be assigned distinct values.

flows respectively corresponding to the solutions GLOBAL_CARDINALITY_LOW_UP $(\langle x_1 = 1, x_2 = 1, x_3 = 2, x_4 = 2, x_5 = 3, x_6 = 5, x_7 = 5 \rangle, \langle \texttt{val} - 1\ \texttt{omin} - 1\ \texttt{omax} - 2, \texttt{val} - 2\ \texttt{omin} - 1\ \texttt{omax} - 2, \texttt{val} - 3\ \texttt{omin} - 1\ \texttt{omax} - 1, \texttt{val} - 4\ \texttt{omin} - 0\ \texttt{omax} - 2, \texttt{val} - 5\ \texttt{omin} - 0\ \texttt{omax} - 2 \rangle)$ and GLOBAL_CARDINALITY_LOW_UP_NO_LOOP $(2, 2, \langle x_1 = 1, x_2 = 2, x_3 = 2, x_4 = 2, x_5 = 1, x_6 = 4, x_7 = 3 \rangle, \langle \texttt{val} - 1\ \texttt{omin} - 1\ \texttt{omax} - 2, \texttt{val} - 2\ \texttt{omin} - 2\ \texttt{omax} - 3, \texttt{val} - 3\ \texttt{omin} - 1\ \texttt{omax} - 1, \texttt{val} - 4\ \texttt{omin} - 1\ \texttt{omax} - 2, \texttt{val} - 5\ \texttt{omin} - 0\ \texttt{omax} - 2 \rangle)$, while red arcs correspond to arcs that cannot carry any flow if the constraint has a solution:

- Within the context of the GLOBAL_CARDINALITY_LOW_UP constraint variables x_1, x_2, x_3 and x_4 take their values within the set $\{1, 2\}$. Since each value in $\{1, 2\}$ can be used at most 2 times, variables different from x_1, x_2, x_3, x_4 cannot be assigned a value in $\{1, 2\}$. Consequently, $x_5 \neq 1$, $x_5 \neq 2$ and $x_6 \neq 2$. Since value 3 is the only remaining value for variable x_5, and since value 3 can be assigned to at most one variable, the assignments $x_6 \neq 3$ and $x_7 \neq 3$ are also forbidden.

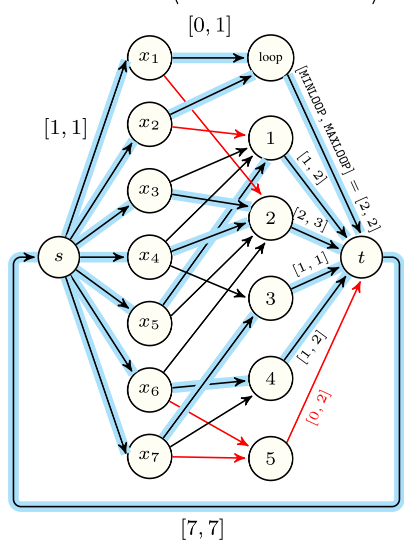
- – On the one hand we should have exactly two assignments of the form $x_i = i$ ($i \in [1,7]$), since the first and second arguments of the constraint

$$\left\| \begin{array}{l} x_1, x_2, x_3, x_4 \in [1,2], \\ x_5 \in [1,2,3], x_6 \in [2,3,4,5], \\ x_7 \in \{3,5\} \\ \text{GCC}\left(\begin{array}{c} \left\langle \begin{array}{c} x_1, x_2, x_3, \\ x_4, x_5, x_6, x_7 \end{array} \right\rangle, \\ \left\langle \begin{array}{c} 1\,1\,2,\ 2\,1\,2,\ 3\,1\,1, \\ 4\,0\,2,\ 5\,0\,2 \end{array} \right\rangle \end{array} \right). \end{array} \right.$$



Blue arcs correspond to the feasible flow associated with solution $x_1 = 1$, $x_2 = 1, x_3 = 2, x_4 = 2, x_5 = 3, x_6 = 5$, and $x_7 = 5$, while red arcs cannot carry any flow in a solution.

$$\left\| \begin{array}{l} x_1, x_2, x_3, x_5 \in [1,2], \\ x_4 \in [1,2,3], x_6 \in [2,4,5], x_7 \in \{3,4,5\} \\ \mathtt{MINLOOP} = 2, \mathtt{MAXLOOP} = 2, \\ \text{GCC\_NO\_LOOP}\left(\begin{array}{c} \mathtt{MINLOOP}, \mathtt{MAXLOOP}, \\ \left\langle \begin{array}{c} x_1, x_2, x_3, \\ x_4, x_5, x_6, x_7 \end{array} \right\rangle, \\ \left\langle \begin{array}{c} 1\,1\,2,\ 2\,2\,3,\ 3\,1\,1, \\ 4\,1\,2,\ 5\,0\,2 \end{array} \right\rangle \end{array} \right). \end{array} \right.$$



Blue arcs correspond to the feasible flow associated with solution $x_1 = 1$, $x_2 = 2, x_3 = 2, x_4 = 2, x_5 = 1, x_6 = 4$, and $x_7 = 3$, while red arcs cannot carry any flow in a solution (an arc from node x_i to node loop *means that $x_i = i$)*

Figure 3.30: Flow models for the GLOBAL_CARDINALITY_LOW_UP and the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraints described in Tables 3.13 and 3.14: in both cases a first layer consists of the variables of the constraint and a second layer corresponds to the values that can be assigned to the variables (in the second model, the loop node represents an *anonymous value* i corresponding to assignments of the form $x_i = i$); each arc has a lower and upper capacity regarding the flow it can carry; arcs entering a node associated with a variable must carry a flow of 1 since each variable must be assigned a single value, while arcs exiting a node associated with a value have a capacity set accordingly the last argument of the constraint (i.e., the collection of values VALUES).

Table 3.14: Domains of the variables and minimum and maximum number of occurrences of each value for the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraint of Figure 3.30; to the entry named loop corresponds the interval $[\texttt{MINLOOP}, \texttt{MAXLOOP}]$ where `MINLOOP` and `MAXLOOP` respectively correspond to the first and second arguments of the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraint.

i	$dom(x_i)$	i	$dom(x_i)$	i	$[omin_i, omax_i]$	i	$[omin_i, omax_i]$
1	$\{1,2\}$	5	$\{1,2\}$	loop	$[2,2]$	4	$[1,2]$
2	$\{1,2\}$	6	$\{2,4,5\}$	1	$[1,2]$	5	$[0,2]$
3	$\{1,2\}$	7	$\{3,4,5\}$	2	$[2,3]$		
4	$\{1,2,3\}$			3	$[1,1]$		

GLOBAL_CARDINALITY_LOW_UP_NO_LOOP are both set to two. Since the two variables x_1 and x_2 are the only variables such that $i \in dom(x_i)$ we must have $x_1 = 1$ and $x_2 = 2$, i.e. $x_1 \neq 2$ and $x_2 \neq 1$.

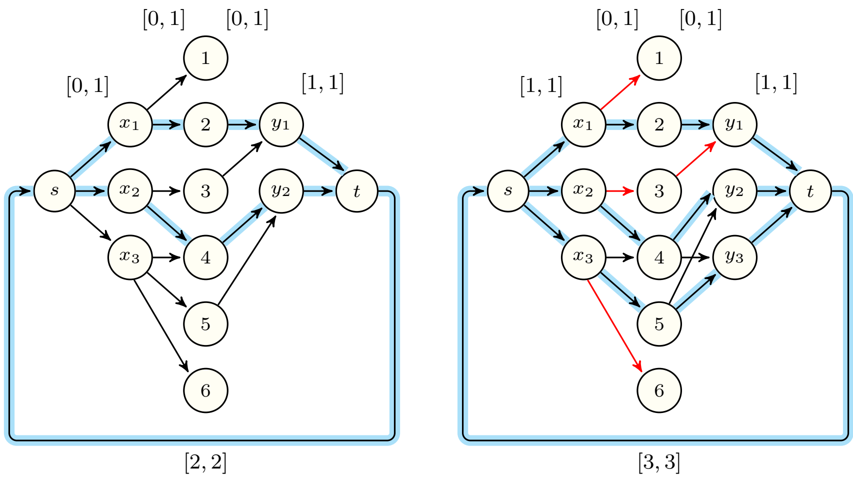
– On the other hand, since we should have at least $1 + 2 + 1 + 1 = 5$ assignments of the form $x_i = j$ ($i \neq j, j \in [1, 4]$) and since only 5 variables x_i (with $i \in [3, 7]$) can be assigned a value j in $[1, 4]$ with $i \neq j$, these variables should not be assigned a value outside interval $[1, 4]$, i.e. $x_6 \neq 5$ and $x_7 \neq 5$.

C. Flow models for the USED_BY and the SAME constraints

Figure 3.31 presents flow models for the USED_BY and the SAME constraints. Blue arcs represent feasible flows respectively corresponding to the solutions USED_BY($\langle x_1 = 2, x_2 = 4, x_3 = 6\rangle, \langle y_1 = 2, y_2 = 4\rangle$) and SAME($\langle x_1 = 2, x_2 = 4, x_3 = 5\rangle, \langle y_1 = 2, y_2 = 4, y_3 = 5\rangle$), while red arcs correspond to arcs that cannot carry any flow if the constraint has a solution. Within the context of the SAME constraint, the assignment $x_1 = 1$ is forbidden since $1 \notin dom(y_1) \cup dom(y_2) \cup dom(y_3)$. Consequently $x_1 = 2$ and, since y_1 is the only variable of $\{y_1, y_2, y_3\}$ that can be assigned value 2, the assignment $y_1 = 3$ is forbidden. Now since $3 \notin dom(y_1) \cup dom(y_2) \cup dom(y_3)$ the assignment $x_2 = 3$ is also forbidden. Finally $x_3 = 6$ is forbidden since $6 \notin dom(y_1) \cup dom(y_2) \cup dom(y_3)$.

$x_1 \in [1,2], x_2 \in [3,4], x_3 \in [4,6],$
$y_1 \in [2,3], y_2 \in [4,5],$
USED_BY$(\langle x_1, x_2, x_3\rangle, \langle y_1, y_2\rangle)$

$x_1 \in [1,2], x_2 \in [3,4], x_3 \in [4,6],$
$y_1 \in [2,3], y_2 \in [4,5], y_3 \in [4,5]$
SAME$(\langle x_1, x_2, x_3\rangle, \langle y_1, y_2, y_3\rangle)$



Blue arcs correspond to the feasible flow associated with the solution USED_BY$(\langle 2,4,6\rangle, \langle 2,4\rangle)$

Blue arcs correspond to the feasible flow associated with the solution SAME$(\langle 2,4,5\rangle, \langle 2,4,5\rangle)$, *while red arcs cannot carry any flow in a solution*

Figure 3.31: Flow models for the USED_BY and the SAME constraints described in Tables 3.15 and 3.16: in both cases a first layer consists of the variables x_i of the first argument of both constraints, a second layer corresponds to the values that can be assigned to the variables x_i and y_i (i.e., the first and second arguments), and a third layer consists of the variables y_i of the second argument of both constraints; there is an arc from a variable x_i (resp. y_i) to a value v if, and only if, value v can be assigned to variable x_i (resp. y_i); each arc has a lower and upper capacity regarding the flow it can carry; since for both constraints each value assigned to a variable of the second argument must also correspond to a value assigned to a variable of the first argument the arcs exiting the y_i must carry a flow of 1.

Table 3.15: Domains of the variables for the USED_BY constraint of Figure 3.31.

i	$dom(x_i)$	i	$dom(y_i)$
1	$\{1,2\}$	1	$\{2,3\}$
2	$\{3,4\}$	2	$\{4,5\}$
3	$\{4,5,6\}$		

Table 3.16: Domains of the variables for the SAME constraint of Figure 3.31.

i	$dom(x_i)$	i	$dom(y_i)$
1	$\{1, 2\}$	1	$\{2, 3\}$
2	$\{3, 4\}$	2	$\{4, 5\}$
3	$\{4, 5, 6\}$	3	$\{4, 5\}$

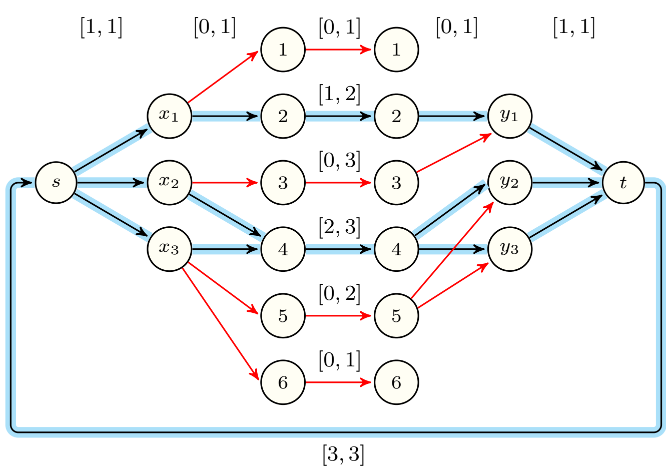
Table 3.17: Domains of the variables and minimum and maximum number of occurrences of each value for the SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraint of Figure 3.32.

i	$dom(x_i)$	i	$dom(y_i)$	i	$[omin_i, omax_i]$	i	$[omin_i, omax_i]$
1	$\{1,2\}$	1	$\{2,3\}$	1	$[0,1]$	4	$[2,3]$
2	$\{3,4\}$	2	$\{4,5\}$	2	$[1,2]$	5	$[0,2]$
3	$\{4,5,6\}$	3	$\{4,5\}$	3	$[0,3]$	6	$[0,1]$

D. Flow model for the SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraint

Figure 3.32 presents a flow model for the SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraint. Blue arcs represent the feasible flow corresponding to the solution SAME_AND_GLOBAL_CARDINALITY_LOW_UP $(\langle x_1 = 2, x_2 = 4, x_3 = 4\rangle, \langle y_1 = 2, y_2 = 4, y_3 = 4\rangle, \langle \texttt{val} - 1\ \texttt{omin} - 0\ \texttt{omax} - 1, \texttt{val} - 2\ \texttt{omin} - 1\ \texttt{omax} - 2, \texttt{val} - 3\ \texttt{omin} - 0\ \texttt{omax} - 3, \texttt{val} - 4\ \texttt{omin} - 2\ \texttt{omax} - 3, \texttt{val} - 5\ \texttt{omin} - 0\ \texttt{omax} - 2, \texttt{val} - 6\ \texttt{omin} - 0\ \texttt{omax} - 1\rangle)$, while red arcs correspond to arcs that cannot carry any flow if the constraint has a solution. The assignment $x_1 = 1$ is forbidden since $1 \notin dom(y_1) \cup dom(y_2) \cup dom(y_3)$. Consequently $x_1 = 2$ and, since y_1 is the only variable of $\{y_1, y_2, y_3\}$ that can be assigned value 2, the assignment $y_1 = 3$ is forbidden. Now since $3 \notin dom(y_1) \cup dom(y_2) \cup dom(y_3)$ the assignment $x_2 = 3$ is also forbidden. The assignment $x_3 = 6$ is forbidden since $6 \notin dom(y_1) \cup dom(y_2) \cup dom(y_3)$. Finally $x_3 = 5$, $y_2 = 5$ and $y_3 = 5$ are also forbidden since value 4 must be assigned to at least two variables.

$$\left\| \begin{array}{l} x_1 \in [1,2],\ x_2 \in [3,4],\ x_3 \in [4,6], \\ y_1 \in [2,3],\ y_2 \in [4,5],\ y_3 \in [4,5], \\ \text{SAME\_AND\_GCC} \left(\begin{array}{c} \langle x_1, x_2, x_3 \rangle, \\ \langle y_1, y_2, y_3 \rangle, \\ \langle 1\ 0\ 1,\ 2\ 1\ 2,\ 3\ 0\ 3,\ 4\ 2\ 3,\ 5\ 0\ 2,\ 6\ 0\ 1 \rangle \end{array} \right) \end{array} \right.$$



Blue arcs correspond to the feasible flow associated with the solution $x_1 = 2$, $x_2 = 4$, $x_3 = 4$, $y_1 = 2$, $y_2 = 4$, $y_3 = 4$, while red arcs cannot carry any flow in a solution.

Figure 3.32: Flow model for the SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraint described in Table 3.17: in both cases a first layer consists of the variables x_i of the first argument of the constraint, a second (resp. third) layer corresponds to the values that can be assigned to the variables x_i (resp. y_i), and a fourth layer consists of the variables y_i of the second argument of the constraint; each arc has a lower and upper capacity regarding the flow it can carry; values are duplicated in two layers in order to model the minimum and maximum number of occurrences of each value; there is an arc from a variable x_i (resp. y_i) to a value v if, and only if, value v can be assigned to variable x_i (resp. y_i); since each variable x_i (resp. y_i) must be assigned a value the arcs exiting s (resp. entering t) must carry a flow of 1.

3.7.106 Frequency allocation problem [1 CONS]

- ALL_MIN_DIST.

A constraint that was used for modelling frequency allocation problems.

3.7.107 Functional dependency [144 CONS]

- ABS_VALUE (intension, first argument),
- ALL_BALANCE (intension, first argument),
- ALLDIFFERENT_SAME_VALUE (intension, first argument),
- AMONG (intension, first argument),
- AMONG_DIFF_0 (intension, first argument),
- AMONG_INTERVAL (intension, first argument),
- AMONG_MODULO (intension, first argument),
- AMONG_VAR (intension, first argument),
- AND (intension, first argument),
- BALANCE (intension, first argument),
- BALANCE_CYCLE (intension, first argument),
- BALANCE_INTERVAL (intension, first argument),
- BALANCE_MODULO (intension, first argument),
- BALANCE_PARTITION (intension, first argument),
- BALANCE_PATH (intension, first argument),
- BALANCE_TREE (intension, first argument),
- BIG_PEAK (intension, first argument),
- BIG_VALLEY (intension, first argument),
- BINARY_TREE (intension, first argument),
- CARDINALITY_ATLEAST (intension, first argument),
- CARDINALITY_ATMOST (intension, first argument),
- CARDINALITY_ATMOST_PARTITION (intension, first argument),
- CASE (extension),
- CHANGE (intension, first argument),
- CHANGE_CONTINUITY (intension, first,second,...,eighth argument),
- CHANGE_PAIR (intension, first argument),
- CHANGE_PARTITION (intension, first argument),

- CHANGE_VECTORS (intension, first argument),
- CIRCULAR_CHANGE (intension, first argument),
- CLIQUE (intension, first argument),
- COLORED_MATRIX (intension, third attribute of fifth argument, third attribute of sixth argument),
- COMMON (intension, first, second argument),
- COMMON_INTERVAL (intension, first, second argument),
- COMMON_MODULO (intension, first, second argument),
- COMMON_PARTITION (intension, first, second argument),
- CONNECT_POINTS (intension, fourth argument),
- CROSSING (intension, first argument),
- CYCLE (intension, first argument),
- CYCLE_OR_ACCESSIBILITY (intension, second argument),
- CYCLIC_CHANGE (intension, first argument),
- CYCLIC_CHANGE_JOKER (intension, first argument),
- DEEPEST_VALLEY (intension, first argument),
- DIFFER_FROM_EXACTLY_K_POS (intension, first argument),
- DISCREPANCY (intension, second argument),
- DISTANCE (intension, third argument),
- DISTANCE_BETWEEN (intension, first argument),
- DISTANCE_CHANGE (intension, first argument),
- ELEM (extension, second attribute of first argument),
- ELEMENT (extension, third argument),
- ELEMENT_PRODUCT (extension, fourth argument),
- ELEMENTS (extension, second attribute of first argument),
- ELEMENTS_ALLDIFFERENT (extension, second attribute of first argument),
- EQ (intension, first, second argument),
- EQ_CST (intension, first, second, and third argument),
- EQUIVALENT (intension, first argument),
- EXACTLY (intension, first argument),
- FIRST_VALUE_DIFF_0 (intension, first argument),
- GCD (intension, third argument),
- GLOBAL_CARDINALITY (intension, second attribute of second argument),
- GLOBAL_CARDINALITY_NO_LOOP (intension, first argument as well as second attribute of third argument),
- GLOBAL_CARDINALITY_WITH_COSTS (intension, second attribute of second argument and fourth argument),
- GRAPH_CROSSING (intension, first argument),
- GROUP (intension, first, second,. . . ,sixth argument),
- GROUP_SKIP_ISOLATED_ITEM (intension, first, second,. . . ,fourth argument),
- HIGHEST_PEAK (intension, first argument),
- IMPLY (intension, first argument),
- INCREASING_NVALUE (intension, second argument),

- INCREASING_SUM (intension, first argument),
- INFLEXION (intension, first argument),
- INVERSE (intension, second and third attributes of first argument),
- INVERSE_EXCEPT_LOOP (intension, second and third attributes of first argument),
- INVERSE_OFFSET (intension, second and third attributes of third argument),
- LENGTH_FIRST_SEQUENCE (intension, first argument),
- LENGTH_LAST_SEQUENCE (intension, first argument),
- LONGEST_CHANGE (intension, first argument),
- LONGEST_DECREASING_SEQUENCE (intension, first argument),
- LONGEST_INCREASING_SEQUENCE (intension, first argument),
- MAP (intension, first, second argument),
- MAX_DECREASING_SLOPE (intension, first argument),
- MAX_INCREASING_SLOPE (intension, first argument),
- MAX_N (intension, first argument),
- MAX_NVALUE (intension, first argument),
- MAX_SIZE_SET_OF_CONSECUTIVE_VAR (intension, first argument),
- MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES (intension, first argument),
- MAX_OCC_OF_SORTED_TUPLES_OF_VALUES (intension, first argument),
- MAX_OCC_OF_TUPLES_OF_VALUES (intension, first argument),
- MAXIMUM (intension, first argument),
- MAXIMUM_MODULO (intension, first argument),
- MIN_DECREASING_SLOPE (intension, first argument),
- MIN_INCREASING_SLOPE (intension, first argument),
- MIN_N (intension, first argument),
- MIN_NVALUE (intension, first argument),
- MIN_SIZE_SET_OF_CONSECUTIVE_VAR (intension, first argument),
- MIN_SURF_PEAK, (intension, first argument),
- MIN_WIDTH_PEAK, (intension, first argument),
- MIN_WIDTH_PLATEAU, (intension, first argument),
- MIN_WIDTH_VALLEY, (intension, first argument),
- MINIMUM (intension, first argument),
- MINIMUM_EXCEPT_0 (intension, first argument),
- MINIMUM_MODULO (intension, first argument),
- MINIMUM_WEIGHT_ALLDIFFERENT (intension, third argument),
- MULTIPLE (intension, third argument),
- NAND (intension, first argument),
- NCLASS (intension, first argument),
- NEQUIVALENCE (intension, first argument),
- NINTERVAL (intension, first argument),
- NOR (intension, first argument),
- NPAIR (intension, first argument),
- NSET_OF_CONSECUTIVE_VALUES (intension, first argument),

- NUMBER_DIGIT (intension, first argument),
- NVALUE (intension, first argument),
- NVALUE_ON_INTERSECTION (intension, first argument),
- NVECTOR (intension, first argument),
- NVISIBLE_FROM_END (intension, first argument),
- NVISIBLE_FROM_START (intension, first argument),
- OPEN_AMONG (intension, second argument),
- OR (intension, first argument),
- ORCHARD (intension, first argument),
- ORDERED_NVECTOR (intension, first argument),
- ORTH_LINK_ORI_SIZ_END (intension, first, second and third attributes of first argument),
- PATH (intension, first argument),
- PEAK (intension, first argument),
- PERIOD (intension, first argument),
- PERIOD_EXCEPT_0 (intension, first argument),
- PERIOD_VECTORS (intension, first argument),
- POWER (intension, third argument),
- PROPER_FOREST (intension, first argument),
- REMAINDER (intension, third argument),
- SIGN_OF (intension, first argument),
- SIZE_MAX_SEQ_ALLDIFFERENT (intension, first argument),
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT (intension, first argument),
- SMOOTH (intension, first argument),
- STAGE_ELEMENT (extension, second attribute of first argument),
- SORT (intension, second argument),
- SORT_PERMUTATION (intension, second, third argument),
- SUM (intension, fourth argument),
- SUM_OF_WEIGHTS_OF_DISTINCT_VALUES (intension, third argument),
- TEMPORAL_PATH (intension, first argument),
- TREE (intension, first argument),
- TREE_RANGE (intension, first, second argument),
- TWO_LAYER_EDGE_CROSSING (intension, first argument),
- VALLEY (intension, first argument),
- WEIGHTED_PARTIAL_ALLDIFF (intension, fourth argument),
- XOR (intension, first argument).

A constraint that allows for representing a *functional dependency* between possibly several domain variables and a single domain variable. A sequence of variables $X_1, X_2, \ldots, X_n$ is said to *functionally determine* another variable Y if and only if each potential tuple of values of $X_1, X_2, \ldots, X_n$ is associated with exactly one potential value of Y (i.e., Y is a function of $X_1, X_2, \ldots, X_n$). For each constraint we indicate whether its functional dependency is defined in *intention* or in *extension*. We also indicate which variable var is determined by the functional dependency. Within the **Arg. properties** slot of a constraint that mentions the functional dependency keyword, we also mention which variables determine var.

Finally, the keyword Pure functional dependency provides the list of constraints that are only defined by one or several functional dependencies. For example, the NVALUE$(n, \langle v_1, v_2, \ldots, v_m)$ constraint is only defined in term of a functional dependency (i.e., n is equal to the number of distinct values in $v_1, v_2, \ldots, v_m$), while the TREE$(n, \langle node_1, node_2, \ldots, node_m)$ constraint is not only defined in term of a functional dependency since, in addition of counting trees, it also enforces no cycle in the corresponding graph.

3.7.108 Geometrical constraint [32 CONS]

- CONNECT_POINTS,
- CONTAINS_SBOXES,
- COVEREDBY_SBOXES,
- COVERS_SBOXES,
- CROSSING,
- CUMULATIVE_TWO_D,
- CYCLE_OR_ACCESSIBILITY,
- DIFFN,
- DIFFN_COLUMN,
- DIFFN_INCLUDE,
- DISJOINT_SBOXES,
- EQUAL_SBOXES,
- GEOST,
- GEOST_TIME,
- GRAPH_CROSSING,
- INSIDE_SBOXES,
- MEET_SBOXES,
- NON_OVERLAP_SBOXES.
- ORCHARD,
- ORTH_ON_THE_GROUND,
- ORTH_ON_TOP_OF_ORTH,
- ORTHS_ARE_CONNECTED,
- OVERLAP_SBOXES,
- PLACE_IN_PYRAMID,
- POLYOMINO,
- SEQUENCE_FOLDING,
- TWO_LAYER_EDGE_CROSSING,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_COLUMN,
- TWO_ORTH_DO_NOT_OVERLAP,
- TWO_ORTH_INCLUDE,
- VISIBLE.

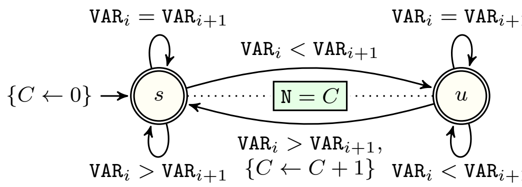
A constraint between geometrical objects (e.g., points, line segments, rectangles, orthotopes) or a constraint selecting a subset of points so that a given geometrical property holds (e.g., distance).

3.7.109 Glue matrix [41 CONS]

- AMONG,
- CHANGE_CONTINUITY (NB_PERIOD_CHANGE),
- CHANGE_CONTINUITY (NB_PERIOD_CONTINUITY),
- CHANGE_CONTINUITY (MIN_SIZE_CHANGE),
- CHANGE_CONTINUITY (MIN_SIZE_CONTINUITY),
- CHANGE_CONTINUITY (MAX_SIZE_CHANGE),
- CHANGE_CONTINUITY (MAX_SIZE_CONTINUITY),
- CHANGE_CONTINUITY (NB_CHANGE),
- CHANGE_CONTINUITY (NB_CONTINUITY),
- DEEPEST_VALLEY,
- EXACTLY,
- FULL_GROUP (NGROUP),
- FULL_GROUP (NVAL),
- GROUP (NGROUP),
- GROUP (MIN_SIZE),
- GROUP (MAX_SIZE),
- GROUP (MIN_DIST),
- GROUP (MAX_DIST),
- GROUP (NVAL),
- GROUP_SKIP_ISOLATED_ITEM (NGROUP),
- GROUP_SKIP_ISOLATED_ITEM (MIN_SIZE),
- GROUP_SKIP_ISOLATED_ITEM (MAX_SIZE),
- GROUP_SKIP_ISOLATED_ITEM (NVAL),
- HIGHEST_PEAK,
- INFLEXION,
- LENGTH_FIRST_SEQUENCE,
- LENGTH_LAST_SEQUENCE,
- LONGEST_CHANGE,
- LONGEST_DECREASING_SEQUENCE,
- LONGEST_INCREASING_SEQUENCE,
- MAXIMUM,
- MAX_DECREASING_SLOPE,
- MAX_INCREASING_SLOPE,
- MIN_DECREASING_SLOPE,
- MIN_INCREASING_SLOPE,
- MIN_WIDTH_PEAK,
- MIN_WIDTH_VALLEY,
- MINIMUM,
- PEAK,
- SMOOTH,
- VALLEY.

A reversible constraint for which the catalogue provides an automaton with counters and a glue matrix [38]. A *glue matrix* is indexed by the states of the automaton associated with the considered constraint as well as by the states of the automaton associated with the reverse of the considered constraint. In the following we assume that the signature constraint involves a consecutive variables of the sequence of variables of the reversible constraint (the signature constraint encodes the mapping of the sequence of variables of the constraint to symbols of the alphabet of the automaton). We consider a sequence of variables and a prefix and suffix of this sequence such that the prefix and suffix have $a-1$ variables in common. Let $\overrightarrow{q}$ (resp. $\overleftarrow{q}$) be the state of the automaton associated with the constraint (resp. the reverse constraint) upon reading the prefix (resp. the reverse of the suffix). The entry of the glue matrix corresponding to the state pair $(\overrightarrow{q}, \overleftarrow{q})$ provides a function for computing the result associated with the sequence from the counters values associated with the prefix and the reverse suffix.

As an example consider the PEAK$(\mathtt{N}, \mathtt{VARIABLES})$, which holds if N is equal to the number of peaks of the sequence of variables VARIABLES. A *peak* corresponds to an increase between consecutive variables followed by a decrease between consecutive variables. Figure 3.33 gives the corresponding automaton that returns the number of peak of a sequence where, to each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ corresponds a signature variable S_i passed to the automaton. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and S_i: $(\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.



	s	u
s	$\overrightarrow{C} + \overleftarrow{C}$	$\overrightarrow{C} + \overleftarrow{C}$
u	$\overrightarrow{C} + \overleftarrow{C}$	$\overrightarrow{C} + 1 + \overleftarrow{C}$

$\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence $\texttt{VARIABLES}$.

Figure 3.33: Automaton of the PEAK constraint and its glue matrix (an accepting state is denoted graphically by a double circle)

Figure 3.34 illustrates the use of the glue matrix of the PEAK constraint on the sequence $1, 1, 4, 8, 6, 2, 7, 1$ decomposed in a prefix $1, 1, 4, 8$ and a suffix $8, 6, 2, 7, 1$ that overlap by one position, one position since the arity of the signature constraint is equal to two. Since the automaton of the PEAK constraint ends up in state u when applied to the prefix and to the reverse suffix we use the lower rightmost entry of the glue matrix to link the total number of peaks of the sequence $1, 1, 4, 8, 6, 2, 7, 1$ with the number of peaks of the prefix $1, 1, 4, 8$ and the suffix $8, 6, 2, 7, 1$.

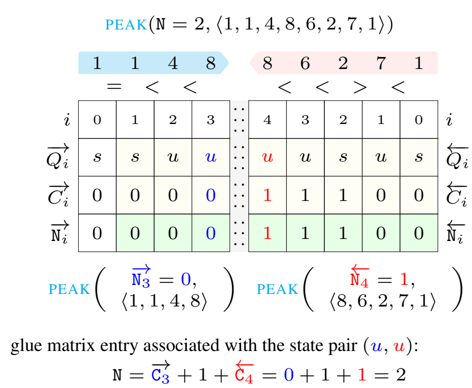


Figure 3.34: Illustrating the use of the state pair (u, u) of the glue matrix for linking $\texttt{N}$ with the counters variables obtained after reading the prefix $1, 1, 4, 8$ and corresponding suffix $8, 6, 2, 7, 1$ of the sequence $1, 1, 4, 8, 6, 2, 7, 1$; note that the suffix $8, 6, 2, 7, 1$ (in pink) is proceed in reverse order; the left (resp. right) table shows the initialisation (for $i = 0$) and the evolution (for $i > 0$) of the state of the automaton and of its counter C upon reading the prefix $1, 1, 4, 8$ (resp. the suffix $1, 7, 2, 6, 8$).

3.7.110 Golomb ruler [2 CONS]

- ALLDIFFERENT,
- GOLOMB.

A constraint that allows for expressing the *Golomb ruler* problem. A *Golomb ruler* is a set of integers (marks) $a_1 < \cdots < a_k$ such that all the differences $a_i - a_j$ $(i > j)$ are distinct.

3.7.111 Graph colouring [3 CONS]

- ALLDIFFERENT,
- INT_VALUE_PRECEDE_CHAIN,
- K_ALLDIFFERENT.

A constraint that can be used for the *graph colouring problem*. The *graph colouring problem* is to colour with a restricted number of colours the vertices of a given undirected graph in such a way that adjacent vertices are coloured with distinct colours.

3.7.112 Graph constraint [40 CONS]

- BALANCE_CYCLE,
- BALANCE_PATH,
- BALANCE_TREE,
- BINARY_TREE,
- BIPARTITE,
- CIRCUIT,
- CIRCUIT_CLUSTER,
- CLIQUE,
- CONNECTED,
- CUTSET,
- CYCLE,
- CYCLE_CARD_ON_PATH,
- CYCLE_OR_ACCESSIBILITY,
- CYCLE_RESOURCE,
- DAG,
- DERANGEMENT,

- DOM_REACHABILITY,
- GRAPH_CROSSING,
- GRAPH_ISOMORPHISM,
- INVERSE,
- INVERSE_EXCEPT_LOOP,
- INVERSE_OFFSET,
- INVERSE_WITHIN_RANGE,
- K_CUT,
- MAP,
- PATH,
- PATH_FROM_TO,
- PROPER_CIRCUIT,
- PROPER_FOREST,
- STABLE_COMPATIBILITY,
- STRONGLY_CONNECTED,
- SUBGRAPH_ISOMORPHISM,
- SYMMETRIC,
- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_LOOP,
- TEMPORAL_PATH,
- TOUR,
- TREE,
- TREE_RANGE,
- TREE_RESOURCE.

A constraint that selects a subgraph from a given initial graph so that this subgraph satisfies a given property and/or belong to a specific graph class.

3.7.113 Graph partitioning constraint [18 CONS]

- BALANCE_CYCLE,
- BALANCE_PATH,
- BALANCE_TREE,
- BINARY_TREE,
- CIRCUIT,
- CYCLE,
- CYCLE_CARD_ON_PATH,
- CYCLE_RESOURCE,
- GRAPH_CROSSING,
- MAP,
- PATH,
- PROPER_CIRCUIT,
- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_LOOP,
- TEMPORAL_PATH,
- TREE,
- TREE_RANGE,
- TREE_RESOURCE.

A constraint that partitions the vertices of a given initial graph and that keeps a single successor for each vertex so that each partition corresponds to a specific pattern.

3.7.114 Guillotine cut [2 CONS]

- DIFFN_COLUMN,
- TWO_ORTH_COLUMN.

A constraint that can enforce some kind of *guillotine cut*. In a lot of cutting problems the stock sheet as well as the pieces to be cut are all shaped as rectangles. In a *guillotine cutting pattern* all cuts must go from one edge of the rectangle corresponding to the stock sheet to the opposite edge.

3.7.115 Hall interval [2 CONS]

- ALLDIFFERENT,
- GLOBAL_CARDINALITY.

A constraint for which some filtering algorithms take advantage of *Hall intervals*. Given a set of domain variables, a *Hall set* is a set of values $H = \{v_1, v_2, \dots, v_h\}$ such that there are h variables whose domains are contained in H. A *Hall interval* is a Hall set that consists of an interval of values (and can therefore be specified by its endpoints).

3.7.116 Hamiltonian [2 CONS]

- CIRCUIT,
- TOUR.

A constraint enforcing to cover a graph with one Hamiltonian circuit or cycle. This corresponds to finding a circuit (respectively a cycle) passing all the vertices exactly once of a given digraph (respectively undirected graph).

3.7.117 Heuristics

[5 CONS]

- ALLDIFFERENT,
- DISCREPANCY,
- INVERSE,
- INVERSE_OFFSET,
- INVERSE_WITHIN_RANGE.

A constraint that was introduced for expressing a heuristic or a constraint (ALLDIFFERENT) for which an algorithm that evaluate the number of solutions was proposed.

Remark: when we do not have good bounds on the cost variable of a constrained optimisation problem, *skewed binary search* was introduced in [389] in order to take advantage of the fact that it is usually easier to improve the current solution cost's than to prove that a problem is not feasible.

3.7.118 Heuristics and Berge-acyclic constraint network

Consider a conjunction $\mathcal{C}$ of constraints such that:

1. The constraint network $\mathcal{N}$ corresponding to the conjunction $\mathcal{C}$ is not Berge-acyclic.
2. The filtering algorithms associated with the different constraints of the conjunction $\mathcal{C}$ all achieve arc-consistency.

In this context, one can design a heuristic that fix enough variables, but not all, so that the remaining constraint network $\mathcal{N}$ becomes Berge-acyclic.[13] This can be achieved by fixing the variables in such a way that some constraints get entailed even though they still mention some variables that are not yet fixed. Let us illustrate that idea on a matrix model where we have a $R \times K$ matrix $\mathcal{M}$ of domain variables taking a value in interval $[1, V]$. Assume that:

- On each row of $\mathcal{M}$ we have a constraint that can be described in term of a counter-free automaton.
- On each column of $\mathcal{M}$ we have a GLOBAL_CARDINALITY_LOW_UP constraint that only imposes a minimum number of occurrences for each value in $[1, V]$ (i.e., the maximum number of occurrences is not constrained at all).

[13] The point is that, as soon as the constraint network becomes Berge-acyclic no search is needed any more to check that there is a solution, provided we achieve arc-consistency on the remaining constraints. This stems from [143], which itself is a consequence of [183].

Note that arc-consistency can be achieved for such constraints. For this constraint pattern, an assignment strategy that systematically tries creating a Berge-acyclic constraint network can be achieved as follows. Fix some variables so that $K - 1$ column constraints (i.e., GLOBAL_CARDINALITY_LOW_UP constraints) get entailed. If this is the case the remaining constraint network consists of R rows constraints and of a single column constraint.

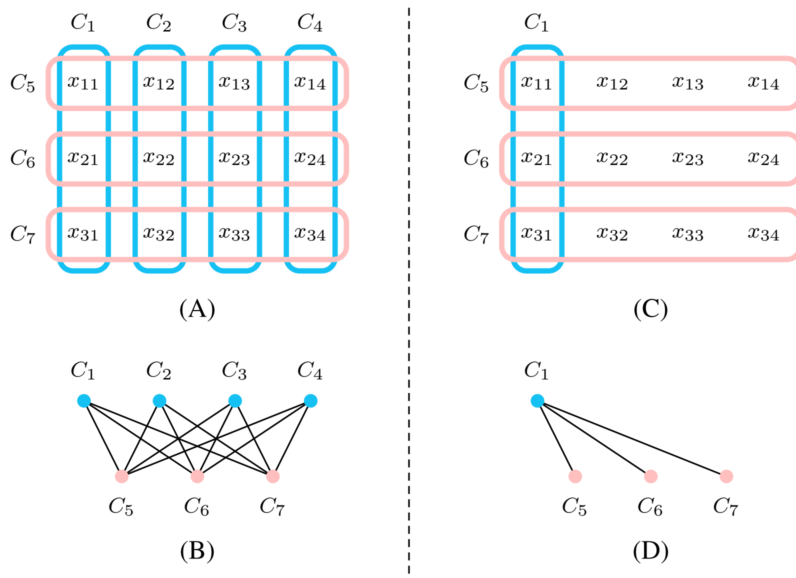


Figure 3.35: (A) Initial constraint network for a $R = 3$ by $K = 4$ matrix (with column constraints C_1, C_2, C_3, C_4 and row constraints C_5, C_6, C_7) and (B) corresponding intersection graph; (C) Berge-acyclic constraint network after the entailment of the column constraints C_2, C_3 and C_4 and (D) corresponding cycle-free intersection graph.

As illustrated by Figure 3.35, this typically corresponds to a Berge-acyclic constraint network. Let us now finally explain how to assign values to a subset of variables of a GLOBAL_CARDINALITY_LOW_UP constraint that only restricts the minimum number of occurrences of certain values so that it becomes entailed. As an example, let us consider a GLOBAL_CARDINALITY_LOW_UP constraint involving 10 variables which forces at least three occurrences of value 1 and one occurrence of value 2. A heuristic needs only fixing 4 variables out of the 10 variables to values 1, 1, 1 and 2 so that the corresponding GLOBAL_CARDINALITY_LOW_UP gets entailed. A typical instance of this pattern corresponds to *nurse scheduling problems* where:

- Each row of $\mathcal{M}$ corresponds to the timetable of a person over K consecutive days. Using a counter free automaton the corresponding row constraint encodes all legal rules of a valid schedule.
- Each column of $\mathcal{M}$ describes the request for a minimum number of services on a given day. Types of work (i.e., values in $[1, V]$) can be, for example, interpreted

as a *morning shift*, an *afternoon shift*, a *night shift* or a *day off*.

The heuristic first addresses the coverage constraints only (i.e., the GLOBAL_CARDINALITY_LOW_UP constraints). It seeks to assign enough nurses to given shifts on given day to satisfy all but one coverage constraints. Once this is done, the remaining variables can be labelled without search.

3.7.119 Heuristics and lexicographical ordering [8 CONS]

- LEX_CHAIN_GREATER,
- LEX_CHAIN_GREATEREQ,
- LEX_CHAIN_LESS,
- LEX_CHAIN_LESSEQ,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ.

Using a constraint that imposes a lexicographical ordering between vectors of variables may influence the heuristic used for fixing the variables. In particular it may be a very bad idea to systematically fix the *less significant components before the most significant components*.

3.7.120 Heuristics for two-dimensional rectangle placement problems [2 CONS]

- DIFFN,
- GEOST.

A constraint for which one of the following heuristics was used in the context of two-dimensional rectangles placement problems where rectangles should not overlap. For easy instances involving non-overlapping constraints where there is enough room, a standard heuristic where one fixes each rectangle successively by trying out its possible values for its x-coordinate and its y-coordinate will do the job. However, for more difficult problems a less aggressive heuristic is usually required, especially when the filtering algorithms attached to the constraints are weak. The paradox is that less aggressive heuristics sometimes do not find rapidly a first solution to easy instances since they may potentially artificially create infeasible subproblems.

Dual strategy for rectangle placement problems with no slack

When the available space is equal to the total area of the rectangles to place (i.e., we have no slack) this is a two-phase search procedure originally introduced in [1] where we first fix all the x-coordinates and then, in the second phase, all the y-coordinates. The intuitions behind this heuristic are:

- To *systematically* fill the placement space from right to left in order to avoid creating small holes that cannot be filled.

- To decrease the combinatorial aspect of the problem by *focussing first on all x-coordinates*. This stems from the fact that it is usually easy to extend a partial solution, where all x-coordinates are fixed, to a full solution.

Fixing the x-coordinates is done by:

- First, compute the minimum min_x over the minimum values of the x-coordinates of the rectangles for which the x-coordinate is not already fixed.

- Second, create a choice point and, in each branch:

  - Fix the x-coordinate of a rectangle R for which the x-coordinates is not already fixed to value min_x. Usually rectangles are considered by decreasing height (and decreasing width in case of tie).

  - On backtracking, enforce that the x-coordinate of rectangle R is strictly greater than min_x.

- Third, fail when all branches issued from a choice point have been tried (since otherwise we would create a hole at position min_x because, on the x axis all rectangles that could start at position min_x were delayed after min_x; in order to not cut valid choices, this third part assumes that the minimum value of the x-coordinate of each rectangle is pruned with respect to the compulsory part profile of the corresponding CUMULATIVE constraint.).

Since, as we said early on, it is usually easy to extend a partial solution, where all x-coordinates are fixed, to a full solution where all y-coordinates are also fixed, the search strategy used for fixing the y-coordinates is usually not so important, at least when strong filtering algorithms are used [41].

Strategy that gradually creates a compulsory part

This is a four-phase search procedure that can be used even when the slack is not equal to zero. We first gradually restrict all the x-coordinates and then, in the second phase, all y-coordinates without fixing them immediately. Then in the third phase we fix all the x-coordinates by trying each value (or by making a binary search). Finally in the last phase we fix all the y-coordinates as in the third phase. The intuitions behind this heuristic are:

- To restrict the x-coordinate of each rectangle R in order to just create some compulsory part for R on the x axis. The hope is that it will trigger the filtering algorithm associated with the CUMULATIVE constraint involved by the non-overlapping constraint, even though the starts of the rectangles on the x axis are not yet completely fixed.
- Again, as in the previous heuristic, to decrease the combinatorial aspect of the problem by first focussing on all x-coordinates.

Restricting gradually the x-coordinates in phase one is done by partitioning the domain of the x-coordinate of each rectangle R into intervals whose sizes induce a compulsory part on the x axis for rectangle R. To achieve this, the size of an interval has to be less than or equal to the size of rectangle R on the x axis. Picking the best fraction of the size of a rectangle on the x axis depends on the problem as well as on the filtering algorithms behind the scene. Within the context of the smallest rectangle area problem [401] and of the **SICStus** implementation of DISJOINT2 and CUMULATIVE H. Simonis and B. O'Sullivan have shown empirically that the best fraction was located within interval $[0.2, 0.3]$. Restricting the y-coordinates in phase two can be done in a way similar to restricting the x-coordinates in phase one.

3.7.121 Hungarian method for the assignment problem [1 CONS]

- MINIMUM_WEIGHT_ALLDIFFERENT.

A constraint that can use the Hungarian method for the assignment problem [254] in order to evaluate the minimum or maximum value of one of its argument. Given n persons, n tasks and a corresponding n by n cost matrix, the assignment problem is the search for an assignment of persons to tasks so that the sum of the costs is maximised.

3.7.122 Hybrid-consistency [2 CONS]

- PROPER_FOREST,
- ROOTS.

Denotes that, for a given constraint involving both domain and set variables, there is a filtering algorithm that ensures *hybrid-consistency*. A constraint *ctr* defined on the distinct domain variables $V_1^d, \ldots, V_n^d$ and the distinct set variables $V_{n+1}^s, \ldots, V_m^s$ is *hybrid-consistent* if and only if:

- For every pair (V^d, v) such that V^d is a domain variable of *ctr* and $v \in dom(V^d)$, there exists at least one solution to *ctr* in which V^d is assigned the value v.
- For every pair (V^s, v) such that V^s is a set variable of *ctr*, if $v \in \underline{V^s}$ then v belongs to the set assigned to V^s in all solutions to *ctr* and if $v \in \overline{V^s} \setminus \underline{V^s}$ then v belongs to the set assigned to V^s in at least one solution and is excluded from this set in at least one solution.

3.7.123 Hypergraph [8 CONS]

- AMONG_SEQ,
- ARITH_SLIDING,
- ORCHARD,
- RELAXED_SLIDING_SUM,
- SIZE_MAX_SEQ_ALLDIFFERENT,
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT,
- SLIDING_DISTRIBUTION,
- SLIDING_SUM.

Denotes that a constraint uses in its definition at least one arc constraint involving more than two vertices.

3.7.124 Included [2 CONS]

- IN,
- IN_SET.

Enforces that a domain or a set variable take a value within a list of values (possibly a single value).

3.7.125 Inclusion [8 CONS]

- K_USED_BY,
- K_USED_BY_INTERVAL,
- K_USED_BY_MODULO,
- USED_BY,
- USED_BY_INTERVAL,
- USED_BY_MODULO,
- USED_BY_PARTITION,
- USES.

Denotes that a constraint can model the inclusion of one multiset within another multiset. Usually we consider multiset of values (e.g., USED_BY) but this can also be multisets of equivalence classes (see, e.g., the USED_BY_INTERVAL, USED_BY_MODULO, and USED_BY_PARTITION constraints).

3.7.126 Incompatible pairs of values [1 CONS]

- ALLDIFFERENT_PARTITION.

A constraint that is related to the fact that some pairs of values are incompatible (i.e., the two values of each pair of values cannot simultaneously be part of a solution).

3.7.127 Indistinguishable values [3 CONS]

- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- SET_VALUE_PRECEDE.

A constraint that can be used for breaking symmetries of *indistinguishable values* [268]. *Indistinguishable values* in a solution to a problem can be swapped to construct another solution to the same problem.

3.7.128 Interval [16 CONS]

- ALLDIFFERENT_INTERVAL,
- AMONG_INTERVAL,
- BALANCE_INTERVAL,
- COMMON_INTERVAL,
- DOMAIN,
- IN_INTERVAL,
- IN_INTERVALS,
- INTERVAL_AND_COUNT,
- INTERVAL_AND_SUM,
- K_SAME_INTERVAL,
- K_USED_BY_INTERVAL,
- NINTERVAL,
- SAME_INTERVAL,
- SOFT_SAME_INTERVAL_VAR,
- SOFT_USED_BY_INTERVAL_VAR,
- USED_BY_INTERVAL.

Denotes that a constraint puts a restriction related to a set of fixed intervals (or on one fixed interval).

3.7.129 Involution [2 CONS]

- SYMMETRIC_ALLDIFFERENT_LOOP,
- SYMMETRIC_ALLDIFFERENT.

Denotes that a constraint can directly model permutations of order 2.

3.7.130 Joker value [14 CONS]

- ALL_EQUAL_EXCEPT_0,
- ALLDIFFERENT_EXCEPT_0,
- AMONG_DIFF_0,
- CONNECT_POINTS,
- CYCLIC_CHANGE_JOKER,
- FIRST_VALUE_DIFF_0,
- ITH_POS_DIFFERENT_FROM_0,
- LEX_ALLDIFFERENT_EXCEPT_0,
- MINIMUM_EXCEPT_0,
- MIN_SIZE_FULL_ZERO_STRETCH,
- NVALUES_EXCEPT_0,
- PERIOD_EXCEPT_0,
- SYMMETRIC_ALLDIFFERENT_EXCEPT_0,
- WEIGHTED_PARTIAL_ALLDIFF.

Denotes that, for some variables of a given constraint, there exists specific values that have a special meaning: for instance they can be assigned without breaking the constraint. As an example consider the ALLDIFFERENT_EXCEPT_0 constraint, which forces a set of variables to take distinct values, except those variables that are assigned to 0.

3.7.131 Klee's measure problem [1 CONS]

- DIFFN.

Denotes that, checking the feasibility of a ground instance of a constraint, is related to the *Klee's measure problem*: given a collection of axis-aligned multi-dimensional boxes, how quickly can one compute the volume of their unions.

3.7.132 Labelling by increasing cost [2 CONS]

- ELEM,
- ELEMENT.

Some optimisation problems involve minimising a cost c consisting of a sum of elementary costs $c_1, c_2, \ldots, c_n$, where each elementary cost c_i $(1 \leq i \leq n)$ is directly linked to the value assigned to a decision variable v_i. Without loss of generality we assume that each decision variable will be assigned a value between 1 and m. The link between a decision variable v_i and its corresponding cost c_i is usually expressed by a constraint of the form ELEMENT$(v_i, \langle c_{i,1}, c_{i,2}, \ldots, c_{i,m}\rangle, c_i)$ stating that $c_i = j \Rightarrow c_i = c_{i,j}$. During search, while enumerating on the different values of a decision variable v_i, we would like to try out values of v_i so that the corresponding cost c_i increases. This means we want to use a permutation $\sigma_1, \sigma_2, \ldots, \sigma_m$ of $1, 2, \ldots, m$ such that $c_{i,\sigma_1} \leq c_{i,\sigma_2} \leq \cdots \leq c_{i,\sigma_m}$. Note that such permutation can be obtained by sorting the costs $c_{i,1}, c_{i,2}, \ldots, c_{i,m}$ by increasing order and by collecting the position σ_j where item $c_{i,j}$ is located in the sorted list. Assuming that we perform arc-consistency on the ELEMENT, we now describe three different ways to obtain the effect we want to achieve:

- A first direct way is to use a built in facility that, given variable v_i and the corresponding list of values $\sigma_1, \sigma_2, \ldots, \sigma_m$ introduced before, creates a choice point and tries to successively assign values $\sigma_1, \sigma_2, \ldots, \sigma_m$ to v_i. Note that, once v_i is fixed there is no need to enumerate on the corresponding elementary cost variable c_i since, by propagation, ELEMENT$(v_i, \langle c_{i,1}, c_{i,2}, \ldots, c_{i,m}\rangle, c_i)$ will fix c_i. Consequently the cost variables do not need to be passed to the search procedure.
- A second indirect way, used when we want to only rely on a standard built in that creates a choice point and tries to assign values to a variable in increasing value order, is to introduce an extra variable u_i. The idea is to link variable u_i to variable v_i in such a way that, when we try to assign values in increasing value

order to variable u_i, both variables v_i and c_i get fixed and, in addition, values of c_i are increasing. This can be modelled by introducing the following two ELEMENT constraints:

1. ELEMENT$(u_i, \langle \sigma_1, \sigma_2, \ldots, \sigma_m \rangle, v_i)$
2. ELEMENT$(v_i, \langle c_{i,1}, c_{i,2}, \ldots, c_{i,m} \rangle, c_i)$

The effect of a dedicated built in that tries to assign values to a variable according to an explicit list of values is achieved by introducing the first ELEMENT constraint. Again, once u_i is fixed the first ELEMENT constraint will fix variable v_i. Then the second ELEMENT constraint will also fix variable c_i. Consequently, both the cost and the decision variables do not need to be passed to the search procedure, i.e., we just need to pass the newly introduced variables u_i.

- Finally, we can first label on the cost variable c_i in increasing value order. If the costs $c_{i,1}, c_{i,2}, \ldots, c_{i,m}$ are all distinct then the ELEMENT$(v_i, \langle c_{i,1}, c_{i,2}, \ldots, c_{i,m} \rangle, c_i)$ constraint will fix v_i by propagation since we assume ELEMENT to perform arc-consistency. Otherwise, when the costs $c_{i,1}, c_{i,2}, \ldots, c_{i,m}$ are not all distinct, we also need to label the decision variable v_i.

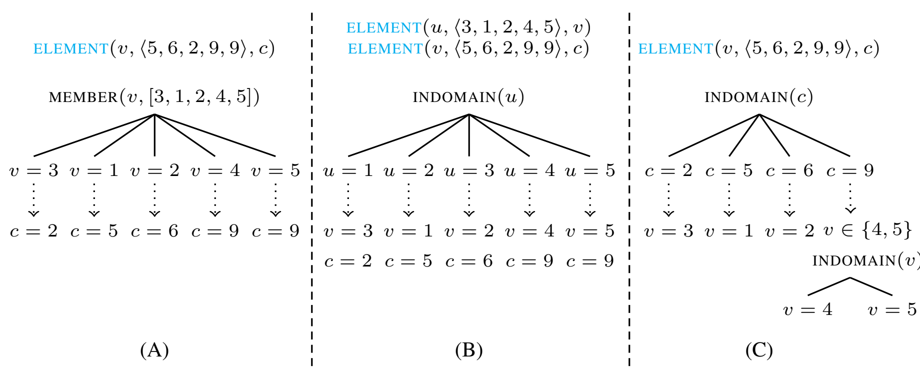


Figure 3.36: Given a decision variable v and a corresponding cost variable c linked by the ELEMENT$(v, \langle 5, 6, 2, 9, 9 \rangle, c)$ constraint, illustration of three ways for labelling by increasing cost: part (A) labels directly on the decision variable v using an appropriate order so that successive values of c are increasing; part (B) introduces a variable u linked to v by the ELEMENT$(u, \langle 3, 1, 2, 4, 5 \rangle, v)$ constraint and labels on u by increasing value order; part (C) labels first on the cost variable c by increasing value order, and then on variable v.

Figure 3.36 illustrates the three ways of labelling previously introduced. The primitive member$(var, list_values)$ creates a choice point and tries to successively assign variable var an integer value from the list $list\_values$ with respect to their ordering. The primitive indomain(var) also creates a choice point and tries to successively assign variable var an integer value of its domain, by increasing value order.

3.7.133 Latin square [1 CONS]

- K_ALLDIFFERENT.

1			
			3
3			
			1

(A)

1	2	3	4
2	1	4	**3**
3	4	1	2
4	3	2	**1**

(B)

Figure 3.37: A partially filled Latin square and a possible completion

A constraint that can be used for modelling the Latin square completion problem. A *Latin square of order* n is an $n \times n$ array in which n distinct numbers in $[1, n]$ are arranged so that each number occurs once in each row and column. The *Latin square completion problem* is to complete a partially filled Latin square. Part (A) of Figure 3.37 gives a partially filled Latin square, while part (B) provides a possible completion. The Latin square completion problem is a pattern that occurs in some applications such that *dynamic wavelength routing* or *sport timetabling*.

3.7.134 Lexicographic order [18 CONS]

- ALLPERM,
- COND_LEX_COST,
- COND_LEX_GREATER,
- COND_LEX_GREATEREQ,
- COND_LEX_LESS,
- COND_LEX_LESSEQ,
- LEX2,
- LEX_BETWEEN,
- LEX_CHAIN_GREATER,
- LEX_CHAIN_GREATEREQ,
- LEX_CHAIN_LESS,
- LEX_CHAIN_LESSEQ,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LEX_LESSEQ_ALLPERM,
- STRICT_LEX2.

A constraint involving a lexicographic ordering relation in its definition.

3.7.135 Limited discrepancy search [1 CONS]

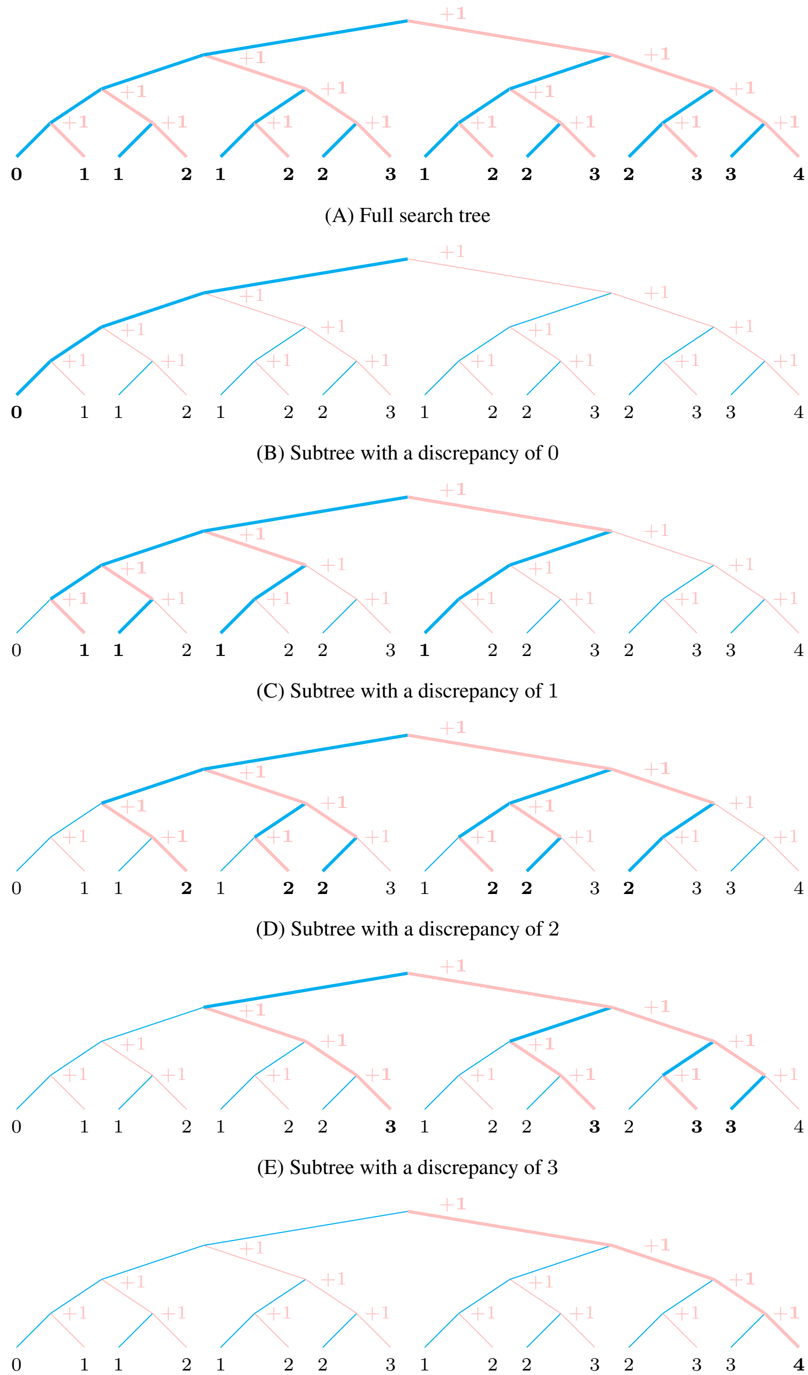
- DISCREPANCY.

A constraint for simulating limited discrepancy search [204]. *Limited discrepancy search* is useful for problems for which there is a successor ordering heuristic that usually leads directly to a solution. It consists of systematically searching all paths that differ from the heuristic path in at most a very small number of discrepancies. Figure 3.38 illustrates the successive search steps (B), (C), (D), (E) and (F) on the search tree depicted by part (A). We successively explore the subtree of (A) corresponding to a discrepancy of 0, 1, 2, 3 and 4. The number on each leave indicates the total number of discrepancies to reach a leave.

3.7.136 Linear programming [18 CONS]

- ALLDIFFERENT,
- AMONG_SEQ,
- CIRCUIT,
- CUMULATIVE,
- DISJUNCTIVE,
- DOMAIN_CONSTRAINT,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- GLOBAL_CARDINALITY_LOW_UP,
- K_ALLDIFFERENT,
- K_CUT,
- LINK_SET_TO_BOOLEANS,
- PATH_FROM_TO,
- REGULAR,
- SLIDING_SUM,
- STRONGLY_CONNECTED,
- SUM,
- TOUR.

A constraint for which a reference provides a linear relaxation (see, e.g., the ALLDIFFERENT, the CIRCUIT, the CUMULATIVE, the SUM, and the REGULAR [134] constraints) or a constraint for which the flow model was derived by reformulating the constraint as a linear program (see, e.g., the AMONG_SEQ and the SLIDING_SUM constraints), or a constraint that was also proposed within the context of linear programming (see, e.g., the CIRCUIT, and DOMAIN_CONSTRAINT constraints). In the context of linear programming the book of John N. Hooker [226] provides a significant set of relaxations for a number of global constraints.



(A) Full search tree

(B) Subtree with a discrepancy of 0

(C) Subtree with a discrepancy of 1

(D) Subtree with a discrepancy of 2

(E) Subtree with a discrepancy of 3

(F) Subtree with a discrepancy of 4

Figure 3.38: Illustration of limited discrepancy search

3.7.137 Line segments intersection [3 CONS]

- CROSSING,
- GRAPH_CROSSING,
- TWO_LAYER_EDGE_CROSSING.

A constraint on the number of line segment intersections.

3.7.138 Logic [17 CONS]

- CONTAINS_SBOXES,
- COVEREDBY_SBOXES,
- COVERS_SBOXES,
- DISJOINT_SBOXES,
- EQUAL_SBOXES,
- GEOST,
- GEOST_TIME,
- INSIDE_SBOXES,
- MEET_SBOXES,
- NON_OVERLAP_SBOXES,
- ORTH_ON_TOP_OF_ORTH,
- OVERLAP_SBOXES,
- PLACE_IN_PYRAMID,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_COLUMN,
- TWO_ORTH_DO_NOT_OVERLAP,
- TWO_ORTH_INCLUDE.

A constraint which can be defined with first order logic formula encoded in the dedicated language introduced in [107].

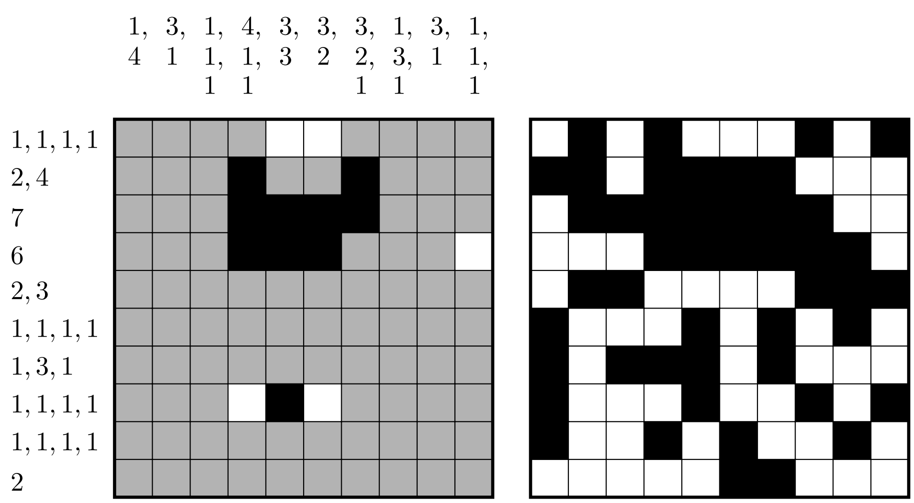
3.7.139 Logigraphe [1 CONS]

- CONSECUTIVE_GROUPS_OF_ONES.

A constraint which can be used for modelling the logigraphe problem. The *logigraphe* problem, see Figure 3.39 for an instance taken from [328, page 36], consists of colouring a board of squares in black or white, so that each row and each column contains a specific number of sequences of black squares of given size. A sequence of integers $s_1, s_2, \ldots, s_m$ $(p \geq 1)$ enforces:

- a first block of s_1 consecutive black squares,
- a second block of s_2 consecutive black squares,
- ....,
- a last block of s_p consecutive black squares.

Each block of consecutive black squares must be separated by at least one white square. Finally, white squares may possibly precede (respectively follow) the first (respectively the last) block of black squares. The *logigraphe* problem is NP-complete [422].



(A) (B)

Figure 3.39: Part (A): an instance of a logigraphe and the initial deductions achieved after posting the constraints, Part (B): the corresponding unique solution.

Part (A) of Figure 3.39 shows an instance of a logigraphe and the corresponding initial deductions achieved after posting the CONSECUTIVE_GROUPS_OF_ONES constraints associated with each row and each column. We assume that each constraint achieves arc-consistency, which is actually the case when the

CONSECUTIVE_GROUPS_OF_ONES constraint is represented as a counter free automaton. A white or black square indicates an initial deduction (i.e., setting a variable to 0 or to 1). Part (B) of Figure 3.39 provides the unique solution found after developing three choices,[14] assuming that variables are assigned from the uppermost to the lowermost row. Within a given row, variables are assigned from the leftmost to the rightmost column. Value 0 is tried first before value 1. Seven additional choices are required for proving that this solution is unique. Figure 3.40 displays the corresponding search tree. Within this figure, a variable $V_{i,j}$ $(1 \leq i, j \leq 10)$ denotes the 0-1 variable associated with the i^{th} row and the j^{th} column of the board.

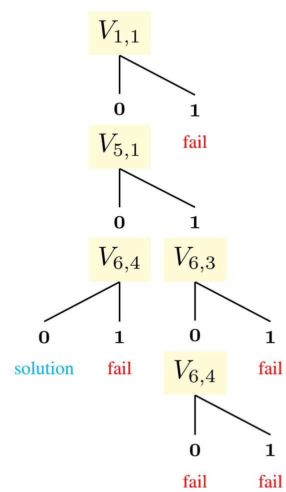


Figure 3.40: Search tree developed for the logigraphe instance of Figure 3.39 (variables that are fixed by propagation were removed from the search tree)

[14]Each time we try to assign a value to a not yet fixed variable, the number of choices is incremented by 1 just before making the assignment.

3.7.140 Magic hexagon [2 CONS]

- ALLDIFFERENT,
- GLOBAL_CARDINALITY_WITH_COSTS.

A constraint that can be used for modelling some parts of the magic hexagon problem. The *magic hexagon* problem, see Figure 3.41 for an example, consists of finding an arrangement of n hexagons, where an integer from 1 to n is assigned to each hexagon so that (1) each integer from 1 to n occurs exactly once, (2) the sum of the numbers along any straight line is the same.

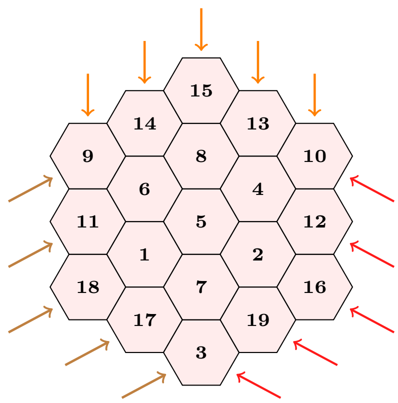


Figure 3.41: A magic hexagon of order 3 filled by integers 1 through 19 where the sum of the integers in each row of cells, in all three directions, is 38

3.7.141 Magic series [1 CONS]

- GLOBAL_CARDINALITY.

A constraint that allows for modelling the *magic series* problem with a single constraint. A non-empty finite sequence $S = (s_0, s_1, \ldots, s_n)$ is *magic* if and only if there are s_i occurrences of i in S for each integer i ranging from 0 to n. $3, 2, 1, 1, 0, 0, 0$ is an example of such a magic series for $n = 6$.

3.7.142 Magic square [2 CONS]

- ALLDIFFERENT,
- GLOBAL_CARDINALITY_WITH_COSTS.

A constraint that can be used for modelling some parts of the magic square problem. The *magic square* problem consists in filling an n by n square with n^2 distinct integers so that the sum of each row and column and of both main diagonals be the same.

3.7.143 Matching [4 CONS]

- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_EXCEPT_0,
- SYMMETRIC_ALLDIFFERENT_LOOP,
- TOUR.

A constraint that allows for expressing that we want to find a *perfect matching* on a graph with an even number of vertices. A *perfect matching* on a graph G with n vertices is a set of $n/2$ edges of G such that no two edges have a vertex in common.

A used generalisation so called *degree-matching* of a graph is a spanning sugraph where every vertex is associated with the bound degree of the matched edges.

3.7.144 Matrix [5 CONS]

- ALLPERM,
- COLORED_MATRIX,
- ELEMENT_MATRIX,
- LEX2,
- STRICT_LEX2.

A constraint on a matrix of domain variables (see, e.g., the ALLPERM, COLORED_MATRIX, LEX2, and STRICT_LEX2 constraints) or a constraint that allows for representing the access to an element of a matrix (see, e.g., the ELEMENT_MATRIX constraint).

3.7.145 Matrix model [4 CONS]

- ALLPERM,
- COLORED_MATRIX,
- LEX2,
- STRICT_LEX2.

A constraint on a matrix of domain variables. A *matrix model* is a model involving one matrix of domain variables.

3.7.146 Matrix symmetry [13 CONS]

- ALLPERM,
- INCREASING_GLOBAL_CARDINALITY,
- LEX2,
- LEX_CHAIN_GREATER,
- LEX_CHAIN_GREATEREQ,
- LEX_CHAIN_LESS,
- LEX_CHAIN_LESSEQ,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LEX_LESSEQ_ALLPERM,
- STRICT_LEX2.

A constraint that can be used for breaking certain types of symmetries within a matrix of domain variables.

3.7.147 Maximum [6 CONS]

- MAX_INDEX,
- MAX_N,
- MAX_NVALUE,
- MAX_SIZE_SET_OF_CONSECUTIVE_VAR,
- MAXIMUM,
- MAXIMUM_MODULO.

A constraint for which the definition involves the notion of maximum.

3.7.148 Maximum clique [4 CONS]

- ALL_MIN_DIST,
- ALLDIFFERENT,
- CLIQUE,
- DISJUNCTIVE.

A constraint (i.e., CLIQUE) that can be used for searching for a *maximum clique* in a graph, or a constraint (i.e., ALL_MIN_DIST, ALLDIFFERENT, DISJUNCTIVE) that can be stated by extracting a large clique [97] from a specific graph of elementary constraints.

A *maximum clique* is a clique of maximum size, a clique being a subset of vertices such that each vertex is connected to all other vertices of the clique.

3.7.149 Maximum number of occurrences [1 CONS]

- MAX_NVALUE.

A constraint that restricts the maximum number of times that a given value is taken.

3.7.150 maxint [4 CONS]

- DEEPEST_VALLEY,
- MIN_N,
- MINIMUM,
- MINIMUM_MODULO.

A constraint that uses maxint in its definition in terms of graph properties or in terms of automata. maxint is the largest integer that can be represented on a machine.

3.7.151 Metro [1 CONS]

- LEQ_CST.

A constraint that can be used for modelling the *metro problem*, i.e., finding the shortest distance from a given metro station to all other stations of the network.

Given an undirected graph $G = (V, E)$, with a non-negative distance attached to each edge of E, a conjunction of LEQ_CST constraints was used by H. Simonis in order to illustrate how propagation for such a conjunction simulates a naïve version of Dijsktra algorithm for computing the shortest distance from a given vertex v_s of V to all other vertices. The potential source of inefficiency comes from the fact that, depending on the scheduling policy of the underlying constraint engine, an inequality constraint can be reconsidered several times before reaching the fixed point. The problem was modelled in the following way:

- To each vertex $v_i \in V$ we associate a *distance variable* D_i, which represents the domain range of the distance between vertex v_i and vertex v_s.

- To each edge $(v_i, v_j) \in E$ we impose two inequality constraints $D_i \leq D_j + d_{i,j}$ and $D_j \leq D_i + d_{i,j}$, where $d_{i,j}$ corresponds to the distance attached to edge (v_i, v_j). This restricts the maximum difference between the distances variables associated with the two extremities of edge (v_i, v_j).

- Finally, we set the distance variable attached to vertex v_s to 0. Propagating the inequalities constraints by using arc-consistency enforces the maximum value of each distance variable D_i to be equal to the shortest distance from vertex v_i to v_s when the fixed point is reached.

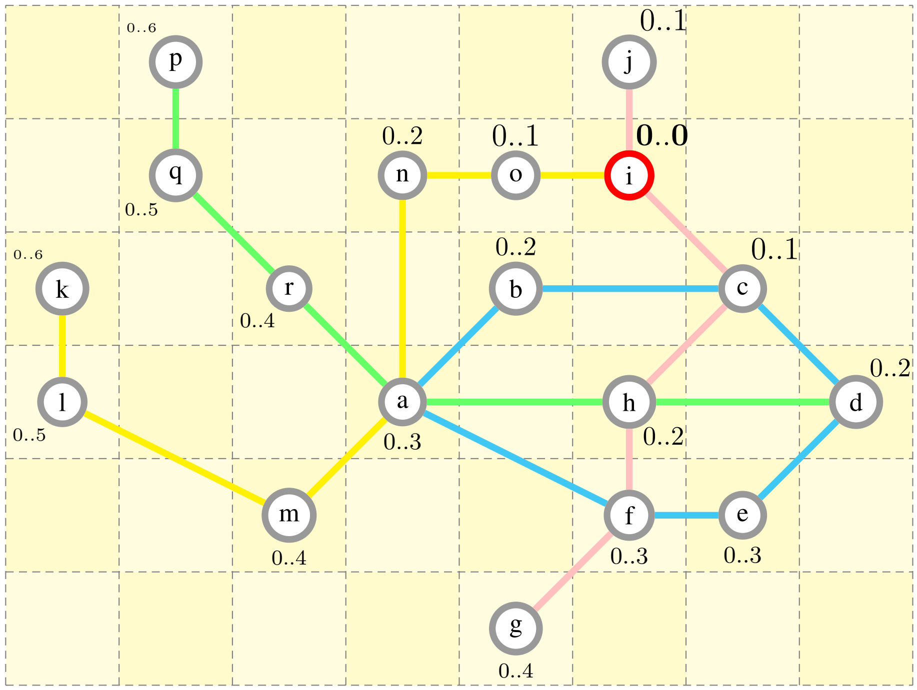


Figure 3.42: A metro map composed of four lines (a blue, a pink, a green and a yellow line) and the corresponding minimum and maximum values of the distance variables attached to each station, under the assumptions (1) that the distance attached to each connection is equal to 1 and (2) that we compute the shortest path from station **i** (in red); the font size used for displaying the bounds of a distance variable is inversely proportional to the length of the shortest path to station **i**.

Figure 3.42 illustrates this problem on a metro map composed of four lines and 18 stations respectively labelled by **a**, **b**, ..., **r**. Its assumes that the distance associated with each connection is equal to 1. The figure displays the status (i.e., the minimum and maximum values) of the distance variables under the assumption that we want to compute the shortest path from station **i**. The inequalities constraints between the distance variables $D_a, D_b, \ldots, D_r$ corresponding to this metro map are:

- *(constraints attached to the connections of the blue metro line)*
  - $D_a \leq D_b + 1$,
  - $D_b \leq D_a + 1$,
  - $D_b \leq D_c + 1$,
  - $D_c \leq D_b + 1$,
  - $D_c \leq D_d + 1$,
  - $D_d \leq D_c + 1$,
  - $D_d \leq D_e + 1$,
  - $D_e \leq D_d + 1$,
  - $D_e \leq D_f + 1$,
  - $D_f \leq D_e + 1$,
  - $D_f \leq D_a + 1$,
  - $D_a \leq D_f + 1$.

- *(constraints attached to the connections of the pink metro line)*
  - $D_g \leq D_f + 1$,
  - $D_f \leq D_g + 1$,
  - $D_f \leq D_h + 1$,
  - $D_h \leq D_f + 1$,
  - $D_h \leq D_c + 1$,
  - $D_c \leq D_h + 1$,
  - $D_c \leq D_i + 1$,
  - $D_i \leq D_c + 1$,
  - $D_i \leq D_j + 1$,
  - $D_j \leq D_i + 1$.
- *(constraints attached to the connections of the green metro line)*
  - $D_p \leq D_q + 1$,
  - $D_q \leq D_p + 1$,
  - $D_q \leq D_r + 1$,
  - $D_r \leq D_q + 1$,
  - $D_r \leq D_a + 1$,
  - $D_a \leq D_r + 1$,
  - $D_a \leq D_h + 1$,
  - $D_h \leq D_a + 1$,
  - $D_h \leq D_d + 1$,
  - $D_d \leq D_h + 1$.
- *(constraints attached to the connections of the yellow metro line)*
  - $D_k \leq D_l + 1$,
  - $D_l \leq D_k + 1$,
  - $D_l \leq D_m + 1$,
  - $D_m \leq D_l + 1$,
  - $D_m \leq D_a + 1$,
  - $D_a \leq D_m + 1$,
  - $D_a \leq D_n + 1$,
  - $D_n \leq D_a + 1$,
  - $D_n \leq D_o + 1$,
  - $D_o \leq D_n + 1$,
  - $D_o \leq D_i + 1$,
  - $D_i \leq D_o + 1$.

3.7.152 Minimum [12 CONS]

- MIN_INDEX,
- MIN_SIZE_SET_OF_CONSECUTIVE_VAR,
- MIN_N,
- MINIMUM,
- MIN_NVALUE,
- MINIMUM_EXCEPT_0,

- MINIMUM_GREATER_THAN,
- MINIMUM_MODULO,
- NEXT_ELEMENT,
- NEXT_GREATER_ELEMENT,
- OPEN_MAXIMUM,
- OPEN_MINIMUM.

A constraint for which the definition involves the notion of minimum.

3.7.153 Minimum cost flow [2 CONS]

- SOFT_ALLDIFFERENT_CTR,
- SOFT_SAME_VAR.

A constraint for which there is a filtering algorithm based on an algorithm that finds a minimum cost flow in a graph. This graph is usually constructed from the variables of the constraint as well as from their potential values. Figure 3.43 illustrates the minimum cost flow model used for the SOFT_SAME_VAR constraint. The demand and the capacity of the arcs are depicted by an interval on top of the corresponding arcs. The weight is given after that interval: a weight of 0 (respectively 1) is depicted by a dotted (respectively plain) arc. Weights of 1 are assigned to arcs linking two values since they model the correction of a discrepancy between variables x_1, x_2, x_3 and variables y_1, y_2, y_3. Blue arcs represent the feasible flow corresponding to the solution SOFT_SAME_VAR$(2, \langle 1,3,3\rangle, \langle 2,2,3\rangle)$.

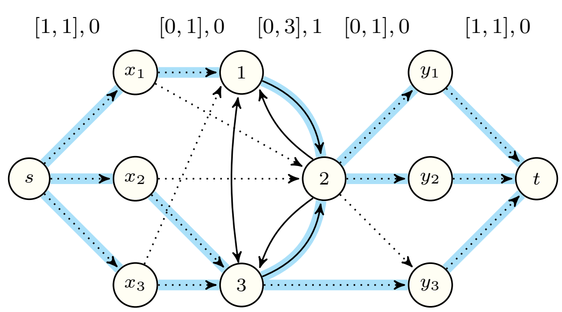


Figure 3.43: Minimum cost flow model for the SOFT_SAME_VAR constraint described in Table 3.18

Table 3.18: Domains of the variables for the SOFT_SAME_VAR constraint of Figure 3.43.

i	$dom(x_i)$	i	$dom(y_i)$
1	$\{1,2\}$	1	$\{2\}$
2	$\{2,3\}$	2	$\{2\}$
3	$\{1,3\}$	3	$\{2,3\}$

3.7.154 Minimum task duration [7 CONS]

- COLOURED_CUMULATIVE,
- COLOURED_CUMULATIVES,
- CUMULATIVE,
- CUMULATIVE_PRODUCT,
- CUMULATIVES,
- DISJUNCTIVE,
- TASKS_INTERSECTION.

A constraint involving one or several collections of tasks, where each task has an origin o, a duration d and an end e linked by the constraint $e = o + d$. From now on we assume that bound-consistency was achieved on constraint $e = o + d$. Assuming that the minimum duration of the task is always equal to the minimum value $\underline{d}$ is not quite true and can weaken reasoning as illustrated by the following examples:

- Using interval $[\overline{o}, \underline{o} + \underline{d}[$ as the compulsory part of the task is an underestimation of the compulsory part when the quantity $\underline{o} + \underline{d}$ is strictly less than the earliest end $\underline{e}$ of the task. To avoid that underestimation one should always use $[\overline{o}, \underline{e}[$ when the duration of the task is not fixed. The problem is that, depending where the task starts, its duration is not necessarily equal to its minimum value as the task has to finish at least at time $\underline{e}$.

- Using the smallest duration $\underline{d}$ for computing the minimum intersection of the task with a fixed interval under the hypothesis that the task starts at a given instant may also leads to an underestimation for the same reason. Figure 3.44 shows how the minimum duration of a task varies with respect to the origin of the task and to constraint $e = o + d$.

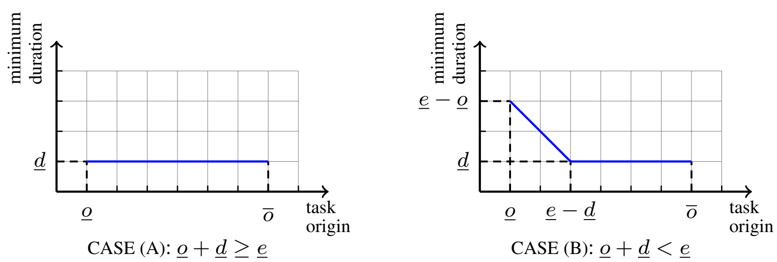


Figure 3.44: The two cases (A) and (B) showing how the minimum duration of a task varies wrt its origin o and to constraint $e = o + d$

3.7.155 Minimum feedback vertex set [1 CONS]

- CUTSET.

Denotes that a constraint is related to the *minimum feedback vertex set problem*: given a connected graph $G = (V, E)$, find out a minimum cardinality subset V' of V such that the graph G' induced by $V \setminus V'$ does not contain any cycle. A survey on the *feedback vertex set problem* is given in [176].

3.7.156 Minimum hitting set cardinality [1 CONS]

- NVALUE.

Denotes that, by reduction to the problem of finding the *cardinality of a minimum hitting set*, deciding whether a constraint has a solution or not, or getting a sharp lower bound for one of its arguments, was shown to be NP-hard. The *cardinality of a minimum hitting set* problem can be described as follows: given a collection C of subsets of a set S, find the minimum cardinality of $S' \subseteq S$ such that S' contains at least one element from each subset in C.

3.7.157 Minimum number of occurrences [1 CONS]

- MIN_NVALUE.

A constraint that restricts the minimum number of times that a given value is taken.

3.7.158 Modulo [12 CONS]

- ALLDIFFERENT_MODULO,
- AMONG_MODULO,
- BALANCE_MODULO,
- COMMON_MODULO,
- K_SAME_MODULO,
- K_USED_BY_MODULO,
- MAXIMUM_MODULO,
- MINIMUM_MODULO,
- SAME_MODULO,
- SOFT_SAME_MODULO_VAR,
- SOFT_USED_BY_MODULO_VAR,
- USED_BY_MODULO.

Denotes that the arc constraint associated with a given constraint mentions the function mod.

3.7.159 Multi-site employee scheduling with calendar constraints [3 CONS]

- CALENDAR,
- DIFFN,
- GEOST.

An international software company located in France and Germany has offices in Paris, Lyon and Marseille as well as in Berlin, Hamburg and Munich. Four types of activities are performed by its employees, namely (1) software development, (2) software deployment, (3) software training courses, and (4) business trips. Software developments tasks and training courses are performed within company's offices, while software deployment and business trips are done at customer's sites. Scheduling activities to employees is typically done on a yearly basis from Jan. 1 of current year to Apr. 30 of next year. Considering the first four months of the next year is done in order to absorb eventual overload and to anticipate the effect of Christmas and winter vacations. Without loss of generality we assume that our planning period is from Jan. 1, 2010 to Apr. 30, 2011. The level of granularity is the individual day. Since employees are located on different home sites, one has to consider the following holidays:

- *Public holidays* that do not fall on a weekend (i.e., a Saturday or a Sunday) are listed below.
  - France: Jan. 1, Apr. 5, May 13, May 24, July 14, Nov. 1, Nov. 11 in 2010, and Apr. 25 in 2011.

- Germany: Jan. 1, Apr. 2, Apr. 5, May 13, May 24 in 2010, and Apr. 22, Apr. 25 in 2011.

- In the context of Germany, *regional holidays* related to the federal state where a home site is situated. For Munich (Bavaria) we have the following additional days off, that all fall outside a weekend: Jan. 6, June 3, Nov. 1 in 2010 and Jan. 6 in 2011.

- Each home site is closed for a known fixed period of nine consecutive days that is located during summer school vacations. In addition each employee has five consecutive days off, a priori known, crossing winter school vacation. Summer and winter school vacations are linked to the country and the area where a home site is located. Regarding school vacations, France is partitioned in three zones, while Germany is divided in 16 federal states. Paris, Lyon and Marseille are located in distinct zones, while Berlin, Hamburg and Munich are situated in different federal states. *Summer vacations periods* are:

  - From July 3, 2010 to Sept. 1, 2010 in Paris, Lyon and Marseille.
  - From July 7, 2010 to Aug. 21, 2010 in Berlin.
  - From July 8, 2010 to Aug. 18, 2010 in Hamburg.
  - From Aug. 2, 2010 to Sept. 13, 2010 in Munich.

  Winter vacations periods are:

  - From Feb. 20, 2010 to Mar. 7, 2010 and from Feb. 13, 2011 to Feb. 27, 2011 in Paris.
  - From Feb. 14, 2010 to Feb. 28, 2010 and from Feb. 27, 2011 to Mar. 13, 2011 in Lyon.
  - From Feb. 7, 2010 to Feb. 21, 2010 and from Feb. 20 2011 to Mar. 7, 2011 in Marseille.
  - From Feb. 1, 2010 to Feb. 6, 2010 and from Jan. 31, 2011 to Feb. 5, 2011 in Berlin.
  - Jan. 29, 2010 and Jan. 31, 2011 in Hamburg.
  - From Feb. 15, 2010 to Feb. 20, 2010 and from Mar. 7, 2011 to Mar. 11, 2011 in Munich.

The goal is to schedule a given set of known tasks to employees in such a way that each employee has 30 days off in 2010, some of them corresponding to the mandatory public and regional holidays depending of the home site of an employee. Each task has:

1. A *type* (i.e., software development, software deployment, software training courses, and business trips).
2. An *earliest start* in 2010.

3. A *latest end* in 2010. Tasks which cannot be allocated with respect to their 2010 time windows must be scheduled in early 2011, i.e., from Jan. 1, 2011 to Apr. 30, 2011.

4. A *duration*.

5. A *number of required employees*.

6. A *list of home sites qualified to perform the task*.

Business trips, training courses and software deployment cannot be interrupted at all, while software development tasks cannot be interrupted by summer vacation. Business trips have to start on a Monday or a Tuesday since the general company policy is to prevent people staying abroad during weekends. Each task has to be allocated to employees, which are all based on the same home site, in such a way that the same set of employees takes care of the task from its start towards its completion. Each employee has:

1. A *home site* (i.e., Paris, Lyon, Marseille, Berlin, Hamburg or Munich).

2. A *five days period of winter 2010 vacation*.

3. A *five days period of winter 2011 vacation*.

4. A *list of task types* (i.e., software development, software deployment, software training courses, business trips) it can handle.

Finally, each home site has a *nine days period of summer 2010 vacation* where the home site is closed down.

3.7.160 Multiset [6 CONS]

- K_SAME,
- K_USED_BY,
- SAME,
- SAME_AND_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP,
- USED_BY.

A constraint using domain variables that can be used for modelling some constraint between multisets.

3.7.161 Multiset ordering [4 CONS]

- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ.

Similar constraints exist also within the context of multisets.

3.7.162 No cycle [1 CONS]

- PROPER_FOREST.

A constraint enforcing the fact that an undirected graph has no cycle.

3.7.163 No loop [32 CONS]

- ALL_DIFFER_FROM_AT_LEAST_K_POS,
- ALL_DIFFER_FROM_AT_MOST_K_POS,
- ALL_DIFFER_FROM_EXACTLY_K_POS,
- ALLDIFFERENT_ON_INTERSECTION,
- ALL_INCOMPARABLE,
- AMONG_LOW_UP,
- AMONG_VAR,
- ARITH_OR,
- ASSIGN_AND_COUNTS,
- ASSIGN_AND_NVALUES,
- BIN_PACKING,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST_PARTITION,
- CARDINALITY_ATMOST,
- CHANGE_CONTINUITY,
- CHANGE_PAIR,
- CHANGE_PARTITION,
- CHANGE,
- COMMON_INTERVAL,
- COMMON_MODULO,
- COMMON_PARTITION,
- COMMON,

- CORRESPONDENCE,
- COUNTS,
- CROSSING,
- CUTSET,
- CYCLIC_CHANGE_JOKER,
- CYCLIC_CHANGE,
- DECREASING,
- INVERSE_WITHIN_RANGE,
- LEX_EQUAL,
- TWO_ORTH_DO_NOT_OVERLAP,
- USES.

Denotes a constraint defined by a graph constraint for which the final graph does not have any loop.

3.7.164 n-Amazons [4 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_CST,
- INVERSE,
- SMOOTH.

A constraint that can be used for modelling the n-*Amazons* problem. Place n Amazons on an n by n chessboard in such a way that no Amazon attacks another. We say that two columns (respectively two rows) of a chessboard are *almost adjacent* if and only if the two columns (respectively the two rows) are separated by a single column (respectively a single row). Two Amazons attack each other if at least one of the following conditions holds:

1. They are located on the same column, on the same row or on the same diagonal.
2. They are located either on adjacent columns and on almost adjacent rows, or on almost adjacent columns and on adjacent rows.

As shown by these conditions, an Amazon combines the movements of a queen and of a knight. Figure 3.45 illustrates the movements of an Amazon. The n-*Amazons* problem has no solution when n is smaller than 10.

We now show how to model the n-*Amazons* problem with six global constraints. We start from the model that is used for the n-queens problem. We associate to the i^{th} column of the chessboard a domain variable X_i that gives the row number where the corresponding queen is located.

- The fact that two Amazons should not be located on the same column, on the same row or on the same diagonal can be modelled as the conjunction of three ALLDIFFERENT constraints:

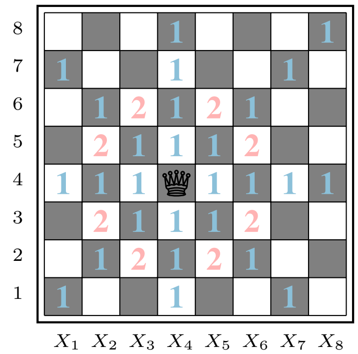


Figure 3.45: Illustration of the moves of an Amazon: moves labelled by 1 correspond to queen's moves, while moves labelled by 2 correspond to knight's moves

- ALLDIFFERENT$(X_1, X_2+1, \ldots, X_n+n-1)$ for the upper-left to lower-right diagonals,
- ALLDIFFERENT$(X_1, X_2, \ldots, X_n)$ for the rows,
- ALLDIFFERENT$(X_1+n-1, X_2+n-2, \ldots, X_n)$ for the lower-right to upper-left diagonals.

- The fact that two Amazons cannot both be located on adjacent columns and on almost adjacent rows can be modelled by disequality constraints of the form $|X_i - X_{i+1}| \neq 2$ $(1 \leq i \leq n-1)$.

- Similarly, the fact that two Amazons cannot both be located on almost adjacent columns and on adjacent rows can be modelled by disequality constraints of the form $|X_i - X_{i+2}| \neq 1$ $(1 \leq i \leq n-2)$. For a reason that will become clear later on, we rewrite this set of disequalities as $|X_{2\cdot i+1} - X_{2\cdot i+3}| \neq 1$ $(0 \leq i \leq \lfloor\frac{n-3}{2}\rfloor)$ and $|X_{2\cdot i} - X_{2\cdot i+2}| \neq 1$ $(1 \leq i \leq \lfloor\frac{n-2}{2}\rfloor)$.

If we combine the constraints of the form $|X_i - X_{i+1}| \neq 2$ $(1 \leq i \leq n-1)$ with the three ALLDIFFERENT constraints we get the conjunction of constraints $X_i - X_{i+1} \neq 0 \wedge |X_i - X_{i+1}| \neq 1 \wedge |X_i - X_{i+1}| \neq 2$ $(1 \leq i \leq n-1)$. This conjunction of three disequalities can be expressed as a single inequality of the form $|X_i - X_{i+1}| > 2$ $(1 \leq i \leq n-1)$. Furthermore all these inequalities can be combined into a single SMOOTH constraint of the form SMOOTH$(n-1, 2, \langle X_1, X_2, \ldots, X_n\rangle)$.[15] Similarly we get the constraints $|X_{2\cdot i+1} - X_{2\cdot i+3}| > 2$ $(0 \leq i \leq \lfloor\frac{n-3}{2}\rfloor)$ and $|X_{2\cdot i} - X_{2\cdot i+2}| > 2$ $(1 \leq i \leq \lfloor\frac{n-2}{2}\rfloor)$. Again we obtain two SMOOTH constraints of the form SMOOTH$(\lfloor\frac{n-1}{2}\rfloor, 2, \langle X_1, X_3, \ldots, X_{n-1+n \bmod 2}\rangle)$ and SMOOTH$(\lfloor\frac{n-2}{2}\rfloor, 2, \langle X_2, X_4, \ldots, X_{n-n \bmod 2}\rangle)$.

Finally, the INVERSE constraint can also be used as a channelling constraint if we want to create an additional variable for each row. This may be the case, for example,

[15] Since we enforce for all pairs of consecutive variables X_i, X_{i+1} $(1 \leq i \leq n-1)$ the constraint $|X_i - X_{i+1}| > 2$, the name SMOOTH seems odd. However the name SMOOTH stands from the situation where the number of inequalities constraints should be minimised.

if we want to have a heuristic for selecting first the column or the row that has the smallest number of possibilities.

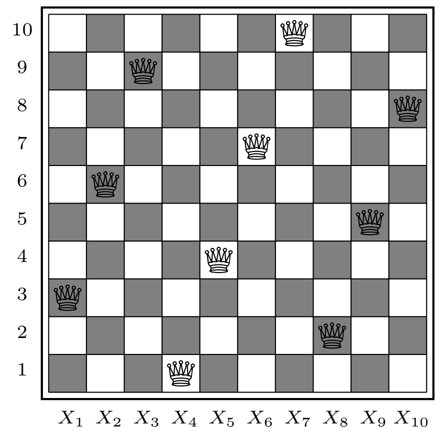


Figure 3.46: The unique solution to the 10-Amazons problem (modulo symmetries)

Figure 3.46 shows the unique solution, modulo symmetries, to the n-Amazons problem for $n = 10$. We have the following conjunction of constraints:

- ALLDIFFERENT_CST ($\langle \texttt{var} - X_1\ \texttt{cst} - 0, \texttt{var} - X_2\ \texttt{cst} - 1,$ $\texttt{var} - X_3\ \texttt{cst} - 2, \texttt{var} - X_4\ \texttt{cst} - 3, \texttt{var} - X_5\ \texttt{cst} - 4,$ $\texttt{var} - X_6\ \texttt{cst} - 5, \texttt{var} - X_7\ \texttt{cst} - 6, \texttt{var} - X_8\ \texttt{cst} - 7,$ $\texttt{var} - X_9\ \texttt{cst} - 8, \texttt{var} - X_{10}\ \texttt{cst} - 9 \rangle)$,
- ALLDIFFERENT$(\langle X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10} \rangle)$,
- ALLDIFFERENT_CST ($\langle \texttt{var} - X_1\ \texttt{cst} - 9, \texttt{var} - X_2\ \texttt{cst} - 8,$ $\texttt{var} - X_3\ \texttt{cst} - 7, \texttt{var} - X_4\ \texttt{cst} - 6, \texttt{var} - X_5\ \texttt{cst} - 5,$ $\texttt{var} - X_6\ \texttt{cst} - 4, \texttt{var} - X_7\ \texttt{cst} - 3, \texttt{var} - X_8\ \texttt{cst} - 2,$ $\texttt{var} - X_9\ \texttt{cst} - 1, \texttt{var} - X_{10}\ \texttt{cst} - 0 \rangle)$,
- SMOOTH$(\langle 9, 2, \langle X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10} \rangle)$,
- SMOOTH$(\langle 4, 2, \langle X_1, X_3, X_5, X_7, X_9 \rangle)$,
- SMOOTH$(\langle 4, 2, \langle X_2, X_4, X_6, X_8, X_{10} \rangle)$.

3.7.165 n-queens [3 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_CST,
- INVERSE.

A constraint that can be used for modelling the n-*queen* problem. Place n queens on an n by n chessboard in such a way that no queen attacks another. Two queens attack each other if they are located on the same column, on the same row or on the same diagonal. A constructive method for arbitrary $n > 3$ was first given in [171]. An effective heuristic for the n-*queen* problem was given in [240]. It consists of starting to place the queens in the centre of the chessboard so that they eliminate the maximum number of potential positions.

3.7.166 Non-deterministic automaton [3 CONS]

- AMONG,
- CHANGE,
- SMOOTH.

A constraint for which the catalogue provides a non-deterministic automaton without counters and without array of counters. For the mentioned constraints it turn out that non-determinism is due to the fact that we introduce transitions labelled by the potential values of a counting variable to a single accepting state (i.e., see Figures 5.63, 5.162, and 5.738).

3.7.167 Non-overlapping [9 CONS]

- DIFFN,
- DISJOINT_TASKS,
- GEOST,
- GEOST_TIME,
- ORTH_ON_TOP_OF_ORTH,
- ORTHS_ARE_CONNECTED,
- PLACE_IN_PYRAMID,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_DO_NOT_OVERLAP.

A constraint that forces a collection of geometrical objets to not pairwise overlap.

3.7.168 Number of changes [8 CONS]

- CHANGE,
- CHANGE_PAIR,
- CHANGE_PARTITION,
- CHANGE_VECTORS,
- CIRCULAR_CHANGE,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- SMOOTH.

A constraint restricting the number of times that a given binary constraint holds on consecutive items of a given collection.

3.7.169 Number of distinct equivalence classes [13 CONS]

- ATLEAST_NVALUE,
- ATLEAST_NVECTOR,
- ATMOST_NVALUE,
- ATMOST_NVECTOR,
- INCREASING_NVALUE,
- NCLASS,
- NEQUIVALENCE,
- NINTERVAL,
- NPAIR,
- NVALUE,

- NVALUES,
- NVECTOR,
- NVECTORS.

A constraint on the number of distinct equivalence classes assigned to a collection of domain variables.

3.7.170 Number of distinct values [11 CONS]

- ATLEAST_NVALUE,
- ATMOST_NVALUE,
- ASSIGN_AND_NVALUES,
- COLOURED_CUMULATIVE,
- COLOURED_CUMULATIVES,
- INCREASING_NVALUE,
- INCREASING_NVALUE_CHAIN,
- NVALUE,
- NVALUE_ON_INTERSECTION,
- NVALUES,
- NVALUES_EXCEPT_0.

A constraint on the number of distinct values assigned to one or several set of variables.

3.7.171 Obscure [9 CONS]

- CONTAINS_SBOXES,
- COVEREDBY_SBOXES,
- COVERS_SBOXES,
- DISJOINT_SBOXES,
- EQUAL_SBOXES,
- INSIDE_SBOXES,
- MEET_SBOXES,
- OVERLAP_SBOXES,
- TWO_LAYER_EDGE_CROSSING.

A constraint for which a better description is needed (i.e., TWO_LAYER_EDGE_CROSSING), or a constraint for which the definition needs to be checked: the eight topological relations of RCC-8 should be mutually incompatible, which is not the case with the current logic based definitions.

3.7.172 One succ [21 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_BETWEEN_SETS,
- ALLDIFFERENT_CST,
- ALLDIFFERENT_EXCEPT_0,
- ALLDIFFERENT_INTERVAL,
- ALLDIFFERENT_MODULO,
- ALLDIFFERENT_PARTITION,
- BALANCE_CYCLE,
- BALANCE_PATH,
- BALANCE_TREE,
- BINARY_TREE,
- CIRCUIT,
- CIRCUIT_CLUSTER,
- CYCLE,
- CYCLE_CARD_ON_PATH,
- DERANGEMENT,
- MINIMUM_WEIGHT_ALLDIFFERENT,
- PATH,
- PERMUTATION,
- PROPER_CIRCUIT,
- TREE.

Denotes that a constraint is defined by a single graph constraint such that:

- All the vertices of its initial graph belong to the final graph,
- All the vertices of its final graph have exactly one successor.

3.7.173 Open automaton constraint [2 CONS]

- OPEN_MAXIMUM,
- OPEN_MINIMUM.

A constraint for which the set of solutions can be recognised by a so called *open automaton*. An *open automaton* is a finite deterministic automaton taking as input a sequence of variables $V_1\ V_2 \ldots V_n$ as well as a sequence of 0-1 variables $B_1\ B_2 \ldots B_n$. A variable B_i $(1 \leq i \leq n)$ set to value 0 means that the corresponding variable V_i is removed from the sequence of variables $V_1\ V_2 \ldots V_n$.

Consider a constraint $\mathcal{C}$ for which we already have a finite deterministic automaton $\mathcal{A}$ that only accepts the set of solutions to $\mathcal{C}$. Constructing the finite deterministic automaton $\mathcal{A}'$ that only recognises the set of solutions to the open version of constraint C can be done in a systematic way from the automaton $\mathcal{A}$. First, to each transition of $\mathcal{A}$ we add the fact the corresponding Boolean variable must also be equal to 1. Second, to each state of $\mathcal{A}$ we add a loop transition for which the corresponding Boolean variable B_i $(1 \leq i \leq n)$ must be equal to 0 (since variable V_i is ignored, we stay within the same state). Figure 3.47 illustrates this construction in the context of the MINIMUM constraint and of its open counterpart, the OPEN_MINIMUM constraint.

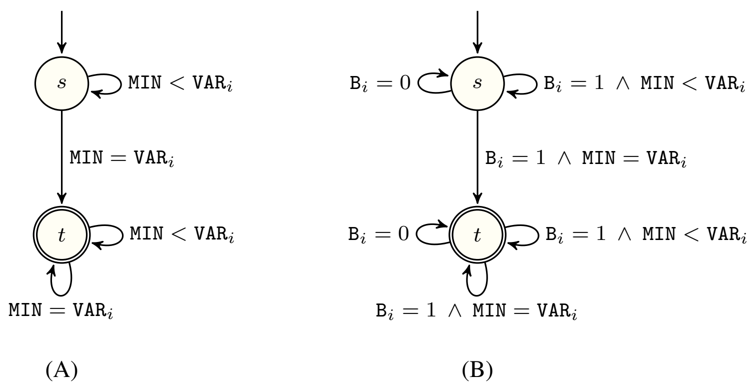


Figure 3.47: Constructing the (B) automaton of the OPEN_MINIMUM($\mathtt{MIN}, \langle \mathtt{VAR}_1\ \mathtt{B}_1, \mathtt{VAR}_2\ \mathtt{B}_2, \ldots, \mathtt{VAR}_n\ \mathtt{B}_n \rangle$) constraint from the (A) automaton of the MINIMUM($\mathtt{MIN}, \langle \mathtt{VAR}_1, \mathtt{VAR}_2, \ldots, \mathtt{VAR}_n \rangle$) constraint (an accepting state is denoted graphically by a double circle)

3.7.174 Open constraint [9 CONS]

- OPEN_ALLDIFFERENT,
- OPEN_AMONG,
- OPEN_ATLEAST,
- OPEN_ATMOST,
- OPEN_GLOBAL_CARDINALITY,
- OPEN_GLOBAL_CARDINALITY_LOW_UP,
- OPEN_MAXIMUM,
- OPEN_MINIMUM,
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT.

A constraint from which all its variables are not completely known when the constraint is posted [438]. In many situations, such as configuration, planning, or scheduling of process dependant activities, the variables of a constraint are not completely known initially when the constraint is posted. Instead, they are revealed during the search process [23, 172, 173]. In practice, an additional argument of the constraint (a set variable or a set of 0-1 variables) provides the initial set of potential variables (the lower bound in the context of a set variable). In Bartak's model [23], an open constraint admits a sequence of domain variables $V_1\ V_2 \ldots V_m\ (m \geq 1)$ as well as an additional variable C which gives the index of the last variable that effectively belongs to the constraint (i.e., variables $V_{C+1}, V_{C+2}, \ldots, V_m$ are discarded). This is the case, for example, for the SIZE_MAX_STARTING_SEQ_ALLDIFFERENT constraint.

Within the context of open constraints, the notion of *contractibility* was introduced in [283] in order to characterise a global constraint for which any pruning rule that removes a value from one of its variable (or which enforces any type of condition) can be reused in the context of the corresponding open global constraint (i.e., the pruning rule still makes valid deductions in the context of the open case). Intuitively, many global constraints which impose a kind of *at most* condition are contractible, while this is typically not the case for global constraints which enforce a kind of *at least* condition.

See also the keywords open automaton constraint, contractible, and extensible.

3.7.175 Order constraint [47 CONS]

- ALLPERM,
- COND_LEX_COST,
- COND_LEX_GREATER,
- COND_LEX_GREATEREQ,
- COND_LEX_LESS,
- COND_LEX_LESSEQ,
- DECREASING,
- INCREASING,
- INCREASING_GLOBAL_CARDINALITY,
- INCREASING_NVALUE,
- INCREASING_NVALUE_CHAIN,
- INCREASING_SUM,
- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- LEX2,
- LEX_BETWEEN,
- LEX_CHAIN_GREATER,
- LEX_CHAIN_GREATEREQ,
- LEX_CHAIN_LESS,
- LEX_CHAIN_LESSEQ,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LEX_LESSEQ_ALLPERM,
- MAX_INDEX,
- MAX_N,
- MAXIMUM,
- MAXIMUM_MODULO,
- MIN_INDEX,
- MIN_N,
- MINIMUM,
- MINIMUM_EXCEPT_0,
- MINIMUM_GREATER_THAN,
- MINIMUM_MODULO,
- NEXT_GREATER_ELEMENT,
- OPEN_MAXIMUM,
- OPEN_MINIMUM,
- ORDERED_ATLEAST_NVECTOR,
- ORDERED_ATMOST_NVECTOR,
- ORDERED_GLOBAL_CARDINALITY,
- ORDERED_NVECTOR,
- PRECEDENCE,
- SET_VALUE_PRECEDE,
- STRICT_LEX2,
- STRICTLY_DECREASING,
- STRICTLY_INCREASING.

A constraint involving an *ordering relation* in its definition. An *ordering relation* R on a set S is a relation such that, for every $a, b, c \in S$:

- $a\ R\ b$ or $b\ R\ a$,
- If $a\ R\ b$ and $b\ R\ c$, then $a\ R\ c$,
- If $a\ R\ b$ and $b\ R\ a$ then $a = b$.

3.7.176 Orthotope [12 CONS]

- DIFFN,
- DIFFN_COLUMN,
- DIFFN_INCLUDE,
- ORTH_LINK_ORI_SIZ_END,
- ORTH_ON_THE_GROUND,
- ORTH_ON_TOP_OF_ORTH,
- ORTHS_ARE_CONNECTED,
- PLACE_IN_PYRAMID,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_COLUMN,
- TWO_ORTH_DO_NOT_OVERLAP,
- TWO_ORTH_INCLUDE.

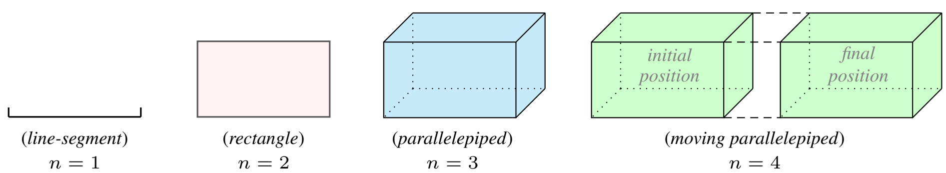


Figure 3.48: Illustration of the notion of orthotope for various number of dimensions n

A constraint involving *orthotopes*. An *orthotope* corresponds to the generalisation of the rectangle and box to the n-dimensional case. In addition its sides are parallel to the axes of the placement space. Figure 3.48 illustrates the notion of orthotope for $n = 1, 2, 3$ and 4. A collection usually named ORTHOTOPE, declared as $\texttt{ORTHOTOPE} - \texttt{collection}(\texttt{ori} - \texttt{dvar}, \texttt{siz} - \texttt{dvar}, \texttt{end} - \texttt{dvar})$, defines for each dimension d (with $d \in [1, n]$) the coordinate of its lower corner, the size and the coordinate of its upper corner in dimension d. Figure 3.49 illustrates the representation of an orthotope for $n = 2$.

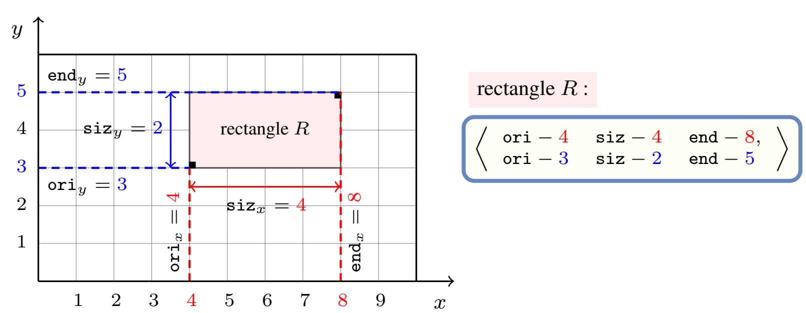


Figure 3.49: Representation of an orthotope when the number of dimensions $n = 2$ in term of the collection $\texttt{ORTHOTOPE}-\texttt{collection}(\texttt{ori}-\texttt{dvar}, \texttt{siz}-\texttt{dvar}, \texttt{end}-\texttt{dvar})$

3.7.177 Overlapping alldifferent [1 CONS]

- K_ALLDIFFERENT.

A constraint expressing several ALLDIFFERENT constraints having some variables in common.

3.7.178 Pair [3 CONS]

- CHANGE_PAIR,
- NPAIR,
- TWIN.

A constraint involving a collection of pairs of variables.

3.7.179 Packing almost squares [2 CONS]

- DIFFN,
- GEOST.

Denotes that a constraint can be used for solving the *packing almost squares* problem: tile a rectangle for which sides are consecutive integers by rectangles of size $1 \times 2, 2 \times 3, \ldots, n \times (n+1)$ which can be rotated by 90 degrees. The problem is described in http://www.stetson.edu/~efriedma/almost/. Since there does not always exist a tiling, one can also consider a variant where the goal is to find the rectangle with minimal area. Figure 3.50 provides a solution for $n = 26$ found by H. Simonis.

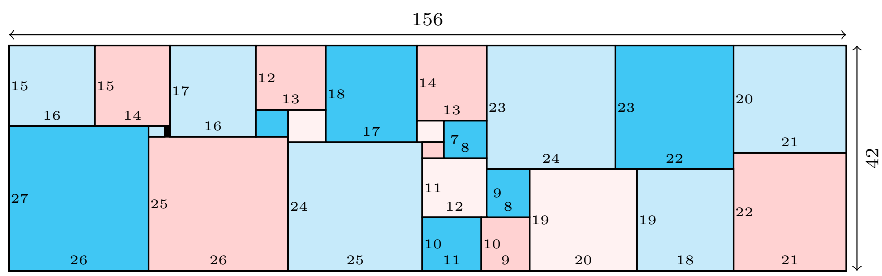


Figure 3.50: A solution to the packing almost squares problem for $n = 26$; the width and the height of each not too small rectangle are respectively shown on the lowest and leftmost borders of the rectangle.

3.7.180 Pallet loading [2 CONS]

- DIFFN,
- GEOST.

A constraint that can be used for modelling the *pallet loading* problem. The *pallet loading* problem consists of packing a maximum number of identical rectangular boxes onto a rectangular pallet in such a way that boxes are placed with their edges parallel to the edges of the pallet. The problem often arises in distribution, when many boxes must be shipped and an increase of the number of boxes on a pallet saves costs. Even though the complexity of the problem is not yet known [303], many solutions have been developed over the past years:

- Exact algorithms based on tree search procedures extend a partial solution by positioning a new box according to different heuristics. One of the most used heuristics is the so called G4 heuristic [384] which recursively divides the placement space into four huge rectangles. Beside the use of an appropriate heuristic, the key point is the use of upper bounds on the maximum number of boxes that can be packed. Some bounds like the Barnes [20] and the Keber [248] bounds consider the geometric structure of the problem. Some other bounds are obtained by solving a linear programming problem [233].

- Approximate algorithms are based on constructive methods (i.e., methods that either divide the pallet into blocks or methods that divide the pallet in a recursive way) or metaheuristics based on genetic algorithms or tabu search [7].

Both in the context of exact and approximates algorithms, the problem is usually first normalised in order to reduce the set of possible solutions [153, 154].

3.7.181 Partition [14 CONS]

- ALLDIFFERENT_PARTITION,
- BALANCE_PARTITION,
- CARDINALITY_ATMOST_PARTITION,
- CHANGE_PARTITION,
- COMMON_PARTITION,
- IN_SAME_PARTITION,
- K_SAME_PARTITION,
- K_USED_BY_PARTITION,
- NCLASS,
- SAME_PARTITION,
- STRETCH_PATH_PARTITION,
- SOFT_SAME_PARTITION_VAR,
- SOFT_USED_BY_PARTITION_VAR,
- USED_BY_PARTITION.

A constraint involving in one of its argument a partitioning of a given finite set of integers.

3.7.182 Path [4 CONS]

- BALANCE_PATH,
- PATH,
- PATH_FROM_TO,
- TEMPORAL_PATH.

A constraint allowing for expressing that we search for one or several vertex-disjoint *simple paths*. Within a digraph a *simple path* is a set of links that are traversed in the same direction and such that each vertex of the simple path is visited exactly once.

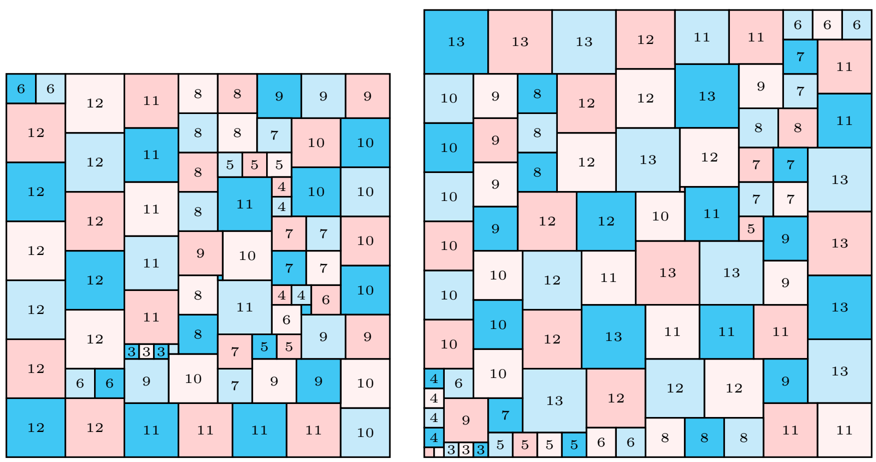
3.7.183 Partridge [2 CONS]

- DIFFN,
- GEOST.

Denotes that a constraint can be used for solving the *Partridge* problem: the Partridge problem consists of tiling a square of size $\frac{n\cdot(n+1)}{2}$ by $\frac{n\cdot(n+1)}{2}$ squares of respective size

- 1 square of size 1,
- 2 squares of size 2,
- $\ldots$,
- n squares of size n.

It was initially proposed by R. Wainwright and is based on the identity $1\cdot 1^2 + 2\cdot 2^2 + \cdots + n\cdot n^2 = (\frac{n\cdot(n+1)}{2})^2$. The problem is described in `http://mathpuzzle.com/partridge.html`. Part (A) of Figure 3.51 gives a solution for $n = 12$ found with GEOST [2], while Part (B) provides a solution for $n = 13$ found by S. Hougardy [230].



(A) (B)

Figure 3.51: (A) a solution to the Partridge problem for $n = 12$, and (B) a solution for $n = 13$

3.7.184 Pattern sequencing [1 CONS]

- CUMULATIVE_CONVEX.

A constraint allowing for expressing the *pattern sequencing problem* as a single global constraint. The *pattern sequencing problem* [177] can be described as follows: given a 0-1 matrix in which each column j $(1 \leq j \leq p)$ corresponds to a product required by the customers and each row i $(1 \leq i \leq c)$ corresponds to the order of a particular customer (The entry c_{ij} is equal to 1 if and only if customer i has ordered some quantity of product j.), the objective is to find a permutation of the products such that the maximum number of open orders at any point in the sequence is minimised. Order i is *open* at point k in the production sequence if there is a product required in order i that appears at or before position k in the sequence and also a product that appears at or after position k in the sequence.

3.7.185 Pentomino [3 CONS]

- DIFFN,
- GEOST,
- POLYOMINO,
- REGULAR.

A constraint (i.e., POLYOMINO) that can be used to model a *pentomino*. A *pentomino* is an arrangement of five unit squares that are joined along their edges.

Also denotes a constraint (i.e., DIFFN, GEOST, REGULAR) that can be used for solving tiling problems involving pentominoes. For example, the GEOST and REGULAR constraints where respectively used in [42] and in [257] to solve such tiling problems.

Figure 3.52 presents a tiling of a rectangle with distinct pentominoes.

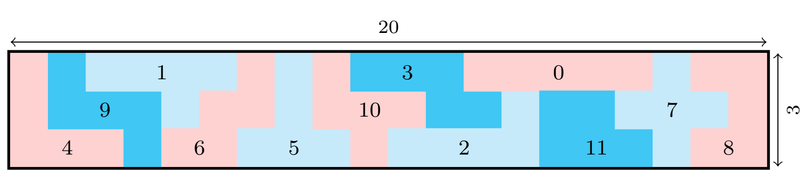


Figure 3.52: Tiling a rectangle with pentominoes

3.7.186 Periodic [3 CONS]

- PERIOD,
- PERIOD_EXCEPT_0,
- PERIOD_VECTORS.

A constraint that can be used for modelling the fact that we are looking for a sequence that has some kind of periodicity.

3.7.187 Permutation [27 CONS]

- ALLDIFFERENT,
- ALLDIFFERENT_CONSECUTIVE_VALUES,
- BALANCE_CYCLE,
- CHANGE_CONTINUITY,
- CIRCUIT,
- CIRCUIT_CLUSTER,
- CORRESPONDENCE,
- CYCLE,
- CYCLE_CARD_ON_PATH,
- DERANGEMENT,
- ELEMENTS_ALLDIFFERENT,
- INVERSE,
- K_ALLDIFFERENT,
- K_SAME,
- K_SAME_INTERVAL,
- K_SAME_MODULO,
- K_SAME_PARTITION,
- PROPER_CIRCUIT,
- SAME,
- SAME_AND_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP,
- SAME_INTERVAL,
- SAME_MODULO,
- SAME_PARTITION,
- SORT,
- SORT_PERMUTATION,
- SYMMETRIC_ALLDIFFERENT.

A constraint that can be used for modelling a permutation or a specific type or characteristic of a permutation. A *permutation* is a rearrangement of elements, where none are changed, added or lost.

3.7.188 Permutation channel [1 CONS]

- INVERSE.

A constraint that allows for modelling the link between a *permutation* and its *inverse permutation*. A *permutation* is a rearrangement of n distinct integers between 1 and n, where none are changed, added or lost. An *inverse permutation* is a permutation in which each number and the number of its position are swapped.

3.7.189 Phi-tree [2 CONS]

- DISJUNCTIVE,
- CUMULATIVE.

A constraint for which one of its filtering algorithms uses a balanced binary tree in order to efficiently evaluate the maximum or minimum value of a formula over all possible subsets of tasks Ω of a given set of tasks Φ. Φ-trees were introduced by P. Vilím, first in the context of unary resources in [443] and in [444, pages 37–40], and later on in the context of cumulative resources [446, 445]. Without loss of generality, let us sketch the main idea behind a Φ-tree in the context of a cumulative resource of capacity C. For this purpose we follow the description given in [446]. Given a set of tasks Φ where each task has an *earliest possible start*, a *latest possible end*, a *duration* and a *resource consumption*, assume we need to evaluate the *earliest completion time* over all tasks of Φ under the hypothesis that we should not exceed the maximum resource capacity C. Let us first introduce some notations:

- Ω denotes any non-empty subset of tasks of Φ.
- est_Ω is the minimum over the earliest starts of the tasks in Ω.
- e_Ω is the sum of the surfaces (i.e., the product of the duration by the resource consumption) of the tasks in Ω.

A common estimation of the earliest completion time over all tasks of Φ is $\max_{\Omega\subseteq\Phi}\left\{est_\Omega+\left\lceil\frac{e_\Omega}{C}\right\rceil\right\}$ which can be rewritten as $\left\lceil\frac{\max_{\Omega\subseteq\Phi}\{C est_\Omega+e_\Omega\}}{C}\right\rceil$. The numerator of the last fraction is called the *energy envelope* of the set of tasks Φ and the purpose of a Φ-tree is to evaluate this quantity efficiently. For a node n, let $\mathcal{L}(n)$ denotes the set of leaves of the sub-tree rooted at n. The leaves of the Φ-tree correspond

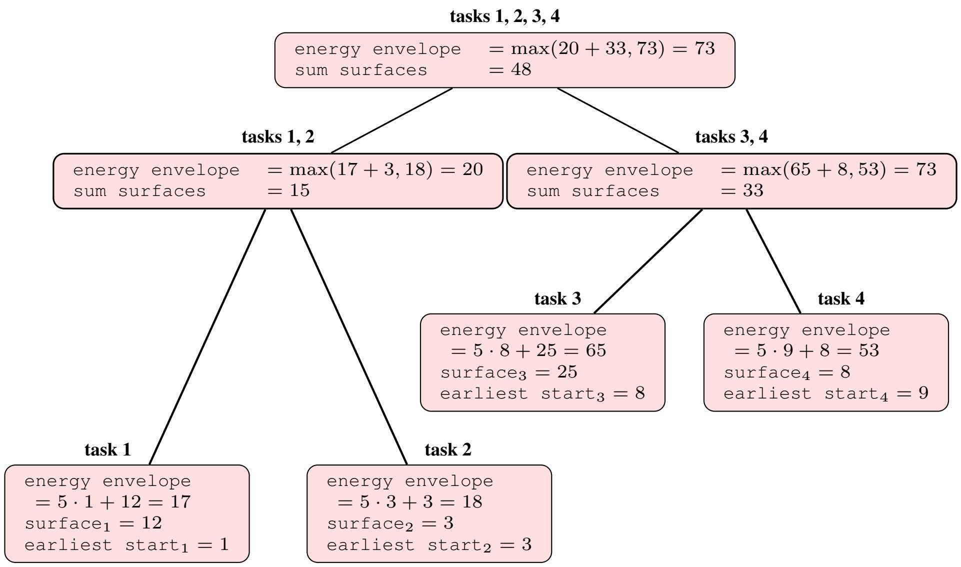


Figure 3.53: Example of Φ-tree associated with four tasks of respective duration and resource consumption 3×4, 1×3, 5×5, 2×4 and of respective earliest start 1, 3, 8, 9 under the assumption that the maximum capacity of the cumulative resource is equal to 5

to the tasks of Φ sorted from left to right by increasing earliest start. Each node n of the Φ-tree records both, the sum of the surfaces of the tasks in $\mathcal{L}(n)$, as well as the energy envelope of the tasks in $\mathcal{L}(n)$. The sum of the surfaces associated with a non-leave node n of the tree corresponds to the sum of the surfaces of the children of n, while the energy envelope of n is equal to the maximum between on the one hand, the energy envelop of its right child and on the other hand the sum of the energy envelop of its left child and the recorded sum of surfaces of its right child (see [446] for a justification of these recursive formulae). Figure 3.53 illustrates the construction of a Φ-tree associated with four given tasks.

3.7.190 Phylogeny [1 CONS]

- STABLE_COMPATIBILITY.

A constraint inspired by the area of phylogeny. Phylogeny is concerned by the classification of organism based on genetic connections between species.

3.7.191 Pick-up delivery [1 CONS]

- CYCLE.

A constraint that was used for modelling a *pick-up delivery problem*. In a *pick-up delivery problem*, vehicles have to transport loads from origins to destinations without any transhipment at intermediate locations.

3.7.192 Planarity test [1 CONS]

- CIRCUIT.

A constraint that can use the *planarity test* in its filtering algorithm. The *planarity test* determines whether a graph can be embedded in the plane.

3.7.193 Polygon [1 CONS]

- DIFFN.

A constraint that can be generalised to handle polygons.

3.7.194 Positioning constraint [4 CONS]

- DIFFN_COLUMN,
- DIFFN_INCLUDE,
- TWO_ORTH_COLUMN,
- TWO_ORTH_INCLUDE.

A constraint restricting the relative positioning of two or more geometrical objects.

3.7.195 Predefined constraint [72 CONS]

- ABS_VALUE,
- ATMOST1,
- BIN_PACKING_CAPA,
- CALENDAR,
- COLORED_MATRIX,
- COMPARE_AND_COUNT,
- CONSECUTIVE_VALUES,
- CUMULATIVE_TWO_D,
- DISTANCE,
- DIVISIBLE,
- DIVISIBLE_OR,
- DOM_REACHABILITY,
- DOMAIN,
- EQ,
- EQ_CST,
- EQ_SET,
- EQUILIBRIUM,
- GCD,
- GEOST,
- GEOST_TIME,
- GEQ,
- GEQ_CST,

- GRAPH_ISOMORPHISM,
- GT,
- IN_INTERVAL_REIFIED,
- IN_INTERVALS,
- IN_SET,
- INCOMPARABLE,
- INCREASING_SUM,
- LEQ,
- LEQ_CST,
- LEX2,
- LEX_ALLDIFFERENT_EXCEPT_0,
- LEX_LESSEQ_ALLPERM,
- LT,
- MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES,
- MAX_OCC_OF_SORTED_TUPLES_OF_VALUES,
- MAX_OCC_OF_TUPLES_OF_VALUES,
- MULTI_GLOBAL_CONTIGUITY,
- MULTI_INTER_DISTANCE,
- MULTIPLE,
- NEQ,
- NEQ_CST,
- NUMBER_DIGIT,
- OPPOSITE_SIGN,
- ORDER,
- PERIOD,
- PERIOD_EXCEPT_0,
- PERIOD_VECTORS,
- POWER,
- PROPER_CIRCUIT,
- REMAINDER,
- SAME_SIGN,
- SCALAR_PRODUCT,
- SET_VALUE_PRECEDE,
- SIGN_OF,
- SOFT_CUMULATIVE,
- STRICT_LEX2,
- SUBGRAPH_ISOMORPHISM,
- SUM_CUBES_CTR,
- SUM_FREE,
- SUM_OF_INCREMENTS,
- SUM_POWERS4_CTR,
- SUM_POWERS5_CTR,
- SUM_POWERS6_CTR,
- SUM_SQUARES_CTR,
- SYMMETRIC_ALLDIFFERENT_EXCEPT_0,
- TASKS_INTERSECTION,
- TWIN,
- VISIBLE,
- ZERO_OR_NOT_ZERO,
- ZERO_OR_NOT_ZERO_VECTORS.

A constraint for which the meaning is not explicitly described in terms of graph properties or in terms of automata or in terms of first order logic.

3.7.196 Preferences [5 CONS]

- COND_LEX_COST,
- COND_LEX_GREATER,
- COND_LEX_GREATEREQ,
- COND_LEX_LESS,
- COND_LEX_LESSEQ.

A constraint that can be used for modelling preferences.

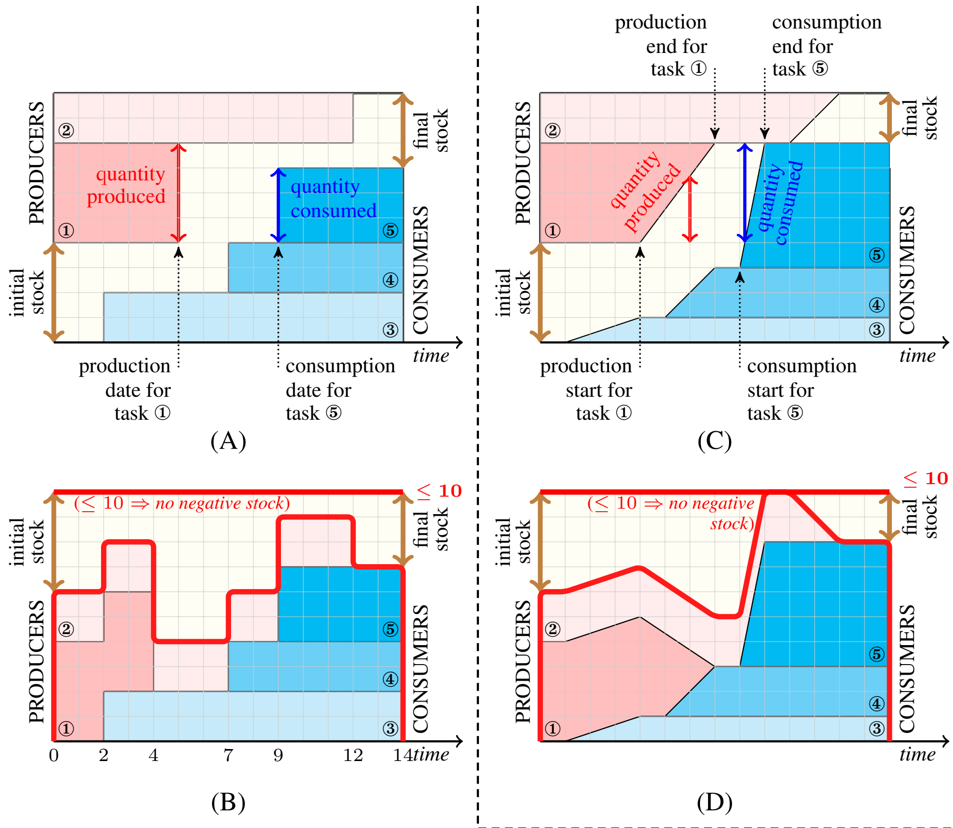
3.7.197 Producer-consumer [2 CONS]

- CUMULATIVE,
- CUMULATIVES.

A constraint that can be used for modelling problems where a first set of tasks produces a non-renewable resource, while a second set of tasks consumes this resource so that a limit on the minimum or the maximum stock at each instant is imposed.

Parts (A) and (B) of Figure 3.54 describes the simplest variant of the producer-consumer problem [399] where no negative stock is allowed. Given an initial stock, a first set of tasks (i.e., producers) add instantaneously their respective productions to the stock (when they are finished), and a second set of tasks (i.e., consumers) take instantaneously from the stock (when they start) the amount of non-renewable resource they need. The problem is to schedule these tasks (i.e., fix the end of the producers and fix the start of the consumers) and to fix for each task the quantity it produces or consumes, so that no negative stock occurs. Part (A) of Figure 3.54 describes an instance of such problem where we respectively have 2 producers and 3 consumers. Part (B) depicts the corresponding cumulative view of the problem. At each timepoint the difference between the top line and the top of the cumulated profile gives the amount of available stock at that timepoint.

A fundamental problem with the previous variant of the producer-consumer problem is that it does not allow to handle the fact that a resource is produced or used gradually. Parts (C) and (D) of Figure 3.54 describes a second variant where this is in fact possible. This is achieved by replacing the rectangle associated with a producer by a task with a decreasing height. At a given instant the cumulated quantity produced by a producer is the difference between the height of that task at its starting time and the height of that task at the considered instant. Conversely a consumer is modelled by a task with an increasing height. At a particular timepoint the cumulated quantity used by a consumer task is the difference between the height of that task at its end



CUMULATIVE(TASKS, 10) with TASKS:

①	o − 0	d − 4	e − 4	h − 4
②	o − 0	d − 12	e − 12	h − 2
③	o − 2	d − 12	e − 14	h − 2
④	o − 7	d − 7	e − 14	h − 2
⑤	o − 9	d − 5	e − 14	h − 3

(o for origin, d for duration, e for end, h for height) (E)

Figure 3.54: Producer-consumer models (A,C) and corresponding CUMULATIVE views (B,D) enforcing that, at any point in time, we do not have any negative stock, i.e. at any point in time we do not consume more that we have produced so far; (E) CUMULATIVE constraint associated with (B)

and the height of that task at the considered instant. Part (C) of Figure 3.54 describes an instance of such problem where, again, we respectively have 2 producers and 3 consumers. Part (D) depicts the corresponding cumulative view of the problem. As before, at each timepoint the difference between the top line and the top of the cumulated profile gives the amount of available stock at that timepoint.

3.7.198 Product [2 CONS]

- CUMULATIVE_PRODUCT,
- PRODUCT_CTR.

A constraint involving a product in its definition.

3.7.199 Program verification [1 CONS]

- CUTSET.

A constraint that was used within the application area of *program verification*.

3.7.200 Proximity constraint [3 CONS]

- ALLDIFFERENT_SAME_VALUE,
- DISTANCE_BETWEEN,
- DISTANCE_CHANGE.

A constraint restricting the distance between two collections of variables according to some measure.

3.7.201 Pure functional dependency [114 CONS]

- ALL_BALANCE,
- ABS_VALUE,
- AMONG,
- AMONG_DIFF_0,
- AMONG_INTERVAL,
- AMONG_MODULO,
- AMONG_VAR,
- AND,
- BALANCE,
- BALANCE_INTERVAL,
- BALANCE_MODULO,
- BALANCE_PARTITION,
- BIG_PEAK,
- BIG_VALLEY,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST,
- CARDINALITY_ATMOST_PARTITION,
- CHANGE,
- CHANGE_PAIR,
- CHANGE_PARTITION,
- CHANGE_VECTORS,
- CIRCULAR_CHANGE,
- COLORED_MATRIX,
- COMMON,
- COMMON_INTERVAL,
- COMMON_MODULO,
- COMMON_PARTITION,
- CROSSING,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- DEEPEST_VALLEY,
- DIFFER_FROM_EXACTLY_K_POS,
- DISCREPANCY,
- DISTANCE,
- DISTANCE_BETWEEN,
- DISTANCE_CHANGE,
- ELEM ,
- ELEMENT,
- ELEMENT_PRODUCT,
- ELEMENTS,
- EQ,
- EQ_CST,
- EQUIVALENT,
- EXACTLY,
- FULL_GROUP,
- GCD,
- GLOBAL_CARDINALITY,
- GLOBAL_CARDINALITY_NO_LOOP,
- GLOBAL_CARDINALITY_WITH_COSTS,
- GRAPH_CROSSING,
- GROUP,
- HIGHEST_PEAK,
- INFLEXION,
- IMPLY,
- INVERSE,
- INVERSE_EXCEPT_LOOP,
- INVERSE_OFFSET,
- LENGTH_FIRST_SEQUENCE,
- LENGTH_LAST_SEQUENCE,
- LONGEST_CHANGE,
- LONGEST_DECREASING_SEQUENCE,
- LONGEST_INCREASING_SEQUENCE,
- MAP,
- MAX_DECREASING_SLOPE,
- MAX_INCREASING_SLOPE,
- MAX_N,
- MAX_NVALUE,
- MAX_SIZE_SET_OF_CONSECUTIVE_VAR,
- MAXIMUM,
- MAXIMUM_MODULO,
- MIN_DECREASING_SLOPE,
- MIN_INCREASING_SLOPE,
- MIN_N,
- MIN_NVALUE,
- MIN_SIZE_SET_OF_CONSECUTIVE_VAR,
- MIN_SURF_PEAK,
- MIN_WIDTH_PEAK,
- MIN_WIDTH_PLATEAU,
- MIN_WIDTH_VALLEY,
- MINIMUM,
- MINIMUM_EXCEPT_0,

- MINIMUM_MODULO,
- NAND,
- NCLASS,
- NEQUIVALENCE,
- NINTERVAL,
- NOR,
- NPAIR,
- NSET_OF_CONSECUTIVE_VALUES,
- NUMBER_DIGIT,
- NVALUE,
- NVALUE_ON_INTERSECTION,
- NVECTOR,
- NVISIBLE_FROM_END,
- NVISIBLE_FROM_START,
- OR,
- ORCHARD,
- ORTH_LINK_ORI_SIZ_END,
- PEAK,
- PERIOD,
- PERIOD_EXCEPT_0,
- PERIOD_VECTORS,
- POWER,
- REMAINDER,
- SIGN_OF,
- SIZE_MAX_SEQ_ALLDIFFERENT,
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT,
- SMOOTH,
- SORT,
- STAGE_ELEMENT,
- SUM_OF_WEIGHTS_OF_DISTINCT_VALUES,
- TWO_LAYER_EDGE_CROSSING,
- VALLEY,
- XOR.

A constraint for which the meaning is completely captured by one or more functional dependancies. The negation of such constraints can be directly expressed as a disjunction between the different functional dependancies. We illustrate this point on different examples:

- The negation of the NVALUE$(n, \langle v_1, v_2, \ldots, v_m \rangle)$ constraint is defined by NVALUE$(p, \langle v_1, v_2, \ldots, v_m \rangle) \wedge n \neq p$.

- The negation of the COMMON$(n_1, n_2, \langle u_1, u_2, \ldots, u_p \rangle, \langle v_1, v_2, \ldots, v_q \rangle)$ constraint is defined by COMMON$(m_1, m_2, \langle u_1, u_2, \ldots, u_p \rangle, \langle v_1, v_2, \ldots, v_q \rangle) \wedge (n_1 \neq m_1 \vee n_2 \neq m_2)$.

- The negation of the ELEMENTS$(\langle \texttt{index} - i_1\ \texttt{value} - u_1, \texttt{index} - i_2\ \texttt{value} - u_2, \ldots, \texttt{index} - i_n\ \texttt{value} - u_n \rangle, \langle \texttt{index} - 1\ \texttt{value} - v_1, \texttt{index} - 2\ \texttt{value} - v_2, \ldots, \texttt{index} - n\ \texttt{value} - v_n \rangle)$ constraint is defined by ELEMENTS$(\langle \texttt{index} - i_1\ \texttt{value} - w_1, \texttt{index} - i_2\ \texttt{value} - w_2, \ldots, \texttt{index} - i_n\ \texttt{value} - w_n \rangle, \langle \texttt{index} - 1\ \texttt{value} - v_1, \texttt{index} - 2\ \texttt{value} - v_2, \ldots, \texttt{index} - n\ \texttt{value} - v_n \rangle) \wedge (u_1 \neq w_1 \vee u_2 \neq w_2 \vee \cdots \vee u_n \neq w_n)$.

- The negation of the SORT$(\langle u_1, u_2, \ldots, u_n \rangle, \langle v_1, v_2, \ldots, v_n \rangle)$ constraint is defined by SORT$(\langle u_1, u_2, \ldots, u_n \rangle, \langle w_1, w_2, \ldots, w_n \rangle) \wedge$ LEX_DIFFERENT$(\langle v_1, v_2, \ldots, v_n \rangle, \langle w_1, w_2, \ldots, w_n \rangle)$.

3.7.202 Quadtree [2 CONS]

- CUMULATIVE_TWO_D,
- DIFFN.

Denotes that, for a given constraint, a *quadtree* can be used within its filtering algorithm. A *quadtree* is a hierarchical data structure based on the recursive decomposition of space. Figure 3.55 illustrates the representation of a two-dimensional binary region (A) with a quadtree (C). A region is subdivided into quadrants, subquadrants, and so on (B), until blocks consist entirely of 1s or entirely of 0s.

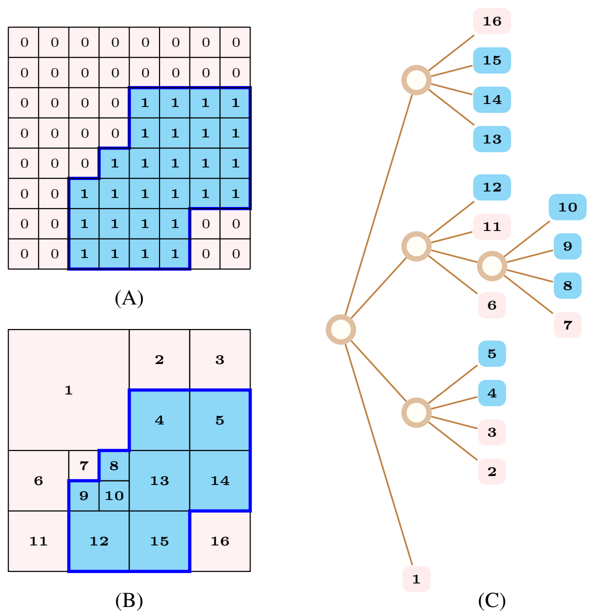


Figure 3.55: (A) A region, (B) its subdivision in maximal blocks, (C) and the corresponding quadtree

3.7.203 Range [1 CONS]

- RANGE_CTR.

An arithmetic constraint involving a difference between a maximum and a minimum value.

3.7.204 Rank [2 CONS]

- MAX_N,
- MIN_N.

A positioning constraint according to an ordering relation.

3.7.205 RCC8 [8 CONS]

- CONTAINS_SBOXES,
- COVEREDBY_SBOXES,
- COVERS_SBOXES,
- DISJOINT_SBOXES,
- EQUAL_SBOXES,
- INSIDE_SBOXES,
- MEET_SBOXES,
- OVERLAP_SBOXES.

Region Connection Calculus (i.e., RCC-8) [349] provides eight topological relations (i.e., *disjoint*, *meet*, *overlap*, *equal*, *covers*, *coveredby*, *contains*, *inside*) between two fixed objects such that any two fixed objects are in one and exactly one of these topological relations. Figure 3.56 illustrates the meaning of each topological relation.

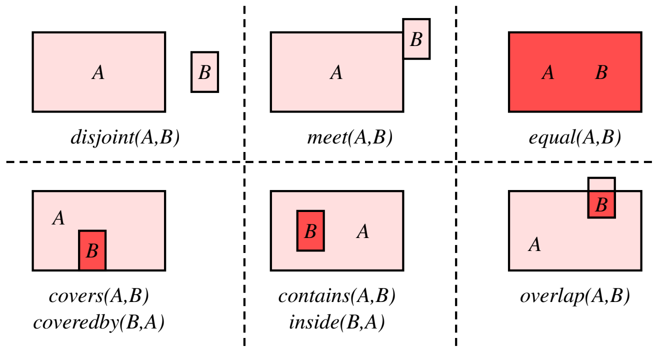


Figure 3.56: The eight topological relations of RCC-8 (non-overlapping parts of rectangles A and B are coloured in pink, while overlapping parts are coloured in red)

3.7.206 Rectangle clique partition [1 CONS]

- NVECTOR.

Denotes that, by reduction to the *rectangle clique partition* problem, deciding whether a constraint has a solution or not was shown to be NP-hard. The *rectangle clique partition* problem can be described as follows: given a rectangle graph, can its set of vertices be partitioned into k subsets of vertices such that all corresponding induced subgraphs correspond to cliques? A *rectangle graph* is a graph that can be associated with a set of fixed rectangles whose sides are parallel to the axes of the placement space: to each rectangle corresponds a vertex of the rectangle graph, while to each pair of intersecting rectangles corresponds an edge.

3.7.207 Regret based heuristics [4 CONS]

- ELEM,
- ELEMENT,
- GLOBAL_CARDINALITY_WITH_COSTS,
- SUM_CTR.

Assume you have a discrete optimisation problem where the sum of some cost variables should be minimised, and where the cost variables typically have holes in their domains. In this context a regret based heuristic first selects among the not yet fixed cost variables, the one with the largest difference between its second smallest value and its smallest value. The idea is to consider first a variable that would cause the biggest increase in cost if it could not be assigned its minimum value.

3.7.208 Regret based heuristics in matrix problems [2 CONS]

- GLOBAL_CARDINALITY_WITH_COSTS,
- SUM_CTR.

Assume you have a discrete optimisation problem involving a matrix $\mathcal{M}$ of decision variables such that there is a cost variable attached to each row of $\mathcal{M}$. Moreover assume that the cost associated with each row corresponds to a sum of elementary costs connected with each decision variable of the same row (e.g., we have a SUM_CTR or a GLOBAL_CARDINALITY_WITH_COSTS constraint on each row of $\mathcal{M}$). Now, suppose we want to use a heuristic for fixing the decision variables of matrix $\mathcal{M}$ row by row. In this context a question is which row to select first. Since the cost variable c_r associated with a row r corresponds to a sum of elementary costs, it is very unlikely that the cost variable c_r has a hole in its domain. Consequently, we cannot any more use a conventional regret based heuristic which relies on the fact that we have holes in the domains of the cost variables. We still want to use the idea of finding the variable that would potentially cause the biggest increase in cost in the worst case, i.e. if it would have to be assigned to its maximum value. For this purpose we consider the variable for which the difference between its largest value and its smallest value is maximal. In our context we select the row r for which the corresponding cost variable maximises such difference. First we enumerate in increasing value order on the cost variable associated with row r. Second we fix all decision variables of row r using, for example, the heuristic described in labelling by increasing cost. Using such cost based heuristics has both some advantage and some drawback:

- The big potential advantage is that, if we can find a first solution at all, then this solution should have a rather small overall cost.
- The potential drawback is that, depending on how strong the row constraints propagate from the maximum total cost associated with a row back to the decision variables of the row, it may be very difficult to find a feasible solution (since assigning the cost variable of a row to its minimum value potentially creates an infeasible problem for which we need to develop a large search tree).

3.7.209 Reified automaton constraint [60 CONS]

- AND,
- ARITH,
- ARITH_OR,
- BETWEEN_MIN_MAX,
- CLAUSE_AND,
- CLAUSE_OR,
- COND_LEX_COST,
- CONSECUTIVE_GROUPS_OF_ONES,
- DECREASING,
- DOMAIN_CONSTRAINT,
- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_MATRIX,
- ELEMENT_SPARSE,
- ELEMENTN,
- EQUIVALENT,
- GLOBAL_CONTIGUITY,
- IMPLY,
- IN,
- IN_INTERVAL,
- IN_SAME_PARTITION,
- INCREASING,
- INCREASING_GLOBAL_CARDINALITY,
- INCREASING_NVALUE,
- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- LEX_BETWEEN,
- LEX_DIFFERENT,
- LEX_EQUAL,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- MAXIMUM,
- MINIMUM,
- MINIMUM_EXCEPT_0,
- MINIMUM_GREATER_THAN,
- NAND,
- NEXT_ELEMENT,
- NO_PEAK,
- NO_VALLEY,
- NOR,
- NOT_ALL_EQUAL,
- NOT_IN,
- OPEN_MAXIMUM,
- OPEN_MINIMUM,
- OR,
- PATTERN,
- SEQUENCE_FOLDING,
- STAGE_ELEMENT,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION,
- STRICTLY_DECREASING,
- STRICTLY_INCREASING,
- TWO_ORTH_ARE_IN_CONTACT,
- TWO_ORTH_DO_NOT_OVERLAP,
- XOR.

A constraint $\mathcal{C}(V_1, V_2, \ldots, V_n)$ for which the reified version can be mechanically constructed from the finite deterministic automaton $\mathcal{A}^{\mathcal{C}}$ that only accepts the set of solutions to constraint $\mathcal{C}$. This is done by deriving from $\mathcal{A}^{\mathcal{C}}$ a so called *reified automaton* $\mathcal{A}^{\mathcal{C}}_{\neg\mathcal{C}}$ by:

- First, adding a 0-1 variable B in front of the sequence of variables $V_1, V_2, \ldots, V_n$. This new sequence of variables will be passed to the reified automaton $\mathcal{A}^{\mathcal{C}}_{\neg\mathcal{C}}$.

- Second, constructing from $\mathcal{A}^{\mathcal{C}}$ the automaton $\mathcal{A}_{\neg\mathcal{C}}$ that only recognises non-solutions to constraint $\mathcal{C}$.
- Third, building from the two automata $\mathcal{A}^{\mathcal{C}}$ and $\mathcal{A}_{\neg\mathcal{C}}$ the automaton $\mathcal{A}^{\mathcal{C}}_{\neg\mathcal{C}}$. This is done by:
  1. Creating the initial state s of $\mathcal{A}^{\mathcal{C}}_{\neg\mathcal{C}}$.
  2. Adding a transition labelled by value 1 from s to the initial state of $\mathcal{A}^{\mathcal{C}}$.
  3. Adding a transition labelled by value 0 from s to the initial state of $\mathcal{A}_{\neg\mathcal{C}}$.

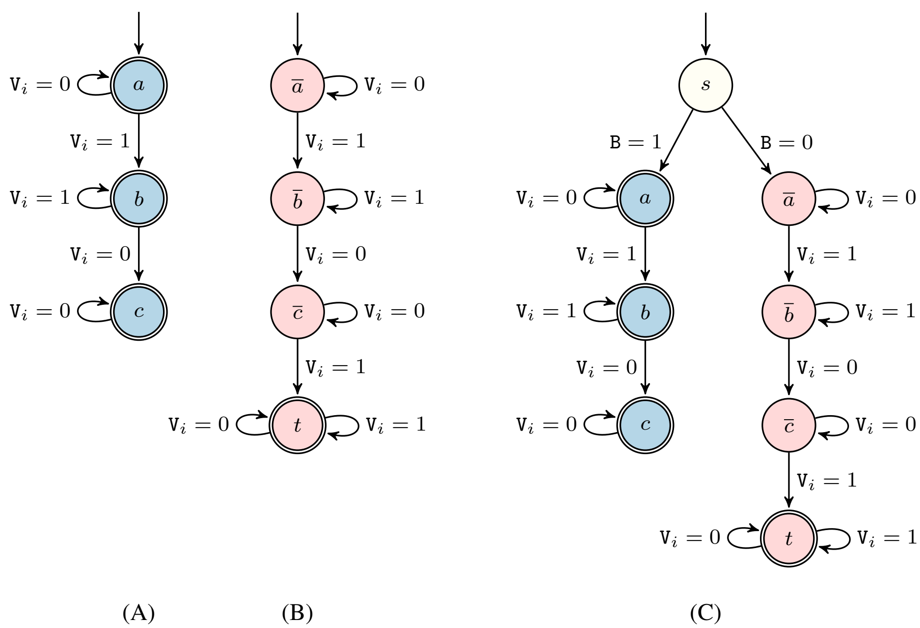


Figure 3.57: (A) The automaton for recognising the solutions to the GLOBAL_CONTIGUITY constraint; (B) the automaton for recognising the non-solutions to the GLOBAL_CONTIGUITY constraint; (C) the automaton for the reified GLOBAL_CONTIGUITY constraint; within an automaton an initial state is indicated by an arc coming from no state and an accepting state is denoted graphically by a double circle.

Figure 3.57 illustrates the construction of a reified automaton in the context of the GLOBAL_CONTIGUITY constraint. Part (A) recalls the automaton that only recognises the solutions to the GLOBAL_CONTIGUITY constraint. Assuming the same alphabet $\{0, 1\}$, Part (B) provides the automaton that only recognises the non-solutions to the GLOBAL_CONTIGUITY constraint. Finally, Part (C) depicts the reified automaton constructed from the two automata given in parts (A) and (B).

3.7.210 Reified constraint [1 CONS]

- IN_INTERVAL_REIFIED (reified version of IN_INTERVAL).

The reified version CR of a given constraint C, where CR has as arguments all arguments of C plus one extra 0-1 variable. This 0-1 variable is set to 1 when constraint C holds, and 0 otherwise. Note that constraint CR inherits from all restrictions of constraint C (i.e., incorrect parameters for constraint C are also incorrect for constraint CR). Within the context of linear programming the extra 0-1 variable is often called an *indicator variable*.

It was shown in [36] how to reify a global constraint by reformulating it as a conjunction of pure functional dependency constraints together with a constraint that can be easily reified (e.g., an automaton with or without counter, or a Boolean combination of linear arithmetic equalities and inequalities and 0-1 variables).

3.7.211 Relation [3 CONS]

- IN_RELATION,
- SYMMETRIC_CARDINALITY,
- SYMMETRIC_GCC.

A constraint that allows for representing the access to an element of a *relation* or to model a *relation*. A *relation* is a subset of the product of several finite sets.

3.7.212 Relaxation [20 CONS]

- ALLDIFFERENT_EXCEPT_0,
- DIFFN,
- GEOST,
- RELAXED_SLIDING_SUM,
- SOFT_ALLDIFFERENT_CTR,
- SOFT_ALLDIFFERENT_VAR,
- SOFT_ALL_EQUAL_MAX_VAR,
- SOFT_ALL_EQUAL_MIN_CTR,

- SOFT_ALL_EQUAL_MIN_VAR,
- SOFT_CUMULATIVE,
- SOFT_SAME_INTERVAL_VAR,
- SOFT_SAME_MODULO_VAR,
- SOFT_SAME_PARTITION_VAR,
- SOFT_SAME_VAR,
- SOFT_USED_BY_INTERVAL_VAR,
- SOFT_USED_BY_MODULO_VAR,
- SOFT_USED_BY_PARTITION_VAR,
- SOFT_USED_BY_VAR,
- SUM_OF_WEIGHTS_OF_DISTINCT_VALUES,
- WEIGHTED_PARTIAL_ALLDIFF.

Denotes that a constraint allows for specifying a partial degree of satisfaction. For the constraints DIFFN and GEOST see the keyword Relaxation dimension.

3.7.213 Relaxation dimension [2 CONS]

- DIFFN,
- GEOST.

A constraint that allows the modelling of constraint relaxation in the context of placement problems. This is achieved by adding an extra dimension to the placement space where objects that are really considered are in the foreground, while objects that are discarded are rejected into the background. As a concrete example, consider a slight modification of the data of the task assignment and scheduling problem that is described at the keyword entry assigning and scheduling tasks that run in parallel. In this problem the four nurses are all not available during the time periods $[0, 0]$, $[7, 7]$, $[12, 12]$ and $[22, 22]$. We now rather consider the following unavailability periods $[0, 0]$, $[8, 8]$, $[12, 12]$ and $[22, 22]$. Under this new hypothesis we cannot anymore schedule all the five surgery tasks t_1, t_2, t_3, t_4 and t_5, i.e., we get a *no solution* answer if we use the model described in assigning and scheduling tasks that run in parallel. In this model we are using a two-dimensional GEOST constraint, where the first and second dimensions respectively correspond to the time and resource axes. Now, in order to permit relaxation, we introduce a third dimension, a *relaxation dimension*. The idea is to map each task to a parallelepiped for which the size in the relaxation dimension is equal to one. In addition, the coordinate of a parallelepiped in the relaxation dimension is a variable taking its value in the interval $[1, n]$, where n represents the number of operations to schedule (i.e., for each surgery task t_i $(1 \leq i \leq n = 5)$ we create a coordinate variable r_i where r stands for *relaxation*. Then, all parallelepipeds for which the coordinate in the relaxation dimension is set to 1 correspond to surgery tasks that are effectively scheduled, while all other parallelepipeds represent surgery tasks that are discarded. On the one hand, this model allows the direct expression of relaxation right from the beginning without introducing any extra soft constraint and without dynamically adding any constraint during search. On the other hand, a disadvantage is that the model does not directly consider an optimisation criterion like, for example, the maximum number of tasks effectively scheduled, or the sum of the durations of the tasks effectively done; this can be modelled using extra constraints but this does not provide sharp bounds on the optimisation criterion. Nevertheless, this gives a compact model, especially in the context where additional constraints complicate the computation of a sharp bound. Going back to the example described at the keyword entry assigning and scheduling tasks that run in parallel, we get the following three-dimensional GEOST constraint:

GEOST(3,←- - - - *number of dimensions of the placement space: time, resources and relaxation*

objects: *subtasks of tasks $t_1, t_2, \ldots, t_5$*

⟨ oid − 1	sid − 2	x − $\langle o_1, a_1, r_1\rangle$,	oid − 2	sid − 2	x − $\langle o_1, s_1, r_1\rangle$,
oid − 3	sid − 2	x − $\langle o_1, n_{11}, r_1\rangle$,	oid − 4	sid − 2	x − $\langle o_1, n_{12}, r_1\rangle$,
oid − 5	sid − 4	x − $\langle o_2, a_2, r_2\rangle$,	oid − 6	sid − 4	x − $\langle o_2, s_2, r_2\rangle$,
oid − 7	sid − 4	x − $\langle o_2, n_2, r_2\rangle$,	oid − 8	sid − 3	x − $\langle o_3, a_3, r_3\rangle$,
oid − 9	sid − 3	x − $\langle o_3, s_{31}, r_3\rangle$,	oid − 10	sid − 3	x − $\langle o_3, s_{32}, r_3\rangle$,
oid − 11	sid − 3	x − $\langle o_3, n_{31}, r_3\rangle$,	oid − 12	sid − 3	x − $\langle o_3, n_{32}, r_3\rangle$,
oid − 13	sid − 2	x − $\langle o_4, a_4, r_4\rangle$,	oid − 14	sid − 2	x − $\langle o_4, s_4, r_4\rangle$,
oid − 15	sid − 2	x − $\langle o_4, n_{41}, r_4\rangle$,	oid − 16	sid − 2	x − $\langle o_4, n_{42}, r_4\rangle$,
oid − 17	sid − 2	x − $\langle o_4, n_{43}, r_4\rangle$,	oid − 18	sid − 6	x − $\langle o_5, a_5, r_5\rangle$,
oid − 19	sid − 6	x − $\langle o_5, s_5, r_5\rangle$,	oid − 20	sid − 6	x − $\langle o_5, n_5, r_5\rangle$,

unavailability periods

oid − 21	sid − 2	x − $\langle 0,1,1\rangle$,	oid − 22	sid − 2	x − $\langle 5,1,1\rangle$,
oid − 23	sid − 5	x − $\langle 12,1,1\rangle$,	oid − 24	sid − 3	x − $\langle 0,2,1\rangle$,
oid − 25	sid − 1	x − $\langle 6,2,1\rangle$,	oid − 26	sid − 1	x − $\langle 15,2,1\rangle$,
oid − 27	sid − 1	x − $\langle 22,2,1\rangle$,	oid − 28	sid − 2	x − $\langle 0,3,1\rangle$,
oid − 29	sid − 2	x − $\langle 8,3,1\rangle$,	oid − 30	sid − 2	x − $\langle 13,3,1\rangle$,
oid − 31	sid − 1	x − $\langle 5,4,1\rangle$,	oid − 32	sid − 2	x − $\langle 20,4,1\rangle$,
oid − 33	sid − 1	x − $\langle 0,5,1\rangle$,	oid − 34	sid − 1	x − $\langle 7,5,1\rangle$,
oid − 35	sid − 1	x − $\langle 12,5,1\rangle$,	oid − 36	sid − 1	x − $\langle 22,5,1\rangle$,
oid − 37	sid − 1	x − $\langle 0,6,1\rangle$,	oid − 38	sid − 1	x − $\langle 7,6,1\rangle$,
oid − 39	sid − 1	x − $\langle 12,6,1\rangle$,	oid − 40	sid − 1	x − $\langle 22,6,1\rangle$,
oid − 41	sid − 1	x − $\langle 0,7,1\rangle$,	oid − 42	sid − 1	x − $\langle 7,7,1\rangle$,
oid − 43	sid − 1	x − $\langle 12,7,1\rangle$,	oid − 44	sid − 1	x − $\langle 22,7,1\rangle$,
oid − 45	sid − 1	x − $\langle 0,8,1\rangle$,	oid − 46	sid − 1	x − $\langle 7,8,1\rangle$,
oid − 47	sid − 1	x − $\langle 12,8,1\rangle$,	oid − 48	sid − 1	x − $\langle 22,8,1\rangle\rangle$,

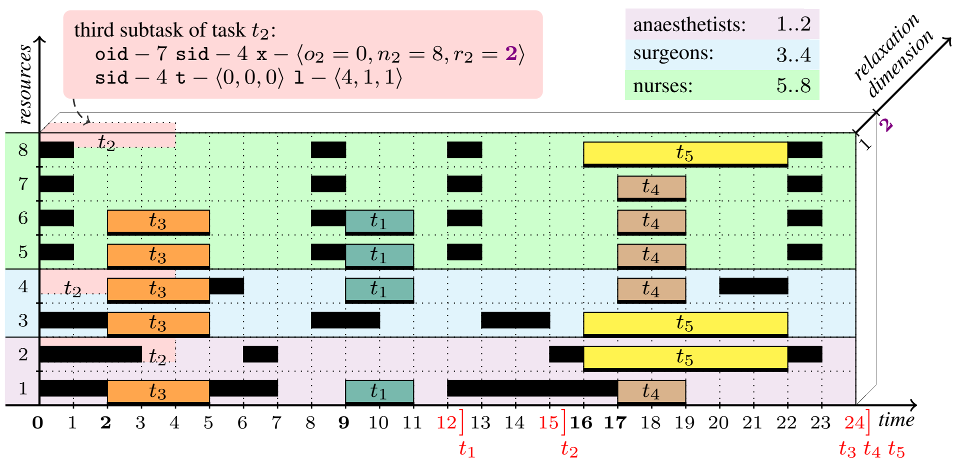
shapes

⟨sid − 1	t − $\langle 0,0,0\rangle$	l − $\langle 1,1,1\rangle$,	sid − 2	t − $\langle 0,0,0\rangle$	l − $\langle 2,1,1\rangle$,
sid − 3	t − $\langle 0,0,0\rangle$	l − $\langle 3,1,1\rangle$,	sid − 4	t − $\langle 0,0,0\rangle$	l − $\langle 4,1,1\rangle$,
sid − 5	t − $\langle 0,0,0\rangle$	l − $\langle 5,1,1\rangle$,	sid − 6	t − $\langle 0,0,0\rangle$	l − $\langle 6,1,1\rangle\rangle$).

Figure 3.58 depicts a solution to the problem corresponding to the assignment

tasks	origin	relaxation	anaesthetist	surgeon	nurse
t_1	$o_1 = 9$	$r_1 = 1$	$a_1 = 1$	$s_1 = 4$	$n_{11} = 5, n_{12} = 6$
t_2	$o_2 = 0$	$r_2 = 2$	$a_2 = 2$	$s_2 = 4$	$n_2 = 8$
t_3	$o_3 = 2$	$r_3 = 1$	$a_3 = 1$	$s_{31} = 3, s_{32} = 4$	$n_{31} = 5, n_{32} = 6$
t_4	$o_4 = 17$	$r_4 = 1$	$a_4 = 1$	$s_4 = 4$	$n_{41} = 5, n_{42} = 6, n_{43} = 7$
t_5	$o_5 = 16$	$r_5 = 1$	$a_5 = 2$	$s_5 = 3$	$n_5 = 8$

During search, the relaxation variables r_1, r_2, r_3, r_4, r_5 are first set to value one (i.e., the corresponding operations are scheduled) and then, upon backtracking, assigned to any value greater than one (i.e., there is no backtrack on the values that are greater than one since we just want to reject an operation into the background).



- operations effectively scheduled (*in the foreground of the relaxation dimension*): t_1, t_3, t_4, t_5
- operations ignored (*into the background of the relaxation dimension, i.e.* $r_i > 1$): t_2

Figure 3.58: A partial solution to the surgery scheduling problem that maximises the number of operations actually performed where only operation t_2 is not scheduled

3.7.214 Resource constraint [19 CONS]

- BIN_PACKING,
- BIN_PACKING_CAPA,
- COLOURED_CUMULATIVE,
- COLOURED_CUMULATIVES,
- CUMULATIVE,
- CUMULATIVE_CONVEX,
- CUMULATIVE_PRODUCT,
- CUMULATIVE_WITH_LEVEL_OF_PRIORITY,
- CUMULATIVES,
- CYCLE_RESOURCE,
- DISJ,
- DISJUNCTIVE,
- DISJUNCTIVE_OR_SAME_END,
- DISJUNCTIVE_OR_SAME_START,
- INTERVAL_AND_COUNT,
- INTERVAL_AND_SUM,
- SOFT_CUMULATIVE,
- TRACK,
- TREE_RESOURCE.

A constraint restricting the utilisation of a resource. The utilisation of a resource is computed from all items that are assigned to that resource.

3.7.215 Reverse of a constraint [31 CONS]

- AMONG,
- CHANGE_CONTINUITY with CTR $\in \{=, \neq\}$,
- CHANGE_CONTINUITY with CTR $\in \{<\}$ (CHANGE_CONTINUITY with CTR $\in \{>\}$),
- CHANGE_CONTINUITY with CTR $\in \{\leq\}$ (CHANGE_CONTINUITY with CTR $\in \{\geq\}$),
- DEEPEST_VALLEY,
- EXACTLY,
- FULL_GROUP,
- GROUP,
- GROUP_SKIP_ISOLATED_ITEM,
- HIGHEST_PEAK,
- INFLEXION,
- LENGTH_FIRST_SEQUENCE (LENGTH_LAST_SEQUENCE),
- LONGEST_CHANGE with CTR $\in \{=, \neq\}$,
- LONGEST_DECREASING_SEQUENCE (LONGEST_INCREASING_SEQUENCE),
- MAXIMUM,
- MAX_DECREASING_SLOPE (MAX_INCREASING_SLOPE),
- MIN_DECREASING_SLOPE (MIN_INCREASING_SLOPE),
- MIN_SURF_PEAK,
- MIN_WIDTH_PEAK,
- MIN_WIDTH_PLATEAU,
- MIN_WIDTH_VALLEY,
- MINIMUM,
- PEAK,
- SMOOTH,
- VALLEY.

A constraint which has a reverse constraint, where the reverse is defined in the following way. Consider two constraints $\mathrm{ctr}(col, r_1, \ldots, r_n)$ and $\mathrm{ctr}'(col, r_1, \ldots, r_n)$ for which, in both cases, the argument col is a collection of items that functionally determines all the other arguments $r_1, \ldots, r_n$.

The constraint ctr' is the *reverse constraint* of constraint ctr if, for any collection of items col, we have the equivalence $\mathrm{ctr}(col, r_1, \ldots, r_n) \Leftrightarrow \mathrm{ctr}'(col^{rev}, r_1, \ldots, r_n)$, where col^{rev} denotes the collection col where the items of the collection are reversed. When constraints ctr and ctr' are identical we say that constraint ctr is its own reverse.

The previous enumeration provides the list of reversible constraints where, for each reversible constraint, we give its reverse only when it is different from the original constraint.

Note that if a constraint can be represented by a counter deterministic automaton with a single counter that is only incremented and for which all states are accepting, then by computing the reverse automaton, the corresponding reverse constraint can be mechanically obtained. However note that the reverse automaton may be non-deterministic and may contain ϵ transitions [297]. Figure 3.59 gives an automaton counting the number of occurrences of words 001 in a sequence and its reverse automaton. Figure 3.60 provides an automaton with one counter and its reverse automaton that has a different number of states.

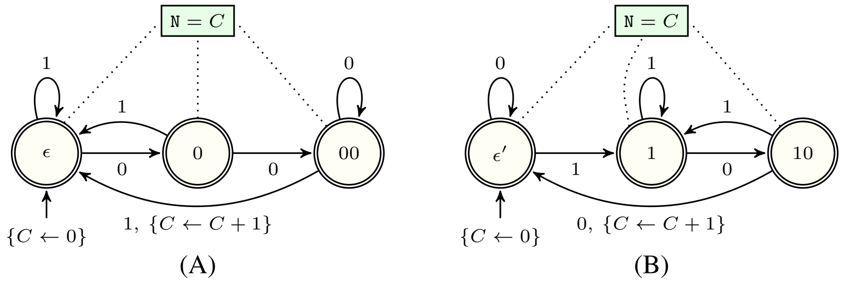


Figure 3.59: (A) Counter automaton returning the number of occurrences N of word 001 in a sequence, and (B) its reverse counter automaton (returning the number of occurrences N of word 100 in a sequence)

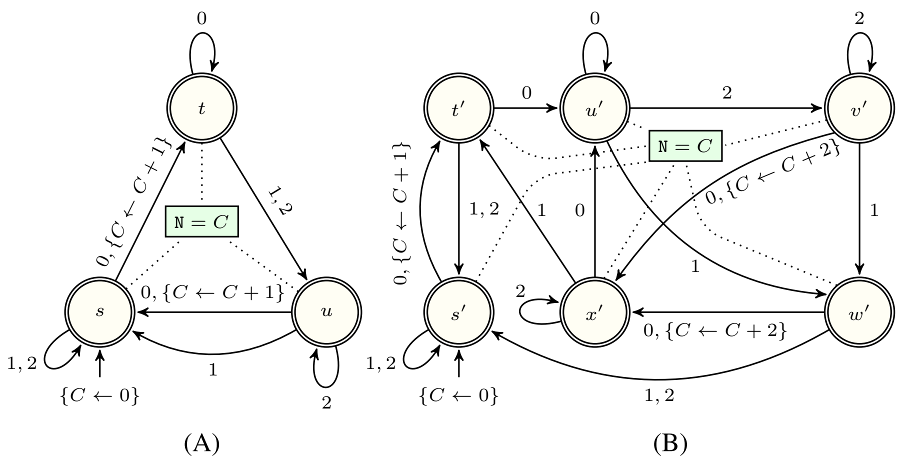


Figure 3.60: (A) Counter automaton, and (B) its reverse counter automaton which has a different number of states (an accepting state is denoted graphically by a double circle)

3.7.216 Run of a permutation **[1 CONS]**

- CHANGE_CONTINUITY.

A constraint that can be used for putting a restriction on the size of the longest *run* of a permutation. A *run* is a maximal increasing contiguous subsequence in a permutation.

3.7.217 SAT [3 CONS]

- ALLDIFFERENT,
- AMONG,
- DIFFN.

A constraint for which a reference provides a reformulation in SAT. Encoding for the ALLDIFFERENT and the AMONG constraints were respectively provided in [201] and in [16]. Based on Fekete *et al.* model of the multi-dimensional orthogonal packing problem [175], an encoding for the DIFFN constraint when all the sizes of all the orthotopes are fixed was described in [209].

3.7.218 Scalar product [1 CONS]

- GLOBAL_CARDINALITY_WITH_COSTS.

A constraint that can be used for modelling a scalar product constraint.

3.7.219 Sequence [52 CONS]

- ALL_EQUAL_PEAK,
- ALL_EQUAL_PEAK_MAX,
- ALL_EQUAL_VALLEY,
- ALL_EQUAL_VALLEY_MIN,
- AMONG_SEQ,
- ARITH_SLIDING,
- BIG_PEAK,
- BIG_VALLEY,
- CHANGE_CONTINUITY,
- CYCLE_CARD_ON_PATH,
- DECREASING_PEAK,
- DECREASING_VALLEY,
- DEEPEST_VALLEY,
- FULL_GROUP,
- GLOBAL_CONTIGUITY,
- GROUP,

- GROUP_SKIP_ISOLATED_ITEM,
- HIGHEST_PEAK,
- INCREASING_PEAK,
- INCREASING_VALLEY,
- INFLEXION,
- LONGEST_DECREASING_SEQUENCE,
- LONGEST_INCREASING_SEQUENCE,
- NO_PEAK,
- NO_VALLEY,
- NVISIBLE_FROM_END,
- NVISIBLE_FROM_START,
- MAX_DECREASING_SLOPE,
- MAX_INCREASING_SLOPE,
- MIN_DECREASING_SLOPE,
- MIN_DIST_BETWEEN_INFLEXION,
- MIN_INCREASING_SLOPE,
- MIN_SIZE_FULL_ZERO_STRETCH,
- MIN_SURF_PEAK,
- MIN_WIDTH_PEAK,
- MIN_WIDTH_PLATEAU,
- MIN_WIDTH_VALLEY,
- MULTI_GLOBAL_CONTIGUITY,
- PEAK,
- PERIOD,
- PERIOD_EXCEPT_0,
- PERIOD_VECTORS,
- RELAXED_SLIDING_SUM,
- SEQUENCE_FOLDING,
- SIZE_MAX_SEQ_ALLDIFFERENT,
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT,
- SLIDING_CARD_SKIP0,
- SLIDING_DISTRIBUTION,
- SLIDING_SUM,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION,
- VALLEY.

Constrains consecutive variables (possibly not all) of a given collection of domain variables or consecutive vertices of a simple path or a simple circuit. Also a constraint restricting a variable (when fixed to 0 the variable may be omitted) according to consecutive variables of a given collection of domain variables.

3.7.220 Sequence dependent set-up [5 CONS]

- DIFFN,
- DISJUNCTIVE,
- ELEM,
- ELEMENT,
- TEMPORAL_PATH.

Denotes that a constraint can be used for modelling sequence dependent set-up between pairs of tasks. Given,

- a collection of n tasks $\mathcal{T}$, where each task $t_i \in \mathcal{T}$ $(1 \leq i \leq n)$ has an *origin* o_i, a *duration* d_i, an *end* e_i $(o_i + d_i = e_i)$ and a *machine* m_i to which it will be assigned,

- and a n by n matrix $\mathcal{M}$ of positive integers δ_{ij} $i, j \in [1, n]$ where row_i denotes the i^{th} row of matrix $\mathcal{M}$,

we want to express that δ_{ij} enforces a *minimum distance* between the completion of task $t_i \in \mathcal{T}$ and the start of task $t_j \in \mathcal{T}$ $(i \neq j)$ under the hypotheses that (a) both tasks are assigned the same machine (i.e., $m_i = m_j$) and that (b) task t_j immediately follows task t_i (i.e., there is no task $t_k \in \mathcal{T}$ $(k \notin \{i, j\})$ such that $m_k = m_i \wedge e_i \leq o_k \wedge e_k \leq o_j$). In addition, tasks assigned to the same machine should not overlap (i.e., $\forall i \in [1, n], \forall j \neq i \in [1, n]$ such that $m_i = m_j$ we have $e_i \leq o_j \vee e_j \leq o_i$). We show how to model the previous sequence dependent set-up constraint under the hypothesis that we have a single machine. Without loss of generality we assume that $\delta_{ii} = 0$ for all $i \in [1, n]$.

In a first phase we create for each task $t_i \in \mathcal{T}$ $(1 \leq i \leq n)$ three additional variables s_i, g_i and c_i that respectively correspond to:

- The *successor variable* $s_i \in [1, n]$ allows to get the immediate successor of task t_i. On the one hand, the assignment $s_i = i$ denotes that task t_i has no immediate successor (i.e., task t_i is the last task running on machine m_i), on the other hand, $s_i = j$ $(j \neq i)$ denotes that task t_j is the immediate successor of task t_i.
- The *gap variable* g_i represents the size of the gap between the end of task t_i and the start of its immediate successor (the gap is equal to 0 when task t_i has no immediate successor).
- The *extended completion* variable c_i represents the sum of the end of task t_i and the corresponding gap variable g_i (i.e., $c_i = e_i + g_i$).

In a second phase we post for each task $t_i \in \mathcal{T}$ $(1 \leq i \leq n)$ the following constraints:

- An ELEMENT(s_i, row_i, g_i) constraint to make the link between the successor variable s_i and the gap variable g_i.
- A SUM_CTR$(\langle e_i, g_i \rangle, =, c_i)$ constraint.

Finally in a third phase we create a collection of nodes `NODES` where each item corresponds to a task $t_i \in \mathcal{T}$ $(1 \leq i \leq n)$ and has an `index` attribute set to i, a `succ` attribute set to s_i, a `start` attribute set to o_i and an `end` attribute set to c_i. We post a TEMPORAL_PATH$(1, \texttt{NODES})$ constraint for linking the successor variables, the start variables and the extended completion variables associated with the different tasks. The first argument of the TEMPORAL_PATH constraint forces a single path corresponding to the succession of the different tasks on the unique machine.

3.7.221 Sequencing with release times and deadlines [5 CONS]

- CUMULATIVE,
- CUMULATIVES,
- DIFFN,
- DISJ,
- DISJUNCTIVE.

Denotes that, by reduction to *sequencing with release times and deadlines*, deciding whether a constraint has a solution or not was shown to be NP-hard. The *sequencing with release times and deadlines* problem can be described as follows: given a set of non-overlapping tasks and, for each task a length, a release time and a deadline the question is to find a schedule that satisfies all release time constraints and meets all the deadlines.

3.7.222 Set channel [2 CONS]

- INVERSE_SET,
- LINK_SET_TO_BOOLEANS.

A channelling constraint involving one or several set variables.

3.7.223 Set packing [1 CONS]

- K_ALLDIFFERENT.

Denotes that, by reduction to *set packing*, deciding whether a constraint has a solution or not was shown to be NP-hard. The *set packing* problem can be described as follows: given a collection C of n finite sets, and a positive integer $m \leq n$, does C contain m disjoint sets?

3.7.224 Shikaku [2 CONS]

- DIFFN,
- GEOST.

Denotes that a constraint can be used for solving the *Shikaku* puzzle. Given a rectangular grid, where exactly n cells contain an integer value, the problem is to tile that grid by n rectangles in such a way that the surface of each rectangle is equal to the single integer it contains.

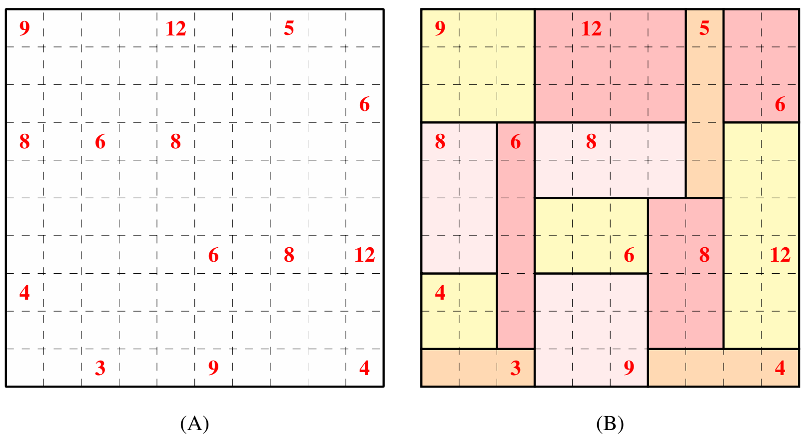


Figure 3.61: (A) An example of a Shikaku puzzle and (B) its corresponding unique solution

Parts (A) and (B) of Figure 3.61 respectively show a small instance of such a puzzle and its corresponding unique solution taken from the Nikoli website `https://member.nikoli.com/index.html`.

3.7.225 Scheduling constraint [20 CONS]

- ALL_MIN_DIST,
- CALENDAR,
- COLOURED_CUMULATIVE,
- COLOURED_CUMULATIVES,
- CUMULATIVE,
- CUMULATIVE_CONVEX,
- CUMULATIVE_PRODUCT,
- CUMULATIVE_WITH_LEVEL_OF_PRIORITY,

- CUMULATIVES,
- DISJOINT_TASKS,
- DISJ,
- DISJUNCTIVE,
- DISJUNCTIVE_OR_SAME_END,
- DISJUNCTIVE_OR_SAME_START,
- MULTI_INTER_DISTANCE,
- PERIOD,
- PERIOD_EXCEPT_0,
- SHIFT,
- SOFT_CUMULATIVE,
- TASKS_INTERSECTION.

A constraint useful for the area of *scheduling*. *Scheduling* is concerned with the allocation or assignment of resources (e.g., manpower, machines, money), over time, to a set of tasks.

3.7.226 Scheduling with machine choice, calendars and preemption [4 CONS]

- CALENDAR,
- CUMULATIVES,
- DIFFN,
- GEOST.

A set of constraints that can be used for modelling a scheduling problem where:

- We have tasks that have both to be assigned to machine and time.
- Each task has a fixed duration.
- Machines can run at most one task at a given instant.
- Each machine has its own fixed unavailability periods (i.e., a calendar of unavailability periods).
- An unavailability period that allows (respectively forbids) a task to be interrupted and resumed just after is called *crossable* (respectively *non-crossable*). A task that can be (respectively cannot be) interrupted by a crossable unavailability period is called *resumable* (respectively *non-resumable*).
- We have a precedence constraint between specific pairs of tasks. Each precedence forces that a given task ends before the start of another given task.

This model illustrates the use of two time coordinates systems:

- The first coordinate system, so called the *virtual coordinate system*, does not consider at all the crossable unavailability periods associated with the different machines. Since resumable tasks can be preempted by machine crossable unavailability, all resource scheduling constraints (i.e., DIFFN, GEOST) are expressed within this first coordinate system. This stands from the fact that resource scheduling constraints like DIFFN or GEOST do not support preemption.

- The second coordinate system, so called the *real coordinate system*, considers all timepoints whether they correspond or not to crossable unavailability periods. All temporal constraints (i.e., precedence constraints represented by LEQ constraints in this model) are expressed with respect to this second coordinate system.

Consequently, each task has a start and an end that are expressed within the virtual coordinate system as well as within the real coordinate system.

- Each task, whether it is resumable or not, is passed to the resource scheduling constraints as well as to the precedence constraints. In addition, we represent each non-crossable unavailability period as a fixed task that is also passed to the resource scheduling constraints.

- The CALENDAR constraint ensures the link between variables (i.e., the start and the end of the tasks no matter whether they are resumable or not) expressed in these two coordinate systems with respect to the crossable unavailability periods.

We now provide the corresponding detailed model. Given:

1. A set of machines $\mathcal{M} = \{m_1, m_2, \ldots, m_p\}$, where each machine has a list of fixed unavailability periods. An unavailability u_i is defined by the following attributes:

   (a) The *crossable flag* c_i tells whether unavailability u_i is crossable ($c_i = 1$) or not ($c_i = 0$).

   (b) The *machine* r_i indicates the machine (i.e., a value in $[1, p]$) to which unavailability u_i corresponds (i.e., since different machines may have different unavailability periods).

   (c) The *start* s_i of the unavailability u_i which indicates the first unavailable timepoint of the unavailability.

   (d) The *end* e_i of the unavailability u_i which gives the last unavailable timepoint of the unavailability.

2. A set of tasks $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$, where each task t_i (with $i \in [1, n]$) has the following attributes which are all domain variables except the resumable flag and the virtual duration:

   (a) The *resumable flag* r_i tells whether task t_i is resumable ($r_i = 1$) or not ($r_i = 0$).

(b) The *machine* m_i indicates the machine (i.e., a value in $[1, p]$) to which task t_i will be assigned.

(c) The *virtual start* vs_i gives the start of task t_i in the virtual coordinate system.

(d) The *virtual duration* vd_i corresponds to the duration of task t_i without counting the eventual unavailability periods crossed by task t_i.

(e) The *virtual end* ve_i provides the end of task t_i in the virtual coordinate system. We have that $vs_i + vd_i = ve_i$.

(f) The *real start* rs_i gives the start of task t_i in the real coordinate system.

(g) The *real duration* rd_i corresponds to the duration of task t_i including the eventual unavailability periods crossed by task t_i. When task t_i is non-resumable (i.e., $r_i = 0$) its real duration is equal to its virtual duration (i.e., $rd_i = vd_i$).

(h) The *real end* re_i indicates the end of task t_i in the real coordinate system. We have that $rs_i + rd_i = re_i$.

The link between the virtual starts (respectively virtual ends) and the real starts (respectively real ends) of the different tasks of $\mathcal{T}$ is ensured by a CALENDAR($\texttt{INSTANTS}, \texttt{MACHINES}$) constraint. More precisely, for each task t_i (with $i \in [1, n]$), no matter whether it is resumable or not, we create the following items for the collection INSTANTS:

$$\begin{array}{l}\left\langle\ \texttt{machine} - m_i \quad \texttt{virtual} - vs_i \quad \texttt{ireal} - rs_i \quad \texttt{flagend} - 0\ \right\rangle,\\ \left\langle\ \texttt{machine} - m_i \quad \texttt{virtual} - ve_i \quad \texttt{ireal} - re_i \quad \texttt{flagend} - 1\ \right\rangle.\end{array}$$

The first item links the virtual and the real start of task t_i, while the second item relates the virtual and real ends. For each machine m_i (with $i \in [1, p]$) and its corresponding list of crossable unavailability periods, denoted $crossable\_unavailability_i$, we create the following item of the collection MACHINES:

$$\left\langle\ \texttt{id} - i \quad \texttt{cal} - crossable\_unavailability_i\ \right\rangle.$$

To express the resource constraint, i.e., the fact that two tasks assigned to the same machine should not overlap in time, we use a GEOST($2, \texttt{OBJECTS}, \texttt{SBOXES}$) constraint. For each task t_i (with $i \in [1, n]$) we create one item for the OBJECTS collection as well as one item for the SBOXES collection:

$$\begin{array}{l}\left\langle\ \texttt{oid} - i \quad \texttt{sid} - i \quad \texttt{x} - \langle m_i, vs_i\rangle\ \right\rangle,\\ \left\langle\ \texttt{sid} - i \quad \texttt{t} - \langle 0, 0\rangle \quad \texttt{l} - \langle 1, vd_i\rangle\ \right\rangle.\end{array}$$

The first item corresponds to an object with i as unique identifier, with a rectangular shape identifier i and with m_i, vs_i as the coordinates of its lower left corner. The second item corresponds to a rectangular shape with i as unique identifier, $\langle 0, 0\rangle$ as shift offset with respect to its lower left corner, and $\langle 1, vd_i\rangle$ as the sizes of the rectangular shape.

Similarly, to express that each task does not overlap a non-crossable unavailability period, we create for each non-crossable unavailability period i one item for the

OBJECTS collection as well as one item for the SBOXES collection:

$$\begin{array}{l}\left\langle\ \texttt{oid}-n+i\quad \texttt{sid}-n+i\quad \texttt{x}-\langle r_i,s_i\rangle\ \right\rangle,\\ \left\langle\ \texttt{sid}-n+i\quad \texttt{t}-\langle 0,0\rangle\quad \texttt{l}-\langle 1,e_i-s_i+1\rangle\ \right\rangle.\end{array}$$

Finally, a precedence constraint between two distinct tasks t_i and t_j (with $i,j \in [1,n]$) is modelled by an inequality constraint between the real end of task t_i and the real start of task t_j, namely $re_i \leq rs_j$. Figure 3.62 provides a toy example of such problem with:

- Four machines, numbered from 1 to 4, where:
  - Machine m_1 has two crossable unavailability periods respectively corresponding to intervals $[2,2]$ and $[6,7]$.
  - Machine m_2 has two crossable unavailability periods respectively corresponding to intervals $[2,2]$ and $[6,7]$, as well as one non-crossable unavailability period corresponding to interval $[3,3]$.
  - Machine m_3 has a single non-crossable unavailability corresponding to interval $[6,8]$.
  - Machine m_4 has a single crossable unavailability period corresponding to interval $[3,4]$.
- Five tasks, numbered from 1 to 5, where:
  - Task t_1 is a non-resumable task that has a virtual duration of 3.
  - Task t_2 is a resumable task that has a virtual duration of 2.
  - Task t_3 is a non-resumable task that has a virtual duration of 3.
  - Task t_4 is a resumable task that has a virtual duration of 5.
  - Task t_5 is a resumable task that has a virtual duration of 2.
- Finally, (1) all five tasks should not overlap, (2) task t_3 should precedes task t_2 and (3) task t_1 should precedes task t_5.

A survey on machine scheduling problems with unavailability constraints both in the deterministic and stochastic cases can be found in [377]. Unavailability can have multiple causes such as:

- In the context of production scheduling, machine unavailability corresponds to accepted orders that were already scheduled for a given date. This can typically corresponds to unavailability periods at the beginning of the planning horizon. Preemptive maintenance can also be another cause of machine unavailability.

- In the context of timetabling, unavailability periods may come from work regulation which enforces not to work in a continuous way more than a given limit. Unavailability periods may also come from scheduled meetings during the working day.

- In the context of distributed computing where cpu time is donated for performing huge tasks, machines are typically partially available [148].

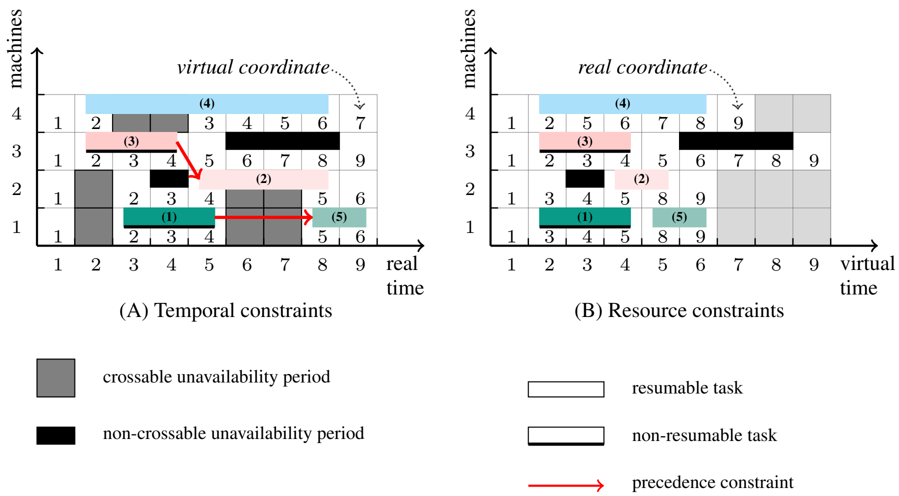


Figure 3.62: Illustration of the scheduling problem with *crossable and non-crossable unavailability periods* as well as with *resumable and non-resumable tasks*: part (A) gives the real time coordinate system where all precedence constraints are stated, while part (B) provides the virtual time coordinate system – from which all crossable unavailability periods are removed – where the non-overlapping constraint is stated

3.7.227 Shared table [3 CONS]

- CASE,
- ELEMENTS,
- ELEMENTS_SPARSE.

A constraint for which the same table is shared by several ELEMENT constraints. Within the context of the CASE constraint, the same directed acyclic graph can be shared by several tuples of variables. This happen, for example, when the CASE constraint is used for encoding all the transitions of an automaton [39].

Within the context of planning, the idea of reusing the same constraint for encoding the transitions of an automaton[16] was proposed under the name *slice encoding* by C. Pralet and G. Verfaillie in [332]. The motivation behind was to avoid to completely unfold the behaviour of the automaton (i.e., the successive triggered transitions) over the full planning horizon. From an implementation point of view, this encoding requires the possibility to reset the domains of the variables to some initial state.

[16]Even though the original work was not presented in the context of automata, it can be partly reinterpreted as the encoding of an automaton.

3.7.228 Schur number [1 CONS]

- SUM_FREE.

Denotes that a constraint was used for solving Schur problems. Given a non-negative integer k, the *Schur number* $S(k)$ is the largest integer n for which the set $\{1, 2, \ldots, n\}$ can be partitioned into k sets $S_1, S_2, \ldots, S_k$ such that $\forall i \in [1, k] : i \in S_i \Rightarrow i + i \notin S_i$.

3.7.229 SLAM problem [1 CONS]

- NVECTOR.

Denotes that a constraint was used in the context of the *simultaneous localisation and map building* (SLAM) problem. Given a mobile autonomous robot that, for some reason do not has a direct way to perform self-location (for example, do not has a GPS), the problem is to dynamically build a map and locate its trajectory on that map from a set of partial snapshots of its environment. Within the context of constraint programming this problem is described in [237, 117].

3.7.230 Sliding cyclic(1) constraint network(1) [7 CONS]

- DECREASING,
- INCREASING,
- NO_PEAK,
- NO_VALLEY,
- NOT_ALL_EQUAL,
- STRICTLY_DECREASING,
- STRICTLY_INCREASING.

A constraint network corresponding to the pattern depicted by Figure 3.63. Circles depict variables, while arcs are represented by a set of variables.

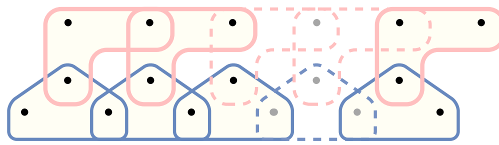


Figure 3.63: Hypergraph associated with a sliding cyclic(1) constraint network(1)

3.7.231 Sliding cyclic(1) constraint network(2) [20 CONS]

- ALL_EQUAL_PEAK,
- ALL_EQUAL_PEAK_MAX,
- ALL_EQUAL_VALLEY,
- ALL_EQUAL_VALLEY_MIN,
- CHANGE,
- CHANGE_CONTINUITY,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- DECREASING_PEAK,
- DECREASING_VALLEY,
- DEEPEST_VALLEY,
- HIGHEST_PEAK,
- INCREASING_PEAK,
- INCREASING_VALLEY,
- INFLEXION,
- LENGTH_FIRST_SEQUENCE,
- LENGTH_LAST_SEQUENCE,
- PEAK,
- SMOOTH,
- VALLEY.

A constraint network corresponding to the pattern depicted by Figure 3.64. Circles depict variables, while arcs are represented by a set of variables.

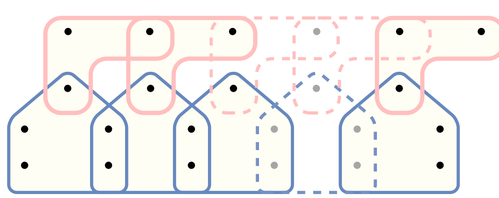


Figure 3.64: Hypergraph associated with a sliding cyclic(1) constraint network(2)

3.7.232 Sliding cyclic(1) constraint network(3) [3 CONS]

- CHANGE_CONTINUITY,
- LONGEST_CHANGE,
- MIN_DIST_BETWEEN_INFLEXION.

A constraint network corresponding to the pattern depicted by Figure 3.65. Circles depict variables, while arcs are represented by a set of variables.

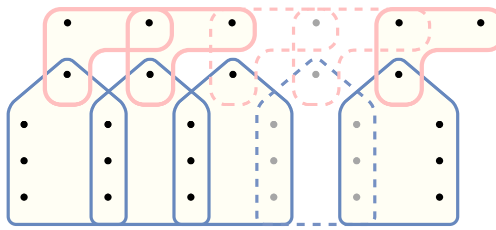


Figure 3.65: Hypergraph associated with a sliding cyclic(1) constraint network(3)

3.7.233 Sliding cyclic(2) constraint network(2) [2 CONS]

- CHANGE_PAIR,
- DISTANCE_CHANGE.

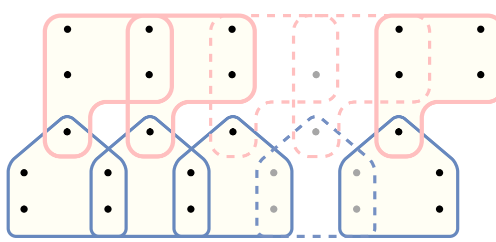


Figure 3.66: Hypergraph associated with a sliding cyclic(2) constraint network(2)

A constraint network corresponding to the pattern depicted by Figure 3.66. Circles depict variables, while arcs are represented by a set of variables.

3.7.234 Sliding sequence constraint [17 CONS]

- AMONG_SEQ,
- ARITH_SLIDING,
- CYCLE_CARD_ON_PATH,
- ELEMENTN,
- PATTERN,
- RELAXED_SLIDING_SUM,
- SLIDING_CARD_SKIP0,
- SLIDING_DISTRIBUTION,
- SIZE_MAX_SEQ_ALLDIFFERENT,
- SIZE_MAX_STARTING_SEQ_ALLDIFFERENT,
- SLIDING_SUM,
- SLIDING_TIME_WINDOW,
- SLIDING_TIME_WINDOW_FROM_START,
- SLIDING_TIME_WINDOW_SUM,
- STRETCH_CIRCUIT,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION.

A constraint enforcing a condition on sliding sequences of domain variables that partially overlap or a constraint computing a quantity from a set of sliding sequences. These sliding sequences can be either initially given or dynamically constructed. In the latter case they can correspond, for example, to adjacent vertices of a path that has to be built.

3.7.235 Smallest square for packing consecutive dominoes

[2 CONS]

- DIFFN,
- GEOST.

Find the smallest square S where one can place n rectangles of respective size $1 \times 2, 2 \times 4, \ldots, n \times 2 \cdot n$ so that they do not overlap and so that their borders are parallel to the borders of S. Each rectangle can be rotated by 90 degrees. The problem is described in http://www.stetson.edu/~efriedma/domino/. Figure 3.67 gives a solution for $n = 22$ found by H. Simonis.

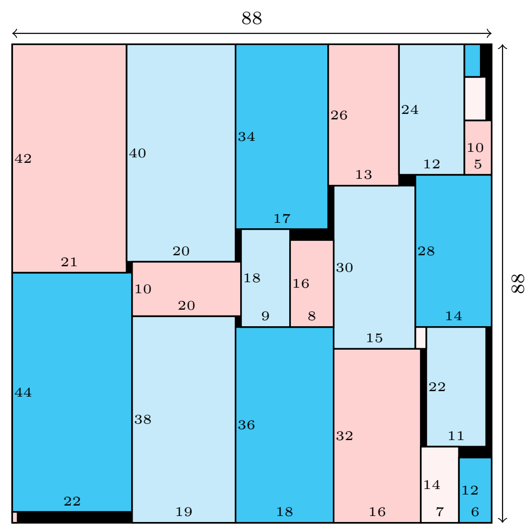


Figure 3.67: A solution to the smallest square for packing consecutive dominoes problem for $n = 22$

3.7.236 Smallest rectangle area [2 CONS]

- DIFFN,
- GEOST.

Denotes that a constraint can be used for finding the smallest rectangle area where one can pack a given set of rectangles (or squares). A first example of such packing problem attributed to S. W. Golomb is to find the smallest square that can contain the set of consecutive squares from 1×1 up to $n \times n$ so that these squares do not overlap each other. A program using the DIFFN constraint was used to construct such a table for $n \in \{1, 2, \ldots, 25, 27, 29, 30\}$ in [31]. New optimal solutions for this problem were found in [401] for $n = 26, 31, 35$. Figure 3.68 gives the solution found for $n = 35$ by H. Simonis and B. O'Sullivan. Algorithms and lower bounds for solving the same problem can also be respectively found in [99] and in [253].

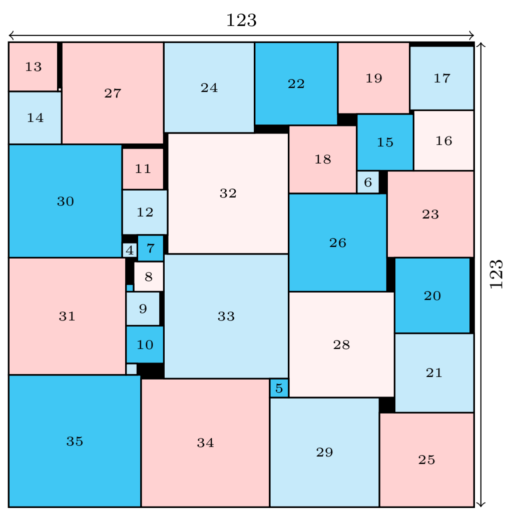


Figure 3.68: Smallest square (of size 123) for packing squares of size $1, 2, \ldots, 35$

In his paper (i.e., [253]), Richard E. Korf also considers the problem of finding the minimum-area rectangle that can contain the set of consecutive squares from 1×1 up to $n \times n$ and solve it up to $n = 25$. In 2008 this value was improved up to $n = 27$ by H. Simonis and B. O'Sullivan [401]. Figure 3.69 gives the solution found for $n = 27$ by H. Simonis and B. O'Sullivan.

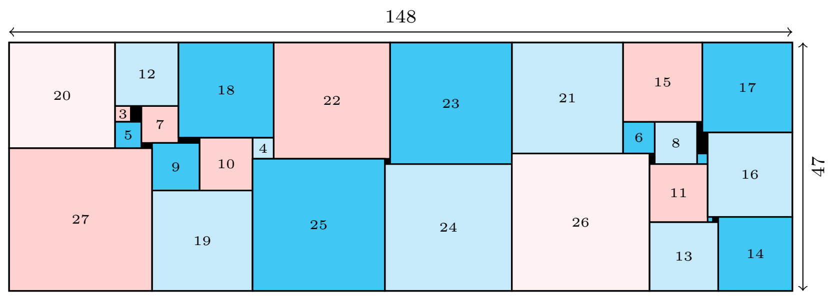


Figure 3.69: Rectangle with the smallest surface (of size 148×47) for packing squares of size $1, 2, \ldots, 27$

3.7.237 Smallest square for packing rectangles with distinct sizes

[2 CONS]

- DIFFN,
- GEOST.

Denotes that a constraint can be used for finding the smallest square where one can pack n rectangles for which all the $2 \cdot n$ sizes are distinct integer values. The problem is described in `http://www.stetson.edu/~efriedma/mathmagic/0899.html`. Figures 3.70, 3.71 and 3.72 present the smallest square (not necessarily optimal) found with GEOST for respectively placing 9, 10, 11, 12, 13 and 14 rectangles of distinct sizes.

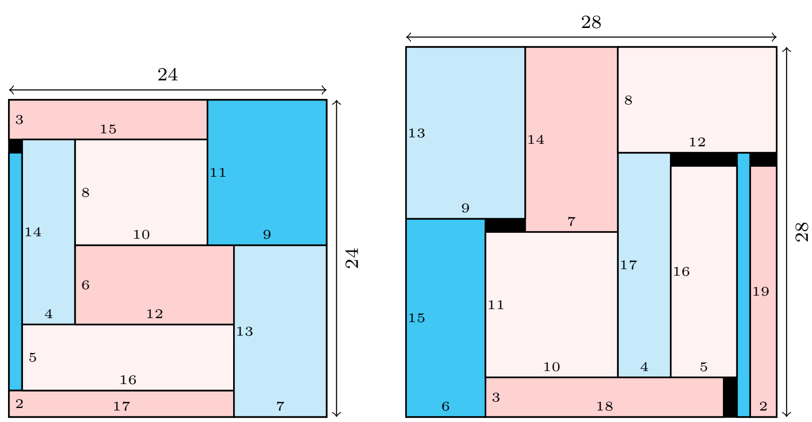


Figure 3.70: (Left) Tiling a square of size 24 with 9 rectangles of distinct sizes 1×18, 17×2, 15×3, 4×14, 16×5, 12×6, 7×13, 10×8, 9×11; (Right) Tiling a square of size 28 with 10 rectangles of distinct sizes 1×20, 2×19, 18×3, 4×17, 5×16, 6×15, 7×14, 12×8, 9×13, 10×11.

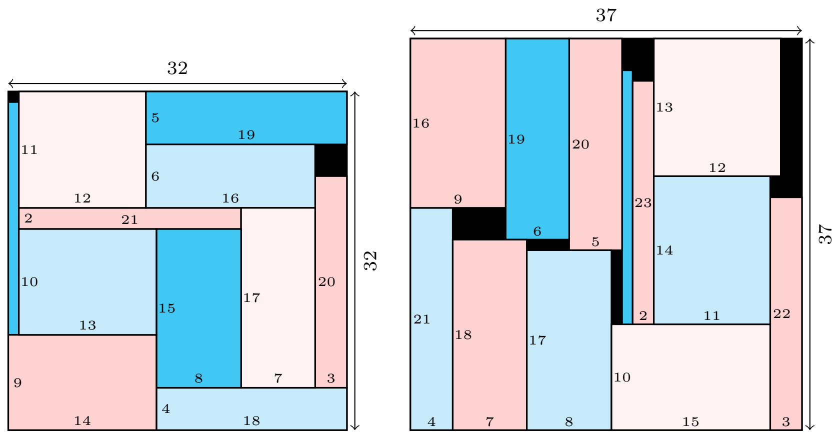


Figure 3.71: (Left) Tiling a square of size 32 with 11 rectangles of distinct sizes 1×22, 21×2, 3×20, 18×4, 19×5, 16×6, 7×17, 8×15, 14×9, 13×10, 12×11; (Right) Tiling a square of size 37 with 12 rectangles of distinct sizes 1×24, 2×23, 3×22, 4×21, 5×20, 6×19, 7×18, 8×17, 9×16, 15×10, 11×14, 12×13.

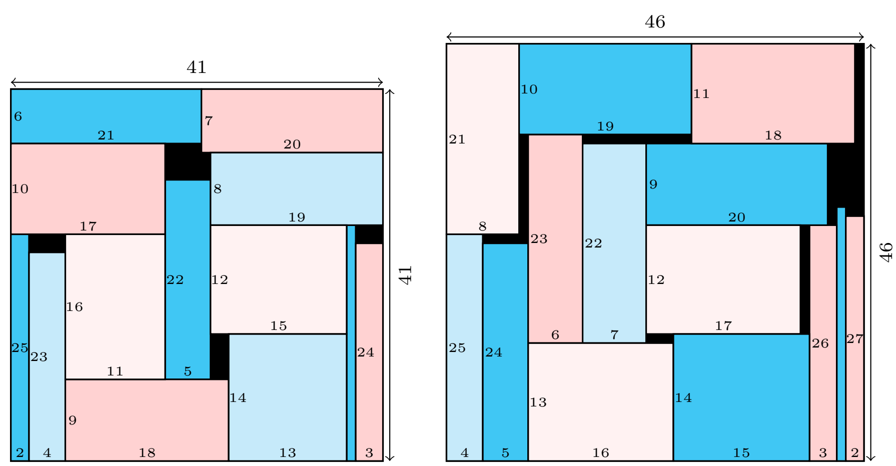


Figure 3.72: (Left) Tiling a square of size 41 with 13 rectangles of distinct sizes 1×26, 2×25, 3×24, 4×23, 5×22, 21×6, 20×7, 19×8, 18×9, 17×10, 11×16, 15×12, 13×14; (Right) Tiling a square of size 46 with 14 rectangles of distinct sizes 1×28, 2×27, 3×26, 4×25, 5×24, 6×23, 7×22, 8×21, 20×9, 19×10, 18×11, 17×12, 16×13, 15×14.

3.7.238 Soft constraint [17 CONS]

- OPEN_ALLDIFFERENT,
- RELAXED_SLIDING_SUM,
- SOFT_ALLDIFFERENT_CTR,
- SOFT_ALLDIFFERENT_VAR,
- SOFT_ALL_EQUAL_MAX_VAR,
- SOFT_ALL_EQUAL_MIN_CTR,
- SOFT_ALL_EQUAL_MIN_VAR,
- SOFT_CUMULATIVE,
- SOFT_SAME_INTERVAL_VAR,
- SOFT_SAME_MODULO_VAR,
- SOFT_SAME_PARTITION_VAR,
- SOFT_SAME_VAR,
- SOFT_USED_BY_INTERVAL_VAR,
- SOFT_USED_BY_MODULO_VAR,
- SOFT_USED_BY_PARTITION_VAR,
- SOFT_USED_BY_VAR,
- WEIGHTED_PARTIAL_ALLDIFF.

A constraint that is a relaxed form of one other constraint.

3.7.239 Sort [2 CONS]

- SORT,
- SORT_PERMUTATION.

A constraint involving the notion of sorting in its definition.

3.7.240 Sort based reformulation [31 CONS]

- ALL_MIN_DIST,
- ALLDIFFERENT,
- ALLDIFFERENT_CONSECUTIVE_VALUES,
- ALLDIFFERENT_CST,
- ALLDIFFERENT_EXCEPT_0,
- ALLDIFFERENT_INTERVAL,
- ALLDIFFERENT_MODULO,
- ALLDIFFERENT_PARTITION,
- ALLDIFFERENT_SAME_VALUE,
- ALLPERM,

- CONSECUTIVE_VALUES,
- DERANGEMENT,
- DISJUNCTIVE,
- K_SAME,
- K_SAME_INTERVAL,
- K_SAME_MODULO,
- K_SAME_PARTITION,
- K_USED_BY,
- K_USED_BY_INTERVAL,
- K_USED_BY_MODULO,
- K_USED_BY_PARTITION,
- PERMUTATION,
- SAME,
- SAME_INTERVAL,
- SAME_MODULO,
- SAME_PARTITION,
- SOME_EQUAL,
- USED_BY,
- USED_BY_INTERVAL,
- USED_BY_MODULO,
- USED_BY_PARTITION.

A constraint using the SORT constraint in one of its reformulation.

3.7.241 Sparse functional dependency [3 CONS]

- CASE,
- ELEMENT_SPARSE,
- ELEMENTS_SPARSE.

A constraint that allows for representing a functional dependency between two domain variables, where both variables have a restricted number of values. A variable X is said to *functionally determine* another variable Y if and only if each potential value of X is associated with exactly one potential value of Y.

3.7.242 Sparse table [2 CONS]

- ELEMENT_SPARSE,
- ELEMENTS_SPARSE.

An ELEMENT constraint for which the table is sparse.

3.7.243 Sport timetabling [2 CONS]

- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_EXCEPT_0.

A constraint used for creating sports schedules.

3.7.244 Squared squares [3 CONS]

- CUMULATIVE,
- DIFFN,
- GEOST.

A constraint that can be used for modelling the *squared squares* problem [137] [440] (also called the *perfect squared squares* problem [156]): a *perfect squared square of order* n is a square that can be tiled with n smaller squares such that each of the smaller squares has a different integer size. It is called *simple* if it does not contain a subset of at least two squares, corresponding to a square or to a rectangle. Duijvestijn has shown in 1962 that no instances exist with less than 21 squares [156]. A single solution depicted by Figure 3.73 exists with 21 squares, where the squares have sizes 2, 4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 24, 25, 27, 29, 33, 35, 37, 42, 50 and must be packed into a square of size 112.

A catalogue of such simple squared squares of orders 21 through 25 is provided in [94]. The following table contains all the problem instances from the previous

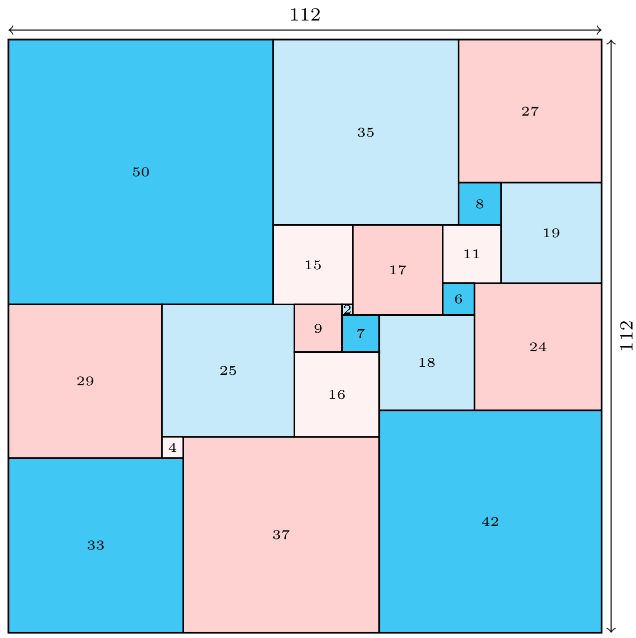


Figure 3.73: A simple perfect squared square of order 21

catalogue. The different fields respectively give the problem number, the number of squares, the size of the master square and a list of the square sizes. Problems 166 and 167, 168 and 169, 182 and 183 are identical, but have two non-isomorphic solutions. A much bigger table can be found at the following link `http://www.squaring.net/`.

When the size of the squares is known four constraint programming approach are respectively reported in [1], in [427], in [398], in [42] and in [41].

1	21	112	2,4,6,7,8,9,11,15,16,17,18,19,24,25,27,29,33,35,37,42,50
2	22	110	2,3,4,6,7,8,12,13,14,15,16,17,18,21,22,23,24,26,27,28,50,60
3	22	110	1,2,3,4,6,8,9,12,14,16,17,18,19,21,22,23,24,26,27,28,50,60
4	22	139	1,2,3,4,7,8,10,17,18,20,21,22,24,27,28,29,30,31,32,38,59,80
5	22	147	1,3,4,5,8,9,17,20,21,23,25,26,29,31,32,40,43,44,47,48,52,55
6	22	147	2,4,8,10,11,12,15,19,21,22,23,25,26,32,34,37,41,43,45,47,55,59
7	22	154	2,5,9,11,16,17,19,21,22,24,26,30,31,33,35,36,41,46,47,50,52,61
8	22	172	1,2,3,4,9,11,13,16,17,18,19,22,24,33,36,38,39,42,44,53,75,97
9	22	192	4,8,9,10,12,14,17,19,26,28,31,35,36,37,41,47,49,57,59,62,71,86
10	23	110	1,2,3,4,5,7,8,10,12,13,14,15,16,19,21,28,29,31,32,37,38,41,44
11	23	139	1,2,7,8,12,13,14,15,16,18,19,20,21,22,24,26,27,28,32,33,38,59,80
12	23	140	1,2,3,4,5,8,10,13,16,19,20,23,27,28,29,31,33,38,42,45,48,53,54
13	23	140	2,3,4,7,8,9,12,15,16,18,22,23,24,26,28,30,33,36,43,44,47,50,60
14	23	145	1,2,3,4,6,8,9,12,15,20,22,24,25,26,27,29,30,31,32,34,36,61,84
15	23	180	2,4,8,10,11,12,15,19,21,22,23,25,26,32,33,34,37,41,43,45,47,88,92
16	23	188	2,4,8,10,11,12,15,19,21,22,23,25,26,32,33,34,37,45,47,49,51,92,96
17	23	208	1,3,4,9,10,11,12,16,17,18,22,23,24,40,41,60,62,65,67,70,71,73,75
18	23	215	1,3,4,9,10,11,12,16,17,18,22,23,24,40,41,60,66,68,70,71,74,76,79
19	23	228	2,7,9,10,15,16,17,18,22,23,25,28,36,39,42,56,57,68,69,72,73,87,99
20	23	257	2,3,9,11,14,15,17,20,22,24,28,29,32,33,49,55,57,60,63,66,79,123,134
21	23	332	1,15,17,24,26,30,31,38,47,48,49,50,53,56,58,68,83,89,91,112,120,123,129
22	24	120	3,4,5,6,8,9,10,12,13,14,15,16,17,19,20,23,25,32,33,34,40,41,46,47

23	24	186	2,3,4,7,8,9,12,15,16,18,22,23,24,26,28,30,33,36,43,46,47,60,90,96
24	24	194	2,3,7,9,10,16,17,18,19,20,23,25,28,34,36,37,42,53,54,61,65,68,69,72
25	24	195	2,4,7,10,11,16,17,18,21,26,27,30,39,41,42,45,47,49,52,53,54,61,63,80
26	24	196	1,2,5,10,11,15,17,18,20,21,24,26,29,31,32,34,36,40,44,47,48,51,91,105
27	24	201	1,3,4,6,9,10,11,12,17,18,20,21,22,23,26,38,40,46,50,52,53,58,98,103
28	24	201	1,4,5,8,9,10,11,15,16,18,19,20,22,24,26,39,42,44,49,52,54,56,93,108
29	24	203	1,2,5,10,11,15,17,18,20,21,24,26,29,31,32,34,36,40,44,48,54,58,98,105
30	24	247	3,5,6,9,12,14,19,23,24,25,28,32,34,36,40,45,46,48,56,62,63,66,111,136
31	24	253	2,4,5,9,13,18,20,23,24,27,28,31,38,40,44,50,61,70,72,77,79,86,88,104
32	24	255	3,5,10,11,16,17,20,22,23,25,26,27,28,32,41,44,52,53,59,63,65,74,118,137
33	24	288	2,7,9,10,15,16,17,18,22,23,25,28,36,39,42,56,57,60,68,72,73,87,129,159
34	24	288	1,5,7,8,9,14,17,20,21,26,30,32,34,36,48,51,54,59,64,69,72,93,123,165
35	24	290	2,3,8,9,11,12,14,17,21,30,31,33,40,42,45,48,59,61,63,65,82,84,124,166
36	24	292	1,2,3,8,12,15,16,17,20,22,24,26,29,33,44,54,57,60,63,67,73,102,117,175
37	24	304	3,5,7,11,12,17,20,22,25,29,35,47,48,55,56,57,69,72,76,92,96,100,116,132
38	24	304	3,4,7,12,16,20,23,24,27,28,30,32,33,36,37,44,53,57,72,76,85,99,129,175
39	24	314	2,4,11,12,16,17,18,19,28,29,40,44,47,59,62,64,65,78,79,96,97,105,113,139
40	24	316	3,9,10,12,13,14,15,23,24,33,36,37,48,52,54,55,57,65,66,78,79,93,144,172
41	24	326	1,6,10,11,14,15,18,24,29,32,43,44,53,56,63,65,71,80,83,101,104,106,119,142
42	24	423	2,9,15,17,27,29,31,32,33,36,47,49,50,60,62,77,105,114,123,127,128,132,168,186
43	24	435	1,2,8,10,13,19,23,33,44,45,56,74,76,78,80,88,93,100,112,131,142,143,150,192
44	24	435	3,5,9,11,12,21,24,27,30,44,45,50,54,55,63,95,101,112,117,123,134,140,178,200
45	24	459	8,9,10,11,16,30,36,38,45,55,57,65,68,84,95,98,100,116,117,126,135,144,180,198
46	24	459	4,6,9,10,17,21,23,25,31,33,36,38,45,50,83,115,117,126,133,135,144,146,180,198
47	24	479	5,6,17,23,24,26,28,29,35,43,44,52,60,68,77,86,130,140,150,155,160,164,174,175
48	25	147	3,4,5,6,8,9,10,12,13,14,15,16,17,19,20,23,25,27,32,33,34,40,41,73,74
49	25	208	1,2,3,4,5,7,8,11,12,17,18,24,26,28,29,30,36,39,44,45,50,59,60,89,119
50	25	213	3,5,6,7,13,16,17,20,21,23,24,25,26,28,31,35,36,47,49,56,58,74,76,81,90
51	25	215	1,4,6,7,11,15,24,26,27,33,37,39,40,41,42,43,45,47,51,55,60,62,63,69,83
52	25	216	1,2,3,4,5,7,8,11,16,17,18,19,25,30,32,33,39,41,45,49,54,59,64,103,113
53	25	236	1,2,4,9,11,12,13,14,15,16,19,24,38,40,44,46,47,48,59,64,65,70,81,85,107
54	25	242	1,3,6,7,9,13,14,16,17,19,23,25,26,28,30,31,47,51,54,57,60,64,67,111,131
55	25	244	1,2,4,5,7,10,15,17,19,20,21,22,26,27,30,37,40,41,45,65,66,68,70,110,134
56	25	252	4,7,10,11,12,13,23,25,29,31,32,34,36,37,38,40,42,44,62,67,68,71,77,108,113
57	25	253	2,4,5,6,9,10,12,14,20,24,27,35,36,37,38,42,43,45,50,54,63,66,70,120,133
58	25	260	1,4,6,7,10,15,24,26,27,28,29,31,33,34,37,38,44,65,70,71,77,78,83,100,112
59	25	264	3,7,8,12,16,18,19,20,22,24,26,31,34,37,38,40,42,53,54,61,64,69,70,130,134
60	25	264	3,8,12,13,16,18,20,21,22,24,26,29,34,38,40,42,43,47,54,59,64,70,71,130,134
61	25	264	1,3,4,6,9,10,11,12,16,17,18,20,21,22,39,42,54,56,61,66,68,69,73,129,135
62	25	265	1,3,4,6,9,10,11,12,16,17,18,20,21,22,39,42,54,56,62,66,68,69,74,130,135
63	25	273	1,4,8,10,11,12,17,19,21,22,27,29,30,33,37,43,52,62,65,86,88,89,91,96,120
64	25	273	1,6,9,14,16,17,18,21,22,23,25,31,32,38,44,46,48,50,54,62,65,68,78,133,140
65	25	275	2,3,7,13,17,24,25,31,33,34,35,37,41,49,51,53,55,60,68,71,74,81,94,100,107
66	25	276	1,5,8,9,11,18,19,21,30,36,41,44,45,46,47,51,53,58,63,69,71,84,87,105,120
67	25	280	5,6,11,17,18,20,21,24,27,28,32,34,41,42,50,53,54,55,68,78,85,88,95,97,117
68	25	280	2,3,7,8,14,18,30,36,37,39,44,50,52,54,56,60,63,64,65,72,75,78,79,96,106
69	25	284	1,2,11,12,14,16,18,19,23,26,29,37,38,39,40,42,59,68,69,77,78,97,106,109,110
70	25	286	1,4,5,7,10,12,15,16,20,23,28,30,32,33,35,37,53,54,64,68,74,79,80,133,153
71	25	289	2,3,5,8,13,14,17,20,21,32,36,41,50,52,60,61,62,68,74,76,83,87,100,102,104
72	25	289	2,3,4,5,7,12,16,17,19,21,23,25,29,31,32,44,57,64,65,68,72,76,84,140,149
73	25	290	1,2,10,11,13,14,15,17,18,28,29,34,36,38,50,56,60,69,77,80,85,91,94,111,119
74	25	293	5,6,11,17,18,20,21,24,27,28,32,34,41,42,50,54,55,66,68,78,85,88,95,110,130
75	25	297	2,7,8,9,10,15,16,17,18,23,25,26,28,36,38,43,53,60,61,68,69,77,99,137,160
76	25	308	1,3,4,7,10,12,13,23,25,34,37,38,39,43,44,45,62,77,79,85,87,108,113,115,116
77	25	308	1,5,6,7,8,9,13,16,19,28,33,36,38,43,45,48,70,71,73,84,86,102,104,120,133
78	25	309	7,8,14,16,23,24,25,26,31,33,34,39,48,56,59,60,62,70,76,82,92,100,101,108,117
79	25	311	2,7,8,9,10,15,16,17,18,23,25,26,28,36,38,43,53,60,61,68,83,91,99,151,160
80	25	314	1,6,7,11,16,22,26,29,32,36,38,44,51,53,64,69,70,73,74,75,85,87,101,116,128
81	25	316	1,3,9,12,21,26,30,33,34,35,38,39,40,41,53,56,59,69,79,85,96,103,111,117,120
82	25	317	1,5,6,7,8,9,16,17,19,32,37,40,42,47,49,52,59,75,81,92,94,110,112,113,126
83	25	320	2,7,8,9,12,14,15,21,23,35,38,44,46,49,53,54,56,63,96,101,103,105,108,112,116
84	25	320	3,8,9,11,17,18,22,25,26,27,29,30,31,33,35,49,51,67,72,73,80,85,95,152,168
85	25	320	1,4,6,7,8,13,14,16,24,28,30,33,34,38,41,42,57,60,69,78,81,90,92,150,170
86	25	320	3,4,6,8,9,14,15,16,24,28,30,31,34,38,39,42,59,60,71,78,79,90,92,150,170

87	25	322	3,4,8,9,10,16,18,20,22,23,24,28,31,38,44,47,64,65,68,76,80,81,97,144,178
88	25	322	3,4,8,10,15,16,18,19,20,22,24,28,35,38,44,53,59,64,68,76,80,85,93,144,178
89	25	323	2,3,4,7,10,13,15,18,23,32,34,35,36,42,46,50,57,60,66,72,78,87,98,159,164
90	25	323	3,8,9,11,17,18,22,25,26,27,29,30,31,33,35,49,51,67,72,73,83,88,95,155,168
91	25	323	2,6,9,11,13,14,18,19,20,23,27,28,29,42,46,48,60,64,72,74,79,82,98,146,177
92	25	325	3,5,6,11,12,13,18,23,25,28,32,37,40,43,45,46,51,79,92,99,103,108,112,114,134
93	25	326	1,4,8,10,12,16,21,22,24,27,28,35,36,37,38,46,49,68,70,75,88,90,93,158,168
94	25	327	2,9,10,12,13,16,19,21,23,26,36,44,46,52,55,61,62,74,84,87,100,103,104,120,140
95	25	328	2,3,4,7,8,10,14,17,26,27,28,36,38,40,42,45,53,58,73,74,79,94,102,152,176
96	25	334	1,4,8,10,12,16,21,22,24,27,28,35,36,37,38,46,49,68,75,78,88,93,98,166,168
97	25	336	2,3,4,7,8,10,14,17,26,27,28,36,38,40,45,50,53,58,73,74,79,94,110,152,184
98	25	338	1,4,8,10,12,16,19,22,24,25,28,36,37,38,39,46,53,68,70,73,94,96,101,164,174
99	25	338	4,5,8,10,12,15,16,21,22,24,28,33,36,38,43,46,57,68,70,77,94,96,97,164,174
100	25	340	1,4,5,6,11,13,16,17,22,24,44,46,50,51,52,53,61,64,66,79,84,85,92,169,171
101	25	344	2,3,8,11,14,17,19,21,23,25,27,36,39,44,48,53,56,71,77,83,86,89,98,169,175
102	25	359	7,8,9,10,14,17,18,23,25,27,29,31,40,41,43,46,69,74,82,85,90,98,102,172,187
103	25	361	2,6,7,8,9,14,20,22,26,27,32,34,36,47,49,56,66,67,74,82,89,98,107,156,205
104	25	363	1,4,6,12,13,20,21,25,26,27,28,32,37,41,45,53,58,64,69,91,97,102,106,155,208
105	25	364	2,3,4,6,8,9,13,14,16,19,23,24,28,29,52,57,64,75,82,91,98,100,109,173,191
106	25	367	1,4,6,12,13,20,21,25,26,27,28,32,37,41,49,53,58,64,69,91,97,102,110,155,212
107	25	368	1,6,15,16,17,18,22,25,31,33,39,42,45,46,47,48,51,69,72,88,91,96,112,160,208
108	25	371	1,2,7,8,20,21,22,24,26,28,30,38,43,46,50,51,64,65,70,90,95,102,109,160,211
109	25	373	3,6,7,8,15,17,22,23,31,32,35,41,43,60,62,68,79,87,104,105,114,120,121,138,148
110	25	378	2,3,10,17,18,20,21,22,24,27,31,38,41,48,51,56,68,78,80,85,87,96,117,165,213
111	25	378	1,2,7,13,15,17,18,25,27,29,30,31,42,43,46,56,61,68,73,93,100,105,112,161,217
112	25	380	4,7,17,18,19,20,21,26,31,33,35,40,45,48,49,60,67,73,79,81,87,107,113,186,194
113	25	380	4,5,6,9,13,15,16,17,22,24,33,38,44,49,50,56,60,67,82,84,95,108,121,177,203
114	25	381	12,13,21,23,25,27,35,36,42,45,54,57,59,60,79,82,84,85,92,95,96,100,110,111,186
115	25	384	1,4,8,9,11,12,19,21,27,32,35,44,45,46,47,51,60,67,84,89,96,108,120,180,204
116	25	384	1,4,8,9,11,12,15,17,19,25,26,31,32,37,44,57,60,81,84,96,99,108,120,180,204
117	25	384	3,5,7,11,12,17,20,22,25,29,35,47,48,55,56,57,69,72,76,80,96,100,116,172,212
118	25	385	1,2,7,13,15,17,18,25,27,29,30,31,43,46,49,56,61,68,73,93,100,105,119,161,224
119	25	392	4,7,8,15,23,26,29,30,31,32,34,43,48,55,56,68,77,88,98,106,116,135,141,151,153
120	25	392	10,12,14,16,19,21,25,27,31,35,39,41,51,52,54,55,73,92,98,115,121,123,129,148,171
121	25	392	1,4,5,8,11,14,16,21,22,24,27,28,30,31,52,64,81,83,96,97,98,99,114,195,197
122	25	393	4,8,16,20,23,24,25,27,29,37,44,45,50,53,64,66,68,69,73,85,91,101,116,186,207
123	25	396	1,4,5,14,16,32,35,36,46,47,48,49,68,69,73,93,94,97,99,104,110,111,125,126,160
124	25	396	1,4,5,8,11,14,16,21,22,24,27,28,30,31,52,64,81,83,98,99,100,101,114,197,199
125	25	396	3,8,9,11,14,16,17,18,31,32,41,45,48,56,60,66,73,75,81,82,98,99,117,180,216
126	25	398	2,6,7,11,15,17,23,28,29,39,44,46,53,56,58,65,68,99,100,119,120,134,144,145,154
127	25	400	3,6,21,23,24,26,29,35,37,40,41,47,53,55,64,76,79,81,99,100,121,122,137,142,179
128	25	404	3,6,7,14,17,20,21,26,28,31,32,39,46,53,54,68,71,80,88,92,100,111,113,199,205
129	25	404	4,7,10,11,12,13,16,18,20,23,25,28,29,32,47,62,70,88,93,96,101,114,127,189,215
130	25	408	2,3,7,13,16,18,20,27,30,33,41,43,46,52,54,57,72,79,84,100,105,108,116,195,213
131	25	412	3,11,12,15,21,26,32,39,43,47,54,60,68,73,83,85,86,87,89,99,114,129,139,144,169
132	25	413	5,7,17,20,34,38,39,48,56,57,59,60,64,65,70,72,75,81,105,106,110,125,148,153,155
133	25	416	2,4,7,11,13,24,25,30,35,37,39,40,44,58,62,65,82,104,112,120,128,135,143,153,169
134	25	416	1,2,3,8,12,15,16,17,20,22,24,26,29,31,64,75,85,88,91,94,98,104,133,179,237
135	25	421	1,2,4,5,7,9,12,16,20,22,23,35,38,48,56,83,94,104,116,118,128,140,150,153,177
136	25	421	5,11,12,17,18,20,23,26,29,36,38,40,44,51,55,59,72,92,97,102,105,107,117,199,222
137	25	422	2,4,7,13,16,18,20,23,28,29,38,43,46,51,59,68,74,79,86,93,100,111,132,179,243
138	25	425	3,4,5,9,10,12,13,14,16,19,20,31,46,48,56,79,102,104,116,126,128,140,142,157,181
139	25	441	5,6,7,16,18,23,24,27,38,39,47,51,52,62,66,72,80,84,92,101,102,118,120,219,222
140	25	454	1,2,11,17,29,34,35,46,48,51,53,55,63,69,79,87,88,91,109,134,136,143,150,161,184
141	25	456	5,7,10,11,13,15,18,19,31,49,50,52,59,60,63,72,77,115,128,129,135,142,148,179,193
142	25	465	6,9,13,14,19,21,24,25,31,32,53,56,64,73,74,82,91,111,125,127,137,139,153,173,201
143	25	472	7,9,13,15,26,34,35,44,47,51,58,61,65,81,87,103,104,115,118,123,128,133,136,148,221
144	25	477	3,5,12,16,19,22,25,26,37,41,49,72,76,77,82,86,87,115,117,135,141,149,167,169,193
145	25	492	2,9,15,17,27,29,31,32,33,36,47,49,50,60,62,69,77,105,114,123,127,128,132,237,255
146	25	492	3,5,9,11,12,21,24,27,30,44,45,50,54,55,57,63,95,101,112,117,123,134,140,235,257
147	25	503	4,15,16,19,22,23,25,27,33,34,50,62,67,87,88,93,100,113,135,143,149,157,167,179,211
148	25	506	1,7,24,26,33,35,40,45,47,51,55,69,87,90,93,96,117,125,134,145,146,147,160,162,199
149	25	507	2,3,7,11,13,15,28,34,43,50,57,64,80,83,86,89,107,115,116,127,149,163,175,183,217
150	25	512	1,7,8,9,10,15,22,32,34,46,51,65,69,71,91,105,109,111,136,139,152,157,173,200,203

151	25	512	1,6,7,8,9,13,17,19,35,45,47,57,62,73,88,93,104,107,128,130,151,163,184,198,221
152	25	513	6,9,10,17,19,24,28,29,37,39,64,65,68,81,98,99,102,115,145,147,153,159,165,189,201
153	25	517	5,6,7,16,20,24,28,33,38,43,63,71,80,83,86,92,98,122,132,148,164,166,173,180,205
154	25	524	9,12,20,21,33,35,37,39,54,55,61,62,87,90,98,101,125,132,135,141,145,159,163,164,220
155	25	527	11,12,13,14,19,30,41,47,50,52,59,68,71,81,94,97,107,132,147,151,155,169,175,183,197
156	25	528	2,9,15,17,27,29,31,32,33,36,47,49,50,60,62,69,77,123,127,128,132,141,150,255,273
157	25	529	9,12,20,21,33,35,37,39,54,55,61,62,87,90,98,101,125,132,140,141,145,159,163,169,225
158	25	531	6,9,10,17,19,24,29,31,39,40,67,68,71,84,101,102,105,118,151,153,159,165,171,195,207
159	25	532	16,18,26,27,33,39,41,50,51,55,69,71,84,87,91,94,132,133,141,143,164,168,169,173,195
160	25	534	11,13,15,17,18,27,38,44,49,52,60,61,68,81,87,94,107,135,149,153,159,171,174,189,210
161	25	535	2,8,26,27,36,41,45,57,62,77,88,95,97,99,101,102,109,114,117,118,141,147,168,192,226
162	25	536	1,8,21,30,31,32,33,41,44,46,49,55,57,61,84,91,113,134,137,139,150,155,176,205,247
163	25	536	3,5,9,11,12,21,24,27,30,44,45,50,54,55,57,63,95,117,123,134,140,145,156,257,279
164	25	540	1,7,8,9,10,14,19,34,36,51,58,69,81,83,97,109,111,115,136,149,152,167,183,208,221
165	25	540	6,13,15,25,28,36,43,47,55,57,58,59,60.65,82,89,91,107,124,127,144,163,183,233,250
166	25	540	8,9,10,11,16,30,36,38,45,55,57,65,68,81,84,95,98,100,116,117,126,135,144,261,279
167	25	540	8,9,10,11,16,30,36,38,45,55,57,65,68,81,84,95,98,100,116,117,126,135,144,261,279
168	25	540	4,6,9,10,17,21,23,25,31,33,36,38,45,50,81,83,115,117,126,133,135,144,146,261,279
169	25	540	4,6,9,10,17,21,23,25,31,33,36,38,45,50,81,83,115,117,126,133,135,144,146,261,279
170	25	541	3,4,11,13,16,17,21,25,26,44,46,64,75,86,87,97,106,109,133,141,165,185,191,215,217
171	25	541	3,5,27,32,33,37,47,50,53,56,57,69,71,78,97,98,109,111,126,144,165,169,183,189,232
172	25	544	1,7,24,26,33,35,40,45,47,51,55,69,87,90,93,96,117,125,134,145,147,184,198,199,200
173	25	544	6,8,20,21,23,41,42,48,59,61,77,80,81,85,90,92,93,102,115,132,139,168,198,207,244
174	25	547	3,5,16,22,26,27,35,47,49,59,67,71,72,85,87,102,103,111,137,144,150,197,200,203,207
175	25	549	4,10,14,24,26,31,34,36,38,40,43,48,59,63,74,89,97,105,117,124,136,152,156,241,308
176	25	550	1,2,5,13,19,20,25,30,39,43,58,59,73,75,76,90,95,103,116,128,130,132,172,262,288
177	25	550	1,11,16,23,24,27,29,36,41,43,44,47,59,70,71,80,99,103,111,116,128,156,167,227,323
178	25	551	3,5,24,25,26,30,35,36,39,40,42,57,68,76,94,109,120,128,152,162,166,175,176,200,223
179	25	552	5,17,18,22,25,27,32,33,39,59,62,87,91,100,102,111,112,135,137,149,165,168,183,201,204
180	25	552	1,3,4,7,8,9,10,15,18,19,21,41,52,54,73,93,95,123,125,136,138,153,168,261,291
181	25	556	6,8,10,13,19,25,32,37,49,54,58,76,84,91,92,100,107,128,145,156,165,185,195,205,206
182	25	556	3,12,13,15,19,23,27,34,35,39,42,45,48,52,53,87,140,145,158,166,171,184,189,201,227
183	25	556	3,12,13,15,19,23,27,34,35,39,42,45,48,52,53,87,140,145,158,166,171,184,189,201,227
184	25	556	1,5,7,8,9,10,12,14,20,27,31,43,47,50,74,93,97,121,125,139,143,153,167,264,292
185	25	562	2,3,5,8,13,19,20,29,33,47,53,54,64,65,76,93,119,123,142,157,161,180,184,221,259
186	25	570	3,9,10,33,36,38,40,42,50,51,60,69,72,75,77,90,113,140,141,151,152,189,200,229,230
187	25	575	4,6,14,16,31,39,63,69,74,81,88,103,107,111,115,120,131,132,133,147,156,159,164,198,218
188	25	576	1,4,9,11,15,19,22,34,36,53,60,76,82,84,104,126,127,128,153,156,165,174,183,219,237
189	25	576	8,9,10,11,16,30,36,38,45,55,57,65,68,81,84,95,98,100,116,135,144,153,162,279,297
190	25	576	4,6,9,10,17,21,23,25,31,33,36,38,45,50,81,83,115,133,135,144,146,153,162,279,297
191	25	580	2,5,7,10,12,13,19,21,22,29,36,40.61,65,74,101,135,139,161,179,183,192,205,209,236
192	25	580	5,6,11,13,16,17,21,25,34,44,54,68,80,88,100,112,120,135,142,145,170,173,195,215,265
193	25	580	11,12,16,17,29,32,39,41,53,55,59,60,68,70,81,84,92,124,125,128,129,156,171,280,300
194	25	593	13,14,15,35,48,51,55,67,73,79,83,91,94,105,109,116,119,124,133,150,171,173,196,217,226
195	25	595	4,13,18,19,22,35,40,48,58,61,62,77,78,82,83,86,118,149,163,168,187,192,202,206,240
196	25	601	7,8,25,34,41,42,46,48,54,55,62,70,71,74,98,103,116,143,168,169,190,192,193,218,240
197	25	603	7,11,12,14,21,25,32,40,52,56,60.67,68,81,91,92,132,144,149,163,177,191,196,235,263
198	25	603	13,23,26,27,35,44,45,49,53,54,57,66,75,99,101,110,122,126,144,158,175,180,189,234,270
199	25	607	6,8,10,13,19,25,32,37,49,54,58,76,84,91,92,100,107,128,156,185,196,205,206,216,246
200	25	609	9,14,15,17,32,45,47,58,67,74,76,79,80,83,97,111,125,126,150,170,186,188,215,224,235
201	25	611	1,10,22,26,32,41,45,54,57,61,62,66,85,86,87,95,97,101,119,132,136,167,176,268,343
202	25	614	15,22,24,31,33,49,53,54,57,60,63,68,74,81,83,104,109,151,155,163,167,217,229,230,234
203	25	634	15,17,24,26,33,43,44,54,57,60,63,73,79,81,88,109,119,160,161,172,173,227,234,235,239
204	25	643	2,9,21,29,38,40,41,42,58,62,67,76,82,83,85,96,104,166,172,186,192,201,207,250.270
205	25	644	7,9,13,18,19,22,31,49,53,61,66,68,71,87,93,94,119,164,178,192,199,206,227,239,253
206	25	655	10,14,15,21,25,26,31,40,51,53,54,57,65,83,84,86,151,152,173,193,194,215,216,246,288
207	25	661	5,7,17,18,23,31,36,38,41,64,73,77,83,84,102,106,111,161,175,196,203,210.238,248,262

3.7.245 Statistics [2 CONS]

- DEVIATION,
- SPREAD.

A constraint representing a function in statistics usually used for obtaining a balanced assignment.

3.7.246 Strip packing [2 CONS]

- DIFFN,
- GEOST.

A constraint that can be used to model the *strip packing problem*: Given a set of rectangles pack them into an open ended strip of given width in order to minimise the total overall height. Borders of the rectangles to pack should be parallel to the borders of the strip and rectangles should not overlap. Some variants of strip packing allow to rotate rectangles from 90 degrees. Benchmarks with known optima can be obtained from Hopper's PhD thesis [229].

3.7.247 Strong articulation point [1 CONS]

- TREE.

A constraint for which the filtering algorithm uses the notion of *strong articulation point*. A *strong articulation point* of a strongly connected digraph G is a vertex such that if we remove it, G is broken into at least two strongly connected components. Figure 3.74 illustrates the notion of strong articulation point on the digraph depicted by part (A). The vertex labelled by 3 is a strong articulation point since its removal creates the three strongly connected components depicted by part (B) (i.e., the first, second and third strongly connected components correspond respectively to the sets of vertices $\{1, 4\}$, $\{2\}$ and $\{5\}$). From an algorithmic point of view, it was shown in [234] how to compute all the strong articulation points of a digraph G in linear time with respect to the number of arcs of G.

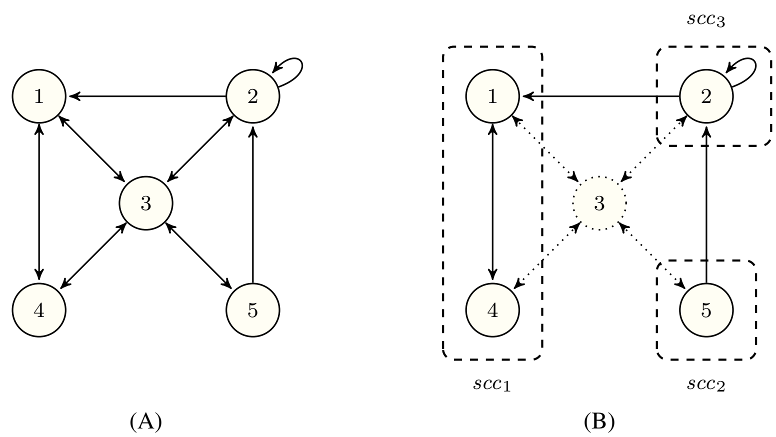


Figure 3.74: (A) A connected digraph, (B) its three strongly connected components scc_1, scc_2 and scc_3 when its unique strong articulation point (i.e., the vertex labelled by 3) is removed

3.7.248 Strong bridge

[2 CONS]

- CIRCUIT,
- CYCLE.

A constraint for which the filtering algorithm may use the notion of *strong bridge* (i.e., enforce arcs corresponding to strong bridges to be part of the solution in order to avoid creating too many strongly connected components). A *strong bridge* of a strongly connected digraph G is an arc such that, if we remove it, G is broken into at least two strongly connected components. Figure 3.75 illustrates the notion of strong bridge on the digraph depicted by part (A). The arc from the vertex labelled by 2 to the vertex labelled by 1 is a strong bridge since its removal creates the three strongly connected components depicted by part (B) (i.e., the first, second and third strongly connected components correspond respectively to the sets of vertices $\{1, 3, 4\}$, $\{2\}$ and $\{5\}$). The other strong bridges of the digraph depicted by part (A) are the arcs $1 \rightarrow 3$ and $5 \rightarrow 2$. From an algorithmic point of view, it was shown in [234] how to compute all the strong bridges of a digraph G in linear time with respect to the number of arcs of G.

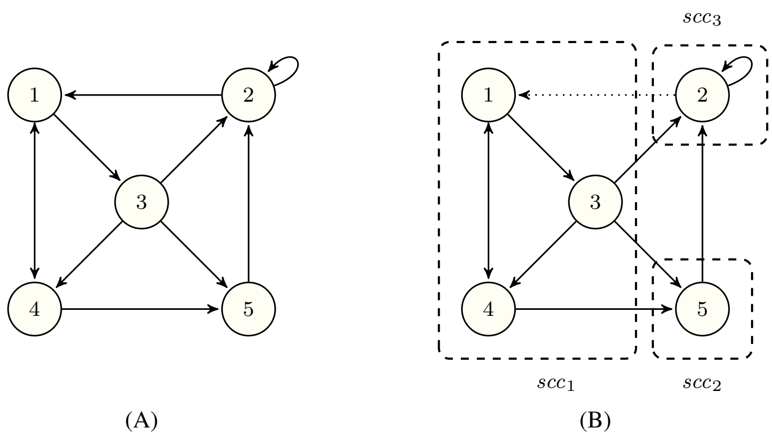


Figure 3.75: (A) A connected digraph, (B) its three strongly connected components scc_1, scc_2 and scc_3 when one of its strong bridge, the arc $2 \rightarrow 1$, is removed

3.7.249 Strongly connected component [25 CONS]

- ATLEAST_NVALUE,
- ATLEAST_NVECTOR,
- ATMOST_NVALUE,
- ATMOST_NVECTOR,
- BALANCE_CYCLE,
- CIRCUIT_CLUSTER,
- CONNECT_POINTS,
- CYCLE,
- CYCLE_OR_ACCESSIBILITY,
- CYCLE_RESOURCE,
- GROUP_SKIP_ISOLATED_ITEM,
- INCREASING_NVALUE,
- NCLASS,
- NEQUIVALENCE,
- NINTERVAL,
- NPAIR,
- NSET_OF_CONSECUTIVE_VALUES,
- NVALUE,
- NVALUES,
- NVALUES_EXCEPT_0,
- NVECTOR,
- NVECTORS,
- POLYOMINO,
- SOFT_ALLDIFFERENT_VAR,
- STRONGLY_CONNECTED.

Denotes that a constraint restricts the strongly connected components of its associated final graph. This is usually done by using a graph property like $\mathbf{MAX\_NSCC}$, $\mathbf{MIN\_NSCC}$ or $\mathbf{NSCC}$.

3.7.250 Subset sum [1 CONS]

- WEIGHTED_PARTIAL_ALLDIFF.

Denotes that, by reduction to *subset sum*, deciding whether a constraint has a solution or not was shown to be NP-hard. The *subset sum* problem can be described as follows: given a finite set of integers in $\mathbb{Z}^+$ and an integer s in $\mathbb{Z}^+$, does any subset sum equal exactly s?

3.7.251 Sudoku [2 CONS]

- ALLDIFFERENT,
- K_ALLDIFFERENT.

A constraint that can be used for modelling the Sudoku puzzle problem. A *Sudoku square* is an 9×9 array in which 9 distinct numbers in $[1, 9]$ are arranged so that the following two conditions hold:

- Each number occurs once in each row and column.
- The numbers in each major 3×3 block are distinct.

The *Sudoku puzzle problem* is to complete a partially filled board in order to get a Sudoku square. Part (A) of Figure 3.76 gives a partially filled Sudoku board, while part (B) provides its unique possible completion.

	2	6				8	1	
3			7		8			6
4				5				7
	5		1		7		9	
		3	9		5	1		
	4		3		2		5	
1				3				2
5			2		4			9
	3	8				4	6	

(A)

7	2	6	4	9	3	8	1	5
3	1	5	7	2	8	9	4	6
4	8	9	6	5	1	2	3	7
8	5	2	1	4	7	6	9	3
6	7	3	9	8	5	1	2	4
9	4	1	3	6	2	7	5	8
1	9	4	8	3	6	5	7	2
5	6	7	2	1	4	3	8	9
2	3	8	5	7	9	4	6	1

(B)

Figure 3.76: (A) A partially filled Sudoku square and (B) its unique completion

3.7.252 Sum **[13 CONS]**

- INCREASING_SUM,
- SCALAR_PRODUCT,
- SLIDING_SUM,
- SLIDING_TIME_WINDOW_SUM,
- SUM,
- SUM_CTR,
- SUM_OF_INCREMENTS,
- SUM_SET,
- SUM_CUBES_CTR,
- SUM_POWERS4_CTR,
- SUM_POWERS5_CTR,
- SUM_POWERS6_CTR,
- SUM_SQUARES_CTR.

A constraint involving one or several sums.

3.7.253 Sweep **[7 CONS]**

- CUMULATIVES,
- DIFFN,
- GEOST,
- GEOST_TIME,
- SOFT_ALL_EQUAL_MIN_VAR,
- SPREAD,
- VISIBLE.

A constraint for which the filtering algorithm may use a *sweep algorithm*. A *sweep algorithm* [333, pages 10–11] solves a problem by moving an imaginary object (usually a line, a plane or sometime a point). The object does not move continuously, but only at particular points where we actually do something. A sweep algorithm uses the following two data structures:

- A data structure called the *sweep status*, which contains information related to the current position of the object that moves,

- A data structure named the *event point series*, which holds the events to process.

The algorithm initialises the sweep status for the initial position of the imaginary object. Then the object jumps from one event to the next event; each event is handled by updating the status of the sweep.

A first typical application reported in [34] of the idea of sweep within the context of constraint programming is to aggregate several constraints that have two variables in common in order to perform more deduction. Let:

- $\mathtt{X}$ and $\mathtt{Y}$ be two distinct variables,

- $C_1(\mathtt{V}_{11}, \ldots, \mathtt{V}_{1n_1}), \ldots, C_m(\mathtt{V}_{m1}, \ldots, \mathtt{V}_{mn_m})$ be a set of m constraints such that all constraints mention $\mathtt{X}$ and $\mathtt{Y}$.

The sweep algorithm tries to adjust the minimum value of $\mathtt{X}$ with respect to the conjunction of the previous constraints by moving a sweep-line from the minimum value of $\mathtt{X}$ to its maximum value. It accumulates within the sweep-line status the values to be currently removed from the domain of $\mathtt{Y}$. If, for the current position Δ of the sweep-line, all values of $\mathtt{Y}$ have to be removed, then the algorithm removes value Δ from the domain of $\mathtt{X}$. The events to process correspond to the starts and ends of forbidden two-dimensional regions with respect to constraints $C_1, \ldots, C_m$ and variables $\mathtt{X}$ and $\mathtt{Y}$. Forbidden regions are a way to represent constraints $C_1, \ldots, C_m$ that is suited for this sweep algorithm. A forbidden region of the constraint C_i with respect to the variables $\mathtt{X}$ and $\mathtt{Y}$ is an ordered pair $([F_x^-, F_x^+], [F_y^-, F_y^+])$ of intervals such that: $\forall x \in [F_x^-, F_x^+], \forall y \in [F_y^-, F_y^+] : C_i(\mathtt{V}_{i1}, \ldots, \mathtt{V}_{in_i})$ has no solution in which $\mathtt{X} = x$ and $\mathtt{Y} = y$.

Figure 3.77 shows five constraints and their respective forbidden regions (in pink) with respect to two given variables $\mathtt{X}$ and $\mathtt{Y}$ and their domains. The first constraint requires that $\mathtt{X}$, $\mathtt{Y}$ and $\mathtt{R}$ be pairwise distinct. Constraints (B,C) are usual arithmetic constraints.[17] Constraint (D) can be interpreted as requiring that two rectangles of respective origins $(\mathtt{X}, \mathtt{Y})$ and $(\mathtt{T}, \mathtt{U})$ and sizes $(2, 4)$ and $(3, 2)$ do not overlap. Finally, constraint (E) is a parity constraint of the sum of $\mathtt{X}$ and $\mathtt{Y}$.

We illustrate the use of the sweep algorithm on a concrete example. Assume that we want to find out the minimum value of variable $\mathtt{X}$ with respect to the conjunction of the five constraints that were introduced by Figure 3.77, that is versus the following

[17] Within the context of continuous variables, Chabert*et al.* [115] shows how to compute a forbidden region that contains a given unfeasible point for numerical constraints with arbitrary mathematical expressions.

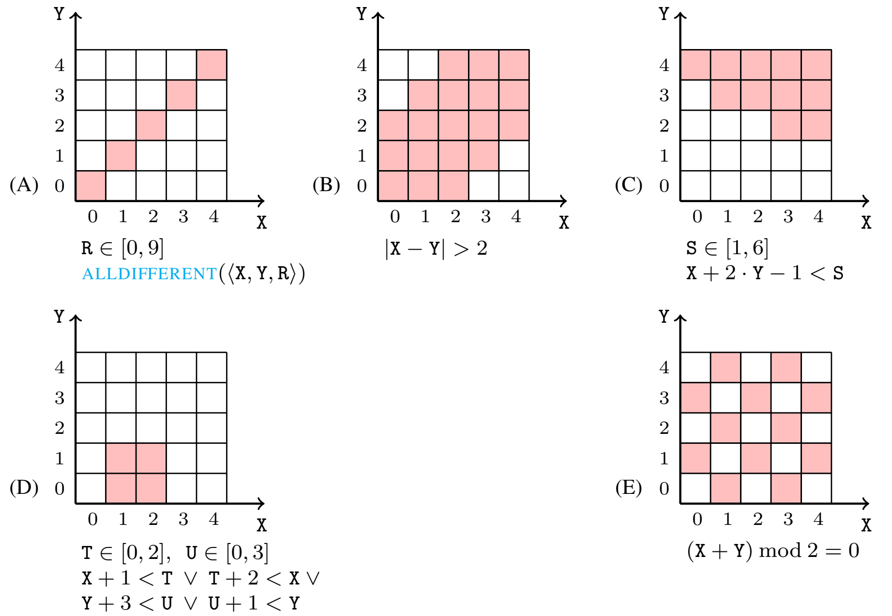


Figure 3.77: Examples of forbidden regions (in pink) according to the two variables X and Y ($\mathtt{X} \in [0, 4]$, $\mathtt{Y} \in [0, 4]$) for five constraints

conjunction of constraints:

$$\begin{cases} \mathtt{X} \in 0..4, \mathtt{Y} \in 0..4, \mathtt{R} \in 0..9, \mathtt{T} \in 0..2, \mathtt{U} \in 0..3 & \\ \text{ALLDIFFERENT}(\langle \mathtt{X}, \mathtt{Y}, \mathtt{R} \rangle) & (\text{A}) \\ |\mathtt{X} - \mathtt{Y}| > 2 & (\text{B}) \\ \mathtt{X} + 2\mathtt{Y} - 1 < \mathtt{S} & (\text{C}) \\ \mathtt{X} + 1 < \mathtt{T} \vee \mathtt{T} + 2 < \mathtt{X} \vee \mathtt{Y} + 3 < \mathtt{U} \vee \mathtt{U} + 1 < \mathtt{Y} & (\text{D}) \\ (\mathtt{X} + \mathtt{Y}) \bmod 2 = 2 & (\text{E}) \end{cases}$$

Figure 3.78 shows the content of the sweep-line status (i.e., the forbidden values for Y according the current position of the sweep-line) for different positions of the sweep-line. More precisely, the sweep-line status can be viewed as an array (see the rightmost part of Figure 3.78) which records for each possible value of Y the number of forbidden regions that currently intersect the sweep-line (see the leftmost part of Figure 3.78 where these forbidden regions are coloured in red). The smallest possible value of X is 4, since this is the first position of the sweep-line where the sweep-line status contains a value of Y which is not forbidden (i.e., $\mathtt{X} = 4$, $\mathtt{Y} = 0$ is not covered by any forbidden region).

A second similar application of the idea of sweep in the context of the *cardinality operator* [430], where all constraints have at least two variables in common, is reported in [33]. As before, each constraint C of the cardinality operator is defined by its forbidden regions with respect to a pair of variables $(\mathtt{X}, \mathtt{Y})$ that occur in every constraint. In addition to that, a constraint C is also defined by its *safe regions*, where a safe region

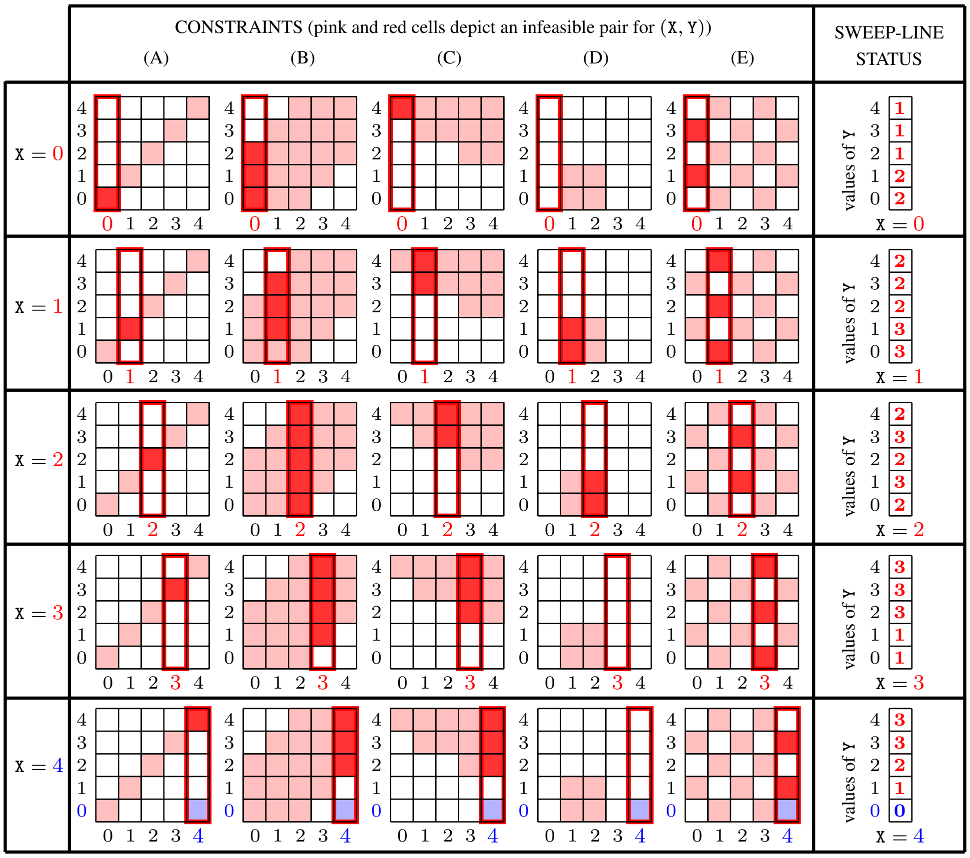


Figure 3.78: Sweep-line status while sweeping through the potential values of variable X (i.e., values from 0 to 4) until a potentially feasible point $\mathtt{X} = 4$, $\mathtt{Y} = 4$ wrt the five constraints (A), (B), (C), (D) and (E) is found; the sweep-line status (i.e., the rightmost column) records for a potential value v of variable X and for each potential value w of variable Y how many constraints are violated when both $\mathtt{X} = v$ and $\mathtt{Y} = w$.

is the set of assignments to the pair $(\mathtt{X}, \mathtt{Y})$ located in a rectangle such that the constraint always holds, no matter which values are taken by the other variables of C. Then the extended sweep algorithm filters the pair of variables $(\mathtt{X}, \mathtt{Y})$ right from the beginning according to the minimum and maximum number of constraints of the cardinality operator that have to hold.

A third typical application reported in [42] and in [107] of the idea of sweep within the context of multi-dimensional placement problems (for example, the DIFFN and the GEOST constraints) for filtering each coordinate of the origin of an object o to place is as follows. To adjust the minimum (respectively maximum) value of a coordinate of the origin we perform a recursive traversal of the placement space in increasing (respectively decreasing) lexicographic order and skips infeasible points that are located in a multi-dimensional forbidden set. Each multi-dimensional forbidden set is computed from a constraint where object o occurs (for example, a non-overlapping constraint in the context of the DIFFN and the GEOST constraints). Figure 3.80 illustrates the k-dimensional lexicographic sweep algorithm in the context of $k = 2$ and of five non-overlapping rectangles r_i ($i \in [1, 5]$) of respective sizes (w_i, h_i) where $w_1 = 2$, $h_1 = 1$, $w_2 = 3$, $h_2 = 1$, $w_3 = 1$, $h_3 = 1$, $w_4 = 1$, $h_4 = 3$, $w_5 = 5$, $h_5 = 4$. For each rectangle r_i we consider the coordinates of its lower leftmost corner with abscissa x_i and ordinate y_i such that $x_1 \in [1, 4]$, $y_1 \in [2, 4]$, $x_2 = 4$, $y_2 = 6$, $x_3 \in [2, 4]$, $y_3 \in [8, 9]$, $x_4 = 7$, $y_4 = 1$, $x_5 \in [1, 8]$ and $y_5 \in [1, 6] \cup [8, 8]$. We focus on the filtering of the minimum value of the abscissa of rectangle r_5 (i.e., variable x_5).

- Since rectangle r_5 should not overlap rectangles r_1, r_2, r_3 and r_4 we can compute for each non-overlapping constraint a set of forbidden points for the origin of r_5. Given two non-overlapping rectangles r_i and r_j of respective origins (x_i, y_i) and (x_j, y_j) and of respective sizes (w_i, h_i) and (w_j, h_j), there is one non-empty forbidden region for the origin of r_i of the form $([\overline{x_j} - w_i + 1, \underline{x_j} + w_j - 1], [\overline{y_j} - h_i + 1, \underline{y_j} + h_j - 1])$ when both $\overline{x_j} - w_i + 1 \leq \underline{x_j} + w_j - 1$ and $\overline{y_j} - h_i + 1 \leq \underline{y_j} + h_j - 1$ hold, and no forbidden region otherwise [34]. Figure 3.79 illustrates the four possible cases leading to a non-empty forbidden region.

  - Since r_5 should not overlap r_1 the points of the forbidden region $([4 - 5 + 1, 1 + 2 - 1], [4 - 4 + 1, 2 + 1 - 1])$, i.e. $([0, 2], [1, 2])$, are forbidden for the origin of r_5.
  - Since r_5 should not overlap r_2 the points of the forbidden region $([4 - 5 + 1, 4 + 3 - 1], [6 - 4 + 1, 6 + 1 - 1])$, i.e. $([0, 6], [3, 6])$, are forbidden for the origin of r_5.
  - Since r_5 should not overlap r_3 the points of the forbidden region $([4 - 5 + 1, 2 + 1 - 1], [9 - 4 + 1, 8 + 1 - 1])$, i.e. $([0, 2], [6, 8])$, are forbidden for the origin of r_5.
  - Since r_5 should not overlap r_4 the points of the forbidden region $([7 - 5 + 1, 7 + 1 - 1], [1 - 4 + 1, 1 + 3 - 1])$, i.e. $([3, 7], [-2, 3])$, are forbidden for the origin of r_5.

  Since $y_5 \neq 7$ the points of the forbidden region $([1, 8], [7, 7])$ are also forbidden for the origin of r_5.

- In Part (A) we represent the extreme positions of rectangles r_1 to r_4: for example, the leftmost lower corner of rectangle r_1 can only be fixed at positions $(1, 2)$, $(1, 3)$, $(1, 4)$, $(2, 2)$, $(2, 3)$, $(2, 4)$, $(3, 2)$, $(3, 3)$, $(3, 4)$, $(4, 2)$, $(4, 3)$ and $(4, 4)$ from which positions $(1, 2)$, $(1, 4)$, $(4, 2)$ and $(4, 4)$ are the extreme positions. Since we prune the abscissa of the coordinate of the origin of r_5 the abscissa and ordinate dimensions of the placement space correspond respectively to the outer and to the inner dimensions of the placement space: the sweep-point first moves along the inner dimension before jumping on the outer dimension.

- Part (B) represents the first step of the sweep-point algorithm. We start the traversal of the placement space at the extreme point $c = (1, 1)$, i.e. the extreme leftmost lower coordinate of the origin of rectangle r_5. Since the sweep-point belongs to the forbidden region associated with the non-overlapping constraint between r_5 and r_1, i.e. the red box with the dash border, we compute the first point outside the forbidden region along the inner sweep dimension. As a consequence the sweep-point moves to the next position $(1, 3)$.

- The process is repeated in Parts (C) to (H) until we finally find a position for the sweep-point that does not belong to any forbidden region derived from the constraints where rectangle r_5 is involved (i.e., point $(3, 8)$ in Part (I)).

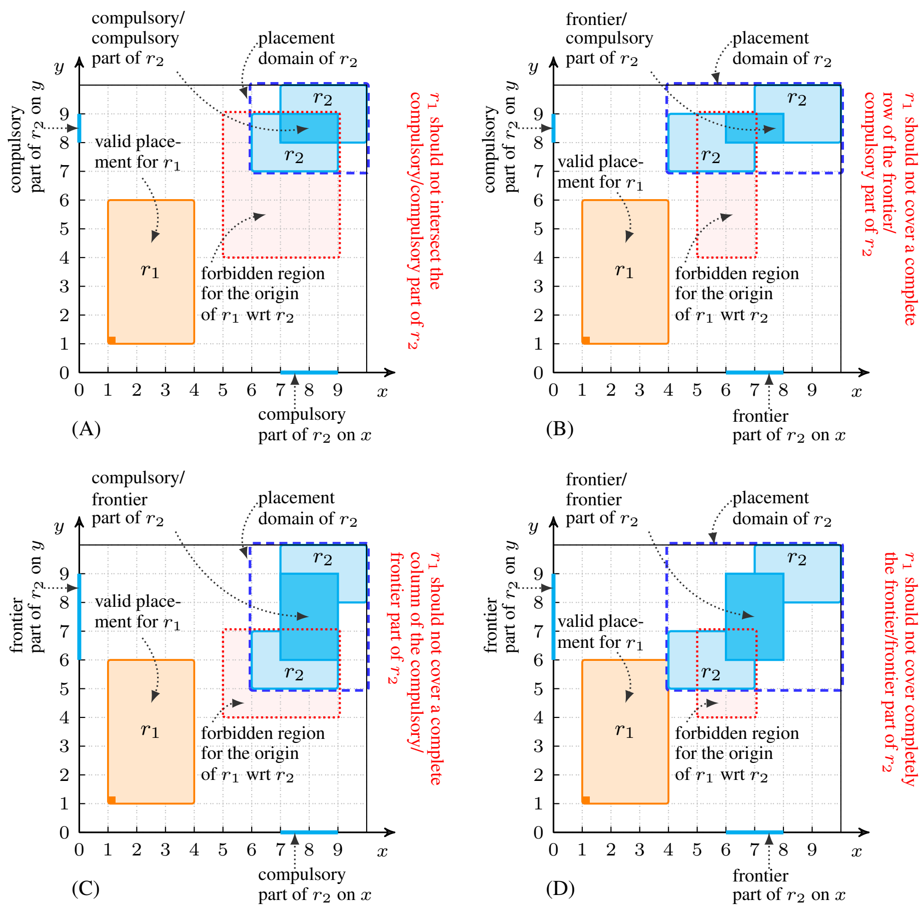


Figure 3.79: Given two rectangles r_1 and r_2 that should not overlap, interpretation of a non-empty forbidden region for the origin of r_1 wrt r_2: four cases (A), (B), (C) and (D) depending of the size of r_2 versus the size of placement domain of r_2; a point belongs to the forbidden region if and only if all compulsory parts of r_2 are intersected by r_1 (i.e., since all points of the compulsory part are occupied) and if all frontier parts of r_2 are completely covered by r_1 (i.e., since at least one point of the frontier part is occupied).

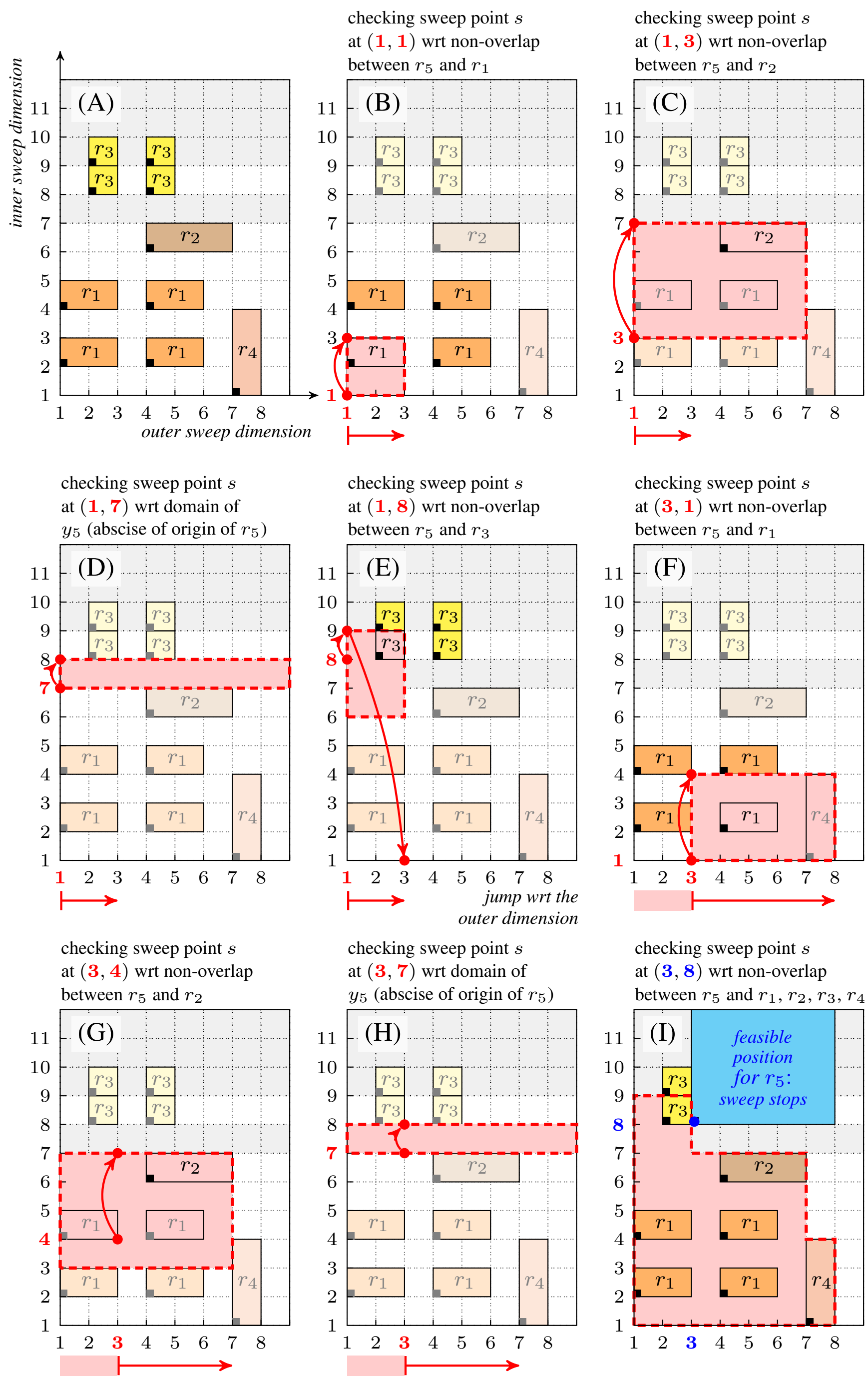


Figure 3.80: Illustration of the sweep point algorithm for adjusting the minimum value of the abscissa of rectangle r_5 (within the placement space, the grey area represents forbidden placement for the origin of r_5 due to the domain of y_5)

The worst case complexity of the k-dimensional lexicographic sweep algorithm depends on the maximum number of jumps of the sweep point. Figure 3.81 shows for $k = 2$ an example where the number of jumps turns out to be proportional to n^2 where n is the number of forbidden regions. We conjecture that we get a similar scheme, i.e. $O(n^k)$ maximum number of jumps, as k increases. However from a practical point of view, experiments have shown that we dont get such bad behavior for small values of k, i.e. up to $k = 7$, which turns out to be enough for many packing problems.

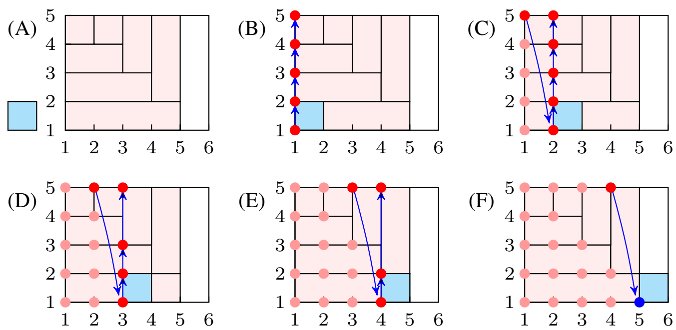


Figure 3.81: Illustration of a configuration in two dimensions leading to a quadratic number of jumps of the sweep-point: (A) the goal is to filter the minimum value abscissa of the origin of the cyan square so that it does not overlap all the seven pink rectangles; (B), (C), (D), (E) first, second, third and fourth steps for saturating the inner sweep dimension, (F) first feasible position of the sweep point of coordinates $(5, 1)$ in blue; note that the forbidden regions for the origin of the cyan square correspond exactly to the fixed pink rectangles since the cyan square has a size of 1.

3.7.254 Symmetric [11 CONS]

- ALL_DIFFER_FROM_AT_LEAST_K_POS,
- ALL_DIFFER_FROM_AT_MOST_K_POS,
- ALL_DIFFER_FROM_EXACTLY_K_POS,
- ALL_INCOMPARABLE,
- BIPARTITE,
- CLIQUE,
- CONNECT_POINTS,
- CONNECTED,
- INVERSE_WITHIN_RANGE,
- PROPER_FOREST,
- SYMMETRIC.

Denotes that a constraint is defined by a graph constraint for which the final graph is symmetric. A digraph is *symmetric* if and only if, if there is an arc from a vertex u to a vertex v, there is also an arc from v to u.

3.7.255 Symmetry [24 CONS]

- ALLPERM,
- INCREASING_GLOBAL_CARDINALITY,
- INCREASING_NVALUE,
- INCREASING_SUM,
- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- GEOST,
- LEX2,
- LEX_BETWEEN,
- LEX_CHAIN_GREATER,
- LEX_CHAIN_GREATEREQ,
- LEX_CHAIN_LESS,
- LEX_CHAIN_LESSEQ,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LEX_LESSEQ_ALLPERM,
- ORDERED_ATLEAST_NVECTOR,
- ORDERED_ATMOST_NVECTOR,
- ORDERED_NVECTOR,
- SET_VALUE_PRECEDE,
- STRICT_LEX2,
- SUBGRAPH_ISOMORPHISM.

A constraint that can be used for breaking certain types of symmetries (i.e., ALLPERM,INT_VALUE_PRECEDE,..., STRICT_LEX2) or for identifying certain symmetries (i.e., SUBGRAPH_ISOMORPHISM).

3.7.256 System of constraints [31 CONS]

- ALL_DIFFER_FROM_AT_LEAST_K_POS (system of DIFFER_FROM_AT_LEAST_K_POS),
- ALL_DIFFER_FROM_AT_MOST_K_POS (system of DIFFER_FROM_AT_MOST_K_POS),
- ALL_DIFFER_FROM_EXACTLY_K_POS (system of DIFFER_FROM_EXACTLY_K_POS),
- ALL_INCOMPARABLE (system of INCOMPARABLE),
- ALLDIFFERENT (system of NEQ),
- ALLPERM (system of LEX_LESSEQ_ALLPERM),
- AMONG_SEQ (system of AMONG_LOW_UP),
- COLORED_MATRIX (system of GLOBAL_CARDINALITY),
- ELEMENTS (system of ELEM or of ELEMENT sharing the same table),
- ELEMENTS_SPARSE (system of ELEMENT_SPARSE sharing the same table),
- GLOBAL_CARDINALITY (system of AMONG),
- K_ALLDIFFERENT (system of ALLDIFFERENT),
- K_DISJOINT (system of DISJOINT),
- K_SAME (system of SAME),
- K_SAME_INTERVAL (system of SAME_INTERVAL),
- K_SAME_MODULO (system of SAME_MODULO),
- K_SAME_PARTITION (system of SAME_PARTITION),
- K_USED_BY (system of USED_BY),
- K_USED_BY_INTERVAL (system of USED_BY_INTERVAL),
- K_USED_BY_MODULO (system of USED_BY_MODULO),
- K_USED_BY_PARTITION (system of USED_BY_PARTITION),
- LEX2 (system of LEX_CHAIN_LESSEQ),
- LEX_BETWEEN (system of LEX_LESSEQ),
- LEX_CHAIN_GREATER (system of LEX_GREATER),
- LEX_CHAIN_GREATEREQ (system of LEX_GREATEREQ),
- LEX_CHAIN_LESSEQ (system of LEX_LESSEQ),
- LEX_CHAIN_LESS (system of LEX_LESS),
- LEX_ALLDIFFERENT (system of LEX_DIFFERENT),
- SLIDING_DISTRIBUTION (system of GLOBAL_CARDINALITY_LOW_UP),
- SLIDING_SUM (system of SUM_CTR),
- STRICT_LEX2 (system of LEX_CHAIN_LESS).

Denotes that a constraint is defined as the conjunction of several identical global constraints that have some variables in common.

3.7.257 Table [15 CONS]

- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENTN,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_PRODUCT,
- ELEMENT_SPARSE,
- ELEMENTS,
- ELEMENTS_ALLDIFFERENT,
- ELEMENTS_SPARSE,
- ITH_POS_DIFFERENT_FROM_0,
- NEXT_ELEMENT,
- NEXT_GREATER_ELEMENT,
- STAGE_ELEMENT.

A constraint that allows for representing the access to an element of a table.

3.7.258 Temporal constraint [17 CONS]

- CALENDAR,
- COLOURED_CUMULATIVE,
- COLOURED_CUMULATIVES,
- CUMULATIVE,
- CUMULATIVE_CONVEX,
- CUMULATIVE_PRODUCT,
- CUMULATIVE_WITH_LEVEL_OF_PRIORITY,
- CUMULATIVES,
- DISJOINT_TASKS,
- INTERVAL_AND_COUNT,
- INTERVAL_AND_SUM,
- SHIFT,
- SLIDING_TIME_WINDOW,
- SLIDING_TIME_WINDOW_FROM_START,
- SLIDING_TIME_WINDOW_SUM,
- SOFT_CUMULATIVE,
- TRACK.

A constraint involving the notion of time.

3.7.259 Ternary constraint [5 CONS]

- DISTANCE,
- ELEMENT_MATRIX,
- GCD,
- POWER,
- REMAINDER.

A constraint involving only three variables.

3.7.260 Timetabling constraint [32 CONS]

- CHANGE,
- CHANGE_CONTINUITY,
- CHANGE_PAIR,
- CHANGE_PARTITION,
- CIRCULAR_CHANGE,
- COLORED_MATRIX,
- CUMULATIVES,
- CYCLIC_CHANGE,
- CYCLIC_CHANGE_JOKER,
- DIFFN,
- FULL_GROUP,
- GEOST,
- GEOST_TIME,
- GROUP,
- GROUP_SKIP_ISOLATED_ITEM,
- INTERVAL_AND_COUNT,
- INTERVAL_AND_SUM,
- LONGEST_CHANGE,
- PATTERN,
- PERIOD,
- PERIOD_EXCEPT_0,
- SHIFT,
- SLIDING_CARD_SKIP0,
- SMOOTH,
- STRETCH_CIRCUIT,
- STRETCH_PATH,
- STRETCH_PATH_PARTITION,
- SYMMETRIC_ALLDIFFERENT,
- SYMMETRIC_ALLDIFFERENT_EXCEPT_0,
- SYMMETRIC_CARDINALITY,
- SYMMETRIC_GCC,
- TRACK.

A constraint that can occur in timetabling problems.

3.7.261 Time window [1 CONS]

- SLIDING_TIME_WINDOW_SUM.

A constraint involving one or several date ranges.

3.7.262 Touch [2 CONS]

- ORTHS_ARE_CONNECTED,
- TWO_ORTH_ARE_IN_CONTACT.

A constraint enforcing that some orthotopes touch each other (see **Contact**).

3.7.263 Tree [9 CONS]

- BALANCE_PATH,
- BALANCE_TREE,
- BINARY_TREE,
- PATH,
- PROPER_FOREST,
- STABLE_COMPATIBILITY,
- TREE,
- TREE_RANGE,
- TREE_RESOURCE.

According to the context, the keyword **tree** has the following meaning:

- In the context of a *digraph*, a constraint that partitions the vertices of a given initial digraph and that keeps a single successor for each vertex so that each partition corresponds to one tree. Each vertex points to its father or to itself if it corresponds to the root of a tree.

- In the context of an *undirected graph* a constraint that partitions the vertices of a given initial undirected graph in a set of connected components with no cycles.

3.7.264 Tuple [2 CONS]

- IN_RELATION,
- VEC_EQ_TUPLE.

A constraint involving a *tuple*. A *tuple* is an element of a *relation*, where a *relation* is a subset of the product of several finite sets.

3.7.265 Two-dimensional orthogonal packing [2 CONS]

- DIFFN,
- GEOST.

A constraint that can be used to model the *two-dimensional orthogonal packing problem*. Given a set of rectangles pack them into a rectangular placement space. Borders of the rectangles should be parallel to the borders of the placement space and rectangles should not overlap. Some variants of strip packing allow to rotate rectangles from 90 degrees. Benchmarks can be obtained from a generator described in the following paper [126].

3.7.266 Unary constraint [5 CONS]

- IN,
- IN_INTERVAL,
- IN_INTERVALS,
- NOT_IN,
- SUM_FREE.

A constraint involving only one variable.

3.7.267 Undirected graph [2 CONS]

- PROPER_FOREST,
- TOUR.

A constraint that deals with an *undirected graph*. An *undirected graph* is a graph whose edges consist of unordered pairs of vertices.

3.7.268 Value constraint [80 CONS]

- ALL_BALANCE,
- ALL_EQUAL,
- ALL_EQUAL_EXCEPT_0,
- ALL_MIN_DIST,
- ALLDIFFERENT,
- ALLDIFFERENT_CST,
- ALLDIFFERENT_CONSECUTIVE_VALUES,
- ALLDIFFERENT_EXCEPT_0,
- ALLDIFFERENT_INTERVAL,
- ALLDIFFERENT_MODULO,
- ALLDIFFERENT_ON_INTERSECTION,
- ALLDIFFERENT_PARTITION,
- AMONG,
- AMONG_DIFF_0,
- AMONG_INTERVAL,
- AMONG_LOW_UP,
- AMONG_MODULO,
- ARITH,
- ARITH_OR,
- ATLEAST,
- ATMOST,

- BALANCE,
- BALANCE_INTERVAL,
- BALANCE_MODULO,
- BALANCE_PARTITION,
- CARDINALITY_ATLEAST,
- CARDINALITY_ATMOST,
- CARDINALITY_ATMOST_PARTITION,
- CONSECUTIVE_VALUES,
- COUNT,
- COUNTS,
- DIFFER_FROM_AT_LEAST_K_POS,
- DIFFER_FROM_AT_MOST_K_POS,
- DIFFER_FROM_EXACTLY_K_POS,
- DISCREPANCY,
- DISJOINT,
- DOMAIN,
- EXACTLY,
- GLOBAL_CARDINALITY,
- GLOBAL_CARDINALITY_LOW_UP,
- GLOBAL_CARDINALITY_LOW_UP_NO_LOOP,
- GLOBAL_CARDINALITY_NO_LOOP,
- IN,
- IN_INTERVAL,
- IN_INTERVAL_REIFIED,
- IN_INTERVALS,
- IN_SAME_PARTITION,
- IN_SET,
- INCREASING_GLOBAL_CARDINALITY,
- K_ALLDIFFERENT,
- K_DISJOINT,
- LENGTH_FIRST_SEQUENCE,
- LENGTH_LAST_SEQUENCE,
- LINK_SET_TO_BOOLEANS,
- MAX_NVALUE,
- MAX_SIZE_SET_OF_CONSECUTIVE_VAR,
- MIN_NVALUE,
- MIN_SIZE_SET_OF_CONSECUTIVE_VAR,
- MULTI_INTER_DISTANCE,
- NOT_ALL_EQUAL,
- NOT_IN,
- NSET_OF_CONSECUTIVE_VALUES,
- OPEN_ALLDIFFERENT,
- OPEN_AMONG,
- OPEN_ATLEAST,
- OPEN_ATMOST,
- OPEN_GLOBAL_CARDINALITY,
- OPEN_GLOBAL_CARDINALITY_LOW_UP,
- ORDERED_GLOBAL_CARDINALITY,
- PERMUTATION,
- ROOTS,
- SAME_AND_GLOBAL_CARDINALITY,
- SAME_AND_GLOBAL_CARDINALITY_LOW_UP,
- SOFT_ALLDIFFERENT_CTR,
- SOFT_ALLDIFFERENT_VAR,
- SOFT_ALL_EQUAL_MAX_VAR,
- SOFT_ALL_EQUAL_MIN_CTR,
- SOFT_ALL_EQUAL_MIN_VAR,
- SOME_EQUAL,
- VEC_EQ_TUPLE.

A constraint that puts a restriction on how values can be assigned to usually one or several collections of variables, or possibly one or two variables. These variables usually correspond to domain variables but can sometimes be set variables.

3.7.269 Value partitioning constraint [14 CONS]

- ATLEAST_NVALUE,
- ATLEAST_NVECTOR,
- ATMOST_NVALUE,
- ATMOST_NVECTOR,
- INCREASING_NVALUE,
- NCLASS,
- NEQUIVALENCE,
- NINTERVAL,
- NPAIR,
- NVALUE,
- NVALUES,
- NVALUES_EXCEPT_0,
- NVECTOR,
- NVECTORS.

A constraint involving a partitioning of values in its definition.

3.7.270 Value precedence [3 CONS]

- INT_VALUE_PRECEDE,
- INT_VALUE_PRECEDE_CHAIN,
- SET_VALUE_PRECEDE.

A constraint that allows for expressing symmetries between values that are assigned to variables: given a solution, values can be uniformly interchanged in order to obtain a new solution. This is the case, for example, in graph colouring problems or in bin-packing problems when all bins are identical (e.g., see the BIN_PACKING constraint):

- For graph coloring problems, a variable and a value correspond respectively to a vertex, and to a colour used for coloring a vertex.
- For bin packing problems, a variable and a value correspond respectively to an item to pack, and to the bin where the item is assigned.

To break such symmetry we can order the values in such a way that the first occurrence of the i^{th} value occurs before the first occurrence of the $(i+1)^{th}$ value.

3.7.271 Variable-based violation measure [11 CONS]

- SOFT_ALLDIFFERENT_VAR,
- SOFT_ALL_EQUAL_MAX_VAR,
- SOFT_ALL_EQUAL_MIN_VAR,
- SOFT_SAME_INTERVAL_VAR,
- SOFT_SAME_MODULO_VAR,
- SOFT_SAME_PARTITION_VAR,
- SOFT_SAME_VAR,
- SOFT_USED_BY_INTERVAL_VAR,
- SOFT_USED_BY_MODULO_VAR,
- SOFT_USED_BY_PARTITION_VAR,
- SOFT_USED_BY_VAR.

A soft constraint for which the violation cost is the minimum number of variables to assign differently in order to get back to a solution.

3.7.272 Variable indexing [7 CONS]

- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_SPARSE,
- INDEXED_SUM.

A constraint where one or several variables are used as an index into an array.

3.7.273 Variable subscript [7 CONS]

- ELEM,
- ELEM_FROM_TO,
- ELEMENT,
- ELEMENT_GREATEREQ,
- ELEMENT_LESSEQ,
- ELEMENT_PRODUCT,
- INDEXED_SUM.

A constraint that can be used to model one or several variables that have a variable subscript.

3.7.274 Vector [40 CONS]

- ALL_DIFFER_FROM_AT_LEAST_K_POS,
- ALL_DIFFER_FROM_EXACTLY_K_POS,
- ALL_INCOMPARABLE,
- ALLPERM,
- ATLEAST_NVECTOR,
- ATMOST_NVECTOR,
- CHANGE_VECTORS,
- COND_LEX_COST,
- COND_LEX_GREATER,
- COND_LEX_GREATEREQ,
- COND_LEX_LESS,
- COND_LEX_LESSEQ,
- DIFFER_FROM_AT_LEAST_K_POS,
- DIFFER_FROM_AT_MOST_K_POS,
- DIFFER_FROM_EXACTLY_K_POS,
- INCOMPARABLE,
- LEX_ALLDIFFERENT,
- LEX_ALLDIFFERENT_EXCEPT_0,
- LEX_BETWEEN,
- LEX_CHAIN_GREATER,
- LEX_CHAIN_GREATEREQ,
- LEX_CHAIN_LESS,
- LEX_CHAIN_LESSEQ,
- LEX_DIFFERENT,
- LEX_EQUAL,
- LEX_GREATER,
- LEX_GREATEREQ,
- LEX_LESS,
- LEX_LESSEQ,
- LEX_LESSEQ_ALLPERM,
- MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES,
- MAX_OCC_OF_SORTED_TUPLES_OF_VALUES,
- MAX_OCC_OF_TUPLES_OF_VALUES,
- NVECTOR,
- NVECTORS,
- ORDERED_ATLEAST_NVECTOR,
- ORDERED_ATMOST_NVECTOR,
- ORDERED_NVECTOR,
- PERIOD_VECTORS,
- ZERO_OR_NOT_ZERO_VECTORS.

Denotes that one (or more) argument of a constraint corresponds to a collection of vectors that all have the same number of components.

3.7.275 Vpartition [1 CONS]

- GROUP.

Denotes that a constraint is defined by two graph constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ such that:

- The two graph constraints have the same initial graph G_i,
- Each vertex of the initial graph G_i belongs to exactly one of the final graphs associated with $\mathcal{C}_1$ and $\mathcal{C}_2$.

3.7.276 Weighted assignment [4 CONS]

- GLOBAL_CARDINALITY_WITH_COSTS,
- MINIMUM_WEIGHT_ALLDIFFERENT,
- SUM_OF_WEIGHTS_OF_DISTINCT_VALUES,
- WEIGHTED_PARTIAL_ALLDIFF.

A constraint expressing an assignment problem such that a cost can be computed from each solution.

3.7.277 Workload covering [1 CONS]

- CUMULATIVES.

A constraint that can be used for modelling problems where a first set of tasks $\mathcal{T}_1$ has to cover a second set of tasks $\mathcal{T}_2$. Each task of $\mathcal{T}_1$ and $\mathcal{T}_2$ is defined by an origin, a duration and a height. At each point in time t the sum of the heights of the tasks of the first set $\mathcal{T}_1$ that overlap t has to be greater than or equal to the sum of the heights of the tasks of the second set $\mathcal{T}_2$ that also overlap t.

3.7.278 Zebra puzzle [4 CONS]

- ALLDIFFERENT,
- ELEM,
- ELEMENT,
- INVERSE.

A constraint that can be used for modelling the *zebra puzzle* problem. Here is the first known publication of that puzzle quoted in italic from Life International, December 17, 1962:

1. *There are five houses.*
2. *The Englishman lives in the red house.*
3. *The Spaniard owns the dog.*
4. *Coffee is drunk in the green house.*
5. *The Ukrainian drinks tea.*
6. *The green house is immediately to the right of the ivory house.*
7. *The Old Gold smoker owns snails.*
8. *Kools are smoked in the yellow house.*
9. *Milk is drunk in the middle house.*
10. *The Norwegian lives in the first house.*
11. *The man who smokes Chesterfields lives in the house next to the man with the fox.*
12. *Kools are smoked in the house next to the house where the horse is kept.*
13. *The Lucky Strike smoker drinks orange juice.*
14. *The Japanese smokes Parliaments.*
15. *The Norwegian lives next to the blue house.*

Now, who drinks water? Who owns the zebra?
In the interest of clarity, it must be added that each of the five houses is painted a different colour, and their inhabitants are of different national extractions, own different pets, drink different beverages and smoke different brands of American cigarettes. In statement 6*, right refers to the reader's right.*

A first model involves ELEMENT constraints with variables in their tables (i.e., the table of an ELEMENT constraint corresponds to its second argument). It consists of creating for each house i ($1 \leq i \leq 5$) five variables C_i, N_i, A_i, D_i, B_i respectively corresponding to the *colour* of house i, the *nationality* of the person leaving in house i, the preferred *pet* of the person leaving in house i, the preferred *beverage* of the person leaving in house i, the preferred *brand of American cigarettes* of the person leaving in house i. We first state the following five ALLDIFFERENT constraints on these variables for expressing that colours, nationalities, pets, beverages, and brands of American cigarettes are distinct:

- ALLDIFFERENT$(\langle C_1, C_2, C_3, C_4, C_5\rangle)$,
- ALLDIFFERENT$(\langle N_1, N_2, N_3, N_4, N_5\rangle)$,
- ALLDIFFERENT$(\langle A_1, A_2, A_3, A_4, A_5\rangle)$,
- ALLDIFFERENT$(\langle D_1, D_2, D_3, D_4, D_5\rangle)$,
- ALLDIFFERENT$(\langle B_1, B_2, B_3, B_4, B_5\rangle)$.

Now note that most statements link two specific attributes (e.g., *The Englishman lives in the red house*). Consequently, in order to ease the encoding of such statements in term of constraints, we will first create for each attribute a variable that indicates the house where an attribute occurs. For example, for the statement *The Englishman lives in the red house* we will create two variables which respectively indicate in which house the Englishman lives and which house is red. We now create all the variables attached to each class of attributes.

For each possible colour $c \in \{red, green, ivory, yellow, blue\}$ we create a variable I_c that corresponds to the index of the house having this colour. For each variable I_c, an ELEMENT constraint links it to the variables C_1, C_2, C_3, C_4, C_5 giving the colour of each house:

- $Red = 1$, $Green = 2$, $Ivory = 3$, $Yellow = 4$, $Blue = 5$,
- ELEMENT$(I_{red}, \langle C_1, C_2, C_3, C_4, C_5\rangle, Red)$,
- ELEMENT$(I_{green}, \langle C_1, C_2, C_3, C_4, C_5\rangle, Green)$,
- ELEMENT$(I_{ivory}, \langle C_1, C_2, C_3, C_4, C_5\rangle, Ivory)$,
- ELEMENT$(I_{yellow}, \langle C_1, C_2, C_3, C_4, C_5\rangle, Yellow)$,
- ELEMENT$(I_{blue}, \langle C_1, C_2, C_3, C_4, C_5\rangle, Blue)$.

Note that we can replace the five previous ELEMENT constraints by the following INVERSE constraint:

- INVERSE $\left(\left\langle \begin{array}{lll} \texttt{index}-1 & \texttt{succ}-C_1 & \texttt{pred}-I_{red}, \\ \texttt{index}-2 & \texttt{succ}-C_2 & \texttt{pred}-I_{green}, \\ \texttt{index}-3 & \texttt{succ}-C_3 & \texttt{pred}-I_{ivory}, \\ \texttt{index}-4 & \texttt{succ}-C_4 & \texttt{pred}-I_{yellow}, \\ \texttt{index}-5 & \texttt{succ}-C_5 & \texttt{pred}-I_{blue} \end{array} \right\rangle\right)$

For each possible nationality $n \in \{englishman, spaniard, ukrainian, norwegian, japanese\}$ we create a variable I_n that corresponds to the index of the house where the person with this nationality lives. For each variable I_n, an ELEMENT constraint links it to the variables N_1, N_2, N_3, N_4, N_5 giving the nationality associated with each house:

- $Englishman = 1$, $Spaniard = 2$, $Ukrainian = 3$, $Norwegian = 4$, $Japanese = 5$,
- ELEMENT$(I_{englishman}, \langle N_1, N_2, N_3, N_4, N_5\rangle, Englishman)$,
- ELEMENT$(I_{spaniard}, \langle N_1, N_2, N_3, N_4, N_5\rangle, Spaniard)$,
- ELEMENT$(I_{ukrainian}, \langle N_1, N_2, N_3, N_4, N_5\rangle, Ukrainian)$,
- ELEMENT$(I_{norwegian}, \langle N_1, N_2, N_3, N_4, N_5\rangle, Norwegian)$,
- ELEMENT$(I_{japanese}, \langle N_1, N_2, N_3, N_4, N_5\rangle, Japanese)$.

Again we can replace the five previous ELEMENT constraints by the following INVERSE constraint:

- INVERSE $\left(\left\langle \begin{array}{lll} \texttt{index}-1 & \texttt{succ}-N_1 & \texttt{pred}-I_{englishman}, \\ \texttt{index}-2 & \texttt{succ}-N_2 & \texttt{pred}-I_{spaniard}, \\ \texttt{index}-3 & \texttt{succ}-N_3 & \texttt{pred}-I_{ukrainian}, \\ \texttt{index}-4 & \texttt{succ}-N_4 & \texttt{pred}-I_{norwegian}, \\ \texttt{index}-5 & \texttt{succ}-N_5 & \texttt{pred}-I_{japanese} \end{array} \right\rangle\right)$

For each possible preferred pet $a \in \{dog, snail, fox, horse, zebra\}$ we create a variable I_a that corresponds to the index of the house where the person that prefers this pet lives. For each variable I_a, an ELEMENT constraint links it to the variables A_1, A_2, A_3, A_4, A_5 giving the preferred pet of each house:

- $Dog = 1$, $Snail = 2$, $Fox = 3$, $Horse = 4$, $Zebra = 5$,
- ELEMENT$(I_{dog}, \langle A_1, A_2, A_3, A_4, A_5\rangle, Dog)$,
- ELEMENT$(I_{snail}, \langle A_1, A_2, A_3, A_4, A_5\rangle, Snail)$,
- ELEMENT$(I_{fox}, \langle A_1, A_2, A_3, A_4, A_5\rangle, Fox)$,
- ELEMENT$(I_{horse}, \langle A_1, A_2, A_3, A_4, A_5\rangle, Horse)$,
- ELEMENT$(I_{zebra}, \langle A_1, A_2, A_3, A_4, A_5\rangle, Zebra)$.

Again we can replace the five previous ELEMENT constraints by the following INVERSE constraint:

- INVERSE $\left(\left\langle \begin{array}{lll} \texttt{index}-1 & \texttt{succ}-A_1 & \texttt{pred}-I_{dog}, \\ \texttt{index}-2 & \texttt{succ}-A_2 & \texttt{pred}-I_{snail}, \\ \texttt{index}-3 & \texttt{succ}-A_3 & \texttt{pred}-I_{fox}, \\ \texttt{index}-4 & \texttt{succ}-A_4 & \texttt{pred}-I_{horse}, \\ \texttt{index}-5 & \texttt{succ}-A_5 & \texttt{pred}-I_{zebra} \end{array} \right\rangle\right)$

For each possible preferred beverage $d \in \{coffee, tea, milk, orange\_juice, water\}$ we create a variable I_d that corresponds to the index of the house where the person that prefers this beverage lives. For each variable I_d, an ELEMENT constraint links it to the variables D_1, D_2, D_3, D_4, D_5 giving the preferred beverage of each house:

- $\mathit{Coffee} = 1$, $\mathit{Tea} = 2$, $\mathit{Milk} = 3$, $\mathit{Orange\_juice} = 4$, $\mathit{Water} = 5$,
- ELEMENT$(I_{\mathit{coffee}}, \langle D_1, D_2, D_3, D_4, D_5\rangle, \mathit{Coffee})$,
- ELEMENT$(I_{\mathit{tea}}, \langle D_1, D_2, D_3, D_4, D_5\rangle, \mathit{Tea})$,
- ELEMENT$(I_{\mathit{milk}}, \langle D_1, D_2, D_3, D_4, D_5\rangle, \mathit{Milk})$,
- ELEMENT$(I_{\mathit{orange_juice}}, \langle D_1, D_2, D_3, D_4, D_5\rangle, \mathit{Orange_juice})$,
- ELEMENT$(I_{\mathit{water}}, \langle D_1, D_2, D_3, D_4, D_5\rangle, \mathit{Water})$.

Again we can replace the five previous ELEMENT constraints by the following INVERSE constraint:

- INVERSE $\left(\left\langle \begin{array}{lll} \texttt{index} - 1 & \texttt{succ} - D_1 & \texttt{pred} - I_{\mathit{coffee}}, \\ \texttt{index} - 2 & \texttt{succ} - D_2 & \texttt{pred} - I_{\mathit{tea}}, \\ \texttt{index} - 3 & \texttt{succ} - D_3 & \texttt{pred} - I_{\mathit{milk}}, \\ \texttt{index} - 4 & \texttt{succ} - D_4 & \texttt{pred} - I_{\mathit{orange\_juice}}, \\ \texttt{index} - 5 & \texttt{succ} - D_5 & \texttt{pred} - I_{\mathit{water}} \end{array} \right\rangle\right)$

For each possible preferred brand of American cigarettes $b \in \{\mathit{old\_gold}, \mathit{kool}, \mathit{chesterfield}, \mathit{lucky\_strike}, \mathit{parliament}\}$ we create a variable I_b that corresponds to the index of the house where the person that prefers this brand lives. For each variable I_b, an ELEMENT constraint links it to the variables B_1, B_2, B_3, B_4, B_5 giving the preferred brand of American cigarettes of each house:

- $\mathit{Old\_gold} = 1$, $\mathit{Kool} = 2$, $\mathit{Chesterfield} = 3$, $\mathit{Lucky\_strike} = 4$, $\mathit{Parliament} = 5$,
- ELEMENT$(I_{\mathit{old_gold}}, \langle B_1, B_2, B_3, B_4, B_5\rangle, \mathit{Old_gold})$,
- ELEMENT$(I_{\mathit{kool}}, \langle B_1, B_2, B_3, B_4, B_5\rangle, Kool)$,
- ELEMENT$(I_{\mathit{chesterfield}}, \langle B_1, B_2, B_3, B_4, B_5\rangle, Chesterfield)$,
- ELEMENT$(I_{\mathit{lucky_strike}}, \langle B_1, B_2, B_3, B_4, B_5\rangle, Lucky_strike)$,
- ELEMENT$(I_{\mathit{parliament}}, \langle B_1, B_2, B_3, B_4, B_5\rangle, Parliament)$.

Again we can replace the five previous ELEMENT constraints by the following INVERSE constraint:

- INVERSE $\left(\left\langle \begin{array}{lll} \texttt{index} - 1 & \texttt{succ} - B_1 & \texttt{pred} - I_{\mathit{old\_gold}}, \\ \texttt{index} - 2 & \texttt{succ} - B_2 & \texttt{pred} - I_{\mathit{kool}}, \\ \texttt{index} - 3 & \texttt{succ} - B_3 & \texttt{pred} - I_{\mathit{chesterfield}}, \\ \texttt{index} - 4 & \texttt{succ} - B_4 & \texttt{pred} - I_{\mathit{lucky\_strike}}, \\ \texttt{index} - 5 & \texttt{succ} - B_5 & \texttt{pred} - I_{\mathit{parliament}} \end{array} \right\rangle\right)$

Finally we state one constraint for each statement from 2 to 15:

- $I_{\mathit{englishman}} = I_{\mathit{red}}$ (the Englishman lives in the red house).

- $I_{spaniard} = I_{dog}$ (the Spaniard owns the dog).
- $I_{coffee} = I_{green}$ (coffee is drunk in the green house).
- $I_{ukrainian} = I_{tea}$ (the Ukrainian drinks tea).
- $I_{green} = I_{ivory} + 1$ (the green house is immediately to the right of the ivory house).
- $I_{old\_gold} = I_{snail}$ (the Old Gold smoker owns snails).
- $I_{kool} = I_{yellow}$ (kools are smoked in the yellow house).
- $I_{milk} = 3$ (milk is drunk in the middle house).
- $I_{norwegian} = 1$ (the Norwegian lives in the first house).
- $|I_{chesterfield} - I_{fox}| = 1$ (the man who smokes Chesterfields lives in the house next to the man with the fox).
- $|I_{kool} - I_{horse}| = 1$ (kools are smoked in the house next to the house where the horse is kept).
- $I_{lucky\_strike} = I_{orange\_juice}$ (the Lucky Strike smoker drinks orange juice).
- $I_{japanese} = I_{parliament}$ (the Japanese smokes Parliaments).
- $|I_{norwegian} - I_{blue}| = 1$ (the Norwegian lives next to the blue house).

Now note that variables C_i, N_i, A_i, D_i, B_i $(1 \leq i \leq 5)$ do not occur at all within the constraints encoding statements 2 to 15. Consequently they can be removed, as long as we replace the five ALLDIFFERENT constraints on these variables by the following ALLDIFFERENT constraints:

- ALLDIFFERENT$(\langle I_{red}, I_{green}, I_{ivory}, I_{yellow}, I_{blue}\rangle)$,
- ALLDIFFERENT$(\langle I_{englishman}, I_{spaniard}, I_{ukrainian}, I_{norwegian}, I_{japanese}\rangle)$,
- ALLDIFFERENT$(\langle I_{dog}, I_{snail}, I_{fox}, I_{horse}, I_{zebra}\rangle)$,
- ALLDIFFERENT$(\langle I_{coffee}, I_{tea}, I_{milk}, I_{orange_juice}, I_{water}\rangle)$,
- ALLDIFFERENT$(\langle I_{old_gold}, I_{kool}, I_{chesterfield}, I_{lucky_strike}, I_{parliament}\rangle)$.

In our experience, when confronted for the first time to this puzzle, a lot of people come up with the model that associates to each house i $(1 \leq i \leq 5)$ five variables C_i, N_i, A_i, D_i, B_i that describe the attributes of the person living in house i. However it is difficult to directly express the constraints according to these variables and the second model which associates to each attribute a variable that gives the corresponding house is more convenient for expressing the constraints.

3.7.279 Zero-duration task [10 CONS]

- COLOURED_CUMULATIVE,
- COLOURED_CUMULATIVES,
- CUMULATIVE,
- CUMULATIVE_PRODUCT,
- CUMULATIVE_WITH_LEVEL_OF_PRIORITY,
- CUMULATIVES,
- DISJUNCTIVE,
- DISJUNCTIVE_OR_SAME_END,
- DISJUNCTIVE_OR_SAME_START,
- TASKS_INTERSECTION.

A resource scheduling constraint that accepts tasks which can potentially have a duration equal to zero. Zero-duration tasks can be used for modelling over-constrained resource scheduling problems where, due to some resource limitations, some tasks have to be discarded. This can be expressed by creating for each task i a duration variable D_i with values 0 and d_i in its initial domain, where d_i is the effective duration of task i when it is not discarded. Then, depending on the relaxation cost C_i associated with the fact that task i is not considered, a reified constraint of the form $D_i = 0 \Leftrightarrow C_i = \alpha_i$ $(\alpha_i > 0)$ is created. The initial domain of the cost variable C_i is set to 0 and α_i, where α_i is the cost associated with the decision of discarding task i. Then all the relaxation costs associated with the different tasks have to be aggregated together, i.e., typically by taking the sum or the maximum of the relaxation costs of the different tasks. On the one hand, the overall advantage of the approach is that it does not require developing any specific algorithm. On the other hand, the disadvantage is the lack of bounds on the overall relaxation cost that can sometimes be compensated by a specific enumeration heuristic.

4

Further Topics

Contents

4.1 Differences from the 2000 report

This section summarises the main differences with the SICS report [27] as well as of the corresponding article [28]. The main differences are listed below:

- We have both simplified and extended the way to generate the vertices of the initial graph and we have introduced a new way of defining set of vertices. We have also removed the CLIQUE(MAX) set of vertices generator since it cannot in general be evaluated in polynomial time. Therefore, we have modified the description of the constraints ASSIGN_AND_COUNTS, ASSIGN_AND_NVALUES, INTERVAL_AND_COUNT, INTERVAL_AND_SUM, BIN_PACKING, CUMULATIVE, CUMULATIVES, COLOURED_CUMULATIVE, COLOURED_CUMULATIVES, CUMULATIVE_TWO_D, which all used this feature.

- We have introduced the new arc generators *PATH_1* and *PATH_N*, which allow for specifying an n-ary constraint for which n is not fixed. The SIZE_MAX_STARTING_SEQ_ALLDIFFERENT and the SIZE_MAX_SEQ_ALLDIFFERENT are examples of global constraints that use these arc generators in order to generate a set of sliding ALLDIFFERENT constraints.

- In addition to traditional domain variables we have introduced *float*, *set* and *multiset* variables as well as several global constraints mentioning float and set variables (for example, the CHOQUET [231] and the ALLDIFFERENT_BETWEEN_SETS constraints). This decision was initially motivated by the fact that several constraint systems and articles mention global constraints dealing with these types of variables. Later on, we realised that set variables also greatly simplify the interface of existing global constraints. This was especially true for those global constraints that explicitly deal with a graph, like CLIQUE or CUTSET. In this context, using a set variable for catching the successors of a vertex is quite natural. This is especially true when a vertex of the final graph can have more than one successor since it allows for avoiding a lot of 0-1 variables.

- We have introduced the possibility of using more than one graph constraint for defining a given global constraint (for example, the CUMULATIVE or the SORT constraints). Therefore we have removed the notion of dual graph, which was initially introduced in the original report. In this context, we now use two graph constraints, for example, CHANGE_CONTINUITY.

- On the one hand, we have introduced the following new graph parameters:
  - **MAX_DRG**,
  - **MAX_OD**,
  - **MIN_DRG**,
  - **MIN_ID**,
  - **MIN_OD**,
  - **NTREE**,
  - **PATH_FROM_TO**,
  - **PROD**,
  - **RANGE**,
  - **RANGE_DRG**,
  - **RANGE_NCC**,
  - **SUM**,
  - **SUM_WEIGHT_ARC**.

  On the other hand, we have removed the following graph parameters:
  - **NCC**(COMP, `val`),
  - **NSCC**(COMP, `val`),
  - **NTREE**(ATTR, COMP, `val`),
  - **NSOURCE_EQ_NSINK**,
  - **NSOURCE_GREATEREQ_NSINK**.

  Finally, **MAX_IN_DEGREE** has been renamed **MAX_ID**.

- We have introduced an iterator over the items of a collection in order to specify in a generic way a set of similar elementary constraints or a set of similar graph properties. This was required for describing some global constraints such as GLOBAL_CARDINALITY, CYCLE_RESOURCE or STRETCH. All these global constraints mention a condition involving some limit depending on the specific values that are actually used. For example, the GLOBAL_CARDINALITY constraint forces each value v to be respectively used at least $\mathtt{atleast}_v$ and at most $\mathtt{atmost}_v$ times. This iterator was also necessary in the context of graph covering constraints where one wants to cover a digraph with some patterns. Each pattern consists of one resource and several tasks. One can now attach specific constraints to the different resources. Both the CYCLE_RESOURCE and the TREE_RESOURCE constraints illustrate this point.

- We have added some standard existing global constraints that were obviously missing from the previous report. This was the case, for example, of the ELEMENT constraint.
- In order to make clear the notion of *family* of global constraints we have computed for each global constraint a *signature*, which summarises its structure. Each signature was inserted into the index so that one can retrieve all the global constraints sharing the same structure.
- We have generalised some existing global constraints. For example, the CHANGE_PAIR constraint extends the CHANGE constraint. Finally we have introduced some novel global constraints like DISJOINT_TASKS or SYMMETRIC_GCC.
- We have defined the rules for specifying arc constraints.

4.2 Differences from the 2005 report

The second edition has more than 1300 pages of new content. The slots describing explicitly the meaning of a global constraint (e.g., the slots **Graph model** and **Automaton**) were moved to the last part of the description. This was motivated by the fact that most users want first to get the informal description of a global constraint (e.g., the slots **Purpose** and **Example**). Effort was not only devoted to the introduction of new constraints but also to a better description of multiple aspects like:

- The slot **Symmetries** describes a set of mapping that preserve the solution to a constraint (see Section 2.2.5).
- The slot **Reformulation** provides reformulation of a global constraint as a conjunction of constraints (see Section 2.5).
- The slot **Systems** gives links to concrete constraint systems.
- The slots **See also** and **Keywords** were redesigned in order to respectively indicate why we point to a given constraint (see Section 2.6) and to group together keywords by meta-keywords (see Section 3.6).
- In addition to the slots **Graph model** and **Automaton** that respectively describe the meaning of a global constraint in terms of graph properties and automaton, we have introduced the slot **Logic** in order to describe some geometrical constraints with first order formulae (see keyword Logic).
- Finally, an evaluator was provided for most global constraints.

4.3 Graph invariants

Within the scope of the graph-based description this section shows how to use implied constraints, which are systematically linked to the description of a global constraint. This usually occurs in the following context:

- Quite often, it happens that one wants to enforce the final graph to satisfy more than one graph property. In this context, these graph properties involve several graph parameters that cannot vary independently.

  EXAMPLE: As a practical example, consider the GROUP constraint and its first graph constraint. It involves the four graph parameters $\mathbf{NCC}$, $\mathbf{MIN\_NCC}$, $\mathbf{MAX\_NCC}$ and $\mathbf{NVERTEX}$, which respectively correspond to the number of connected components, the number of vertices of the smallest connected component, the number of vertices of the largest connected component and the number of vertices of the final graph. In this example the number of connected components of the final graph cannot vary independently from the size of the smallest connected component. The same remark applies also for the size of the largest connected component. Having a graph invariant that directly relates the four graph parameters can dramatically improve the propagation.

- Even though the description of a global constraint involves a single graph parameter $\mathbf{C}$, we can introduce the number of vertices, $\mathbf{NVERTEX}$, and the number of arcs, $\mathbf{NARC}$, of the final digraph. In this context, we can take advantage of graph invariants linking $\mathbf{C}$, $\mathbf{NARC}$ and $\mathbf{NVERTEX}$.

- It also happens that we enforce two graph constraints $\mathcal{GC}_1$ and $\mathcal{GC}_2$ that have the same initial graph G. In this context we consider the following situations:

  – Each arc of G belongs to one of the final graphs associated with $\mathcal{GC}_1$ or with $\mathcal{GC}_2$ (but not to both). An example of such global constraint is the CHANGE_CONTINUITY constraint. Within the graph invariants this situation is denoted by `apartition`.

  – Each vertex of G belongs to one of the final graphs associated with $\mathcal{GC}_1$ or with $\mathcal{GC}_2$ (but not to both). An example of such global constraint is the GROUP constraint. Within the graph invariants this situation is denoted by `vpartition`.

  In these situations the graph properties associated with the two graph constraints are also not independent.

In practice the graphs associated with global constraints have a regular structure that comes from the initial graph or from the property of the arc constraints. So, in addition to graph invariants that hold for any graph, we want also tighter graph invariants that hold for specific graph classes. The next section introduces the graph classes we consider, while the two other sections give the graph invariants on one and two graphs.

4.3.1 Graph classes

By definition, a graph invariant has to hold for any digraph. For example, we have the graph invariant $\mathbf{NARC} \leq \mathbf{NVERTEX}^2$, which relates the number of arcs and the number of vertices of any digraph. This invariant is sharp since the equality is reached for a clique. However, by considering the structure of a digraph, we can get sharper invariants. For example, if our digraph is a subset of an elementary path (e.g., we use the $PATH$ arc generator depicted by Figure 2.8) we have that $\mathbf{NARC} \leq \mathbf{NVERTEX} - 1$, which is a tighter bound of the maximum number of arcs since $\mathbf{NVERTEX} - 1 < \mathbf{NVERTEX}^2$. For this reason, we consider recurring graph classes that show up for different global constraints of the catalogue. Beside the graph classes that were introduced in Section 2.3.2 we also have the following classes relating several graph constraints:

- `apartition`: constraint defined by two graph constraints having the same initial graph, where each arc of the initial graph belongs to one of the final graphs (but not to both).

- `vpartition`: constraint defined by two graph constraints having the same initial graph, where each vertex of the initial graph belongs to one of the final graphs (but not to both).

In addition, we also consider graph constraints such that their final graphs is a subset of the graph generated by the arc generators:

- $CHAIN$,
- $CIRCUIT$,
- $CLIQUE$,
- $CLIQUE$(`Comparison`)
- $GRID$,
- $LOOP$,
- $PATH$,
- $PRODUCT$,
- $PRODUCT$(`Comparison`),
- $SYMMETRIC\_PRODUCT$,
- $SYMMETRIC\_PRODUCT$(`Comparison`),

where `Comparison` is one of the following comparison operators $\leq$, $\geq$, $<$, $>$, $=$, $\neq$.

4.3.2 Format of an invariant

As we previously saw, we have graph invariants that hold for any digraph as well as tighter graph invariants for specific graph classes. As a consequence, we partition the database in groups of graph invariants. A *group of graph invariants* corresponds to several invariants such that all invariants relate the same subset of graph parameters and such that all invariants are variations of the first invariant of the group taking into accounts the graph class. Therefore, the first invariant of a group has no precondition, while all other invariants have a non-empty precondition that characterises the graph class for which they hold.

EXAMPLE: As a first example consider the group of invariants denoted by Proposition 68, which relate the number of arcs $\mathbf{NARC}$ with the number of vertices of the smallest and largest connected component (i.e., $\mathbf{MIN\_NCC}$ and $\mathbf{MAX\_NCC}$).

$$\mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \mathbf{NARC} \geq \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} - 2 + (\mathbf{MIN\_NCC} = 1)$$

$$\texttt{equivalence} : \mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \mathbf{NARC} \geq \mathbf{MIN\_NCC}^2 + \mathbf{MAX\_NCC}^2$$

On the one hand, since the first rule has no precondition it corresponds to a general graph invariant. On the other hand the second rule specifies a tighter condition (since $\mathbf{MIN\_NCC}^2+\mathbf{MAX\_NCC}^2$ is greater than or equal to $\mathbf{MIN\_NCC}+\mathbf{MAX\_NCC}-2+(\mathbf{MIN\_NCC}=1)$), which only holds for a final graph that is reflexive, symmetric and transitive.

EXAMPLE: As a second example, consider the following group of invariants corresponding to Proposition 51, which relate the number of arcs $\mathbf{NARC}$ to the number of vertices $\mathbf{NVERTEX}$ according to the arc generator (see Figure 2.8) used for generating the initial digraph:

$$\mathbf{NARC} \leq \mathbf{NVERTEX}^2$$
$$\mathbf{arc\_gen} = \mathit{CIRCUIT} : \mathbf{NARC} \leq \mathbf{NVERTEX}$$
$$\mathbf{arc\_gen} = \mathit{CHAIN} : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2$$
$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\leq) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX}+1)}{2}$$
$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\geq) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX}+1)}{2}$$
$$\mathbf{arc\_gen} = \mathit{CLIQUE}(<) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX}-1)}{2}$$
$$\mathbf{arc\_gen} = \mathit{CLIQUE}(>) : \mathbf{NARC} \leq \frac{\mathbf{NVERTEX} \cdot (\mathbf{NVERTEX}-1)}{2}$$
$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\neq) : \mathbf{NARC} \leq \mathbf{NVERTEX}^2 - \mathbf{NVERTEX}$$
$$\mathbf{arc\_gen} = \mathit{CYCLE} : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX}$$
$$\mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NARC} \leq \mathbf{NVERTEX} - 1$$

4.3.3 Using the database of invariants

The purpose of this section is to provide a set of graph invariants, each invariant relating a given set of graph parameters. Once we have these graph invariants we can use them systematically by applying the following steps:

- For a given graph constraint we extract all the graph parameters occurring in its description. This can be done automatically by scanning the corresponding graph properties. Let $\mathcal{GP}$ denote this subset of graph parameters. For each graph parameter gp of $\mathcal{GP}$ we check if we have a graph property of the form $gp = var$ where var is a domain variable. If this is the case we record the pair (gp, var); if not, we create a new domain variable var and also record the pair (gp, var).

- We then search for all groups of graph invariants involving a subset of the previous graph parameters $\mathcal{GP}$. For each selected group we filter out those graph invariants for which the preconditions are not compatible with the graph class of the graph constraint under consideration. In each group we finally keep those invariants that have the maximum number of preconditions (i.e., the most specialised graph invariants).

- Finally we state all the previous collected graph invariants as implied constraints. This is achieved by using the variables associated with each graph parameter.

EXAMPLE: We continue with the example of the GROUP constraint and its first graph constraint. The steps for creating the implied constraints are:

- We first extract the graph parameters **NCC**, **MIN_NCC**, **MAX_NCC** and **NVERTEX** from the first graph constraint of the GROUP constraint. Since all the graph properties attached to the previous graph parameters have the form $gc = var$ we extract the corresponding domain variables and get the following pairs (**NCC**, NGROUP), (**MIN_NCC**, MIN_SIZE), (**MAX_NCC**, MAX_SIZE) and (**NVERTEX**, NVAL).
- We search for all groups of graph invariants involving the graph parameters **NCC**, **MIN_NCC**, **MAX_NCC** and **NVERTEX** and filter out the irrelevant graph invariants that cannot be applied on the graph class associated with the GROUP constraint.
- We state all the previous invariants by substituting each graph parameter by its corresponding variable, which leads to a set of implied constraints.

4.3.4 The database of graph invariants

For each combination of graph parameters we give the number of graph invariants we currently have. The items are sorted first in increasing number of graph parameters of the invariant, second in alphabetic order on the name of the parameters. All graph invariants assume a digraph for which each vertex has at least one arc. For some propositions, a figure depicts the corresponding final graph, that minimises or maximises a given graph parameter. The propositions of this section and their corresponding proofs use the notations introduced in Section 2.3.2 on page 68.

- Graph invariants involving one graph parameter of a final graph:
  - **MAX_NCC**: 1 (see Proposition 1),
  - **MAX_NSCC**: 2 (see Propositions 2 and 3),
  - **MIN_NCC**: 1 (see Proposition 4),
  - **MIN_NSCC**: 2 (see Propositions 5 and 6),
  - **NARC**: 1 (see Proposition 7),
  - **NCC**: 2 (see Propositions 8 and 9),
  - **NSCC**: 1 (see Proposition 10),
  - **NSINK**: 1 (see Proposition 11),
  - **NSOURCE**: 1 (see Proposition 12),
  - **NVERTEX**: 1 (see Proposition 13).
- Graph invariants involving two graph parameters of a final graph:

- MAX_NCC, MAX_NSCC: 2 (see Propositions 14 and 15),
- MAX_NCC, MIN_NCC: 2 (see Propositions 16 and 17),
- MAX_NCC, NARC: 2 (see Propositions 18 and 19),
- MAX_NCC, NSINK: 2 (see Propositions 20 and 21),
- MAX_NCC, NSOURCE: 2 (see Propositions 22 and 23),
- MAX_NCC, NVERTEX: 2 (see Propositions 24 and 25),
- MAX_NSCC, MIN_NSCC: 2 (see Propositions 26 and 27),
- MAX_NSCC, NARC: 2 (see Propositions 28 and 29),
- MAX_NSCC, NVERTEX: 2 (see Propositions 30 and 31),
- MIN_NCC, MIN_NSCC: 2 (see Propositions 32 and 33),
- MIN_NCC, NARC: 2 (see Propositions 34 and 35),
- MIN_NCC, NCC: 1 (see Proposition 36),
- MIN_NCC, NVERTEX: 3 (see Propositions 37, 38 and 39),
- MIN_NSCC, NARC: 2 (see Propositions 40 and 41),
- MIN_NSCC, NVERTEX: 2 (see Propositions 42 and 43),
- NARC, NCC: 2 (see Propositions 44 and 45),
- NARC, NSCC: 2 (see Propositions 46 and 47),
- NARC, NSINK: 1 (see Proposition 48),
- NARC, NSOURCE: 1 (see Proposition 49),
- NARC, NVERTEX: 4 (see Propositions 50, 51, 52 and 53),
- NCC, NSCC: 2 (see Propositions 54 and 55),
- NCC, NVERTEX: 3 (see Propositions 56 and 57 and 58),
- NSCC, NSINK: 1 (see Proposition 59),
- NSCC, NSOURCE: 1 (see Proposition 60),
- NSCC, NVERTEX: 3 (see Propositions 61, 62 and 63),
- NSINK, NVERTEX: 2 (see Propositions 64 and 65),
- NSOURCE, NVERTEX: 2 (see Propositions 66 and 67).

- Graph invariants involving three graph parameters of a final graph:
  - MAX_NCC, MIN_NCC, NARC: 1 (see Proposition 68),
  - MAX_NCC, MIN_NCC, NCC: 1 (see Proposition 69),
  - MAX_NCC, MIN_NCC, NVERTEX: 5 (see Propositions 70, 71, 72, 73 and 74),
  - MAX_NCC, NARC, NCC: 2 (see Propositions 75 and 76),
  - MAX_NCC, NARC, NVERTEX: 2 (see Propositions 77 and 78),
  - MAX_NCC, NCC, NSINK: 1 (see Proposition 79),
  - MAX_NCC, NCC, NSOURCE: 1 (see Proposition 80),
  - MAX_NCC, NCC, NVERTEX: 2 (see Propositions 81 and 82),
  - MAX_NSCC, MIN_NSCC, NARC: 1 (see Proposition 83),
  - MAX_NSCC, MIN_NSCC, NSCC: 1 (see Proposition 84),
  - MAX_NSCC, MIN_NSCC, NVERTEX: 2 (see Propositions 85 and 86),
  - MAX_NSCC, NCC, NVERTEX: 1 (see Proposition 87),
  - MAX_NSCC, NSCC, NVERTEX: 2 (see Propositions 88 and 89),

- **MIN_NCC**, **NARC**, **NVERTEX**: 2 (see Propositions 90 and 91),
- **MIN_NCC**, **NCC**, **NVERTEX**: 2 (see Propositions 92 and 93),
- **MIN_NSCC**, **NARC**, **NVERTEX**: 1 (see Proposition 94),
- **MIN_NSCC**, **NCC**, **NVERTEX**: 1 (see Proposition 95),
- **MIN_NSCC**, **NSCC**, **NVERTEX**: 2 (see Propositions 96 and 97),
- **NARC**, **NCC**, **NVERTEX**: 2 (see Propositions 98 and 99),
- **NARC**, **NSCC**, **NVERTEX**: 4 (see Propositions 100, 101, 102 and 103),
- **NARC**, **NSINK**, **NVERTEX**: 2 (see Propositions 104 and 105),
- **NARC**, **NSOURCE**, **NVERTEX**: 2 (see Propositions 106 and 107),
- **NSCC**, **NSINK**, **NSOURCE**: 1 (see Proposition 108),
- **NSINK**, **NSOURCE**, **NVERTEX**: 1 (see Proposition 109).

• Graph invariants involving four graph parameters of a final graph:

- **MAX_NCC**, **MIN_NCC**, **NARC**, **NCC**: 2 (see Propositions 110 and 111),
- **MAX_NCC**, **MIN_NCC**, **NCC**, **NVERTEX**: 2 (see Propositions 112 and 113),
- **MAX_NCC**, **NCC**, **NSINK**, **NSOURCE**: 1 (see Proposition 114),
- **MAX_NSCC**, **MIN_NSCC**, **NARC**, **NSCC**: 2 (see Propositions 115 and 116),
- **MAX_NSCC**, **MIN_NSCC**, **NSCC**, **NVERTEX**: 2 (see Propositions 117 and 118),
- **MIN_NCC**, **NARC**, **NCC**, **NVERTEX**: 1 (see Proposition 119),
- **NARC**, **NCC**, **NSCC**, **NVERTEX**: 2 (see Propositions 120 and 121),
- **NARC**, **NSINK**, **NSOURCE**, **NVERTEX**: 1 (see Proposition 122).

• Graph invariants involving five graph parameters of a final graph:

- **MAX_NCC**, **MIN_NCC**, **NARC**, **NCC**, **NVERTEX**: 1 (see Proposition 123),
- **MIN_NCC**, **NARC**, **NCC**, **NSCC**, **NVERTEX**: 1 (see Proposition 124).

• Graph invariants relating two parameters of two final graphs:

- $\mathbf{MAX\_NCC}_1$, $\mathbf{MIN\_NCC}_1$: 1 (see Proposition 125),
- $\mathbf{MAX\_NCC}_2$, $\mathbf{MIN\_NCC}_2$: 1 (see Proposition 126),
- $\mathbf{MAX\_NCC}_1$, $\mathbf{NCC}_2$: 1 (see Proposition 127),
- $\mathbf{MAX\_NCC}_2$, $\mathbf{NCC}_1$: 1 (see Proposition 128),
- $\mathbf{MIN\_NCC}_1$, $\mathbf{NCC}_2$: 1 (see Proposition 129),
- $\mathbf{MIN\_NCC}_2$, $\mathbf{NCC}_1$: 1 (see Proposition 130),
- $\mathbf{NARC}_1$, $\mathbf{NARC}_2$: 1 (see Proposition 131),
- $\mathbf{NCC}_1$, $\mathbf{NCC}_2$: 2 (see Propositions 132 and 133),
- $\mathbf{NVERTEX}_1$, $\mathbf{NVERTEX}_2$: 1 (see Proposition 134).

• Graph invariants relating three parameters of two final graphs:

- $\mathbf{MAX\_NCC}_1, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2$: 3 (see Propositions 135, 136 and 137),
- $\mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_2, \mathbf{MIN\_NCC}_1$: 3 (see Propositions 138, 139 and 140),
- $\mathbf{MAX\_NCC}_1, \mathbf{MIN\_NCC}_1, \mathbf{NVERTEX}_2$: 1 (see Proposition 141),
- $\mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_2, \mathbf{NVERTEX}_1$: 1 (see Proposition 142),

- $\mathbf{MIN\_NCC}_1, \mathbf{NARC}_2, \mathbf{NCC}_1$: 1 (see Proposition 143),
- $\mathbf{MIN\_NCC}_2, \mathbf{NARC}_1, \mathbf{NCC}_2$: 1 (see Proposition 144).

- Graph invariants relating four parameters of two final graphs:
  - $\mathbf{MAX\_NCC}_1, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_1$: 2 (see Propositions 145 and 146),
  - $\mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{NCC}_2$: 2 (see Propositions 147 and 148),
  - $\mathbf{MAX\_NCC}_1, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NVERTEX}_2$: 1 (see Proposition 149),
  - $\mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{NVERTEX}_1$: 1 (see Proposition 150).

- Graph invariants relating five parameters of two final graphs:
  - $\mathbf{MAX\_NCC}_1, \mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_1$: 7 (see Propositions 151, 152, 153, 154, 155, 156 and 157).
  - $\mathbf{MAX\_NCC}_1, \mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_2$: 7 (see Propositions 158, 159, 160, 161, 162, 163 and 164).

- Graph invariants relating six parameters of two final graphs:
  - $\mathbf{MAX\_NCC}_1$, $\mathbf{MAX\_NCC}_2$, $\mathbf{MIN\_NCC}_1$, $\mathbf{MIN\_NCC}_2$, $\mathbf{NCC}_1$, $\mathbf{NCC}_2$: 2 (see Propositions 165 and 166).

Graph invariants involving one parameter of a final graph

$\boxed{\mathbf{MAX\_NCC}}$

Proposition 1.

$$\texttt{no loop} : \mathbf{MAX\_NCC} \neq 1 \tag{4.1}$$

Proof. Since we do not have any loop, a non-empty connected component has at least two vertices. □

$\boxed{\mathbf{MAX\_NSCC}}$

Proposition 2.

$$\texttt{acyclic} : \mathbf{MAX\_NSCC} \leq 1 \tag{4.2}$$

Proof. Since we do not have any circuit, a non-empty strongly connected component consists of a single vertex. □

Proposition 3.

$$\texttt{no loop} : \mathbf{MAX\_NSCC} \neq 1 \tag{4.3}$$

Proof. Since we do not have any loop, a non-empty strongly connected component has at least two vertices. □

$\boxed{\mathbf{MIN\_NCC}}$

Proposition 4.

$$\texttt{no loop} : \mathbf{MIN\_NCC} \neq 1 \tag{4.4}$$

Proof. Since we do not have any loop, a non-empty connected component has at least two vertices. □

$\boxed{\mathbf{MIN\_NSCC}}$

Proposition 5.

$$\texttt{acyclic} : \mathbf{MIN\_NSCC} \leq 1 \tag{4.5}$$

Proof. Since we do not have any circuit, a non-empty strongly connected component consists of a single vertex. □

Proposition 6.

$$\texttt{no loop} : \mathbf{MIN\_NSCC} \neq 1 \tag{4.6}$$

Proof. Since we do not have any loop, a non-empty strongly connected component has at least two vertices. □

$\boxed{\mathbf{NARC}}$

Proposition 7.

$$\texttt{one\_succ} : \mathbf{NARC} = \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{4.7}$$

Proof. By definition of `one_succ`. □

$\boxed{\mathbf{NCC}}$

Proposition 8.

$$\texttt{no loop} : 2 \cdot \mathbf{NCC} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{4.8}$$

Proof. By definition of `no loop`, each connected component has at least two vertices. □

Proposition 9.

$$\texttt{consecutive\_loops\_are\_connected} : 2 \cdot \mathbf{NCC} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} + 1 \tag{4.9}$$

Proof. By definition of `consecutive_loops_are_connected`. □

$\boxed{\mathbf{NSCC}}$

Proposition 10.

$$\texttt{no loop} : 2 \cdot \mathbf{NSCC} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{4.10}$$

Proof. By definition of `no loop`, each strongly connected component has at least two vertices. □

$\boxed{\mathbf{NSINK}}$

Proposition 11.

$$\texttt{symmetric} : \mathbf{NSINK} = 0 \tag{4.11}$$

Proof. Since we do not have any isolated vertex. □

$\boxed{\mathbf{NSOURCE}}$

Proposition 12.

$$\texttt{symmetric} : \mathbf{NSOURCE} = 0 \tag{4.12}$$

Proof. Since we do not have any isolated vertex. □

$\boxed{\mathbf{NVERTEX}}$

Proposition 13.

$$\texttt{one\_succ} : \mathbf{NVERTEX} = \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{4.13}$$

Proof. By definition of `one_succ`. □

Graph invariants involving two parameters of a final graph

MAX_NCC, MAX_NSCC

Proposition 14.

$$\mathbf{MAX\_NCC} = 0 \Leftrightarrow \mathbf{MAX\_NSCC} = 0 \tag{4.14}$$

Proof. By definition of **MAX_NCC** and of **MAX_NSCC**. □

Proposition 15.

$$\mathbf{MAX\_NSCC} \leq \mathbf{MAX\_NCC} \tag{4.15}$$

Proof. **MAX_NSCC** is a lower bound of the size of the largest connected component since the largest strongly connected component is for sure included within a connected component. □

MAX_NCC, MIN_NCC

Proposition 16.

$$\mathbf{MAX\_NCC} = 0 \Leftrightarrow \mathbf{MIN\_NCC} = 0 \tag{4.16}$$

Proof. By definition of **MAX_NCC** and of **MIN_NCC**. □

Proposition 17.

$$\mathbf{MIN\_NCC} \leq \mathbf{MAX\_NCC} \tag{4.17}$$

Proof. By definition of **MIN_NCC** and of **MAX_NCC**. □

MAX_NCC, NARC

Proposition 18.

$$\mathbf{MAX\_NCC} = 0 \Leftrightarrow \mathbf{NARC} = 0 \tag{4.18}$$

Proof. By definition of **MAX_NCC** and of **NARC**. □

Proposition 19.

$$\mathbf{MAX\_NCC} > 0 \Rightarrow \mathbf{NARC} \geq \max(1, \mathbf{MAX\_NCC} - 1) \tag{4.19}$$

$$\texttt{symmetric} : \mathbf{MAX\_NCC} > 0 \Rightarrow \mathbf{NARC} \geq \max(1, 2 \cdot \mathbf{MAX\_NCC} - 2) \tag{4.20}$$

$$\texttt{equivalence} : \mathbf{NARC} \geq \mathbf{MAX\_NCC}^2 \tag{4.21}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \geq \mathbf{MAX\_NCC} - 1 \tag{4.22}$$

Proof.
(4.19) $\mathbf{MAX\_NCC}-1$ arcs are needed to connect **MAX_NCC** vertices that belong to a given connected component containing at least two vertices. And one arc is required for a connected component containing a single vertex.
(4.20) Similarly, when the graph is symmetric, $2 \cdot \mathbf{MAX\_NCC} - 2$ arcs are needed to connect **MAX_NCC** vertices that belong to a given connected component containing at least two vertices.
(4.21) Finally, when the graph is reflexive, symmetric and transitive, $\mathbf{MAX\_NCC}^2$ arcs are needed to connect **MAX_NCC** vertices that belong to a given connected component.
(4.22) When the initial graph corresponds to a path, the minimum number of arcs of a connected component involving n vertices is equal to $n - 1$.

□

$\boxed{\mathbf{MAX\_NCC}, \mathbf{NSINK}}$

Proposition 20.

$$\mathbf{MAX\_NCC} = 0 \Rightarrow \mathbf{NSINK} = 0 \quad (4.23)$$

Proof. By definition of **MAX_NCC** and of **NSINK**. □

Proposition 21.

$$\mathbf{NSINK} \geq 1 \Rightarrow \mathbf{MAX\_NCC} \geq 2 \quad (4.24)$$

Proof. Since we do not have any isolated vertex a sink is connected to at least one other vertex. Therefore, if the graph has a sink, there exists at least one connected component with at least two vertices. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{NSOURCE}}$

Proposition 22.

$$\mathbf{MAX\_NCC} = 0 \Rightarrow \mathbf{NSOURCE} = 0 \quad (4.25)$$

Proof. By definition of **MAX_NCC** and of **NSOURCE**. □

Proposition 23.

$$\mathbf{NSOURCE} \geq 1 \Rightarrow \mathbf{MAX\_NCC} \geq 2 \quad (4.26)$$

Proof. Since we do not have any isolated vertex a source is connected to at least one other vertex. Therefore, if the graph has a source, there exists at least one connected component with at least two vertices. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{NVERTEX}}$

Proposition 24.

$$\mathbf{MAX\_NCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \quad (4.27)$$

Proof. By definition of **MAX_NCC** and of **NVERTEX**. □

Proposition 25.

$$\mathbf{NVERTEX} \geq \mathbf{MAX\_NCC} \quad (4.28)$$

Proof. By definition of **MAX_NCC**. □

$\boxed{\mathbf{MAX\_NSCC}, \mathbf{MIN\_NSCC}}$

Proposition 26.

$$\mathbf{MAX\_NSCC} = 0 \Leftrightarrow \mathbf{MIN\_NSCC} = 0 \quad (4.29)$$

Proof. By definition of **MAX_NSCC** and of **MIN_NSCC**. □

Proposition 27.

$$\mathbf{MIN\_NSCC} \leq \mathbf{MAX\_NSCC} \quad (4.30)$$

Proof. By definition of **MIN_NSCC** and of **MAX_NSCC**. □

$\boxed{\textbf{MAX\_NSCC}, \textbf{NARC}}$

Proposition 28.

$$\textbf{MAX\_NSCC} = 0 \Leftrightarrow \textbf{NARC} = 0 \tag{4.31}$$

Proof. By definition of **MAX_NSCC** and of **NARC**. □

Proposition 29.

$$\textbf{NARC} \geq \textbf{MAX\_NSCC} \tag{4.32}$$

$$\texttt{symmetric} : \textbf{NARC} \geq 2 \cdot \textbf{MAX\_NSCC} \tag{4.33}$$

$$\texttt{equivalence} : \textbf{NARC} \geq \textbf{MAX\_NSCC}^2 \tag{4.34}$$

Proof. (4.32) In a strongly connected component at least one arc has to leave each vertex. Since we have at least one strongly connected component of **MAX_NSCC** vertices this leads to the previous inequality. □

$\boxed{\textbf{MAX\_NSCC}, \textbf{NVERTEX}}$

Proposition 30.

$$\textbf{MAX\_NSCC} = 0 \Leftrightarrow \textbf{NVERTEX} = 0 \tag{4.35}$$

Proof. By definition of **MAX_NSCC** and of **NVERTEX**. □

Proposition 31.

$$\textbf{NVERTEX} \geq \textbf{MAX\_NSCC} \tag{4.36}$$

Proof. By definition of **MAX_NSCC**. □

$\boxed{\textbf{MIN\_NCC}, \textbf{MIN\_NSCC}}$

Proposition 32.

$$\textbf{MIN\_NCC} = 0 \Leftrightarrow \textbf{MIN\_NSCC} = 0 \tag{4.37}$$

Proof. By definition of **MIN_NCC** and of **MIN_NSCC**. □

Proposition 33.

$$\textbf{MIN\_NCC} \geq \textbf{MIN\_NSCC} \tag{4.38}$$

Proof. By construction **MIN_NCC** is an upper bound of the number of vertices of the smallest strongly connected component. □

MIN_NCC, NARC

Proposition 34.

$$\mathbf{MIN\_NCC} = 0 \Leftrightarrow \mathbf{NARC} = 0 \tag{4.39}$$

Proof. By definition of MIN_NCC and of NARC. □

Proposition 35.

$$\mathbf{MIN\_NCC} > 0 \Rightarrow \mathbf{NARC} \geq \max(1, \mathbf{MIN\_NCC} - 1) \tag{4.40}$$

$$\texttt{symmetric} : \mathbf{MIN\_NCC} > 0 \Rightarrow \mathbf{NARC} \geq \max(1, 2 \cdot \mathbf{MIN\_NCC} - 2) \tag{4.41}$$

$$\texttt{equivalence} : \mathbf{NARC} \geq \mathbf{MIN\_NCC}^2 \tag{4.42}$$

$$\mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NARC} \geq \mathbf{MIN\_NCC} - 1 \tag{4.43}$$

Proof. Similar to Proposition 19. □

MIN_NCC, NCC

Proposition 36.

$$\texttt{consecutive\_loops\_are\_connected} : (\mathbf{MIN\_NCC}+1) \cdot \mathbf{NCC} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} + 1 \tag{4.44}$$

Proof. By definition of `consecutive_loops_are_connected`. □

MIN_NCC, NVERTEX

Proposition 37.

$$\mathbf{MIN\_NCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{4.45}$$

Proof. By definition of MIN_NCC and of NVERTEX. □

Proposition 38.

$$\mathbf{NVERTEX} \geq \mathbf{MIN\_NCC} \tag{4.46}$$

Proof. By definition of MIN_NCC. □

Proposition 39.

$$\mathbf{MIN\_NCC} \notin \left[\min\left(\left\lfloor \frac{\mathbf{NVERTEX}}{2} \right\rfloor, \left\lfloor \frac{\mathbf{NVERTEX}_{\texttt{INITIAL}} - 1}{2} \right\rfloor\right) + 1, \mathbf{NVERTEX} - 1\right] \tag{4.47}$$

Proof. On the one hand, if $\mathbf{NCC} \leq 1$, we have that $\mathbf{MIN\_NCC} \geq \mathbf{NVERTEX}$. On the other hand, if $\mathbf{NCC} > 1$, we have that $\mathbf{MIN\_NCC} + \mathbf{MIN\_NCC} \leq \mathbf{NVERTEX}$ and that $\mathbf{MIN\_NCC} + \mathbf{MIN\_NCC} + 1 \leq \mathbf{NVERTEX}_{\texttt{INITIAL}}$, which by isolating MIN_NCC and by grouping the two inequalities leads to $\mathbf{MIN\_NCC} \leq \min\left(\left\lfloor \frac{\mathbf{NVERTEX}}{2} \right\rfloor, \left\lfloor \frac{\mathbf{NVERTEX}_{\texttt{INITIAL}}-1}{2} \right\rfloor\right)$. The result follows. □

$\boxed{\mathbf{MIN\_NSCC}, \mathbf{NARC}}$

Proposition 40.

$$\mathbf{MIN\_NSCC} = 0 \Leftrightarrow \mathbf{NARC} = 0 \tag{4.48}$$

Proof. By definition of $\mathbf{MIN\_NSCC}$ and of $\mathbf{NARC}$. □

Proposition 41.

$$\mathbf{NARC} \geq \mathbf{MIN\_NSCC} \tag{4.49}$$

$$\texttt{symmetric} : \mathbf{NARC} \geq 2 \cdot \mathbf{MIN\_NSCC} \tag{4.50}$$

$$\texttt{equivalence} : \mathbf{NARC} \geq \mathbf{MIN\_NSCC}^2 \tag{4.51}$$

Proof. Similar to Proposition 29. □

$\boxed{\mathbf{MIN\_NSCC}, \mathbf{NVERTEX}}$

Proposition 42.

$$\mathbf{MIN\_NSCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{4.52}$$

Proof. By definition of $\mathbf{MIN\_NSCC}$ and of $\mathbf{NVERTEX}$. □

Proposition 43.

$$\mathbf{NVERTEX} \geq \mathbf{MIN\_NSCC} \tag{4.53}$$

Proof. By definition of $\mathbf{MIN\_NSCC}$. □

$\boxed{\mathbf{NARC}, \mathbf{NCC}}$

Proposition 44.

$$\mathbf{NARC} = 0 \Leftrightarrow \mathbf{NCC} = 0 \tag{4.54}$$

Proof. By definition of $\mathbf{NARC}$ and of $\mathbf{NCC}$. □

Proposition 45.

$$\mathbf{NARC} \geq \mathbf{NCC} \tag{4.55}$$

Proof. Each connected component contains at least one arc (since, by hypothesis, each vertex has at least one arc). □

$\boxed{\mathbf{NARC}, \mathbf{NSCC}}$

Proposition 46.

$$\mathbf{NARC} = 0 \Leftrightarrow \mathbf{NSCC} = 0 \tag{4.56}$$

Proof. By definition of $\mathbf{NARC}$ and of $\mathbf{NSCC}$. □

Proposition 47.

$$\mathbf{NARC} \geq \mathbf{NSCC} \tag{4.57}$$

$$\texttt{no loop} : \mathbf{NARC} \geq 2 \cdot \mathbf{NSCC} \tag{4.58}$$

Proof. 4.57 (respectively 4.58) holds since each strongly connected component contains at least one (respectively two) arc(s). □

$\boxed{\textbf{NARC}, \textbf{NSINK}}$

Proposition 48.

$$\textbf{NARC} \geq \textbf{NSINK} \tag{4.59}$$

Proof. Since isolated vertices are not allowed, each sink has a distinct ingoing arc. □

$\boxed{\textbf{NARC}, \textbf{NSOURCE}}$

Proposition 49.

$$\textbf{NARC} \geq \textbf{NSOURCE} \tag{4.60}$$

Proof. Since isolated vertices are not allowed, each source has a distinct outgoing arc. □

$\boxed{\textbf{NARC}, \textbf{NVERTEX}}$

Proposition 50.

$$\textbf{NARC} = 0 \Leftrightarrow \textbf{NVERTEX} = 0 \tag{4.61}$$

Proof. By definition of **NARC** and of **NVERTEX**. □

Proposition 51.

$$\textbf{NARC} \leq \textbf{NVERTEX}^2 \tag{4.62}$$

$$\textbf{arc\_gen} = \mathit{CIRCUIT} : \textbf{NARC} \leq \textbf{NVERTEX} \tag{4.63}$$

$$\textbf{arc\_gen} = \mathit{CHAIN} : \textbf{NARC} \leq 2 \cdot \textbf{NVERTEX} - 2 \tag{4.64}$$

$$\textbf{arc\_gen} = \mathit{CLIQUE}(\leq) : \textbf{NARC} \leq \frac{\textbf{NVERTEX} \cdot (\textbf{NVERTEX} + 1)}{2} \tag{4.65}$$

$$\textbf{arc\_gen} = \mathit{CLIQUE}(\geq) : \textbf{NARC} \leq \frac{\textbf{NVERTEX} \cdot (\textbf{NVERTEX} + 1)}{2} \tag{4.66}$$

$$\textbf{arc\_gen} = \mathit{CLIQUE}(<) : \textbf{NARC} \leq \frac{\textbf{NVERTEX} \cdot (\textbf{NVERTEX} - 1)}{2} \tag{4.67}$$

$$\textbf{arc\_gen} = \mathit{CLIQUE}(>) : \textbf{NARC} \leq \frac{\textbf{NVERTEX} \cdot (\textbf{NVERTEX} - 1)}{2} \tag{4.68}$$

$$\textbf{arc\_gen} = \mathit{CLIQUE}(\neq) : \textbf{NARC} \leq \textbf{NVERTEX}^2 - \textbf{NVERTEX} \tag{4.69}$$

$$\textbf{arc\_gen} = \mathit{CYCLE} : \textbf{NARC} \leq 2 \cdot \textbf{NVERTEX} \tag{4.70}$$

$$\textbf{arc\_gen} = \mathit{PATH} : \textbf{NARC} \leq \textbf{NVERTEX} - 1 \tag{4.71}$$

Proof. 4.62 holds since each vertex of a digraph can have at most **NVERTEX** successors. The next items correspond to the maximum number of arcs that can be achieved according to a specific arc generator. □

Note that, when the equality is reached in 4.62, the corresponding extreme graph is in fact the graph initially generated. The same observation holds for inequalities 4.63 to 4.71. As a consequence all U-arcs have to be turned into T-arcs.

Proposition 52.

$$2 \cdot \textbf{NARC} \geq \textbf{NVERTEX} \tag{4.72}$$

Proof. By induction on the number of vertices of a graph G:

1. If $\textbf{NVERTEX}(G)$ is equal to 1 or 2 Proposition 52 holds.

2. Assume that $\mathbf{NVERTEX}(G) \geq 3$.
   - Assume there exists a vertex v such that, if we remove v, we do not create any isolated vertex in the remaining graph. We have $\mathbf{NARC}(G) \geq \mathbf{NARC}(G - v) + 1$. Thus $2 \cdot \mathbf{NARC}(G) \geq 2 \cdot \mathbf{NARC}(G - v) + 1$. Since by induction hypothesis $2 \cdot \mathbf{NARC}(G - v) \geq \mathbf{NVERTEX}(G - v) = \mathbf{NVERTEX}(G) - 1$ the result holds.
   - Otherwise, all the connected components of G are reduced to two elements with only one arc. We remove one of such connected component (v, w).
     Thus $\mathbf{NARC}(G) = \mathbf{NARC}(G - \{v, w\}) + 1$. As by induction hypothesis, $2 \cdot \mathbf{NARC}(G - \{v, w\}) \geq \mathbf{NVERTEX}(G - \{v, w\}) = \mathbf{NVERTEX}(G) - 2$ the result holds.

□

Note that, when the equality is reached in 52, the corresponding extreme graph is in fact a perfect matching of the graph. As a consequence all U-arcs that do not belong to any perfect matching have to be turned into F-arcs.

Proposition 53.

$$\mathbf{arc\_gen} = LOOP : \mathbf{NARC} = \mathbf{NVERTEX} \tag{4.73}$$

Proof. From the definition of $LOOP$. □

NCC, NSCC

Proposition 54.

$$\mathbf{NCC} = 0 \Leftrightarrow \mathbf{NSCC} = 0 \tag{4.74}$$

Proof. By definition of **NCC** and of **NSCC**. □

Proposition 55.

$$\mathbf{NCC} \leq \mathbf{NSCC} \tag{4.75}$$

Proof. Holds since each connected component contains at least one strongly connected component. □

Note that, when the equality is reached in 55, each connected component of the corresponding extreme graph is strongly connected. As a consequence all sink vertices of the graph induced by the T-vertices and the T-arcs should have at least one successor.

NCC, NVERTEX

Proposition 56.

$$\mathbf{NCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{4.76}$$

Proof. By definition of **NCC** and of **NVERTEX**. □

Proposition 57.

$$\mathbf{NCC} \leq \mathbf{NVERTEX} \tag{4.77}$$

$$\texttt{no loop} : 2 \cdot \mathbf{NCC} \leq \mathbf{NVERTEX} \tag{4.78}$$

Proof. 4.77 (respectively 4.78) holds since each connected component contains at least one (respectively two) vertex. □

Note that, when the equality is reached in 4.77, the corresponding extreme graph does not contain any arc between two distinct vertices. As a consequence any U-arc between two distinct vertices is turned into a F-vertex.

Proposition 58.

$$\begin{array}{c}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}: \\ \mathbf{NVERTEX} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} - (\mathbf{NCC} - 1)\end{array} \tag{4.79}$$

Proof. Holds since between two "consecutive" connected components of the initial graph there is at least one vertex that is missing. □

NSCC, NSINK

Proposition 59.

$$\mathbf{NSCC} \geq \mathbf{NSINK} + 1 \tag{4.80}$$

Proof. Since each sink cannot belong to a circuit and since no isolated vertex is allowed at least one extra non-sink vertex is required the result follows. □

NSCC, NSOURCE

Proposition 60.

$$\mathbf{NSCC} \geq \mathbf{NSOURCE} + 1 \tag{4.81}$$

Proof. Since each source cannot belong to a circuit and since no isolated vertex is allowed at least one extra non-source vertex is required the result follows. □

NSCC, NVERTEX

Proposition 61.

$$\mathbf{NSCC} = 0 \Leftrightarrow \mathbf{NVERTEX} = 0 \tag{4.82}$$

Proof. By definition of **NSCC** and of **NVERTEX**. □

Proposition 62.

$$\mathbf{NSCC} \leq \mathbf{NVERTEX} \tag{4.83}$$

Proof. Proposition 62 holds since each strongly connected component contains at least one vertex. □

Proposition 63.

$$\texttt{acyclic}: \mathbf{NSCC} = \mathbf{NVERTEX} \tag{4.84}$$

Proof. In a directed acyclic graph we have that each vertex corresponds to a strongly connected component involving only that vertex. □

$\boxed{\mathbf{NSINK}, \mathbf{NVERTEX}}$

Proposition 64.

$$\mathbf{NVERTEX} = 0 \Rightarrow \mathbf{NSINK} = 0 \tag{4.85}$$

Proof. By definition of **NVERTEX** and of **NSINK**. □

Proposition 65.

$$\mathbf{NVERTEX} > 0 \Rightarrow \mathbf{NSINK} < \mathbf{NVERTEX} \tag{4.86}$$

Proof. Holds since each sink must have a predecessor that cannot be a sink and since each vertex has at least one arc. □

$\boxed{\mathbf{NSOURCE}, \mathbf{NVERTEX}}$

Proposition 66.

$$\mathbf{NVERTEX} = 0 \Rightarrow \mathbf{NSOURCE} = 0 \tag{4.87}$$

Proof. By definition of **NVERTEX** and of **NSOURCE**. □

Proposition 67.

$$\mathbf{NVERTEX} > 0 \Rightarrow \mathbf{NSOURCE} < \mathbf{NVERTEX} \tag{4.88}$$

Proof. Holds since each source must have a successor that cannot be a source and since each vertex has at least one arc. □

Graph invariants involving three parameters of a final graph

$\boxed{\mathbf{MAX\_NCC}, \mathbf{MIN\_NCC}, \mathbf{NARC}}$

Proposition 68.

$$\begin{aligned}&\mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow\\ &\mathbf{NARC} \geq \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} - 2 + (\mathbf{MIN\_NCC} = 1)\end{aligned} \tag{4.89}$$

$$\begin{aligned}&\texttt{equivalence} : \mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow\\ &\mathbf{NARC} \geq \mathbf{MIN\_NCC}^2 + \mathbf{MAX\_NCC}^2\end{aligned} \tag{4.90}$$

Proof. (4.89) $n - 1$ arcs are needed to connect n $(n > 1)$ vertices that all belong to a given connected component. Since we have two connected components, which respectively have **MIN_NCC** and **MAX_NCC** vertices, this leads to the previous inequality. When **MIN_NCC** is equal to one we need an extra arc. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{MIN\_NCC}, \mathbf{NCC}}$

Proposition 69.

$$\mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \mathbf{NCC} \geq 2 \tag{4.91}$$

Proof. If **MIN_NCC** and **MAX_NCC** are different then they correspond for sure to at least two distinct connected components. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{MIN\_NCC}, \mathbf{NVERTEX}}$

Proposition 70.

$$\mathbf{MIN\_NCC} \neq \mathbf{MAX\_NCC} \Rightarrow \mathbf{NVERTEX} \geq \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} \tag{4.92}$$

Proof. Since we have at least two distinct connected components, which respectively have **MIN_NCC** and **MAX_NCC** vertices, this leads to the previous inequality. □

Proposition 71.

$$\mathbf{MAX\_NCC} \leq \max(\mathbf{MIN\_NCC}, \mathbf{NVERTEX} - \max(1, \mathbf{MIN\_NCC})) \tag{4.93}$$

Proof. On the one hand, if $\mathbf{NCC} \leq 1$, we have that $\mathbf{MAX\_NCC} \leq \mathbf{MIN\_NCC}$. On the other hand, if $\mathbf{NCC} > 1$, we have that $\mathbf{NVERTEX} \geq \max(1, \mathbf{MIN\_NCC}) + \mathbf{MAX\_NCC}$ (i.e., $\mathbf{MAX\_NCC} \leq \mathbf{NVERTEX} - \max(1, \mathbf{MIN\_NCC})$). The result is obtained by taking the maximum value of the right-hand sides of the two inequalities. □

Proposition 72.

$$\mathbf{MIN\_NCC} \notin [\mathbf{NVERTEX} - \max(1, \mathbf{MAX\_NCC}) + 1, \mathbf{NVERTEX} - 1] \tag{4.94}$$

Proof. On the one hand, if $\mathbf{NCC} \leq 1$, we have that $\mathbf{MIN\_NCC} \geq \mathbf{NVERTEX}$. On the other hand, if $\mathbf{NCC} > 1$, we have that $\mathbf{MIN\_NCC} + \max(1, \mathbf{MAX\_NCC}) \leq \mathbf{NVERTEX}$ (i.e., $\mathbf{MIN\_NCC} \leq \mathbf{NVERTEX} - \max(1, \mathbf{MAX\_NCC})$). The result follows. □

Proposition 73.

$$\mathbf{NVERTEX} \notin [\mathbf{MIN\_NCC} + 1, \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} - 1] \tag{4.95}$$

Proof. On the one hand, if $\mathbf{NCC} \leq 1$, we have that $\mathbf{NVERTEX} \leq \mathbf{MIN\_NCC}$. On the other hand, if $\mathbf{NCC} > 1$, we have that $\mathbf{NVERTEX} \geq \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC}$. Since $\mathbf{MIN\_NCC} \leq \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC}$ the result follows. ☐

Proposition 74.

$$\texttt{if } \overline{\mathbf{MIN\_NCC}} > 0$$
$$\texttt{then } k_{inf} = \left\lfloor \frac{\underline{\underline{\mathbf{NVERTEX}}} + \overline{\mathbf{MIN\_NCC}}}{\overline{\mathbf{MIN\_NCC}}} \right\rfloor \texttt{ else } k_{inf} = 1$$

$$\texttt{if } \underline{\mathbf{MAX\_NCC}} > 0$$
$$\texttt{then } k_{sup_1} = \left\lfloor \frac{\overline{\mathbf{NVERTEX}} - 1}{\underline{\mathbf{MAX\_NCC}}} \right\rfloor \texttt{ else } k_{sup_1} = \overline{\mathbf{NVERTEX}}$$

$$\texttt{if } \underline{\mathbf{MAX\_NCC}} < \overline{\mathbf{MIN\_NCC}}$$
$$\texttt{then } k_{sup_2} = \left\lfloor \frac{\overline{\mathbf{MIN\_NCC}} - 2}{\underline{\mathbf{MAX\_NCC}} - \overline{\mathbf{MIN\_NCC}}} \right\rfloor \texttt{ else } k_{sup_2} = \overline{\mathbf{NVERTEX}}$$

$$k_{sup} = \min(k_{sup_1}, k_{sup_2})$$

$$\forall k \in [k_{inf}, k_{sup}] : \mathbf{NVERTEX} \notin [k \cdot \mathbf{MAX\_NCC} + 1, (k+1) \cdot \mathbf{MIN\_NCC} - 1] \tag{4.96}$$

Proof. We make the proof for $k \in \mathbb{N}$ (the interval $[k_{inf}, k_{sup}]$ is only used for restricting the number of intervals to check). We have that $\mathbf{NVERTEX} \in [k \cdot \mathbf{MIN\_NCC}, k \cdot \mathbf{MAX\_NCC}]$. A *forbidden interval* $[k \cdot \mathbf{MAX\_NCC} + 1, (k+1) \cdot \mathbf{MIN\_NCC} - 1]$ corresponds to an interval between the end of interval $[k \cdot \mathbf{MIN\_NCC}, k \cdot \mathbf{MAX\_NCC}]$ and the start of the next interval $[(k+1) \cdot \mathbf{MIN\_NCC}, (k+1) \cdot \mathbf{MAX\_NCC}]$. Since all intervals $[i \cdot \mathbf{MIN\_NCC}, i \cdot \mathbf{MAX\_NCC}]$ $(i < k)$ end before $k \cdot \mathbf{MAX\_NCC}$ and since all intervals $[j \cdot \mathbf{MIN\_NCC}, j \cdot \mathbf{MAX\_NCC}]$ $(j > k)$ start after $(k+1) \cdot \mathbf{MIN\_NCC}$, they do not use any value in $[k \cdot \mathbf{MAX\_NCC} + 1, (k+1) \cdot \mathbf{MIN\_NCC} - 1]$. ☐

$\boxed{\mathbf{MAX\_NCC}, \mathbf{NARC}, \mathbf{NCC}}$

Proposition 75.

$$\mathbf{NARC} \leq \mathbf{NCC} \cdot \mathbf{MAX\_NCC}^2 \tag{4.97}$$

$$\mathbf{arc\_gen} = PATH : \mathbf{NARC} \leq \mathbf{NCC} \cdot (\mathbf{MAX\_NCC} - 1) \tag{4.98}$$

Proof. On the one hand, (4.97) holds since the maximum number of arcs is achieved by taking $\mathbf{NCC}$ connected components where each connected component is a clique involving $\mathbf{MAX\_NCC}$ vertices. On the other hand, (4.98) holds since a tree of n vertices has $n-1$ arcs. ☐

Proposition 76.

$$\mathbf{NARC} \geq \mathbf{MAX\_NCC} + \mathbf{NCC} - 2 \tag{4.99}$$

Proof. The minimum number of arcs is achieved by taking one connected component with $\mathbf{MAX\_NCC}$ vertices and $\mathbf{MAX\_NCC} - 1$ arcs as well as $\mathbf{NCC} - 1$ connected components with a single vertex and a loop. ☐

MAX_NCC, NARC, NVERTEX

Proposition 77.

$$\mathbf{NARC} \leq \mathbf{MAX\_NCC}^2 \cdot \left\lfloor \frac{\mathbf{NVERTEX}}{\max(1, \mathbf{MAX\_NCC})} \right\rfloor + (\mathbf{NVERTEX} \bmod \max(1, \mathbf{MAX\_NCC}))^2 \tag{4.100}$$

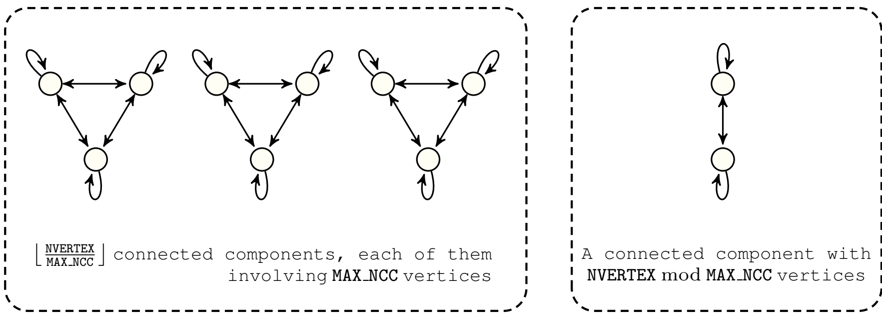


Figure 4.1: Illustration of Proposition 77. A graph that achieves the maximum number of arcs according to the size of the largest connected component as well as to a fixed number of vertices ($\texttt{MAX\_NCC} = 3, \texttt{NVERTEX} = 11, \texttt{NARC} = 3^2 \cdot \left\lfloor \frac{11}{\max(1,3)} \right\rfloor + (11 \bmod \max(1,3))^2 = 31$)

Proof. If $\mathbf{MAX\_NCC} = 0$ we get $\mathbf{NARC} \leq 0$ which holds since the set of vertices is empty. We now assume that $\mathbf{MAX\_NCC} > 0$. We first begin with the following claim:
let G be a graph such that $V(G) - \mathbf{NCC}(G, \mathbf{MAX\_NCC}(G)) * \mathbf{MAX\_NCC}(G) \geq \mathbf{MAX\_NCC}(G)$, then there exists a graph G' such that $V(G') = V(G)$, $\mathbf{MAX\_NCC}(G') = \mathbf{MAX\_NCC}(G)$, $\mathbf{NCC}(G', \mathbf{MAX\_NCC}(G')) = \mathbf{NCC}(G, \mathbf{MAX\_NCC}(G)) + 1$ and $|E(G)| \leq |E(G')|$.

Proof of the claim
Let $(C_i)_{i \in [n]}$ be the connected components of G on less than $\mathbf{MAX\_NCC}(G)$ vertices and such that $|C_i| \geq |C_{i+1}|$. By hypothesis there exists $k \leq n$ such that $|\bigcup_{i=1}^{k-1} C_i| < \mathbf{MAX\_NCC}(G)$ and $|\bigcup_{i=1}^{k} C_i| \geq \mathbf{MAX\_NCC}(G)$.

- Either $|\bigcup_{i=1}^{k} C_i| = \mathbf{MAX\_NCC}(G)$, and then with G' such that G' restricted to the $\bigcup_{i=1}^{k} C_i$ be a complete graph and G' restricted to $V(G) - \bigcup_{i=1}^{k} C_i$ being exactly G restricted to $V(G) - \bigcup_{i=1}^{k} C_i$ we obtain the claim.
- Or $|\bigcup_{i=1}^{k} C_i| > \mathbf{MAX\_NCC}(G)$. Then $C_k = C_k^1 \uplus C_k^2$ such that $|(\bigcup_{i=1}^{k-1} C_i) \cup C_k^1| = \mathbf{MAX\_NCC}(G)$ and $|C_k^2| < |C_1|$ (notice that $k \geq 2$). Then with G' such that G' restricted to $(\bigcup_{i=1}^{k-1} C_i) \cup C_k^1$ is a complete graph and G' restricted to $V(G) - ((\bigcup_{i=1}^{k-1} C_i) \cup C_k^1)$ is exactly G restricted to $V(G) - ((\bigcup_{i=1}^{k-1} C_i) \cup C_k^1)$ we obtain the claim.

End of proof of the claim

We prove by induction on $r(G) = \left\lfloor \frac{\mathbf{NVERTEX}(G)}{\mathbf{MAX\_NCC}(G)} \right\rfloor - \mathbf{NCC}(G, \mathbf{MAX\_NCC}(G))$, where G is any graph. For $r(G) = 0$ the result holds (see Prop 44). Otherwise, since $r(G) > 0$ we have that $V(G) - \mathbf{NCC}(G, \mathbf{MAX\_NCC}(G)) * \mathbf{MAX\_NCC}(G) \geq \mathbf{MAX\_NCC}(G)$, by the previous claim there exists G' with the same number of vertices and the same number of vertices in the largest connected component, such that $r(G') = r(G) - 1$. Consequently the result holds by induction. □

Proposition 78.

$$\mathbf{NARC} \geq \mathbf{MAX\_NCC} - 1 + \left\lfloor \frac{\mathbf{NVERTEX} - \mathbf{MAX\_NCC} + 1}{2} \right\rfloor \tag{4.101}$$

Proof. Let G be a graph, let X be a maximal size connected component of G, then we have $G = G[X] \oplus G[V(G) - X]$. On the one hand, as $G[X]$ is connected, by setting $\mathbf{NCC} = 1$ in 4.143 of Proposition 99, we have $|E(G[X]) \geq |X| - 1$, on the other hand, by Proposition 52, $|E(G[V(G) - X])| \geq \left\lceil \frac{|V(G)-X|}{2} \right\rceil$. Thus the result follows. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{NCC}, \mathbf{NSINK}}$

Proposition 79.

$$\mathbf{NSINK} \leq \mathbf{NCC} \cdot \max(0, \mathbf{MAX\_NCC} - 1) \tag{4.102}$$

Proof. Since a connected component contains at most **MAX_NCC** vertices and since it does not contain any isolated vertex a connected component involves at most $\mathbf{MAX\_NCC} - 1$ sinks. Thus the result follows. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{NCC}, \mathbf{NSOURCE}}$

Proposition 80.

$$\mathbf{NSOURCE} \leq \mathbf{NCC} \cdot \max(0, \mathbf{MAX\_NCC} - 1) \tag{4.103}$$

Proof. Similar to Proposition 79. □

$\boxed{\mathbf{MAX\_NCC}, \mathbf{NCC}, \mathbf{NVERTEX}}$

Proposition 81.

$$\mathbf{NVERTEX} \leq \mathbf{NCC} \cdot \mathbf{MAX\_NCC} \tag{4.104}$$

Proof. The number of vertices is less than or equal to the number of connected components multiplied by the largest number of vertices in a connected component. □

Proposition 82.

$$\mathbf{NVERTEX} \geq \mathbf{MAX\_NCC} + \max(0, \mathbf{NCC} - 1) \tag{4.105}$$

$$\texttt{no loop} : \mathbf{NVERTEX} \geq \mathbf{MAX\_NCC} + \max(0, 2 \cdot \mathbf{NCC} - 2) \tag{4.106}$$

Proof. (4.105) The minimum number of vertices according to a fixed number of connected components **NCC** such that one of the connected components contains **MAX_NCC** vertices is obtained as follows: we get **MAX_NCC** vertices from the connected component involving **MAX_NCC** vertices and one vertex for each remaining connected component. □

$\boxed{\mathbf{MAX\_NSCC}, \mathbf{MIN\_NSCC}, \mathbf{NARC}}$

Proposition 83.

$$\mathbf{MIN\_NSCC} \neq \mathbf{MAX\_NSCC} \Rightarrow \mathbf{NARC} \geq \mathbf{MIN\_NSCC} + \mathbf{MAX\_NSCC} \quad (4.107)$$

$$\begin{gathered}\texttt{equivalence}: \mathbf{MIN\_NSCC} \neq \mathbf{MAX\_NSCC} \Rightarrow \\ \mathbf{NARC} \geq \mathbf{MIN\_NSCC}^2 + \mathbf{MAX\_NSCC}^2\end{gathered} \quad (4.108)$$

Proof. (4.107) In a strongly connected component at least one arc has to leave each arc. Since we have two strongly connected components, which respectively have **MIN_NSCC** and **MAX_NSCC** vertices, this leads to the previous inequality. □

$\boxed{\mathbf{MAX\_NSCC}, \mathbf{MIN\_NSCC}, \mathbf{NSCC}}$

Proposition 84.

$$\mathbf{MIN\_NSCC} \neq \mathbf{MAX\_NSCC} \Rightarrow \mathbf{NSCC} \geq 2 \quad (4.109)$$

Proof. Follows from the definitions of **MIN_NSCC** and of **MAX_NSCC**. □

$\boxed{\mathbf{MAX\_NSCC}, \mathbf{MIN\_NSCC}, \mathbf{NVERTEX}}$

Proposition 85.

$$\mathbf{MIN\_NSCC} \neq \mathbf{MAX\_NSCC} \Rightarrow \mathbf{NVERTEX} \geq \mathbf{MIN\_NSCC} + \mathbf{MAX\_NSCC} \quad (4.110)$$

Proof. Since we have at least two distinct strongly connected components, which respectively have **MIN_NSCC** and **MAX_NSCC** vertices, this leads to the previous inequality. □

Proposition 86.

$$\begin{gathered}\texttt{if } \overline{\mathbf{MIN\_NSCC}} > 0 \\ \texttt{then } k_{inf} = \left\lfloor \frac{\underline{\mathbf{NVERTEX}} + \overline{\mathbf{MIN\_NSCC}}}{\overline{\mathbf{MIN\_NSCC}}} \right\rfloor \texttt{ else } k_{inf} = 1 \\ \\ \texttt{if } \underline{\mathbf{MAX\_NSCC}} > 0 \\ \texttt{then } k_{sup_1} = \left\lfloor \frac{\overline{\mathbf{NVERTEX}} - 1}{\underline{\mathbf{MAX\_NSCC}}} \right\rfloor \texttt{ else } k_{sup_1} = \overline{\mathbf{NVERTEX}} \\ \\ \texttt{if } \underline{\mathbf{MAX\_NSCC}} < \overline{\mathbf{MIN\_NSCC}} \\ \texttt{then } k_{sup_2} = \left\lfloor \frac{\overline{\mathbf{MIN\_NSCC}} - 2}{\underline{\mathbf{MAX\_NSCC}} - \overline{\mathbf{MIN\_NSCC}}} \right\rfloor \texttt{ else } k_{sup_2} = \overline{\mathbf{NVERTEX}} \\ \\ k_{sup} = \min(k_{sup_1}, k_{sup_2})\end{gathered}$$

$$\forall k \in [k_{inf}, k_{sup}] : \mathbf{NVERTEX} \notin [k \cdot \mathbf{MAX\_NSCC} + 1, (k+1) \cdot \mathbf{MIN\_NSCC} - 1] \quad (4.111)$$

Proof. Similar to Proposition 74. □

$\boxed{\mathbf{MAX\_NSCC}, \mathbf{NCC}, \mathbf{NVERTEX}}$

Proposition 87.

$$\mathbf{NVERTEX} \leq \mathbf{NCC} \cdot \mathbf{MAX\_NSCC} \tag{4.112}$$

Proof. The largest number of vertices is obtained by putting within each connected component the number of vertices of the largest strongly connected component. □

$\boxed{\mathbf{MAX\_NSCC}, \mathbf{NSCC}, \mathbf{NVERTEX}}$

Proposition 88.

$$\mathbf{NVERTEX} \leq \mathbf{NSCC} \cdot \mathbf{MAX\_NSCC} \tag{4.113}$$

Proof. Since each strongly connected component contains at most **MAX_NSCC** vertices the total number of vertices is less than or equal to $\mathbf{NSCC} \cdot \mathbf{MAX\_NSCC}$. □

Proposition 89.

$$\mathbf{NVERTEX} \geq \mathbf{MAX\_NSCC} + \max(0, \mathbf{NSCC} - 1) \tag{4.114}$$

$$\texttt{no loop} : \mathbf{NVERTEX} \geq \mathbf{MAX\_NSCC} + \max(0, 2 \cdot \mathbf{NSCC} - 2) \tag{4.115}$$

Proof. (4.114) The minimum number of vertices according to a fixed number of strongly connected components **NSCC** such that one of them contains **MAX_NSCC** vertices is equal to $\mathbf{MAX\_NSCC} + \max(0, \mathbf{NSCC} - 1)$. □

$\boxed{\mathbf{MIN\_NCC}, \mathbf{NARC}, \mathbf{NVERTEX}}$

Proposition 90.

$$\mathbf{NARC} \leq \mathbf{MIN\_NCC}^2 + (\mathbf{NVERTEX} - \mathbf{MIN\_NCC})^2 \tag{4.116}$$

$$\mathbf{arc\_gen} = \mathit{CIRCUIT} : \mathbf{NARC} \leq \mathbf{NVERTEX} - 2 \cdot (\mathbf{MIN\_NCC} < \mathbf{NVERTEX}) \tag{4.117}$$

$$\mathbf{arc\_gen} = \mathit{CHAIN} : \mathbf{NARC} \leq \mathbf{NVERTEX} - 2 \cdot (\mathbf{MIN\_NCC} < \mathbf{NVERTEX}) \tag{4.118}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\leq) : \mathbf{NARC} \leq \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC} + 1)}{2} + \frac{(\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NCC} + 1)}{2} \tag{4.119}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\geq) : \mathbf{NARC} \leq \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC} + 1)}{2} + \frac{(\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NCC} + 1)}{2} \tag{4.120}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(<) : \mathbf{NARC} \leq \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC} - 1)}{2} + \frac{(\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NCC} - 1)}{2} \tag{4.121}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(>) : \mathbf{NARC} \leq \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC} - 1)}{2} + \frac{(\mathbf{NVERTEX} - \mathbf{MIN\_NCC}) \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NCC} - 1)}{2} \tag{4.122}$$

$$\begin{aligned}\mathbf{arc\_gen} = \mathit{CLIQUE}(\neq) : \mathbf{NARC} \leq \mathbf{MIN\_NCC}^2 - \mathbf{MIN\_NCC} + \\ (\mathbf{NVERTEX} - \mathbf{MIN\_NCC})^2 - (\mathbf{NVERTEX} - \mathbf{MIN\_NCC})\end{aligned} \tag{4.123}$$

$$\mathbf{arc\_gen} = \mathit{CYCLE} : \mathbf{NARC} \leq \mathbf{NVERTEX} - 4 \cdot (\mathbf{MIN\_NCC} < \mathbf{NVERTEX}) \tag{4.124}$$

$$\begin{aligned}\mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NARC} \leq \max(0, \mathbf{MIN\_NCC} - 1) + \\ \max(0, \mathbf{NVERTEX} - \mathbf{MIN\_NCC} - 1)\end{aligned} \tag{4.125}$$

Proof. (4.116) The maximum number of vertices according to a fixed number of vertices **NVERTEX** and to the fact there is a connected component with **MIN_NCC** vertices is obtained by:

- Building a connected component with **MIN_NCC** vertices and creating an arc between each pair of vertices.
- Building a connected component with all the **NVERTEX** − **MIN_NCC** remaining vertices and creating an arc between each pair of vertices.

□

Proposition 91.

$$\begin{aligned}&\mathbf{MIN\_NCC} > 1 \Rightarrow \\ &\mathbf{NARC} \geq \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{MIN\_NCC}} \right\rfloor \cdot (\mathbf{MIN\_NCC} - 1) + \mathbf{NVERTEX} \bmod \mathbf{MIN\_NCC}\end{aligned} \tag{4.126}$$

Proof. Achieving the minimum number of arcs with a fixed number of vertices and with a minimum number of vertices greater than or equal to one in each connected component is achieved in the following way:

- Since the minimum number of arcs of a connected component of n vertices is $n-1$, splitting a connected component into k parts that all have more than one vertex saves $k-1$ arcs. Therefore we build a maximum number of connected components. Since each connected component has at least **MIN_NCC** vertices we get $\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{MIN\_NCC}} \rfloor$ connected components.
- Since we cannot build a connected component with the rest of the vertices (i.e., $\mathbf{NVERTEX} \bmod \mathbf{MIN\_NCC}$ vertices left) we have to incorporate them in the previous connected components and this costs one arc for each vertex.

□

When $\mathbf{MIN\_NCC} = 1$, note that Proposition 52 provides a lower bound on the number of arcs.

$\boxed{\textbf{MIN\_NCC}, \textbf{NCC}, \textbf{NVERTEX}}$

Proposition 92.

$$\mathbf{NVERTEX} \geq \mathbf{NCC} \cdot \mathbf{MIN\_NCC} \tag{4.127}$$

Proof. The smallest number of vertices is obtained by taking all connected components to their minimum number of vertices **MIN_NCC**. □

Proposition 93.

$$\mathbf{NVERTEX} > \mathbf{MIN\_NCC} \Rightarrow \mathbf{NCC} \geq 2 \tag{4.128}$$

Proof. If all vertices do not fit within the smallest connected component then we have at least two connected components. □

$\boxed{\textbf{MIN\_NSCC}, \textbf{NARC}, \textbf{NVERTEX}}$

Proposition 94.

$$\mathbf{NARC} \leq \mathbf{NVERTEX}^2 + \mathbf{MIN\_NSCC}^2 - \mathbf{NVERTEX} \cdot \mathbf{MIN\_NSCC} \tag{4.129}$$

Proof. Achieving the maximum number of arcs, provided that we have at least one strongly connected component with **MIN_NSCC** vertices, is done by:

- Building a first strongly connected component $\mathcal{C}_1$ with **MIN_NSCC** vertices and adding an arc between each pair of vertices of $\mathcal{C}_1$.
- Building a second strongly connected component $\mathcal{C}_2$ with $\mathbf{NVERTEX} - \mathbf{MIN\_NSCC}$ vertices and adding an arc between each pair of vertices of $\mathcal{C}_2$.

Finally, we add an arc from every vertex of $\mathcal{C}_1$ to every vertex of $\mathcal{C}_2$. This leads to a total number of arcs of $\mathbf{MIN\_NSCC}^2 + (\mathbf{NVERTEX} - \mathbf{MIN\_NSCC})^2 + \mathbf{MIN\_NSCC} \cdot (\mathbf{NVERTEX} - \mathbf{MIN\_NSCC})$. □

$\boxed{\textbf{MIN\_NSCC}, \textbf{NCC}, \textbf{NVERTEX}}$

Proposition 95.

$$\mathbf{NVERTEX} \geq \mathbf{NCC} \cdot \mathbf{MIN\_NSCC} \tag{4.130}$$

Proof. The smallest number of vertices is obtained by putting within each connected component the number of vertices of the smallest strongly connected component. □

$\boxed{\textbf{MIN\_NSCC}, \textbf{NSCC}, \textbf{NVERTEX}}$

Proposition 96.

$$\mathbf{NVERTEX} \geq \mathbf{NSCC} \cdot \mathbf{MIN\_NSCC} \tag{4.131}$$

Proof. Since each strongly connected component contains at least **MIN_NSCC** vertices the total number of vertices is greater than or equal to $\mathbf{NSCC} \cdot \mathbf{MIN\_NSCC}$. □

Proposition 97.

$$\mathbf{NVERTEX} > \mathbf{MIN\_NSCC} \Rightarrow \mathbf{NSCC} \geq 2 \tag{4.132}$$

Proof. If all vertices do not fit within the smallest strongly connected component then we have at least two strongly connected components. □

NARC, NCC, NVERTEX

Proposition 98.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NCC} + 1)^2 + \mathbf{NCC} - 1 \tag{4.133}$$

$$\mathbf{arc\_gen} = \mathit{CIRCUIT} : \mathbf{NARC} \leq \mathbf{NVERTEX} - \mathbf{NCC} + 1 - (\mathbf{NCC} \neq 1) \tag{4.134}$$

$$\mathbf{arc\_gen} = \mathit{CHAIN} : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2 \cdot \mathbf{NCC} \tag{4.135}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\leq) : \mathbf{NARC} \leq \mathbf{NCC} - 1 + \frac{(\mathbf{NVERTEX} - \mathbf{NCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 2)}{2} \tag{4.136}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\geq) : \mathbf{NARC} \leq \mathbf{NCC} - 1 + \frac{(\mathbf{NVERTEX} - \mathbf{NCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 2)}{2} \tag{4.137}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(<) : \mathbf{NARC} \leq \mathbf{NCC} - 1 + \frac{(\mathbf{NVERTEX} - \mathbf{NCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC})}{2} \tag{4.138}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(>) : \mathbf{NARC} \leq \mathbf{NCC} - 1 + \frac{(\mathbf{NVERTEX} - \mathbf{NCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC})}{2} \tag{4.139}$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\neq) : \mathbf{NARC} \leq \max(0, \mathbf{NCC} - 1) + (\mathbf{NVERTEX} - \mathbf{NCC} + 1)^2 - (\mathbf{NVERTEX} - \mathbf{NCC} + 1) \tag{4.140}$$

$$\mathbf{arc\_gen} = \mathit{CYCLE} : \mathbf{NARC} \leq 2 \cdot \mathbf{NVERTEX} - 2 \cdot \mathbf{NCC} + 2 \cdot (\mathbf{NCC} = 1) \tag{4.141}$$

$$\mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NARC} = \mathbf{NVERTEX} - \mathbf{NCC} \tag{4.142}$$

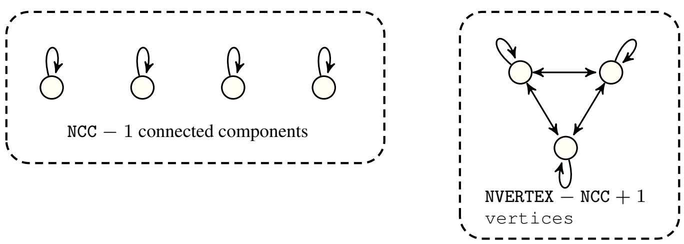


Figure 4.2: Illustration of Proposition 98. A graph that achieves the maximum number of arcs according to a fixed number of connected components as well as to a fixed number of vertices ($\mathtt{NCC} = 5, \mathtt{NVERTEX} = 7, \mathtt{NARC} = (7 - 5 + 1)^2 + 5 - 1 = 13$)

Proof. (4.133) We proceed by induction on $T(G) = \mathbf{NVERTEX}(G) - |X| - (\mathbf{NCC}(G) - 1)$, where X is any connected component of G of maximum cardinality. For $T(G) = 0$ then either $\mathbf{NCC}(G) = 1$ and thus the formula is clearly true, or all the connected components of G, but possibly X, are reduced to one element. Since isolated vertices are not allowed, the formula holds.

Assume that $T(G) \geq 1$. Then there exists Y, a connected component of G distinct from X, with more than one vertex. Let $y \in Y$ and let G' be the graph such that $V(G') = V(G)$ and $E(G')$ is defined by:

- For all Z connected components of G distinct from X and Y we have $G'[Z] = G[Z]$.
- With $X' = X \cup \{y\}$ and $Y' = Y - \{y\}$, we have $G'[Y'] = G[Y']$ and $E(G'[X']) = E(G[X]) \cup (\bigcup_{x \in X'} \{(x, y), (y, x)\})$.

Clearly $|E(G')| - |E(G)| \geq 2 \cdot |X| + 1 - (2 \cdot |Y| - 1)$ and since X is of maximal cardinality the difference is strictly positive. Now as $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G)$, $\mathbf{NCC}(G') = \mathbf{NCC}(G)$ and as $T(G') = T(G) - 1$ the result holds by induction hypothesis. □

Proposition 99.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \mathbf{NCC} \tag{4.143}$$

$$\texttt{equivalence} : \mathbf{NCC} > 0 \Rightarrow$$
$$\mathbf{NARC} \geq (\mathbf{NVERTEX} \bmod \mathbf{NCC}) \cdot \left(\left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NCC}} \right\rfloor + 1\right)^2 +$$
$$(\mathbf{NCC} - \mathbf{NVERTEX} \bmod \mathbf{NCC}) \cdot \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NCC}} \right\rfloor^2 \tag{4.144}$$

Proof. (4.143) By induction of the number of vertices. The formula holds for one vertex. Let G a graph with $n+1$ vertices ($n \geq 1$). First assume there exists x in G such that $G - x$ has the same number of connected components than G. Since $\mathbf{NARC}(G) \geq \mathbf{NARC}(G - x) + 1$, and by induction hypothesis $\mathbf{NARC}(G - x) \geq \mathbf{NVERTEX}(G - x) - \mathbf{NCC}(G - x)$ the result holds. Otherwise all connected components of G are reduced to one vertex and the formula holds. □

$\boxed{\mathbf{NARC}, \mathbf{NSCC}, \mathbf{NVERTEX}}$

Proposition 100.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NSCC} + 1) \cdot \mathbf{NVERTEX} + \frac{\mathbf{NSCC} \cdot (\mathbf{NSCC} - 1)}{2} \tag{4.145}$$

$$\texttt{equivalence} : \mathbf{NARC} \leq \mathbf{NSCC} - 1 + (\mathbf{NVERTEX} - \mathbf{NSCC} + 1)^2 \tag{4.146}$$

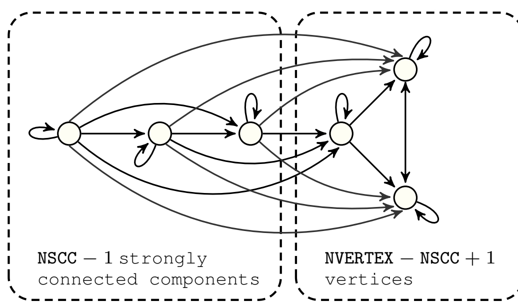


Figure 4.3: Illustration of Proposition 100(4.145). A graph that achieves the maximum number of arcs according to a fixed number of strongly connected components as well as to a fixed number of vertices ($\texttt{NSCC} = 5, \texttt{NVERTEX} = 6, \texttt{NARC} = (6 - 5 + 1) \cdot 6 + \frac{5 \cdot (5-1)}{2} = 22$)

Proof. For proving 4.145, it is easier to rewrite the formula as $\mathbf{NARC} \leq (\mathbf{NVERTEX} - (\mathbf{NSCC} - 1))^2 + (\mathbf{NCC} - 1) \cdot (\mathbf{NVERTEX} - (\mathbf{NSCC} - 1)) + \frac{\mathbf{NSCC} \cdot (\mathbf{NSCC} - 1)}{2}$. We proceed by induction on $T(G) = \mathbf{NVERTEX}(G) - |X| - (\mathbf{NSCC}(G) - 1)$, where X is any strongly connected component of G of maximum cardinality.

For $T(G) = 0$ then either $\mathbf{NSCC}(G) = 1$ and thus the formula is clearly true, or all the strongly connected components of G, but possibly X, are reduced to one element. Since the maximum number of arcs in a directed acyclic graph of n vertices is $\frac{n\cdot(n+1)}{2}$, and as the subgraph of G induced by all the strongly connected components of G excepted X is acyclic, the formula clearly holds.

Assume that $T(G) \geq 1$, let $(X_i)_{i\in I}$ be the family of strongly connected components of G, and let G_r be the reduced graph of G induced by $(X_i)_{i\in I}$ (that is $V(G_r) = I$ and $\forall i_1, i_2 \in I$, $(i_1, i_2) \in E(G_r)$ if and only if $\exists x_1 \in X_{i_1}$, $\exists x_2 \in X_{i_2}$ such that $(x_1, x_2) \in E$). Consider G' such that $V(G') = V(G)$ and $E(G')$ is defined by:

- For all strongly connected components Z of G we have $G'[Z] = G[Z]$.
- For σ be any topological sort of G_r, $\forall x_i \in X_i$, $\forall x_j \in X_j$, $(x_i, x_j) \in E(G')$ whenever i is less than j with respect to σ.

Notice that G' satisfies the following properties: $T(G') = T(G)$, $V(G') = V(G)$, $\mathbf{NSCC}(G') = \mathbf{NSCC}(G)$, $E(G) \subseteq E(G')$, $(X_i)_{i\in I}$ is still the family of strongly connected components of G', and moreover, for every $i \in I$ and every $x_i \in X_i$ we have that x_i is connected to any vertex outside X_i, that is the number of arcs incident to x_i and incident to vertices outside X_i is exactly $|V(G')| - |X_i|$.

Now, as $T(G') \geq 1$, there exists Y, a strongly connected component of G' distinct from X, with more than one vertex. Let $y \in Y$ and let G'' be the graph such that $V(G'') = V(G')$ and $E(G'')$ is defined by:

- $G''[V(G) - \{y\}] = G'[V(G) - \{y\}]$.
- With $X' = X \cup \{y\}$, we have $G''[Y'] = G'[Y']$ and $E(G''[X']) = E(G'[X]) \cup (\bigcup_{x\in X'}\{(x, y), (y, x)\})$.
- Assume that $X = X_j$ for $j \in I$. Then $\forall i \in I - \{j\}$, $\forall x_i \in X_i$, $(x_i, y) \in E(G'')$ whenever i is less than j with respect to σ and $(y, x_i) \in E(G'')$ whenever j is less than i with respect to σ.

Clearly $|E(G'')| - |E(G')| \geq 2|X| + 1 + |V(G')| - |X| - (2\cdot|Y| - 1 + |V(G')| - |Y|) = |X| - |Y| + 2$ and since X is of maximal cardinality the difference is strictly positive. As $E(G) \subseteq E(G')$, $|E(G'')| - |E(G)|$ is also strictly positive. Now as $\mathbf{NVERTEX}(G'') = \mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G)$, $\mathbf{NSCC}(G'') = \mathbf{NSCC}(G') = \mathbf{NSCC}(G)$ and as $T(G'') = T(G') - 1 = T(G) - 1$ the result holds by induction hypothesis. ☐

Proposition 101.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \left\lfloor \frac{\mathbf{NSCC} - 1}{2} \right\rfloor \tag{4.147}$$

$$\begin{gathered}\texttt{equivalence} : \mathbf{NSCC} > 0 \Rightarrow \\ \mathbf{NARC} \geq (\mathbf{NVERTEX} \bmod \mathbf{NSCC}) \cdot \left(\left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NSCC}} \right\rfloor + 1\right)^2 + \\ (\mathbf{NSCC} - \mathbf{NVERTEX} \bmod \mathbf{NSCC}) \cdot \left\lfloor \frac{\mathbf{NVERTEX}}{\mathbf{NSCC}} \right\rfloor^2\end{gathered} \tag{4.148}$$

Proof. For proving part 4.147 of Proposition 101 we proceed by induction on $\mathbf{NSCC}(G)$. If $\mathbf{NSCC}(G) = 1$ then, we have $\mathbf{NARC}(G) \geq \mathbf{NVERTEX}(G)$ (i.e., for one vertex this is true since every vertex has at least one arc, otherwise every vertex v has an arc arriving on v as well as an arc starting from v, thus we have $\mathbf{NARC} \geq \frac{2\cdot\mathbf{NVERTEX}}{2}$). If $\mathbf{NSCC}(G) > 1$ let X be a strongly connected component of G. Then $\mathbf{NARC}(G) \geq \mathbf{NARC}(G[V(G) - X]) + \mathbf{NARC}(G[X])$. By induction hypothesis $\mathbf{NARC}(G[V(G) - X]) \geq |V(G) - X| -$

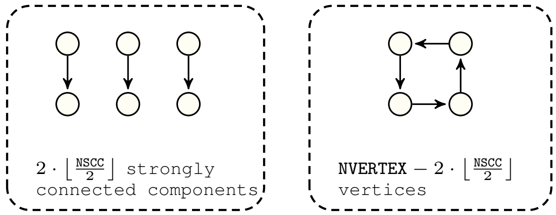


Figure 4.4: Illustration of Proposition 4.147. A graph that achieves the minimum number of arcs according to a fixed number of strongly connected components as well as to a fixed number of vertices ($\texttt{NSCC} = 7, \texttt{NVERTEX} = 10, \texttt{NARC} = 10 - \lfloor \frac{7}{2} \rfloor = 7$)

$\left\lfloor \frac{\mathbf{NSCC}(G[V(G)-X])-1}{2} \right\rfloor$, thus $\mathbf{NARC}(G[V(G)-X]) \geq |V(G)-X| - \left\lfloor \frac{(\mathbf{NSCC}(G)-1)-1}{2} \right\rfloor$. Since $\mathbf{NARC}(G[X]) \geq |X|$ we obtain $\mathbf{NARC}(G) \geq |V(G)| - \left\lfloor \frac{(\mathbf{NSCC}(G)-1)-1}{2} \right\rfloor$, and thus the result holds. □

Proposition 102.

$$\texttt{equivalence} : \mathbf{NVERTEX} > 0 \Rightarrow \mathbf{NSCC} \geq \left\lceil \frac{\mathbf{NVERTEX}^2}{\mathbf{NARC}} \right\rceil \tag{4.149}$$

Proof. As shown in [68], a lower bound for the minimum number of equivalence classes (e.g., strongly connected components) is the independence number of the graph and the right-hand side of Proposition 102 corresponds to a lower bound of the independence number proposed by Turán [421]. □

Proposition 103.

$$\texttt{equivalence} : \mathbf{NVERTEX} > 0 \Rightarrow \mathbf{NSCC} \geq \left\lceil \frac{2 \cdot \mathbf{NVERTEX} - \frac{\mathbf{NARC}-\mathbf{NVERTEX}}{\left\lceil \frac{\mathbf{NARC}-\mathbf{NVERTEX}}{\mathbf{NVERTEX}} \right\rceil}}{\left\lceil \frac{\mathbf{NARC}-\mathbf{NVERTEX}}{\mathbf{NVERTEX}} \right\rceil + 1} \right\rceil \tag{4.150}$$

Proof. See [211] and [174]. □

NARC, NSINK, NVERTEX

Proposition 104.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NSINK}) \cdot \mathbf{NVERTEX} \tag{4.151}$$

Proof. The maximum number of arcs is achieved by the following pattern: for all non-sink vertices we have an arc to all vertices. □

Proposition 105.

$$\mathbf{NARC} \geq \mathbf{NSINK} + max(0, \mathbf{NVERTEX} - 2 \cdot \mathbf{NSINK}) \tag{4.152}$$

Proof. Recall that for $x \in V(G)$, we have that $d^+_G(x) + d^-_G(x) \geq 1$. If x is a sink then $d^-_G(x) \geq 1$, consequently $\mathbf{NARC}(G) \geq \mathbf{NSINK}(G)$. If x is not a sink then $d^+_G(x) \geq 1$, consequently $\mathbf{NARC}(G) \geq |V(G)| - \mathbf{NSINK}(G)$. □

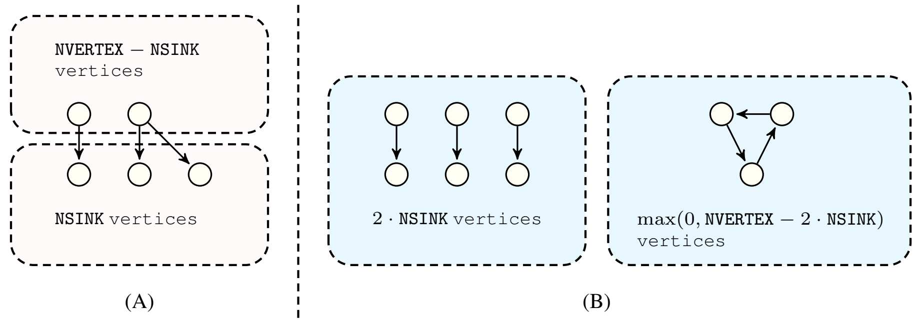


Figure 4.5: Illustration of Proposition 105. Two graphs that achieve the minimum number of arcs according to a fixed number of sinks as well as to a fixed number of vertices: (A) $\texttt{NSINK} = 3, \texttt{NVERTEX} = 5, \texttt{NARC} = 3 + \max(0, 5 - 2 \cdot 3) = 3$; (B) $\texttt{NSINK} = 3, \texttt{NVERTEX} = 9, \texttt{NARC} = 3 + \max(0, 9 - 2 \cdot 3) = 6$.

NARC, NSOURCE, NVERTEX

Proposition 106.

$$\mathbf{NARC} \leq (\mathbf{NVERTEX} - \mathbf{NSOURCE}) \cdot \mathbf{NVERTEX} \tag{4.153}$$

Proof. The maximum number of arcs is achieved by the following pattern: for all non-source vertices we have an arc from all vertices. □

Proposition 107.

$$\mathbf{NARC} \geq \mathbf{NSOURCE} + max(0, \mathbf{NVERTEX} - 2 \cdot \mathbf{NSOURCE}) \tag{4.154}$$

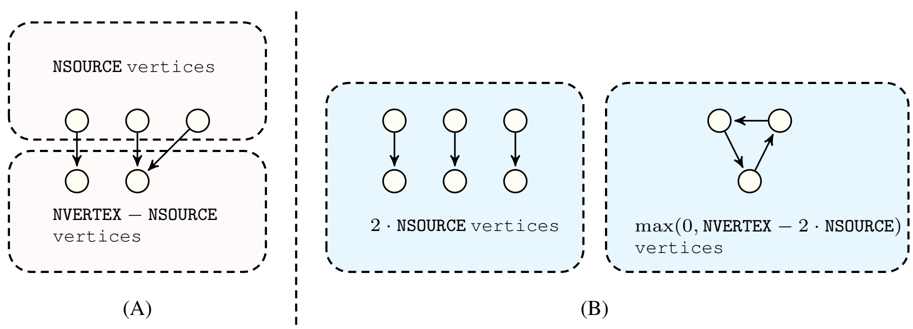


Figure 4.6: Illustration of Proposition 107. Two graphs that achieve the minimum number of arcs according to a fixed number of sources as well as to a fixed number of vertices: (A) $\texttt{NSOURCE} = 3, \texttt{NVERTEX} = 5, \texttt{NARC} = 3 + \max(0, 5 - 2 \cdot 3) = 3$; (B) $\texttt{NSOURCE} = 3, \texttt{NVERTEX} = 9, \texttt{NARC} = 3 + \max(0, 9 - 2 \cdot 3) = 6$.

Proof. Similar to Proposition 105. □

$\boxed{\textbf{NSCC}, \textbf{NSINK}, \textbf{NSOURCE}}$

Proposition 108.

$$\mathbf{NSCC} \geq \mathbf{NSINK} + \mathbf{NSOURCE} \tag{4.155}$$

Proof. Since sinks and sources cannot belong to a circuit and since they cannot coincide (i.e., because isolated vertices are not allowed) the result follows. □

$\boxed{\textbf{NSINK}, \textbf{NSOURCE}, \textbf{NVERTEX}}$

Proposition 109.

$$\mathbf{NVERTEX} \geq \mathbf{NSINK} + \mathbf{NSOURCE} \tag{4.156}$$

Proof. No vertex can be both a source and a sink (isolated vertices are removed). □

Graph invariants involving four parameters of a final graph

MAX_NCC, MIN_NCC, NARC, NCC

Proposition 110. *Let* α *denote* $\max(0, \mathbf{NCC}-1)$.

$$\mathbf{NARC} \leq \alpha \cdot \mathbf{MAX\_NCC}^2 + \mathbf{MIN\_NCC}^2 \quad (4.157)$$

$$\mathbf{arc\_gen} = \mathit{CIRCUIT} : \mathbf{NARC} \leq \alpha \cdot \mathbf{MAX\_NCC} + \mathbf{MIN\_NCC} \quad (4.158)$$

$$\mathbf{arc\_gen} = \mathit{CHAIN} : \mathbf{NARC} \leq \alpha \cdot (2 \cdot \mathbf{MAX\_NCC} - 2) + 2 \cdot \mathbf{MIN\_NCC} - 2 \quad (4.159)$$

$$\mathbf{arc\_gen} \in \{\mathit{CLIQUE}(\leq), \mathit{CLIQUE}(\geq)\} : \mathbf{NARC} \leq \alpha \cdot \frac{\mathbf{MAX\_NCC} \cdot (\mathbf{MAX\_NCC}+1)}{2} + \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC}+1)}{2} \quad (4.160)$$

$$\mathbf{arc\_gen} \in \{\mathit{CLIQUE}(<), \mathit{CLIQUE}(>)\} : \mathbf{NARC} \leq \alpha \cdot \frac{\mathbf{MAX\_NCC} \cdot (\mathbf{MAX\_NCC}-1)}{2} + \frac{\mathbf{MIN\_NCC} \cdot (\mathbf{MIN\_NCC}-1)}{2} \quad (4.161)$$

$$\mathbf{arc\_gen} = \mathit{CLIQUE}(\neq) : \mathbf{NARC} \leq \mathbf{MIN\_NCC}^2 - \mathbf{MIN\_NCC} + \alpha \cdot (\mathbf{MAX\_NCC}^2 - \mathbf{MAX\_NCC}) \quad (4.162)$$

$$\mathbf{arc\_gen} = \mathit{CYCLE} : \mathbf{NARC} \leq 2 \cdot \alpha \cdot \mathbf{MAX\_NCC} + 2 \cdot \mathbf{MIN\_NCC} \quad (4.163)$$

$$\mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NARC} \leq \alpha \cdot (\mathbf{MAX\_NCC} - 1) + \mathbf{MIN\_NCC} - 1 \quad (4.164)$$

Proof. We construct $\mathbf{NCC} - 1$ connected components with **MAX_NCC** vertices and one connected component with **MIN_NCC** vertices. n^2 corresponds to the maximum number of arcs in a connected component. n, $2 \cdot n - 2$, $\frac{n \cdot (n+1)}{2}$, $\frac{n \cdot (n+1)}{2}$, $\frac{n \cdot (n-1)}{2}$, $\frac{n \cdot (n-1)}{2}$, $n^2 - n$, $2 \cdot n$ and $n - 1$ respectively correspond to the maximum number of arcs in a connected component of n vertices according to the fact that we use the arc generator $\mathit{CIRCUIT}$, CHAIN, $\mathit{CLIQUE}(\leq)$ $\mathit{CLIQUE}(\geq)$ $\mathit{CLIQUE}(<)$ $\mathit{CLIQUE}(>)$ $\mathit{CLIQUE}(\neq)$ CYCLE or PATH. ☐

Proposition 111.

$$\mathbf{NCC} > 0 \Rightarrow \mathbf{NARC} \geq (\mathbf{NCC}-1) \cdot \max(1, \mathbf{MIN\_NCC}-1) + \max(1, \mathbf{MAX\_NCC}-1) \quad (4.165)$$

$$\mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NARC} \geq \max(0, \mathbf{NCC}-1) \cdot (\mathbf{MIN\_NCC}-1) + \mathbf{MAX\_NCC} - 1 \quad (4.166)$$

Proof. (4.165) We construct $\mathbf{NCC} - 1$ connected components with **MIN_NCC** vertices and one connected component with **MAX_NCC** vertices. The quantity $\max(1, n-1)$ corresponds to the minimum number of arcs in a connected component of n $(n > 0)$ vertices. ☐

MAX_NCC, MIN_NCC, NCC, NVERTEX

Proposition 112.

$$\mathbf{NVERTEX} \leq \max(0, \mathbf{NCC} - 1) \cdot \mathbf{MAX\_NCC} + \mathbf{MIN\_NCC} \tag{4.167}$$

Proof. Derived from the definitions of MIN_NCC and MAX_NCC. □

Proposition 113.

$$\mathbf{NVERTEX} \geq \max(0, \mathbf{NCC} - 1) \cdot \mathbf{MIN\_NCC} + \mathbf{MAX\_NCC} \tag{4.168}$$

Proof. Derived from the definitions of MIN_NCC and MAX_NCC. □

MAX_NCC, NARC, NSOURCE, NVERTEX

Proposition 114.

$$\mathbf{NSINK} + \mathbf{NSOURCE} \leq \mathbf{NCC} \cdot \max(0, \mathbf{MAX\_NCC} - 1) \tag{4.169}$$

Proof. Since a connected component contains at most MAX_NCC vertices and since it does not contain any isolated vertex and since a same vertex cannot be both a sink and a source a connected component involves at most $\mathbf{MAX\_NCC} - 1$ sinks and sources all together. Thus the result follows. □

MAX_NSCC, MIN_NSCC, NARC, NSCC

Proposition 115.

$$\begin{aligned} \mathbf{NARC} \leq\ & \max(0, \mathbf{NSCC} - 1) \cdot \mathbf{MAX\_NSCC}^2 + \mathbf{MIN\_NSCC}^2 + \\ & \max(0, \mathbf{NSCC} - 1) \cdot \mathbf{MIN\_NSCC} \cdot \mathbf{MAX\_NSCC} + \\ & \mathbf{MAX\_NSCC}^2 \cdot \tfrac{\max(0, \mathbf{NSCC}-2) \cdot \max(0, \mathbf{NSCC}-1)}{2} \end{aligned} \tag{4.170}$$

Proof. We assume that we have at least two strongly connected components (the case with one being obvious). Let $(SCC_i)_{i \in [\mathbf{NCC}(G)]}$ be the family of strongly connected components of G. Then $|E(G)| \leq \sum_{i \in [\mathbf{NCC}(G)]} |E(G[SCC_i])| + k$, where k is the number of arcs between the distinct strongly connected components of G. For any strongly connected component SCC_i the number of arcs it has with the other strongly connected components is bounded by $|SCC_i| \cdot (|V(G) - SCC_i|)$. Consequently, $k \leq \frac{1}{2} \cdot \sum_{i \in [\mathbf{NCC}(G)]} |SCC_i| \cdot (|V(G) - SCC_i|)$. W.l.o.g. we assume $|SCC_1| = \mathbf{MIN\_NCC}$. Then we get $k \leq \frac{1}{2} \cdot (\mathbf{MIN\_NCC} \cdot (\mathbf{NCC} - 1) \cdot \mathbf{MAX\_NCC} + \mathbf{MAX\_NCC} \cdot ((\mathbf{NCC} - 2) \cdot \mathbf{MAX\_NCC} + \mathbf{MIN\_NCC}))$. □

Proposition 116.

$$\mathbf{NARC} \geq \max(0, \mathbf{NSCC} - 1) \cdot \mathbf{MIN\_NSCC} + \mathbf{MAX\_NSCC} \tag{4.171}$$

Proof. Let $(SCC_i)_{i \in [\mathbf{NCC}(G)]}$ be the family of strongly connected components of G, as $|E(G)| \geq \sum_{i \in [\mathbf{NCC}(G)]} |E(G[SCC_i])|$, we obtain the result since in a strongly connected graph the number of edges is at least its number of vertices. □

MAX_NSCC, MIN_NSCC, NSCC, NVERTEX

Proposition 117.

$$\mathbf{NVERTEX} \leq \max(0, \mathbf{NSCC}-1) \cdot \mathbf{MAX\_NSCC} + \mathbf{MIN\_NSCC} \tag{4.172}$$

Proof. Derived from the definitions of MIN_NSCC and MAX_NSCC. □

Proposition 118.

$$\mathbf{NVERTEX} \geq \max(0, \mathbf{NSCC}-1) \cdot \mathbf{MIN\_NSCC} + \mathbf{MAX\_NSCC} \tag{4.173}$$

Proof. Derived from the definitions of MIN_NSCC and MAX_NSCC. □

MIN_NCC, NARC, NCC, NVERTEX

Proposition 119. *Let* α*,* β *and* γ *respectively denote* $\max(0, \mathbf{NCC}-1)$*,* $\mathbf{NVERTEX} - \alpha \cdot \mathbf{MIN\_NCC}$ *and* $\mathbf{MIN\_NCC}$*.*

$$\mathbf{NARC} \leq \alpha \cdot \gamma^2 + \beta^2 \tag{4.174}$$

$$\mathbf{arc\_gen} \in \{CLIQUE(\leq), CLIQUE(\geq)\} : \mathbf{NARC} \leq \alpha \cdot \frac{\gamma \cdot (\gamma+1)}{2} + \frac{\beta \cdot (\beta+1)}{2} \tag{4.175}$$

$$\mathbf{arc\_gen} \in \{CLIQUE(<), CLIQUE(>)\} : \mathbf{NARC} \leq \alpha \cdot \frac{\gamma \cdot (\gamma-1)}{2} + \frac{\beta \cdot (\beta-1)}{2} \tag{4.176}$$

$$\mathbf{arc\_gen} = CLIQUE(\neq) : \mathbf{NARC} \leq \alpha \cdot \gamma \cdot (\gamma-1) + \beta \cdot (\beta-1) \tag{4.177}$$

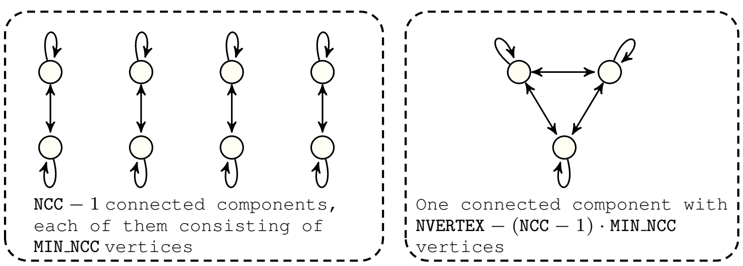


Figure 4.7: Illustration of Proposition 119(4.174). Graph that achieves the maximum number of arcs according to a minimum number of vertices in a connected component, to a number of connected components, as well as to a fixed number of vertices ($\mathtt{MIN\_NCC} = 2, \mathtt{NCC} = 5, \mathtt{NVERTEX} = 11, \mathtt{NARC} = (11-(5-1)\cdot 2)^2 + (5-1)\cdot 2^2 = 25$)

Proof. For proving inequality 4.174 we proceed by induction on the number of vertices of G. First note that if all the connected components are reduced to one element the result is obvious. Thus we assume that the number of vertices in the maximal sized connected component of G is at least 2. Let x be an element of the maximal sized connected component of G. Then, $G-x$ satisfies $\alpha(G-x) = \alpha(G)$, $\gamma(G-x) = \gamma(G)$ and $\beta(G-x) = \beta(G)-1$. Since by induction hypothesis $|E(G-x)| \leq \alpha(G-x)\cdot\gamma(G-x)^2 + \beta(G-x)^2$, and since the number of arcs of G incident to x is at most $2\cdot(\beta(G)-1)+1$, we have that $|E(G)| \leq \alpha(G)\cdot\gamma(G)^2 + (\beta(G)-1)^2 + 2\cdot(\beta(G)-1)+1$. And thus the result follows. □

NARC, NCC, NSCC, NVERTEX

Proposition 120.

$$\mathbf{NARC} \leq \mathbf{NCC} - 1 + (\mathbf{NVERTEX} - \mathbf{NSCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 1) + \frac{(\mathbf{NSCC} - \mathbf{NCC} + 1) \cdot (\mathbf{NSCC} - \mathbf{NCC})}{2} \quad (4.178)$$

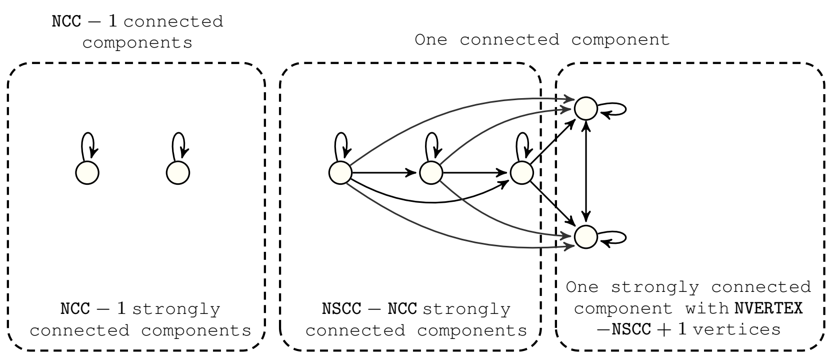


Figure 4.8: Illustration of Proposition 120. A graph that achieves the maximum number of arcs according to a fixed number of connected components, to a fixed number of strongly connected components as well as to a fixed number of vertices ($\texttt{NCC} = 3, \texttt{NSCC} = 6, \texttt{NVERTEX} = 7, \texttt{NARC} = 3 - 1 + (7-6+1) \cdot (7-3+1) + \frac{(6-3+1)\cdot(6-3)}{2} = 18$)

Proof. We proceed by induction on $T(G) = \mathbf{NVERTEX}(G) - |X| - (\mathbf{NCC}(G) - 1)$, where X is any connected component of G of maximum cardinality. For $T(G) = 0$ then either $\mathbf{NCC}(G) = 1$ and thus the formula is clearly true, by Proposition 4.145 or all the connected components of G, but possibly X, are reduced to one element. Since isolated vertices are not allowed, again by Proposition 4.145 applied on $G[X]$, the formula holds indeed $\mathbf{NVERTEX}(G[X]) = \mathbf{NVERTEX}(G) - (\mathbf{NCC}(G) - 1)$ and $\mathbf{NSCC}(G[X]) = \mathbf{NSCC}(G) - (\mathbf{NCC}(G) - 1)$.

Assume that $T(G) \geq 1$. Then there exists Y, a connected component of G distinct from X, with more than one vertex.

- Firstly assume that $G[Y]$ is strongly connected. Let $y \in Y$ and let G' be the graph such that $V(G') = V(G)$ and $E(G')$ is defined by:
  - For all Z connected components of G distinct from X and Y we have $G'[Z] = G[Z]$.
  - With $X' = X \cup (Y - \{y\})$ and $Y' = \{y\}$, we have $E(G'[Y']) = \{(y, y)\}$, $E(G'[X']) = E(G[X]) \cup \{(z, x) : z \in Y - \{y\}, x \in X\} \cup \{(z, t) : z, t \in Y - \{y\}\}$.

  Clearly we have that $|E(G')| - |E(G)| \geq (|Y| - 1) \cdot |X| - 2 \cdot (|Y| - 1)$ and since $|X| \geq |Y| \geq 2$, the difference is positive or null. Now as $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G)$, $\mathbf{NCC}(G') = \mathbf{NCC}(G)$, $\mathbf{NSCC}(G') = \mathbf{NSCC}(G)$ (since $G'[Y - \{y\}]$ is strongly connected because $E(G'[Y - \{y\}]) = \{(z, t) : z, t \in Y - \{y\}\}$

and since the reduced graph of the strongly connected components of $G'[X']$ is exactly the reduced graph of the strongly connected components of $G[X]$ to which a unique source has been added) and as $T(G') \leq T(G) - 1$, the result holds by induction hypothesis.

- Secondly assume that $G[Y]$ is not strongly connected. Let $Z \subset Y$ such that Z is a strongly connected component of $G[Y]$ corresponding to a source in the reduced graph of the strongly connected components of $G[Y]$. Let G' be the graph such that $V(G') = V(G)$ and $E(G')$ is defined by:
  - For all W connected components of G distinct from X and Y we have $G'[W] = G[W]$.
  - With $X' = X \cup Z$ and $Y' = Y - Z$, we have $E(G'[Y']) = E(G[Y'])$ if $|Y'| > 1$ and $E(G'[Y']) = \{(y, y)\}$ if $Y' = \{y\}$. $E(G'[X']) = E(G[X]) \cup \{(z, x) : z \in Z, x \in X\}$.

  Clearly we have that $|E(G')| - |E(G)| \geq |Z| \cdot |X| - |Z| \cdot (|Y| - |Z|)$ and since $|X| > |Y| - |Z|$, the difference is strictly positive. Now as $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G)$, $\mathbf{NCC}(G') = \mathbf{NCC}(G)$, $\mathbf{NSCC}(G') = \mathbf{NSCC}(G)$ and as $T(G') \leq T(G) - 1$, the result holds by induction hypothesis.

□

Proposition 121.

$$\mathbf{NARC} \geq \mathbf{NVERTEX} - \max(0, \min(\mathbf{NCC}, \mathbf{NSCC} - \mathbf{NCC})) \tag{4.179}$$

Proof. We prove that the invariant is valid for any digraph G. First notice that for an operational behaviour, since we cannot assume that Proposition 55 (i.e., $\mathbf{NCC}(G) \leq \mathbf{NSCC}(G)$) was already triggered, we use the $\max$ operator. But since any strongly connected component is connected, then $\mathbf{NSCC}(G) - \mathbf{NCC}(G)$ is never negative. Consequently we only show by induction on $\mathbf{NSCC}(G)$ that $\mathbf{NARC}(G) \geq \mathbf{NVERTEX}(G) - \min(\mathbf{NCC}(G), \mathbf{NSCC}(G) - \mathbf{NCC}(G))$. To begin notice that if X is a strongly (non void) connected component then either $\mathbf{NARC}(G[X]) \geq |X|$ or $\mathbf{NARC}(G[X]) = 0$ and in this latter case we have that both $|X| = 1$ and X is strictly included in a connected component of G (recall that isolated vertices are not allowed). Thus we can directly assume that $\mathbf{NSCC}(G) = k > 1$.

First, consider that there exists a connected component of G, say X, which is also strongly connected. Let $G' = G - X$, consequently we have $\mathbf{NSCC}(G') = \mathbf{NSCC}(G) - 1$, $\mathbf{NCC}(G') = \mathbf{NCC}(G) - 1$, $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G) - |X|$, and $\mathbf{NARC}(G) \geq |X| + \mathbf{NARC}(G')$. Then $\mathbf{NARC}(G) \geq |X| + \mathbf{NVERTEX}(G') - \min(\mathbf{NCC}(G'), \mathbf{NSCC}(G') - \mathbf{NCC}(G'))$ and thus $\mathbf{NARC}(G) \geq \mathbf{NVERTEX}(G) - \min(\mathbf{NCC}(G) - 1, \mathbf{NSCC}(G) - \mathbf{NCC}(G))$, which immediately gives the result.

Second consider that any strongly connected component is strictly included in a connected component of G. Then, either there exists a strongly connected component X such that $|X| \geq 2$. Let $G' = G - X$, consequently we have $\mathbf{NSCC}(G') = \mathbf{NSCC}(G) - 1$, $\mathbf{NCC}(G') = \mathbf{NCC}(G)$, $\mathbf{NVERTEX}(G') = \mathbf{NVERTEX}(G) - |X|$, and $\mathbf{NARC}(G) \geq |X| + 1 + \mathbf{NARC}(G')$. Then $\mathbf{NARC}(G) \geq |X| + 1 + \mathbf{NVERTEX}(G') - \min(\mathbf{NCC}(G'), \mathbf{NSCC}(G') - \mathbf{NCC}(G'))$ and thus $\mathbf{NARC}(G) \geq \mathbf{NVERTEX}(G) + 1 - \min(\mathbf{NCC}(G), \mathbf{NSCC}(G) - \mathbf{NCC}(G) + 1)$, which immediately gives the result. Or, all the strongly connected components are reduced to one element, so we have $\mathbf{NSCC}(G) = \mathbf{NVERTEX}(G)$, and thus we obtain that $\mathbf{NVERTEX}(G) - \min(\mathbf{NCC}(G), \mathbf{NSCC}(G) - \mathbf{NCC}(G)) = \min(\mathbf{NCC}(G), \mathbf{NVERTEX}(G) - \mathbf{NCC}(G))$, which gives the result by, for example, Proposition 99 (4.143). □

This bound is tight: take, for example, any circuit.

$\boxed{\mathbf{NARC}, \mathbf{NSINK}, \mathbf{NSOURCE}, \mathbf{NVERTEX}}$

Proposition 122.

$$\begin{aligned}\mathbf{NARC} \leq \mathbf{NVERTEX}^2 &- \mathbf{NVERTEX} \cdot \mathbf{NSOURCE} \\ &- \mathbf{NVERTEX} \cdot \mathbf{NSINK} + \mathbf{NSOURCE} \cdot \mathbf{NSINK}\end{aligned} \tag{4.180}$$

Proof. Since the maximum number of arcs of a digraph is $\mathbf{NVERTEX}^2$, and since:

- No vertex can have a source as a successor we lose $\mathbf{NVERTEX} \cdot \mathbf{NSOURCE}$ arcs,
- No sink can have a successor we lose $\mathbf{NVERTEX} \cdot \mathbf{NSINK}$ arcs.

In these two sets of arcs we count twice the arcs from the sinks to the sources, so we finally get a maximum number of arcs corresponding to the right-hand side of the inequality to prove. □

Graph invariants involving five parameters of a final graph

$\boxed{\mathbf{MAX\_NCC}, \mathbf{MIN\_NCC}, \mathbf{NARC}, \mathbf{NCC}, \mathbf{NVERTEX}}$

Proposition 123.
Let:

- $\Delta = \mathbf{NVERTEX} - \mathbf{NCC} \cdot \mathbf{MIN\_NCC}$,
- $\delta = \lfloor \frac{\Delta}{\max(1, \mathbf{MAX\_NCC} - \mathbf{MIN\_NCC})} \rfloor$,
- $r = \Delta \bmod \max(1, \mathbf{MAX\_NCC} - \mathbf{MIN\_NCC})$,
- $\epsilon = (r > 0)$.

$$\Delta = 0 \vee (\mathbf{MAX\_NCC} \neq \mathbf{MIN\_NCC} \wedge \delta + \epsilon \leq \mathbf{NCC}) \tag{4.181}$$

$$\mathbf{NARC} \leq (\mathbf{NCC} - \delta - \epsilon) \cdot \mathbf{MIN\_NCC}^2 + \epsilon \cdot (\mathbf{MIN\_NCC} + r)^2 + \delta \cdot \mathbf{MAX\_NCC}^2 \tag{4.182}$$

Proposition 123 is currently a conjecture.

$\boxed{\mathbf{MIN\_NCC}, \mathbf{NARC}, \mathbf{NCC}, \mathbf{NSCC}, \mathbf{NVERTEX}}$

Proposition 124.

$$\begin{aligned} \mathbf{NARC} \leq & (\mathbf{NCC} - 1) \cdot \max(1, (\mathbf{MIN\_NCC} - 1)) + \\ & (\mathbf{NVERTEX} - \mathbf{NSCC} + 1) \cdot (\mathbf{NVERTEX} - \mathbf{NCC} + 1) + \\ & \frac{(\mathbf{NSCC} - \mathbf{NCC} + 1) \cdot (\mathbf{NSCC} - \mathbf{NCC})}{2} \end{aligned} \tag{4.183}$$

Proposition 124 is currently a conjecture.

Graph invariants relating two parameters of two final graphs

$\boxed{\mathbf{MAX\_NCC_1, MIN\_NCC_1}}$

Proposition 125.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \mathbf{MIN\_NCC}_1 \notin [\mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC}_1, \mathbf{MAX\_NCC}_1 - 1]\end{array} \tag{4.184}$$

Proof. We show that the conjunction $\mathbf{MIN\_NCC}_1 \geq \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC}_1$ and $\mathbf{MIN\_NCC}_1 \leq \mathbf{MAX\_NCC}_1 - 1$ leads to a contradiction.

Since $\mathbf{MIN\_NCC}_1 \leq \mathbf{MAX\_NCC}_1 - 1$ we have that $\mathbf{MIN\_NCC}_1 \neq \mathbf{MAX\_NCC}_1$ and the minimum required size for the different groups is $\mathbf{MIN\_NCC}_1 + 1 + \mathbf{MAX\_NCC}_1$. This minimum required size should not exceed the number of vertices $\mathbf{NVERTEX}_{\texttt{INITIAL}}$ of the initial graph. But since, by hypothesis, $\mathbf{MIN\_NCC}_1 \geq \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC}_1$, this is impossible. □

$\boxed{\mathbf{MAX\_NCC_2, MIN\_NCC_2}}$

Proposition 126.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \mathbf{MIN\_NCC}_2 \notin [\mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC}_2, \mathbf{MAX\_NCC}_2 - 1]\end{array} \tag{4.185}$$

Proof. Similar to Proposition 125. □

$\boxed{\mathbf{MAX\_NCC_1, NCC_2}}$

Proposition 127.

$$\texttt{vpartition} : \mathbf{MAX\_NCC}_1 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_2 > 0 \tag{4.186}$$

$$\texttt{apartition} : \mathbf{MAX\_NCC}_1 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_2 > 0 \tag{4.187}$$

Proof. (4.186) Since we have the precondition `vpartition`, we know that each vertex of the initial graph belongs to the first or to the second final graphs (but not to both).

1. On the one hand, if the largest connected component of the first final graph cannot contain all the vertices of the initial graph, then the second final graph has at least one connected component.
2. On the other hand, if the second final graph has at least one connected component then the largest connected component of the first final graph cannot be equal to the initial graph.

(4.187) holds for a similar reason. □

$\boxed{\mathbf{MAX\_NCC_2, NCC_1}}$

Proposition 128.

$$\texttt{vpartition} : \mathbf{MAX\_NCC}_2 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_1 > 0 \tag{4.188}$$

$$\texttt{apartition} : \mathbf{MAX\_NCC}_2 < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC}_1 > 0 \tag{4.189}$$

Proof. Similar to Proposition 127. □

$\boxed{\mathbf{MIN\_NCC_1, NCC_2}}$

Proposition 129.

$$\texttt{vpartition} : \mathbf{MIN\_NCC_1} < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC_2} > 0 \tag{4.190}$$

Proof. Since we have the precondition `vpartition`, we know that each vertex of the initial graph belongs to the first or to the second final graphs (but not to both).

1. On the one hand, if the smallest connected component of the first final graph cannot contain all the vertices of the initial graph, then the second final graph has at least one connected component.
2. On the other hand, if the second final graph has at least one connected component then the smallest connected component of the first final graph cannot be equal to the initial graph.

□

$\boxed{\mathbf{MIN\_NCC_2, NCC_1}}$

Proposition 130.

$$\texttt{vpartition} : \mathbf{MIN\_NCC_2} < \mathbf{NVERTEX}_{\texttt{INITIAL}} \Leftrightarrow \mathbf{NCC_1} > 0 \tag{4.191}$$

Proof. Similar to Proposition 129. □

$\boxed{\mathbf{NARC_1, NARC_2}}$

Proposition 131.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NARC_1} + \mathbf{NARC_2} = \mathbf{NVERTEX}_{\texttt{INITIAL}} - 1 \tag{4.192}$$

Proof. Holds since each arc of the initial graph belongs to one of the two final graphs and since the initial graph has $\mathbf{NVERTEX}_{\texttt{INITIAL}} - 1$ arcs. □

$\boxed{\mathbf{NCC_1, NCC_2}}$

Proposition 132.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = \mathit{PATH} : |\mathbf{NCC_1} - \mathbf{NCC_2}| \leq 1 \tag{4.193}$$

$$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : |\mathbf{NCC_1} - \mathbf{NCC_2}| \leq 1 \tag{4.194}$$

Proof. Holds because the two initial graphs correspond to a path and because consecutive connected components do not come from the same graph constraint. □

Proposition 133.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = \mathit{PATH} : \mathbf{NCC_1} + \mathbf{NCC_2} < \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{4.195}$$

Proof. Holds because the initial graph is a path. □

$\boxed{\mathbf{NVERTEX_1, NVERTEX_2}}$

Proposition 134.

$$\texttt{vpartition} : \mathbf{NVERTEX_1} + \mathbf{NVERTEX_2} = \mathbf{NVERTEX}_{\texttt{INITIAL}} \tag{4.196}$$

Proof. By definition of `vpartition` each vertex of the initial graph belongs to one of the two final graphs (but not to both). □

Graph invariants relating three parameters of two final graphs

$\boxed{\mathbf{MAX\_NCC_1, MIN\_NCC_1, MIN\_NCC_2}}$

Proposition 135.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : \\ \max(2, \mathbf{MIN\_NCC_1}) + \max(3, \mathbf{MIN\_NCC_1} + 1, \mathbf{MAX\_NCC_1}) + \\ \max(2, \mathbf{MIN\_NCC_2}) - 2 > \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{MIN\_NCC_1} = \mathbf{MAX\_NCC_1}\end{array} \tag{4.197}$$

Proof. The quantity $\max(2, \mathbf{MIN\_NCC_1}) + \max(3, \mathbf{MIN\_NCC_1} + 1, \mathbf{MAX\_NCC_1}) + \max(2, \mathbf{MIN\_NCC_2}) - 2$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size $\mathbf{MIN\_NCC_1}$ and $\mathbf{MAX\_NCC_1}$ such that $\mathbf{MAX\_NCC_1}$ is strictly greater than $\mathbf{MIN\_NCC_1}$. If this quantity is greater than the total number of variables we have that $\mathbf{MIN\_NCC_1} = \mathbf{MAX\_NCC_1}$. □

Proposition 136.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \max(1, \mathbf{MIN\_NCC_1}) + \max(2, \mathbf{MIN\_NCC_1} + 1, \mathbf{MAX\_NCC_1}) + \\ \max(1, \mathbf{MIN\_NCC_2}) > \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{MIN\_NCC_1} = \mathbf{MAX\_NCC_1}\end{array} \tag{4.198}$$

Proof. The quantity $\max(1, \mathbf{MIN\_NCC_1}) + \max(2, \mathbf{MIN\_NCC_1} + 1, \mathbf{MAX\_NCC_1}) + \max(1, \mathbf{MIN\_NCC_2})$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size $\mathbf{MIN\_NCC_1}$ and $\mathbf{MAX\_NCC_1}$ such that $\mathbf{MAX\_NCC_1}$ is strictly greater than $\mathbf{MIN\_NCC_1}$. If this quantity is greater than the total number of variables we have that $\mathbf{MIN\_NCC_1} = \mathbf{MAX\_NCC_1}$. □

Proposition 137.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \mathbf{MIN\_NCC_2} \notin \left[\max\left(\mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC_1} - \mathbf{MIN\_NCC_1} + 1, \left\lfloor \frac{\mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC_1} + 2}{2} \right\rfloor\right), \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC_1} - 1\right]\end{array} \tag{4.199}$$

Proof. A value v is not a possible number of vertices for the smallest connected component of type 2 if the following two conditions hold:

- $v + \mathbf{MAX\_NCC_1}$ does not allow to cover all the vertices of the initial graph: we need at least one extra connected component of type 1 or 2.
- If we add an additional connected component of type 1 or 2 we exceed the number of vertices of the initial graph.

□

$\boxed{\mathbf{MAX\_NCC_2, MIN\_NCC_2, MIN\_NCC_1}}$

Proposition 138.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : \\ \max(2, \mathbf{MIN\_NCC_2}) + \max(3, \mathbf{MIN\_NCC_2} + 1, \mathbf{MAX\_NCC_2}) + \\ \max(2, \mathbf{MIN\_NCC_1}) - 2 > \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{MIN\_NCC_2} = \mathbf{MAX\_NCC_2}\end{array} \quad (4.200)$$

Proof. Similar to Proposition 135. □

Proposition 139.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \max(1, \mathbf{MIN\_NCC_2}) + \max(2, \mathbf{MIN\_NCC_2} + 1, \mathbf{MAX\_NCC_2}) + \\ \max(1, \mathbf{MIN\_NCC_1}) > \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{MIN\_NCC_2} = \mathbf{MAX\_NCC_2}\end{array} \quad (4.201)$$

Proof. Similar to Proposition 136. □

Proposition 140.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \mathbf{MIN\_NCC_1} \notin \Bigg[\max\Bigg(\mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC_2} - \mathbf{MIN\_NCC_2} + 1, \\ \qquad \left\lfloor \dfrac{\mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC_2} + 2}{2} \right\rfloor\Bigg), \\ \qquad \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC_2} - 1\Bigg]\end{array} \quad (4.202)$$

Proof. Similar to Proposition 137. □

$\boxed{\mathbf{MAX\_NCC_1, MIN\_NCC_1, NVERTEX_2}}$

Proposition 141.

$$\begin{array}{l}\texttt{vpartition} : \mathbf{MIN\_NCC_1} = \mathbf{MAX\_NCC_1} \wedge \mathbf{MIN\_NCC_1} \bmod 2 = 0 \Rightarrow \\ \qquad \mathbf{NVERTEX_2} \bmod 2 = \mathbf{NVERTEX}_{\texttt{INITIAL}} \bmod 2\end{array} \quad (4.203)$$

Proof. If the number of vertices of the first graph is even then the number of vertices of the second graph has the same parity as the number of vertices of the initial graph (since a vertex of the initial graph belongs either to the first graph, either to the second graph (but not to both). □

$\boxed{\mathbf{MAX\_NCC_2, MIN\_NCC_2, NVERTEX_1}}$

Proposition 142.

$$\begin{array}{l}\texttt{vpartition} : \mathbf{MIN\_NCC_2} = \mathbf{MAX\_NCC_2} \wedge \mathbf{MIN\_NCC_2} \bmod 2 = 0 \Rightarrow \\ \qquad \mathbf{NVERTEX_1} \bmod 2 = \mathbf{NVERTEX}_{\texttt{INITIAL}} \bmod 2\end{array} \quad (4.204)$$

Proof. Similar to Proposition 141. □

$\boxed{\mathbf{MIN\_NCC_1}, \mathbf{NARC_2}, \mathbf{NCC_1}}$

Proposition 143.

$$\begin{aligned}&\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{NVERTEX}_{\texttt{INITIAL}} > 0 : \\ &\quad \mathbf{NCC_1} = 1 \Leftrightarrow \mathbf{MIN\_NCC_1} + \mathbf{NARC_2} = \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{aligned} \tag{4.205}$$

Proof. When $\mathbf{MIN\_NCC_1} + \mathbf{NARC_2} = \mathbf{NVERTEX}_{\texttt{INITIAL}}$ there is no more room for an extra connected component for the first final graph. □

$\boxed{\mathbf{MIN\_NCC_1}, \mathbf{NARC_2}, \mathbf{NCC_1}}$

Proposition 144.

$$\begin{aligned}&\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{NVERTEX}_{\texttt{INITIAL}} > 0 : \\ &\quad \mathbf{NCC_2} = 1 \Leftrightarrow \mathbf{MIN\_NCC_2} + \mathbf{NARC_1} = \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{aligned} \tag{4.206}$$

Proof. Similar to Proposition 143. □

Graph invariants relating four parameters of two final graphs

$\boxed{\mathbf{MAX\_NCC_1, MIN\_NCC_1, MIN\_NCC_2, NCC_1}}$

Proposition 145.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : \\ \max(2, \mathbf{MIN\_NCC_1}) + \max(2, \mathbf{MAX\_NCC_1}) + \max(2, \mathbf{MIN\_NCC_2}) - 2 > \\ \quad \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{NCC_1} \leq 1\end{array} \tag{4.207}$$

Proof. The quantity $\max(2, \mathbf{MIN\_NCC_1}) + \max(2, \mathbf{MAX\_NCC_1}) + \max(2, \mathbf{MIN\_NCC_2}) - 2$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size $\mathbf{MIN\_NCC_1}$ and $\mathbf{MAX\_NCC_1}$. If this quantity is greater than the total number of variables we have that $\mathbf{NCC_1} \leq 1$. □

Proposition 146.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \max(1, \mathbf{MIN\_NCC_1}) + \max(1, \mathbf{MAX\_NCC_1}) + \max(1, \mathbf{MIN\_NCC_2}) > \\ \quad \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{NCC_1} \leq 1\end{array} \tag{4.208}$$

Proof. The quantity $\max(1, \mathbf{MIN\_NCC_1}) + \max(1, \mathbf{MAX\_NCC_1}) + \max(1, \mathbf{MIN\_NCC_2})$ corresponds to the minimum number of variables needed for building two non-empty connected components of respective size $\mathbf{MIN\_NCC_1}$ and $\mathbf{MAX\_NCC_1}$. If this quantity is greater than the total number of variables we have that $\mathbf{NCC_1} \leq 1$. □

$\boxed{\mathbf{MAX\_NCC_2, MIN\_NCC_2, MIN\_NCC_1, NCC_2}}$

Proposition 147.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH : \\ \max(2, \mathbf{MIN\_NCC_2}) + \max(2, \mathbf{MAX\_NCC_2}) + \max(2, \mathbf{MIN\_NCC_1}) - 2 > \\ \quad \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{NCC_2} \leq 1\end{array} \tag{4.209}$$

Proof. Similar to Proposition 145. □

Proposition 148.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected} : \\ \max(1, \mathbf{MIN\_NCC_2}) + \max(1, \mathbf{MAX\_NCC_2}) + \max(1, \mathbf{MIN\_NCC_1}) > \\ \quad \mathbf{NVERTEX}_{\texttt{INITIAL}} \Rightarrow \mathbf{NCC_2} \leq 1\end{array} \tag{4.210}$$

Proof. Similar to Proposition 146. □

$\boxed{\mathbf{MAX\_NCC_1, MIN\_NCC_1, MIN\_NCC_2, NVERTEX_2}}$

Proposition 149.

$$\texttt{vpartition} \land \texttt{consecutive\_loops\_are\_connected}:$$
$$\mathbf{MIN\_NCC_2} \notin \left[\left\lfloor \frac{\mathbf{NVERTEX_2}}{2} \right\rfloor + 1, \right.$$
$$\left. \mathbf{NVERTEX_{\texttt{INITIAL}}} - \mathbf{MIN\_NCC_1} - \mathbf{MAX\_NCC_1} - 1 \right] \quad (4.211)$$

Proof. First, note that, when $\mathbf{NCC_2} > 1$, we have that $\mathbf{MIN\_NCC_2} \leq \left\lfloor \frac{\mathbf{NVERTEX_2}}{2} \right\rfloor$. Second, note that, when $\mathbf{NCC_2} \leq 1$, we have that $\mathbf{MIN\_NCC_2} \geq \mathbf{NVERTEX_{\texttt{INITIAL}}} - \mathbf{MIN\_NCC_1} - \mathbf{MAX\_NCC_1}$. Since $\mathbf{NCC_2}$ has to have at least one value the result follows. $\square$

$\boxed{\mathbf{MAX\_NCC_2, MIN\_NCC_2, MIN\_NCC_1, NVERTEX_1}}$

Proposition 150.

$$\texttt{vpartition} \land \texttt{consecutive\_loops\_are\_connected}:$$
$$\mathbf{MIN\_NCC_1} \notin \left[\left\lfloor \frac{\mathbf{NVERTEX_1}}{2} \right\rfloor + 1, \right.$$
$$\left. \mathbf{NVERTEX_{\texttt{INITIAL}}} - \mathbf{MIN\_NCC_2} - \mathbf{MAX\_NCC_2} - 1 \right] \quad (4.212)$$

Proof. Similar to Proposition 149. $\square$

Graph invariants relating five parameters of two final graphs

$\boxed{\mathbf{MAX\_NCC}_1, \mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_1}$

Proposition 151.

$$\begin{aligned}&\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:\\ &\mathbf{MIN\_NCC}_1 \cdot \max(0, \mathbf{NCC}_1 - 1) + \mathbf{MAX\_NCC}_1 + \\ &\mathbf{MIN\_NCC}_2 \cdot \max(0, \mathbf{NCC}_1 - 2) + \mathbf{MAX\_NCC}_2 \leq \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{aligned} \tag{4.213}$$

Proof. The left-hand side of 151 corresponds to the minimum number of vertices of the two final graphs provided that we build the smallest possible connected components. □

Proposition 152.

$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:$

$$\mathbf{NCC}_1 \leq (\overline{\mathbf{MAX\_NCC}_1} > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + \left(\alpha \bmod \beta \geq \max(1, \underline{\mathbf{MIN\_NCC}_1})\right)$$

$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \max(1, \underline{\mathbf{MAX\_NCC}_1}) - \max(1, \underline{\mathbf{MAX\_NCC}_2})), \\ \bullet\ \beta = \max(1, \underline{\mathbf{MIN\_NCC}_1}) + \max(1, \underline{\mathbf{MIN\_NCC}_2}). \end{cases} \tag{4.214}$$

Proof. The maximum number of connected components is achieved by building non-empty groups as small as possible, except for two groups of respective size $\max(1, \underline{\mathbf{MAX\_NCC}_1})$ and $\max(1, \underline{\mathbf{MAX\_NCC}_2})$, which have to be built. □

Proposition 153.

$$\begin{aligned}&\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:\\ &\mathbf{MAX\_NCC}_1 \cdot \max(0, \mathbf{NCC}_1 - 1) + \mathbf{MIN\_NCC}_1 + \\ &\mathbf{MAX\_NCC}_2 \cdot \mathbf{NCC}_1 + \mathbf{MIN\_NCC}_2 \geq \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{aligned} \tag{4.215}$$

Proof. The left-hand side of 153 corresponds to the maximum number of vertices of the two final graphs provided that we build the largest possible connected components. □

Proposition 154.

$\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:$

$$\mathbf{NCC}_1 \geq (\overline{\mathbf{MAX\_NCC}_2} < \mathbf{NVERTEX}_{\texttt{INITIAL}}) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + \left(\alpha \bmod \beta > \overline{\mathbf{MAX\_NCC}_2}\right)$$

$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \overline{\mathbf{MIN\_NCC}_1} - \overline{\mathbf{MIN\_NCC}_2}, \\ \bullet\ \beta = \max(1, \overline{\mathbf{MAX\_NCC}_1}) + \max(1, \overline{\mathbf{MAX\_NCC}_2}). \end{cases} \tag{4.216}$$

Proof. The minimum number of connected components is achieved by taking the groups as large as possible except for two groups of respective size $\overline{\mathbf{MIN\_NCC}_2}$ and $\overline{\mathbf{MIN\_NCC}_1}$, which have to be built. □

Proposition 155.

$$\begin{array}{l}\texttt{vpartition} \land \texttt{consecutive\_loops\_are\_connected}: \\ \mathbf{MAX\_NCC}_2 \leq \max(\mathbf{MIN\_NCC}_2, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \alpha), \texttt{with}: \\ \bullet\ \alpha = \mathbf{MIN\_NCC}_1 \cdot \max(0, \mathbf{NCC}_1 - 1) + \mathbf{MAX\_NCC}_1 + \\ \qquad \mathbf{MIN\_NCC}_2 + \mathbf{MIN\_NCC}_2 \cdot \max(0, \mathbf{NCC}_1 - 3)\end{array} \tag{4.217}$$

Proof. If $\mathbf{NCC}_1 \leq 1$ we have that $\mathbf{MAX\_NCC}_2 \leq \mathbf{MIN\_NCC}_2$. Otherwise, when $\mathbf{NCC}_1 > 1$, we have that $\mathbf{MIN\_NCC}_1 \cdot \max(0, \mathbf{NCC}_1 - 1) + \mathbf{MAX\_NCC}_1 + \mathbf{MIN\_NCC}_2 + \mathbf{MAX\_NCC}_2 + \mathbf{MIN\_NCC}_2 \cdot \max(0, \mathbf{NCC}_1 - 3) \leq \mathbf{NVERTEX}_{\texttt{INITIAL}}$. $\mathbf{NCC}_1 - 3$ comes from the fact that we build the minimum number of connected components in the second final graph (i.e., $\mathbf{NCC}_1 - 1$ connected components) and that we have already built two connected components of respective size $\mathbf{MIN\_NCC}_2$ and $\mathbf{MAX\_NCC}_2$. By isolating $\mathbf{MAX\_NCC}_2$ in the previous expression and by grouping the two inequalities the result follows. □

Proposition 156.

$$\begin{array}{l}\texttt{apartition} \land \textbf{arc\_gen} = \mathit{PATH} \land \mathbf{MIN\_NCC}_1 > 1 \land \mathbf{MIN\_NCC}_2 > 1: \\ \mathbf{NCC}_1 \leq (\mathbf{MAX\_NCC}_1 > 0) + \left\lfloor \dfrac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 \geq \mathbf{MIN\_NCC}_1), \ \texttt{with}: \\ \left\{ \begin{array}{l} \bullet\ \ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC}_1 - \mathbf{MAX\_NCC}_2 + 1), \\ \bullet\ \ \beta = \mathbf{MIN\_NCC}_1 + \mathbf{MIN\_NCC}_2 - 2. \end{array} \right.\end{array} \tag{4.218}$$

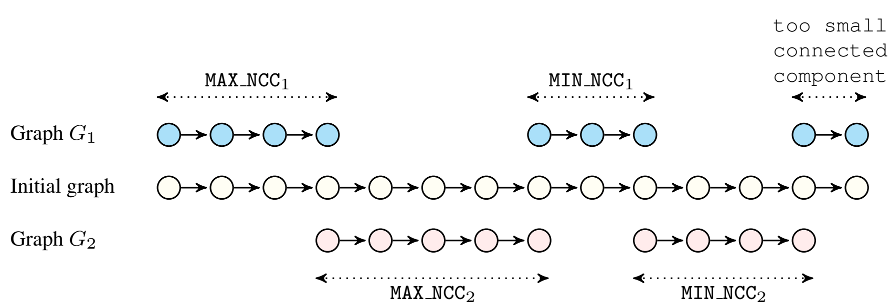


Figure 4.9: Illustration of Proposition 156. Configuration achieving the maximum number of connected components for G_1 according to the size of the smallest and largest connected components of G_1 and G_2 and to an initial number of vertices ($\texttt{MAX\_NCC}_1 = 4, \texttt{MAX\_NCC}_2 = 5, \texttt{MIN\_NCC}_1 = 3, \texttt{MIN\_NCC}_2 = 4, \texttt{NVERTEX}_{\texttt{INITIAL}} = 14, \alpha = \max(0, 14-4-5+1) = 6, \beta = \max(2, 3+4-2) = 5, \texttt{NCC}_1 = (4 > 0) + \lfloor \frac{6}{5} \rfloor + (((6 \bmod 5) + 1) \geq 3) = 2$); since the two rightmost vertices of graph G_1 correspond to a too small connected component, they will have to be dispatched in the other connected components of graph G_1.

Proof. The maximum number of connected components of G_1 is achieved by:

- Building a first connected component of G_1 involving $\mathbf{MAX\_NCC}_1$ vertices,
- Building a first connected component of G_2 involving $\mathbf{MAX\_NCC}_2$ vertices,

- Building alternatively a connected component of G_1 and a connected component of G_2 involving respectively $\mathbf{MIN\_NCC}_1$ and $\mathbf{MIN\_NCC}_2$ vertices,
- Finally, if this is possible, building a connected component of G_1 involving $\mathbf{MIN\_NCC}_1$ vertices.

□

Proposition 157.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = \mathit{PATH} \wedge \mathbf{MIN\_NCC}_1 > 1 \wedge \mathbf{MIN\_NCC}_2 > 1 :$$
$$\mathbf{NCC}_1 \geq (\mathbf{MIN\_NCC}_1 > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 > \mathbf{MAX\_NCC}_2),\ \texttt{with} :$$
$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MIN\_NCC}_1 - \mathbf{MIN\_NCC}_2 + 1), \\ \bullet\ \beta = \mathbf{MAX\_NCC}_1 + \mathbf{MAX\_NCC}_2 - 2. \end{cases} \tag{4.219}$$

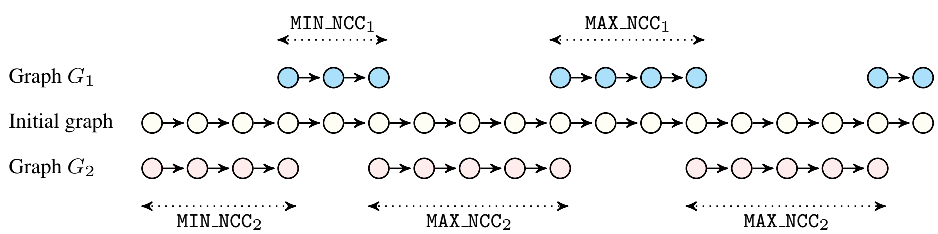


Figure 4.10: Illustration of Proposition 157. Configuration achieving the minimum number of connected components for G_1 according to the size of the smallest and largest connected components of G_1 and G_2 and to an initial number of vertices ($\texttt{MAX\_NCC}_1 = 4, \texttt{MAX\_NCC}_2 = 5, \texttt{MIN\_NCC}_1 = 3, \texttt{MIN\_NCC}_2 = 4, \texttt{NVERTEX}_{\texttt{INITIAL}} = 18, \alpha = \max(0, 18 - 3 - 4 + 1) = 12, \beta = \max(2, 4 + 5 - 2) = 7, \texttt{NCC}_1 = (3 > 0) + \lfloor \frac{12}{7} \rfloor + (((12 \bmod 7) + 1) > 5) = 3$)

Proof. The minimum number of connected components of G_1 is achieved by:

- Building a first connected component of G_2 involving $\mathbf{MIN\_NCC}_2$ vertices,
- Building a first connected component of G_1 involving $\mathbf{MIN\_NCC}_1$ vertices,
- Building alternatively a connected component of G_2 and a connected component of G_1 involving respectively $\mathbf{MAX\_NCC}_2$ and $\mathbf{MAX\_NCC}_1$ vertices,
- Finally, if this is possible, building a connected component of G_2 involving $\mathbf{MAX\_NCC}_2$ vertices and a connected component of G_1 with the remaining vertices. Note that these remaining vertices cannot be incorporated in the connected components previously built.

□

$\boxed{\mathbf{MAX\_NCC_1, MAX\_NCC_2, MIN\_NCC_1, MIN\_NCC_2, NCC_2}}$

Proposition 158.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:\\ \quad \mathbf{MIN\_NCC_2} \cdot \max(0, \mathbf{NCC_2} - 1) + \mathbf{MAX\_NCC_2} + \\ \quad \mathbf{MIN\_NCC_1} \cdot \max(0, \mathbf{NCC_2} - 2) + \mathbf{MAX\_NCC_1} \leq \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{array} \tag{4.220}$$

Proof. Similar to Proposition 151. □

Proposition 159.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:\\ \mathbf{NCC_2} \leq (\overline{\mathbf{MAX\_NCC_2}} > 0) + \left\lfloor \dfrac{\alpha}{\beta} \right\rfloor + \big(\alpha \bmod \beta \geq \max(1, \underline{\mathbf{MIN\_NCC_2}})\big)\\ \left\{\begin{array}{l}\bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \max(1, \underline{\mathbf{MAX\_NCC_2}}) - \max(1, \underline{\mathbf{MAX\_NCC_1}})),\\ \bullet\ \beta = \max(1, \underline{\mathbf{MIN\_NCC_2}}) + \max(1, \underline{\mathbf{MIN\_NCC_1}}).\end{array}\right.\end{array} \tag{4.221}$$

Proof. Similar to Proposition 152. □

Proposition 160.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:\\ \quad \mathbf{MAX\_NCC_2} \cdot \max(0, \mathbf{NCC_2} - 1) + \mathbf{MIN\_NCC_2} + \\ \quad \mathbf{MAX\_NCC_1} \cdot \mathbf{NCC_2} + \mathbf{MIN\_NCC_1} \geq \mathbf{NVERTEX}_{\texttt{INITIAL}}\end{array} \tag{4.222}$$

Proof. Similar to Proposition 153. □

Proposition 161.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:\\ \mathbf{NCC_2} \geq (\overline{\mathbf{MAX\_NCC_1}} < \mathbf{NVERTEX}_{\texttt{INITIAL}}) + \left\lfloor \dfrac{\alpha}{\beta} \right\rfloor + \big(\alpha \bmod \beta > \overline{\mathbf{MAX\_NCC_1}}\big)\\ \left\{\begin{array}{l}\bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \overline{\mathbf{MIN\_NCC_2}} - \overline{\mathbf{MIN\_NCC_1}},\\ \bullet\ \beta = \max(1, \overline{\mathbf{MAX\_NCC_2}}) + \max(1, \overline{\mathbf{MAX\_NCC_1}}).\end{array}\right.\end{array} \tag{4.223}$$

Proof. Similar to Proposition 154. □

Proposition 162.

$$\begin{array}{l}\texttt{vpartition} \wedge \texttt{consecutive\_loops\_are\_connected}:\\ \quad \mathbf{MAX\_NCC_1} \leq \max(\mathbf{MIN\_NCC_1}, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \alpha), \texttt{with}:\\ \quad \bullet\ \alpha = \mathbf{MIN\_NCC_2} \cdot \max(0, \mathbf{NCC_2} - 1) + \mathbf{MAX\_NCC_2} + \\ \qquad\quad \mathbf{MIN\_NCC_1} + \mathbf{MIN\_NCC_1} \cdot \max(0, \mathbf{NCC_2} - 3)\end{array} \tag{4.224}$$

Proof. Similar to Proposition 155. □

Proposition 163.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{MIN\_NCC_1} > 1 \wedge \mathbf{MIN\_NCC_2} > 1 :$$
$$\mathbf{NCC_2} \leq (\mathbf{MAX\_NCC_2} > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 \geq \mathbf{MIN\_NCC_2}),\ \texttt{with} :$$
$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MAX\_NCC_1} - \mathbf{MAX\_NCC_2} + 1), \\ \bullet\ \beta = \mathbf{MIN\_NCC_1} + \mathbf{MIN\_NCC_2} - 2. \end{cases} \tag{4.225}$$

Proof. Similar to Proposition 156. □

Proposition 164.

$$\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{MIN\_NCC_1} > 1 \wedge \mathbf{MIN\_NCC_2} > 1 :$$
$$\mathbf{NCC_2} \geq (\mathbf{MIN\_NCC_2} > 0) + \left\lfloor \frac{\alpha}{\beta} \right\rfloor + ((\alpha \bmod \beta) + 1 > \mathbf{MAX\_NCC_1},\ \texttt{with} :$$
$$\begin{cases} \bullet\ \alpha = \max(0, \mathbf{NVERTEX}_{\texttt{INITIAL}} - \mathbf{MIN\_NCC_1} - \mathbf{MIN\_NCC_2} + 1), \\ \bullet\ \beta = \mathbf{MAX\_NCC_1} + \mathbf{MAX\_NCC_2} - 2. \end{cases} \tag{4.226}$$

Proof. Similar to Proposition 157. □

Graph invariants relating six parameters of two final graphs

$\boxed{\mathbf{MAX\_NCC}_1, \mathbf{MAX\_NCC}_2, \mathbf{MIN\_NCC}_1, \mathbf{MIN\_NCC}_2, \mathbf{NCC}_1, \mathbf{NCC}_2}$

Proposition 165.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{NVERTEX}_{\texttt{INITIAL}} > 0 : \\ \alpha \cdot \mathbf{MIN\_NCC}_1 + \mathbf{MAX\_NCC}_1 + \\ \beta \cdot \mathbf{MIN\_NCC}_2 + \mathbf{MAX\_NCC}_2 \leq \mathbf{NVERTEX}_{\texttt{INITIAL}} + \mathbf{NCC}_1 + \mathbf{NCC}_2 - 1, \ \texttt{with} : \\ \left\{\begin{array}{l} \bullet \ \alpha = \max(0, \mathbf{NCC}_1 - 1), \\ \bullet \ \beta = \max(0, \mathbf{NCC}_2 - 1). \end{array}\right.\end{array} \tag{4.227}$$

Proof. Let $CC(G_1) = \{CC_a^1 : a \in [\mathbf{NCC}1]\}$ and $CC(G_2) = \{CC_a^2 : a \in [\mathbf{NCC}2]\}$ be respectively the set of connected components of the first and the second final graphs. Since the initial graph is a path, and since each arc of the initial graph belongs to the first or to the second final graphs (but not to both), there exists $(A_i)_{i \in [\mathbf{NCC}_1 + \mathbf{NCC}_2]}$ and there exists $j \in [2]$ such that $A_i \in CC(G_{1+(j \bmod 2)})$, for $i \bmod 2 = 0$ and $A_i \in CC(G_{1+((j+1) \bmod 2)})$ for $i \bmod 2 = 1$ and $A_i \cap A_{i+1} \neq \emptyset$ for $i \in [\mathbf{NCC}_1 + \mathbf{NCC}_2 - 1]$.
By inclusion-exclusion principle, since $A_i \cap A_j = \emptyset$ whenever $j \neq i + 1$, we obtain $\mathbf{NVERTEX}_{\texttt{INITIAL}} = \Sigma_{a \in [\mathbf{NCC1}]} |CC_a^1| + \Sigma_{a \in [\mathbf{NCC2}]} |CC_a^2| - \Sigma_{i \in [\mathbf{NCC1} + \mathbf{NCC2} - 1]} |A_i \cap A_{i+1}|$. Since $|A_i \cap A_{i+1}|$ is equal to 1 for every well defined i, we obtain $\Sigma_{a \in [\mathbf{NCC1}]} |CC_a^1| + \Sigma_{a \in [\mathbf{NCC2}]} |CC_a^2| = \mathbf{NVERTEX}_{\texttt{INITIAL}} + \mathbf{NCC}1 + \mathbf{NCC}2 - 1$.
Since $\alpha \cdot \mathbf{MIN\_NCC}_1 + \mathbf{MAX\_NCC}_1 + \beta \cdot \mathbf{MIN\_NCC}_2 + \mathbf{MAX\_NCC}_2 \leq \Sigma_{a \in [\mathbf{NCC1}]} |CC_a^1| + \Sigma_{a \in [\mathbf{NCC2}]} |CC_a^2|$ the result follows. □

Proposition 166.

$$\begin{array}{l}\texttt{apartition} \wedge \mathbf{arc\_gen} = PATH \wedge \mathbf{NVERTEX}_{\texttt{INITIAL}} > 0 : \\ \alpha \cdot \mathbf{MAX\_NCC}_1 + \mathbf{MIN\_NCC}_1 + \\ \beta \cdot \mathbf{MAX\_NCC}_2 + \mathbf{MIN\_NCC}_2 \geq \mathbf{NVERTEX}_{\texttt{INITIAL}} + \mathbf{NCC}_1 + \mathbf{NCC}_2 - 1, \ \texttt{with} : \\ \left\{\begin{array}{l} \bullet \ \alpha = \max(0, \mathbf{NCC}_1 - 1), \\ \bullet \ \beta = \max(0, \mathbf{NCC}_2 - 1). \end{array}\right.\end{array} \tag{4.228}$$

Proof. Similar to Proposition 165. □

4.4 Functional dependency invariants

This section provides invariants relating functionally dependent constraints arguments.

4.4.1 Functional dependency invariants involving two constraints

Proposition 167. *Given the constraints*

- INFLEXION($\mathtt{NINF}, \mathtt{VARIABLES}$),
- NVALUE($\mathtt{NVAL}, \mathtt{VARIABLES}$):

$$\mathtt{NVAL} = 1 \Rightarrow \mathtt{NINF} = 0 \tag{4.229}$$

Proof. Since a single value leads to a plateau. □

Proposition 168. *Given the constraints*

- LENGTH_FIRST_SEQUENCE($\mathtt{LEN}, \mathtt{VARIABLES}$),
- PEAK($\mathtt{P}, \mathtt{VARIABLES}$):

$$2 \cdot \mathtt{P} \leq |\mathtt{VARIABLES}| - \mathtt{LEN} \tag{4.230}$$

Proof. Beside a first sequence with a small value, we alternate between large and small values in order to maximise the number of peaks. □

Proposition 169. *Given the constraints*

- LENGTH_FIRST_SEQUENCE($\mathtt{LEN}, \mathtt{VARIABLES}$),
- VALLEY($\mathtt{V}, \mathtt{VARIABLES}$):

$$2 \cdot \mathtt{V} \leq |\mathtt{VARIABLES}| - \mathtt{LEN} \tag{4.231}$$

Proof. Beside a first sequence with a large value, we alternate between small and large values in order to maximise the number of valleys. □

Proposition 170. *Given the constraints*

- LENGTH_LAST_SEQUENCE($\mathtt{LEN}, \mathtt{VARIABLES}$),
- PEAK($\mathtt{P}, \mathtt{VARIABLES}$):

$$2 \cdot \mathtt{P} \leq |\mathtt{VARIABLES}| - \mathtt{LEN} \tag{4.232}$$

Proof. Beside a last sequence with a small value, we alternate between large and small values in order to maximise the number of peaks. □

Proposition 171. *Given the constraints*

- LENGTH_LAST_SEQUENCE($\mathtt{LEN}, \mathtt{VARIABLES}$),
- VALLEY($\mathtt{V}, \mathtt{VARIABLES}$):

$$2 \cdot \mathtt{V} \leq |\mathtt{VARIABLES}| - \mathtt{LEN} \tag{4.233}$$

Proof. Beside a last sequence with a large value, we alternate between small and large values in order to maximise the number of valleys. □

Proposition 172. *Given the constraints*

- LONGEST_DECREASING_SEQUENCE($\mathtt{L}, \mathtt{VARIABLES}$),

- MAX_DECREASING_SLOPE($\mathtt{MAX}, \mathtt{VARIABLES}$):

$$\mathtt{L} \geq \mathtt{MAX} \tag{4.234}$$

Proof. By definition of the LONGEST_DECREASING_SEQUENCE and MAX_DECREASING_SLOPE constraints. □

Proposition 173. *Given the constraints*

- LONGEST_INCREASING_SEQUENCE($\mathtt{L}, \mathtt{VARIABLES}$),
- MAX_INCREASING_SLOPE($\mathtt{MAX}, \mathtt{VARIABLES}$):

$$\mathtt{L} \geq \mathtt{MAX} \tag{4.235}$$

Proof. By definition of the LONGEST_INCREASING_SEQUENCE and MAX_INCREASING_SLOPE constraints. □

Proposition 174. *Given the constraints*

- MAX_DECREASING_SLOPE($\mathtt{MAX}, \mathtt{VARIABLES}$),
- MIN_DECREASING_SLOPE($\mathtt{MIN}, \mathtt{VARIABLES}$):

$$\mathtt{MAX} \geq \mathtt{MIN} \tag{4.236}$$

Proof. By definition of the MAX_DECREASING_SLOPE and MIN_DECREASING_SLOPE constraints. □

Proposition 175. *Given the constraints*

- MAX_INCREASING_SLOPE($\mathtt{MAX}, \mathtt{VARIABLES}$),
- MIN_INCREASING_SLOPE($\mathtt{MIN}, \mathtt{VARIABLES}$):

$$\mathtt{MAX} \geq \mathtt{MIN} \tag{4.237}$$

Proof. By definition of the MAX_INCREASING_SLOPE and MIN_INCREASING_SLOPE constraints. □

Proposition 176. *Given the constraints*

- MAXIMUM($\mathtt{MAX}, \mathtt{VARIABLES}$),
- MINIMUM($\mathtt{MIN}, \mathtt{VARIABLES}$):

$$\mathtt{MAX} \geq \mathtt{MIN} \tag{4.238}$$

Proof. By definition of the MAXIMUM and the MINIMUM constraints. □

Proposition 177. *Given the constraints*

- MAXIMUM($\mathtt{MAX}, \mathtt{VARIABLES}$),
- SUM_CTR($\mathtt{VARIABLES}, =, \mathtt{SUM}$):

$$\mathtt{SUM} \leq |\mathtt{VARIABLES}| \cdot \mathtt{MAX} \tag{4.239}$$

Proof. By definition of the MAXIMUM and SUM_CTR constraints. □

Proposition 178. *Given the constraints*

- MINIMUM($\mathtt{MIN}, \mathtt{VARIABLES}$),
- SUM_CTR($\mathtt{VARIABLES}, =, \mathtt{SUM}$):

$$\texttt{SUM} \geq |\texttt{VARIABLES}| \cdot \texttt{MIN} \tag{4.240}$$

Proof. By definition of the MINIMUM and SUM_CTR constraints. □

Proposition 179. *Given the constraints*

- MIN_WIDTH_PEAK(MIN_WIDTH, VARIABLES),
- PEAK(P, VARIABLES):

$$\texttt{P} \cdot \texttt{MIN\_WIDTH} \leq |\texttt{VARIABLES}| \tag{4.241}$$

Proof. Cumulated minimum width of the different peaks cannot exceed size of sequence. □

Proposition 180. *Given the constraints*

- MIN_WIDTH_PEAK(MIN_WIDTH, VARIABLES),
- PEAK(P, VARIABLES):

The automaton depicted by Figure 4.11 provides a necessary condition.

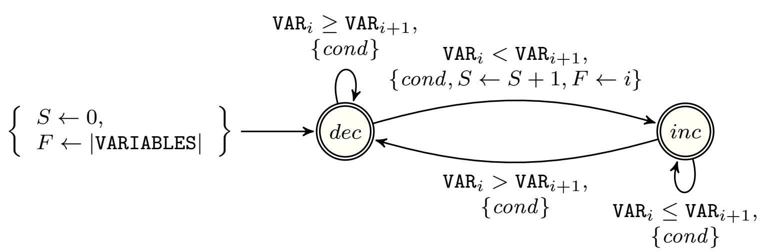


Figure 4.11: Automaton for a redundant constraint between the PEAK(P, VARIABLES) and the MIN_WIDTH_PEAK(MIN_WIDTH, VARIABLES) constraints when $\underline{\texttt{P}} > 0$ and $\underline{\texttt{MIN\_WIDTH}} > 0$, where $cond$ is the condition $|\texttt{VARIABLES}| - i > \max(0, \max(\underline{\texttt{P}} - \max(S-1, 0), 0) - 1) \cdot (\underline{\texttt{MIN\_WIDTH}} + 1) + (\max(\underline{\texttt{P}} - \max(S-1,0), 0) \geq 1) \cdot (\underline{\texttt{MIN\_WIDTH}} - \max(i - F, 0))$, and where i, S and F respectively stand for the index of the current pairs ($i \in [1, |\texttt{VARIABLES}| - 1]$), the number of start of potential peaks already encountered, the position of the start of the last potential peak encountered; the quantity $\max(S-1, 0)$ denotes the number of peaks already encountered, while the quantity $\max(\underline{\texttt{P}} - \max(S-1,0), 0)$ denotes the minimum number of peaks that remain to done from position i.

Proof. The condition associated with each transition of the automaton checks that there is enough space between the current position and the end of the sequence to place the remaining minimum number of required peaks. □

Proposition 181. *Given the constraints*

- MIN_WIDTH_VALLEY(MIN_WIDTH, VARIABLES),
- VALLEY(V, VARIABLES):

$$\texttt{V} \cdot \texttt{MIN\_WIDTH} \leq |\texttt{VARIABLES}| \tag{4.242}$$

Proof. Cumulated minimum width of the different valleys cannot exceed size of sequence. □

Proposition 182. *Given the constraints*

- MIN_WIDTH_VALLEY(MIN_WIDTH, VARIABLES),
- VALLEY(V, VARIABLES):

The automaton depicted by Figure 4.12 provides a necessary condition.

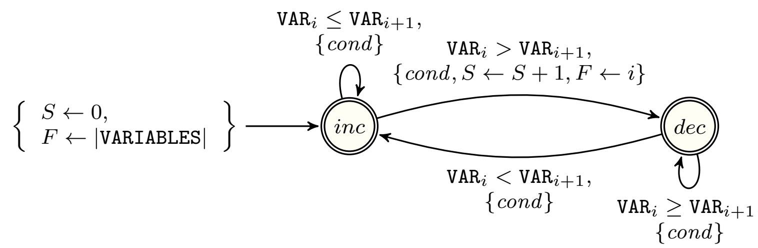


Figure 4.12: Automaton for a redundant constraint between the VALLEY(V, VARIABLES) and the MIN_WIDTH_VALLEY(MIN_WIDTH, VARIABLES) constraints when $\underline{\mathtt{V}} > 0$ and $\underline{\mathtt{MIN\_WIDTH}} > 0$, where *cond* is the condition $|\mathtt{VARIABLES}| - i > \max(0, \max(\underline{\mathtt{V}} - \max(S-1,0),0)-1) \cdot \underline{\mathtt{MIN\_WIDTH}} + (\max(\underline{\mathtt{V}} - \max(S-1,0),0) \geq 1) \cdot (\underline{\mathtt{MIN\_WIDTH}} - \max(i-F,0))$, and where i, S and F respectively stand for the index of the current pairs ($i \in [1, |\mathtt{VARIABLES}|-1]$), the number of start of potential valleys already encountered, the position of the start of the last potential valley encountered; the quantity $\max(S-1,0)$ denotes the number of valleys already encountered, while the quantity $\max(\underline{\mathtt{V}} - \max(S-1,0),0)$ denotes the minimum number of valleys that remain to done from position i.

Proof. The condition associated with each transition of the automaton checks that there is enough space between the current position and the end of the sequence to place the remaining minimum number of required valleys. □

Proposition 183. *Given the constraints*

- NVALUE(NVAL, VARIABLES),
- NVISIBLE_FROM_END(N, VARIABLES):

$$\mathtt{NVAL} \geq \mathtt{N} \tag{4.243}$$

Proof. Since stairs visible from the end are located at different altitudes. □

Proposition 184. *Given the constraints*

- NVALUE(NVAL, VARIABLES),
- NVISIBLE_FROM_START(N, VARIABLES):

$$\mathtt{NVAL} \geq \mathtt{N} \tag{4.244}$$

Proof. Since stairs visible from the start are located at different altitudes. □

Proposition 185. *Given the constraints*

- PEAK(P, VARIABLES),

- VALLEY(V, VARIABLES):

$$|\mathtt{P} - \mathtt{V}| \leq 1 \tag{4.245}$$

Proof. Since peaks and valleys can only alternate. □

Proposition 186. *Given the constraints*

- PEAK(P, VARIABLES),
- VALLEY(V, VARIABLES),

with $\mathtt{P} > \mathtt{V}$*, the automaton depicted by Figure 4.13 provides a necessary condition.*

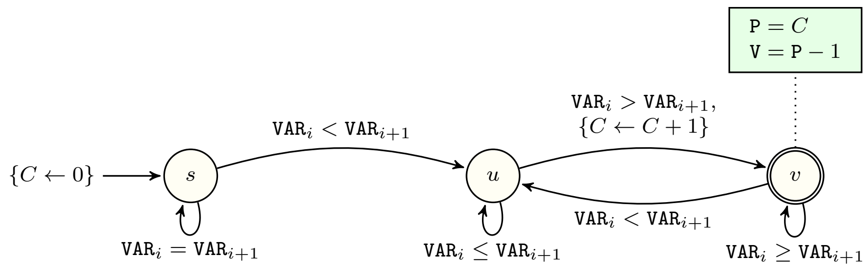


Figure 4.13: Automaton for a redundant constraint for the conjunction PEAK(P, VARIABLES) $\wedge$ VALLEY(V, VARIABLES) $\wedge$ $\mathtt{P} > \mathtt{V}$ (we have both to start with a peak and to finish with a peak)

Proof. Since peaks and valleys can only alternate and since having more peaks than valleys enforces to start and end on a peak. □

Proposition 187. *Given the constraints*

- PEAK(P, VARIABLES),
- VALLEY(V, VARIABLES),

with $\mathtt{V} > \mathtt{P}$ *the automaton depicted by Figure 4.14 provides a necessary condition.*

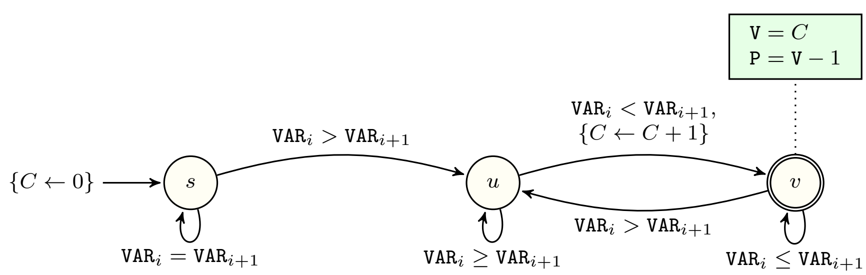


Figure 4.14: Automaton for a redundant constraint for the conjunction VALLEY(V, VARIABLES) $\wedge$ PEAK(P, VARIABLES) $\wedge$ $\mathtt{V} > \mathtt{P}$ (we have both to start with a valley and to finish with a valley)

Proof. Since valleys and peaks can only alternate and since having more valleys than peaks enforces to start and end on a valley. □

4.4.2 Functional dependency invariants involving three constraints

Proposition 188. *Given the constraints*

- INFLEXION($\mathtt{I}, \mathtt{VARIABLES}$),
- PEAK($\mathtt{P}, \mathtt{VARIABLES}$),
- VALLEY($\mathtt{V}, \mathtt{VARIABLES}$):

$$\mathtt{I} = \mathtt{P} + \mathtt{V} \tag{4.246}$$

Proof. By definition of the INFLEXION, PEAK and VALLEY constraints. □

Proposition 189. *Given the constraints*

- LENGTH_FIRST_SEQUENCE($\mathtt{LEN\_FIRST}, \mathtt{VARIABLES}$),
- LENGTH_LAST_SEQUENCE($\mathtt{LEN\_LAST}, \mathtt{VARIABLES}$),
- NVALUE($\mathtt{NVAL}, \mathtt{VARIABLES}$):

$$\mathtt{NVAL} > 1 \Rightarrow \mathtt{LEN\_FIRST} + \mathtt{LEN\_LAST} + \mathtt{NVAL} - 2 \leq |\mathtt{VARIABLES}| \tag{4.247}$$

Proof. Since we have at least two distinct values the first and last sequences do not overlap. Since we have at least $\mathtt{NVAL}$ distinct values we have at least $\mathtt{NVAL} - 2$ additional values excluding the first and last sequences, which use at most two distinct values. □

Proposition 190. *Given the constraints*

- LONGEST_DECREASING_SEQUENCE($\mathtt{L}, \mathtt{VARIABLES}$),
- MAXIMUM($\mathtt{MAX}, \mathtt{VARIABLES}$),
- MINIMUM($\mathtt{MIN}, \mathtt{VARIABLES}$):

$$\mathtt{MAX} - \mathtt{MIN} \geq \mathtt{L} \tag{4.248}$$

Proof. By definition of the LONGEST_DECREASING_SEQUENCE constraint. □

Proposition 191. *Given the constraints*

- LONGEST_INCREASING_SEQUENCE($\mathtt{L}, \mathtt{VARIABLES}$),
- MAXIMUM($\mathtt{MAX}, \mathtt{VARIABLES}$),
- MINIMUM($\mathtt{MIN}, \mathtt{VARIABLES}$):

$$\mathtt{MAX} - \mathtt{MIN} \geq \mathtt{L} \tag{4.249}$$

Proof. By definition of the LONGEST_INCREASING_SEQUENCE constraint. □

Proposition 192. *Given the constraints*

- MAXIMUM($\mathtt{MAX}, \mathtt{VARIABLES}$),
- MINIMUM($\mathtt{MIN}, \mathtt{VARIABLES}$),
- NVALUE($\mathtt{NVAL}, \mathtt{VARIABLES}$):

$$\mathtt{NVAL} \leq \mathtt{MAX} - \mathtt{MIN} + 1 \tag{4.250}$$

Proof. Since taking all values between $\mathtt{MIN}$ and $\mathtt{MAX}$ gives the maximum number of distinct values. □

Proposition 193. *Given the constraints*

- MAXIMUM$(\mathtt{MAX}, \mathtt{VARIABLES})$,
- MINIMUM$(\mathtt{MIN}, \mathtt{VARIABLES})$,
- NVALUE$(\mathtt{NVAL}, \mathtt{VARIABLES})$:

$$\mathtt{MAX} > \mathtt{MIN} \Rightarrow \mathtt{NVAL} > 1 \tag{4.251}$$

Proof. Since at least two distinct values if $\mathtt{MIN}$ and $\mathtt{MAX}$ are distinct. □

Proposition 194. *Given the constraints*

- MAXIMUM$(\mathtt{MAX}, \mathtt{VARIABLES})$,
- MINIMUM$(\mathtt{MIN}, \mathtt{VARIABLES})$,
- SUM_CTR$(\mathtt{VARIABLES}, =, \mathtt{SUM})$:

$$\mathtt{SUM} \geq (|\mathtt{VARIABLES}| - 1) \cdot \mathtt{MIN} + \mathtt{MAX} \tag{4.252}$$

Proof. Since also has to use the maximum value at least once in the sum. □

Proposition 195. *Given the constraints*

- MAXIMUM$(\mathtt{MAX}, \mathtt{VARIABLES})$,
- MINIMUM$(\mathtt{MIN}, \mathtt{VARIABLES})$,
- SUM_CTR$(\mathtt{VARIABLES}, =, \mathtt{SUM})$:

$$\mathtt{SUM} \leq (|\mathtt{VARIABLES}| - 1) \cdot \mathtt{MAX} + \mathtt{MIN} \tag{4.253}$$

Proof. Since also has to use the minimum value at least once in the sum. □

4.4.3 Functional dependency invariants involving four constraints

Proposition 196. *Given the constraints*

- LENGTH_FIRST_SEQUENCE$(\mathtt{LEN_FIRST}, \mathtt{VARIABLES})$,
- LENGTH_LAST_SEQUENCE$(\mathtt{LEN_LAST}, \mathtt{VARIABLES})$,
- NVALUE$(\mathtt{NVAL}, \mathtt{VARIABLES})$,
- PEAK$(\mathtt{P}, \mathtt{VARIABLES})$:

$$\mathtt{NVAL} > 1 \Rightarrow 2 \cdot \mathtt{P} \leq |\mathtt{VARIABLES}| - \max(1, \mathtt{LEN\_FIRST}) - \max(1, \mathtt{LEN\_LAST}) + 1 \tag{4.254}$$

Proof. Beside the first and the last sequence with a small value, we alternate between large and small values. □

Proposition 197. *Given the constraints*

- LENGTH_FIRST_SEQUENCE$(\mathtt{LEN_FIRST}, \mathtt{VARIABLES})$,
- LENGTH_LAST_SEQUENCE$(\mathtt{LEN_LAST}, \mathtt{VARIABLES})$,
- NVALUE$(\mathtt{NVAL}, \mathtt{VARIABLES})$,
- VALLEY$(\mathtt{V}, \mathtt{VARIABLES})$:

$$\mathtt{NVAL} > 1 \Rightarrow 2 \cdot \mathtt{V} \leq |\mathtt{VARIABLES}| - \max(1, \mathtt{LEN\_FIRST}) - \max(1, \mathtt{LEN\_LAST}) + 1 \tag{4.255}$$

Proof. Beside the first and the last sequence with a large value, we alternate between small and large values. □

4.5 The electronic version of the catalogue

4.5.1 Prolog facts describing a constraint

An electronic version of the catalogue containing every global constraint of the catalogue is given in Appendix B. In addition the entry "Utilities" contains a set of shared utilities used for evaluating the constraints. This electronic version was used for generating the LaTeX file of this catalogue, the figures associated with the graph-based description and a filtering algorithm for some of the constraints that use the automaton-based description. Within the electronic version, each constraint is described in terms of meta-data. A typical entry is:

```
ctr_date(minimum, ['20000128','20030820','20040530',
                   '20041230','20060811','20090416']).

ctr_origin(minimum, '\\index{CHIP|indexuse}CHIP', []).

ctr_arguments(minimum, ['MIN'-dvar, 'VARIABLES'-collection(var-dvar)]).

ctr_exchangeable(minimum,
                 [items('VARIABLES',all),
                  vals(['VARIABLES'^var],int,=\=,all,in),
                  translate(['MIN','VARIABLES'^var])]).

ctr_synonyms(minimum, [min]).

ctr_restrictions(minimum, [size('VARIABLES')>0, required('VARIABLES',var)]).

ctr_typical(minimum, [size('VARIABLES') > 1, range('VARIABLES'^var) > 1]).

ctr_pure_functional_dependency(minimum, []).
ctr_functional_dependency(minimum, 1, [2]).

ctr_aggregate(minimum, [], [min, union]).

ctr_graph(minimum,
          ['VARIABLES'],
          2,
          ['CLIQUE'>>collection(variables1,variables2)],
          [variables1^key=variables2^key #\/ variables1^var<variables2^var],
          ['ORDER'(0,'MAXINT',var)='MIN'],
          []).
```

```
ctr_example(minimum,
            [minimum(2,[[var-3],[var-2],[var-7],[var-2],[var-6]]),
             minimum(7,[[var-8],[var-8],[var-7],[var-8],[var-7]])]).

ctr_cond_imply(minimum, deepest_valley,
               [first('VARIABLES'^var) > 'MIN',
                last('VARIABLES'^var) > 'MIN'], [], id).

ctr_see_also(minimum,
 [link('generalisation', minimum_modulo,
       '%e replaced by %e', [variable, variable mod constant]),
  link('specialisation', min_n,
       'minimum or order %e replaced by absolute minimum', [n]),
  link('comparison swapped', maximum, '', []),
  link('common keyword', maximum, '%k', ['order constraint']),
  link('soft variant', open_minimum, '%k', ['open constraint']),
  link('soft variant', minimum_except_0, 'value %e is ignored', [0]),
  link('implies', between_min_max, '', []),
  link('implies', in, '', []),
  link('implied by', and, '', [])]).

ctr_key_words(minimum,['order constraint'                        ,
                       'minimum'                                 ,
                       'maxint'                                  ,
                       'automaton'                               ,
                       'automaton without counters'              ,
                       'reified automaton constraint'            ,
                       'centered cyclic(1) constraint network(1)',
                       'arc-consistency'                         ]).

ctr_persons(minimum,['Beldiceanu N.']).

ctr_eval(minimum, [builtin(minimum_b), automaton(minimum_a)]).

minimum_b(MIN, VARIABLES) :-
    check_type(dvar, MIN),      collection(VARIABLES, [dvar]),
    length(VARIABLES, N),       N > 0,
    get_attr1(VARIABLES, VARS), minimum(MIN, VARS).

minimum_a(MIN, VARIABLES) :-   % 0: MIN<VAR, 1: MIN=VAR, 2: MIN>VAR
    minimum_signature(VARIABLES, SIGNATURE, MIN),
    automaton(SIGNATURE, _, SIGNATURE,
              [source(s),sink(t)],
              [arc(s,0,s),arc(s,1,t),arc(t,1,t),arc(t,0,t)],
              [],[],[]).

minimum_signature([], [], _).
minimum_signature([[var-VAR]|VARs], [S|Ss], MIN) :-
    S in 0..2,
    MIN #< VAR #<=> S #= 0, MIN #= VAR #<=> S #= 1, MIN #> VAR #<=> S #= 2,
    minimum_signature(VARs, Ss, MIN).

ctr_sol(minimum,2,0,2,9,[0-5,1-3,2-1]).
ctr_sol(minimum,3,0,3,64,[0-37,1-19,2-7,3-1]).
ctr_sol(minimum,4,0,4,625,[0-369,1-175,2-65,3-15,4-1]).
ctr_sol(minimum,5,0,5,7776,[0-4651,1-2101,2-781,3-211,4-31,5-1]).
ctr_sol(minimum,6,0,6,117649,[0-70993,1-31031,2-11529,3-3367,
                              4-665,5-63,6-1]).
ctr_sol(minimum,7,0,7,2097152,[0-1273609,1-543607,2-201811,3-61741,
                               4-14197,5-2059,6-127,7-1]).
ctr_sol(minimum,8,0,8,43046721,[0-26269505,1-11012415,2-4085185,
                                3-1288991,4-325089,5-58975,6-6305,7-255,
                                8-1]).
```

and consists of the following Prolog facts, where CONSTRAINT_NAME is the name of the constraint under consideration. The facts are organised in the following 20 items:

- Items 1, 2, 3, 4, 16 and 17 provide general information about a global constraint,
- Items 5, 6, 7 and 8 describe the arguments of a global constraint.
- Items 9 and 10 give properties of the arguments of a global constraint, e.g. functional dependency, contractibility.
- Item 11 describes symmetries of a global constraint, i.e. list of mappings (e.g., permutation of arguments) that preserve the solutions to a global constraint.
- Items 12 and 13 describes the meaning of a global constraint in terms of a graph-based representation.
- Item 14 provides one or several ground instances which hold.
- Item 18 gives the list of available evaluators of a global constraint.
- Item 19 provides the number of solutions to the constraint under various assumptions on the number of variables and on their initial domains.
- Item 20 describes the meaning of a global constraint in terms of a set of first order logic formulae.

Items 1, 2, 6 and 14 are mandatory, while all other items are optional. We now give the different items:

1. `ctr_date`(`CONSTRAINT_NAME`, `LIST_OF_DATES_OF_MODIFICATIONS`)

   - `LIST_OF_DATES_OF_MODIFICATIONS` is a list of dates when the description of the constraint was modified.

2. `ctr_origin`(`CONSTRAINT_NAME`, `STRING`, `LIST_OF_CONSTRAINTS_NAMES`)

   - `STRING` is a string denoting the origin of the constraint. `LIST_OF_CONSTRAINTS_NAMES` is a possibly empty list of constraint names related to the origin of the constraint.

3. `ctr_usual_name`(`CONSTRAINT_NAME`, `USUAL_NAME`)

   - When, for some reason, the constraint name used in the catalogue does not correspond to the usual name of the constraint, `USUAL_NAME` provides the usual name of the constraint. This stems from the fact that each entry of the catalogue should have a distinct name. This is the case, for example, for the STRETCH_PATH and the STRETCH_CIRCUIT constraints which are both usually called STRETCH.

4. `ctr_synonyms`(`CONSTRAINT_NAME`, `LIST_OF_SYNONYMS`)

   - `LIST_OF_SYNONYMS` is a list of synonyms for the constraint. This stems from the fact that, quite often, different authors use a different name for the same constraint. This is the case, for example, for the ALLDIFFERENT and the SYMMETRIC_ALLDIFFERENT constraints.

5. ctr_types(CONSTRAINT_NAME, LIST_OF_TYPES_DECLARATIONS)

   - LIST_OF_TYPES_DECLARATIONS is a list of elements of the form name-type, where name is the name of a new type and type the type itself (usually a collection). Basic and compound data types were respectively introduced in sections 2.2.1 and 2.2.2 on page 14. This field is only used when we need to declare a new type that will be used for specifying the type of the arguments of the constraint. This is the case, for example, when one argument of the constraint is a collection for which the type of one attribute is also a collection. This is the case, for example, for the DIFFN constraint where the unique argument ORTHOTOPES is a collection of ORTHOTOPE; ORTHOTOPE refers to a new type declared in LIST_OF_TYPES_DECLARATIONS.

6. ctr_arguments(CONSTRAINT_NAME, LIST_OF_ARGUMENTS_DECLARATIONS)

   - LIST_OF_ARGUMENTS_DECLARATIONS is a list of elements of the form arg-type, where arg is the name of an argument of the constraint and type the type of the argument. Basic and compound data types were respectively introduced in sections 2.2.1 and 2.2.2 on page 14.

7. ctr_restrictions(CONSTRAINT_NAME, LIST_OF_RESTRICTIONS)

   - LIST_OF_RESTRICTIONS is a list of restrictions on the different arguments of the constraint. Possible restrictions were described in Section 2.2.3 on page 17.

8. ctr_typical(CONSTRAINT_NAME, LIST_OF_RESTRICTIONS)

   - LIST_OF_RESTRICTIONS is a list of typical restrictions on the different arguments of the constraint, i.e. even though these restrictions are not mandatory they usually hold. Possible restrictions were described in Section 2.2.3 on page 17.

9. ctr_pure_functional_dependency(CONSTRAINT_NAME, LIST_OF_RESTRICTIONS)

   - Indicate that, under the assumption that all restrictions described by LIST_OF_RESTRICTIONS hold, at least one of the arguments of the constraint is functionally determined by the other arguments. Possible restrictions were described in Section 2.2.3 on page 17. Which argument ARG is determined by which subset of arguments LIST_ARGS is described by the fact ctr_functional_dependency(CONSTRAINT_NAME, ARG, LIST_ARGS) where arguments are denoted by their positions within the constraint.

10. ctr_aggregate(CONSTRAINT_NAME, LIST_OF_RESTRICTIONS, ARG_AGGREGATION)
    Denotes that, given two instances of the constraint that both satisfy all restrictions described by LIST_OF_RESTRICTIONS, we can combine (i.e., aggregate) these two instances in order to obtain an implied constraint, which has the same name as the first two constraints. We use ARG_AGGREGATION in order to obtain the arguments of the implied constraint as described in the keyword Aggregate.

11. ctr_exchangeable(CONSTRAINT_NAME, LIST_OF_SYMMETRIES)

    - LIST_OF_SYMMETRIES is a list of mappings preserving the solutions to the constraint. Possible mappings were described in Section 2.2.5 on page 27.

12. ctr_derived_collections(CONSTRAINT_NAME, LIST_OF_DERIVED_COLLECTIONS)

    - LIST_OF_DERIVED_COLLECTIONS is a list of derived collections. Derived collections are collections that are computed from the arguments of the constraint and are used in the graph-based description. Derived collections were described in Section 2.3.2 on page 51.

13. ctr_graph(CONSTRAINT_NAME, LIST_OF_ARC_INPUT, ARC_ARITY,
    ARC_GENERATORS, ARC_CONSTRAINTS, GRAPH_PROPERTIES)

    - LIST_OF_ARC_INPUT is a list of collections used for creating the vertices of the initial graph. This was described at page 80 of Section 2.3.3.
    - ARC_ARITY is the number of vertices of an arc. Arc arity was explained at page 82 of Section 2.3.3.
    - ARC_GENERATORS is a list of arc generators. Arc generators were introduced at page 80 of Section 2.3.3.
    - ARC_CONSTRAINTS is a list of arc constraints. Arc constraints were defined in Section 2.3.2 on page 57.
    - GRAPH_PROPERTIES is a list of graph properties. Graph properties were described in Section 2.3.2 on page 68.

14. ctr_example(CONSTRAINT_NAME, LIST_OF_EXAMPLES)

    - LIST_OF_EXAMPLES is a list of examples (usually one). Each example corresponds to a ground instance for which the constraint holds.

15. ctr_cond_imply(CONSTRAINT_NAME, IMPLIED_CONSTRAINT,
    LIST_OF_RESTRICTIONS_CTR, LIST_OF_RESTRICTIONS_IMPLIED_CTR, MATCHING)

    IMPLIED_CONSTRAINT is a constraint implied by CONSTRAINT_NAME provided that

    - the list of restrictions LIST_OF_RESTRICTIONS_CTR holds for the constraint CONSTRAINT_NAME, and
    - the list of restrictions LIST_OF_RESTRICTIONS_IMPLIED_CTR holds for the constraint IMPLIED_CONSTRAINT.

    MATCHING describes how to create the arguments of IMPLIED_CONSTRAINT from the arguments of CONSTRAINT_NAME. Note that some arguments of IMPLIED_CONSTRAINT may not be explicitly mentioned since they correspond to existentially quantified variables.

16. ctr_see_also(CONSTRAINT_NAME, LIST_OF_CONSTRAINTS)

    - LIST_OF_CONSTRAINTS is a list of constraints that are related in some way to the constraint. Each element of the list is a fact of the form link(TYPE_OF_LINK, CONSTRAINT, STRING, SYMBOLS), where:
      - TYPE_OF_LINK is a semantic link that explains why we refer to CONSTRAINT. Semantic links were described in Section 2.6 on page 94.
      - CONSTRAINT is the name of the constraint that is linked to CONSTRAINT_NAME.
      - STRING is a string providing contextual explanation.

– SYMBOLS is a list of symbols (e.g., keywords, constraint names, mathematical expressions) that are inserted in STRING.

17. ctr_key_words(CONSTRAINT_NAME, LIST_OF_KEYWORDS)

- LIST_OF_KEYWORDS is a list of keywords associated with the constraint. Keywords may be linked to the *meaning* of the constraint, to a *typical pattern* where the constraint can be applied or to a *specific problem* where the constraint is useful. All keywords used in the catalogue are listed in alphabetic order in Section 3.7 on page 161. Each keyword has an entry explaining its meaning and providing the list of global constraints using that keyword.

18. ctr_eval(CONSTRAINT_NAME, LIST_OF_EVALUATORS)

- For many of the constraints of the catalogue one or several evaluators are provided. Each evaluator is explicitly described in LIST_OF_EVALUATORS by an element of the form method(predicate_name), where predicate_name is the name of the Prolog predicate to call in order to evaluate the constraint,[1], and method can be one of the following keywords:
  - builtin when the corresponding evaluator uses a **SICStus** built-in. This is the case, for example, for the ALLDIFFERENT constraint.
  - reformulation when the corresponding evaluator reformulates the constraints in terms of a conjunction of constraints of the catalogue and/or in term of a conjunction of reified constraints. This is the case, for example, for the TREE constraint.
  - automaton when the corresponding evaluator is based on an automaton that describes the set of solutions accepted by the constraint. The evaluator corresponds to the Prolog code that creates the signature constraints as well as the automata (usually one) associated with the constraint. A fact of the form automaton/9 lists the states and the transitions of the automata used for describing the set of solutions accepted by the constraint. It follows the description provided in Section 2.4.2 on page 92. The PATTERN constraint is an example of constraint for which an automaton is provided.
  - logic when the corresponding evaluator is based on a first order logic formula that describes the meaning of the constraint. This is the case, for example, for the MEET_SBOXES constraint.
  - checker when the corresponding evaluator only accepts ground instances of the constraint. This is the case, for example, for the CYCLE constraint.

19. ctr_sol(CONSTRAINT_NAME, SIZE, MINVAL, MAXVAL, NSOL, ARG_NSOL)

- SIZE is the number of variables of the collection of variables.
- MINVAL is the smallest value of the variables of the sequence of variables.
- MAXVAL is the largest value of the variables of the sequence of variables.
- NSOL is the total number of solutions to the constraint, assuming a collection with SIZE items and all variables within interval [MINVAL, MAXVAL].

[1] Note that this predicate name should be different from existing **SICStus** built-ins

- `ARG_NSOL` is a list, which for each value that can be assigned to an argument of the constraint, gives the corresponding number of solutions assuming we have a collection with `SIZE` items, and that all variables of the collection are assigned a value in interval [`MINVAL`, `MAXVAL`].

20. `ctr_logic( CONSTRAINT_NAME, LIST_OF_FIRST_ORDER_LOGIC_FORMULAE )`

    - `LIST_OF_FIRST_ORDER_LOGIC_FORMULAE` is a list of first order logical formulae that describe the meaning of the constraint [107].

4.5.2 XML schema associated with a global constraint

In this section we describe an XML schema associated with the global constraint catalogue. We present the motivation for this schema, how it integrates with the description of the constraint in the catalogue, and how the schema information is updated when the catalogue is modified.

Related work

There have been a number of approaches to defining an exchange format for constraint models.

The seminal OPL language [428] provides a modelling language for constraint programs, which is linked to Ilog's solver products. Its use an exchange format is limited by its proprietary background. MiniZinc [304] is a subset of the Zinc modelling language intended to be compiled to multiple solver implementations. First, a model in FlatZinc is generated from the MiniZinc model, removing all iteration (respectively recursion). The flat model can then be compiled into different solver implementations, currently Mercury, **ECLiPSe** and **Gecode**. The development of new back-ends is facilitated by the co-development of the Cadmium [410] term-rewriting system, which can parse and transform FlatZinc code.

The work most closely related to our format probably is the XML format used for the CSP solver competitions [425]. We reviewed an earlier draft version before generating our own schema for the catalogue, the 2007 version (for the 2008 competition) is described in [307]. It is intended as a solver independent format, which can be used by all participants of the competition. As a design choice, the authors decided not to fully structure the format, e.g. to use string values to hold structured information. In order to understand the actual meaning of the model, these strings need to be parsed and analysed as well. This may have size advantages for CSP data given in extensional form, but makes it more complex to check validity of a data file.

Key features

The following list summarises the core features of our XML format and the associated schema:

language independent The underlying description of the constraint in the catalogue is provided as Prolog facts. These may be difficult/tedious to read in other programming languages. The use of XML as an exchange format allows use with most programming languages via provided XML parsers.

machine readable, precise format The format is precisely defined, using XML schema data types throughout, so that validity of a model can be checked with standard XML tools.

one-to-one match with the data format used for the catalog The internal structure of the schema follows the data format for the constraints in the rest of the catalogue. This minimises the need for relearning, once the basic format of the catalogue description has been understood.

detailed description of the allowed format for arguments For each global constraint, the allowed format of the argument is specified in great detail. As the complexity of global constraints increases, this becomes more and more important to simplify the generation of valid problem files.

automated generation of schema from the catalogue data files The schema is automatically generated from the catalogue data files by the simple generator program. This keeps the schema up-to-date with changes of the catalogue, and reduces the task of schema maintenance.

generation of examples for each constraint Example XML files based on the examples in the catalogue can be generated automatically, so that a link to these examples can be added to each catalogue entry.

generation of diagrams describing schema for each constraint At the same time, graph structures of the schema for each constraint can be automatically generated using the *graphviz* [193] tool. This can help a human user to produce XML data for a particular constraint without reading the details of the schema.

Structure of schema

Model The top-level element for the schema is *model*, which contains an optional *variables* element and a required *constraints* element.

Variables The *variables* element consists of a non-empty sequence of *variable* elements, each describing a single variable which may occur in some of the constraints. Each variable has some attributes, an required *id*, an optional *name* and a required *external*. The *id* is an XML schema **ID** used to refer to the variable in the constraints of the model, the *name* is a **string** which describes the variable to the user, and *external* is a “yes”/“no” **string** which states if the variable is visible outside the model.

The domains of variables are not described as part of the variables section, special unary constraints (e.g., IN_INTERVAL) are used in the constraint section instead.

constraints The *constraints* element consists of one or more elements representing constraints in the catalogue. The constraints can be stated in any order, with the understanding that the order may influence the sequence in which they are introduced to the solver.

For each constraint in the catalogue, a specific element with the same name is described in the schema. This imposes restrictions on the names of constraints in the catalogue, only alphanumerical names (with underscores) should be used.

Each constraint has attributes *id* (type **ID**), a *name* (type **string**) and an optional *description* (type **string**). The *name* and *description* can be used to include user-readable information about the constraint, for example, for debugging or explanations.

For each introduced element, a sequence of arguments is defined to define the arguments of the constraints in the same order as described in the catalogue. Each of the arguments has a specific type, which is defined in accordance with the catalogue definition. The argument names can be reused throughout the catalogue, as long as they are unique within each constraint. Arguments can have atomic values (i.e., consist of a single value), or they may be *collection* elements.

collection Roughly, collections correspond to lists in Prolog. Collections can be empty, or must contain entries of the same type. Collections can be nested as required.

item Items correspond to terms in Prolog. Items have named arguments, for which the same rules apply as for the arguments of constraints. The different arguments of an item can be of different type.

Generating schema from the catalogue

There are two programs which can be used to build the schema description from the data describing the catalogue. They should be run whenever a description of a constraint in the catalogue has been changed.

schema.ecl The **ECLiPSe** [11] program *schema.ecl* can be used to re-generate the schema when the catalogue description has been modified. The query *schema.* produces the schema from descriptions in the **src** directory, the query *top.* produces example files for each constraint in the **xml** directory.

The predicate *handle_table* defines which of the restrictions in the constraint description are included as part of the schema information. Many of the more complex rules cannot be easily checked by the schema, an entry in *handle_table* says to ignore the restriction for the moment.

schema_dot.ecl The program *schema_dot.ecl* can be used to generate *graphviz* **dot** files from the schema file *schema.xsd*. The generated files are placed in the **images** directory, and a *dot* command to produce **.png** and **.eps** output is run in the same directory. The pixel based png files are intended for use in web pages, the scalable eps files can be used in LaTeX files producing postscript or pdf documents.

There are some predicates in *schema_dot.ecl* which control the format of the generated graph. They are:

- The predicate *range_style* controls the display of range information, and optional/required choices for attributes.
- The predicate *type_shape* defines the shape and colour of the different elements in the schema for a constraint.
- The predicate *match_builtin* provides an abbreviated element name for some of the predefined element types in the schema. This is required as the graphs should not become too big to fit onto a single A4 page in the output.

Conclusion

We have described the rationale and details for an XML schema attached to the global constraint catalogue. It allows to describe models using the constraints of the catalogue as flat XML files, which are a good exchange format for generating and/or parsing constraint data.

5

Global Constraint Catalogue

Contents

5.1 ABS_VALUE

► ▷ DESCRIPTION LINKS

Origin Arithmetic.

Constraint ABS_VALUE(Y, X)

Usual name ABS

Synonym ABSOLUTE_VALUE.

Arguments

Y : dvar
X : dvar

Restriction $\mathtt{Y} \geq 0$

Purpose Enforce the fact that the first variable is equal to the absolute value of the second variable.

Example $(8, -8)$

The ABS_VALUE constraint holds since 8 is equal to $|-8|$.

All solutions Figure 5.1 gives all solutions to the following non ground instance of the ABS_VALUE constraint: $\mathtt{Y} \in [1, 6]$, $\mathtt{X} \in [-2, 3]$, ABS_VALUE(Y, X).

① $(2, -2)$
② $(1, -1)$
③ $(1, \ \ 1)$
④ $(2, \ \ 2)$
⑤ $(3, \ \ 3)$

Figure 5.1: All solutions corresponding to the non ground example of the ABS_VALUE constraint of the **All solutions** slot

Arg. properties Functional dependency: Y determined by X.

Systems ABS in **Choco**, ABS in **Gecode**.

See also **implied by:** EQ.

implies: GEQ, ZERO_OR_NOT_ZERO.

implies (if swap arguments): OPPOSITE_SIGN, ZERO_OR_NOT_ZERO.

Keywords

constraint arguments: binary constraint, pure functional dependency.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

modelling: functional dependency.

5.2 ALL_BALANCE

► ▷ ◄ DESCRIPTION LINKS

Origin derived from BALANCE in [77]

Constraint ALL_BALANCE(BALANCE, VARIABLES, M)

Arguments

BALANCE	:	dvar
VARIABLES	:	collection(var−dvar)
M	:	int

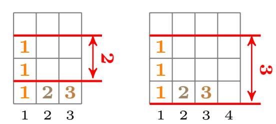
Restrictions

BALANCE ≥ 0
BALANCE $\leq$ |VARIABLES|
required(VARIABLES, var)
VARIABLES.var ≥ 1
VARIABLES.var $\leq$ M
M ≥ 1
M $\leq$ |VARIABLES|

Purpose BALANCE is equal to the difference between the number of occurrence of the value that occurs the most and the value that occurs the least within the collection of variables VARIABLES, where we consider the number of occurrences of each value in $[1, \mathtt{M}]$.

Example

$(2, \langle 3, 1, 2, 1, 1\rangle, 3)$
$(3, \langle 3, 1, 2, 1, 1\rangle, 4)$



1. In the first example, values $1, 2$ and $\mathtt{M} = 3$ are respectively used $3, 1$ and 1 times. The corresponding ALL_BALANCE constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., $3 - 1$).
2. In the first example, values $1, 2, 3$ and $\mathtt{M} = 4$ are respectively used $3, 1, 1$ and 0 times. The corresponding ALL_BALANCE constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., $3 - 0$).

All solutions Figure 5.2 gives all solutions to the following non ground instance of the ALL_BALANCE constraint: BALANCE $\in \{\mathbf{0}, \mathbf{3}\}$, $\mathtt{V}_1 \in [0, 5]$, $\mathtt{V}_2 \in [2, 6]$, $\mathtt{V}_3 \in [0, 1]$, $\mathtt{V}_4 \in [1, 2]$, ALL_BALANCE(BALANCE, $\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4\rangle, 4)$.

Typical

BALANCE $\leq 2 +$ |VARIABLES|$/10$
|VARIABLES| > 2

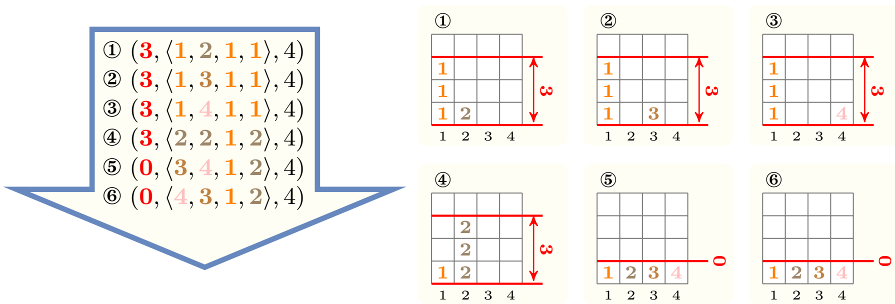


Figure 5.2: All solutions corresponding to the non ground example of the ALL_BALANCE constraint of the **All solutions** slot

Typical model $\texttt{nval}(\texttt{VARIABLES.var}) > 2$

Arg. properties Functional dependency: BALANCE determined by VARIABLES and M.

Usage An application of the ALL_BALANCE constraint is to enforce a *balanced assignment* of values, no matter how many distinct values will be used. In this case one will *push down* the maximum value of the first argument of the ALL_BALANCE constraint.

- On the one hand the ALL_BALANCE constraint should be used on problems where only consecutive values are used. This is the case, for example, in some resource assignment problems where we know in advance that all resources will be employed.
- On the other hand the BALANCE constraint should be used on problems for which, due to some constraints, not all consecutive values will be assigned. This is the case, for example, for some frequency assignment problems where, due to some interference, all used values are not necessarily consecutive.

Algorithm When the lower bound of the BALANCE argument is unconstrained, a flow-based polynomial filtering algorithm for the ALL_BALANCE constraint achieving arc-consistency is described in [77].

Reformulation An efficient reformulation of the ALL_BALANCE constraint is provided in [77]. It is based on the following ingredients:

- First, a GLOBAL_CARDINALITY$(\texttt{VARIABLES}, \langle 1\ o_1, 2\ o_2, \ldots, \texttt{M}\ o_{\texttt{M}}\rangle)$ constraint is stated for exposing the number of occurrences $o_1, o_2, \ldots, o_{\texttt{M}}$ of values $1, 2, \ldots, \texttt{M}$ within the values assigned to VARIABLES.
- Second, a MINIMUM$(min, \langle o_1, o_2, \ldots, o_{\texttt{M}}\rangle)$ and a MAXIMUM$(max, \langle o_1, o_2, \ldots, o_{\texttt{M}}\rangle)$ constraints on variables $o_1, o_2, \ldots, o_{\texttt{M}}$ are stated to get the corresponding smallest and largest values min and max. Then, to express the link with BALANCE, the constraint $\texttt{BALANCE} = max - min$ is stated.
- Third, the following necessary conditions are used:
  (i) $\texttt{M} \cdot max - (\texttt{M} - 1) \cdot \texttt{BALANCE} \leq |\texttt{VARIABLES}|$,
  (ii) $\texttt{M} \cdot min + (\texttt{M} - 1) \cdot \texttt{BALANCE} \geq |\texttt{VARIABLES}|$,

(iii) $\texttt{BALANCE} \neq 1 + \left\lfloor \frac{|\texttt{VARIABLES}|}{\texttt{M}} \right\rfloor - \left\lceil \frac{|\texttt{VARIABLES}|}{\texttt{M}} \right\rceil$.

In the same paper [77], it is also shown how to generate stronger necessary conditions.

See also **related:** BALANCE *(ignore unused values)*.

Keywords **application area:** assignment.

constraint arguments: pure functional dependency.

constraint type: value constraint.

filtering: flow.

modelling: balanced assignment, functional dependency.

5.3 ALL_DIFFER_FROM_AT_LEAST_K_POS

▶ ▷ ◀ DESCRIPTION LINKS GRAPH

Origin Inspired by [188].

Constraint ALL_DIFFER_FROM_AT_LEAST_K_POS(K, VECTORS)

Type VECTOR : collection(var−dvar)

Arguments
K : int
VECTORS : collection(vec − VECTOR)

Restrictions
required(VECTOR, var)
$|\text{VECTOR}| \geq 1$
$|\text{VECTOR}| \geq \text{K}$
$\text{K} \geq 0$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose Enforce all pairs of distinct vectors of the VECTORS collection to differ from at least K positions.

Example $(2, \langle \text{vec} - \langle 2, 5, 2, 0 \rangle, \text{vec} - \langle 3, 6, 2, 1 \rangle, \text{vec} - \langle 3, 6, 1, 0 \rangle \rangle)$

The ALL_DIFFER_FROM_AT_LEAST_K_POS constraint holds since:

- The first and second vectors differ from 3 positions, which is greater than or equal to $\text{K} = 2$.
- The first and third vectors differ from 3 positions, which is greater than or equal to $\text{K} = 2$.
- The second and third vectors differ from 2 positions, which is greater than or equal to $\text{K} = 2$.

Typical
$\text{K} > 0$
$|\text{VECTORS}| > 1$

Symmetries
- Items of VECTORS are permutable.
- Items of VECTORS.vec are permutable (*same permutation used*).

Arg. properties
- Contractible wrt. VECTORS.
- Extensible wrt. VECTORS.vec (*add items at same position*).

See also

implied by: ALL_DIFFER_FROM_EXACTLY_K_POS *($\geq$ `K` replaced by $=$ `K`)*.

part of system of constraints: DIFFER_FROM_AT_LEAST_K_POS.

used in graph description: DIFFER_FROM_AT_LEAST_K_POS.

Keywords

application area: bioinformatics.

characteristic of a constraint: disequality, vector.

constraint type: system of constraints, decomposition.

final graph structure: no loop, symmetric.

Cond. implications

ALL_DIFFER_FROM_AT_LEAST_K_POS(K, VECTORS)
with K $\leq$ |VECTORS|
implies ATLEAST_NVECTOR(NVEC, VECTORS).

Arc input(s) VECTORS

Arc generator $CLIQUE(\neq) \mapsto$ collection(vectors1, vectors2)

Arc arity 2

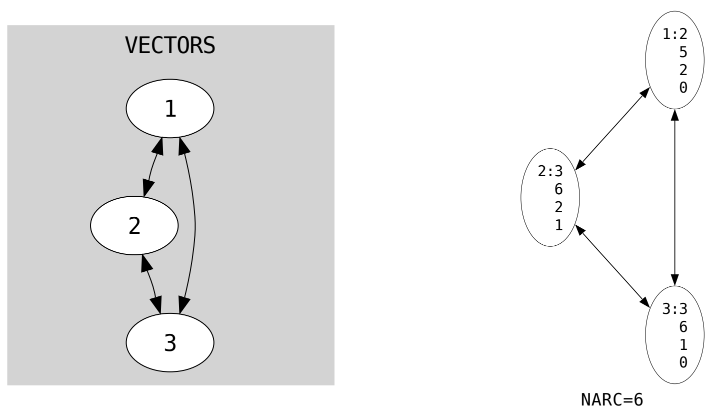
Arc constraint(s) DIFFER_FROM_AT_LEAST_K_POS(K, vectors1.vec, vectors2.vec)

Graph property(ies) $\mathbf{NARC}$= |VECTORS| $*$ |VECTORS| $-$ |VECTORS|

Graph class
- NO_LOOP
- SYMMETRIC

Graph model The **Arc constraint(s)** slot uses the DIFFER_FROM_AT_LEAST_K_POS constraint defined in this catalogue.

Parts (A) and (B) of Figure 5.3 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold. The previous constraint holds since exactly $3 \cdot (3-1) = 6$ arc constraints hold.



(A) **(B)**

Figure 5.3: Initial and final graph of the ALL_DIFFER_FROM_AT_LEAST_K_POS constraint

Signature Since we use the $CLIQUE(\neq)$ arc generator on the items of the VECTORS collection, the expression |VECTORS| $\cdot$ |VECTORS| $-$ |VECTORS| corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\mathbf{NARC}$ = |VECTORS| $\cdot$ |VECTORS| $-$ |VECTORS| to $\mathbf{NARC} \geq$ |VECTORS| $\cdot$ |VECTORS| $-$ |VECTORS|. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.4 ALL_DIFFER_FROM_AT_MOST_K_POS

DESCRIPTION LINKS GRAPH

Origin Inspired by ALL_DIFFER_FROM_AT_LEAST_K_POS.

Constraint ALL_DIFFER_FROM_AT_MOST_K_POS(K, VECTORS)

Type VECTOR : collection(var−dvar)

Arguments

K : int
VECTORS : collection(vec − VECTOR)

Restrictions

required(VECTOR, var)
$|\texttt{VECTOR}| \geq 1$
$|\texttt{VECTOR}| \geq \texttt{K}$
$\texttt{K} \geq 0$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose Enforce all pairs of distinct vectors of the VECTORS collection to differ from at most K positions.

Example $(2, \langle \texttt{vec} - \langle 0, 3, 0, 6\rangle, \texttt{vec} - \langle 0, 3, 4, 1\rangle, \texttt{vec} - \langle 0, 3, 4, 6\rangle\rangle)$

The ALL_DIFFER_FROM_AT_MOST_K_POS constraint holds since:

- The first and second vectors differ from 2 positions, which is less than or equal to $\texttt{K} = 2$.
- The first and third vectors differ from 1 position, which is less than or equal to $\texttt{K} = 2$.
- The second and third vectors differ from 1 position, which is less than or equal to $\texttt{K} = 2$.

Typical

$\texttt{K} > 0$
$\texttt{K} < |\texttt{VECTOR}|$
$|\texttt{VECTORS}| > 1$

Symmetries

- Items of VECTORS are permutable.
- Items of VECTORS.vec are permutable (*same permutation used*).

Arg. properties

- Contractible wrt. VECTORS.
- Contractible wrt. VECTORS.vec (*remove items from same position*).

See also

implied by: ALL_DIFFER_FROM_EXACTLY_K_POS *($\leq$ K replaced by $=$ K)*.

part of system of constraints: DIFFER_FROM_AT_MOST_K_POS.

used in graph description: DIFFER_FROM_AT_MOST_K_POS.

Keywords

characteristic of a constraint: disequality, vector.

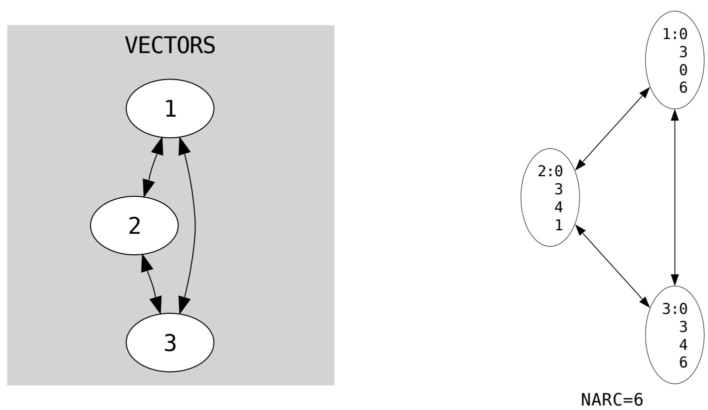
constraint type: system of constraints, decomposition.

final graph structure: no loop, symmetric.

Arc input(s)	VECTORS
Arc generator	$CLIQUE(\neq) \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	DIFFER_FROM_AT_MOST_K_POS(K, vectors1.vec, vectors2.vec)
Graph property(ies)	$\mathbf{NARC}$= |VECTORS| $*$ |VECTORS| $-$ |VECTORS|
Graph class	• NO_LOOP • SYMMETRIC

Graph model The **Arc constraint(s)** slot uses the DIFFER_FROM_AT_MOST_K_POS constraint defined in this catalogue.

Parts (A) and (B) of Figure 5.4 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold. The previous constraint holds since exactly $3 \cdot (3-1) = 6$ arc constraints hold.



(A) (B)

Figure 5.4: Initial and final graph of the ALL_DIFFER_FROM_AT_MOST_K_POS constraint

Signature Since we use the $CLIQUE(\neq)$ arc generator on the items of the VECTORS collection, the expression $|\text{VECTORS}| \cdot |\text{VECTORS}| - |\text{VECTORS}|$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\mathbf{NARC} = |\text{VECTORS}| \cdot |\text{VECTORS}| - |\text{VECTORS}|$ to $\mathbf{NARC} \geq |\text{VECTORS}| \cdot |\text{VECTORS}| - |\text{VECTORS}|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.5 ALL_DIFFER_FROM_EXACTLY_K_POS

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin Inspired by ALL_DIFFER_FROM_AT_LEAST_K_POS.

Constraint ALL_DIFFER_FROM_EXACTLY_K_POS(K, VECTORS)

Type VECTOR : collection(var−dvar)

Arguments

K : int
VECTORS : collection(vec − VECTOR)

Restrictions

required(VECTOR, var)
$|\text{VECTOR}| \geq 1$
$|\text{VECTOR}| \geq \text{K}$
$\text{K} \geq 0$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose Enforce all pairs of distinct vectors of the VECTORS collection to differ from exactly K positions. Enforce $\text{K} = 0$ when $|\text{VECTORS}| < 2$.

Example $(2, \langle \text{vec} - \langle 0,3,0,6\rangle, \text{vec} - \langle 0,3,4,1\rangle, \text{vec} - \langle 9,3,4,6\rangle \rangle)$

The ALL_DIFFER_FROM_EXACTLY_K_POS constraint holds since:

- The first and second vectors differ from 2 positions, which is equal to $\text{K} = 2$.
- The first and third vectors differ from 2 positions, which is equal to $\text{K} = 2$.
- The second and third vectors differ from 2 positions, which is equal to $\text{K} = 2$.

All solutions Consider the following colouring problem where we have to colour each block of a parquet so that in any pair of rows (respectively columns) exactly two adjacent blocks are coloured with the same colour (pink or cyan). The problem is known under the name *parquets anallagmatiques* in [280]. To model this problem we post two ALL_DIFFER_FROM_EXACTLY_K_POS constraints: a first one for the rows and a second one for the columns. To break some symmetry (1) we fix X_{11} to 0 since values 0 and 1 are interchangeable, and (2) we order lexicographically the rows by enforcing a LEX_CHAIN_LESSEQ constraint. Finally to limit the number of solutions we restrict X_{12} to 1. Figure 5.5 provides the six solutions to the previous problem.

$X_{11} \in [0,1]$, $X_{12} \in [0,1]$, $X_{13} \in [0,1]$, $X_{14} \in [0,1]$,
$X_{21} \in [0,1]$, $X_{22} \in [0,1]$, $X_{23} \in [0,1]$, $X_{24} \in [0,1]$,
$X_{31} \in [0,1]$, $X_{32} \in [0,1]$, $X_{33} \in [0,1]$, $X_{34} \in [0,1]$,
$X_{41} \in [0,1]$, $X_{42} \in [0,1]$, $X_{43} \in [0,1]$, $X_{44} \in [0,1]$,
$X_{11} = 0$,

$$
\begin{array}{l}
\text{LEX\_CHAIN\_LESSEQ}\left(\left\langle\begin{array}{l}\langle X_{11},X_{12},X_{13},X_{14}\rangle,\\ \langle X_{21},X_{22},X_{33},X_{24}\rangle,\\ \langle X_{31},X_{32},X_{33},X_{34}\rangle,\\ \langle X_{41},X_{42},X_{43},X_{44}\rangle\end{array}\right\rangle\right),\\
\text{ALL\_DIFFER\_FROM\_EXACTLY\_K\_POS}\left(\left\langle 2,\ \begin{array}{l}\langle X_{11},X_{12},X_{13},X_{14}\rangle,\\ \langle X_{21},X_{22},X_{33},X_{24}\rangle,\\ \langle X_{31},X_{32},X_{33},X_{34}\rangle,\\ \langle X_{41},X_{42},X_{43},X_{44}\rangle\end{array}\right\rangle\right),\\
\text{ALL\_DIFFER\_FROM\_EXACTLY\_K\_POS}\left(\left\langle 2,\ \begin{array}{l}\langle X_{11},X_{21},X_{31},X_{41}\rangle,\\ \langle X_{12},X_{22},X_{32},X_{42}\rangle,\\ \langle X_{13},X_{23},X_{33},X_{43}\rangle,\\ \langle X_{14},X_{24},X_{34},X_{44}\rangle\end{array}\right\rangle\right),\\
X_{12}=1.
\end{array}
$$

① $(\langle\langle 0,1,0,0\rangle,\langle 0,1,1,1\rangle,\langle 1,1,0,1\rangle,\langle 1,1,1,0\rangle\rangle)$
② $(\langle\langle 0,1,0,0\rangle,\langle 1,0,0,0\rangle,\langle 1,1,0,1\rangle,\langle 1,1,1,0\rangle\rangle)$
③ $(\langle\langle 0,1,0,1\rangle,\langle 0,1,1,0\rangle,\langle 1,1,0,0\rangle,\langle 1,1,1,1\rangle\rangle)$
④ $(\langle\langle 0,1,0,1\rangle,\langle 1,0,0,1\rangle,\langle 1,1,0,0\rangle,\langle 1,1,1,1\rangle\rangle)$
⑤ $(\langle\langle 0,1,1,0\rangle,\langle 1,0,1,0\rangle,\langle 1,1,0,0\rangle,\langle 1,1,1,1\rangle\rangle)$
⑥ $(\langle\langle 0,1,1,1\rangle,\langle 1,0,1,1\rangle,\langle 1,1,0,1\rangle,\langle 1,1,1,0\rangle\rangle)$

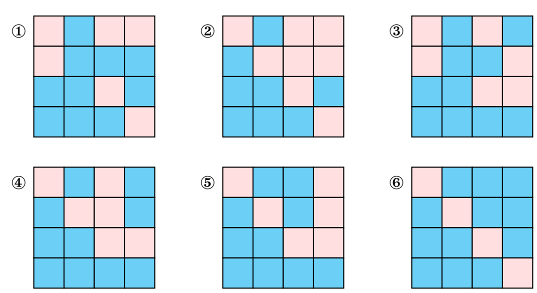


Figure 5.5: Parquet flooring such that for every pair of rows and for every pair of columns we have exactly two adjacent blocks with the same colour (to every variable corresponds a block for which the value is mapped to a colour, pink for 0 and cyan for 1)

Typical

$\texttt{K} > 0$
$\texttt{K} < |\texttt{VECTOR}|$
$|\texttt{VECTORS}| > 1$

Symmetries

- Items of `VECTORS` are permutable.
- Items of `VECTORS.vec` are permutable (*same permutation used*).

Arg. properties Contractible wrt. VECTORS.

See also **implies:** ALL_DIFFER_FROM_AT_LEAST_K_POS (= K *replaced by* $\geq$ K), ALL_DIFFER_FROM_AT_MOST_K_POS (= K *replaced by* $\leq$ K).

part of system of constraints: DIFFER_FROM_EXACTLY_K_POS.

used in graph description: DIFFER_FROM_EXACTLY_K_POS.

Keywords **characteristic of a constraint:** disequality, vector.

constraint type: system of constraints, decomposition.

final graph structure: no loop, symmetric.

Cond. implications ALL_DIFFER_FROM_EXACTLY_K_POS(K, VECTORS)
with K $\leq$ |VECTORS|
implies ATLEAST_NVECTOR(NVEC, VECTORS).

Arc input(s)	VECTORS
Arc generator	$CLIQUE(\neq) \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	DIFFER_FROM_EXACTLY_K_POS(K, vectors1.vec, vectors2.vec)
Graph property(ies)	**NARC**= |VECTORS| $*$ |VECTORS| $-$ |VECTORS|
Graph class	• NO_LOOP • SYMMETRIC

Graph model The **Arc constraint(s)** slot uses the DIFFER_FROM_EXACTLY_K_POS constraint defined in this catalogue.

Parts (A) and (B) of Figure 5.6 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The previous constraint holds since exactly $3 \cdot (3-1) = 6$ arc constraints hold.

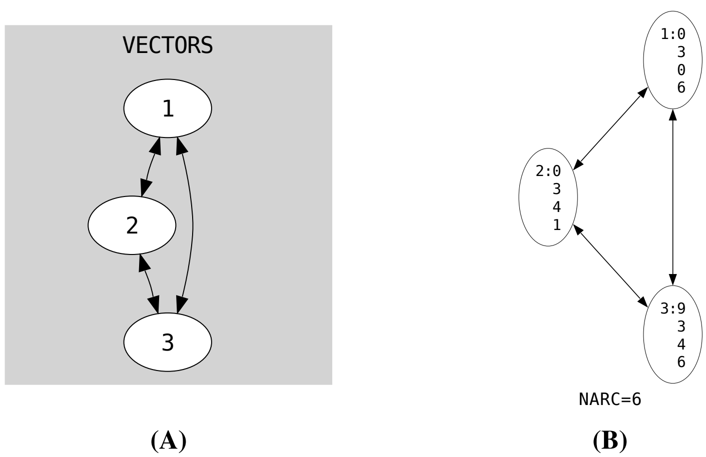


Figure 5.6: Initial and final graph of the ALL_DIFFER_FROM_EXACTLY_K_POS constraint

Signature Since we use the $CLIQUE(\neq)$ arc generator on the items of the VECTORS collection, the expression |VECTORS| $\cdot$ |VECTORS| $-$ |VECTORS| corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property **NARC** $=$ |VECTORS| $\cdot$ |VECTORS| $-$ |VECTORS| to **NARC** $\geq$ |VECTORS| $\cdot$ |VECTORS| $-$ |VECTORS|. This leads to simplify $\overline{\textbf{NARC}}$ to $\overline{\textbf{NARC}}$.

5.6 ALL_EQUAL

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from SOFT_ALL_EQUAL_MIN_CTR

Constraint ALL_EQUAL(VARIABLES)

Synonym REL.

Argument VARIABLES : collection(var−dvar)

Restrictions required(VARIABLES, var)
$|\texttt{VARIABLES}| > 0$

Purpose Enforce all variables of the collection VARIABLES to take the same value.

Example $(\langle 5, 5, 5, 5\rangle)$

The ALL_EQUAL constraint holds since all its variables are fixed to value 5.

All solutions Figure 5.7 gives all solutions to the following non ground instance of the ALL_EQUAL constraint: $\texttt{V}_1 \in [0, 6]$, $\texttt{V}_2 \in [0, 2]$, $\texttt{V}_3 \in [0, 2]$, $\texttt{V}_4 \in [1, 4]$, ALL_EQUAL$(\langle \texttt{V}_1, \texttt{V}_2, \texttt{V}_3, \texttt{V}_4\rangle)$.

① $(\langle 1, 1, 1, 1\rangle)$
② $(\langle 2, 2, 2, 2\rangle)$

Figure 5.7: All solutions corresponding to the non ground example of the ALL_EQUAL constraint of the **All solutions** slot

Typical $|\texttt{VARIABLES}| > 2$
minval(VARIABLES.var) $\neq 0$

Typical model nval(VARIABLES.var) > 2

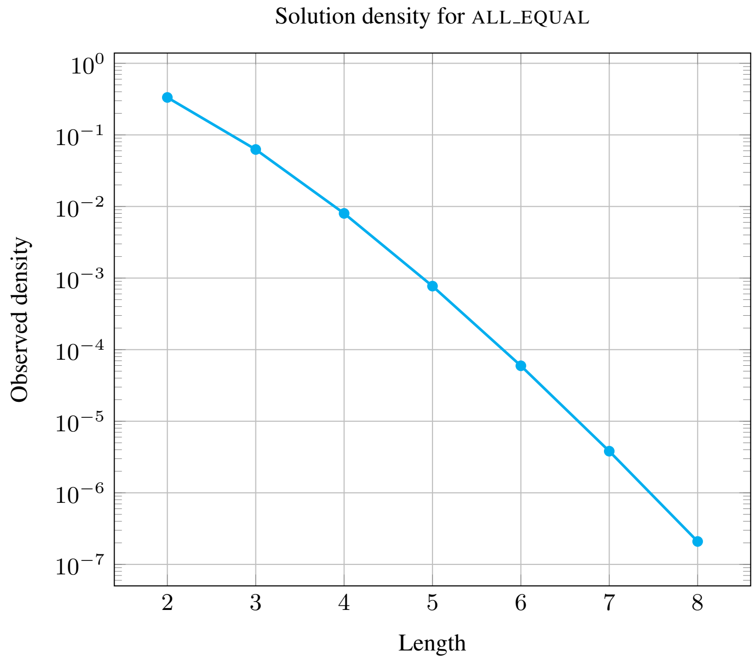
Symmetries
- Items of VARIABLES are permutable.
- All occurrences of a value of VARIABLES.var can be renamed to any unused value.

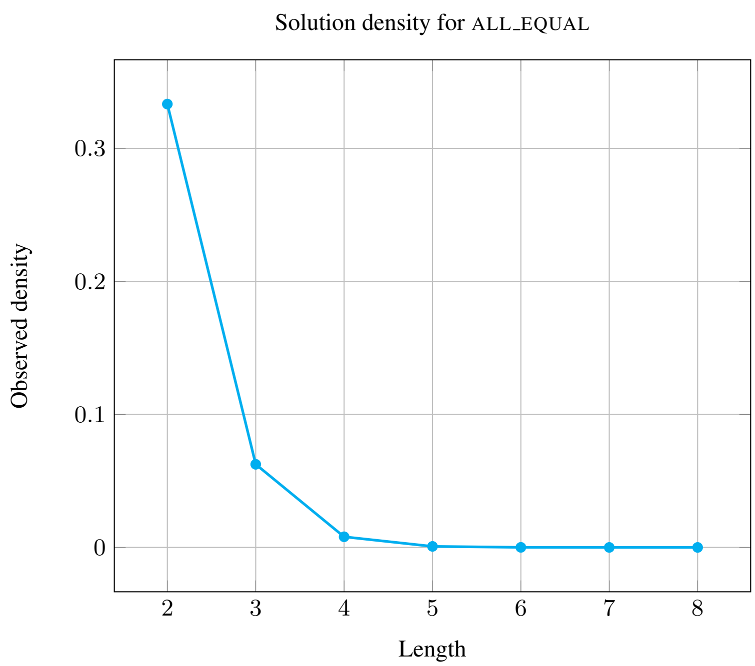
Arg. properties Contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	4	5	6	7	8	9

Number of solutions for ALL_EQUAL: domains $0..n$





Systems ATMOSTNVALUE in **Choco**, REL in **Gecode**, ALL_EQUAL in **MiniZinc**.

See also **generalisation:** NVALUE *(a variable counting the number of distinct values is introduced)*.

implies: ALL_EQUAL_EXCEPT_0, CONSECUTIVE_VALUES, DECREASING, INCREASING, MULTI_GLOBAL_CONTIGUITY.

negation: NOT_ALL_EQUAL.

soft variant: SOFT_ALL_EQUAL_MAX_VAR, SOFT_ALL_EQUAL_MIN_CTR *(decomposition-based violation measure)*, SOFT_ALL_EQUAL_MIN_VAR *(variable-based violation measure)*.

specialisation: EQ *(equality between just two variables)*.

Keywords **constraint type:** value constraint.

Cond. implications ALL_EQUAL(VARIABLES)
with $|\texttt{VARIABLES}| > 1$
implies SOME_EQUAL(VARIABLES).

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NARC** = |VARIABLES| − 1

Graph model We use the arc generator $PATH$ in order to link consecutive variables of the collection VARIABLES by a binary equality constraint.

Parts (A) and (B) of Figure 5.8 respectively show the initial and final graph of the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

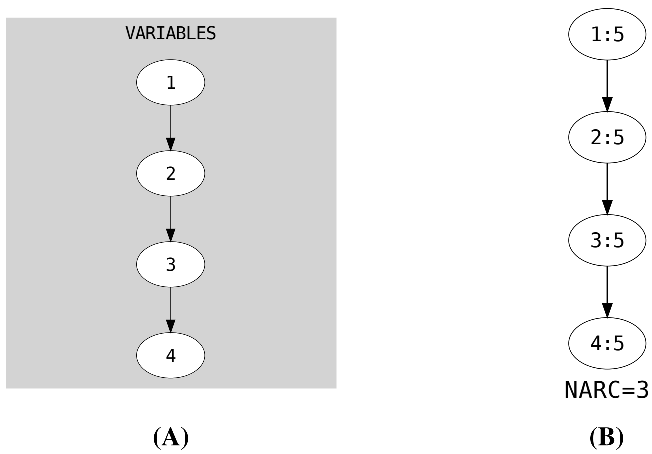


Figure 5.8: Initial and final graph of the ALL_EQUAL constraint

5.7 ALL_EQUAL_EXCEPT_0

▶ ▷ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from ALL_EQUAL

Constraint ALL_EQUAL_EXCEPT_0(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions
required(VARIABLES, var)
$|\texttt{VARIABLES}| > 0$

Purpose Enforce all variables of the collection VARIABLES that are different from 0 to take the same value.

Example $(\langle 5, 0, 5, 5\rangle)$

The ALL_EQUAL_EXCEPT_0 constraint holds since all its variables are fixed to values 0 or 5.

Typical $|\texttt{VARIABLES}| > 1$

Typical model
ATLEAST(2, VARIABLES, 0)
nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var that are both different from 0 can be swapped; all occurrences of a value of VARIABLES.var that is different from 0 can be renamed to any unused value that is also different from 0.

Arg. properties Contractible wrt. VARIABLES.

See also **implied by:** ALL_EQUAL, GLOBAL_CONTIGUITY.

Keywords **characteristic of a constraint:** joker value, automaton, automaton with counters.

constraint type: value constraint.

Automaton

Figure 5.9 depicts an automaton that only accepts all the solutions to the ALL_EQUAL_EXCEPT_0 constraint. This automaton uses a counter in order to record the value of the first non-zero variable $\mathtt{VAR}_i$ already encountered. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i \neq 0 \Leftrightarrow S_i$.

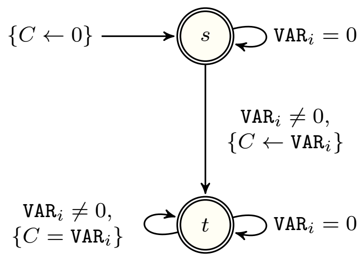


Figure 5.9: Automaton (with one counter) of the ALL_EQUAL_EXCEPT_0 constraint

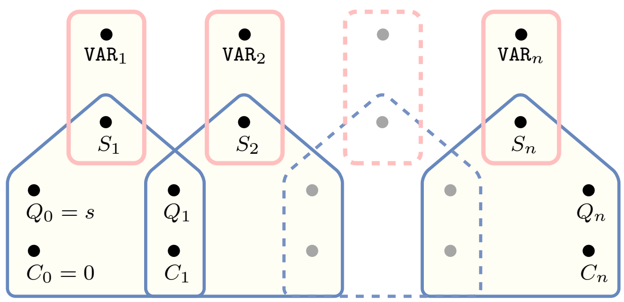


Figure 5.10: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the ALL_EQUAL_EXCEPT_0 constraint

5.8 ALL_EQUAL_PEAK

▶ ▷ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from PEAK and ALL_EQUAL.

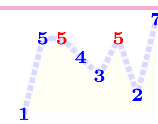
Constraint ALL_EQUAL_PEAK(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions
|VARIABLES| > 0
required(VARIABLES, var)

Purpose
A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \ldots, V_m$ is a *peak* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k > V_{k+1}$.
Enforce all the peaks of the sequence VARIABLES to be assigned the same value, i.e. to be located at the same altitude.

Example $(\langle 1, 5, 5, 4, 3, 5, 2, 7\rangle)$



The ALL_EQUAL_PEAK constraint holds since the two peaks, in bold, of the sequence 1 5 **5** 4 3 **5** 2 7 are located at the same altitude 5. Figure 5.11 depicts the solution associated with the example.

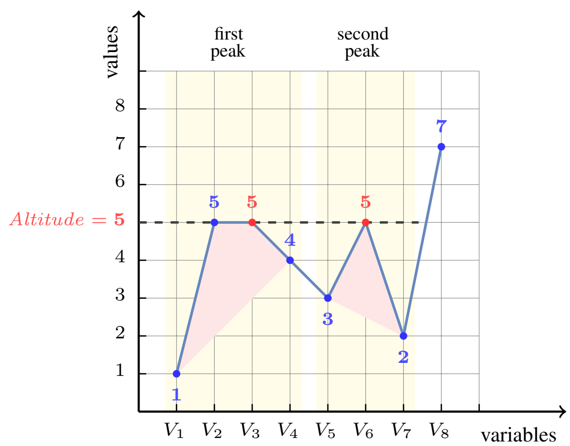


Figure 5.11: Illustration of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values 1, 5, 5, 4, 3, 5, 2, 7 and its corresponding two peaks, in red, both located at altitude 5

Note that the ALL_EQUAL_PEAK constraint does not enforce that the maximum value of the sequence VARIABLES corresponds to the altitude of its peaks since, as shown by the example, the sequence can ends up with an increasing subsequence that go beyond the altitude of its peaks. It also does not enforce that the sequence VARIABLES contains at least one peak.

All solutions

Figure 5.12 gives all solutions to the following non ground instance of the ALL_EQUAL_PEAK constraint: $\mathtt{V}_1 \in \{0,5\}$, $\mathtt{V}_2 \in [2,3]$, $\mathtt{V}_3 = 2$, $\mathtt{V}_4 \in [3,4]$, $\mathtt{V}_5 = 1$, ALL_EQUAL_PEAK$(\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4, \mathtt{V}_5 \rangle)$.

① $(\langle 0,2,2,\mathbf{3},1\rangle)$
② $(\langle 0,2,2,\mathbf{4},1\rangle)$
③ $(\langle 0,\mathbf{3},2,\mathbf{3},1\rangle)$
④ $(\langle 5,2,2,\mathbf{3},1\rangle)$
⑤ $(\langle 5,2,2,\mathbf{4},1\rangle)$
⑥ $(\langle 5,\mathbf{3},2,\mathbf{3},1\rangle)$
⑦ $(\langle 5,3,2,\mathbf{4},1\rangle)$

Figure 5.12: All solutions corresponding to the non ground example of the ALL_EQUAL_PEAK constraint of the **All solutions** slot where each peak is coloured in orange

Typical

$|\mathtt{VARIABLES}| \geq 5$
range(VARIABLES.var) > 1
PEAK(VARIABLES.var) ≥ 2

Typical model

nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

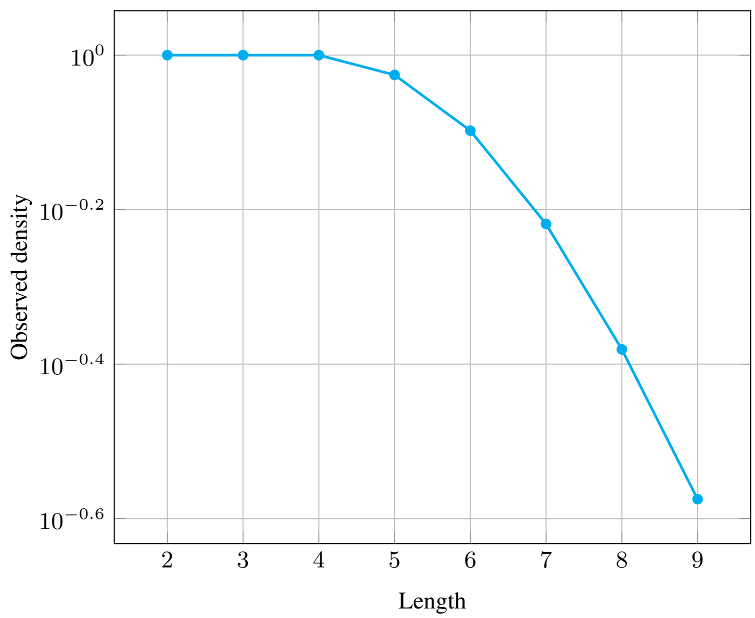
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Counting

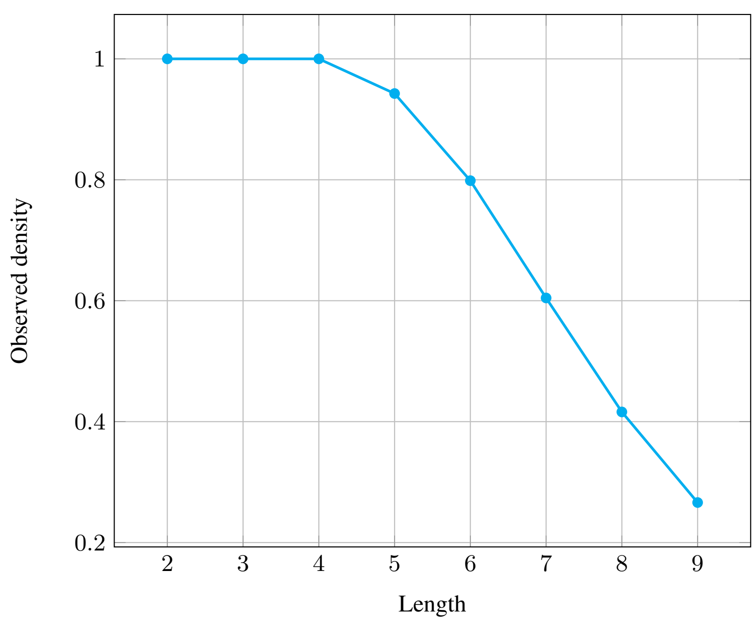
Length (n)	2	3	4	5	6	7	8	9
Solutions	9	64	625	7330	93947	1267790	17908059	266201992

Number of solutions for ALL_EQUAL_PEAK: domains $0..n$

Solution density for ALL_EQUAL_PEAK



Solution density for ALL_EQUAL_PEAK



See also

implied by: ALL_EQUAL_PEAK_MAX.

implies: DECREASING_PEAK, INCREASING_PEAK.

related: ALL_EQUAL_VALLEY, PEAK.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

Cond. implications

- ALL_EQUAL_PEAK(VARIABLES)
  with PEAK(VARIABLES.var) > 1
  implies SOME_EQUAL(VARIABLES).

- ALL_EQUAL_PEAK(VARIABLES)
  with PEAK(VARIABLES.var) > 0
  implies NOT_ALL_EQUAL(VARIABLES).

Automaton

Figure 5.13 depicts the automaton associated with the ALL_EQUAL_PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: initial stationary or decreasing mode	$(\{= \mid >\}^*)$
i	: increasing (before first potential peak) mode	$(< \{< \mid =\}^*)$
j	: decreasing (after a peak) mode	$(> \{> \mid =\}^*)$
k	: increasing (after a peak) mode	$(< \{< \mid =\}^*)$

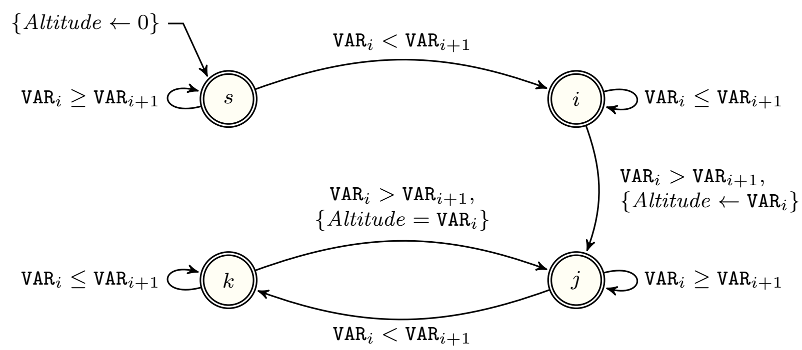


Figure 5.13: Automaton for the ALL_EQUAL_PEAK constraint (note the conditional transition from state k to state j testing that the counter $Altitude$ is equal to $\mathtt{VAR}_i$ for enforcing that all peaks are located at the same altitude)

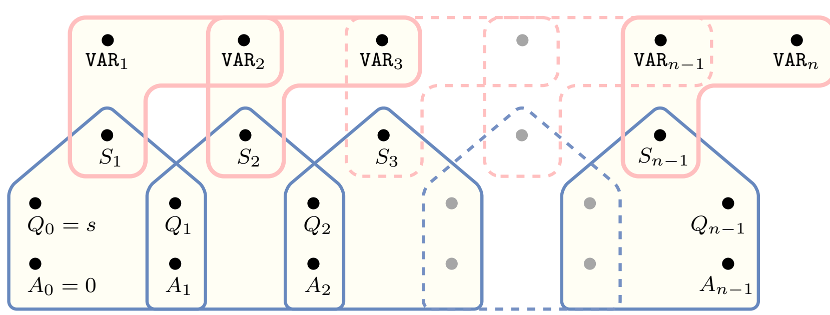


Figure 5.14: Hypergraph of the reformulation corresponding to the automaton of the ALL_EQUAL_PEAK constraint where A_i stands for the value of the counter $Altitude$ (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.9 ALL_EQUAL_PEAK_MAX

DESCRIPTION LINKS AUTOMATON

Origin Derived from PEAK and ALL_EQUAL.

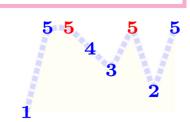
Constraint ALL_EQUAL_PEAK_MAX(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions
|VARIABLES| > 0
required(VARIABLES, var)

Purpose
A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \ldots, V_m$ is a *peak* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k > V_{k+1}$.
Enforce all the peaks of the sequence VARIABLES to be assigned the same value, i.e. to be located at the same altitude corresponding to the maximum value of the sequence VARIABLES.

Example $(\langle 1, 5, 5, 4, 3, 5, 2, 5 \rangle)$



The ALL_EQUAL_PEAK_MAX constraint holds since the two peaks, in bold, of the sequence 1 5 **5** 4 3 **5** 2 5 are located at the same altitude 5 that is also the maximum value of the sequence 1 5 5 4 3 5 2 5. Figure 5.15 depicts the solution associated with the example.

Note that the ALL_EQUAL_PEAK_MAX constraint does not enforce that the sequence VARIABLES contains at least one peak.

Typical
|VARIABLES| ≥ 5
range(VARIABLES.var) > 1
PEAK(VARIABLES.var) ≥ 2

Typical model nval(VARIABLES.var) > 2

Symmetries
- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Counting

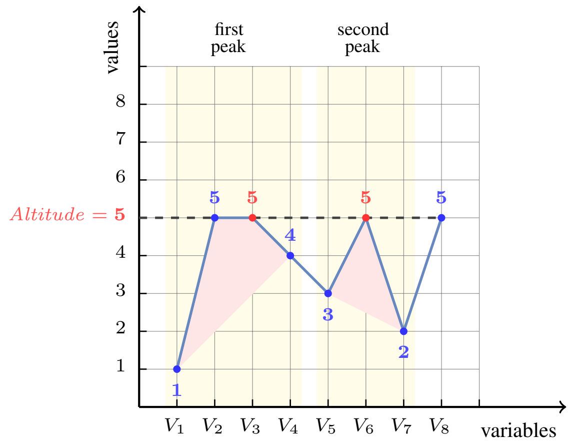
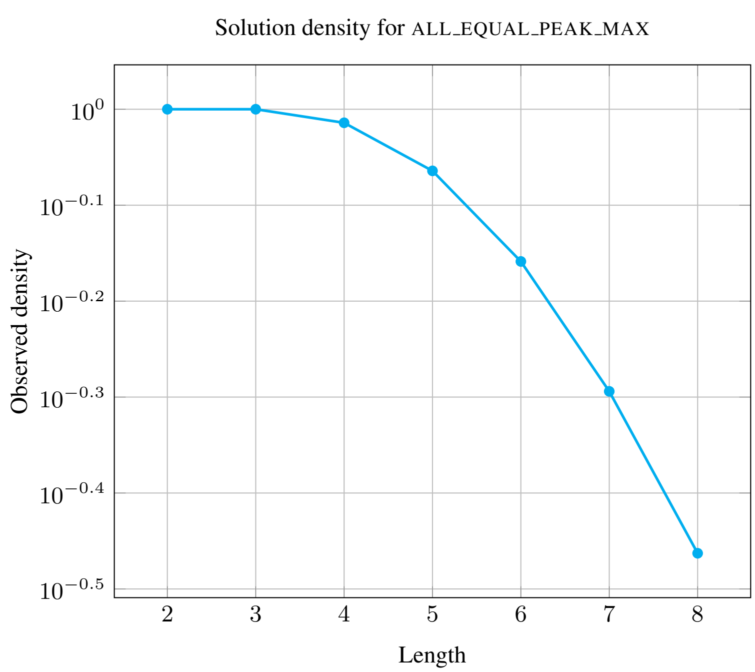
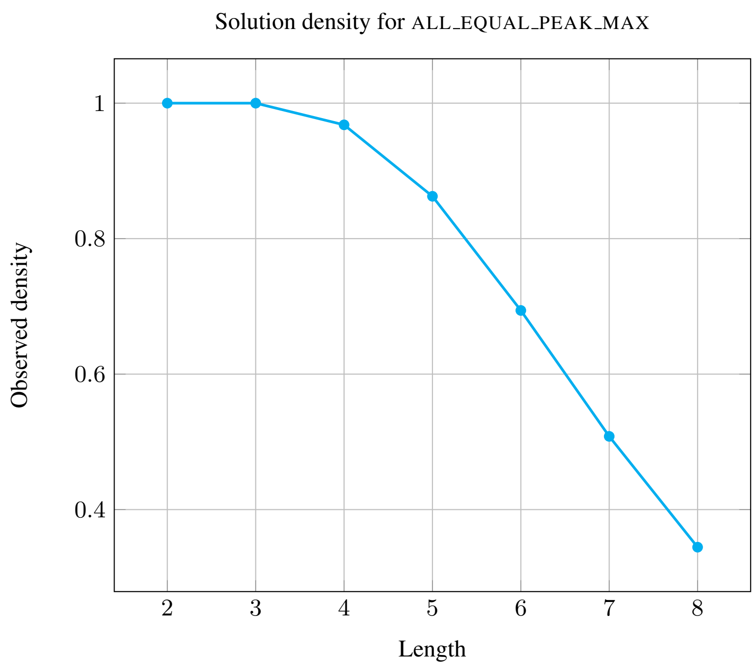


Figure 5.15: Illustration of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values 1, 5, 5, 4, 3, 5, 2, 5 and its corresponding two peaks, in red, both located at altitude 5 that also corresponds to the maximum value of the sequence

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	605	6707	81648	1065542	14829903

Number of solutions for ALL_EQUAL_PEAK_MAX: domains $0..n$





See also **implied by:** NO_PEAK.

implies: ALL_EQUAL_PEAK.

related: ALL_EQUAL_VALLEY_MIN, PEAK.

Keywords **characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

Cond. implications

- ALL_EQUAL_PEAK_MAX($\texttt{VARIABLES}$)
  with PEAK($\texttt{VARIABLES.var}$) > 1
  implies SOME_EQUAL($\texttt{VARIABLES}$).

- ALL_EQUAL_PEAK_MAX($\texttt{VARIABLES}$)
  with PEAK($\texttt{VARIABLES.var}$) > 0
  implies NOT_ALL_EQUAL($\texttt{VARIABLES}$).

Automaton Figure 5.16 depicts the automaton associated with the ALL_EQUAL_PEAK_MAX constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: initial stationary or decreasing mode	$(\{= \mid >\}^*)$
i	: increasing (before first potential peak) mode	$(< \{< \mid =\}^*)$
j	: decreasing (after a peak) mode	$(> \{> \mid =\}^*)$
k	: increasing (after a peak) mode	$(< \{< \mid =\}^*)$

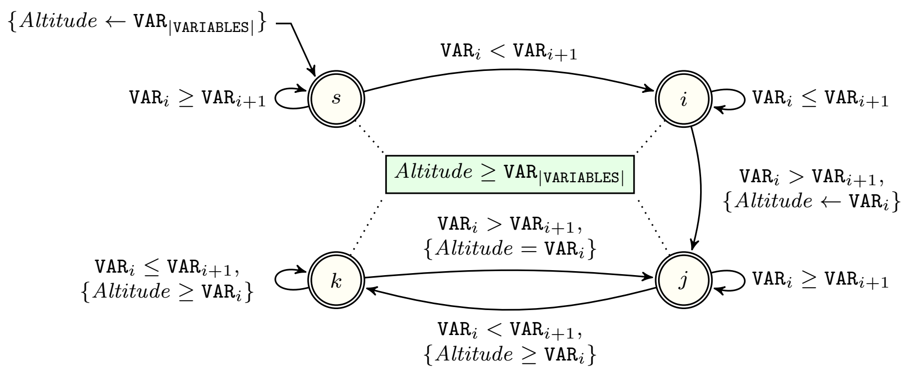


Figure 5.16: Automaton for the ALL_EQUAL_PEAK_MAX constraint; note the conditional transition from state k to state j testing that the counter $Altitude$ is equal to $\mathtt{VAR}_i$ for enforcing that all peaks are located at the same altitude; the conditional transitions from j to k and from k to k and the final check $Altitude \geq \mathtt{VAR}_{|\mathtt{VARIABLES}|}$ enforce the maximum value of the sequence VARIABLES to not exceed the altitude of the eventual peaks.

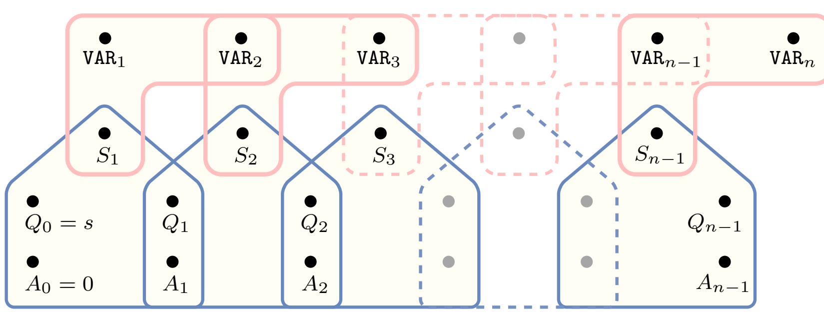


Figure 5.17: Hypergraph of the reformulation corresponding to the automaton of the ALL_EQUAL_PEAK_MAX constraint where A stands for the value of the counter $Altitude$ (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.10 ALL_EQUAL_VALLEY

▶ ▷ ◀ DESCRIPTION LINKS AUTOMATON

Origin

Derived from VALLEY and ALL_EQUAL.

Constraint

ALL_EQUAL_VALLEY(VARIABLES)

Argument

VARIABLES : collection(var−dvar)

Restrictions

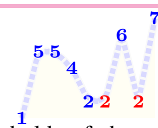
|VARIABLES| > 0
required(VARIABLES, var)

Purpose

A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \ldots, V_m$ is a *valley* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k < V_{k+1}$.
Enforce all the valleys of the sequence VARIABLES to be assigned the same value, i.e. to be located at the same altitude.

Example

$(\langle 1, 5, 5, 4, 2, 2, 6, 2, 7\rangle)$



The ALL_EQUAL_VALLEY constraint holds since the two valleys, in bold, of the sequence 1 5 5 4 2 2 6 2 7 are located at the same altitude 2. Figure 5.18 depicts the solution associated with the example.

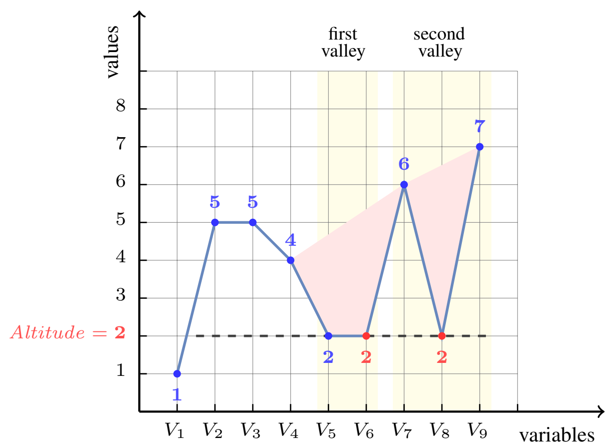


Figure 5.18: Illustration of the **Example** slot: a sequence of nine variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9 respectively fixed to values 1, 5, 5, 4, 2, 2, 6, 2, 7 and its corresponding two valleys, in red, both located at altitude 2

Note that the ALL_EQUAL_VALLEY constraint does not enforce that the minimum value of the sequence VARIABLES corresponds to the altitude of its valleys since, as shown by the example, the sequence can starts with an increasing subsequence that start below the altitude of its valleys. It also does not enforce that the sequence VARIABLES contains at least one valley.

All solutions

Figure 5.19 gives all solutions to the following non ground instance of the ALL_EQUAL_VALLEY constraint: $\mathtt{V}_1 \in \{0, 5\}$, $\mathtt{V}_2 \in [2, 3]$, $\mathtt{V}_3 = 4$, $\mathtt{V}_4 \in [1, 2]$, $\mathtt{V}_5 \in [4, 5]$, ALL_EQUAL_VALLEY$(\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4, \mathtt{V}_5 \rangle)$.

① $(\langle 0, 2, 4, \mathbf{1}, 4 \rangle)$
② $(\langle 0, 2, 4, \mathbf{1}, 5 \rangle)$
③ $(\langle 0, 2, 4, \mathbf{2}, 4 \rangle)$
④ $(\langle 0, 2, 4, \mathbf{2}, 5 \rangle)$
⑤ $(\langle 0, 3, 4, \mathbf{1}, 4 \rangle)$

⑥ $(\langle 0, 3, 4, \mathbf{1}, 5 \rangle)$
⑦ $(\langle 0, 3, 4, \mathbf{2}, 4 \rangle)$
⑧ $(\langle 0, 3, 4, \mathbf{2}, 5 \rangle)$
⑨ $(\langle 5, \mathbf{2}, 4, \mathbf{2}, 4 \rangle)$
⑩ $(\langle 5, \mathbf{2}, 4, \mathbf{2}, 5 \rangle)$

Figure 5.19: All solutions corresponding to the non ground example of the ALL_EQUAL_VALLEY constraint of the **All solutions** slot where each valley is coloured in orange

Typical

$|\mathtt{VARIABLES}| \geq 5$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 1$
$\mathrm{VALLEY}(\mathtt{VARIABLES.var}) \geq 2$

Typical model

$\mathtt{nval}(\mathtt{VARIABLES.var}) > 2$

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

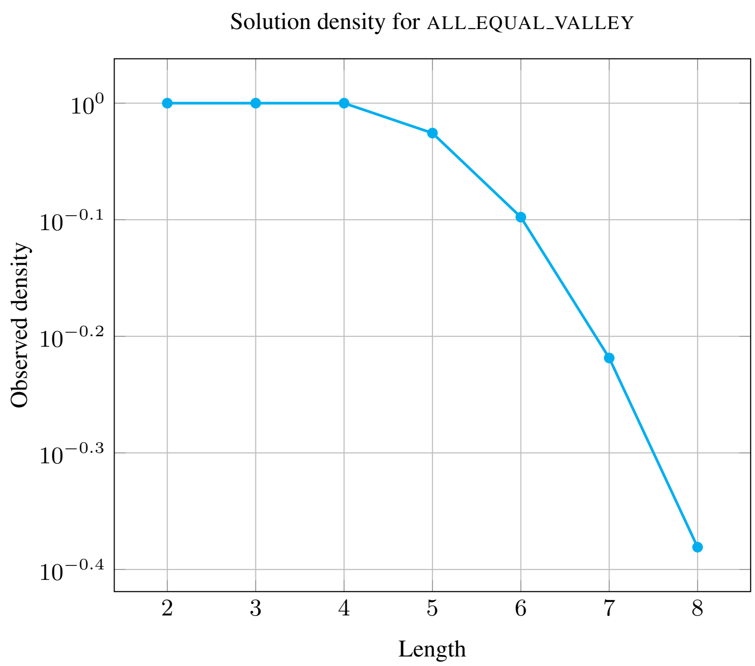
Arg. properties

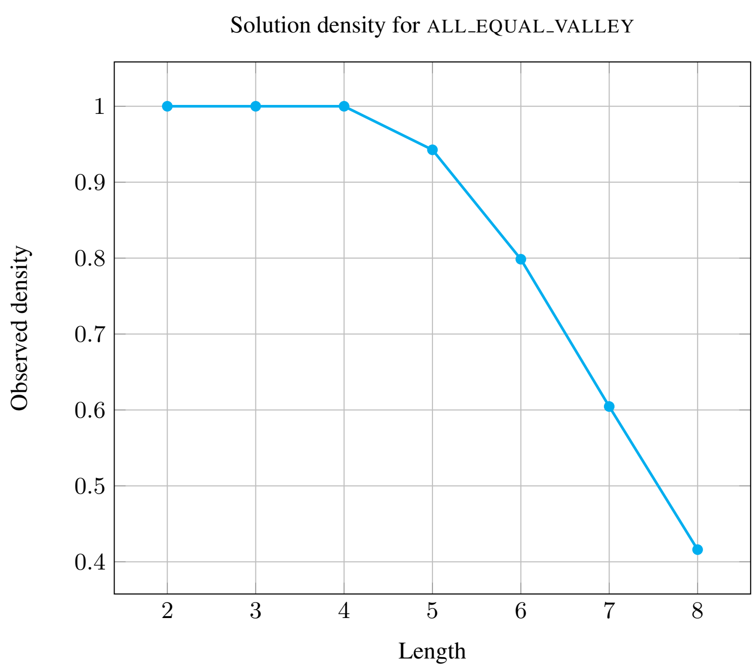
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7330	93947	1267790	17908059

Number of solutions for ALL_EQUAL_VALLEY: domains $0..n$





See also

implied by: ALL_EQUAL_VALLEY_MIN.

implies: DECREASING_VALLEY, INCREASING_VALLEY.

related: ALL_EQUAL_PEAK, VALLEY.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

Cond. implications

- ALL_EQUAL_VALLEY(VARIABLES)
  with VALLEY(VARIABLES.var) > 1
  implies SOME_EQUAL(VARIABLES).

- ALL_EQUAL_VALLEY(VARIABLES)
  with VALLEY(VARIABLES.var) > 0
  implies NOT_ALL_EQUAL(VARIABLES).

Automaton Figure 5.20 depicts the automaton associated with the ALL_EQUAL_VALLEY constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and S_i: $(\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

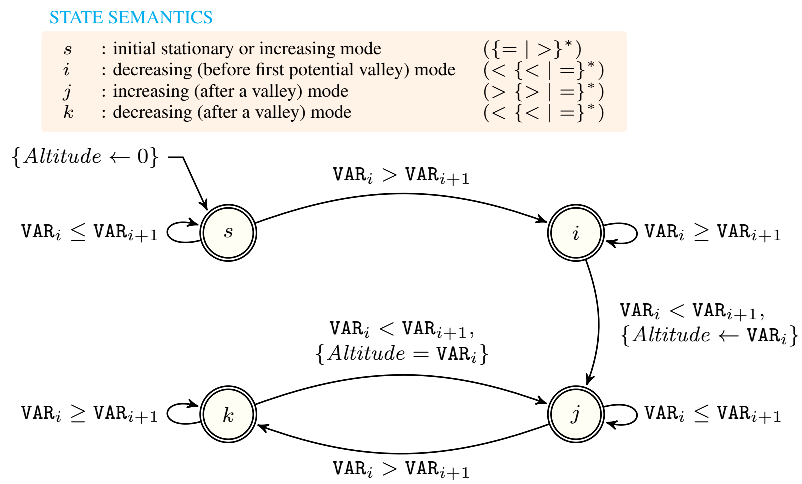


Figure 5.20: Automaton for the ALL_EQUAL_VALLEY constraint (note the conditional transition from state k to state j testing that the counter $Altitude$ is equal to $\texttt{VAR}_i$ for enforcing that all valleys are located at the same altitude)

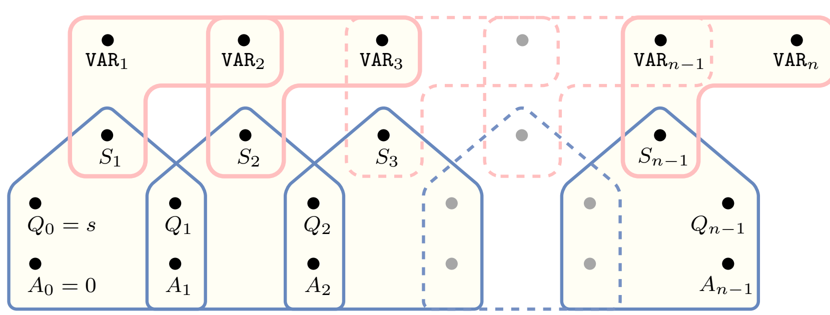


Figure 5.21: Hypergraph of the reformulation corresponding to the automaton of the ALL_EQUAL_VALLEY constraint where A_i stands for the value of the counter $Altitude$ (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.11 ALL_EQUAL_VALLEY_MIN

► ▷ ◄ DESCRIPTION LINKS AUTOMATON

Origin

Derived from VALLEY and ALL_EQUAL.

Constraint

ALL_EQUAL_VALLEY_MIN(VARIABLES)

Argument

VARIABLES : collection(var−dvar)

Restrictions

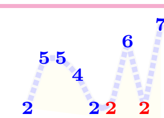
$|\texttt{VARIABLES}| > 0$
required(VARIABLES, var)

Purpose

A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \ldots, V_m$ is a *valley* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k < V_{k+1}$.
Enforce all the valleys of the sequence VARIABLES to be assigned the same value, i.e. to be located at the same altitude corresponding to the minimum value of the sequence VARIABLES.

Example

$(\langle 2, 5, 5, 4, 2, 2, 6, 2, 7\rangle)$



The ALL_EQUAL_VALLEY_MIN constraint holds since the two valleys, in bold, of the sequence 2 5 5 4 2 **2** 6 **2** 7 are located at the same altitude 2 that is also the minimum value of the sequence 2 5 5 4 2 2 6 2 7. Figure 5.22 depicts the solution associated with the example.

Note that the ALL_EQUAL_VALLEY_MIN constraint does not enforce that the sequence VARIABLES contains at least one valley.

Typical

$|\texttt{VARIABLES}| \geq 5$
range(VARIABLES.var) > 1
VALLEY(VARIABLES.var) ≥ 2

Typical model

nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Counting

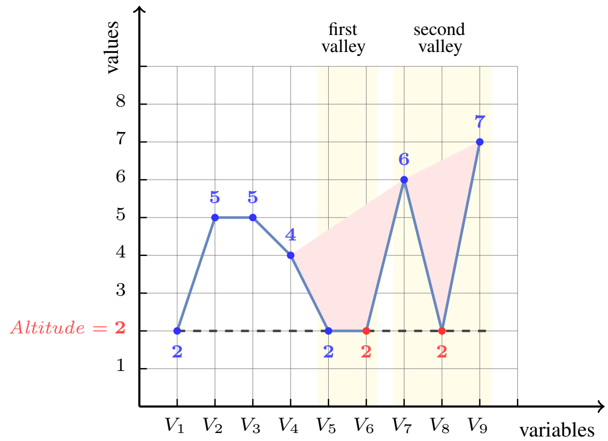
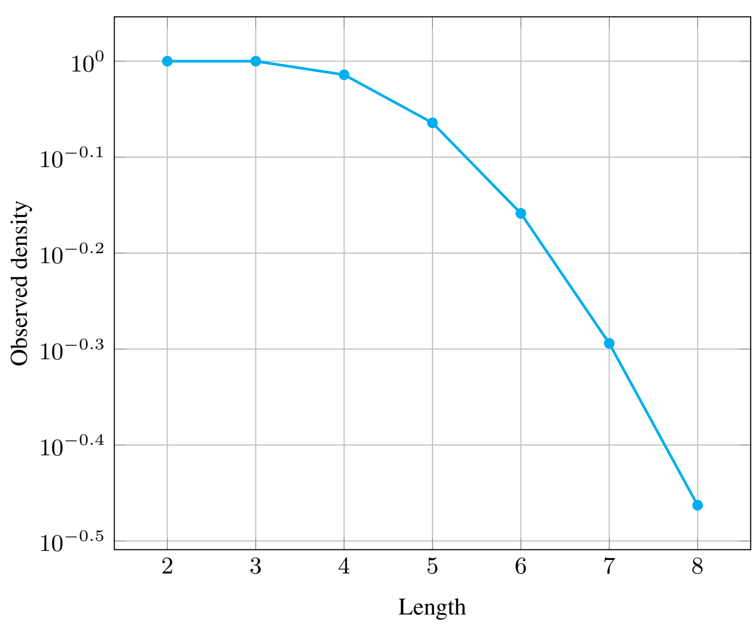


Figure 5.22: Illustration of the **Example** slot: a sequence of nine variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9 respectively fixed to values 2, 5, 5, 4, 2, 2, 6, 2, 7 and its corresponding two valleys, in red, both located at altitude 2 that also corresponds to the minimum value of the sequence

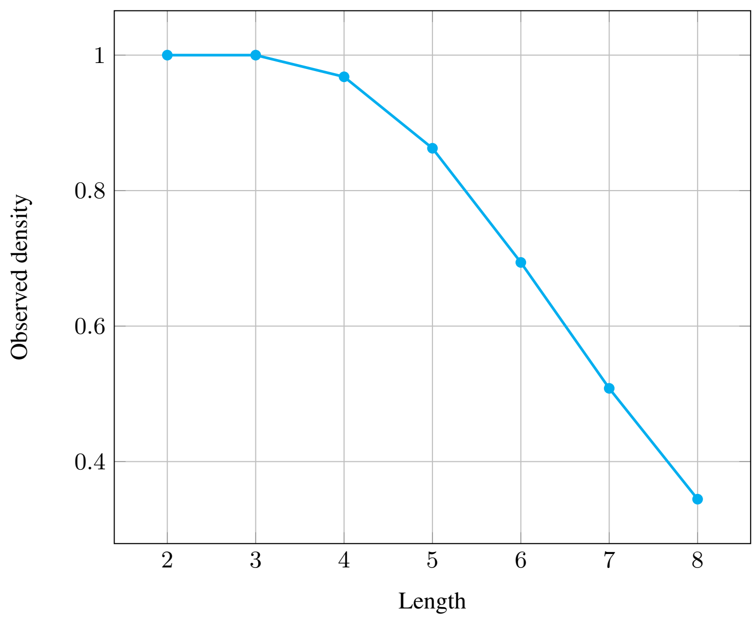
Length (n)	2	3	4	5	6	7	8
Solutions	9	64	605	6707	81648	1065542	14829903

Number of solutions for ALL_EQUAL_VALLEY_MIN: domains $0..n$

Solution density for ALL_EQUAL_VALLEY_MIN



Solution density for ALL_EQUAL_VALLEY_MIN



See also

implied by: NO_VALLEY.

implies: ALL_EQUAL_VALLEY.

related: ALL_EQUAL_PEAK_MAX, VALLEY.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

Cond. implications

- ALL_EQUAL_VALLEY_MIN(VARIABLES)
  with VALLEY(VARIABLES.var) > 1
  implies SOME_EQUAL(VARIABLES).

- ALL_EQUAL_VALLEY_MIN(VARIABLES)
  with VALLEY(VARIABLES.var) > 0
  implies NOT_ALL_EQUAL(VARIABLES).

Automaton Figure 5.23 depicts the automaton associated with the ALL_EQUAL_VALLEY_MIN constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: initial stationary or increasing mode	$(\{= \mid >\}^*)$
i	: decreasing (before first potential valley) mode	$(< \{< \mid =\}^*)$
j	: increasing (after a valley) mode	$(> \{> \mid =\}^*)$
k	: decreasing (after a valley) mode	$(< \{< \mid =\}^*)$

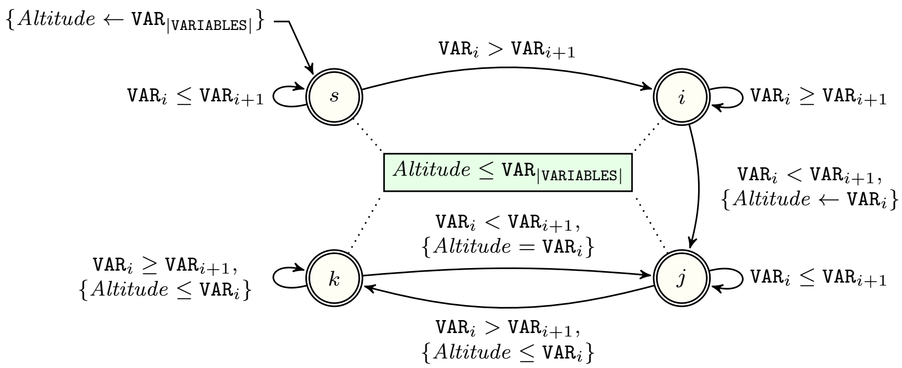


Figure 5.23: Automaton for the ALL_EQUAL_VALLEY_MIN constraint; note the conditional transition from state k to state j testing that the counter $Altitude$ is equal to $\mathtt{VAR}_i$ for enforcing that all valleys are located at the same altitude; the conditional transitions from j to k and from k to k and the final check $Altitude \leq \mathtt{VAR}_{|\mathtt{VARIABLES}|}$ enforce the minimum value of the sequence VARIABLES to not be located below the altitude of the eventual valleys.

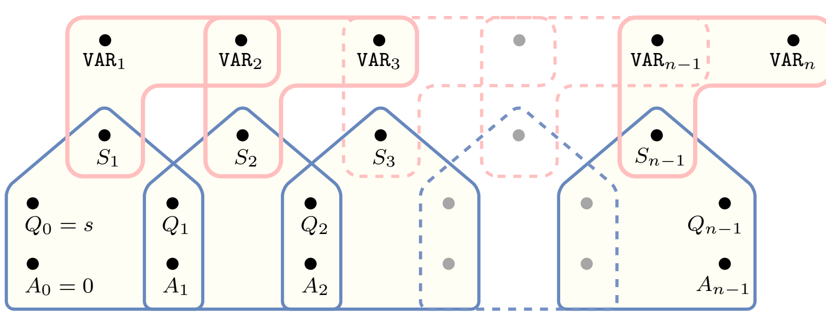


Figure 5.24: Hypergraph of the reformulation corresponding to the automaton of the ALL_EQUAL_VALLEY_MIN constraint where A stands for the value of the counter $Altitude$ (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.12 ALL_INCOMPARABLE

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin Inspired by incomparable rectangles.

Constraint ALL_INCOMPARABLE(VECTORS)

Synonym ALL_INCOMPARABLES.

Type VECTOR : collection(var−dvar)

Argument VECTORS : collection(vec − VECTOR)

Restrictions
required(VECTOR, var)
$|\text{VECTOR}| \geq 1$
required(VECTORS, vec)
$|\text{VECTORS}| \geq 1$
same_size(VECTORS, vec)

Purpose Enforce for each pair of distinct vectors of the VECTORS collection the fact that they are incomparable. Two vectors VECTOR1 and VECTOR2 are incomparable if and only, when the components of both vectors are ordered, and respectively denoted by SVECTOR1 and SVECTOR2, we neither have $\text{SVECTOR1}[i].\text{var} \leq \text{SVECTOR2}[i].\text{var}$ (for all $i \in [1, |\text{SVECTOR1}|]$) nor have $\text{SVECTOR2}[i].\text{var} \leq \text{SVECTOR1}[i].\text{var}$ (for all $i \in [1, |\text{SVECTOR1}|]$).

Example

$$\left(\begin{array}{l}\text{vec} - \langle 1, 18\rangle, \\ \text{vec} - \langle 2, 16\rangle, \\ \text{vec} - \langle 3, 13\rangle, \\ \text{vec} - \langle 4, 11\rangle, \\ \text{vec} - \langle 5, 10\rangle, \\ \text{vec} - \langle 6, 9\rangle, \\ \text{vec} - \langle 7, 7\rangle\end{array}\right)$$

The ALL_INCOMPARABLE constraint holds since all distinct pairs of vectors are incomparable as illustrated by Figure 5.25.

All solutions Figure 5.26 gives all solutions to the following non ground instance of the ALL_INCOMPARABLE constraint: $U_1 \in [1, 2]$, $V_1 \in [0, 5]$, $U_2 \in [3, 5]$, $V_2 \in [2, 3]$, $U_3 \in [0, 6]$, $V_3 \in [2, 5]$, ALL_INCOMPARABLE($\langle\langle U_1, V_1\rangle, \langle U_2, V_2\rangle, \langle U_3, V_3\rangle\rangle$).

Typical
$|\text{VECTOR}| > 1$
$|\text{VECTORS}| > 1$
$|\text{VECTORS}| > |\text{VECTOR}|$

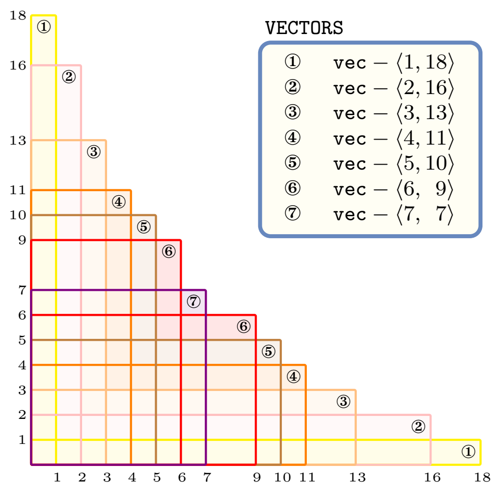


Figure 5.25: Illustrating the incomparability of vectors $\langle 1,18\rangle$, $\langle 2,16\rangle$, $\langle 3,13\rangle$, $\langle 4,11\rangle$, $\langle 5,10\rangle$, $\langle 6,9\rangle$, $\langle 7,7\rangle$: first to each vector we associate a rectangle whose sizes are the components of the vector; second no matter whether we rotate a rectangle from 90° or not, one rectangle can not be included in another rectangle.

Figure 5.26: All solutions corresponding to the non ground example of the ALL_INCOMPARABLE constraint of the **All solutions** slot

Symmetry Items of VECTORS are permutable.

Arg. properties Contractible wrt. VECTORS.

Usage Figure 5.27 illustrates the use of the ALL_INCOMPARABLE constraint in the context of a tiling problem.

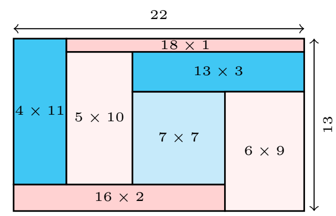
See also **implies:** LEX_ALLDIFFERENT.

part of system of constraints: INCOMPARABLE.

used in graph description: INCOMPARABLE.

Keywords **characteristic of a constraint:** vector.

$$\forall i \in [1,7] : \ell_i \in [1,22], h_i \in [1,13]$$
$$\forall i \in [1,7] : x_i \in [1,22], x_i + \ell_i \leq 23$$
$$\forall i \in [1,7] : y_i \in [1,13], y_i + h_i \leq 14$$
$$\textstyle\sum_{i \in [1,7]} \ell_i h_i = 22 \cdot 13$$



$$\text{ALL\_INC}\left(\left\langle\begin{array}{l} \texttt{vec} - \langle \ell_1, h_1\rangle, \\ \texttt{vec} - \langle \ell_2, h_2\rangle, \\ \texttt{vec} - \langle \ell_3, h_3\rangle, \\ \texttt{vec} - \langle \ell_4, h_4\rangle, \\ \texttt{vec} - \langle \ell_5, h_5\rangle, \\ \texttt{vec} - \langle \ell_6, h_6\rangle, \\ \texttt{vec} - \langle \ell_7, h_7\rangle \end{array}\right\rangle\right) \quad \text{DIFFN}\left(\left\langle\begin{array}{l} \texttt{orth} - \langle \texttt{ori} - x_1\ \texttt{siz} - \ell_1, \texttt{ori} - y_1\ \texttt{siz} - h_1\rangle, \\ \texttt{orth} - \langle \texttt{ori} - x_2\ \texttt{siz} - \ell_2, \texttt{ori} - y_2\ \texttt{siz} - h_2\rangle, \\ \texttt{orth} - \langle \texttt{ori} - x_3\ \texttt{siz} - \ell_3, \texttt{ori} - y_3\ \texttt{siz} - h_3\rangle, \\ \texttt{orth} - \langle \texttt{ori} - x_4\ \texttt{siz} - \ell_4, \texttt{ori} - y_4\ \texttt{siz} - h_4\rangle, \\ \texttt{orth} - \langle \texttt{ori} - x_5\ \texttt{siz} - \ell_5, \texttt{ori} - y_5\ \texttt{siz} - h_5\rangle, \\ \texttt{orth} - \langle \texttt{ori} - x_6\ \texttt{siz} - \ell_6, \texttt{ori} - y_6\ \texttt{siz} - h_6\rangle, \\ \texttt{orth} - \langle \texttt{ori} - x_7\ \texttt{siz} - \ell_7, \texttt{ori} - y_7\ \texttt{siz} - h_7\rangle \end{array}\right\rangle\right)$$

Figure 5.27: Model and solution for tiling a rectangle of size 22×13 with 7 rectangles of incomparable sizes, where ALL_INC is a shortcut for ALL_INCOMPARABLE; the constraints $\sum_{i \in [1,7]} \ell_i h_i = 22 \cdot 13$, ALL_INCOMPARABLE, and DIFFN respectively express the tiling of the 22×13 big rectangle, the incomparability of the sizes of the 7 rectangles to place, and the non-overlap of the 7 rectangles.

constraint type: system of constraints, decomposition.

final graph structure: no loop, symmetric.

Cond. implications

- ALL_INCOMPARABLE(VECTORS)
  with $|\text{VECTOR}| = 2$
  implies K_DISJOINT(SETS : VECTORS).

- ALL_INCOMPARABLE(VECTORS)
  with $|\text{VECTOR}| = 2$
  implies TWIN(PAIRS : VECTORS).

Arc input(s) VECTORS

Arc generator $CLIQUE(\neq) \mapsto$ collection(vectors1, vectors2)

Arc arity 2

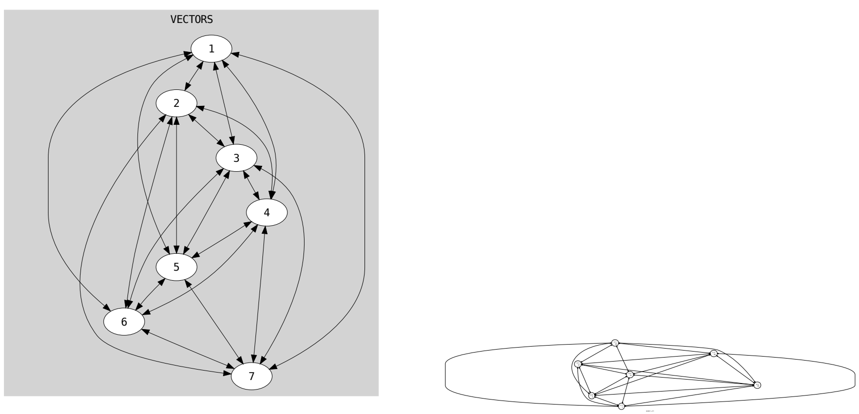
Arc constraint(s) INCOMPARABLE(vectors1.vec, vectors2.vec)

Graph property(ies) **NARC**$= |\texttt{VECTORS}| * |\texttt{VECTORS}| - |\texttt{VECTORS}|$

Graph class
- NO_LOOP
- SYMMETRIC

Graph model The **Arc constraint(s)** slot uses the INCOMPARABLE constraint defined in this catalogue.

Parts (A) and (B) of Figure 5.28 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The previous constraint holds since exactly $3 \cdot (3-1) = 6$ arc constraints hold.



(A) **(B)**

Figure 5.28: Initial and final graph of the ALL_INCOMPARABLE constraint

5.13 ALL_MIN_DIST

▶ ▷ ◀ DESCRIPTION LINKS GRAPH

Origin [354]

Constraint ALL_MIN_DIST(MINDIST, VARIABLES)

Synonyms MINIMUM_DISTANCE, INTER_DISTANCE.

Arguments

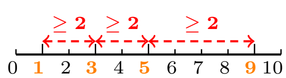
MINDIST : int
VARIABLES : collection(var−dvar)

Restrictions

$\texttt{MINDIST} > 0$
$|\texttt{VARIABLES}| < 2 \vee \texttt{MINDIST} < \texttt{range}(\texttt{VARIABLES.var})$
$\texttt{required}(\texttt{VARIABLES}, \texttt{var})$

Purpose Enforce for each pair $(\texttt{var}_i, \texttt{var}_j)$ of distinct variables of the collection VARIABLES that $|\texttt{var}_i - \texttt{var}_j| \geq \texttt{MINDIST}$.

Example $(2, \langle 5, 1, 9, 3\rangle)$



The ALL_MIN_DIST constraint holds since the following expressions $|5-1|$, $|5-9|$, $|5-3|$, $|1-9|$, $|1-3|$, $|9-3|$ are all greater than or equal to the first argument $\texttt{MINDIST} = 2$ of the ALL_MIN_DIST constraint.

All solutions Figure 5.29 gives all solutions to the following non ground instance of the ALL_MIN_DIST constraint: $\texttt{V}_1 \in [0,5]$, $\texttt{V}_2 \in [3,9]$, $\texttt{V}_3 \in [5,7]$, $\texttt{V}_4 \in [2,10]$, ALL_MIN_DIST$(\mathbf{3}, \langle \texttt{V}_1, \texttt{V}_2, \texttt{V}_3, \texttt{V}_4\rangle)$.

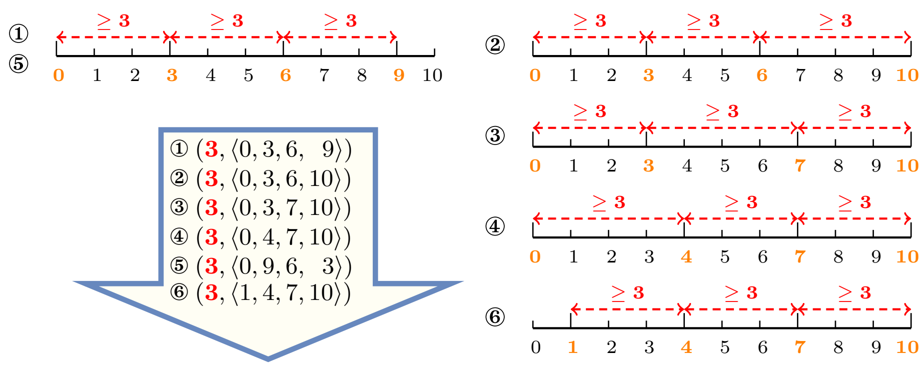


Figure 5.29: All solutions corresponding to the non ground example of the ALL_MIN_DIST constraint of the **All solutions** slot

Typical

$\texttt{MINDIST} > 1$
$|\texttt{VARIABLES}| > 1$

Symmetries

- MINDIST can be decreased to any value ≥ 1.
- Items of VARIABLES are permutable.
- Two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Contractible wrt. VARIABLES.

Usage

The ALL_MIN_DIST constraint was initially created for handling frequency allocation problems. In [12] it is used for scheduling tasks that all have the same fixed duration in the context of air traffic management in the terminal radar control area of airports.

Remark

The ALL_MIN_DIST constraint can be modelled as a set of tasks that should not overlap. For each variable var of the VARIABLES collection we create a task t where var and MINDIST respectively correspond to the origin and the duration of t.

Some solvers use in a pre-processing phase, while stating constraints of the form $|X_i - X_j| \geq D_{ij}$ (where X_i and X_j are domain variables and D_{ij} is a constant), an algorithm for automatically extracting large cliques [97] from such inequalities in order to state ALL_MIN_DIST constraints.

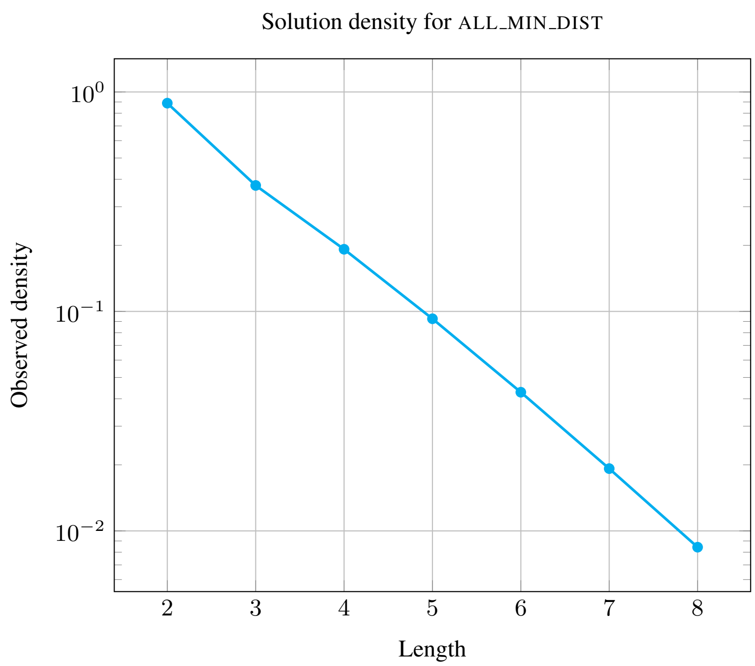
Algorithm

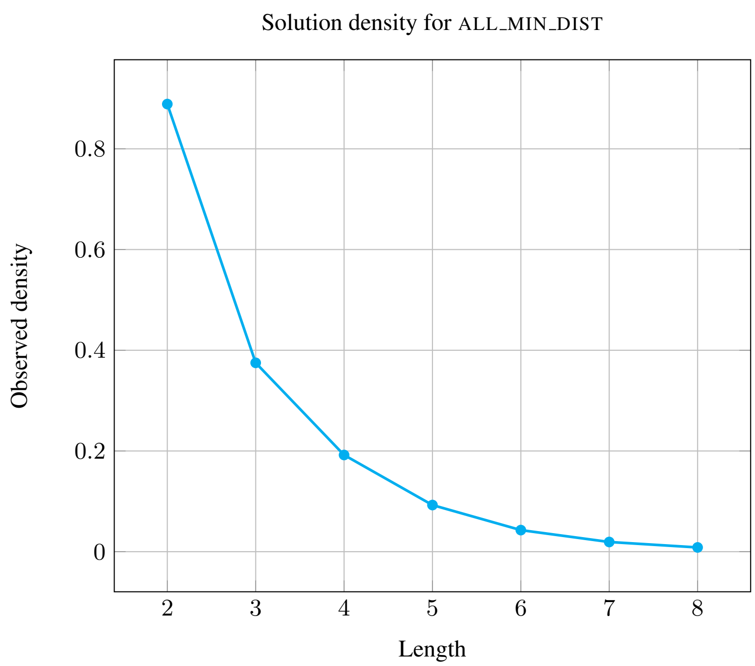
K. Artiouchine and P. Baptiste came up with a cubic time complexity algorithm achieving bound-consistency in [12, 13] based on the adaptation of a feasibility test algorithm from M.R. Garey *et al.* [196]. Later on, C.-G. Quimper *et al.*, proposed a quadratic algorithm achieving the same level of consistency in [343].

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	8	24	120	720	5040	40320	362880

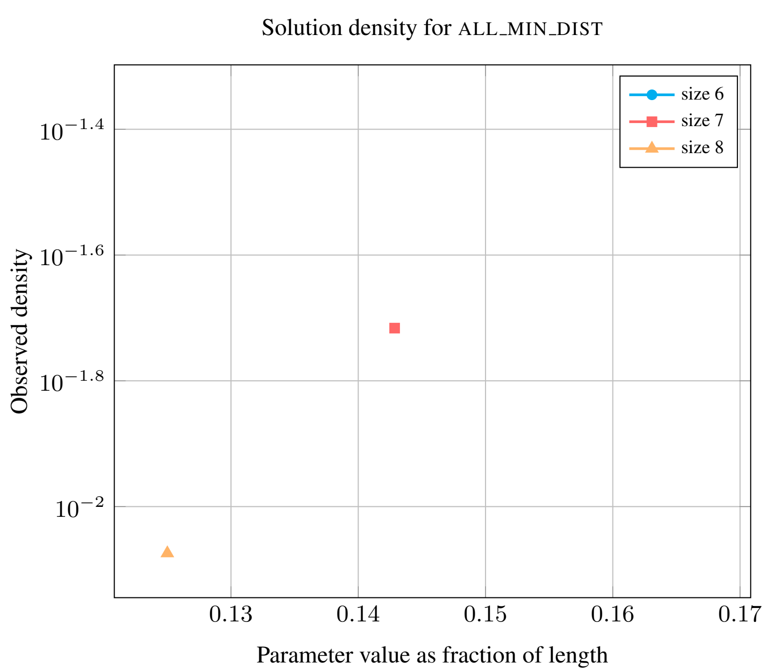
Number of solutions for ALL_MIN_DIST: domains $0..n$

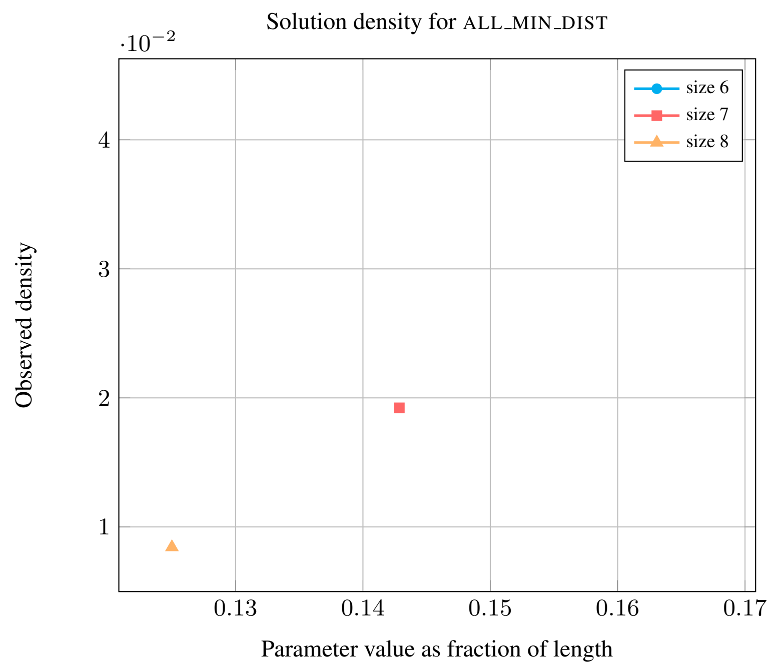

Solution density for ALL_MIN_DIST
Observed density
10^{0}
10^{-1}
10^{-2}
2 3 4 5 6 7 8
Length


Solution density for ALL_MIN_DIST
Observed density
0.8
0.6
0.4
0.2
0
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		8	24	120	720	5040	40320	362880
Parameter value	1	6	24	120	720	5040	40320	362880
	2	2	-	-	-	-	-	-

Solution count for ALL_MIN_DIST: domains $0..n$





See also

generalisation: DIFFN (`line segment`, *of same length, replaced by orthotope*), DISJUNCTIVE (`line segment`, *of same length, replaced by* `line segment`), MULTI_INTER_DISTANCE (`LIMIT` *parameter introduced to specify capacity* ≥ 1).

implies: ALLDIFFERENT_INTERVAL, SOFT_ALLDIFFERENT_VAR.

related: DISTANCE.

specialisation: ALLDIFFERENT (`line segment`, *of same length, replaced by* `variable`).

Keywords

application area: frequency allocation problem, air traffic management.

characteristic of a constraint: sort based reformulation.

constraint type: value constraint, decomposition, scheduling constraint.

filtering: bound-consistency.

final graph structure: acyclic.

problems: maximum clique.

Cond. implications

ALL_MIN_DIST(MINDIST, VARIABLES)
implies SOFT_ALL_EQUAL_MAX_VAR(N, VARIABLES)
when $\mathtt{N} \geq |\mathtt{VARIABLES}| - 1$.

Arc input(s) VARIABLES

Arc generator $CLIQUE(<) \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) abs(variables1.var − variables2.var) $\geq$ MINDIST

Graph property(ies) **NARC** = |VARIABLES| $*$ (|VARIABLES| − 1)/2

Graph class
- ACYCLIC
- NO_LOOP

Graph model We generate a *clique* with a minimum distance constraint between each pair of distinct vertices and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Parts (A) and (B) of Figure 5.30 respectively show the initial and final graph associated with the **Example** slot. The ALL_MIN_DIST constraint holds since all the arcs of the initial graph belong to the final graph: all the minimum distance constraints are satisfied.

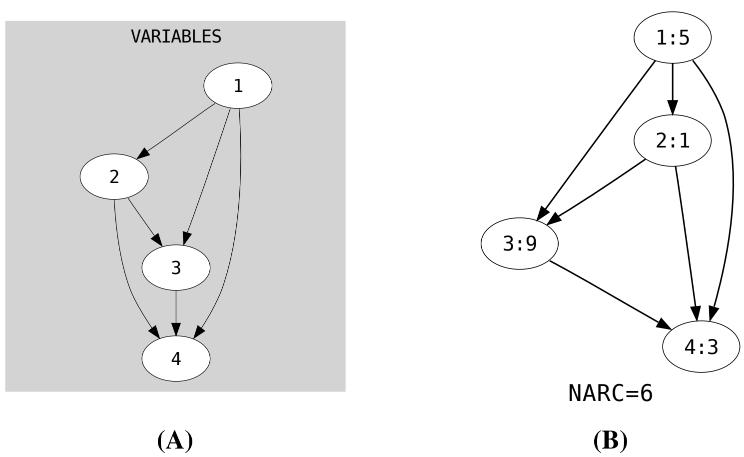


Figure 5.30: Initial and final graph of the ALL_MIN_DIST constraint

5.14 ALLDIFFERENT

► ▷ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [267]

Constraint ALLDIFFERENT(VARIABLES)

Synonyms ALLDIFF, ALLDISTINCT, DISTINCT, BOUND_ALLDIFFERENT, BOUND_ALLDIFF, BOUND_DISTINCT, REL.

Argument VARIABLES : collection(var−dvar)

Restriction required(VARIABLES, var)

Purpose Enforce all variables of the collection VARIABLES to take distinct values.

Example $(\langle 5, 1, 9, 3\rangle)$

The ALLDIFFERENT constraint holds since all the values 5, 1, 9 and 3 are distinct.

All solutions Figure 5.31 gives all solutions to the following non ground instance of the ALLDIFFERENT constraint: $\mathtt{V}_1 \in [2, 4]$, $\mathtt{V}_2 \in [2, 3]$, $\mathtt{V}_3 \in [1, 6]$, $\mathtt{V}_4 \in [2, 5]$, $\mathtt{V}_5 \in [2, 3]$, $\mathtt{V}_6 \in [1, 6]$, ALLDIFFERENT$(\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4, \mathtt{V}_5, \mathtt{V}_6\rangle)$.

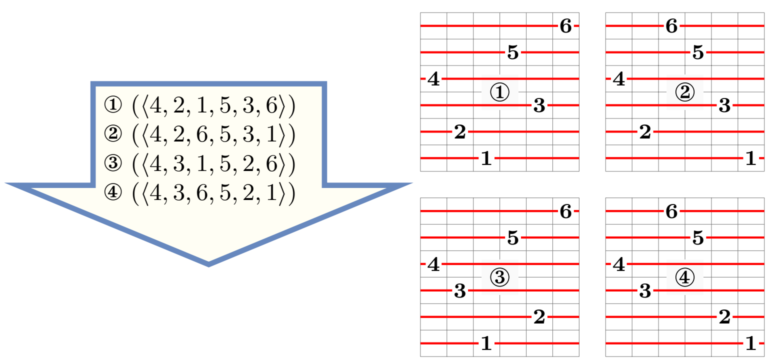


Figure 5.31: All solutions corresponding to the non ground example of the ALLDIFFERENT constraint of the **All solutions** slot

Typical $|\mathtt{VARIABLES}| > 2$

Symmetries

- Items of VARIABLES are permutable.
- Two distinct values of VARIABLES.var can be swapped; a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Contractible wrt. VARIABLES.

Usage The ALLDIFFERENT constraint occurs in most practical problems directly or indirectly. A classical example is the n-queens chess puzzle problem: Place n queens on an n by n chessboard in such a way that no queen attacks another. Two queens attack each other if they are located on the same column, on the same row, or on the same diagonal. This can be modelled as the conjunction of three ALLDIFFERENT constraints. We associate to column i of the chessboard a domain variable X_i that gives the row number where the corresponding queen is located. The three ALLDIFFERENT constraints are:

- alldifferent$(X_1, X_2 + 1, \ldots, X_n + n - 1)$ for the descending diagonals,
- alldifferent$(X_1, X_2, \ldots, X_n)$ for the rows,
- alldifferent$(X_1 + n - 1, X_2 + n - 2, \ldots, X_n)$ for the ascending diagonals.

They are respectively depicted by parts (A), (C) and (D) of Figure 5.32. Figure 5.33 makes explicit the link between the two families of diagonals and the corresponding ALLDIFFERENT constraints. Note that this model matches the checker introduced by Gauss to test whether a permutation of row numbers is a solution or not to the 8 queens problem: first add the numbers 1 up to 8 to the permutation and check that the resulting numbers are distinct, second add the numbers 8 down to 1 and perform the same check [451, pages 165–166].

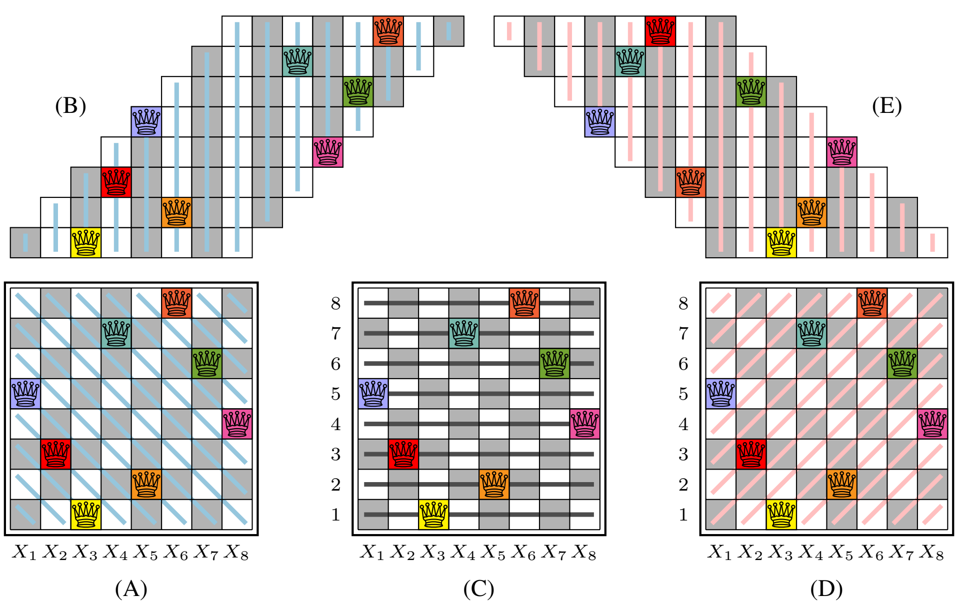


Figure 5.32: Descending diagonals (A-B), rows (C) and ascending diagonals (D-E)

A second example taken from [15], where the bipartite graph associated with the ALLDIFFERENT constraint is convex, is a *ski assignment problem*: "a set of skiers have each specified the smallest and largest ski sizes they will accept from a given set of ski sizes". The task is to find a ski size for each skier.

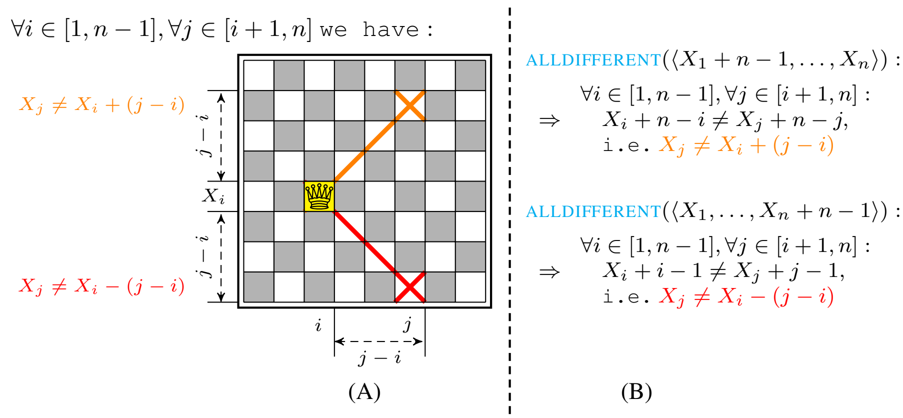


Figure 5.33: (A) For every pair of columns i, j ($i < j$), given the position X_i of the queen on column i, we respectively have from the ascending and descending diagonals that $X_j \neq X_i + (j-i)$ and $X_j \neq X_i - (j-i)$ (B) Equivalence of the two ALLDIFFERENT constraints respectively associated with the ascending and descending diagonals with the two families of disequalities (i.e., the orange and the red one) depicted in Part (A)

Examples such as Costas arrays and Golomb rulers involve one or several ALLDIFFERENT constraints on *differences* of variables.

Quite often, the ALLDIFFERENT constraint is also used in conjunction with several ELEMENT constraints, especially in the context of assignment problems [226, pages 372–374], or with several precedence constraints, especially in the context of symmetry breaking or scheduling problems [82]. How to handle an ALLDIFFERENT constraint together with a linear inequality constraint, where the coefficients are assumed to be fixed to one, is presented in [40].

Other examples involving several ALLDIFFERENT constraints sharing some variables can be found in the **Usage** slot of the K_ALLDIFFERENT constraint.

Remark

Even if the ALLDIFFERENT constraint did not have this form, it was specified in ALICE [266, 267] by asking for an injective correspondence f between variables and values: $x \neq y \Rightarrow f(x) \neq f(y)$. From an algorithmic point of view, the algorithm for computing the cardinality of the maximum matching of a bipartite graph was not used in ALICE for checking the feasibility of the ALLDIFFERENT constraint, even though the algorithm was already known in 1976. This is because the goal of ALICE was to show that a general system could be as efficient as dedicated algorithms. For this reason the concluding part of [266] explicitly mentions that specialised algorithms should be discarded. On the one hand, many people, especially from the OR community, have complained about such a radical statement [375, page 28]. On the other hand, the motivation of such a statement stands from the fact that a truly intelligent system should not rely on black-box algorithms, but should rather be able to reconstruct them from some kind of first principles. How to achieve this is still an open question.

Some solvers use, in a pre-processing phase before stating all constraints, an algorithm for *automatically extracting large cliques* [97, 162] from a set of binary disequalities in order to replace them by ALLDIFFERENT constraints.

W.-J. van Hoeve provides a survey about the ALLDIFFERENT constraint in [432].

For possible relaxation of the ALLDIFFERENT constraints see the ALLDIFFERENT_EXCEPT_0, the K_ALLDIFFERENT (i.e., SOME_DIFFERENT), the SOFT_ALLDIFFERENT_CTR, the SOFT_ALLDIFFERENT_VAR and the WEIGHTED_PARTIAL_ALLDIFF constraints, and Figure 2.4 of Section 2.1.5.

Within the context of linear programming, relaxations of the ALLDIFFERENT constraint are described in [452] and in [226, pages 362–367].

Within the context of *constraint-centered search heuristics*, G. Pesant and A. Zanarini [458] have proposed several estimators for evaluating the number of solutions of an ALLDIFFERENT constraint (since counting the total number of maximum matchings of the corresponding variable-value graph is #*P*-complete [424]). Faster, but less accurate estimators, based on upper bounds of the number of solutions were proposed three years later by the same authors [459].

Given n variables taking their values within the interval $[1, n]$, the total number of solutions to the corresponding ALLDIFFERENT constraint corresponds to the sequence A000142 of the On-Line Encyclopaedia of Integer Sequences [403].

Algorithm

The first complete filtering algorithm was independently found by M.-C. Costa [132] and J.-C. Régin [351]. This algorithm is based on a corollary of C. Berge that characterises the edges of a graph that belong to a maximum matching but not to all [63, page 120].[1] Similarly, Dulmage-Mendelsohn decomposition [157] was also used recently by [138, 139] to characterise such edges and prune the corresponding variables both for the ALLDIFFERENT constraint and for other constraints like ALLDIFFERENT_EXCEPT_0, CORRESPONDENCE, INVERSE, SAME, USED_BY, GLOBAL_CARDINALITY_LOW_UP, SOFT_ALLDIFFERENT_VAR, SOFT_SAME_VAR, SOFT_USED_BY_VAR.[2] Assuming that all variables have no holes in their domains, M. Leconte came up with a filtering algorithm [270] based on edge finding. A first bound-consistency algorithm was proposed by Bleuzen-Guernalec *et al.* [85]. Later on, two different approaches were used to design bound-consistency algorithms. Both approaches model the constraint as a bipartite graph. The first identifies Hall intervals in this graph [335, 277] and the second applies the same algorithm that is used to compute arc-consistency, but achieves a speedup by exploiting the simpler structure [205] of the graph [292]. Ian P. Gent *et al.* discuss in [200] implementations issues behind the complete filtering algorithm and in particular the computation of the strongly connected components of the residual graph (i.e., a graph constructed from a maximum variable-value matching and from the possible values of the variables of the ALLDIFFERENT constraint), which appears to be the main bottleneck in practice. Figures 2.1 and 2.2 of Section 2.1.3 illustrate the filtering of the ALLDIFFERENT constraint with respect to arc-consistency and bound-consistency. The leftmost part of Figure 3.29 illustrates a flow model for the ALLDIFFERENT constraint where there is a one-to-one correspondence between feasible flows in the flow model and solutions to the ALLDIFFERENT constraint.

From a worst case complexity point of view, assuming that n is the number of variables and m the sum of the domains sizes, we have the following complexity results:

[1] A similar result is in fact given in [320].

[2] Note that Dulmage-Mendelsohn decomposition [157] was also used in the context of geometric constraints [5, 90] for decomposing systems of equations.

- Complete filtering is achieved in $O(m\sqrt{n})$ by Régin's algorithm [351].
- Range consistency is done in $O(n^2)$ by Leconte's algorithm [270].
- Bound-consistency is performed in $O(n \log n)$ in [335, 292, 277]. If sort can be achieved in linear time, typically when the ALLDIFFERENT constraint encodes a permutation,[3] the worst case complexity of the algorithms described in [292, 277] goes down to $O(n)$.

Within the context of *explanations* [239], the explanation of the filtering algorithm that achieves arc-consistency for the ALLDIFFERENT constraint is described in [370, pages 60–61]. Given the residual graph (i.e., a graph constructed from a maximum variable-value matching and from the possible values of the variables of the ALLDIFFERENT constraint), the removal of an arc starting from a vertex belonging to a strongly connected component $\mathcal{C}_1$ to a distinct strongly connected component $\mathcal{C}_2$ is explained by all missing arcs starting from a descendant component of $\mathcal{C}_2$ and ending in an ancestor component of $\mathcal{C}_1$ (i.e., since the addition of any of these missing arcs would merge the strongly connected components $\mathcal{C}_1$ and $\mathcal{C}_2$). Let us illustrate this on a concrete example. For this purpose assume we have the following variables and the values that can potentially be assigned to each of them, $A \in \{1,2\}$, $B \in \{1,2\}$, $C \in \{2,3,4,6\}$, $D \in \{3,4\}$, $E \in \{5,6\}$, $F \in \{5,6\}$, $G \in \{6,7,8\}$, $H \in \{6,7,8\}$. Figure 5.34 represents the residual graph associated with the maximum matching corresponding to the assignment $A = 1$, $B = 2$, $C = 3$, $D = 4$, $E = 5$, $F = 6$, $G = 7$, $H = 8$. It has four strongly connected components containing respectively vertices $\{A, B, 1, 2\}$, $\{C, D, 3, 4\}$, $\{E, F, 5, 6\}$ and $\{G, H, 7, 8\}$. Arcs that are between strongly connected components correspond to values that can be removed:

- The removal of value 2 from variable C is explained by the absence of the arcs corresponding to the assignments $A = 3$, $A = 4$, $B = 3$ and $B = 4$ (since adding any of these missing arcs would merge the blue and the pink strongly connected components containing the vertices corresponding to value 2 and variable C).
- The removal of value 6 from variable C is explained by the absence of the arcs corresponding to the assignments $E = 3$, $E = 4$, $F = 3$ and $F = 4$. Again adding the corresponding arcs would merge the two strongly connected components containing the vertices corresponding to value 6 and variable C.
- The removal of value 6 from variable G is explained by the absence of the arcs corresponding to the assignments $E = 7$, $E = 8$, $F = 7$ and $F = 8$.
- The removal of value 6 from variable H is explained by the absence of the arcs corresponding to the assignments $E = 7$, $E = 8$, $F = 7$ and $F = 8$.

An additional example for illustrating the generation of explanations for the ALLDIFFERENT constraint when there are more values than variables is provided by Figure 2.3 of Section 2.1.4.

After applying bound-consistency the following property holds for all variables of an ALLDIFFERENT constraint. Given a Hall interval $[l, u]$, any variable V whose range $[\underline{V}, \overline{V}]$ intersects $[l, u]$ without being included in $[l, u]$ has its minimum value $\underline{V}$ (respectively maximum value $\overline{V}$) that is located before (respectively after) the Hall interval (i.e., $\underline{V} < l \leq u < \overline{V}$).

[3] In this context the total number of values that can be assigned to the variables of the ALLDIFFERENT constraint is equal to the number of variables. Under this assumption sorting the variables on their minimum or maximum values can be achieved in linear time.

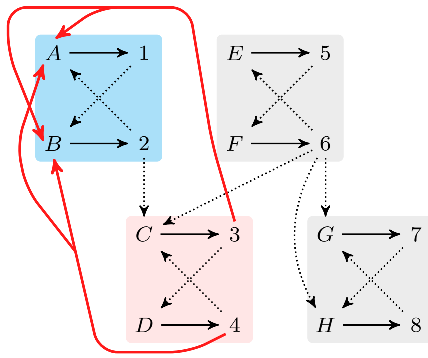


Figure 5.34: Strongly connected components of the residual graph illustrating the explanation of the removal of a value for the constraint ALLDIFFERENT$(\langle A, B, C, D, E, F, G, H\rangle)$, $A \in \{1,2\}$, $B \in \{1,2\}$, $C \in \{2,3,4,6\}$, $D \in \{3,4\}$, $E \in \{5,6\}$, $F \in \{5,6\}$, $G \in \{6,7,8\}$, $H \in \{6,7,8\}$: the explanation why value 2 is removed from variable C corresponds to all missing arcs whose addition would merge the blue and the pink strongly connected components (i.e., the missing arcs corresponding to the assignments $A = 3$, $A = 4$, $B = 3$ and $B = 4$ that are depicted by thick pink lines)

The ALLDIFFERENT constraint is entailed if and only if there is no value v that can be assigned two distinct variables of the VARIABLES collection (i.e., the intersection of the two sets of potential values of any pair of variables is empty).

Reformulation The ALLDIFFERENT constraint can be reformulated into a set of disequalities constraints. This model neither preserves bound-consistency nor arc-consistency:

- On the one hand a model, involving linear constraints, preserving bound-consistency was introduced in [78]. For each potential interval $[l, u]$ of consecutive values this model uses $|\texttt{VARIABLES}|$ 0-1 variables $B_{1,l,u}, B_{2,l,u}, \ldots, B_{|\texttt{VARIABLES}|,l,u}$ for modelling that each variable of the collection VARIABLES is assigned a value within interval $[l, u]$ (i.e., $\forall i \in [1, |\texttt{VARIABLES}|] : B_{i,l,u} \Leftrightarrow \texttt{VARIABLES}[i].\texttt{var} \in [l, u]$),[4] and an inequality constraint for enforcing the condition that the sum of the corresponding 0-1 variables is less than or equal to the size $u - l + 1$ of the corresponding interval (i.e. $B_{1,l,u} + B_{2,l,u} + \cdots + B_{|\texttt{VARIABLES}|,l,u} \leq u - l + 1$).
- On the other hand, it was shown in [81] that there is no polynomial sized decomposition that preserves arc-consistency.

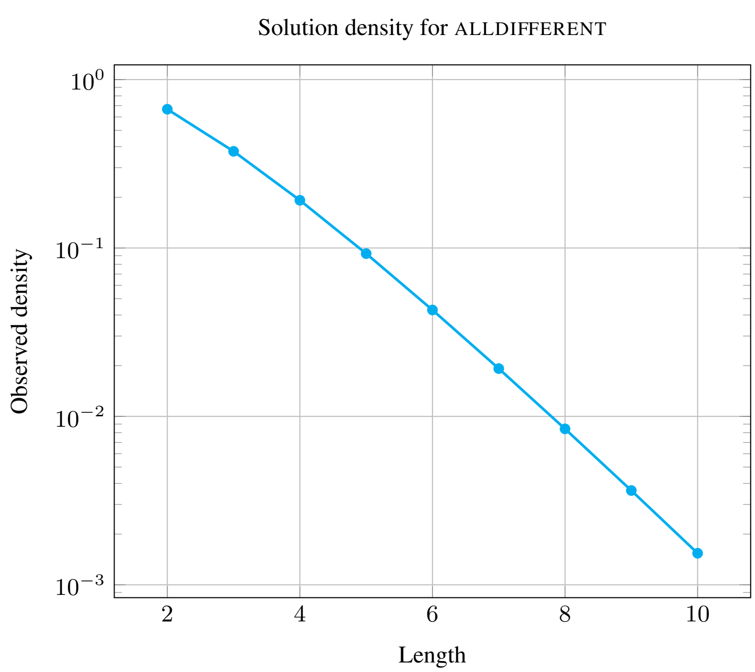
Finally the ALLDIFFERENT(VARIABLES) constraint can also be reformulated as the conjunction SORT(VARIABLES, SORTED_VARIABLES) $\wedge$ STRICTLY_INCREASING(SORTED_VARIABLES). Unlike the naive reformulation, i.e., a DISEQUALITY constraint between each pair of variables, the SORT-based reformulation is linear in space.

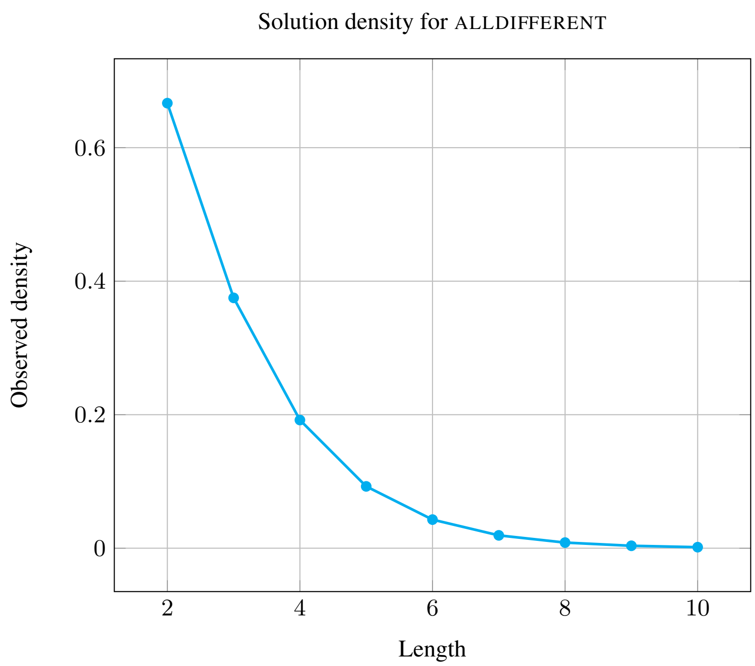
[4]How to encode the reified constraint $B_{i,l,u} \Leftrightarrow \texttt{VARIABLES}[i].\texttt{var} \in [l, u]$ with linear constraints is described in the **Reformulation slot** of the IN_INTERVAL_REIFIED constraint.

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	6	24	120	720	5040	40320	362880	3628800	39916800

Number of solutions for ALLDIFFERENT: domains $0..n$





Systems ALLDIFFERENT in **Choco**, LINEAR in **Gecode**, ALLDIFFERENT in **JaCoP**, ALLDIFF in **JaCoP**, ALLDISTINCT in **JaCoP**, ALL_DIFFERENT in **MiniZinc**, ALL_DIFFERENT in **SICStus**, ALL_DISTINCT in **SICStus**.

Used in ALLDIFFERENT_CONSECUTIVE_VALUES, CIRCUIT_CLUSTER, CORRESPONDENCE, CUMULATIVE_CONVEX, MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES, MAX_OCC_OF_SORTED_TUPLES_OF_VALUES, SIZE_MAX_SEQ_ALLDIFFERENT, SIZE_MAX_STARTING_SEQ_ALLDIFFERENT, SORT_PERMUTATION.

See also **common keyword:** CIRCUIT, CIRCUIT_CLUSTER, CYCLE, DERANGEMENT *(permutation)*, GOLOMB *(all different)*, PROPER_CIRCUIT *(permutation)*, SIZE_MAX_SEQ_ALLDIFFERENT, SIZE_MAX_STARTING_SEQ_ALLDIFFERENT *(all different,disequality)*, SYMMETRIC_ALLDIFFERENT *(permutation)*.

cost variant: MINIMUM_WEIGHT_ALLDIFFERENT, WEIGHTED_PARTIAL_ALLDIFF.

generalisation: ALL_MIN_DIST *(*`variable` *replaced by* `line segment`*, all of the same size)*, ALLDIFFERENT_BETWEEN_SETS *(*`variable` *replaced by* `set variable`*)*, ALLDIFFERENT_CST *(*`variable` *replaced by* `variable` + `constant`*)*, ALLDIFFERENT_INTERVAL *(*`variable` *replaced by* `variable/constant`*)*, ALLDIFFERENT_MODULO *(*`variable` *replaced by* `variable` mod `constant`*)*, ALLDIFFERENT_PARTITION *(*`variable` *replaced by* `variable` $\in$ `partition`*)*, DIFFN *(*`variable` *replaced by orthotope)*, DISJUNCTIVE *(*`variable` *replaced by* `task`*)*, GLOBAL_CARDINALITY *(control the number of occurrence of each value with a counter variable)*, GLOBAL_CARDINALITY_LOW_UP *(control the number of occurrence of each value with an interval)*, LEX_ALLDIFFERENT *(*`variable` *replaced by* `vector`*)*, NVALUE *(count number of distinct values)*.

implied by: ALLDIFFERENT_CONSECUTIVE_VALUES, CIRCUIT, CYCLE, STRICTLY_DECREASING, STRICTLY_INCREASING.

implies: ALLDIFFERENT_EXCEPT_0, MULTI_GLOBAL_CONTIGUITY, NOT_ALL_EQUAL.

negation: SOME_EQUAL.

part of system of constraints: NEQ.

shift of concept: ALLDIFFERENT_ON_INTERSECTION, ALLDIFFERENT_SAME_VALUE.

soft variant: ALLDIFFERENT_EXCEPT_0 *(value* 0 *can be used several times)*, OPEN_ALLDIFFERENT *(open constraint)*, SOFT_ALLDIFFERENT_CTR *(decomposition-based violation measure)*, SOFT_ALLDIFFERENT_VAR *(variable-based violation measure)*.

system of constraints: K_ALLDIFFERENT.

used in reformulation: IN_INTERVAL_REIFIED *(bound-consistency preserving reformulation)*, SORT, STRICTLY_INCREASING.

uses in its reformulation: CYCLE, ELEMENTS_ALLDIFFERENT, SORT_PERMUTATION.

Keywords

characteristic of a constraint: core, all different, disequality, sort based reformulation, automaton, automaton with array of counters.

combinatorial object: permutation.

constraint type: system of constraints, value constraint.

filtering: bipartite matching, bipartite matching in convex bipartite graphs, convex bipartite graph, flow, Hall interval, arc-consistency, bound-consistency, SAT, DFS-bottleneck, entailment.

final graph structure: one_succ.

modelling exercises: n-Amazons, zebra puzzle.

problems: maximum clique, graph colouring.

puzzles: n-Amazons, n-queens, Costas arrays, Euler knight, Golomb ruler, magic hexagon, magic square, zebra puzzle, Sudoku.

Cond. implications

- ALLDIFFERENT(VARIABLES)
  implies LEX_ALLDIFFERENT(VECTORS : VARIABLES).

- ALLDIFFERENT(VARIABLES)
  implies SOFT_ALLDIFFERENT_CTR(C, VARIABLES).

- ALLDIFFERENT(VARIABLES)
  implies BALANCE(BALANCE, VARIABLES)
  when $\text{BALANCE} = 0$.

- ALLDIFFERENT(VARIABLES)
  implies SOFT_ALL_EQUAL_MAX_VAR(N, VARIABLES)
  when $\text{N} < |\text{VARIABLES}|$.

- ALLDIFFERENT(VARIABLES)
  implies SOFT_ALL_EQUAL_MIN_VAR(N, VARIABLES)
  when $\text{N} > |\text{VARIABLES}|$.

- ALLDIFFERENT(VARIABLES)
  implies CHANGE(NCHANGE, VARIABLES, CTR)
  when NCHANGE $=$ |VARIABLES| $-$ 1
  and CTR $\in [\neq]$.

- ALLDIFFERENT(VARIABLES)
  implies CIRCULAR_CHANGE(NCHANGE, VARIABLES, CTR)
  when NCHANGE $=$ |VARIABLES|
  and CTR $\in [\neq]$.

- ALLDIFFERENT(VARIABLES)
  implies LONGEST_CHANGE(SIZE, VARIABLES, CTR)
  when SIZE $=$ |VARIABLES|
  and CTR $\in [\neq]$.

- ALLDIFFERENT(VARIABLES)
  with |VARIABLES| > 0
  implies LENGTH_FIRST_SEQUENCE(LEN, VARIABLES)
  when LEN $= 1$.

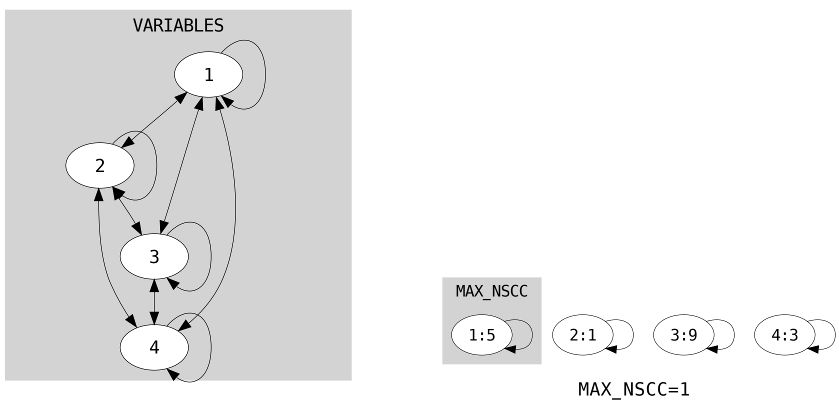
- ALLDIFFERENT(VARIABLES)
  with |VARIABLES| > 0
  implies LENGTH_LAST_SEQUENCE(LEN, VARIABLES)
  when LEN $= 1$.

- ALLDIFFERENT(VARIABLES)
  with |VARIABLES| > 0
  implies MIN_NVALUE(MIN, VARIABLES)
  when MIN $= 1$.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**MAX_NSCC** ≤ 1
Graph class	ONE_SUCC

Graph model We generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.

Parts (A) and (B) of Figure 5.35 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected component of the final graph. The ALLDIFFERENT holds since all the strongly connected components have at most one vertex: a value is used at most once.



(A) **(B)**

Figure 5.35: Initial and final graph of the ALLDIFFERENT constraint

Automaton Figure 5.36 depicts the automaton associated with the ALLDIFFERENT constraint. To each item of the collection VARIABLES corresponds a signature variable $\mathtt{S}_i$ that is equal to 1. The automaton counts the number of occurrences of each value and finally imposes that each value is taken at most one time.

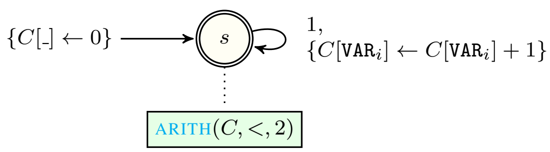


Figure 5.36: Automaton of the ALLDIFFERENT constraint

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint ALLDIFFERENT$(\langle 5, 1, 4, 8, 1\rangle)$ hold?

B. Does the constraint ALLDIFFERENT$(\langle 8, 2, 4, 3\rangle)$ hold?

C. Does the constraint ALLDIFFERENT$(\langle 0\rangle)$ hold?

[a]Hint: go back to the definition of ALLDIFFERENT.

EXERCISE 2 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\begin{cases} V_1 \in [3,5], \quad V_2 \in [3,4], \quad V_3 \in [2,7], \\ V_4 \in [3,4], \quad V_5 \in [2,7], \\ \text{ALLDIFFERENT}(\langle V_1, V_2, V_3, 6, V_4, V_5\rangle). \end{cases}$$

[a]Hint: identify infeasible values, enumerate solutions in lexicographic order.

EXERCISE 3 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\begin{cases} V_1 \in [4,6], \quad V_2 \in [1,3], \quad V_3 \in [1,4], \quad V_4 \in [1,2], \\ V_5 \in [4,7], \quad V_6 \in [4,6], \quad V_7 \in [1,2], \\ \text{ALLDIFFERENT}(\langle V_1, V_2, V_3, V_4, V_5, V_6, V_7\rangle). \end{cases}$$

[a]Hint: focus on variables with smallest domain first, identify Hall intervals for finding infeasible values, enumerate solutions in lexicographic order.

EXERCISE 4 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\begin{cases} V_1 \in \{1,3\}, & V_2 \in \{1,2,3,4\}, \\ V_3 \in \{1,5\}, & V_4 \in \{1,2,3,4,5,6\}, \\ V_5 \in \{3,5\}, & \\ \text{ALLDIFFERENT}(\langle V_1, V_2, V_3, V_4, V_5 \rangle). & \end{cases}$$

[a]Hint: focus on variables with smallest domain first, identify Hall sets for finding infeasible values, enumerate solutions in lexicographic order.

EXERCISE 5 (identifying infeasible values)[a]

Identify all variable-value pairs (V_i, val) $(1 \leq i \leq 6)$, such that the following constraint has no solution when variable V_i is assigned value val:

$$\begin{cases} V_1 \in \{1,2,4\}, & V_2 \in \{1,2,3,4,6\}, & V_3 \in \{1,2,6\}, \\ V_4 \in [1,6], & V_5 \in \{1,4,6\}, & V_6 \in \{2,4,6\}, \\ \text{ALLDIFFERENT}(\langle V_1, V_2, V_3, V_4, V_5, V_6 \rangle). & & \end{cases}$$

[a]Hint: focus on variables with smallest domain first, identify Hall sets for finding infeasible values.

EXERCISE 6 (identifying infeasible values and counting)[a]

A. Identify six variable-value pairs (V_i, val) $(1 \leq i \leq 9)$, such that the following conjunction of constraints has no solution when variable V_i is assigned value val.

$$\begin{cases} V_1 \in [1,7], V_2 \in [1,7], V_3 \in [1,7], \\ V_4 \in [1,4], V_5 \in [1,4], V_6 \in [1,4], \\ V_7 \in [3,6], V_8 \in [3,6], V_9 \in [3,6], \\ \text{ALLDIFFERENT}(\langle V_1, V_2, V_3, V_4, V_5, V_6 \rangle), \\ \text{ALLDIFFERENT}(\langle V_1, V_2, V_3, V_7, V_8, V_9 \rangle). \end{cases}$$

B. Describe concisely the structure of the set of solutions and derive the total number of solutions.

[a]Hint: group together variables that belong to the same set of constraints and reason on the number of distinct values assigned to such groups.

EXERCISE 7 (variable-based degree of violation)[a]

Compute the variable-based degree of violation[b] of the following constraints:

A. ALLDIFFERENT($\langle 2, 2, 2, 2\rangle$),

B. ALLDIFFERENT($\langle 3, 1, 5, 2, 7\rangle$),

C. ALLDIFFERENT($\langle 5, 5, 0, 5, 5, 0, 7\rangle$).

[a]Hint: focus on the groups of variables that are assigned the same value.

[b]Given a constraint for which all variables are fixed, the *variable-based degree of violation* is the minimum number of variables to assign differently in order to satisfy the constraint.

EXERCISE 8 (preventing conflict around the table[a b]**)**

Provide a concise and efficient model for the following problem. Given a set $\mathcal{M}$ of n men, a set $\mathcal{W}$ of n women, a set of pairs $\mathcal{C}$ where each pair $(m, w) \in \mathcal{C}$ represents a conflict between the man m ($m \in \mathcal{M}$) and the woman w ($w \in \mathcal{W}$), a rectangular table, the goal is to place on one side of the table all the n men and on the opposite side all the n women in such a way that two persons that are in conflict do not sit face to face.

[a]Adapted from the 2011 constraint programming exam at Polytechnique, C. Dürr.

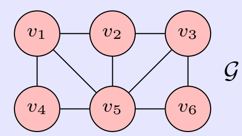
[b]Hint: break some symmetry of the problem.

EXERCISE 9 (identifying equalities from a clique of disequalities[a])

[CONTEXT] Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, colour each vertex $v \in \mathcal{V}$ in such a way that (1) two vertices that are linked by an edge of the set of edges $\mathcal{E}$ are not assigned the same colour, and (2) no more than m distinct colours are used to colour all the vertices of $\mathcal{G}$. The goal of the exercise is to find out necessary conditions for this problem that go beyond the cardinality of a (maximum) clique.

A. [IDENTIFYING PAIRS OF VERTICES THAT SHOULD BE ASSIGNED THE SAME COLOUR]

(a) Given the vertices of the following graph $\mathcal{G}$ to colour with at most three distinct colours, explain why vertices v_1 and v_3 should be assigned the same colour.



(b) For graph $\mathcal{G}$ provide all pairs of vertices that should be assigned the same colour if no more than three distinct colours have to be used.

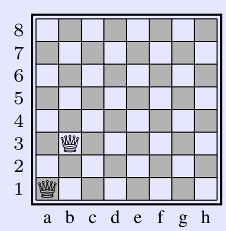
B. [GENERALISING THE NECESSARY CONDITION]

Given a clique of n vertices $\mathcal{C}$ of the graph $\mathcal{G}$, let $\mathcal{V}_\mathcal{C}$ denotes the set of vertices that do not belong to $\mathcal{C}$ and that are all connected to all vertices of $\mathcal{C}$. Assuming that one should use at most m distinct colours provide a necessary condition on the set $\mathcal{V}_\mathcal{C}$.

[a]Hint: restrict extra values wrt a clique of disequalities.

EXERCISE 10 (8-queens: unfeasibility of a partial solution [a])

Consider the 8-queens problem[b] where we start filling the chessboard in a systematic way: we place a first queen in a1 and a second queen in b3. Prove that it is not possible to extend this partial assignment to a complete solution.



[a]Hint: consider each of the 4 remaining positions on column c; extract information from the conjunction of the three ALLDIFFERENT constraints that allows the modelling of the n-queens problem.

[b]Place 8 queens on an 8 by 8 chessboard in such a way that no queen attacks another. Two queens attack each other if they are located on the same column, on the same row, or on the same diagonal.

SOLUTION TO EXERCISE 1

A. *No, since value* 1 *is used twice.*

B. *Yes, since all values* 8, 2, 4 *and* 3 *are distinct.*

C. *Yes, since value* 0 *is only used once.*

SOLUTION TO EXERCISE 2

the four solutions

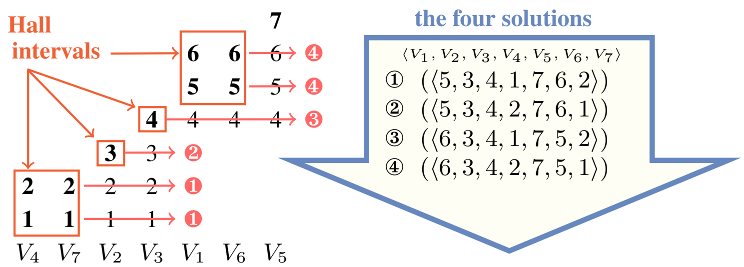
$\langle V_1, V_2, V_3, V_4, V_5 \rangle$

① $(\langle 5, 3, 2, 4, 7 \rangle)$
② $(\langle 5, 3, 7, 4, 2 \rangle)$
③ $(\langle 5, 4, 2, 3, 7 \rangle)$
④ $(\langle 5, 4, 7, 3, 2 \rangle)$

Values 3 *and* 4 *have to be assigned to the two variables* V_2 *and* V_4*. Consequently,* V_1, V_3 *and* V_5 *are different from* 3 *and* 4.

Values 3, 4 *and* 5 *have to be assigned to* V_1, V_2 *and* V_4*. Value* 6 *is directly mentioned in the constraint. Consequently the two remaining variables* V_3 *and* V_5 *can only be assigned values* 2 *and* 7.

SOLUTION TO EXERCISE 3



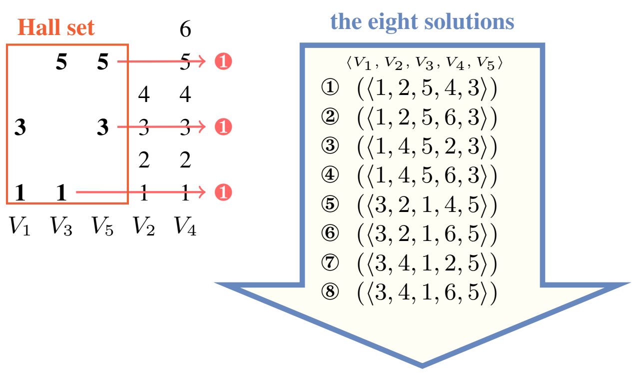
Let us reorder the variables by increasing minimum value, and by increasing maximum value in case of tie, for example, V_4, V_7, V_2, V_3, V_1, V_6, V_5.

- ❶ *Since values 1 and 2 have to be assigned to V_4 and V_7 (interval $[1, 2]$ is a Hall interval[a]), they cannot be assigned to the other variables and consequently V_2 is fixed to 3.*
- ❷ *Since V_2 is fixed to 3, V_3 is fixed to 4.*
- ❸ *Since V_3 is fixed to 4, V_1 and V_6 can only be assigned values 5 or 6 (interval $[5, 6]$ is a Hall interval).*
- ❹ *Since values 5 and 6 cannot be assigned to V_5, V_5 is fixed to 7.*

---

[a]Given a set of domain variables, a *Hall interval* is an interval of values $[\ell, u]$ such that there are $u - \ell + 1$ variables whose domains are contained in $[\ell, u]$.

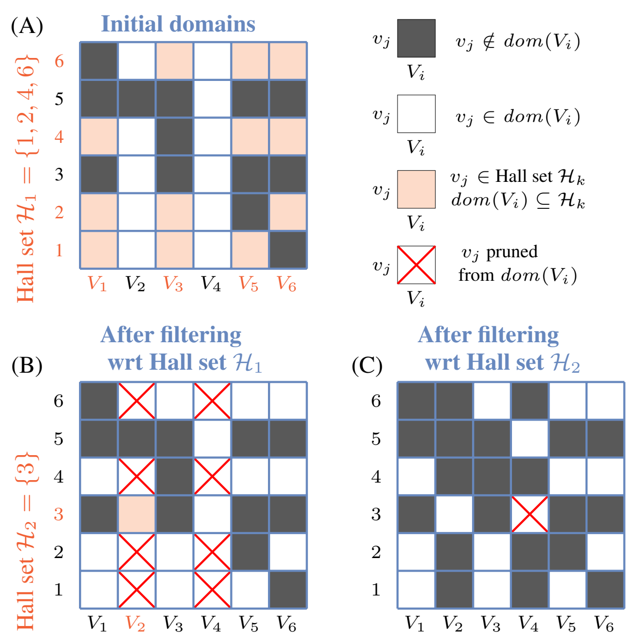
SOLUTION TO EXERCISE 4



Let us reorder the variables by increasing domain size, increasing minimum, and increasing maximum in case of tie, i.e., V_1, V_3, V_5, V_2, V_4. *Since values* 1, 3 *and* 5 *have to be assigned to* V_1, V_3 *and* V_5 ($\{1,3,5\}$ *is a Hall set*[a]), *they cannot be assigned to the other variables and consequently values* 1, 3 *and* 5 *are removed from* V_2 *and* V_4 *(see* ❶*).*

[a]Given a set of domain variables, a *Hall set* is a set of values $\mathcal{H}$ such that there are $|\mathcal{H}|$ variables whose domains are contained in $\mathcal{H}$.

SOLUTION TO EXERCISE 5



1. *In part* (A) *we first identify the Hall set*[a] $\mathcal{H}_1 = \{1, 2, 4, 6\}$ *which contains the domains of variables* V_1, V_3, V_5 *and* V_6.
2. *In part* (B) *we remove values* 1, 2, 4 *and* 6 *from those variables for which the domain is not included within the Hall set* $\mathcal{H}_1$, *namely* V_2 *and* V_4, *see* $\times$.
3. *After the previous filtering we identify in part* (B) *a new Hall set* $\mathcal{H}_2 = \{3\}$ *which contains the domain of* V_2.
4. *Finally in part* (C) *we remove value* 3 *from those variables for which the domain is not included within the Hall set* $\mathcal{H}_2$, *namely* V_4, *see* $\times$.

[a] Given a set of domain variables, a *Hall set* is a set of values $\mathcal{H}$ such that there are $|\mathcal{H}|$ variables whose domains are contained in $\mathcal{H}$.

SOLUTION TO EXERCISE 6

A. **(i)** *The cardinality of the union of the domains of* $V_1, V_2, \ldots, V_9$ *is equal to* 7. *Since* V_1, V_2 *and* V_3 *will be assigned* 3 *distinct values, the remaining variables* $V_4, V_5, \ldots, V_9$ *should not be assigned more than* $7 - 3 = 4$ *distinct values.*

(ii) $V_4, V_5, \ldots, V_9$ *can be partitioned in two sets* $\{V_4, V_5, V_6\}$ *and* $\{V_7, V_8, V_9\}$ *which respectively correspond to the variables that only belong to the first and to the second* ALLDIFFERENT. *The first set will be assigned distinct values in interval* $[1, 4]$, *while the second set will be assigned distinct values in interval* $[3, 6]$.

(iii) *Since* $V_4, V_5, \ldots, V_9$ *should not be assigned more than* 4 *distinct values, the two values* 3 *and* 4 *that belong both to* $[1, 4]$ *and* $[3, 6]$ *should be both assigned to* $\{V_4, V_5, V_6\}$ *and to* $\{V_7, V_8, V_9\}$. *Consequently values* 3 *and* 4 *cannot be assigned to variables* V_1, V_2 *and* V_3.

B. As illustrated by the next figure, we have four families of solutions ①, ②, ③ and ④ where the three sets of variables $\{V_1, V_2, V_3\}$, $\{V_4, V_5, V_6\}$ and $\{V_7, V_8, V_9\}$ are assigned values from three distinct set of values. This leads to a total number of solutions of $4 \cdot 3! \cdot 3! \cdot 3! = 864$.

①	②	③	④
$\{V_1, V_2, V_3\}$	$\{V_1, V_2, V_3\}$	$\{V_1, V_2, V_3\}$	$\{V_1, V_2, V_3\}$
$\{1, 5, \mathbf{7}\}$	$\{1, 6, \mathbf{7}\}$	$\{2, 5, \mathbf{7}\}$	$\{2, 6, \mathbf{7}\}$
$\{V_4, V_5, V_6\}$	$\{V_4, V_5, V_6\}$	$\{V_4, V_5, V_6\}$	$\{V_4, V_5, V_6\}$
$\{2, \mathbf{3}, \mathbf{4}\}$	$\{2, \mathbf{3}, \mathbf{4}\}$	$\{1, \mathbf{3}, \mathbf{4}\}$	$\{1, \mathbf{3}, \mathbf{4}\}$
$\{V_7, V_8, V_9\}$	$\{V_7, V_8, V_9\}$	$\{V_7, V_8, V_9\}$	$\{V_7, V_8, V_9\}$
$\{\mathbf{3}, \mathbf{4}, 6\}$	$\{\mathbf{3}, \mathbf{4}, 5\}$	$\{\mathbf{3}, \mathbf{4}, 6\}$	$\{\mathbf{3}, \mathbf{4}, 5\}$

SOLUTION TO EXERCISE 7

A. *The degree of violation is equal to* 3 *since at least three occurrences of value* 2 *(e.g. the three in red) out of the four occurrences of value* 2 *need to be assigned differently (e.g.,* 3*,* 4*,* 5 *in blue) in order to obtain a solution.*

$$\text{ALLDIFFERENT}(\langle 2, \overbrace{\mathbf{2}, \mathbf{2}, \mathbf{2}}^{\mathbf{3,4,5}}\rangle)$$

B. *The degree of violation is equal to* 0 *since the constraint holds, i.e. no value needs to be assigned differently.*

C. *The degree of violation is equal to* 4 *since at least three occurrences of value* 5 *and one occurrence of value* 0 *(e.g. the three* 5 *and the* 0 *in red) need to be assigned differently (e.g.,* 1*,* 3*,* 6*,* 4 *in blue) in order to obtain a solution.*

$$\text{ALLDIFFERENT}(\langle 5, \overbrace{\mathbf{5}, \mathbf{0}, \mathbf{5}, \mathbf{5}}^{\mathbf{1,3,6,4}}, 0, 7\rangle)$$

SOLUTION TO EXERCISE 8

Without loss of generality let us assume that the sets $\mathcal{M}$ *and* $\mathcal{W}$ *are both equal to* $\{1, 2, \ldots, n\}$*. We associate to each woman* w *in* $\mathcal{W}$ *a variable* F_w *providing the man which sits in front of* w*.*[a]

① *To prevent any conflict, the initial domain of each variable* F_w $(w \in \mathcal{W})$ *is set to all the men of* $\mathcal{M}$ *that are not in conflict with woman* w*, i.e. the men* $m \in \mathcal{M}$ *such that* $(m, w) \notin \mathcal{C}$*.*

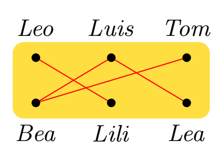
② *To enforce the fact that each woman can only sit in front of a single man we enforce an* ALLDIFFERENT$(\langle F_1, F_2, \ldots, F_n\rangle)$ *constraint.*

[a] Note that this model does not introduce variables for the men, i.e. each man is a value and each woman a variable. The model also eliminates some symmetry, i.e., it does not care where a woman sits.

AN INSTANCE

$\mathcal{W} = \{Bea, Lea, Lili\}$
$\mathcal{M} = \{Leo, Luis, Tom\}$
$\mathcal{C} = \{(Luis, Bea),$
$(Tom, Bea),$
$(Luis, Lea),$
$(Leo, Lili)\}$

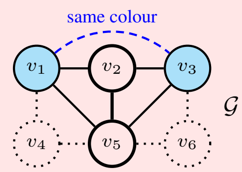
A SOLUTION
(*red edges correspond to conflicts*)



SOLUTION TO EXERCISE 9

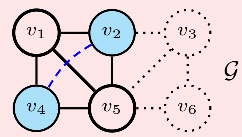
A. [*IDENTIFYING PAIRS OF VERTICES THAT SHOULD BE ASSIGNED THE SAME COLOUR*]

(a) *Since we have to use at most* 3 *distinct colours, since* v_2 *and* v_5 *use two distinct colours, and since* v_1 *and* v_3 *are both linked to* v_2 *and* v_5*, we infer that* v_1 *and* v_3 *both have to use the third and last available colour.*

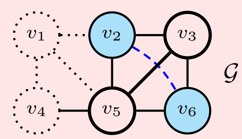


(b) *For a similar reason:*

- v_2 *and* v_4 *use the same colour since they are both linked to* v_1 *and* v_5.

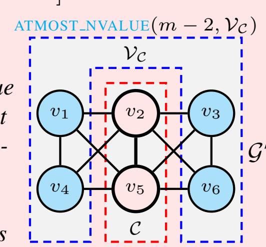


- v_2 *and* v_6 *use the same colour since they are both linked to* v_3 *and* v_5.



B. [*GENERALISING THE NECESSARY CONDITION*]

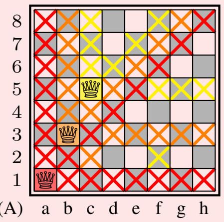
Since we should use at most m *distinct colours and since all vertices of the set* $\mathcal{V}_\mathcal{C}$ *are linked to all vertices of the clique* $\mathcal{C}$ *of* n *vertices, the set* $\mathcal{V}_\mathcal{C}$ *should use at most* $m - n$ *distinct colours. In the previous setting we had* $m = 3$ *and* $n = 2$*, i.e., one unique colour for all elements of* $\mathcal{V}_\mathcal{C}$*. The figure on the right illustrates the constraint generated wrt the clique* $\mathcal{C} = \{v_2, v_5\}$.



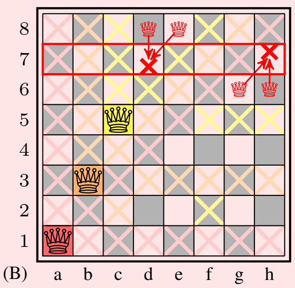
SOLUTION TO EXERCISE 10

We do case reasoning depending on where we place the queen on the third column. After placing the third queen we mark all cells that are located on the same column, row, or diagonal of one of the three already placed queens. Then we focus on the rows or columns for which no more than three consecutive cells are still empty since it allows the conjunction of constraints to prune the next row or column.

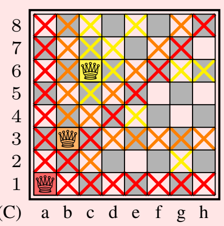
- [*PLACING A QUEEN ON* c5]
  After marking all cells that are located on a same column, row, or diagonal than a1, b3, *and* c5 *we get the chessboard shown on the right.*


(A)

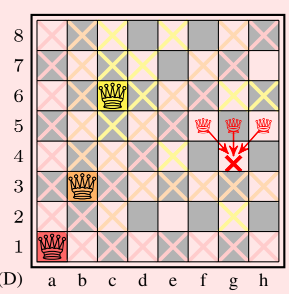
  Then we focus on row 8, *which contains only two consecutive free cells. Since the eighth row must contain one queen this queen will be located at position* d8 *or* e8. *In both cases the cell* d7 *will be forbidden. By performing the same reasoning on the sixth row we also find out that the cell* h7 *is forbidden. As a result no queen can be placed on row* 7.

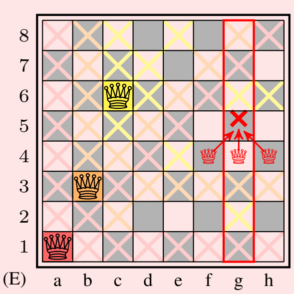

(B)

- [*MOVING THE THIRD QUEEN FROM* c5 *TO* c6]
  After marking all cells that are located on a same column, row, or diagonal than a1, b3, *and* c6 *we get the chessboard shown on the right.*


(C)

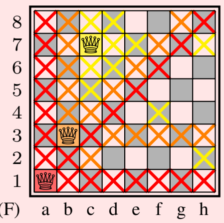
  We focus on row 5, *which contains only three consecutive free cells, see* (D). *Since row* 5 *must contain a queen, this queen will be located at position* f5, g5, *or* h5. *In all three cases the cell* g4 *will be forbidden. Similarly by considering the fourth row, see* (E), *we find out that no queen can be placed on* g5. *As a result no queen can be placed on column* g.

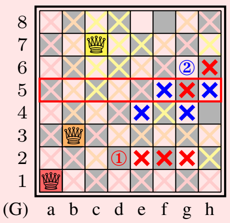

(D)


(E)

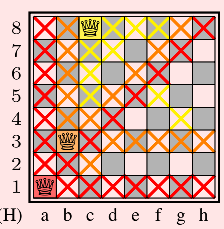
SOLUTION TO EXERCISE 10 (continued)

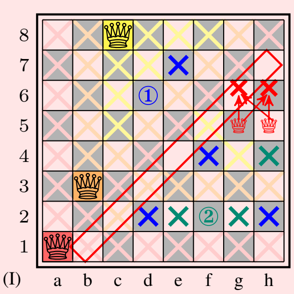
- [*MOVING THE THIRD QUEEN FROM* c6 *TO* c7]
  After marking all cells that are located on a same column, row, or diagonal than a1, b3, *and* c7 *we get the chessboard shown in* (F).
  Since on column d *only position* d2 *is free a queen* ① *is placed on* d2 *and we mark with small red crosses the new forbidden positions, see (G). Then on row* 6 *only position* g6 *is free. Consequently a queen* ② *is placed on* g6, *which forbids all remaining free positions on row* 5.


(F)


(G)

- [*MOVING THE THIRD QUEEN FROM* c7 *TO* c8]
  After marking all cells that are located on a same column, row, or diagonal than a1, b3, *and* c8 *we get the chessboard shown in* (H).
  Then we focus on row 5, *see* (I), *which contains only two consecutive free cells. Since the fifth row must contain one queen this queen will be located at position* g5 *or* h5. *In both cases the cell* g6 *is forbidden.*
  By performing the same reasoning we also find out that cell h6 *is forbidden. Since on row* 6 *only position* d6 *is free, a queen* ① *is placed on* d6 *and we mark with small blue crosses the new forbidden positions. Then on column* f *only position* f2 *is free. Consequently a queen* ② *is placed on* f2 *and we mark with small green crosses the new forbidden positions. Now row* 7 *and column* e *have only one free cell, namely* h7 *and* e4 *which are located on the same diagonal. Consequently, we cannot place the last two queens on* e4 *and* h7. *Hence the first two queens cannot be placed on* a1 *and* b3.


(H)


(I)

5.15 ALLDIFFERENT_BETWEEN_SETS

▶ ▷ ◀ DESCRIPTION LINKS GRAPH

Origin ILOG

Constraint ALLDIFFERENT_BETWEEN_SETS(VARIABLES)

Synonyms ALL_NULL_INTERSECT, ALLDIFF_BETWEEN_SETS, ALLDISTINCT_BETWEEN_SETS, ALLDIFF_ON_SETS, ALLDISTINCT_ON_SETS, ALLDIFFERENT_ON_SETS.

Argument VARIABLES : collection(var−svar)

Restriction required(VARIABLES, var)

Purpose Enforce all sets of the collection VARIABLES to be distinct.

Example $(\langle \texttt{var} - \{3,5\}, \texttt{var} - \emptyset, \texttt{var} - \{3\}, \texttt{var} - \{3,5,7\}\rangle)$

The ALLDIFFERENT_BETWEEN_SETS constraint holds since all the sets $\{3,5\}$, $\emptyset$, $\{3\}$ and $\{3,5,7\}$ are distinct.

Typical $|\texttt{VARIABLES}| > 2$

Symmetry Items of VARIABLES are permutable.

Arg. properties Contractible wrt. VARIABLES.

Usage This constraint was available in some configuration library offered by Ilog.

Algorithm A filtering algorithm for the ALLDIFFERENT_BETWEEN_SETS is proposed by C.-G. Quimper and T. Walsh in [346] and a longer version is available in [347] and in [348].

See also **common keyword:** LINK_SET_TO_BOOLEANS *(constraint involving set variables)*.

specialisation: ALLDIFFERENT *(set variable replaced by variable)*.

used in graph description: EQ_SET.

Keywords **characteristic of a constraint:** all different, disequality.

constraint arguments: constraint involving set variables.

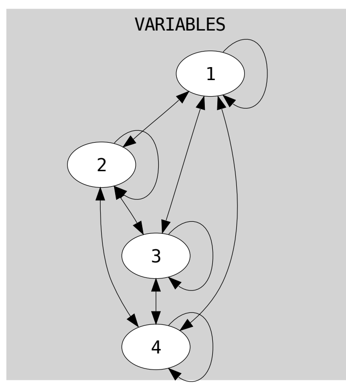
filtering: bipartite matching.

final graph structure: one_succ.

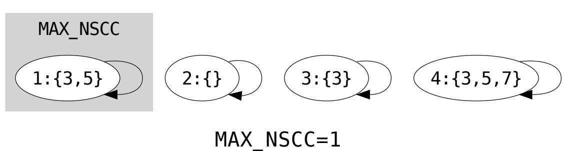
Arc input(s)	VARIABLES
Arc generator	*CLIQUE*↦collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	EQ_SET(variables1.var, variables2.var)
Graph property(ies)	**MAX_NSCC** ≤ 1
Graph class	ONE_SUCC

Graph model We generate a *clique* with binary *set equalities* constraints between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed 1.

Parts (A) and (B) of Figure 5.37 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph. The ALLDIFFERENT_BETWEEN_SETS holds since all the strongly connected components have at most one vertex.



(A)



(B)

Figure 5.37: Initial and final graph of the ALLDIFFERENT_BETWEEN_SETS constraint

5.16 ALLDIFFERENT_CONSECUTIVE_VALUES

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from ALLDIFFERENT.

Constraint ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions
required(VARIABLES, var)
ALLDIFFERENT(VARIABLES)

Purpose Enforce (1) all variables of the collection VARIABLES to take distinct values and (2) constraint the difference between the largest and the smallest values of the VARIABLES collection to be equal to the number of variables minus one (i.e., there is no holes at all within the used values).

Example $(\langle 5, 4, 3, 6\rangle)$

The ALLDIFFERENT_CONSECUTIVE_VALUES constraint holds since (1) all the values 5, 4, 3 and 6 are distinct and since (2) all values between value 3 and value 6 are actually used.

All solutions Figure 5.38 gives all solutions to the following non ground instance of the ALLDIFFERENT_CONSECUTIVE_VALUES constraint: $V_1 \in \{0, 1, 3, 4, 5, 6, 7, 8\}$, $V_2 \in [4, 5]$, $V_3 \in [3, 4]$, $V_4 \in [0, 7]$, $V_5 \in [3, 4]$, ALLDIFFERENT_CONSECUTIVE_VALUES$(\langle V_1, V_2, V_3, V_4, V_5\rangle)$.

① $(\langle \mathbf{1}, \mathbf{5}, 3, 2, 4\rangle)$
② $(\langle \mathbf{1}, \mathbf{5}, 4, 2, 3\rangle)$
③ $(\langle \mathbf{6}, 5, 3, \mathbf{2}, 4\rangle)$
④ $(\langle 6, 5, \mathbf{3}, \mathbf{7}, 4\rangle)$

⑤ $(\langle \mathbf{6}, 5, 4, \mathbf{2}, 3\rangle)$
⑥ $(\langle 6, 5, 4, \mathbf{7}, \mathbf{3}\rangle)$
⑦ $(\langle \mathbf{7}, 5, \mathbf{3}, 6, 4\rangle)$
⑧ $(\langle \mathbf{7}, 5, 4, 6, \mathbf{3}\rangle)$

Figure 5.38: All solutions corresponding to the non ground example of the ALLDIFFERENT_CONSECUTIVE_VALUES constraint of the **All solutions** slot, where the smallest and largest values are respectively coloured in orange and red

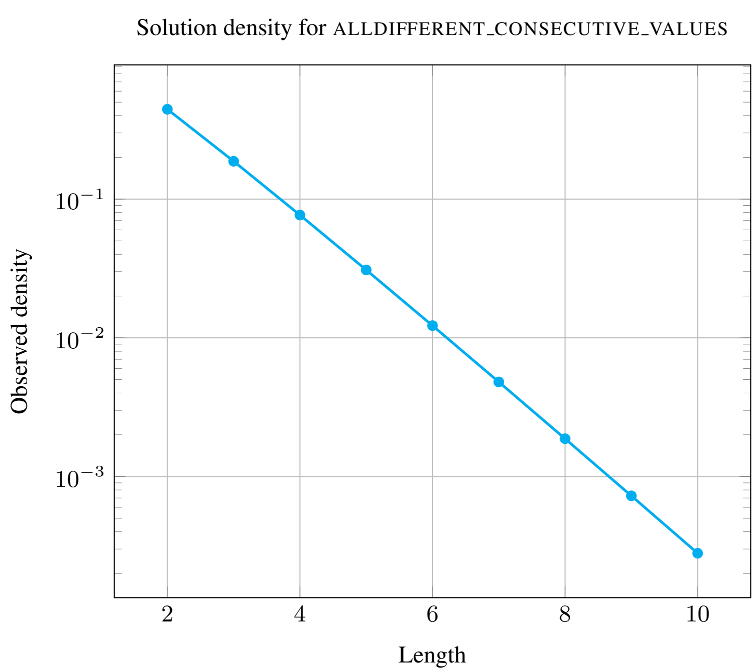
Typical $|\text{VARIABLES}| > 2$

Symmetries
- Items of VARIABLES are permutable.
- Two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

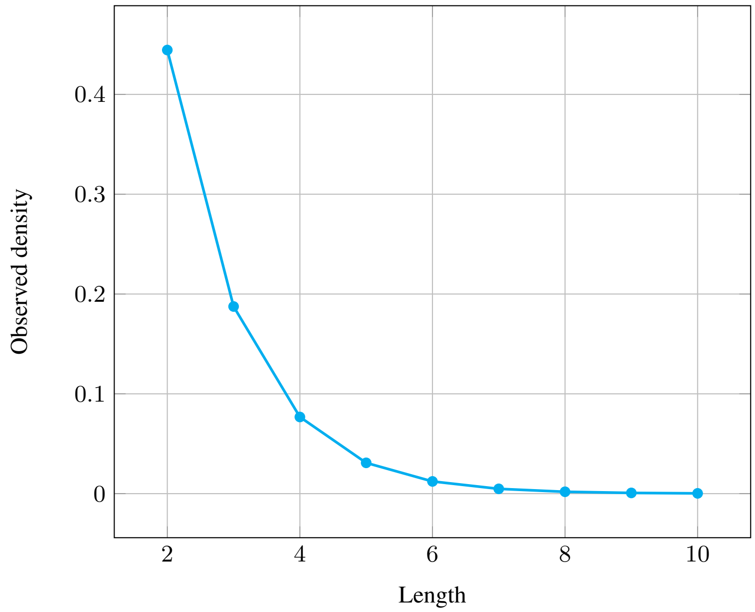
Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	4	12	48	240	1440	10080	80640	725760	7257600

Number of solutions for ALLDIFFERENT_CONSECUTIVE_VALUES: domains $0..n$



Solution density for ALLDIFFERENT_CONSECUTIVE_VALUES



See also

implied by: PERMUTATION.

implies: ALLDIFFERENT, CONSECUTIVE_VALUES.

Keywords

characteristic of a constraint: all different, disequality, sort based reformulation.

combinatorial object: permutation.

constraint type: value constraint.

Cond. implications

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  with minval(VARIABLES.var) ≤ 0
  and maxval(VARIABLES.var) ≥ 0
  implies AMONG_DIFF_0(NVAR, VARIABLES)
  when NVAR $=$ |VARIABLES| $- 1$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  with minval(VARIABLES.var) > 0
  implies AMONG_DIFF_0(NVAR, VARIABLES)
  when NVAR $=$ |VARIABLES|.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  with maxval(VARIABLES.var) < 0
  implies AMONG_DIFF_0(NVAR, VARIABLES)
  when NVAR $=$ |VARIABLES|.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  implies BALANCE(BALANCE, VARIABLES)
  when BALANCE $= 0$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  with $|\texttt{VARIABLES}| > 0$
  implies LENGTH_FIRST_SEQUENCE(LEN, VARIABLES)
  when $\texttt{LEN} = 1$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  with $|\texttt{VARIABLES}| > 0$
  implies LENGTH_LAST_SEQUENCE(LEN, VARIABLES)
  when $\texttt{LEN} = 1$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  implies MAX_N(MAX, RANK, VARIABLES)
  when $\texttt{MAX} = \texttt{maxval}(\texttt{VARIABLES.var}) - \texttt{RANK}$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  implies MIN_N(MIN, RANK, VARIABLES)
  when $\texttt{MIN} = \texttt{minval}(\texttt{VARIABLES.var}) + \texttt{RANK}$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  with $|\texttt{VARIABLES}| > 0$
  implies MIN_NVALUE(MIN, VARIABLES)
  when $\texttt{MIN} = 1$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  with $\texttt{minval}(\texttt{VARIABLES.var}) = 0$
  implies NINTERVAL(NVAL, VARIABLES, SIZE_INTERVAL)
  when $\texttt{NVAL} = (|\texttt{VARIABLES}| + \texttt{SIZE\_INTERVAL} - 1)/\texttt{SIZE\_INTERVAL}$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  implies RANGE_CTR(VARIABLES, CTR, VARIABLES)
  when $\texttt{CTR} \in [\leq]$
  and $\texttt{R} = |\texttt{VARIABLES}|$.

- ALLDIFFERENT_CONSECUTIVE_VALUES(VARIABLES)
  implies SOFT_ALLDIFFERENT_CTR(C, VARIABLES).

Arc input(s)	VARIABLES
Arc generator	$SELF \mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	TRUE
Graph property(ies)	**RANGE**(VARIABLES, var) $=$ $\|$VARIABLES$\|$ $-$ 1

5.17 ALLDIFFERENT_CST

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin CHIP

Constraint ALLDIFFERENT_CST(VARIABLES)

Synonyms ALLDIFF_CST, ALLDISTINCT_CST.

Argument VARIABLES : collection(var−dvar, cst−int)

Restriction required(VARIABLES, [var, cst])

Purpose For all pairs of items (VARIABLES$[i]$, VARIABLES$[j]$) ($i \neq j$) of the collection VARIABLES enforce VARIABLES$[i]$.var + VARIABLES$[i]$.cst $\neq$ VARIABLES$[j]$.var + VARIABLES$[j]$.cst.

Example

$$\left(\left\langle \begin{array}{ll} \texttt{var}-5 & \texttt{cst}-0, \\ \texttt{var}-1 & \texttt{cst}-1, \\ \texttt{var}-9 & \texttt{cst}-0, \\ \texttt{var}-3 & \texttt{cst}-4 \end{array} \right\rangle\right)$$

The ALLDIFFERENT_CST constraint holds since all the expressions $5+0=5$, $1+1=2$, $9+0=9$ and $3+4=7$ correspond to distinct values.

All solutions Figure 5.39 gives all solutions to the following non ground instance of the ALLDIFFERENT_CST constraint: $\texttt{V}_1 \in [0,2]$, $\texttt{V}_2 \in [4,5]$, $\texttt{V}_3 = 4$, $\texttt{V}_4 \in [0,1]$, ALLDIFFERENT_CST($\langle\langle \texttt{V}_1, 0\rangle, \langle \texttt{V}_2, 1\rangle, \langle \texttt{V}_3, 2\rangle, \langle \texttt{V}_4, 3\rangle\rangle$).

① $(\langle 0{+}0, 4{+}1, 4{+}2, 0{+}3\rangle)$
② $(\langle 0{+}0, 4{+}1, 4{+}2, 1{+}3\rangle)$
③ $(\langle 1{+}0, 4{+}1, 4{+}2, 0{+}3\rangle)$

④ $(\langle 1{+}0, 4{+}1, 4{+}2, 1{+}3\rangle)$
⑤ $(\langle 2{+}0, 4{+}1, 4{+}2, 0{+}3\rangle)$
⑥ $(\langle 2{+}0, 4{+}1, 4{+}2, 1{+}3\rangle)$

Figure 5.39: All solutions corresponding to the non ground example of the ALLDIFFERENT_CST constraint of the **All solutions** slot

Typical
|VARIABLES| > 2
range(VARIABLES.var) > 1
2∗range(VARIABLES.var) < 3 ∗ |VARIABLES|
range(VARIABLES.cst) > 1

Symmetries

- Items of VARIABLES are permutable.
- Attributes of VARIABLES are permutable w.r.t. permutation (var, cst) (*permutation not necessarily applied to all items*).
- One and the same constant can be added to the var attribute of all items of VARIABLES.
- One and the same constant can be added to the cst attribute of all items of VARIABLES.

Arg. properties

Contractible wrt. VARIABLES.

Usage

The ALLDIFFERENT_CST constraint was originally introduced in **CHIP** in order to express the n-queen problem with 3 global constraints (see the **Usage** slot of the ALLDIFFERENT constraint).

Algorithm

See the filtering algorithms of the ALLDIFFERENT constraint.

Systems

LINEAR in **Gecode**.

See also

implies (items to collection): LEX_ALLDIFFERENT.

specialisation: ALLDIFFERENT (variable + constant *replaced by* variable).

Keywords

characteristic of a constraint: all different, disequality, sort based reformulation.

constraint type: value constraint.

filtering: bipartite matching, bipartite matching in convex bipartite graphs, convex bipartite graph, arc-consistency.

final graph structure: one_succ.

modelling exercises: n-Amazons.

puzzles: n-Amazons, n-queens.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*↦collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var + variables1.cst = variables2.var + variables2.cst
Graph property(ies)	**MAX_NSCC** ≤ 1
Graph class	ONE_SUCC

Graph model We generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.

Parts (A) and (B) of Figure 5.40 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph. The ALLDIFFERENT_CST holds since all the strongly connected components have at most one vertex: a value is used at most once.

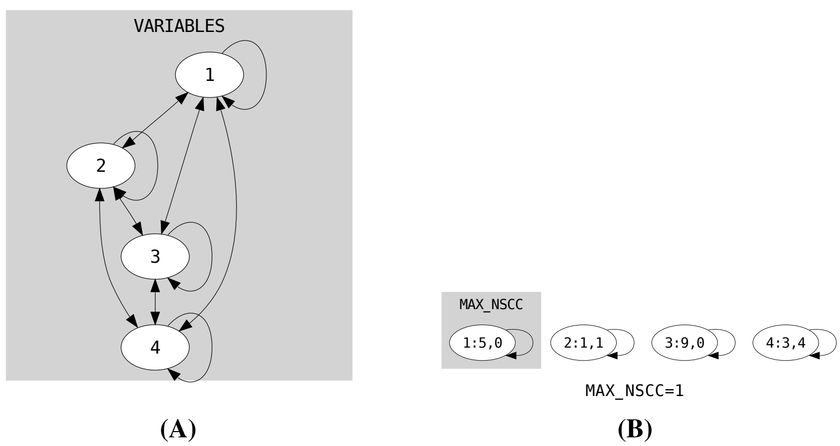


Figure 5.40: Initial and final graph of the ALLDIFFERENT_CST constraint

5.18 ALLDIFFERENT_EXCEPT_0

► ▷ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from ALLDIFFERENT.

Constraint ALLDIFFERENT_EXCEPT_0(VARIABLES)

Synonyms ALLDIFF_EXCEPT_0, ALLDISTINCT_EXCEPT_0.

Argument VARIABLES : collection(var−dvar)

Restriction required(VARIABLES, var)

Purpose Enforce all variables of the collection VARIABLES to take distinct values, except those variables that are assigned value 0.

Example $(\langle 5, 0, 1, 9, 0, 3\rangle)$

The ALLDIFFERENT_EXCEPT_0 constraint holds since all the values (that are different from 0) 5, 1, 9 and 3 are distinct.

All solutions Figure 5.41 gives all solutions to the following non ground instance of the ALLDIFFERENT_EXCEPT_0 constraint: $V_1 \in [0, 4]$, $V_2 \in [1, 2]$, $V_3 \in [1, 2]$, $V_4 \in [0, 1]$, ALLDIFFERENT_EXCEPT_0$(\langle V_1, V_2, V_3, V_4\rangle)$.

① $(\langle 0, 1, 2, 0\rangle)$
② $(\langle 0, 2, 1, 0\rangle)$
③ $(\langle 3, 1, 2, 0\rangle)$

④ $(\langle 3, 2, 1, 0\rangle)$
⑤ $(\langle 4, 1, 2, 0\rangle)$
⑥ $(\langle 4, 2, 1, 0\rangle)$

Figure 5.41: All solutions corresponding to the non ground example of the ALLDIFFERENT_EXCEPT_0 constraint of the **All solutions** slot

Typical
$|\texttt{VARIABLES}| > 2$
ATLEAST(2, VARIABLES, 0)
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES are permutable.
- Two distinct values of VARIABLES.var that are both different from 0 can be swapped; a value of VARIABLES.var that is different from 0 can be renamed to any unused value that is also different from 0.

Arg. properties

Contractible wrt. VARIABLES.

Usage

Quite often it appears that, for some modelling reason, you create a *joker value*. You do not want that normal constraints hold for variables that take this *joker value*. For this purpose we modify the binary arc constraint in order to discard the vertices for which the corresponding variables are assigned value 0. This will be effectively the case since all the corresponding arcs constraints will not hold.

Algorithm

An arc-consistency filtering algorithm for the ALLDIFFERENT_EXCEPT_0 constraint is described in [138]. The algorithm is based on the following ideas:

- First, one can map solutions to the ALLDIFFERENT_EXCEPT_0 constraint to var-perfect matchings[5] in a bipartite graph derived from the domain of the variables of the constraint in the following way: to each variable of the ALLDIFFERENT_EXCEPT_0 constraint corresponds a *variable* and a *joker* vertices, while to each potential value corresponds a *value* vertex; there is an edge between a variable vertex and a value vertex if and only if that value belongs to the domain of the corresponding variable; there is an edge between a variable vertex and its corresponding value vertex.

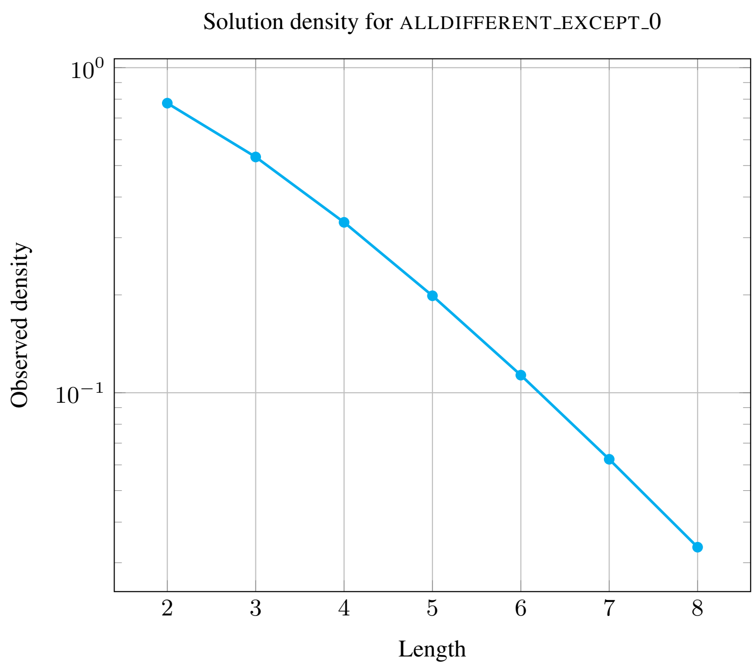
- Second, Dulmage-Mendelsohn decomposition [157] is used to characterise all edges that do not belong to any var-perfect matching, and therefore prune the corresponding variables.

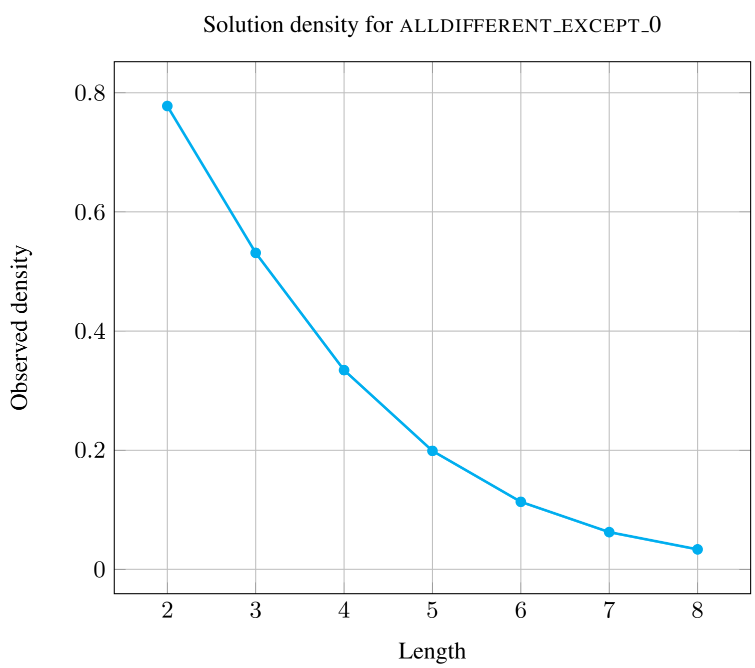
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	7	34	209	1546	13327	130922	1441729

Number of solutions for ALLDIFFERENT_EXCEPT_0: domains $0..n$

[5]A *var-perfect matching* is a maximum matching covering all vertices representing variables.





Systems ALL_DIFFERENT in **MiniZinc**.

See also **cost variant:** WEIGHTED_PARTIAL_ALLDIFF.

hard version: ALLDIFFERENT.

implied by: ALLDIFFERENT.

implies: MULTI_GLOBAL_CONTIGUITY.

Keywords

characteristic of a constraint: joker value, all different, sort based reformulation, automaton, automaton with array of counters.

constraint type: value constraint, relaxation.

filtering: bipartite matching, arc-consistency.

final graph structure: one_succ.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• variables1.var $\neq 0$ • variables1.var $=$ variables2.var
Graph property(ies)	**MAX_NSCC** ≤ 1

Graph model The graph model is the same as the one used for the ALLDIFFERENT constraint, except that we discard all variables that are assigned value 0.

Parts (A) and (B) of Figure 5.42 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph. The ALLDIFFERENT_EXCEPT_0 holds since all the strongly connected components have at most one vertex: a value different from 0 is used at most once.

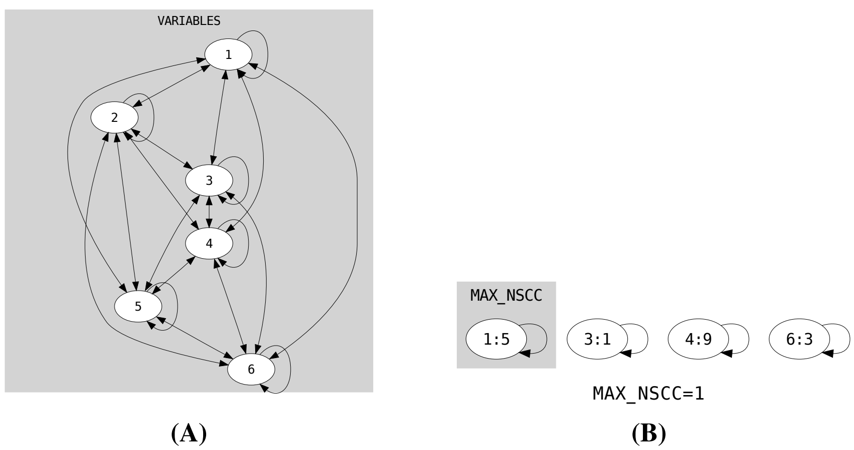


Figure 5.42: Initial and final graph of the ALLDIFFERENT_EXCEPT_0 constraint

Automaton Figure 5.43 depicts the automaton associated with the ALLDIFFERENT_EXCEPT_0 constraint. To each variable $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR}_i$ and $\texttt{S}_i$: $\texttt{VAR}_i \neq 0 \Leftrightarrow \texttt{S}_i$. The automaton counts the number of occurrences of each value different from 0 and finally imposes that each non-zero value is taken at most one time.

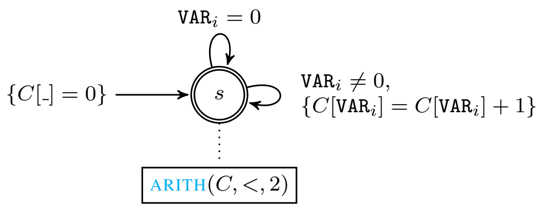


Figure 5.43: Automaton of the ALLDIFFERENT_EXCEPT_0 constraint

5.19 ALLDIFFERENT_INTERVAL

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from ALLDIFFERENT.

Constraint ALLDIFFERENT_INTERVAL(VARIABLES, SIZE_INTERVAL)

Synonyms ALLDIFF_INTERVAL, ALLDISTINCT_INTERVAL.

Arguments

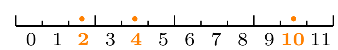
VARIABLES : collection(var−dvar)
SIZE_INTERVAL : int

Restrictions

required(VARIABLES, var)
SIZE_INTERVAL > 0

Purpose Enforce all variables of the collection VARIABLES to belong to distinct intervals. The intervals are defined by [SIZE_INTERVAL $\cdot k$, SIZE_INTERVAL $\cdot k +$ SIZE_INTERVAL $- 1$] where k is an integer.

Example $(\langle 2, 4, 10\rangle, 3)$



In the example, the second argument SIZE_INTERVAL $=$ 3 defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Since the three variables of the collection VARIABLES take values that are respectively located within the three following distinct intervals $[0, 2]$, $[3, 5]$ and $[9, 11]$, the ALLDIFFERENT_INTERVAL constraint holds.

All solutions Figure 5.44 gives all solutions to the following non ground instance of the ALLDIFFERENT_INTERVAL constraint: $V_1 \in [0, 7]$, $V_2 \in [1, 2]$, $V_3 \in [2, 3]$, $V_4 \in [0, 9]$, ALLDIFFERENT_INTERVAL$(\langle V_1, V_2, V_3, V_4\rangle, \mathbf{3})$.

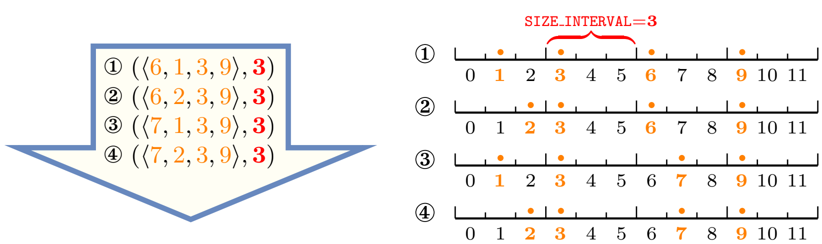


Figure 5.44: All solutions corresponding to the non ground example of the ALLDIFFERENT_INTERVAL constraint of the **All solutions** slot

Typical

$|$VARIABLES$| > 1$
SIZE_INTERVAL > 1
SIZE_INTERVAL $<$ max(3,range(VARIABLES.var))

Symmetries

- Items of `VARIABLES` are permutable.
- A value of `VARIABLES.var` that belongs to the k-th interval, of size `SIZE_INTERVAL`, can be renamed to any unused value of the same interval.
- Two distinct values of `VARIABLES.var` that belong to two distinct intervals, of size `SIZE_INTERVAL`, can be swapped.

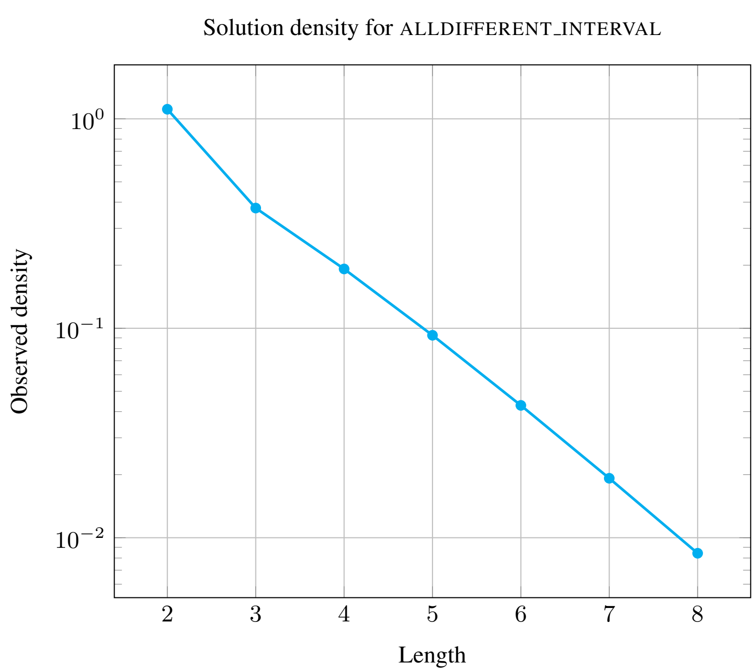
Arg. properties

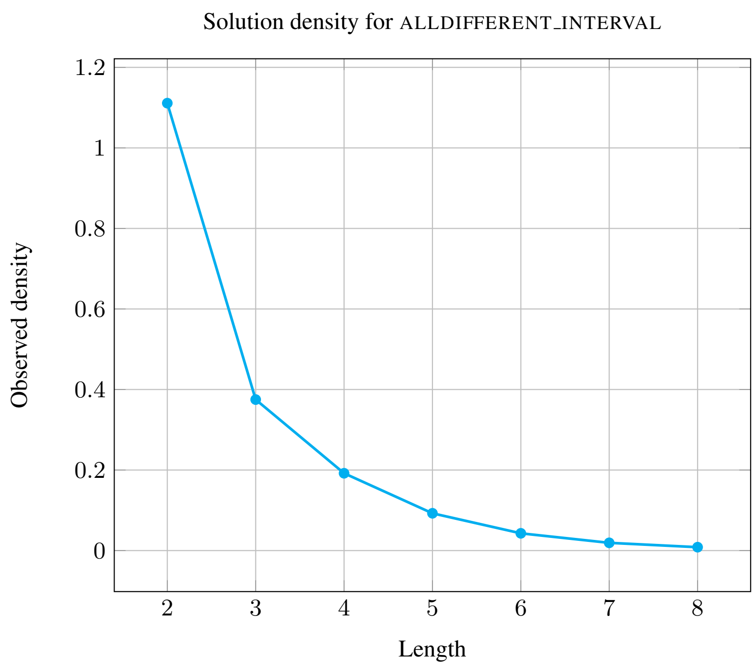
Contractible wrt. `VARIABLES`.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	10	24	120	720	5040	40320	362880

Number of solutions for ALLDIFFERENT_INTERVAL: domains $0..n$

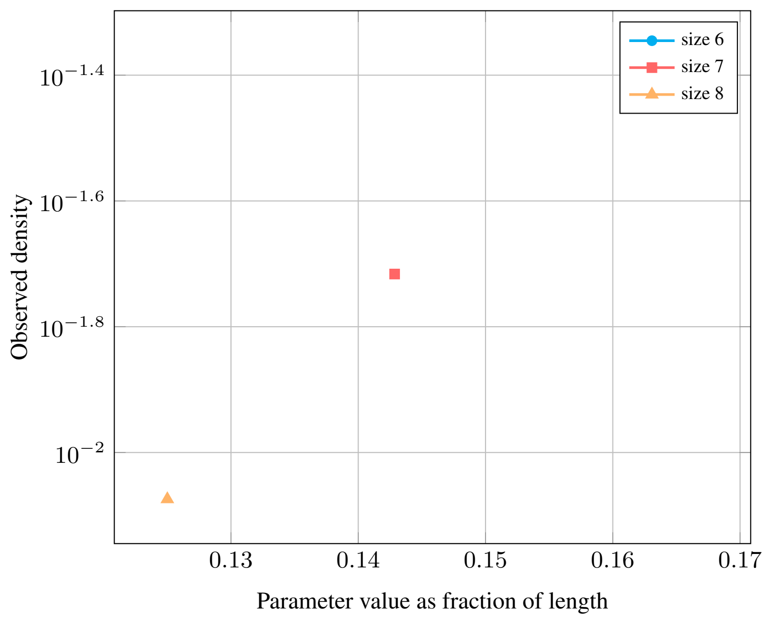




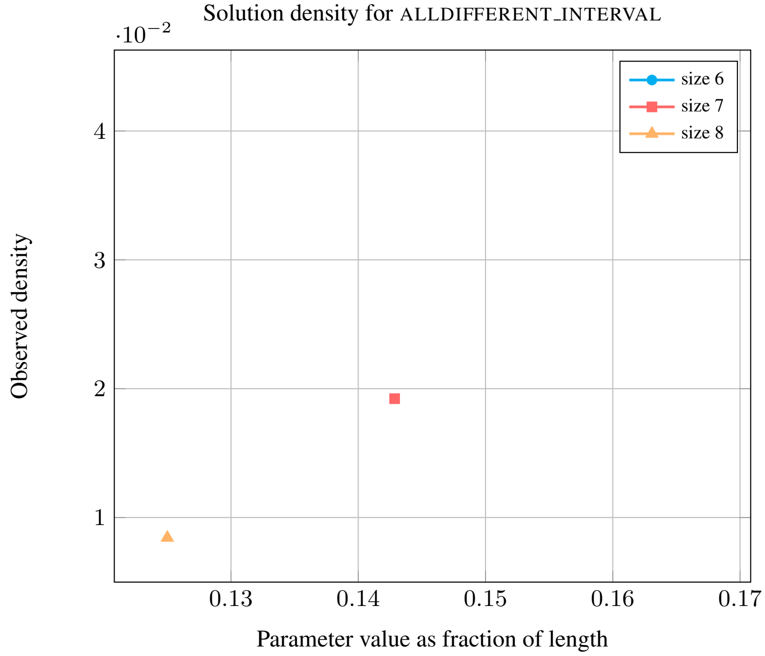
Length (n)		2	3	4	5	6	7	8
Total		10	24	120	720	5040	40320	362880
Parameter value	1	6	24	120	720	5040	40320	362880
	2	4	-	-	-	-	-	-

Solution count for ALLDIFFERENT_INTERVAL: domains $0..n$

Solution density for ALLDIFFERENT_INTERVAL



Solution density for ALLDIFFERENT_INTERVAL



See also **implied by:** ALL_MIN_DIST.

specialisation: ALLDIFFERENT (`variable`/`constant` *replaced by* `variable`).

Keywords

characteristic of a constraint: all different, sort based reformulation, automaton, automaton with array of counters.

constraint type: value constraint.

filtering: arc-consistency.

final graph structure: one_succ.

modelling: interval.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL
Graph property(ies)	**MAX_NSCC** ≤ 1
Graph class	ONE_SUCC

Graph model Similar to the ALLDIFFERENT constraint, but we replace the binary *equality* constraint of the ALLDIFFERENT constraint by the fact that two variables are respectively assigned to two values that belong to the same interval. We generate a *clique* with a *belong to the same interval* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed 1.

Parts (A) and (B) of Figure 5.45 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected component of the final graph.

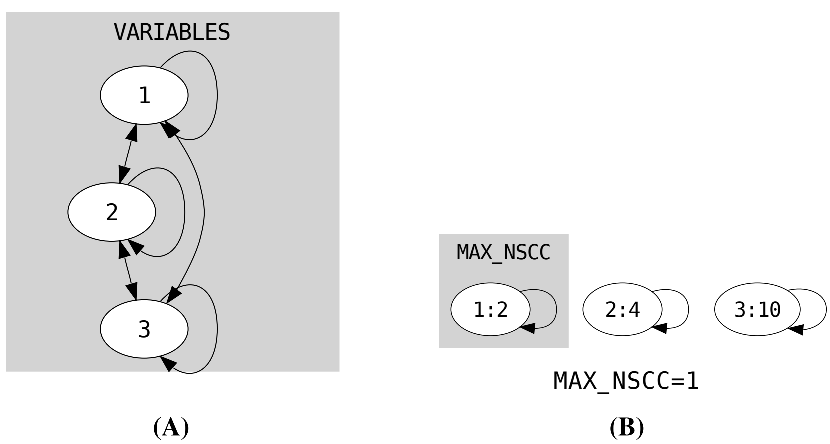


Figure 5.45: Initial and final graph of the ALLDIFFERENT_INTERVAL constraint

Automaton Figure 5.46 depicts the automaton associated with the ALLDIFFERENT_INTERVAL constraint. To each item of the collection VARIABLES corresponds a signature variable $\mathtt{S}_i$ that is equal to 1. For each interval $[\mathtt{SIZE\_INTERVAL}\cdot k, \mathtt{SIZE\_INTERVAL}\cdot k+\mathtt{SIZE\_INTERVAL}-1]$ of values the automaton counts the number of occurrences of its values and finally imposes that the values of an interval are taken at most once.

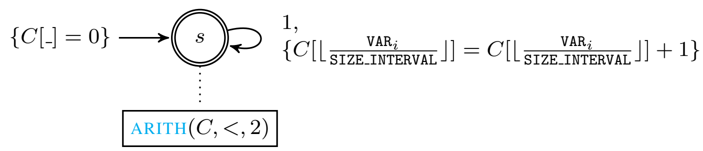


Figure 5.46: Automaton of the ALLDIFFERENT_INTERVAL constraint

5.20 ALLDIFFERENT_MODULO

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from ALLDIFFERENT.

Constraint ALLDIFFERENT_MODULO(VARIABLES, M)

Synonyms ALLDIFF_MODULO, ALLDISTINCT_MODULO.

Arguments

VARIABLES	:	collection(var−dvar)
M	:	int

Restrictions

required(VARIABLES, var)
$\mathtt{M} > 0$
$\mathtt{M} \geq |\mathtt{VARIABLES}|$

Purpose Enforce all variables of the collection VARIABLES to have a distinct rest when divided by M.

Example $(\langle 25, 1, 14, 3\rangle, 5)$

The equivalence classes associated with values 25, 1, 14 and 3 are respectively equal to $25 \bmod 5 = 0$, $1 \bmod 5 = 1$, $14 \bmod 5 = 4$ and $3 \bmod 5 = 3$. Since they are distinct the ALLDIFFERENT_MODULO constraint holds.

All solutions Figure 5.47 gives all solutions to the following non ground instance of the ALLDIFFERENT_MODULO constraint: $\mathtt{V}_1 \in \{0, 5\}$, $\mathtt{V}_2 \in [2, 3]$, $\mathtt{V}_3 \in [3, 4]$, $\mathtt{V}_4 \in [1, 2]$, $\mathtt{V}_5 \in [6, 10]$, ALLDIFFERENT_MODULO$(\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4, \mathtt{V}_5\rangle, 5)$.

① $(\langle 0_0, 2_2, 3_3, 1_1, 9_4\rangle, 5)$
② $(\langle 0_0, 2_2, 4_4, 1_1, 8_3\rangle, 5)$
③ $(\langle 0_0, 3_3, 4_4, 1_1, 7_2\rangle, 5)$
④ $(\langle 0_0, 3_3, 4_4, 2_2, 6_1\rangle, 5)$

⑤ $(\langle 5_0, 2_2, 3_3, 1_1, 9_4\rangle, 5)$
⑥ $(\langle 5_0, 2_2, 4_4, 1_1, 8_3\rangle, 5)$
⑦ $(\langle 5_0, 3_3, 4_4, 1_1, 7_2\rangle, 5)$
⑧ $(\langle 5_0, 3_3, 4_4, 2_2, 6_1\rangle, 5)$

Figure 5.47: All solutions corresponding to the non ground example of the ALLDIFFERENT_MODULO constraint of the **All solutions** slot, where the indices (in orange) correspond to the values modulo $\mathtt{M} = 5$: all indices attached to a solution are distinct.

Typical

$|\mathtt{VARIABLES}| > 2$
$\mathtt{M} > 1$

Symmetries

- Items of `VARIABLES` are permutable.
- A value u of `VARIABLES.var` can be renamed to any value v such that v is congruent to u modulo `M`.
- Two distinct values u and v of `VARIABLES.var` such that $u \bmod \mathtt{M} \neq v \bmod \mathtt{M}$ can be swapped.

Arg. properties

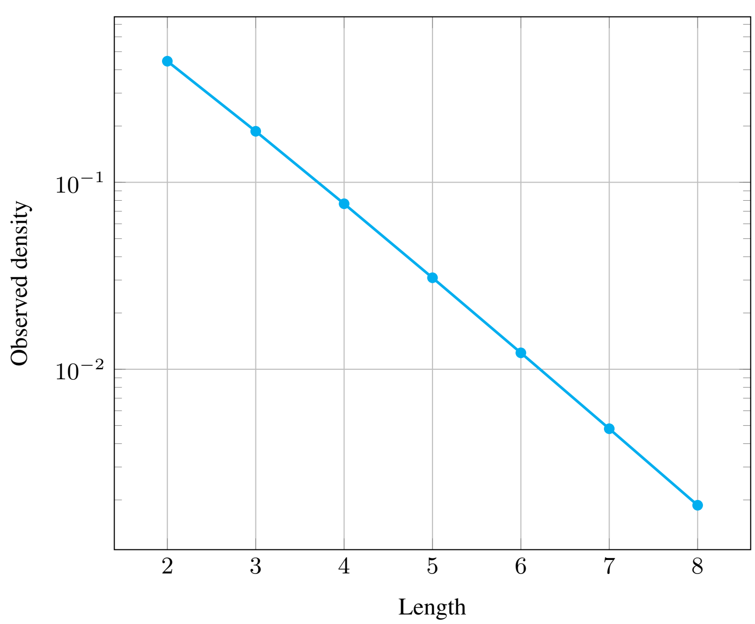
Contractible wrt. `VARIABLES`.

Counting

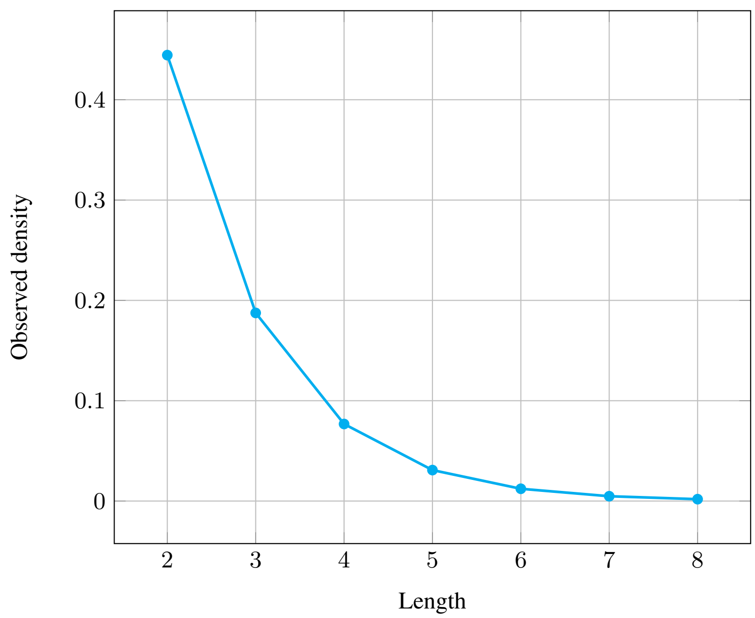
Length (n)	2	3	4	5	6	7	8
Solutions	4	12	48	240	1440	10080	80640

Number of solutions for ALLDIFFERENT_MODULO: domains $0..n$

Solution density for ALLDIFFERENT_MODULO

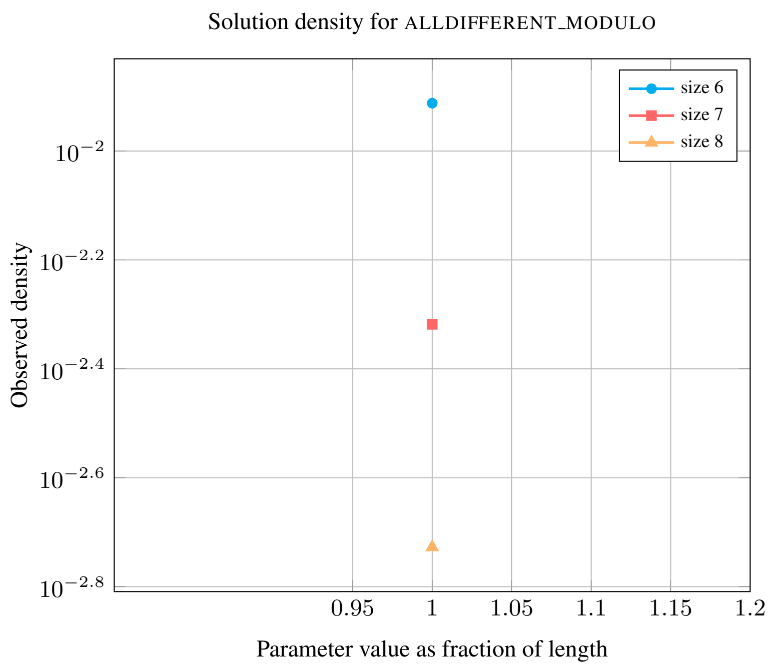


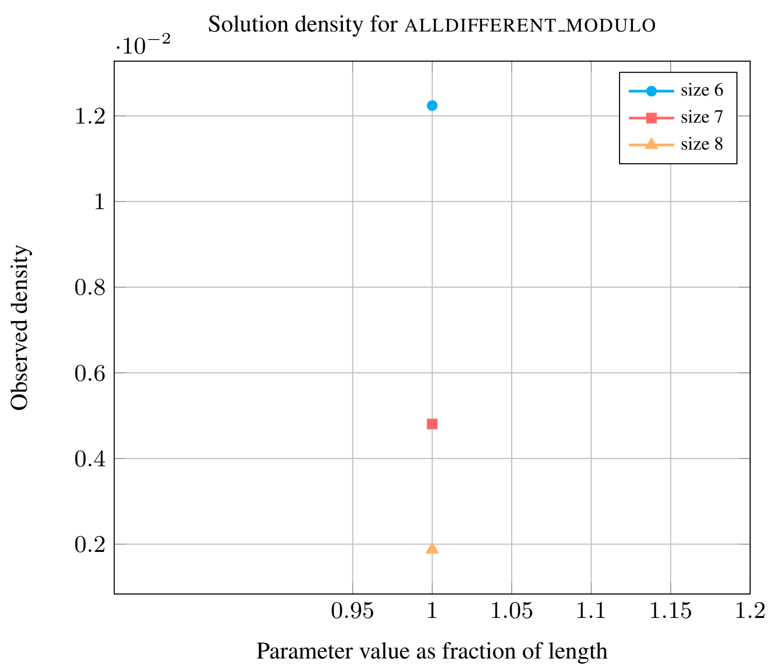
Solution density for ALLDIFFERENT_MODULO



Length (n)		2	3	4	5	6	7	8
Total		4	12	48	240	1440	10080	80640
Parameter value	2	4	-	-	-	-	-	-
	3	-	12	-	-	-	-	-
	4	-	-	48	-	-	-	-
	5	-	-	-	240	-	-	-
	6	-	-	-	-	1440	-	-
	7	-	-	-	-	-	10080	-
	8	-	-	-	-	-	-	80640

Solution count for ALLDIFFERENT_MODULO: domains $0..n$





See also

implies: SOFT_ALLDIFFERENT_VAR.

specialisation: ALLDIFFERENT (`variable` mod `constant` *replaced by* `variable`).

Keywords

characteristic of a constraint: modulo, all different, sort based reformulation, automaton, automaton with array of counters.

constraint type: value constraint.

filtering: arc-consistency.

final graph structure: one_succ.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var mod M = variables2.var mod M
Graph property(ies)	**MAX_NSCC** ≤ 1
Graph class	ONE_SUCC

Graph model Exploit the same model used for the ALLDIFFERENT constraint. We replace the binary *equality* constraint by another equivalence relation depicted by the arc constraint. We generate a *clique* with a binary *equality modulo* M constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed 1.

Parts (A) and (B) of Figure 5.48 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph.

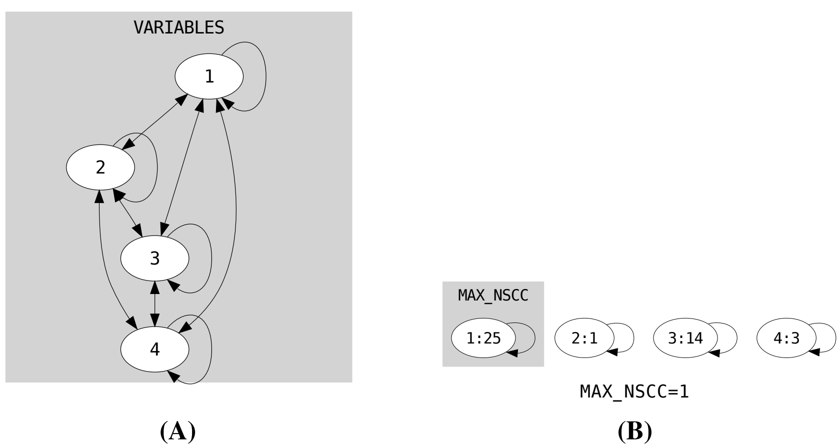


Figure 5.48: Initial and final graph of the ALLDIFFERENT_MODULO constraint

Automaton Figure 5.49 depicts the automaton associated with the ALLDIFFERENT_MODULO constraint. To each item of the collection VARIABLES corresponds a signature variable S_i that is equal to 1. The automaton counts for each equivalence class the number of used values and finally imposes that each equivalence class is used at most one time.

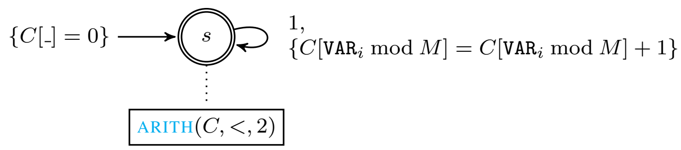


Figure 5.49: Automaton of the ALLDIFFERENT_MODULO constraint

5.21 ALLDIFFERENT_ON_INTERSECTION

► ▷ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from COMMON and ALLDIFFERENT.

Constraint ALLDIFFERENT_ON_INTERSECTION(VARIABLES1, VARIABLES2)

Synonyms ALLDIFF_ON_INTERSECTION, ALLDISTINCT_ON_INTERSECTION.

Arguments

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

Restrictions

required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose The values that both occur in the VARIABLES1 and VARIABLES2 collections have only one occurrence.

Example $(\langle 5,9,1,5\rangle, \langle 2,1,6,9,6,2\rangle)$

The ALLDIFFERENT_ON_INTERSECTION constraint holds since the values 9 and 1 that both occur in $\langle 5,9,1,5\rangle$ as well as in $\langle 2,1,6,9,6,2\rangle$ have exactly one occurrence in each collection.

All solutions Figure 5.50 gives all solutions to the following non ground instance of the ALLDIFFERENT_ON_INTERSECTION constraint: $U_1 \in [2,3]$, $U_2 \in [1,2]$, $V_1 \in [2,3]$, $V_2 \in [2,2]$, $V_3 \in [0,1]$, ALLDIFFERENT_ON_INTERSECTION$(\langle U_1, U_2\rangle, \langle V_1, V_2, V_3\rangle)$.

① $(\langle \mathbf{2},1\rangle,\langle 3,\mathbf{2},0\rangle)$
② $(\langle \mathbf{2},\mathbf{1}\rangle,\langle 3,\mathbf{2},\mathbf{1}\rangle)$
③ $(\langle 3,1\rangle,\langle 2,2,0\rangle)$
④ $(\langle 3,\mathbf{1}\rangle,\langle 2,2,\mathbf{1}\rangle)$

⑤ $(\langle \mathbf{3},1\rangle,\langle \mathbf{3},2,0\rangle)$
⑥ $(\langle \mathbf{3},\mathbf{1}\rangle,\langle \mathbf{3},2,\mathbf{1}\rangle)$
⑦ $(\langle \mathbf{3},\mathbf{2}\rangle,\langle \mathbf{3},\mathbf{2},0\rangle)$
⑧ $(\langle \mathbf{3},\mathbf{2}\rangle,\langle \mathbf{3},\mathbf{2},1\rangle)$

Figure 5.50: All solutions corresponding to the non ground example of the ALLDIFFERENT_ON_INTERSECTION constraint of the **All solutions** slot, where values that occur in both collections are coloured in orange

Typical

$|\texttt{VARIABLES1}| > 1$
$|\texttt{VARIABLES2}| > 1$

Symmetries

- Arguments are permutable w.r.t. permutation (`VARIABLES1`, `VARIABLES2`).
- Items of `VARIABLES1` are permutable.
- Items of `VARIABLES2` are permutable.
- All occurrences of two distinct values in `VARIABLES1.var` or `VARIABLES2.var` can be swapped; all occurrences of a value in `VARIABLES1.var` or `VARIABLES2.var` can be renamed to any unused value.

Arg. properties

- Contractible wrt. `VARIABLES1`.
- Contractible wrt. `VARIABLES2`.

See also

common keyword: COMMON, NVALUE_ON_INTERSECTION *(constraint on the intersection)*.

implied by: DISJOINT.

implies: SAME_INTERSECTION.

root concept: ALLDIFFERENT.

Keywords

characteristic of a constraint: all different, automaton, automaton with array of counters.

constraint arguments: constraint between two collections of variables.

constraint type: constraint on the intersection, value constraint.

final graph structure: connected component, acyclic, bipartite, no loop.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**MAX_NCC** ≤ 2
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.51 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NCC** graph property we show one of the largest connected components of the final graph. The ALLDIFFERENT_ON_INTERSECTION constraint holds since each connected component has at most two vertices. Note that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.

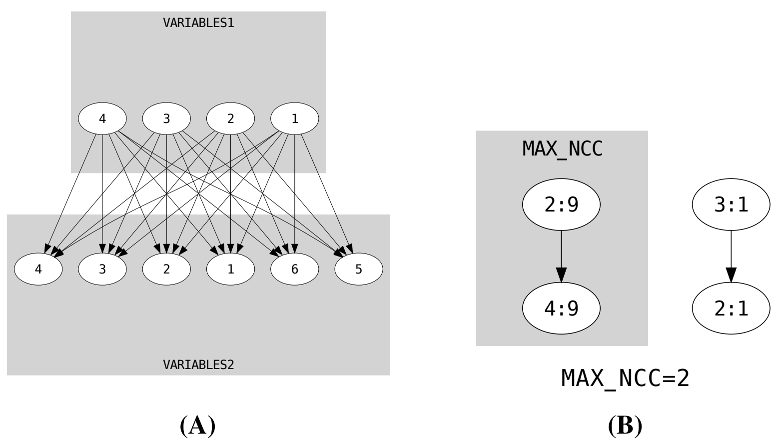


Figure 5.51: Initial and final graph of the ALLDIFFERENT_ON_INTERSECTION constraint

Automaton Figure 5.52 depicts the automaton associated with the ALLDIFFERENT_ON_INTERSECTION constraint. To each variable $\texttt{VAR1}_i$ of the collection VARIABLES1 corresponds a signature variable $\texttt{S}_i$ that is equal to 0. To each variable $\texttt{VAR2}_i$ of the collection VARIABLES2 corresponds a signature variable $\texttt{S}_{i+|\texttt{VARIABLES1}|}$ that is equal to 1. The automaton first counts the number of occurrences of each value assigned to the variables of the VARIABLES1 collection. It then counts the number of occurrences of each value assigned to the variables of the VARIABLES2 collection. Finally, the automaton imposes that each value is not taken by two variables of both collections.

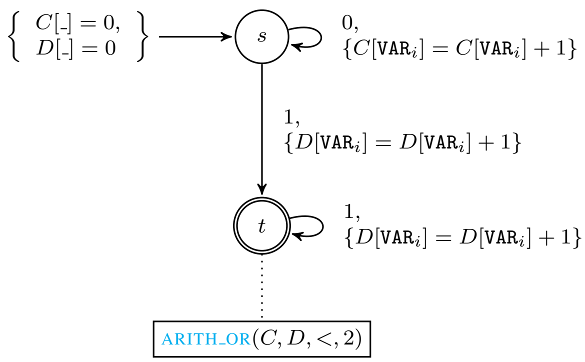


Figure 5.52: Automaton of the ALLDIFFERENT_ON_INTERSECTION constraint

5.22 ALLDIFFERENT_PARTITION

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from ALLDIFFERENT.

Constraint ALLDIFFERENT_PARTITION(VARIABLES, PARTITIONS)

Synonyms ALLDIFF_PARTITION, ALLDISTINCT_PARTITION.

Type

VALUES : collection(val−int)

Arguments

VARIABLES : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

$|\texttt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)
$|\texttt{VARIABLES}| \leq |\texttt{PARTITIONS}|$
required(VARIABLES, var)
$|\texttt{PARTITIONS}| \geq 2$
required(PARTITIONS, p)

Purpose

Enforce all variables of the collection VARIABLES to take values that belong to distinct partitions.

Example

$$(\langle 6, 3, 4\rangle, \langle \texttt{p} - \langle 1, 3\rangle, \texttt{p} - \langle 4\rangle, \texttt{p} - \langle 2, 6\rangle\rangle)$$

Since all variables take values that are located within distinct partitions the ALLDIFFERENT_PARTITION constraint holds.

Typical $|\texttt{VARIABLES}| > 2$

Symmetries

- Items of VARIABLES are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- A value of VARIABLES.var can be renamed to any value that belongs to the same partition of PARTITIONS.
- Two distinct values of VARIABLES.var that do not belong to the same partition of PARTITIONS can be swapped.

Arg. properties Contractible wrt. VARIABLES.

See also

common keyword: IN_SAME_PARTITION *(partition)*.

specialisation: ALLDIFFERENT *(*`variable` $\in$ `partition` *replaced by* `variable`*)*.

used in graph description: IN_SAME_PARTITION.

Keywords

characteristic of a constraint: partition, all different, sort based reformulation.

constraint type: value constraint.

filtering: arc-consistency.

final graph structure: one_succ.

modelling: incompatible pairs of values.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*↦collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	**MAX_NSCC** ≤ 1
Graph class	ONE_SUCC

Graph model Similar to the ALLDIFFERENT constraint, but we replace the binary *equality* constraint of the ALLDIFFERENT constraint by the fact that two variables are respectively assigned to two values that belong to the same partition. We generate a *clique* with a IN_SAME_PARTITION constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed 1.

Parts (A) and (B) of Figure 5.53 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph.

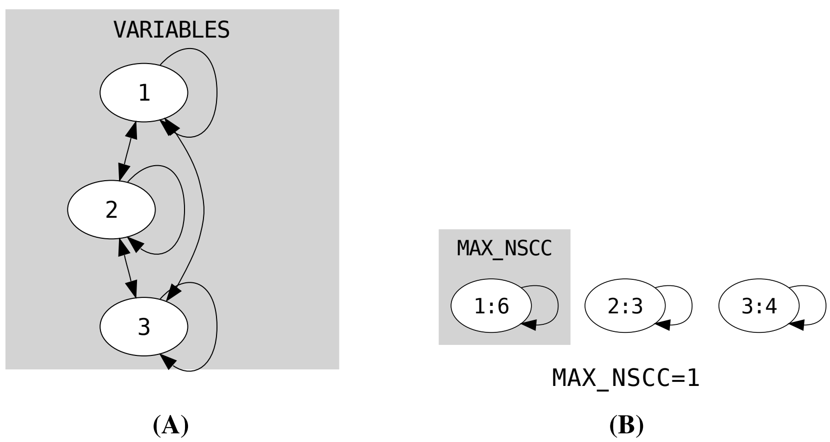


Figure 5.53: Initial and final graph of the ALLDIFFERENT_PARTITION constraint

5.23 ALLDIFFERENT_SAME_VALUE

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from ALLDIFFERENT.

Constraint ALLDIFFERENT_SAME_VALUE(NSAME, VARIABLES1, VARIABLES2)

Synonyms ALLDIFF_SAME_VALUE, ALLDISTINCT_SAME_VALUE.

Arguments

NSAME	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)

Restrictions

$\texttt{NSAME} \geq 0$
$\texttt{NSAME} \leq |\texttt{VARIABLES1}|$
$|\texttt{VARIABLES1}| = |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose All the values assigned to the variables of the collection VARIABLES1 are pairwise distinct. NSAME is equal to number of constraints of the form $\texttt{VARIABLES1}[i].\texttt{var} = \texttt{VARIABLES2}[i].\texttt{var}$ $(1 \leq i \leq |\texttt{VARIABLES1}|)$ that hold.

Example $(2, \langle 7, 3, 1, 5\rangle, \langle 1, 3, 1, 7\rangle)$

The ALLDIFFERENT_SAME_VALUE constraint holds since:

- All the values 7, 3, 1 and 5 are distinct,
- Among the four expressions $7 = 1$, $3 = 3$, $1 = 1$ and $5 = 7$ exactly 2 conditions hold.

All solutions Figure 5.54 gives all solutions to the following non ground instance of the ALLDIFFERENT_SAME_VALUE constraint: $\texttt{U}_1 \in [2, 4]$, $\texttt{U}_2 \in [1, 2]$, $\texttt{U}_3 \in [1, 4]$, $\texttt{U}_4 \in [2, 4]$, $\texttt{V}_1 \in [2, 3]$, $\texttt{V}_2 = 2$, $\texttt{V}_3 \in [0, 1]$, $\texttt{V}_4 \in [0, 3]$, ALLDIFFERENT_SAME_VALUE$(\mathbf{3}, \langle \texttt{U}_1, \texttt{U}_2, \texttt{U}_3, \texttt{U}_4\rangle, \langle \texttt{V}_1, \texttt{V}_2, \texttt{V}_3, \texttt{V}_4\rangle)$.

Typical

$\texttt{NSAME} < |\texttt{VARIABLES1}|$
$|\texttt{VARIABLES1}| > 2$

Symmetries

- Items of VARIABLES1 and VARIABLES2 are permutable (*same permutation used*).
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Arg. properties Functional dependency: NSAME determined by VARIABLES1 and VARIABLES2.

① $(\mathbf{3}, \langle \mathbf{3}, \mathbf{2}, \mathbf{1}, 4\rangle, \langle \mathbf{3}, \mathbf{2}, \mathbf{1}, 0\rangle)$
② $(\mathbf{3}, \langle \mathbf{3}, \mathbf{2}, \mathbf{1}, 4\rangle, \langle \mathbf{3}, \mathbf{2}, \mathbf{1}, 1\rangle)$
③ $(\mathbf{3}, \langle \mathbf{3}, \mathbf{2}, \mathbf{1}, 4\rangle, \langle \mathbf{3}, \mathbf{2}, \mathbf{1}, 2\rangle)$

④ $(\mathbf{3}, \langle \mathbf{3}, \mathbf{2}, \mathbf{1}, 4\rangle, \langle \mathbf{3}, \mathbf{2}, \mathbf{1}, 3\rangle)$
⑤ $(\mathbf{3}, \langle 4, \mathbf{2}, \mathbf{1}, \mathbf{3}\rangle, \langle 2, \mathbf{2}, \mathbf{1}, \mathbf{3}\rangle)$
⑥ $(\mathbf{3}, \langle 4, \mathbf{2}, \mathbf{1}, \mathbf{3}\rangle, \langle 3, \mathbf{2}, \mathbf{1}, \mathbf{3}\rangle)$

Figure 5.54: All solutions corresponding to the non ground example of the ALLDIFFERENT_SAME_VALUE constraint of the **All solutions** slot, where identical values at a same position in both collections are coloured in orange

Usage When all variables of the second collection are initially bound to distinct values the ALLDIFFERENT_SAME_VALUE constraint can be explained in the following way:

- We interpret the variables of the second collection as the previous solution to a problem where all variables have to be distinct.
- We interpret the variables of the first collection as the current solution to find, where all variables should again be pairwise distinct.

The variable `NSAME` measures the `distance` of the current solution from the previous solution. This corresponds to the number of variables of `VARIABLES2` that are assigned to the same previous value.

See also **root concept:** ALLDIFFERENT.

Keywords **characteristic of a constraint:** sort based reformulation, automaton, automaton with array of counters.

constraint type: proximity constraint.

modelling: functional dependency.

Cond. implications ALLDIFFERENT_SAME_VALUE(`NSAME`, `VARIABLES1`, `VARIABLES2`)
with $2 * \texttt{NSAME} = |\texttt{VARIABLES1}|$
implies DIFFER_FROM_EXACTLY_K_POS(`K`, `VECTOR1`, `VECTOR2`).

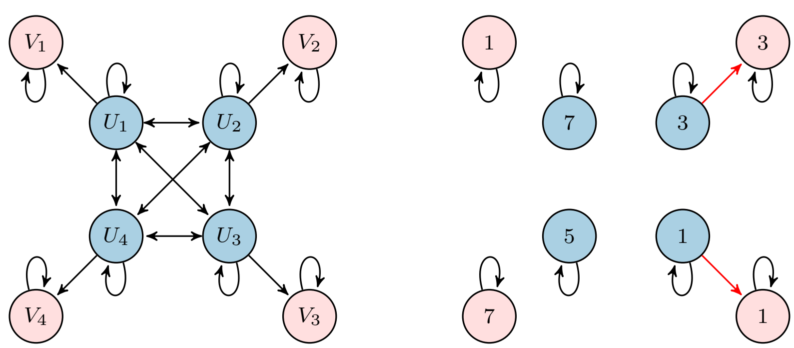
Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT*(*CLIQUE*,*LOOP*,=) $\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NARC_NO_LOOP** = NSAME

Graph model

The arc generator *PRODUCT*(*CLIQUE*,*LOOP*,=) is used in order to generate all the arcs of the initial graph:

- The arc generator *CLIQUE* creates all links between the items of the first collection VARIABLES1,
- The arc generator *LOOP* creates a loop for each item of the second collection VARIABLES2,
- Finally the arc generator *PRODUCT*(=) creates an arc between items located at the same position in the collections VARIABLES1 and VARIABLES2.

Part (A) of Figure 5.55 gives the initial graph associated with the **Example** slot. Variables of collection VARIABLES1 are coloured, while variables of collection VARIABLES2 are kept in white. Part (B) represents the final graph associated with the **Example** slot. In this graph each vertex constitutes a strongly connected component and the number of arcs that do not correspond to a loop is equal to 2 (i.e., NSAME).



(A) (B)

Figure 5.55: (A) Initial and (B) final graph of the ALLDIFFERENT_SAME_VALUE$(2, \langle U_1, U_2, U_3, U_4\rangle, \langle V_1, V_2, V_3, V_4\rangle)$ constraint with $U_1 = 7$, $U_2 = 3$, $U_3 = 1$, $U_4 = 5$ and $V_1 = 1$, $V_2 = 3$, $V_3 = 1$, $V_4 = 7$ (in Part (B) arcs in red correspond to the arcs counted by the argument NSAME)

Automaton Figure 5.56 depicts the automaton associated with the ALLDIFFERENT_SAME_VALUE constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively denote the i^{th} variables of the VARIABLES1 and VARIABLES2 collections. To each pair of variables $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable $\texttt{S}_i$. The following signature constraint links $\texttt{VAR1}_i$, $\texttt{VAR2}_i$ and $\texttt{S}_i$: $\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow \texttt{S}_i$.

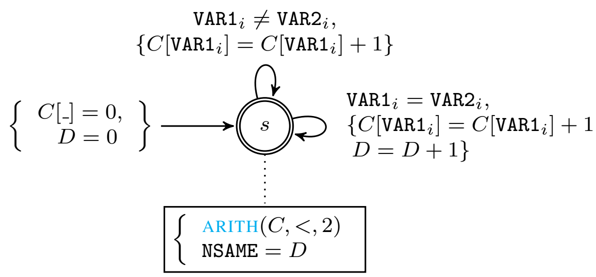


Figure 5.56: Automaton of the ALLDIFFERENT_SAME_VALUE constraint

5.24 ALLPERM

▶ ▷ ◀ DESCRIPTION LINKS GRAPH

Origin [179]

Constraint ALLPERM(MATRIX)

Synonyms ALL_PERM, ALL_PERMUTATIONS.

Type VECTOR : collection(var−dvar)

Argument MATRIX : collection(vec − VECTOR)

Restrictions
|VECTOR| ≥ 1
required(VECTOR, var)
required(MATRIX, vec)
same_size(MATRIX, vec)

Purpose Given a matrix $\mathcal{M}$ of domain variables, enforces that the first row is lexicographically less than or equal to all permutations of all other rows. Note that the components of a given vector of the matrix $\mathcal{M}$ may be equal.

Example $(\langle \texttt{vec} - \langle 1,2,3\rangle, \texttt{vec} - \langle 3,1,2\rangle\rangle)$

The ALLPERM constraint holds since vector $\langle 1,2,3\rangle$ is lexicographically less than or equal to all the permutations of vector $\langle 3,1,2\rangle$ (i.e., $\langle 1,2,3\rangle$, $\langle 1,3,2\rangle$, $\langle 2,1,3\rangle$, $\langle 2,3,1\rangle$, $\langle 3,1,2\rangle$, $\langle 3,2,1\rangle$).

All solutions Figure 5.57 gives all solutions to the following non ground instance of the ALLPERM constraint: $\mathtt{U}_1 \in [1,2]$, $\mathtt{U}_2 \in [1,2]$, $\mathtt{U}_3 \in [1,3]$, $\mathtt{V}_1 \in [0,1]$, $\mathtt{V}_2 \in [1,2]$, $\mathtt{V}_3 \in [0,2]$, ALLPERM$(\langle\langle \mathtt{U}_1, \mathtt{U}_2, \mathtt{U}_3\rangle, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3\rangle\rangle)$.

① $(\langle\langle 1,1,1\rangle, \langle 1,1,1\rangle\rangle)$
② $(\langle\langle 1,1,1\rangle, \langle 1,1,2\rangle\rangle)$
③ $(\langle\langle 1,1,1\rangle, \langle 1,2,1\rangle\rangle)$
④ $(\langle\langle 1,1,1\rangle, \langle 1,2,2\rangle\rangle)$
⑤ $(\langle\langle 1,1,2\rangle, \langle 1,1,2\rangle\rangle)$

⑥ $(\langle\langle 1,1,2\rangle, \langle 1,2,1\rangle\rangle)$
⑦ $(\langle\langle 1,1,2\rangle, \langle 1,2,2\rangle\rangle)$
⑧ $(\langle\langle 1,1,3\rangle, \langle 1,2,2\rangle\rangle)$
⑨ $(\langle\langle 1,2,1\rangle, \langle 1,2,2\rangle\rangle)$
⑩ $(\langle\langle 1,2,2\rangle, \langle 1,2,2\rangle\rangle)$

Figure 5.57: All solutions corresponding to the non ground example of the ALLPERM constraint of the **All solutions** slot

Typical

$|\texttt{VECTOR}| > 1$
$|\texttt{MATRIX}| > 1$

Symmetry

One and the same constant can be added to the `var` attribute of all items of `MATRIX.vec`.

Arg. properties

Suffix-contractible wrt. `MATRIX.vec` (*remove items from same position*).

Usage

A *symmetry-breaking* constraint.

See also

common keyword: LEX2, LEX_CHAIN_LESSEQ *(matrix symmetry,lexicographic order)*, LEX_LESSEQ *(lexicographic order)*, LEX_LESSEQ_ALLPERM *(matrix symmetry,lexicographic order)*, STRICT_LEX2 *(lexicographic order)*.

part of system of constraints: LEX_LESSEQ_ALLPERM.

used in graph description: LEX_LESSEQ_ALLPERM.

Keywords

characteristic of a constraint: sort based reformulation, vector.

constraint type: order constraint, system of constraints.

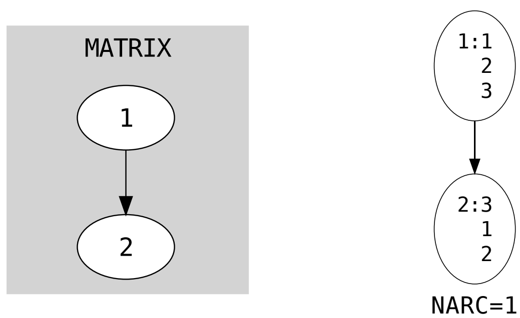
final graph structure: acyclic, bipartite.

modelling: matrix, matrix model.

symmetry: matrix symmetry, symmetry, lexicographic order.

Arc input(s)	MATRIX
Arc generator	$CLIQUE(<) \mapsto$ collection(matrix1, matrix2)
Arc arity	2
Arc constraint(s)	• matrix1.key $= 1$ • matrix2.key > 1 • LEX_LESSEQ_ALLPERM(matrix1.vec, matrix2.vec)
Graph property(ies)	**NARC** $= \vert$MATRIX$\vert - 1$
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model We generate a graph with an arc constraint LEX_LESSEQ_ALLPERM between the vertex corresponding to the first item of the MATRIX collection and the vertices associated with all other items of the MATRIX collection. This is achieved by specifying that (1) an arc should start from the first item (i.e., matrix1.key $= 1$) and (2) an arc should not end on the first item (i.e., matrix2.key > 1). We finally state that all these arcs should belong to the final graph. Parts (A) and (B) of Figure 5.58 respectively show the initial and final graph associated with the **Example** slot.



(A) **(B)**

Figure 5.58: Initial and final graph of the ALLPERM constraint

5.25 AMONG

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [47]

Constraint AMONG(NVAR, VARIABLES, VALUES)

Synonyms BETWEEN, COUNT.

Arguments

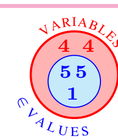
NVAR	:	dvar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int)

Restrictions

NVAR ≥ 0
NVAR $\leq$ |VARIABLES|
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose NVAR is the number of variables of the collection VARIABLES that take their values in VALUES.

Example $(3, \langle 4, 5, 5, 4, 1\rangle, \langle 1, 5, 8\rangle)$



The AMONG constraint holds since exactly 3 values of the collection of variables $\langle 4, 5, 5, 4, 1\rangle$ belong to the set of values $\{1, 5, 8\}$.

All solutions Figure 5.59 gives all solutions to the following non ground instance of the AMONG constraint: $\mathtt{V}_1 \in [1, 5]$, $\mathtt{V}_2 \in [3, 9]$, $\mathtt{V}_3 \in [5, 6]$, $\mathtt{V}_4 \in [2, 3]$, AMONG($\mathbf{3}$, $\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4\rangle$, $\langle \mathbf{2}, \mathbf{4}\rangle$).

① $(\mathbf{3}, \langle \mathbf{2}, \mathbf{4}, 5, \mathbf{2}\rangle, \langle \mathbf{2}, \mathbf{4}\rangle)$
② $(\mathbf{3}, \langle \mathbf{2}, \mathbf{4}, 6, \mathbf{2}\rangle, \langle \mathbf{2}, \mathbf{4}\rangle)$
③ $(\mathbf{3}, \langle \mathbf{4}, \mathbf{4}, 5, \mathbf{2}\rangle, \langle \mathbf{2}, \mathbf{4}\rangle)$
④ $(\mathbf{3}, \langle \mathbf{4}, \mathbf{4}, 6, \mathbf{2}\rangle, \langle \mathbf{2}, \mathbf{4}\rangle)$

Figure 5.59: All solutions corresponding to the non ground example of the AMONG constraint of the **All solutions** slot, where the number of variables assigned a value from $\{\mathbf{2}, \mathbf{4}\}$ is equal to NVAR $= \mathbf{3}$

Typical

$\mathtt{NVAR} > 0$
$\mathtt{NVAR} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$
$|\mathtt{VALUES}| > 1$
$|\mathtt{VARIABLES}| > |\mathtt{VALUES}|$

Symmetries

- Items of VARIABLES are permutable.
- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).

Arg. properties

- Functional dependency: NVAR determined by VARIABLES and VALUES.
- Contractible wrt. VARIABLES when $\mathtt{NVAR} = 0$.
- Contractible wrt. VARIABLES when $\mathtt{NVAR} = |\mathtt{VARIABLES}|$.
- Aggregate: NVAR(+), VARIABLES(union), VALUES(sunion).

Remark

A similar constraint called BETWEEN was introduced in **CHIP** in 1990.

The COMMON constraint can be seen as a generalisation of the AMONG constraint where we allow the val attributes of the VALUES collection to be domain variables.

A generalisation of this constraint when the values of VALUES are not initially fixed is called AMONG_VAR.

When the variable NVAR (i.e., the first argument of the AMONG constraint) does not occur in any other constraints of the problem, it may be operationally more efficient to replace the AMONG constraint by an AMONG_LOW_UP constraint where NVAR is replaced by the corresponding interval $[\underline{\mathtt{NVAR}}, \overline{\mathtt{NVAR}}]$. This stands for two reasons:

- First, by using an AMONG_LOW_UP constraint rather than an AMONG constraint, we avoid the filtering algorithm related to NVAR.
- Second, unlike the AMONG constraint where we need to fix all its variables to get entailment, the AMONG_LOW_UP constraint can be entailed before all its variables get fixed. As a result, this potentially avoid unnecessary calls to its filtering algorithm.

It was shown in [116] that achieving bound-consistency for a conjunction of AMONG constraints where all sets of values are arbitrary intervals can be done in polynomial time.

Algorithm

A filtering algorithm achieving arc-consistency was given by Bessière *et al.* in [67, 70].

Systems

AMONG in **Choco**, COUNT in **Gecode**, AMONG in **JaCoP**, AMONG in **MiniZinc**.

See also

common keyword: ARITH, ATLEAST, ATMOST *(value constraint)*, COUNT *(counting constraint)*, COUNTS *(value constraint,counting constraint)*, DISCREPANCY, MAX_NVALUE, MIN_NVALUE, NVALUE *(counting constraint)*.

generalisation: AMONG_VAR *(*constant *replaced by* variable*)*.

implies: AMONG_VAR, CARDINALITY_ATMOST.

related: ROOTS *(can be used for expressing* AMONG*)*, SLIDING_CARD_SKIP0 *(counting constraint on maximal sequences)*.

shift of concept: AMONG_SEQ *(*`variable` *replaced by* `interval` *and constraint applied in a sliding way)*, COMMON.

soft variant: OPEN_AMONG *(open constraint)*.

specialisation: AMONG_DIFF_0 *(*`variable` $\in$ `values` *replaced by* `variable` *different from* 0*)*, AMONG_INTERVAL *(*`variable` $\in$ `values` *replaced by* `variable` $\in$ `interval`*)*, AMONG_LOW_UP *(*`variable` *replaced by* `interval`*)*, AMONG_MODULO *(list of* `values` *replaced by list of* `values` `v` *such that* `v` mod `QUOTIENT` = `REMAINDER`*)*, EXACTLY *(*`variable` *replaced by* `constant` *and* `values` *replaced by one single* `value`*)*.

system of constraints: GLOBAL_CARDINALITY *(count the number of occurrences of different values)*.

used in graph description: IN.

uses in its reformulation: COUNT.

Keywords

characteristic of a constraint: automaton, automaton with counters, non-deterministic automaton.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: alpha-acyclic constraint network(2), Berge-acyclic constraint network.

constraint type: value constraint, counting constraint.

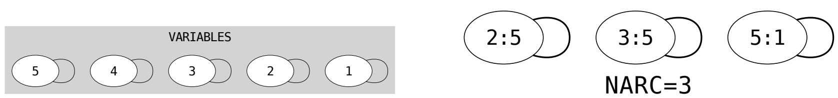
filtering: glue matrix, arc-consistency, SAT.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$SELF \mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	IN(variables.var, VALUES)
Graph property(ies)	**NARC** = NVAR

Graph model The arc constraint corresponds to the unary constraint IN(variables.var, VALUES) defined in this catalogue. Since this is a unary constraint we employ the $SELF$ arc generator in order to produce an initial graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.60 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.



(A) **(B)**

Figure 5.60: Initial and final graph of the AMONG constraint

Automaton Figure 5.61 depicts a first automaton that only accepts all the solutions to the AMONG constraint. This automaton uses a counter in order to record the number of satisfied constraints of the form $\text{VAR}_i \in \text{VALUES}$ already encountered. To each variable VAR_i of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links VAR_i and S_i: $\text{VAR}_i \in \text{VALUES} \Leftrightarrow S_i$. The automaton counts the number of variables of the VARIABLES collection that take their values in VALUES and finally assigns this number to NVAR.

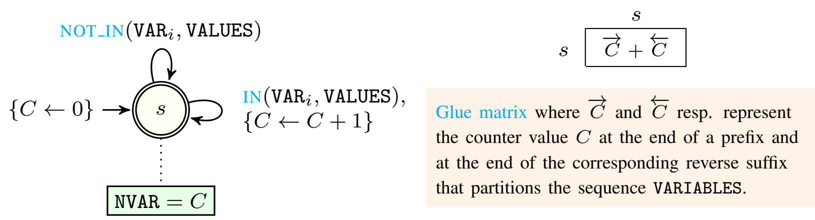


Figure 5.61: Automaton (with one counter) of the AMONG constraint and its glue matrix

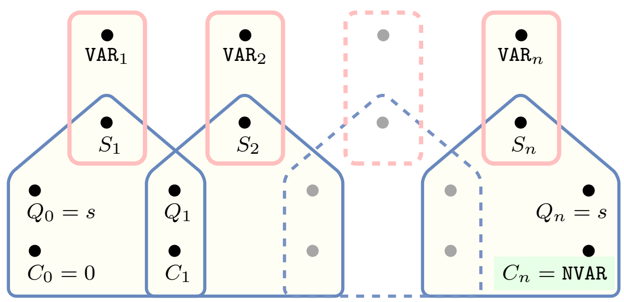


Figure 5.62: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the AMONG constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

We now describe a second counter free automaton that also only accepts all the solutions to AMONG constraint. Without loss of generality, assume that the collection of variables VARIABLES contains at least one variable (i.e., $|\text{VARIABLES}| \geq 1$). Let n and $\mathcal{D}$ respectively denote the number of variables of the collection VARIABLES, and the union of the domains of the variables of VARIABLES. Clearly, the maximum number of variables of VARIABLES that are assigned a value in VALUES cannot exceed the quantity $m = \min(n, \overline{\text{NVAR}})$. The $m+2$ states of the automaton that only accepts all the solutions to the AMONG constraint can be defined in the following way:

- We have an initial state labelled by s_0.
- We have m intermediate states labelled by s_i $(1 \leq i \leq m)$. The intermediate states are indexed by the number of already encountered satisfied constraints of the form $\text{VAR}_k \in \text{VALUES}$ from the initial state s_0 to the state s_i.

- We have an accepting state labelled by s_F.

Three classes of transitions are respectively defined in the following way:

1. There is a transition, labeled by j, ($j \in \mathcal{D} \setminus$ VALUES), from every state s_i, ($i \in [0, m]$), to itself.
2. There is a transition, labeled by j, ($j \in$ VALUES), from every state s_i, ($i \in [0, m-1]$), to the state s_{i+1}.
3. There is a transition, labelled by i, from every state s_i, ($i \in [0, m]$), to the accepting state s_F.

This leads to an automaton that has $m \cdot |\mathcal{D}| + |\mathcal{D} \setminus \texttt{VALUES}| + m + 1$ transitions. Since the maximum value of m is equal to n, in the worst case we have $n \cdot |\mathcal{D}| + |\mathcal{D} \setminus \texttt{VALUES}| + n + 1$ transitions.

Figure 5.63 depicts a counter free non deterministic automaton associated with the AMONG constraint under the hypothesis that (1) all variables of VARIABLES are assigned a value in $\{0, 1, 2, 3\}$, (2) |VARIABLES| is equal to 3, (3) VALUES corresponds to odd values. The sequence $\texttt{VAR}_1, \texttt{VAR}_2, \ldots, \texttt{VAR}_{|\texttt{VARIABLES}|}, \texttt{NVAR}$ is passed to this automaton. A state s_i ($1 \leq i \leq 3$) represents the fact that i odd values were already encountered, while s_F represents the accepting state. A transition from s_i ($1 \leq i \leq 3$) to s_F is labelled by i and represents the fact that we can only go in the accepting state from a state that is compatible with the total number of odd values enforced by NVAR. Note that non determinism only occurs if there is a non-empty intersection between the set of potential values that can be assigned to the variables of VARIABLES and the potential value of the NVAR. While the counter free non deterministic automaton depicted by Figure 5.63 has 5 states and 18 transitions, its minimum-state deterministic counterpart shown in Figure 5.64 has 7 states and 23 transitions.

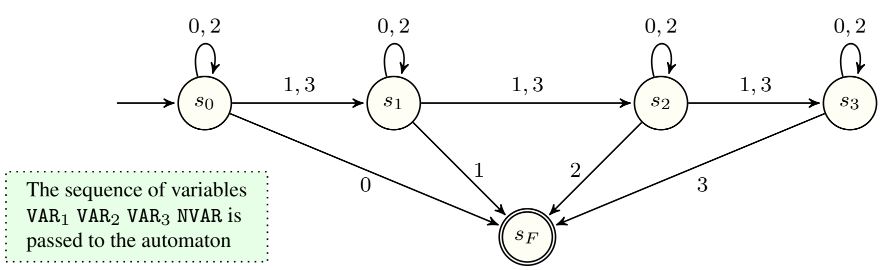


Figure 5.63: Counter free non deterministic automaton of the $\textsc{among}(\texttt{NVAR}, \langle \texttt{VAR}_1, \texttt{VAR}_2, \texttt{VAR}_3 \rangle, \langle 1, 3 \rangle)$ constraint assuming $\texttt{VAR}_i \in [0, 3]$ $(1 \leq i \leq 3)$, with initial state s_0 and accepting state s_F

We make the following final observation. Since the **Symmetries** slot of the AMONG constraint indicates that the variables of VARIABLES are permutable, and since all incoming transitions to any state of the automaton depicted by Figure 5.63 are labelled with distinct values, we can mechanically construct from this automaton a counter free deterministic automaton that takes as input the sequence NVAR, $\texttt{VAR}_3$, $\texttt{VAR}_2$, $\texttt{VAR}_1$ rather than the sequence $\texttt{VAR}_1$, $\texttt{VAR}_2$, $\texttt{VAR}_3$, NVAR. This is achieved by respectively making s_F and s_0 the initial and the accepting state, and by reversing each transition.

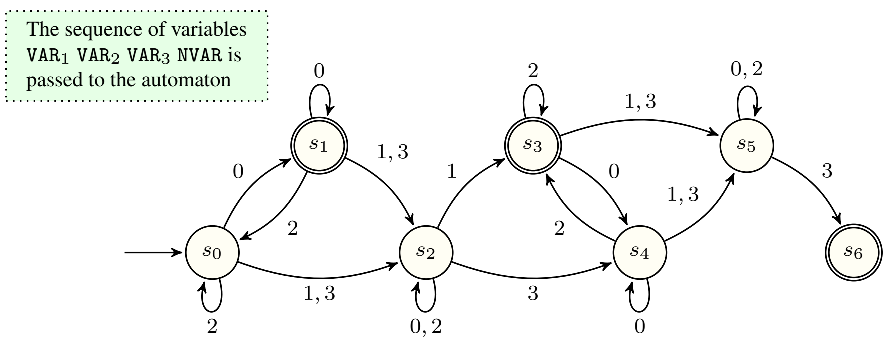


Figure 5.64: Counter free minimum-state deterministic automaton of the AMONG(NVAR, $\langle \mathtt{VAR}_1, \mathtt{VAR}_2, \mathtt{VAR}_3\rangle, \langle 1, 3\rangle$) constraint assuming $\mathtt{VAR}_i \in [0, 3]$ $(1 \leq i \leq 3)$, with initial state s_0 and accepting states s_1, s_3, s_6

5.26 AMONG_DIFF_0

► ▷ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin

Used in the automaton of NVALUE.

Constraint

AMONG_DIFF_0(NVAR, VARIABLES)

Arguments

NVAR : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\mathtt{NVAR} \geq 0$
$\mathtt{NVAR} \leq |\mathtt{VARIABLES}|$
required(VARIABLES, var)

Purpose

NVAR is the number of variables of the collection VARIABLES that take a value different from 0.

Example

$(3, \langle 0, 5, 5, 0, 1\rangle)$
$(0, \langle 0, 0, 0, 0, 0\rangle)$
$(1, \langle 0, 0, 0, 6, 0\rangle)$

The first AMONG_DIFF_0 constraint holds since exactly 3 values of the collection of values $\langle 0, 5, 5, 0, 1\rangle$ are different from 0.

All solutions

Figure 5.65 gives all solutions to the following non ground instance of the AMONG_DIFF_0 constraint: $\mathtt{V}_1 \in \{0, 3\}$, $\mathtt{V}_2 \in [0, 1]$, $\mathtt{V}_3 \in [5, 6]$, $\mathtt{V}_4 \in [0, 2]$, AMONG_DIFF_0$(\mathbf{2}, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4\rangle)$.

① $(\mathbf{2}, \langle 0, 0, \mathbf{5}, \mathbf{1}\rangle)$
② $(\mathbf{2}, \langle 0, 0, \mathbf{5}, \mathbf{2}\rangle)$
③ $(\mathbf{2}, \langle 0, 0, \mathbf{6}, \mathbf{1}\rangle)$
④ $(\mathbf{2}, \langle 0, 0, \mathbf{6}, \mathbf{2}\rangle)$

⑤ $(\mathbf{2}, \langle 0, \mathbf{1}, \mathbf{5}, 0\rangle)$
⑥ $(\mathbf{2}, \langle 0, \mathbf{1}, \mathbf{6}, 0\rangle)$
⑦ $(\mathbf{2}, \langle \mathbf{3}, 0, \mathbf{5}, 0\rangle)$
⑧ $(\mathbf{2}, \langle \mathbf{3}, 0, \mathbf{6}, 0\rangle)$

Figure 5.65: All solutions corresponding to the non ground example of the AMONG_DIFF_0 constraint of the **All solutions** slot, where the number of variables assigned a value different from zero is equal to $\mathtt{NVAR} = \mathbf{2}$

Typical

$\mathtt{NVAR} > 0$
$\mathtt{NVAR} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$
ATLEAST(1, VARIABLES, 0)
$2 * $ AMONG_DIFF_0(VARIABLES.var) $> |\mathtt{VARIABLES}|$

Typical model

ATLEAST$(2,$ VARIABLES$, 0)$

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that is different from 0 can be replaced by any other value that is also different from 0.

Arg. properties

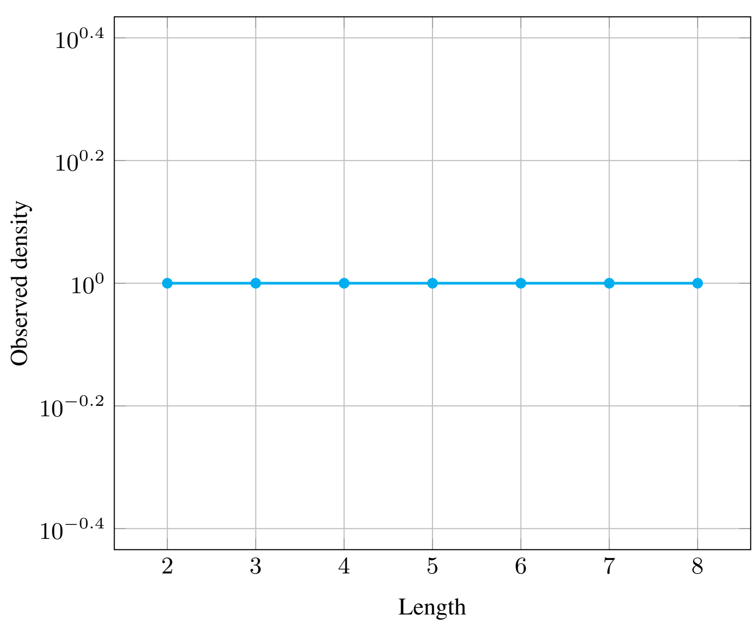
- Functional dependency: NVAR determined by VARIABLES.
- Contractible wrt. VARIABLES when NVAR $= 0$.
- Contractible wrt. VARIABLES when NVAR $= |$VARIABLES$|$.
- Aggregate: NVAR$(+)$, VARIABLES(union).

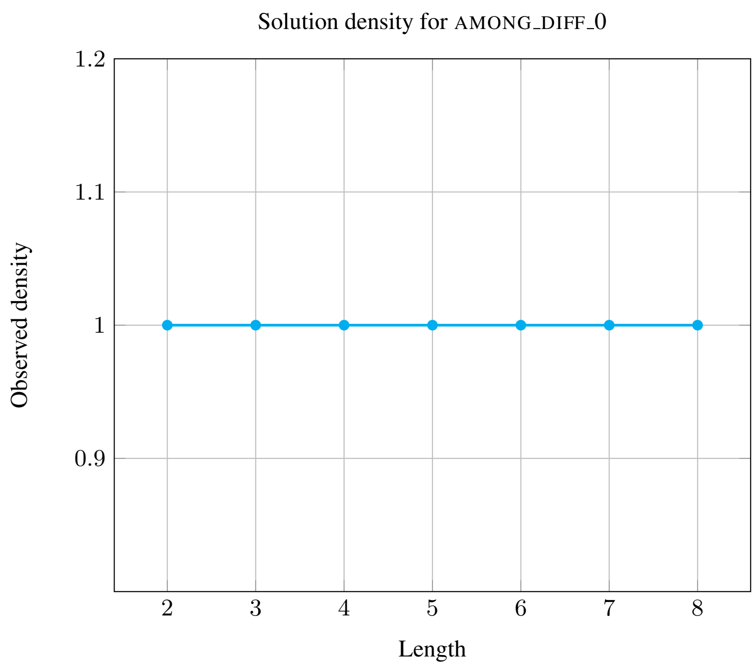
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for AMONG_DIFF_0: domains $0..n$

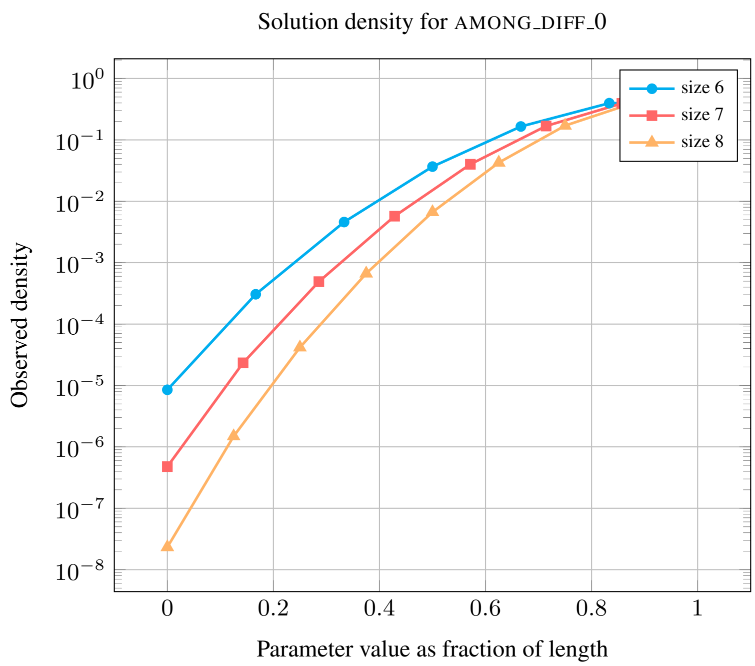
Solution density for AMONG_DIFF_0

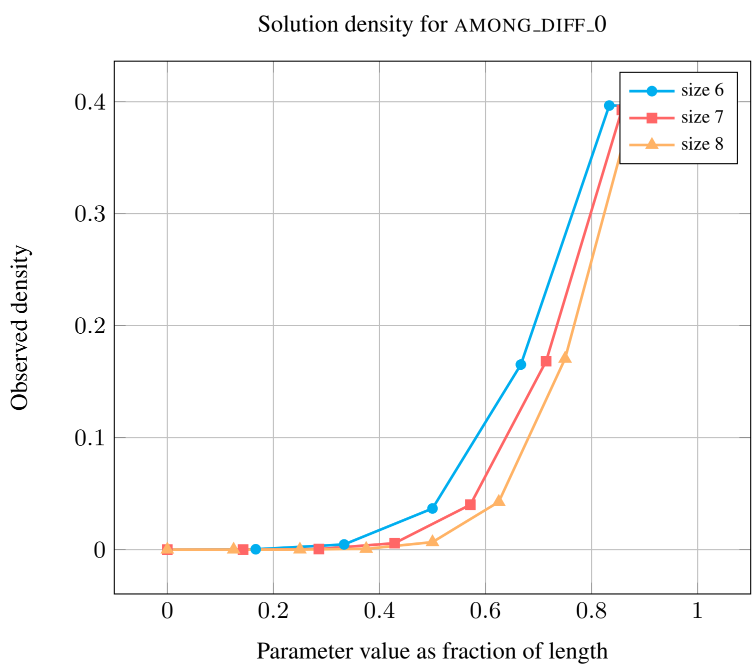




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	1	1	1	1	1	1	1
	1	4	9	16	25	36	49	64
	2	4	27	96	250	540	1029	1792
	3	-	27	256	1250	4320	12005	28672
	4	-	-	256	3125	19440	84035	286720
	5	-	-	-	3125	46656	352947	1835008
	6	-	-	-	-	46656	823543	7340032
	7	-	-	-	-	-	823543	16777216
	8	-	-	-	-	-	-	16777216

Solution count for AMONG_DIFF_0: domains $0..n$





See also

common keyword: NVALUE *(counting constraint)*.

generalisation: AMONG (`variable` $\neq 0$ *replaced by* `variable` $\in$ `values`).

Keywords

characteristic of a constraint: joker value, automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

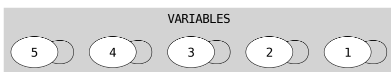
filtering: arc-consistency.

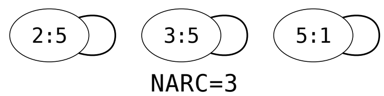
modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var $\neq 0$
Graph property(ies)	**NARC** = NVAR

Graph model Since this is a unary constraint we employ the *SELF* arc generator in order to produce an initial graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.66 respectively show the initial and final graph associated with first example of the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.





(A) (B)

Figure 5.66: Initial and final graph of the AMONG_DIFF_0 constraint

Automaton Figure 5.67 depicts the automaton associated with the AMONG_DIFF_0 constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i \neq 0 \Leftrightarrow S_i$. The automaton counts the number of variables of the VARIABLES collection that take a value different from 0 and finally assigns this number to NVAR.

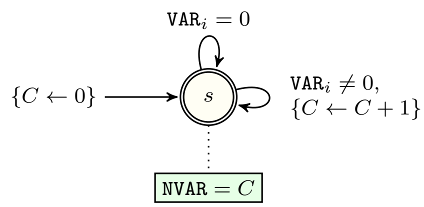


Figure 5.67: Automaton of the AMONG_DIFF_0 constraint

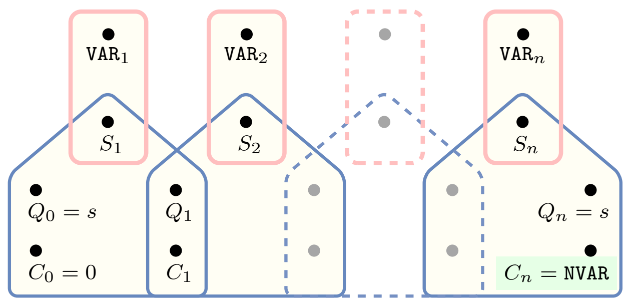


Figure 5.68: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the AMONG_DIFF_0 constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.27 AMONG_INTERVAL

► ▷ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin

Derived from AMONG.

Constraint

AMONG_INTERVAL(NVAR, VARIABLES, LOW, UP)

Arguments

NVAR	:	dvar
VARIABLES	:	collection(var−dvar)
LOW	:	int
UP	:	int

Restrictions

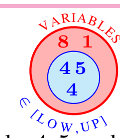
NVAR ≥ 0
NVAR $\leq$ |VARIABLES|
required(VARIABLES, var)
LOW $\leq$ UP

Purpose

NVAR is the number of variables of the collection VARIABLES taking a value that is located within interval [LOW, UP].

Example

$(3, \langle 4, 5, 8, 4, 1\rangle, 3, 5)$



The AMONG_INTERVAL constraint holds since we have 3 values, namely 4, 5 and 4 that are situated within interval [3, 5].

All solutions

Figure 5.69 gives all solutions to the following non ground instance of the AMONG_INTERVAL constraint: $V_1 \in [2, 9]$, $V_2 \in [0, 1]$, $V_3 \in [5, 6]$, $V_4 \in [1, 2]$, AMONG_INTERVAL$(\mathbf{3}, \langle V_1, V_2, V_3, V_4\rangle, \mathbf{0}, \mathbf{2})$.

① $(\mathbf{3}, \langle \mathbf{2}, \mathbf{0}, 5, \mathbf{1}\rangle, \mathbf{0}, \mathbf{2})$
② $(\mathbf{3}, \langle \mathbf{2}, \mathbf{0}, 5, \mathbf{2}\rangle, \mathbf{0}, \mathbf{2})$
③ $(\mathbf{3}, \langle \mathbf{2}, \mathbf{0}, 6, \mathbf{1}\rangle, \mathbf{0}, \mathbf{2})$
④ $(\mathbf{3}, \langle \mathbf{2}, \mathbf{0}, 6, \mathbf{2}\rangle, \mathbf{0}, \mathbf{2})$
⑤ $(\mathbf{3}, \langle \mathbf{2}, \mathbf{1}, 5, \mathbf{1}\rangle, \mathbf{0}, \mathbf{2})$
⑥ $(\mathbf{3}, \langle \mathbf{2}, \mathbf{1}, 5, \mathbf{2}\rangle, \mathbf{0}, \mathbf{2})$
⑦ $(\mathbf{3}, \langle \mathbf{2}, \mathbf{1}, 6, \mathbf{1}\rangle, \mathbf{0}, \mathbf{2})$
⑧ $(\mathbf{3}, \langle \mathbf{2}, \mathbf{1}, 6, \mathbf{2}\rangle, \mathbf{0}, \mathbf{2})$

Figure 5.69: All solutions corresponding to the non ground example of the AMONG_INTERVAL constraint of the **All solutions** slot, where the number of variables assigned a value in [LOW $= \mathbf{0}$, UP $= \mathbf{2}$] is equal to NVAR $= \mathbf{3}$

Typical

$\texttt{NVAR} > 0$
$\texttt{NVAR} < |\texttt{VARIABLES}|$
$|\texttt{VARIABLES}| > 1$
$\texttt{LOW} < \texttt{UP}$
$\texttt{LOW} \leq \texttt{maxval}(\texttt{VARIABLES.var})$
$\texttt{UP} \geq \texttt{minval}(\texttt{VARIABLES.var})$

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to [LOW, UP] (resp. does not belong to [LOW, UP]) can be replaced by any other value in [LOW, UP]) (resp. not in [LOW, UP]).

Arg. properties

- Functional dependency: NVAR determined by VARIABLES, LOW and UP.
- Contractible wrt. VARIABLES when $\texttt{NVAR} = 0$.
- Contractible wrt. VARIABLES when $\texttt{NVAR} = |\texttt{VARIABLES}|$.
- Aggregate: NVAR(+), VARIABLES(union), LOW(id), UP(id).

Remark

By giving explicitly all values of the interval [LOW, UP] the AMONG_INTERVAL constraint can be modelled with the AMONG constraint. However when $\texttt{LOW} - \texttt{UP} + 1$ is a large quantity the AMONG_INTERVAL constraint provides a more compact form.

See also

generalisation: AMONG *(*variable *in* interval *replaced by* variable $\in$ values*)*.

Keywords

characteristic of a constraint: automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

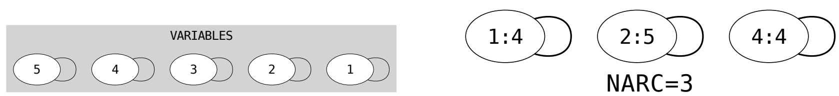
filtering: arc-consistency.

modelling: interval, functional dependency.

Arc input(s)	VARIABLES
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	• LOW $\leq$ variables.var • variables.var $\leq$ UP
Graph property(ies)	**NARC** = NVAR

Graph model The arc constraint corresponds to a unary constraint. For this reason we employ the *SELF* arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.70 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.



(A) **(B)**

Figure 5.70: Initial and final graph of the AMONG_INTERVAL constraint

Automaton Figure 5.71 depicts the automaton associated with the AMONG_INTERVAL constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{LOW} \leq \mathtt{VAR}_i \wedge \mathtt{VAR}_i \leq \mathtt{UP} \Leftrightarrow S_i$. The automaton counts the number of variables of the VARIABLES collection that take their values in [LOW, UP] and finally assigns this number to NVAR.

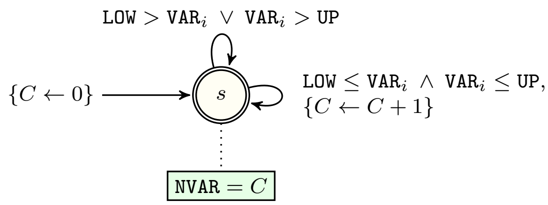


Figure 5.71: Automaton of the AMONG_INTERVAL constraint

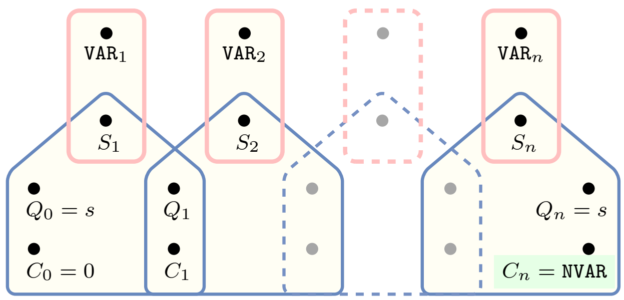


Figure 5.72: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the AMONG_INTERVAL constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.28 AMONG_LOW_UP

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [47]

Constraint AMONG_LOW_UP(LOW, UP, VARIABLES, VALUES)

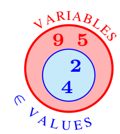
Arguments

LOW	:	int
UP	:	int
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int)

Restrictions

LOW $\geq$ 0
LOW $\leq$ |VARIABLES|
UP $\geq$ 0
UP $\leq$ |VARIABLES|
UP $\geq$ LOW
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose Between LOW and UP variables of the VARIABLES collection are assigned a value of the VALUES collection.



Example $(1, 2, \langle 9, 2, 4, 5\rangle, \langle 0, 2, 4, 6, 8\rangle)$

The AMONG_LOW_UP constraint holds since between 1 and 2 values (i.e., in fact 2 values) of the collection of values $\langle 9, 2, 4, 5\rangle$ belong to the set of values $\{0, 2, 4, 6, 8\}$.

All solutions Figure 5.73 gives all solutions to the following non ground instance of the AMONG_LOW_UP constraint: $V_1 \in [1, 2]$, $V_2 \in [8, 9]$, $V_3 \in [5, 6]$, $V_4 \in [2, 3]$, AMONG_LOW_UP(**3**, **4**, $\langle V_1, V_2, V_3, V_4\rangle$, $\langle$**0**, **2**, **4**, **6**, **8**$\rangle$).

Typical

LOW < |VARIABLES|
UP > 0
LOW < UP
|VARIABLES| > 1
|VALUES| > 1
|VARIABLES| > |VALUES|
LOW > 0 $\vee$ UP < |VARIABLES|

① $(\mathbf{3}, \mathbf{4}, \langle 1, \mathbf{8}, \mathbf{6}, \mathbf{2}\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
② $(\mathbf{3}, \mathbf{4}, \langle \mathbf{2}, \mathbf{8}, 5, \mathbf{2}\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
③ $(\mathbf{3}, \mathbf{4}, \langle \mathbf{2}, \mathbf{8}, \mathbf{6}, \mathbf{2}\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
④ $(\mathbf{3}, \mathbf{4}, \langle \mathbf{2}, \mathbf{8}, \mathbf{6}, 3\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
⑤ $(\mathbf{3}, \mathbf{4}, \langle \mathbf{2}, 9, \mathbf{6}, \mathbf{2}\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$

Figure 5.73: All solutions corresponding to the non ground example of the AMONG_LOW_UP constraint of the **All solutions** slot, where at least three variables (LOW $= \mathbf{3}$, UP $= \mathbf{4}$) are assigned a value from $\{\mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\}$

Symmetries

- Items of VARIABLES are permutable.
- Items of VALUES are permutable.
- LOW can be decreased to any value ≥ 0.
- UP can be increased to any value $\leq$ |VARIABLES|.
- An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).

Arg. properties

- Contractible wrt. VARIABLES when UP $= 0$.
- Contractible wrt. VARIABLES when UP $=$ |VARIABLES|.
- Aggregate: LOW$(+)$, UP$(+)$, VARIABLES(union), VALUES(sunion).

Algorithm

The AMONG_LOW_UP constraint is entailed if and only if the following two conditions hold:

1. The number of variables of the VARIABLES collection assigned a value of the VALUES collection is greater than or equal to LOW.
2. The number of variables of the VARIABLES collection that can potentially be assigned a value of the VALUES collection is less than or equal to UP.

Used in

AMONG_SEQ, CYCLE_CARD_ON_PATH, INTERVAL_AND_COUNT, SLIDING_CARD_SKIP0.

See also

assignment dimension added: INTERVAL_AND_COUNT *(assignment dimension corresponding to intervals added)*.

generalisation: AMONG (interval *replaced by* variable), SLIDING_CARD_SKIP0 *(full sequence replaced by maximal sequences of non-zeros)*.

system of constraints: AMONG_SEQ.

Keywords

characteristic of a constraint: automaton, automaton with counters.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

filtering: arc-consistency, entailment.

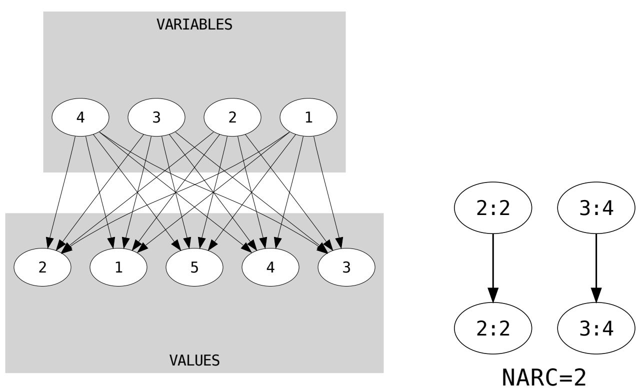
final graph structure: acyclic, bipartite, no loop.

Cond. implications AMONG_LOW_UP(LOW, UP, VARIABLES, VALUES)
with distinct(VARIABLES, var)
implies AMONG_LOW_UP(LOW, UP, VALUES, VARIABLES).

Arc input(s)	VARIABLES VALUES
Arc generator	$PRODUCT \mapsto$ collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	• **NARC** $\geq$ LOW • **NARC** $\leq$ UP
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Each arc constraint of the final graph corresponds to the fact that a variable is assigned to a value that belong to the VALUES collection. The two graph properties restrict the total number of arcs to the interval [LOW, UP].

Parts (A) and (B) of Figure 5.74 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.74: Initial and final graph of the AMONG_LOW_UP constraint

Automaton Figure 5.75 depicts the automaton associated with the AMONG_LOW_UP constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i \in \mathtt{VALUES} \Leftrightarrow S_i$. The automaton counts the number of variables of the VARIABLES collection that take their values in VALUES and finally checks that this number is within the interval $[\mathtt{LOW}, \mathtt{UP}]$.

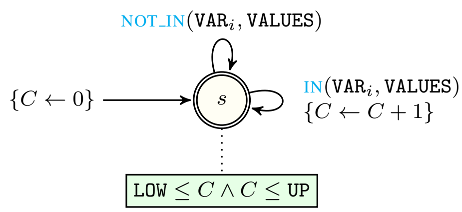


Figure 5.75: Automaton of the AMONG_LOW_UP constraint

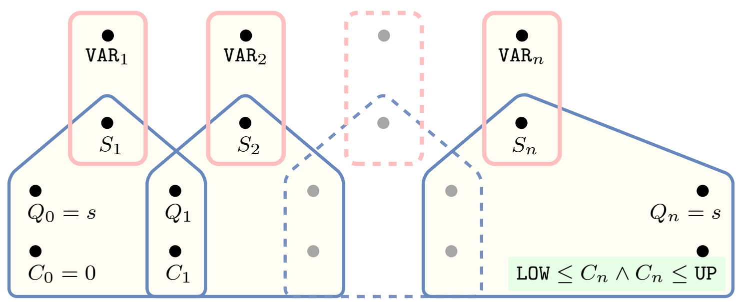


Figure 5.76: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the AMONG_LOW_UP constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.29 AMONG_MODULO

► ▷ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from AMONG.

Constraint AMONG_MODULO(NVAR, VARIABLES, REMAINDER, QUOTIENT)

Arguments

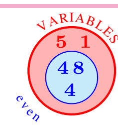
NVAR	:	dvar
VARIABLES	:	collection(var−dvar)
REMAINDER	:	int
QUOTIENT	:	int

Restrictions

$\mathtt{NVAR} \geq 0$
$\mathtt{NVAR} \leq |\mathtt{VARIABLES}|$
required(VARIABLES, var)
$\mathtt{REMAINDER} \geq 0$
$\mathtt{REMAINDER} < \mathtt{QUOTIENT}$
$\mathtt{QUOTIENT} > 0$

Purpose NVAR is the number of variables of the collection VARIABLES taking a value that is congruent to REMAINDER modulo QUOTIENT.

Example $(3, \langle 4, 5, 8, 4, 1\rangle, 0, 2)$



In this example $\mathtt{REMAINDER} = 0$ and $\mathtt{QUOTIENT} = 2$ specifies that we count the number of even values taken by the different variables. As a consequence the AMONG_MODULO constraint holds since exactly 3 values of the collection $\langle 4, 5, 8, 4, 1\rangle$ are even.

All solutions Figure 5.77 gives all solutions to the following non ground instance of the AMONG_MODULO constraint: $\mathtt{NVAR} \in [3, 4]$, $\mathtt{V}_1 \in [1, 2]$, $\mathtt{V}_2 \in [8, 9]$, $\mathtt{V}_3 \in [5, 6]$, $\mathtt{V}_4 \in [2, 3]$, AMONG_MODULO($\mathtt{NVAR}, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4\rangle, 1, 2$).

① $(3, \langle 1, 8, 5, 3\rangle, 1, 2)$
② $(3, \langle 1, 9, 5, 2\rangle, 1, 2)$
③ $(4, \langle 1, 9, 5, 3\rangle, 1, 2)$
④ $(3, \langle 1, 9, 6, 3\rangle, 1, 2)$
⑤ $(3, \langle 2, 9, 5, 3\rangle, 1, 2)$

Figure 5.77: All solutions corresponding to the non ground example of the AMONG_MODULO constraint of the **All solutions** slot, where the number of variables assigned an odd value ($\mathtt{REMAINDER} = 1$, $\mathtt{QUOTIENT} = 2$) is constrained to be equal to $\mathtt{NVAR} \in [3, 4]$

Typical

$\mathtt{NVAR} > 0$
$\mathtt{NVAR} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$
$\mathtt{QUOTIENT} > 1$
$\mathtt{QUOTIENT} <$ maxval$(\mathtt{VARIABLES.var})$

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value u of VARIABLES.var such that $u \bmod \mathtt{QUOTIENT} = \mathtt{REMAINDER}$ (resp. $u \bmod \mathtt{QUOTIENT} \neq \mathtt{REMAINDER}$) can be replaced by any other value v such that $v \bmod \mathtt{QUOTIENT} = \mathtt{REMAINDER}$ (resp. $u \bmod \mathtt{QUOTIENT} \neq \mathtt{REMAINDER}$).

Arg. properties

- Functional dependency: NVAR determined by VARIABLES, REMAINDER and QUOTIENT.
- Contractible wrt. VARIABLES when $\mathtt{NVAR} = 0$.
- Contractible wrt. VARIABLES when $\mathtt{NVAR} = |\mathtt{VARIABLES}|$.
- Aggregate: NVAR$(+)$, VARIABLES(union), REMAINDER(id), QUOTIENT(id).

Remark

By giving explicitly all values v that satisfy the equality $v \bmod \mathtt{QUOTIENT} = \mathtt{REMAINDER}$, the AMONG_MODULO constraint can be modelled with the AMONG constraint. However the AMONG_MODULO constraint provides a more compact form.

See also

generalisation: AMONG *(list of* values v *such that* v mod QUOTIENT = REMAINDER *replaced by list of* values*)*.

Keywords

characteristic of a constraint: modulo, automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

filtering: arc-consistency.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$SELF \mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var mod QUOTIENT = REMAINDER
Graph property(ies)	**NARC** = NVAR

Graph model The arc constraint corresponds to a unary constraint. For this reason we employ the $SELF$ arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.78 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

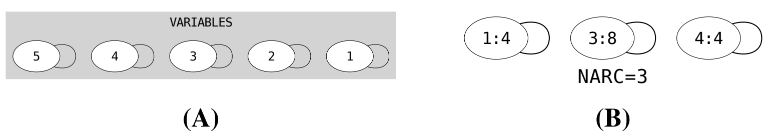


Figure 5.78: Initial and final graph of the AMONG_MODULO constraint

Automaton Figure 5.79 depicts the automaton associated with the AMONG_MODULO constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i \bmod \mathtt{QUOTIENT} = \mathtt{REMAINDER} \Leftrightarrow S_i$.

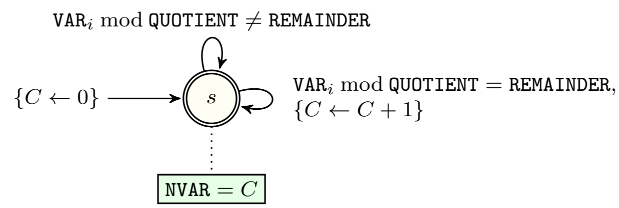


Figure 5.79: Automaton of the AMONG_MODULO constraint

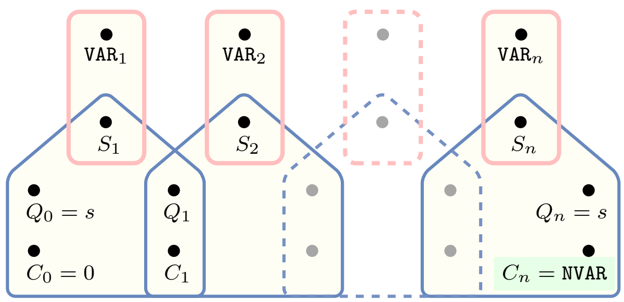


Figure 5.80: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the AMONG_MODULO constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.30 AMONG_SEQ

▶ ▷ ◀ DESCRIPTION LINKS GRAPH

Origin [47]

Constraint AMONG_SEQ(LOW, UP, SEQ, VARIABLES, VALUES)

Synonym SEQUENCE.

Arguments

LOW	:	int
UP	:	int
SEQ	:	int
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int)

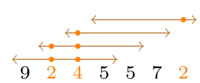
Restrictions

$\texttt{LOW} \geq 0$
$\texttt{LOW} \leq |\texttt{VARIABLES}|$
$\texttt{UP} \geq \texttt{LOW}$
$\texttt{SEQ} > 0$
$\texttt{SEQ} \geq \texttt{LOW}$
$\texttt{SEQ} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose Constrains all sequences of SEQ consecutive variables of the collection VARIABLES to take at least LOW values in VALUES and at most UP values in VALUES.

Example

$(1, 2, 4, \langle 9, 2, 4, 5, 5, 7, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$



The AMONG_SEQ constraint holds since the different sequences of 4 consecutive variables contains respectively 2, 2, 1 and 1 even numbers.

All solutions Figure 5.81 gives all solutions to the following non ground instance of the AMONG_SEQ constraint: $V_1 \in [1, 2]$, $V_2 \in [8, 9]$, $V_3 \in [5, 6]$, $V_4 \in [2, 3]$, AMONG_SEQ$(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle V_1, V_2, V_3, V_4\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$.

Typical

$\texttt{LOW} < \texttt{SEQ}$
$\texttt{UP} > 0$
$\texttt{SEQ} > 1$
$\texttt{SEQ} < |\texttt{VARIABLES}|$
$|\texttt{VARIABLES}| > 1$
$|\texttt{VALUES}| > 0$
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$
$\texttt{LOW} > 0 \lor \texttt{UP} < \texttt{SEQ}$

① $(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle 1, \mathbf{8}, 5, \mathbf{2}\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
② $(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle 1, \mathbf{8}, 5, 3\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
③ $(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle 1, 9, 5, \mathbf{2}\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
④ $(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle 1, 9, 5, 3\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
⑤ $(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle 1, 9, \mathbf{6}, 3\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
⑥ $(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle \mathbf{2}, 9, 5, \mathbf{2}\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
⑦ $(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle \mathbf{2}, 9, 5, 3\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$
⑧ $(\mathbf{0}, \mathbf{1}, \mathbf{2}, \langle \mathbf{2}, 9, \mathbf{6}, 3\rangle, \langle \mathbf{0}, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}\rangle)$

Figure 5.81: All solutions corresponding to the non ground example of the AMONG_SEQ constraint of the **All solutions** slot, where each sequence of two consecutive variables (`SEQ` $= \mathbf{2}$) does not contain more than one occurrence (`LOW` $= \mathbf{0}$, `UP` $= \mathbf{1}$) of values $\mathbf{0}$, $\mathbf{2}$, $\mathbf{4}$, $\mathbf{6}$, $\mathbf{8}$

Symmetries

- Items of `VARIABLES` can be reversed.
- Items of `VALUES` are permutable.
- `LOW` can be decreased to any value ≥ 0.
- `UP` can be increased to any value $\leq$ `SEQ`.
- An occurrence of a value of `VARIABLES.var` that belongs to `VALUES.val` (resp. does not belong to `VALUES.val`) can be replaced by any other value in `VALUES.val` (resp. not in `VALUES.val`).

Arg. properties

- Contractible wrt. `VARIABLES` when `UP` $= 0$.
- Contractible wrt. `VARIABLES` when `SEQ` $= 1$.
- Prefix-contractible wrt. `VARIABLES`.
- Suffix-contractible wrt. `VARIABLES`.

Usage

The AMONG_SEQ constraint occurs in many timetabling problems. As a typical example taken from [437], consider, for example, a nurse-rostering problem where each nurse can work at most 2 night shifts during every period of 7 consecutive days.

Algorithm

Beldiceanu and Carlsson [32] have proposed a first incomplete filtering algorithm for the AMONG_SEQ constraint. Later on, W.-J. van Hoeve *et al.* proposed two filtering algorithms [437] establishing arc-consistency as well as an incomplete filtering algorithm based on dynamic programming concepts. In 2007 Brand *et al.* came up with a reformulation [96] that provides a complete filtering algorithm. One year later, Maher *et al.* use a reformulation in term of a linear program [284] where (1) each coefficient is an integer in $\{-1, 0, 1\}$, (2) each column has a block of consecutive 1's or -1's. From this reformulation they derive a flow model that leads to an algorithm that achieves a complete filtering in $O(n^2)$ along a branch of the search tree.

Systems

SEQUENCE in **Gecode**, SEQUENCE in **JaCoP**.

See also

generalisation: SLIDING_DISTRIBUTION *(single set of values replaced by individual values)*.

part of system of constraints: AMONG_LOW_UP.

root concept: AMONG.

used in graph description: AMONG_LOW_UP.

Keywords

characteristic of a constraint: hypergraph.

combinatorial object: sequence.

constraint type: system of constraints, decomposition, sliding sequence constraint.

filtering: arc-consistency, linear programming, flow.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection
Arc arity	SEQ
Arc constraint(s)	AMONG_LOW_UP(LOW, UP, collection, VALUES)
Graph property(ies)	**NARC** $= \lvert\mathtt{VARIABLES}\rvert - \mathtt{SEQ} + 1$

Graph model A constraint on sliding sequences of consecutive variables. Each vertex of the graph corresponds to a variable. Since they link SEQ variables, the arcs of the graph correspond to hyperarcs. In order to link SEQ consecutive variables we use the arc generator *PATH*. The constraint associated with an arc corresponds to the AMONG_LOW_UP constraint defined at another entry of this catalogue.

Signature Since we use the *PATH* arc generator with an arity of SEQ on the items of the VARIABLES collection, the expression $\lvert\mathtt{VARIABLES}\rvert - \mathtt{SEQ} + 1$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property **NARC** $= \lvert\mathtt{VARIABLES}\rvert - \mathtt{SEQ} + 1$ to **NARC** $\geq \lvert\mathtt{VARIABLES}\rvert - \mathtt{SEQ} + 1$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

5.31 AMONG_VAR

▶ ▷ ◀ DESCRIPTION LINKS GRAPH

Origin Generalisation of AMONG

Constraint AMONG_VAR(NVAR, VARIABLES, VALUES)

Arguments

NVAR	:	dvar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−dvar)

Restrictions

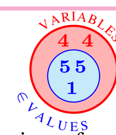
NVAR ≥ 0
NVAR $\leq$ |VARIABLES|
required(VARIABLES, var)
required(VALUES, val)

Purpose

NVAR is the number of variables of the collection VARIABLES that are equal to one of the variables of the collection VALUES.

Example

$(3, \langle 4, 5, 5, 4, 1\rangle, \langle 1, 5, 8, 1\rangle)$



The AMONG_VAR constraint holds since exactly 3 values of the collection of variables $\langle 4, 5, 5, 4, 1\rangle$ occurs within the collection $\langle 1, 5, 8, 1\rangle$.

All solutions

Figure 5.82 gives all solutions to the following non ground instance of the AMONG_VAR constraint: NVAR $\in [3, 4]$, $V_1 \in [1, 2]$, $V_2 \in [8, 9]$, $V_3 \in [5, 6]$, $V_4 \in [2, 3]$, AMONG_VAR(NVAR, $\langle V_1, V_2, V_3, V_4\rangle$, $\langle 0, 2, 4, 6, 8\rangle$).

① $(3, \langle 1, 8, 6, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$
② $(3, \langle 2, 8, 5, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$
③ $(4, \langle 2, 8, 6, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$
④ $(3, \langle 2, 8, 6, 3\rangle, \langle 0, 2, 4, 6, 8\rangle)$
⑤ $(3, \langle 2, 9, 6, 2\rangle, \langle 0, 2, 4, 6, 8\rangle)$

Figure 5.82: All solutions corresponding to the non ground example of the AMONG_VAR constraint of the **All solutions** slot, where the number of variables assigned a value in $\{0, 2, 4, 6, 8\}$ is equal to NVAR $\in [3, 4]$

Typical

|VARIABLES| > 1
|VALUES| > 1
|VARIABLES| $>$ |VALUES|

Symmetries

- Items of VARIABLES are permutable.
- Items of VALUES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.
- An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).

Arg. properties

- Functional dependency: NVAR determined by VARIABLES and VALUES.
- Contractible wrt. VARIABLES when $\text{NVAR} = 0$.
- Contractible wrt. VARIABLES when $\text{NVAR} = |\text{VARIABLES}|$.
- Aggregate: NVAR(+), VARIABLES(union), VALUES(union).

Systems

AMONG in **Choco**, COUNT in **Gecode**, AMONGVAR in **JaCoP**.

See also

implied by: AMONG.

related: COMMON.

specialisation: AMONG (variable *replaced by* constant *within list of* values VALUES).

uses in its reformulation: MIN_N.

Keywords

constraint arguments: pure functional dependency.

constraint type: counting constraint.

final graph structure: acyclic, bipartite, no loop.

modelling: functional dependency.

Arc input(s)	VARIABLES VALUES
Arc generator	$PRODUCT \mapsto$ collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	**NSOURCE** = NVAR
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.83 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** graph property, the source vertices of the final graph are stressed with a double circle. Since the final graph has only 3 sources the variables NVAR is fixed to 3.

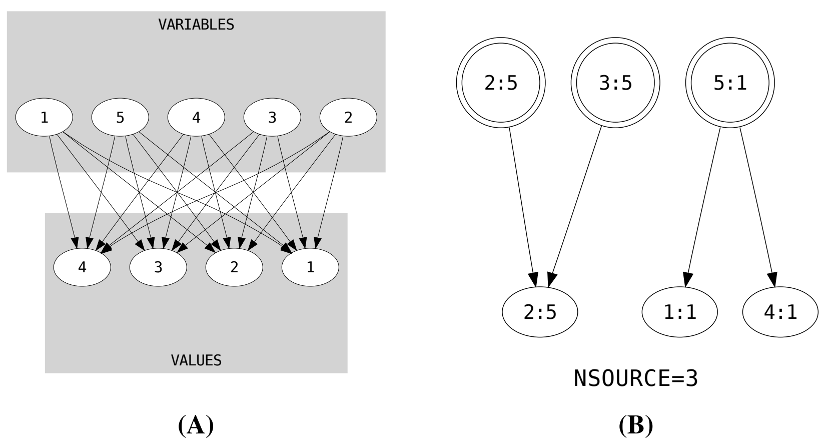


Figure 5.83: Initial and final graph of the AMONG_VAR constraint

5.32 AND

▶ ▷ ◀ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint AND(VAR, VARIABLES)

Synonym REL.

Arguments

VAR : dvar
VARIABLES : collection(var−dvar)

Restrictions

VAR ≥ 0
VAR ≤ 1
$|$VARIABLES$| \geq 2$
required(VARIABLES, var)
VARIABLES.var ≥ 0
VARIABLES.var ≤ 1

Purpose Let VARIABLES be a collection of 0-1 variables $\mathtt{VAR}_1, \mathtt{VAR}_2, \ldots, \mathtt{VAR}_n$ $(n \geq 2)$. Enforce $\mathtt{VAR} = \mathtt{VAR}_1 \wedge \mathtt{VAR}_2 \wedge \cdots \wedge \mathtt{VAR}_n$.

Example

$(0, \langle 0, 0\rangle)$
$(0, \langle 0, 1\rangle)$
$(0, \langle 1, 0\rangle)$
$(1, \langle 1, 1\rangle)$
$(0, \langle 1, 0, 1\rangle)$

All solutions Figure 5.84 gives all solutions to the following non ground instance of the AND constraint: $\mathtt{VAR} \in [0, 1]$, $\mathtt{V}_1 \in [0, 1]$, $\mathtt{V}_2 = 1$, $\mathtt{V}_3 \in [0, 1]$, $\mathtt{V}_4 = 1$, AND$(\mathtt{VAR}, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4\rangle)$.

① $(\mathbf{0}, \langle 0, 1, 0, 1\rangle)$
② $(\mathbf{0}, \langle 0, 1, 1, 1\rangle)$
③ $(\mathbf{0}, \langle 1, 1, 0, 1\rangle)$
④ $(\mathbf{1}, \langle 1, 1, 1, 1\rangle)$

Figure 5.84: All solutions corresponding to the non ground example of the AND constraint of the **All solutions** slot

Symmetry Items of VARIABLES are permutable.

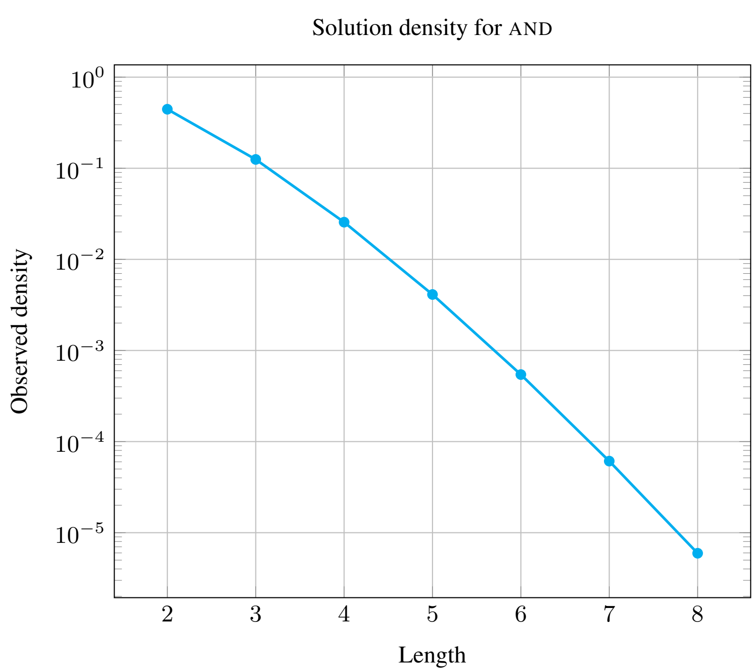
Arg. properties

- Functional dependency: VAR determined by VARIABLES.
- Extensible wrt. VARIABLES when VAR $= 0$.
- Aggregate: VAR($\wedge$), VARIABLES(union).

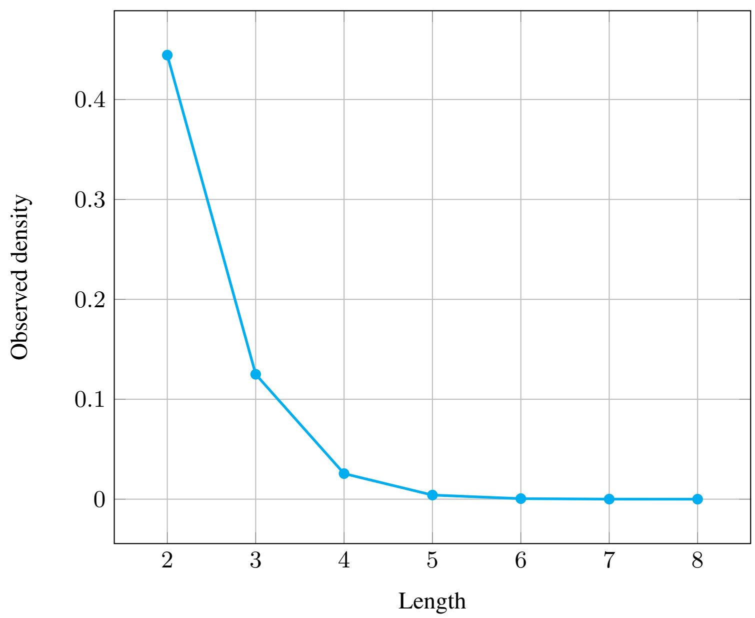
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	4	8	16	32	64	128	256

Number of solutions for AND: domains $0..n$



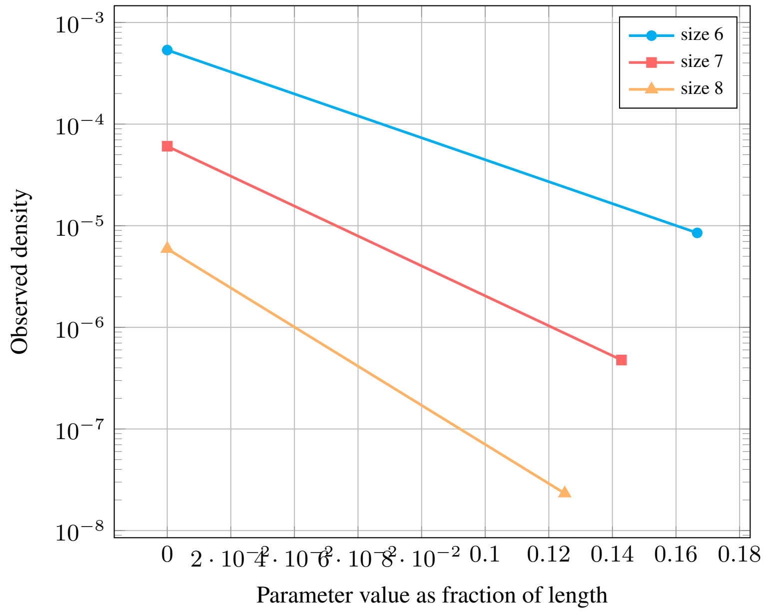
Solution density for AND



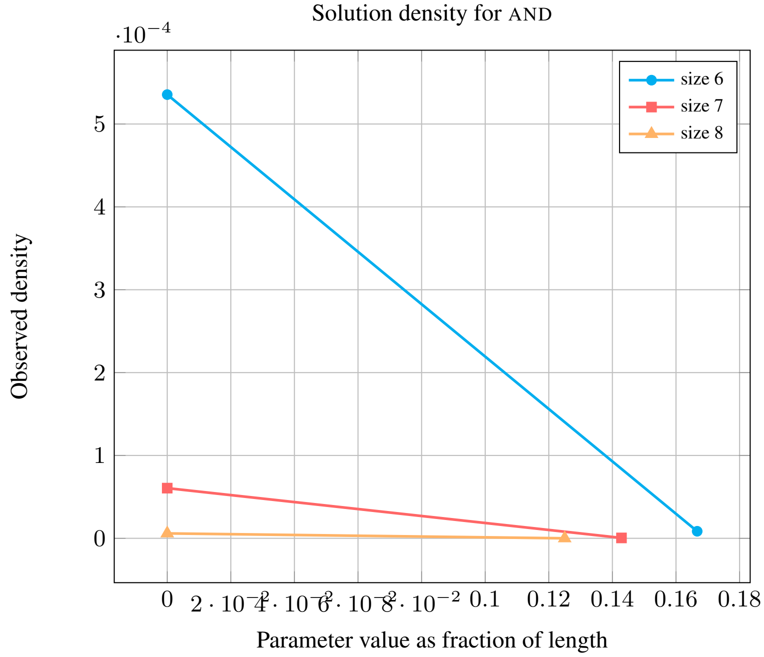
Length (n)		2	3	4	5	6	7	8
Total		4	8	16	32	64	128	256
Parameter value	0	3	7	15	31	63	127	255
	1	1	1	1	1	1	1	1

Solution count for AND: domains $0..n$

Solution density for AND



Solution density for AND



Systems REIFIEDAND in **Choco**, REL in **Gecode**, ANDBOOL in **JaCoP**, #/\ in **SICStus**.

See also **common keyword:** CLAUSE_AND, EQUIVALENT, IMPLY, NAND, NOR, OR, XOR (*Boolean* *constraint*).

implies: ATLEAST_NVALUE, BETWEEN_MIN_MAX, MINIMUM, SOFT_ALL_EQUAL_MIN_CTR.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: functional dependency.

Cond. implications

- AND($\mathtt{VAR}, \mathtt{VARIABLES}$)
  with $|\mathtt{VARIABLES}| > 2$
  implies SOME_EQUAL($\mathtt{VARIABLES}$).

- AND($\mathtt{VAR}, \mathtt{VARIABLES}$)
  with $\mathtt{VAR} = 0$
  implies NAND($\mathtt{VAR}, \mathtt{VARIABLES}$)
  when $\mathtt{VAR} = 1$.

- AND($\mathtt{VAR}, \mathtt{VARIABLES}$)
  with $\mathtt{VAR} = 1$
  implies NAND($\mathtt{VAR}, \mathtt{VARIABLES}$)
  when $\mathtt{VAR} = 0$.

Automaton Figure 5.85 depicts a first deterministic automaton without counter associated with the AND constraint. To the first argument VAR of the AND constraint corresponds the first signature variable. To each variable $\mathtt{VAR}_i$ of the second argument VARIABLES of the AND constraint corresponds the next signature variable. There is no signature constraint.

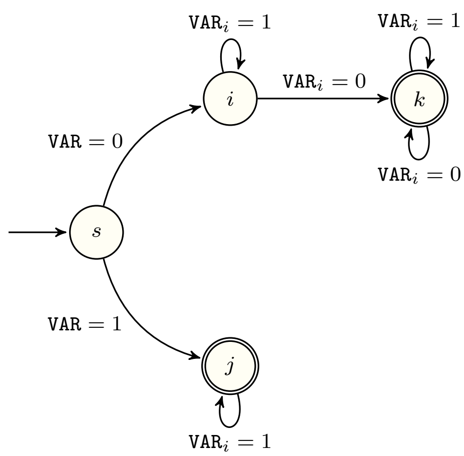


Figure 5.85: Counter free automaton of the AND($\mathtt{VAR}, \langle \mathtt{VAR}_1, \mathtt{VAR}_2, \ldots, \mathtt{VAR}_n \rangle$) constraint (the transition $i \xrightarrow{\mathtt{VAR}_i=0} k$ represents the fact that at least one variable $\mathtt{VAR}_i$ should be set to 0 when $\mathtt{VAR} = 0$, while the transition $j \xrightarrow{\mathtt{VAR}_i=1} j$ represents the fact that all $\mathtt{VAR}_i$ should be set to 1 when $\mathtt{VAR} = 1$)

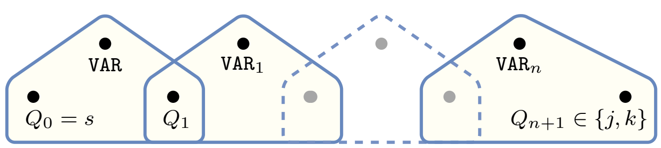


Figure 5.86: Hypergraph of the reformulation corresponding to the automaton of the AND constraint

Figure 5.87 depicts a second deterministic automaton with one counter associated with the AND constraint, where the argument VAR is unified to the final value of the counter.

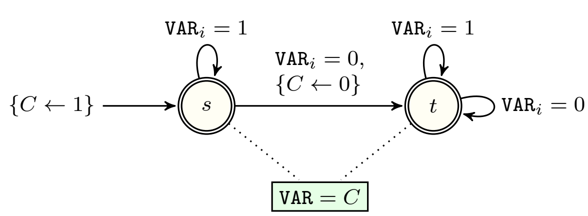


Figure 5.87: Automaton (with one counter) of the AND constraint

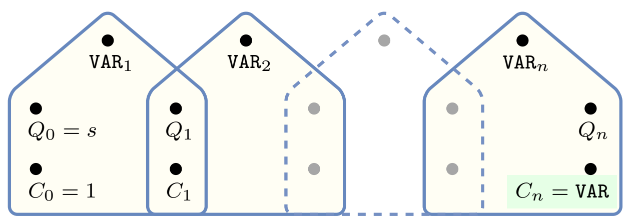


Figure 5.88: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the AND constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

5.33 ARITH

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Used in the definition of several automata

Constraint ARITH(VARIABLES, RELOP, VALUE)

Synonym REL.

Arguments

VARIABLES	:	collection(var−dvar)
RELOP	:	atom
VALUE	:	int

Restrictions

required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Enforce for all variables var of the VARIABLES collection to have var RELOP VALUE.

Example $(\langle 4, 5, 7, 4, 5\rangle, <, 9)$

The ARITH constraint holds since all values of the collection $\langle 4, 5, 7, 4, 5\rangle$ are strictly less than 9.

All solutions Figure 5.89 gives all solutions to the following non ground instance of the ARITH constraint: $V_1 \in [0, 5]$, $V_2 \in [2, 3]$, $V_3 \in [2, 4]$, $V_4 \in [1, 6]$, ARITH$(\langle V_1, V_2, V_3, V_4\rangle, \leq, 2)$.

① $(\langle 0, 2, 2, 1\rangle, 2)$
② $(\langle 0, 2, 2, 2\rangle, 2)$
③ $(\langle 1, 2, 2, 1\rangle, 2)$

④ $(\langle 1, 2, 2, 2\rangle, 2)$
⑤ $(\langle 2, 2, 2, 1\rangle, 2)$
⑥ $(\langle 2, 2, 2, 2\rangle, 2)$

Figure 5.89: All solutions corresponding to the non ground example of the ARITH constraint of the **All solutions** slot

Typical

$|$VARIABLES$| > 1$
RELOP $\in [=]$

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any value of VARIABLES.var.

Arg. properties Contractible wrt. VARIABLES.

Systems EQ in **Choco**, NEQ in **Choco**, GEQ in **Choco**, GT in **Choco**, LEQ in **Choco**, LT in **Choco**, REL in **Gecode**, #< in **SICStus**, #=< in **SICStus**, #> in **SICStus**, #>= in **SICStus**, #= in **SICStus**, #\= in **SICStus**.

Used in ARITH_SLIDING.

See also **common keyword:** AMONG, COUNT *(value constraint)*.

generalisation: ARITH_OR *(*`variable RELOP VALUE` *replaced by* `variable RELOP VALUE` $\vee$ `variable RELOP VALUE`*)*.

system of constraints: ARITH_SLIDING.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: decomposition, value constraint.

filtering: arc-consistency.

modelling: domain definition.

Cond. implications

ARITH(VARIABLES, RELOP, VALUE)
with RELOP $\in [<]$
and minval(VARIABLES.var) ≥ 0
implies RANGE_CTR(VARIABLES, CTR, R)
when CTR $\in [<]$.

Arc input(s)	VARIABLES
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var RELOP VALUE
Graph property(ies)	**NARC** = |VARIABLES|

Graph model Parts (A) and (B) of Figure 5.90 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

(A) (B)

Figure 5.90: Initial and final graph of the ARITH constraint

Automaton Figure 5.91 depicts the automaton associated with the ARITH constraint. To each variable $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\texttt{VAR}_i$ and S_i: $\texttt{VAR}_i$ RELOP VALUE $\Leftrightarrow S_i$. The automaton enforces for each variable $\texttt{VAR}_i$ the condition $\texttt{VAR}_i$ RELOP VALUE.

Figure 5.91: Automaton of the ARITH constraint

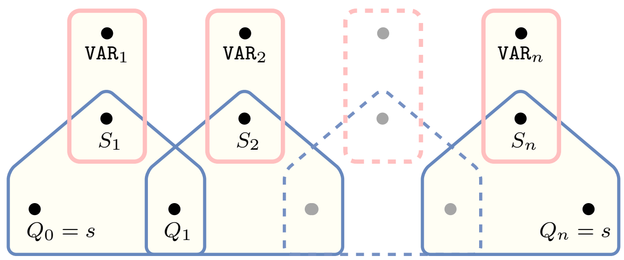


Figure 5.92: Hypergraph of the reformulation corresponding to the automaton of the ARITH constraint

5.34 ARITH_OR

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Used in the definition of several automata

Constraint ARITH_OR(VARIABLES1, VARIABLES2, RELOP, VALUE)

Arguments

VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
RELOP	:	atom
VALUE	:	int

Restrictions

required(VARIABLES1, var)
required(VARIABLES2, var)
$|\mathtt{VARIABLES1}| = |\mathtt{VARIABLES2}|$
$\mathtt{RELOP} \in [=, \neq, <, \geq, >, \leq]$

Purpose Enforce for all pairs of variables $\mathtt{var1}_i, \mathtt{var2}_i$ of the VARIABLES1 and VARIABLES2 collections to have $\mathtt{var1}_i$ RELOP VALUE $\vee$ $\mathtt{var2}_i$ RELOP VALUE.

Example $(\langle 0, 1, 0, 0, 1\rangle, \langle 0, 0, 0, 1, 0\rangle, =, 0)$

The constraint ARITH_OR holds since, for all pairs of variables $\mathtt{var1}_i, \mathtt{var2}_i$ of the VARIABLES1 and VARIABLES2 collections, there is at least one variable that is equal to 0.

All solutions Figure 5.93 gives all solutions to the following non ground instance of the ARITH_OR constraint: $\mathtt{U}_1 \in [3, 4]$, $\mathtt{U}_2 \in [1, 2]$, $\mathtt{U}_3 \in [1, 4]$, $\mathtt{V}_1 \in [2, 3]$, $\mathtt{V}_2 \in [2, 2]$, $\mathtt{V}_3 \in [0, 1]$, ARITH_OR$(\langle \mathtt{U}_1, \mathtt{U}_2, \mathtt{U}_3\rangle, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3\rangle, =, 2)$

① $(\langle 3, 1, 2\rangle, \langle 2, 2, 0\rangle, =, 2)$
② $(\langle 3, 1, 2\rangle, \langle 2, 2, 1\rangle, =, 2)$
③ $(\langle 3, 2, 2\rangle, \langle 2, 2, 0\rangle, =, 2)$
④ $(\langle 3, 2, 2\rangle, \langle 2, 2, 1\rangle, =, 2)$

⑤ $(\langle 4, 1, 2\rangle, \langle 2, 2, 0\rangle, =, 2)$
⑥ $(\langle 4, 1, 2\rangle, \langle 2, 2, 1\rangle, =, 2)$
⑦ $(\langle 4, 2, 2\rangle, \langle 2, 2, 0\rangle, =, 2)$
⑧ $(\langle 4, 2, 2\rangle, \langle 2, 2, 1\rangle, =, 2)$

Figure 5.93: All solutions corresponding to the non ground example of the ARITH_OR constraint of the **All solutions** slot

Typical

$|\mathtt{VARIABLES1}| > 0$
$\mathtt{RELOP} \in [=]$

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2) (RELOP) (VALUE).
- Items of VARIABLES1 and VARIABLES2 are permutable (*same permutation used*).

Arg. properties

Contractible wrt. `VARIABLES1` and `VARIABLES2` (*remove items from same position*).

See also

specialisation: ARITH *(*`variable` `RELOP` `VALUE` $\vee$ `variable` `RELOP` `VALUE` *replaced by* `variable` `RELOP` `VALUE`*)*.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: decomposition, value constraint.

filtering: arc-consistency.

final graph structure: acyclic, bipartite, no loop.

modelling: disjunction.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT(=) \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var RELOP VALUE $\vee$ variables2.var RELOP VALUE
Graph property(ies)	**NARC**$=$ |VARIABLES1|
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.94 respectively show the initial and final graphs associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

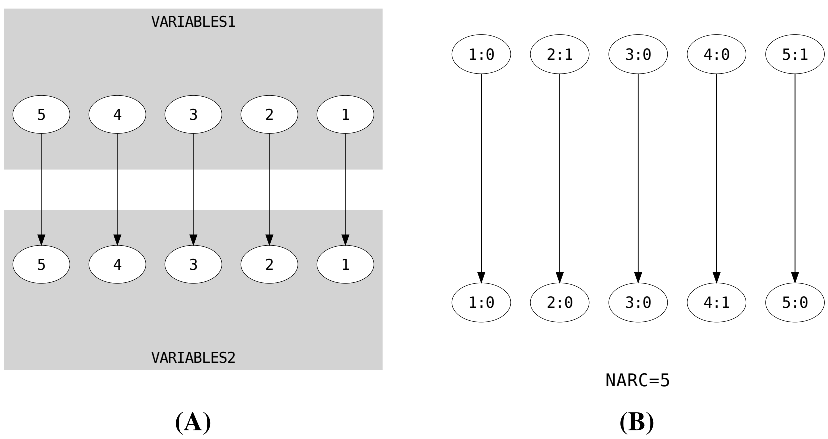


Figure 5.94: Initial and final graph of the ARITH_OR constraint

Automaton Figure 5.95 depicts the automaton associated with the ARITH_OR constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ be the i^{th} variables of the VARIABLES1 and VARIABLES2 collections. To each pair of variables $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable S_i. The following signature constraint links $\texttt{VAR1}_i$, $\texttt{VAR2}_i$ and S_i: $\texttt{VAR1}_i$ RELOP VALUE $\vee$ $\texttt{VAR2}_i$ RELOP VALUE $\Leftrightarrow S_i$. The automaton enforces for each pair of variables $\texttt{VAR1}_i$,$\texttt{VAR2}_i$ the condition $\texttt{VAR1}_i$ RELOP VALUE $\vee$ $\texttt{VAR2}_i$ RELOP VALUE.

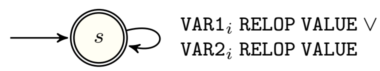


Figure 5.95: Automaton of the ARITH_OR constraint

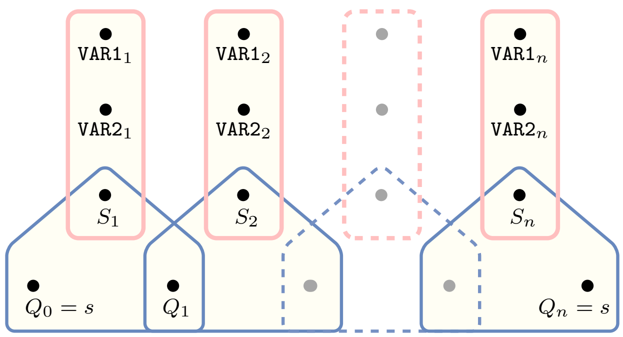


Figure 5.96: Hypergraph of the reformulation corresponding to the automaton of the ARITH_OR constraint

5.35 ARITH_SLIDING

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Used in the definition of some automaton

Constraint ARITH_SLIDING(VARIABLES, RELOP, VALUE)

Arguments

VARIABLES	:	collection(var−dvar)
RELOP	:	atom
VALUE	:	int

Restrictions

required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Enforce for all sequences of variables $\texttt{var}_1, \texttt{var}_2, \ldots, \texttt{var}_i$ $(1 \leq i \leq |\texttt{VARIABLES}|)$ of the VARIABLES collection to have $(\texttt{var}_1 + \texttt{var}_2 + \cdots + \texttt{var}_i)$ RELOP VALUE.

Example $(\langle 0, 0, 1, 2, 0, 0, -3 \rangle, <, 4)$

The ARITH_SLIDING constraint holds since all the following seven inequalities hold:

- $0 < 4$,
- $0 + 0 < 4$,
- $0 + 0 + 1 < 4$,
- $0 + 0 + 1 + 2 < 4$,
- $0 + 0 + 1 + 2 + 0 < 4$,
- $0 + 0 + 1 + 2 + 0 + 0 < 4$,
- $0 + 0 + 1 + 2 + 0 + 0 - 3 < 4$.

All solutions Figure 5.97 gives all solutions to the following non ground instance of the ARITH_SLIDING constraint: $\texttt{V}_1 \in [0, 5]$, $\texttt{V}_2 \in [2, 3]$, $\texttt{V}_3 \in [0, 4]$, ARITH_SLIDING$(\langle \texttt{V}_1, \texttt{V}_2, \texttt{V}_3 \rangle, \leq, 3)$.

① $(\langle 0, 2, 0 \rangle, \leq, 3)$
② $(\langle 0, 2, 1 \rangle, \leq, 3)$
③ $(\langle 0, 3, 0 \rangle, \leq, 3)$
④ $(\langle 1, 2, 0 \rangle, \leq, 3)$

Figure 5.97: All solutions corresponding to the non ground example of the ARITH_SLIDING constraint of the **All solutions** slot

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{RELOP} \in [<, \geq, >, \leq]$

Arg. properties

- Contractible wrt. VARIABLES when $\texttt{RELOP} \in [<, \leq]$ and $\texttt{minval}(\texttt{VARIABLES.var}) \geq 0$.
- Suffix-contractible wrt. VARIABLES.

See also

common keyword: SUM_CTR *(arithmetic constraint)*.

implies: SUM_CTR.

part of system of constraints: ARITH.

used in graph description: ARITH.

Keywords

characteristic of a constraint: hypergraph, automaton, automaton with counters.

combinatorial object: sequence.

constraint type: arithmetic constraint, decomposition, sliding sequence constraint.

Arc input(s)	VARIABLES
Arc generator	$PATH\_1 \mapsto$ collection
Arc arity	$*$
Arc constraint(s)	ARITH(collection, RELOP, VALUE)
Graph property(ies)	**NARC** $=$ |VARIABLES|

Automaton Figure 5.98 depicts the automaton associated with the ARITH_SLIDING constraint. To each item of the collection VARIABLES corresponds a signature variable S_i that is equal to 0.

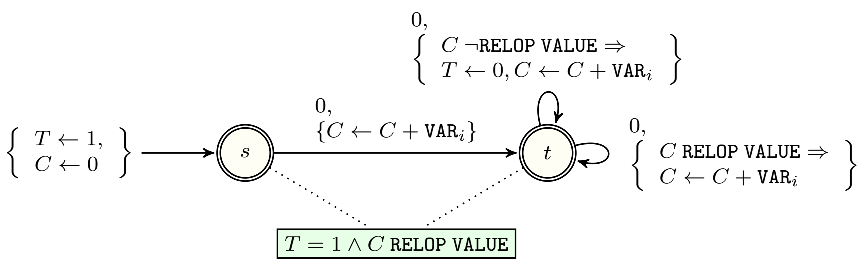


Figure 5.98: Automaton of the ARITH_SLIDING constraint (T is initially set to 1 and reset to 0 as soon as one of the sliding constraints does not hold)

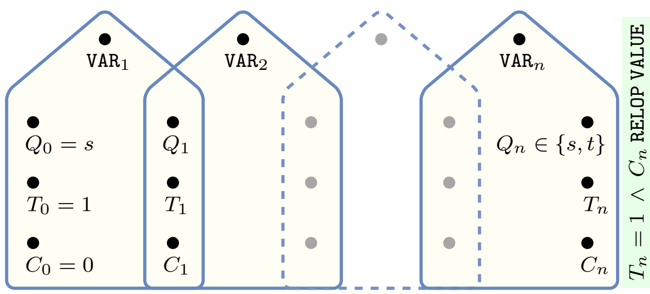


Figure 5.99: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the ARITH_SLIDING constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

5.36 ASSIGN_AND_COUNTS

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint ASSIGN_AND_COUNTS(COLOURS, ITEMS, RELOP, LIMIT)

Arguments

COLOURS	:	collection(val−int)
ITEMS	:	collection(bin−dvar, colour−dvar)
RELOP	:	atom
LIMIT	:	dvar

Restrictions

required(COLOURS, val)
distinct(COLOURS, val)
required(ITEMS, [bin, colour])
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Given several items (each of them having a specific colour that may not be initially fixed), and different bins, assign each item to a bin, so that the total number n of items of colour COLOURS in each bin satisfies the condition n RELOP LIMIT.

Example

$$\left(\begin{array}{l}\langle 4\rangle,\\ \left\langle\begin{array}{ll}\texttt{bin}-1 & \texttt{colour}-4,\\ \texttt{bin}-3 & \texttt{colour}-4,\\ \texttt{bin}-1 & \texttt{colour}-4,\\ \texttt{bin}-1 & \texttt{colour}-5\end{array}\right\rangle, \leq, 2\end{array}\right)$$

Figure 5.100 shows the solution associated with the example. The items and the bins are respectively represented by little squares and by the different columns. Each little square contains the value of the key attribute of the item to which it corresponds. The items for which the colour attribute is equal to 4 are located under the thick line.

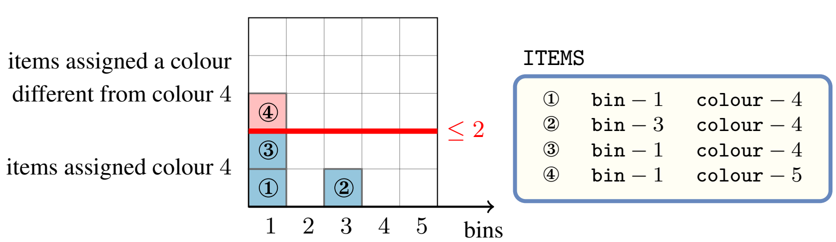


Figure 5.100: Assignment of the items to the bins

The ASSIGN_AND_COUNTS constraint holds since for each used bin (i.e., namely bins 1 and 3) the number of assigned items for which the colour attribute is equal to 4 is less than or equal to the limit 2.

All solutions

Figure 5.101 gives all solutions to the following non ground instance of the ASSIGN_AND_COUNTS constraint: $\mathtt{B}_1 \in [1,2]$, $\mathtt{B}_2 \in [2,3]$, $\mathtt{B}_3 \in [2,3]$, $\mathtt{B}_4 \in [3,4]$, $\mathtt{C}_1 \in [0,1]$, $\mathtt{C}_2 \in [0,1]$, $\mathtt{C}_3 \in [0,0]$, $\mathtt{C}_4 \in [0,1]$, ASSIGN_AND_COUNTS$(\langle 0\rangle, \langle \mathtt{B}_1\ \mathtt{C}_1, \mathtt{B}_2\ \mathtt{C}_2, \mathtt{B}_3\ \mathtt{C}_3, \mathtt{B}_4\ \mathtt{C}_4\rangle, \geq, 3)$.

① $(\langle 1\ 1,\ \mathbf{3}\ \mathbf{0},\ \mathbf{3}\ \mathbf{0},\ \mathbf{3}\ \mathbf{0}\rangle, \geq, \mathbf{3})$
② $(\langle \mathbf{2}\ \mathbf{0},\ \mathbf{2}\ \mathbf{0},\ \mathbf{2}\ \mathbf{0},\ 3\ 1\rangle, \geq, \mathbf{3})$
③ $(\langle \mathbf{2}\ \mathbf{0},\ \mathbf{2}\ \mathbf{0},\ \mathbf{2}\ \mathbf{0},\ 4\ 1\rangle, \geq, \mathbf{3})$
④ $(\langle 2\ 1,\ \mathbf{3}\ \mathbf{0},\ \mathbf{3}\ \mathbf{0},\ \mathbf{3}\ \mathbf{0}\rangle, \geq, \mathbf{3})$

Figure 5.101: All solutions corresponding to the non ground example of the ASSIGN_AND_COUNTS constraint of the **All solutions** slot, where items that are assigned colour **0** are shown in orange

Typical

$|\mathtt{COLOURS}| > 0$
$|\mathtt{ITEMS}| > 1$
$\mathtt{range}(\mathtt{ITEMS.bin}) > 1$
$\mathtt{RELOP} \in [<, \leq]$
$\mathtt{LIMIT} > 0$
$\mathtt{LIMIT} < |\mathtt{ITEMS}|$

Symmetries

- Items of COLOURS are permutable.
- Items of ITEMS are permutable.
- All occurrences of two distinct values of ITEMS.bin can be swapped; all occurrences of a value of ITEMS.bin can be renamed to any unused value.

Arg. properties

- Contractible wrt. ITEMS when $\mathtt{RELOP} \in [<, \leq]$.
- Extensible wrt. ITEMS when $\mathtt{RELOP} \in [\geq, >]$.

Usage

Some persons have pointed out that it is impossible to use constraints such as AMONG, ATLEAST, ATMOST, COUNT, or GLOBAL_CARDINALITY if the set of variables is not initially known. For example, this is required in practice for some timetabling problems.

See also

assignment dimension removed: COUNT, COUNTS.

used in graph description: COUNTS.

Keywords

application area: assignment.

characteristic of a constraint: coloured, automaton, automaton with array of counters, derived collection.

final graph structure: acyclic, bipartite, no loop.

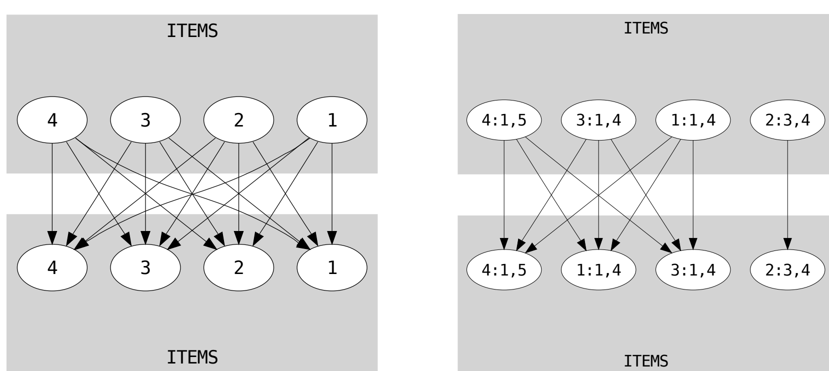
modelling: assignment dimension.

Derived Collection	$\mathtt{col}(\mathtt{VALUES}-\mathtt{collection}(\mathtt{val}-\mathtt{int}),[\mathtt{item}(\mathtt{val}-\mathtt{COLOURS.val})])$
Arc input(s)	ITEMS ITEMS
Arc generator	$PRODUCT \mapsto \mathtt{collection}(\mathtt{items1},\mathtt{items2})$
Arc arity	2
Arc constraint(s)	$\mathtt{items1.bin} = \mathtt{items2.bin}$
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP
Sets	$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\mathtt{source},\\ \mathtt{variables}-\mathtt{col}\left(\begin{array}{l}\mathtt{VARIABLES}-\mathtt{collection}(\mathtt{var}-\mathtt{dvar}),\\ {[\mathtt{item}(\mathtt{var}-\mathtt{ITEMS.colour})]}\end{array}\right)\end{array}\right]$
Constraint(s) on sets	$\mathrm{COUNTS}(\mathtt{VALUES},\mathtt{variables},\mathtt{RELOP},\mathtt{LIMIT})$

Graph model We enforce the COUNTS constraint on the colour of the items that are assigned to the same bin.

Parts (A) and (B) of Figure 5.102 respectively show the initial and final graph associated with the **Example** slot. The final graph consists of the following two connected components:

- The connected component containing six vertices corresponds to the items that are assigned to bin 1.
- The connected component containing two vertices corresponds to the items that are assigned to bin 3.



(A) **(B)**

Figure 5.102: Initial and final graph of the ASSIGN_AND_COUNTS constraint

The ASSIGN_AND_COUNTS constraint holds since for each set of successors of the vertices of the final graph no more than two items take colour 4.

Automaton Figure 5.103 depicts the automaton associated with the ASSIGN_AND_COUNTS constraint. To each colour attribute COLOUR_i of the collection ITEMS corresponds a 0-1 signature variable S_i. The following signature constraint links COLOUR_i and S_i: $\text{COLOUR}_i \in \text{COLOURS} \Leftrightarrow S_i$. For all items of the collection ITEMS for which the colour attribute takes its value in COLOURS, counts for each value assigned to the bin attribute its number of occurrences n, and finally imposes the condition n RELOP LIMIT.

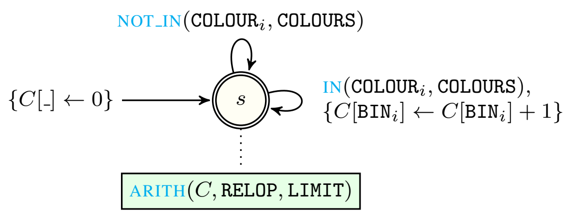


Figure 5.103: Automaton of the ASSIGN_AND_COUNTS constraint

5.37 ASSIGN_AND_NVALUES

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from ASSIGN_AND_COUNTS and NVALUES.

Constraint ASSIGN_AND_NVALUES(ITEMS, RELOP, LIMIT)

Arguments

ITEMS : collection(bin−dvar, value−dvar)
RELOP : atom
LIMIT : dvar

Restrictions

required(ITEMS, [bin, value])
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Given several items (each of them having a specific value that may not be initially fixed), and different bins, assign each item to a bin, so that the number n of distinct values in each bin satisfies the condition n RELOP LIMIT.

Example

$$\left(\left\langle\begin{array}{ll}\texttt{bin}-2 & \texttt{value}-3,\\ \texttt{bin}-1 & \texttt{value}-5,\\ \texttt{bin}-2 & \texttt{value}-3,\\ \texttt{bin}-2 & \texttt{value}-3,\\ \texttt{bin}-2 & \texttt{value}-4\end{array}\right\rangle, \leq, 2\right)$$

Figure 5.104 depicts the solution corresponding to the example.

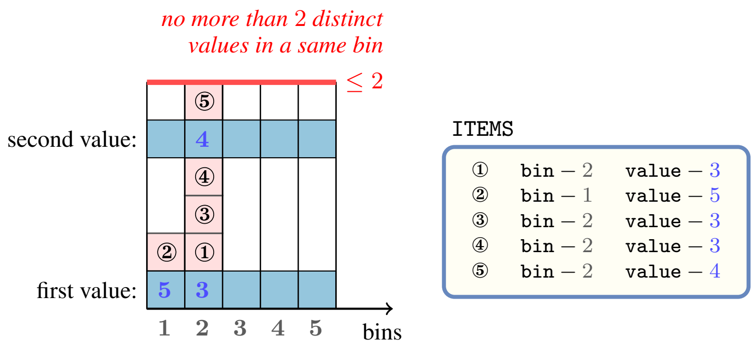


Figure 5.104: An assignment with at most two distinct values in parallel (values 3 and 4 in bin 2 and value 5 in bin 1)

The ASSIGN_AND_NVALUES constraint holds since for each used bin (i.e., namely bins 1 and 2) the number of distinct colours of the corresponding assigned items is less than or equal to the limit 2.

Typical

$|\texttt{ITEMS}| > 1$
$\texttt{range}(\texttt{ITEMS.bin}) > 1$
$\texttt{range}(\texttt{ITEMS.value}) > 1$
$\texttt{RELOP} \in [<, \leq]$
$\texttt{LIMIT} > 1$
$\texttt{LIMIT} < |\texttt{ITEMS}|$

Symmetries

- Items of ITEMS are permutable.
- All occurrences of two distinct values of ITEMS.bin can be swapped; all occurrences of a value of ITEMS.bin can be renamed to any unused value.

Arg. properties

- Contractible wrt. ITEMS when $\texttt{RELOP} \in [<, \leq]$.
- Extensible wrt. ITEMS when $\texttt{RELOP} \in [\geq, >]$.

Usage

Let us give two examples where the ASSIGN_AND_NVALUES constraint is useful:

- Quite often, in bin-packing problems, each item has a specific type, and one wants to assign items of similar type to each bin.
- In a vehicle routing problem, one wants to restrict the number of towns visited by each vehicle. Note that several customers may be located at the same town. In this example, each bin would correspond to a vehicle, each item would correspond to a visit to a customer, and the colour of an item would be the location of the corresponding customer.

See also

assignment dimension removed: NVALUE, NVALUES.

common keyword: NVALUES_EXCEPT_0 (*number of distinct values*).

related: ROOTS.

used in graph description: NVALUES.

Keywords

application area: assignment.

final graph structure: acyclic, bipartite, no loop.

modelling: assignment dimension, number of distinct values.

Arc input(s) ITEMS ITEMS

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{items1}, \texttt{items2})$

Arc arity 2

Arc constraint(s) $\texttt{items1.bin} = \texttt{items2.bin}$

Graph class
- ACYCLIC
- BIPARTITE
- NO_LOOP

Sets

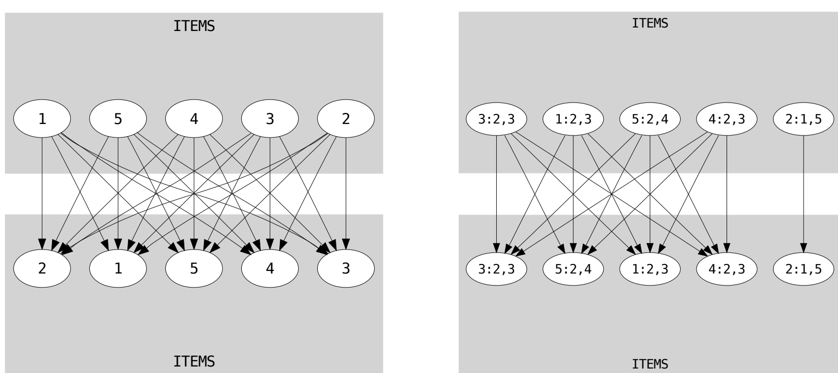
$$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source}, \\ \texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}), \\ [\texttt{item}(\texttt{var} - \texttt{ITEMS.value})]\end{array}\right)\end{array}\right]$$

Constraint(s) on sets NVALUES(variables, RELOP, LIMIT)

Graph model We enforce the NVALUES constraint on the items that are assigned to the same bin.

Parts (A) and (B) of Figure 5.105 respectively show the initial and final graph associated with the **Example** slot. The final graph consists of the following two connected components:

- The connected component containing 8 vertices corresponds to the items that are assigned to bin 2.
- The connected component containing 2 vertices corresponds to the items that are assigned to bin 1.



(A) **(B)**

Figure 5.105: Initial and final graph of the ASSIGN_AND_NVALUES constraint

The ASSIGN_AND_NVALUES constraint holds since for each set of successors of the vertices of the final graph no more than two distinct values are used:

- The unique item assigned to bin 1 uses value 5.

- Items assigned to bin 2 use values 3 and 4.

5.38 ATLEAST

► ▷ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint ATLEAST(N, VARIABLES, VALUE)

Synonym COUNT.

Arguments

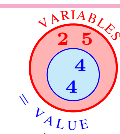
N	:	int
VARIABLES	:	collection(var−dvar)
VALUE	:	int

Restrictions

$\mathtt{N} \geq 0$
$\mathtt{N} \leq |\mathtt{VARIABLES}|$
required(VARIABLES, var)

Purpose At least N variables of the VARIABLES collection are assigned value VALUE.

Example $(2, \langle 4, 2, 4, 5\rangle, 4)$



The ATLEAST constraint holds since at least 2 values of the collection $\langle 4, 2, 4, 5\rangle$ are equal to value 4.

All solutions Figure 5.106 gives all solutions to the following non ground instance of the ATLEAST constraint: $\mathtt{V}_1 \in [3, 5]$, $\mathtt{V}_2 \in [1, 2]$, $\mathtt{V}_3 \in [5, 6]$, $\mathtt{V}_4 \in [7, 9]$, ATLEAST$(\mathbf{2}, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4\rangle, \mathbf{5})$.

① $(\mathbf{2}, \langle \mathbf{5}, 1, \mathbf{5}, 7\rangle, \mathbf{5})$
② $(\mathbf{2}, \langle \mathbf{5}, 1, \mathbf{5}, 8\rangle, \mathbf{5})$
③ $(\mathbf{2}, \langle \mathbf{5}, 1, \mathbf{5}, 9\rangle, \mathbf{5})$

④ $(\mathbf{2}, \langle \mathbf{5}, 2, \mathbf{5}, 7\rangle, \mathbf{5})$
⑤ $(\mathbf{2}, \langle \mathbf{5}, 2, \mathbf{5}, 8\rangle, \mathbf{5})$
⑥ $(\mathbf{2}, \langle \mathbf{5}, 2, \mathbf{5}, 9\rangle, \mathbf{5})$

Figure 5.106: All solutions corresponding to the non ground example of the ATLEAST constraint of the **All solutions** slot

Typical

$\mathtt{N} > 0$
$\mathtt{N} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$

Symmetries

- Items of VARIABLES are permutable.
- N can be decreased to any value ≥ 0.
- An occurrence of a value of VARIABLES.var that is different from VALUE can be replaced by any other value.

Arg. properties Extensible wrt. VARIABLES.

Systems OCCURENCEMIN in **Choco**, COUNT in **Gecode**, ATLEAST in **Gecode**, COUNT in **JaCoP**, AT_LEAST in **MiniZinc**, COUNT in **SICStus**.

Used in ALLDIFFERENT_EXCEPT_0, AMONG_DIFF_0, ATMOST, INT_VALUE_PRECEDE, ITH_POS_DIFFERENT_FROM_0, MINIMUM_EXCEPT_0, NVALUES_EXCEPT_0, PERIOD_EXCEPT_0, SLIDING_CARD_SKIP0, WEIGHTED_PARTIAL_ALLDIFF.

See also

common keyword: AMONG *(value constraint)*.

comparison swapped: ATMOST.

implied by: EXACTLY *($\geq$ N replaced by $=$ N)*.

related: ROOTS.

soft variant: OPEN_ATLEAST *(open constraint)*.

Keywords

characteristic of a constraint: automaton, automaton with counters.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint.

filtering: arc-consistency.

modelling: at least.

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUE
Graph property(ies)	**NARC** $\geq$ N

Graph model Since each arc constraint involves only one vertex (VALUE is fixed), we employ the *SELF* arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.107 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

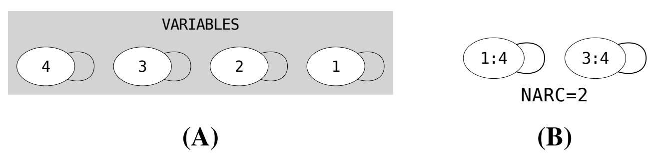


Figure 5.107: Initial and final graph of the ATLEAST constraint

Automaton Figure 5.108 depicts the automaton associated with the ATLEAST constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i = \mathtt{VALUE} \Leftrightarrow S_i$. The automaton counts the number of variables of the VARIABLES collection that are assigned value VALUE and finally checks that this number is greater than or equal to N.

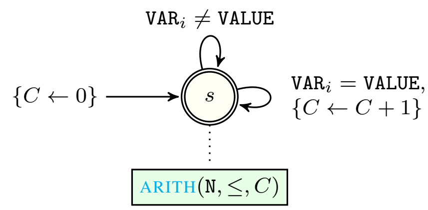


Figure 5.108: Automaton of the ATLEAST constraint

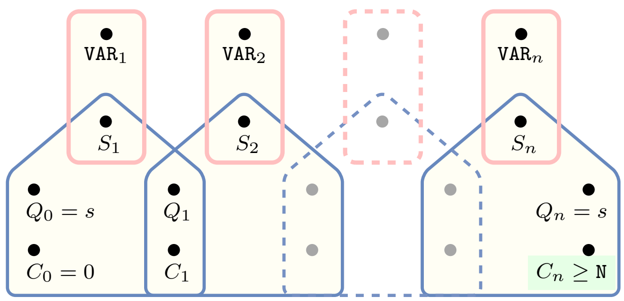


Figure 5.109: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the ATLEAST constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.39 ATLEAST_NVALUE

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin [352]

Constraint ATLEAST_NVALUE(NVAL, VARIABLES)

Synonym K_DIFF.

Arguments

NVAL : dvar
VARIABLES : collection(var−dvar)

Restrictions

required(VARIABLES, var)
$\texttt{NVAL} \geq 0$
$\texttt{NVAL} \leq |\texttt{VARIABLES}|$
$\texttt{NVAL} \leq$range(VARIABLES.var)

Purpose

The number of distinct values taken by the variables of the collection VARIABLES is greater than or equal to NVAL.

Example

$(2, \langle 3, 1, 7, 1, 6\rangle)$
$(4, \langle 3, 1, 7, 1, 6\rangle)$
$(5, \langle 3, 1, 7, 0, 6\rangle)$

The first ATLEAST_NVALUE constraint holds since the collection $\langle 3, 1, 7, 1, 6\rangle$ involves at least 2 distinct values (i.e., in fact 4 distinct values).

All solutions

Figure 5.110 gives all solutions to the following non ground instance of the ATLEAST_NVALUE constraint: $\texttt{NVAL} \in [3, 4]$, $\texttt{V}_1 \in [1, 2]$, $\texttt{V}_2 = 3$, $\texttt{V}_3 \in [3, 4]$, $\texttt{V}_4 \in [2, 3]$, ATLEAST_NVALUE(NVAL, $\langle \texttt{V}_1, \texttt{V}_2, \texttt{V}_3, \texttt{V}_4\rangle$).

① $(\mathbf{3}, \langle 1, 3, 3, 2\rangle)$
② $(\mathbf{3}, \langle 1, 3, 4, 2\rangle)$
③ $(\mathbf{4}, \langle 1, 3, 4, 2\rangle)$

④ $(\mathbf{3}, \langle 1, 3, 4, 3\rangle)$
⑤ $(\mathbf{3}, \langle 2, 3, 4, 2\rangle)$
⑥ $(\mathbf{3}, \langle 2, 3, 4, 3\rangle)$

Figure 5.110: All solutions corresponding to the non ground example of the ATLEAST_NVALUE constraint of the **All solutions** slot

Typical

$\texttt{NVAL} > 0$
$\texttt{NVAL} < |\texttt{VARIABLES}|$
$\texttt{NVAL} <$range(VARIABLES.var)
$|\texttt{VARIABLES}| > 1$

Typical model nval(VARIABLES.var) > 2

Symmetries

- NVAL can be decreased to any value ≥ 0.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Extensible wrt. VARIABLES.

Remark The ATLEAST_NVALUE constraint was first introduced by J.-C. Régin under the name K_DIFF in [352]. Later on the ATLEAST_NVALUE constraint was introduced together with the ATMOST_NVALUE constraint by C. Bessière *et al.* in an article [68] providing filtering algorithms for the NVALUE constraint.

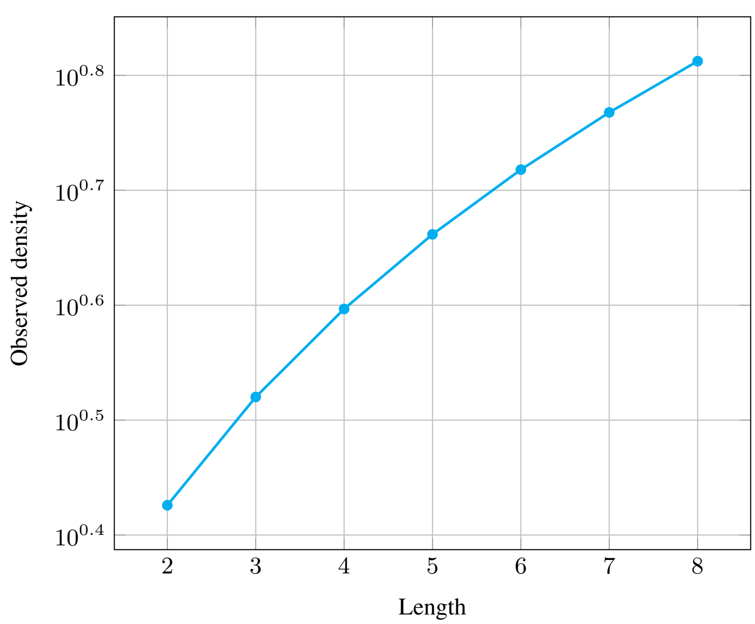
Algorithm [68] provides a sketch of a filtering algorithm enforcing arc-consistency for the ATLEAST_NVALUE constraint. This algorithm is based on the maximal matching in a bipartite graph.

Counting

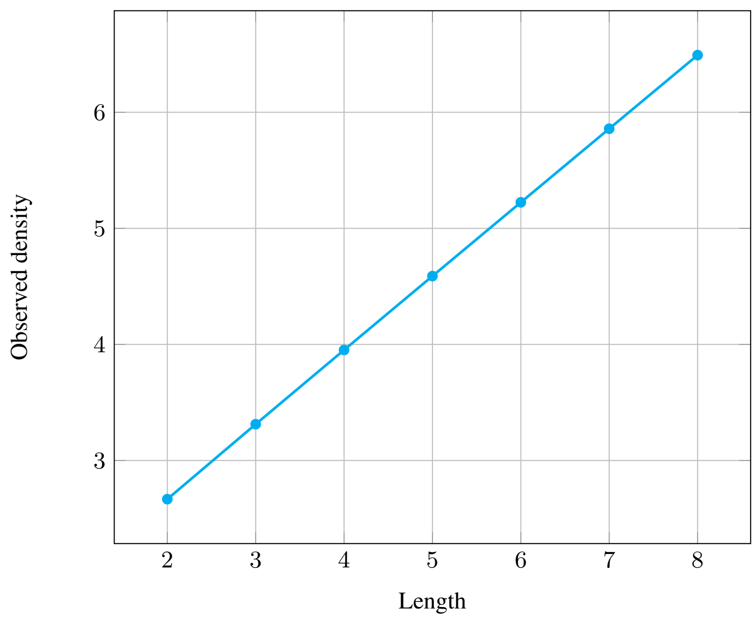
Length (n)	2	3	4	5	6	7	8
Solutions	24	212	2470	35682	614600	12286024	279472266

Number of solutions for ATLEAST_NVALUE: domains $0..n$

Solution density for ATLEAST_NVALUE

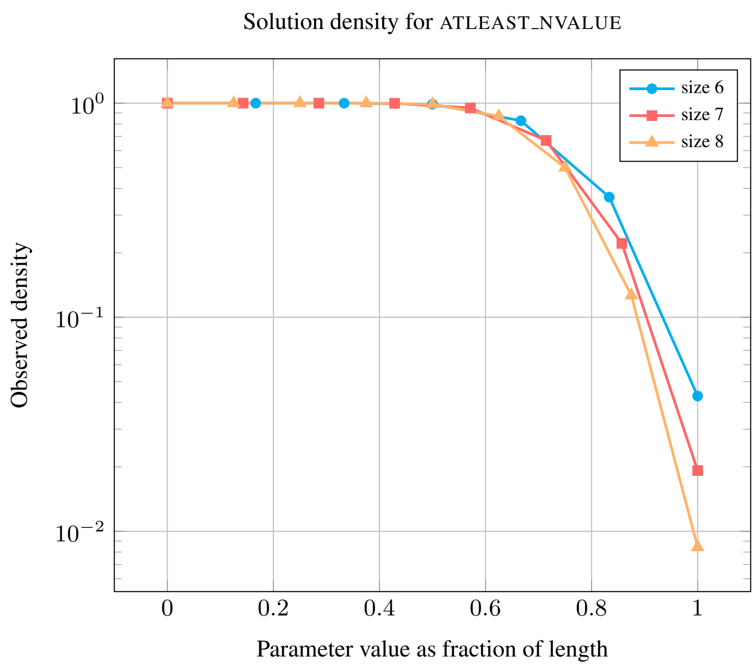


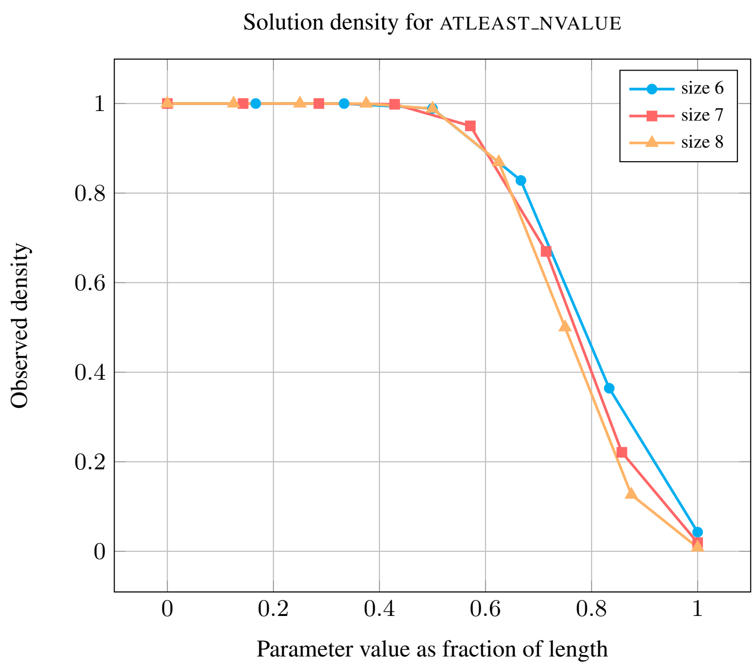
Solution density for ATLEAST_NVALUE



Length (n)		2	3	4	5	6	7	8
Total		24	212	2470	35682	614600	12286024	279472266
Parameter value	0	9	64	625	7776	117649	2097152	43046721
	1	9	64	625	7776	117649	2097152	43046721
	2	6	60	620	7770	117642	2097144	43046712
	3	-	24	480	7320	116340	2093616	43037568
	4	-	-	120	4320	97440	1992480	42550704
	5	-	-	-	720	42840	1404480	37406880
	6	-	-	-	-	5040	463680	21530880
	7	-	-	-	-	-	40320	5443200
	8	-	-	-	-	-	-	362880

Solution count for ATLEAST_NVALUE: domains $0..n$





See also **comparison swapped:** ATMOST_NVALUE.

implied by: AND, EQUIVALENT, IMPLY, NAND, NOR, NVALUE ($\geq$ NVAL *replaced by* $=$ NVAL), NVISIBLE_FROM_END, NVISIBLE_FROM_START, OR,

SIZE_MAX_SEQ_ALLDIFFERENT, SIZE_MAX_STARTING_SEQ_ALLDIFFERENT, XOR.

uses in its reformulation: NOT_ALL_EQUAL.

Keywords

constraint type: counting constraint, value partitioning constraint.

filtering: bipartite matching, arc-consistency.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, number of distinct values.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSCC** $\geq$ NVAL
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.111 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a specific value that is assigned to some variables of the VARIABLES collection. The 4 following values 1, 3, 6 and 7 are used by the variables of the VARIABLES collection.

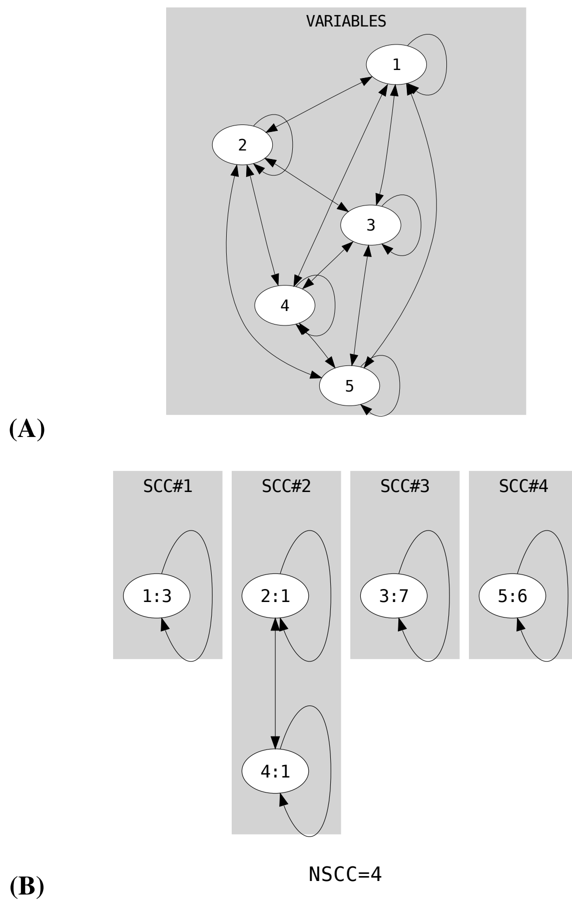


Figure 5.111: Initial and final graph of the ATLEAST_NVALUE constraint

5.40 ATLEAST_NVECTOR

▶ ▷ ◀ DESCRIPTION LINKS GRAPH

Origin

Derived from NVECTOR

Constraint

ATLEAST_NVECTOR(NVEC, VECTORS)

Type

VECTOR : collection(var−dvar)

Arguments

NVEC : dvar
VECTORS : collection(vec − VECTOR)

Restrictions

$|\mathtt{VECTOR}| \geq 1$
$\mathtt{NVEC} \geq 0$
$\mathtt{NVEC} \leq |\mathtt{VECTORS}|$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose

The number of distinct tuples of values taken by the vectors of the collection VECTORS is greater than or equal to NVEC. Two tuples of values $\langle A_1, A_2, \ldots, A_m\rangle$ and $\langle B_1, B_2, \ldots, B_m\rangle$ are *distinct* if and only if there exist an integer $i \in [1, m]$ such that $A_i \neq B_i$.

Example

$$\left(2, \left\langle \begin{array}{l} \mathtt{vec} - \langle 5,6\rangle, \\ \mathtt{vec} - \langle 5,6\rangle, \\ \mathtt{vec} - \langle 9,3\rangle, \\ \mathtt{vec} - \langle 5,6\rangle, \\ \mathtt{vec} - \langle 9,4\rangle \end{array} \right\rangle \right)$$

The ATLEAST_NVECTOR constraint holds since the collection VECTORS involves at least 2 distinct tuples of values (i.e., in fact the 3 distinct tuples $\langle 5,6\rangle$, $\langle 9,3\rangle$ and $\langle 9,4\rangle$).

Typical

$|\mathtt{VECTOR}| > 1$
$\mathtt{NVEC} > 1$
$\mathtt{NVEC} < |\mathtt{VECTORS}|$
$|\mathtt{VECTORS}| > 1$

Symmetries

- NVEC can be decreased to any value ≥ 0.
- Items of VECTORS are permutable.
- Items of VECTORS.vec are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values of VECTORS.vec can be swapped; all occurrences of a tuple of values of VECTORS.vec can be renamed to any unused tuple of values.

Arg. properties Extensible wrt. VECTORS.

Reformulation By introducing an extra variable $\mathtt{NV} \in [0, |\mathtt{VECTORS}|]$, the ATLEAST_NVECTOR(NV, VECTORS) constraint can be expressed in term of an NVECTOR(NV, VECTORS) constraint and of an inequality constraint $\mathtt{NV} \geq \mathtt{NVEC}$.

See also

comparison swapped: ATMOST_NVECTOR.

implied by: NVECTOR *($\geq$ NVEC replaced by $=$ NVEC)*, ORDERED_ATLEAST_NVECTOR.

used in graph description: LEX_EQUAL.

Keywords

characteristic of a constraint: vector.

constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes.

problems: domination.

Arc input(s)	VECTORS
Arc generator	$CLIQUE \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_EQUAL(vectors1.vec, vectors2.vec)
Graph property(ies)	**NSCC** $\geq$ NVEC
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.112 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 3 following tuple of values $\langle 5, 6\rangle$, $\langle 9, 3\rangle$ and $\langle 9, 4\rangle$ are used by the vectors of the VECTORS collection.

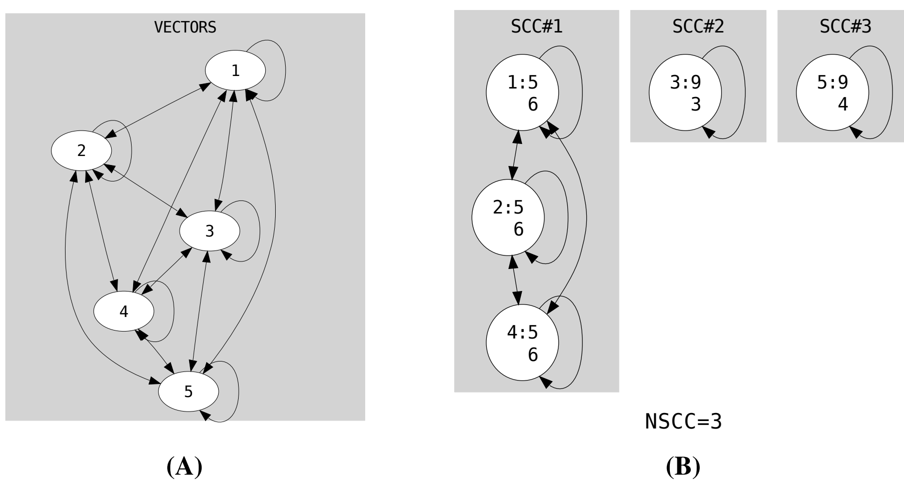


Figure 5.112: Initial and final graph of the ATLEAST_NVECTOR constraint

5.41 ATMOST

▶ ▷ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint ATMOST(N, VARIABLES, VALUE)

Synonym COUNT.

Arguments

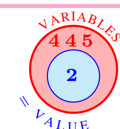
N	:	int
VARIABLES	:	collection(var−dvar)
VALUE	:	int

Restrictions

$\mathtt{N} \geq 0$
required(VARIABLES, var)

Purpose At most N variables of the VARIABLES collection are assigned value VALUE.

Example $(1, \langle 4, 2, 4, 5\rangle, 2)$



The ATMOST constraint holds since at most 1 value of the collection $\langle 4, 2, 4, 5\rangle$ is equal to value 2.

All solutions Figure 5.113 gives all solutions to the following non ground instance of the ATMOST constraint: $\mathtt{V}_1 \in [1, 2]$, $\mathtt{V}_2 \in [2, 3]$, $\mathtt{V}_3 \in [5, 6]$, $\mathtt{V}_4 \in [2, 3]$, ATMOST$(1, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4\rangle, 2)$.

① $(1, \langle 1, 2, 5, 3\rangle, 2)$
② $(1, \langle 1, 2, 6, 3\rangle, 2)$
③ $(1, \langle 1, 3, 5, 2\rangle, 2)$
④ $(1, \langle 1, 3, 5, 3\rangle, 2)$

⑤ $(1, \langle 1, 3, 6, 2\rangle, 2)$
⑥ $(1, \langle 1, 3, 6, 3\rangle, 2)$
⑦ $(1, \langle 2, 3, 5, 3\rangle, 2)$
⑧ $(1, \langle 2, 3, 6, 3\rangle, 2)$

Figure 5.113: All solutions corresponding to the non ground example of the ATMOST constraint of the **All solutions** slot

Typical

$\mathtt{N} > 0$
$\mathtt{N} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$
ATLEAST(1, VARIABLES, VALUE)

Symmetries

- Items of VARIABLES are permutable.
- N can be increased.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that is different from VALUE.

Arg. properties Contractible wrt. VARIABLES.

Systems OCCURENCEMAX in **Choco**, COUNT in **Gecode**, ATMOST in **Gecode**, COUNT in **JaCoP**, AT_MOST in **MiniZinc**, COUNT in **SICStus**.

See also

common keyword: AMONG *(value constraint)*.

comparison swapped: ATLEAST.

generalisation: CUMULATIVE *(*variable *replaced by* task*)*.

implied by: EXACTLY *(*$\leq$N *replaced by* $=$N*)*.

related: ROOTS.

soft variant: OPEN_ATMOST *(open constraint)*.

Keywords

characteristic of a constraint: automaton, automaton with counters.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint.

filtering: arc-consistency.

modelling: at most.

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var $=$ VALUE
Graph property(ies)	**NARC** $\leq$ N

Graph model Since each arc constraint involves only one vertex (VALUE is fixed), we employ the *SELF* arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.114 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

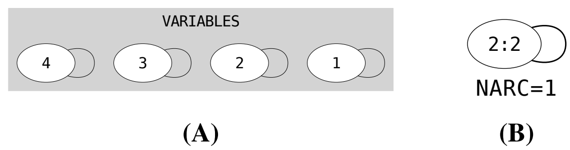


Figure 5.114: Initial and final graph of the ATMOST constraint

Automaton Figure 5.115 depicts the automaton associated with the ATMOST constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i = \mathtt{VALUE} \Leftrightarrow S_i$. The automaton counts the number of variables of the VARIABLES collection that are assigned value VALUE and finally checks that this number is less than or equal to N.

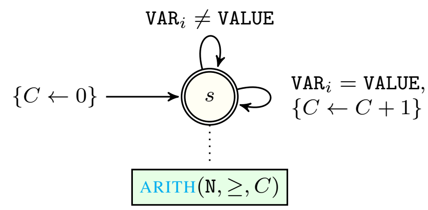


Figure 5.115: Automaton of the ATMOST constraint

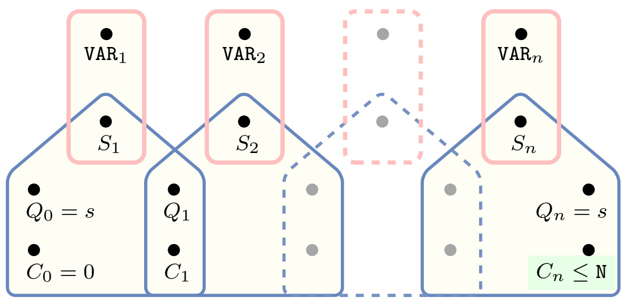


Figure 5.116: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the ATMOST constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.42 ATMOST1

► ▷ ◄ DESCRIPTION LINKS

Origin [376]

Constraint ATMOST1(SETS)

Synonym PAIR_ATMOST1.

Argument SETS : collection(s−svar, c−int)

Restrictions required(SETS, [s, c])
SETS.c ≥ 1

Purpose Given a collection of set variables $s_1, s_2, \ldots, s_n$ and their respective cardinalities $c_1, c_2, \ldots, c_n$, the ATMOST1 constraint forces the following two conditions:

- $\forall i \in [1, n] : |s_i| = c_i$,
- $\forall i, j \in [1, n]\ (i < j) : \ |s_i \bigcap s_j| \leq 1$.

Example

$$\left(\left\langle\begin{array}{ll} \texttt{s}-\{5,8\} & \texttt{c}-2, \\ \texttt{s}-\{5\} & \texttt{c}-1, \\ \texttt{s}-\{5,6,7\} & \texttt{c}-3, \\ \texttt{s}-\{1,4\} & \texttt{c}-2 \end{array}\right\rangle\right)$$

The ATMOST1 constraint holds since:

- $|\{5,8\}| = 2$, $|\{5\}| = 1$, $|\{5,6,7\}| = 3$, $|\{1,4\}| = 2$.
- $|\{5,8\} \bigcap \{5\}| \leq 1$, $|\{5,8\} \bigcap \{5,6,7\}| \leq 1$, $|\{5,8\} \bigcap \{1,4\}| \leq 1$,
  $|\{5\} \bigcap \{5,6,7\}| \leq 1$, $|\{5\} \bigcap \{1,4\}| \leq 1$,
  $|\{5,6,7\} \bigcap \{1,4\}| \leq 1$.

Typical $|\text{SETS}| > 1$

Symmetries

- Items of SETS are permutable.
- All occurrences of two distinct values of SETS.s can be swapped; all occurrences of a value of SETS.s can be renamed to any unused value.

Arg. properties Contractible wrt. SETS.

Remark When we have only two set variables the ATMOST1 constraint was called PAIR_ATMOST1 in [439].

Algorithm C. Bessière *et al.* have shown in [74] that it is NP-hard to enforce bound consistency for the ATMOST1 constraint. Consequently, following the first filtering algorithm from A. Sadler and C. Gervet [376], W.-J. van Hoeve and A. Sabharwal have proposed an algorithm that enforces bound-consistency when the ATMOST1 constraint involves only two sets variables [439].

Systems AT_MOST1 in **MiniZinc**.

Keywords **constraint arguments:** constraint involving set variables.

constraint type: predefined constraint.

filtering: bound-consistency.

5.43 ATMOST_NVALUE

▶ ▷ ◀ DESCRIPTION LINKS GRAPH

Origin [68]

Constraint ATMOST_NVALUE(NVAL, VARIABLES)

Synonyms SOFT_ALLDIFF_MAX_VAR, SOFT_ALLDIFFERENT_MAX_VAR, SOFT_ALLDISTINCT_MAX_VAR.

Arguments

NVAL	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

NVAL $\geq$ min(1, |VARIABLES|)
required(VARIABLES, var)

Purpose The number of distinct values taken by the variables of the collection VARIABLES is less than or equal to NVAL.

Example

$(4, \langle 3, 1, 3, 1, 6\rangle)$
$(3, \langle 3, 1, 3, 1, 6\rangle)$
$(1, \langle 3, 3, 3, 3, 3\rangle)$

The first ATMOST_NVALUE constraint holds since the collection $\langle 3, 1, 3, 1, 6\rangle$ involves at most 4 distinct values (i.e., in fact 3 distinct values).

Typical

NVAL > 1
NVAL $<$ |VARIABLES|
|VARIABLES| > 1

Symmetries

- NVAL can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.
- An occurrence of a value of VARIABLES.var can be replaced by any value of VARIABLES.var.

Arg. properties Contractible wrt. VARIABLES.

Remark This constraint was introduced together with the ATLEAST_NVALUE constraint by C. Bessière *et al.* in an article [68] providing filtering algorithms for the NVALUE constraint.

It was shown in [75] that, finding out whether a ATMOST_NVALUE constraint has a solution or not is NP-hard. This was achieved by reduction from 3-SAT.

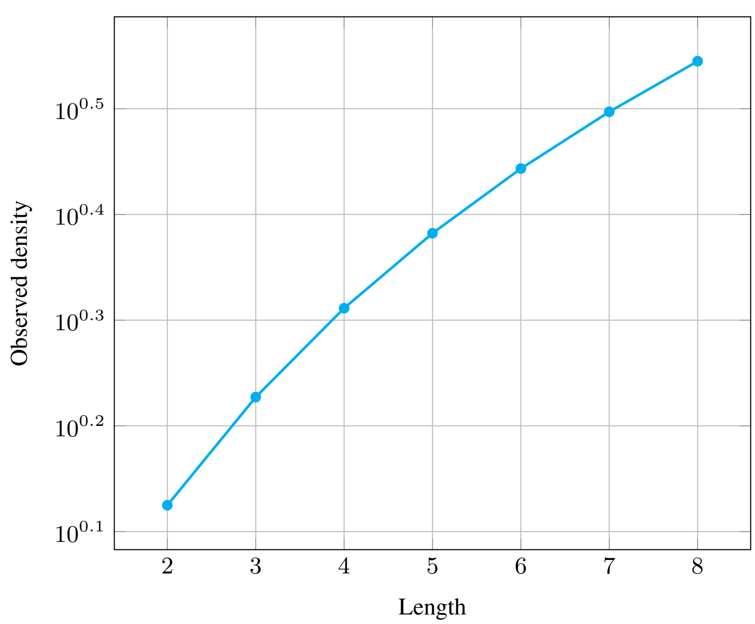
Algorithm [29] provides an algorithm that achieves bound consistency. [46] provides two filtering algorithms, while [68] provides a greedy algorithm and a graph invariant for evaluating the minimum number of distinct values. [68] also gives a linear relaxation for approximating the minimum number of distinct values.

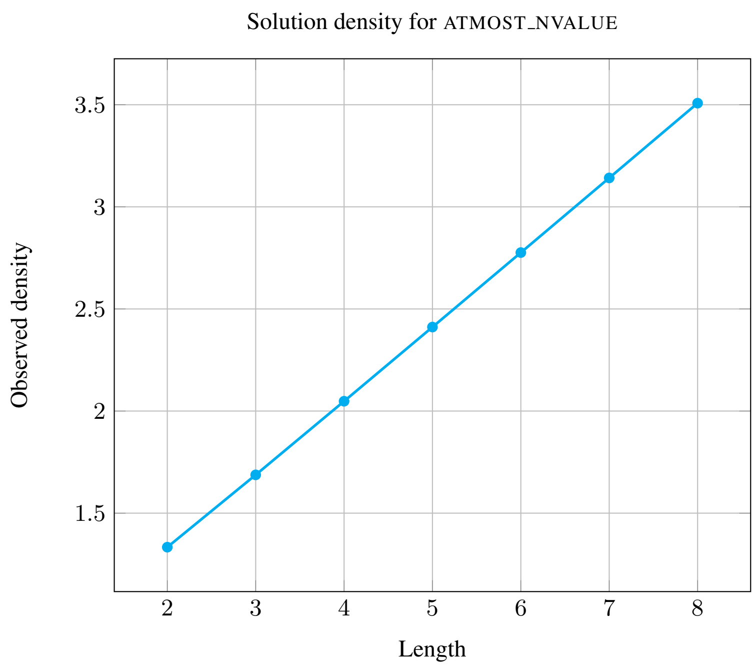
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	12	108	1280	18750	326592	6588344	150994944

Number of solutions for ATMOST_NVALUE: domains $0..n$

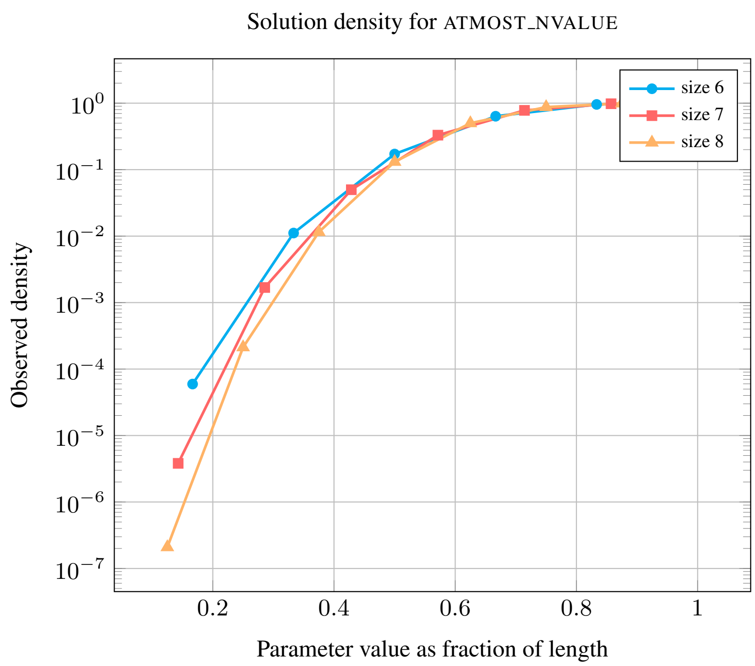
Solution density for ATMOST_NVALUE

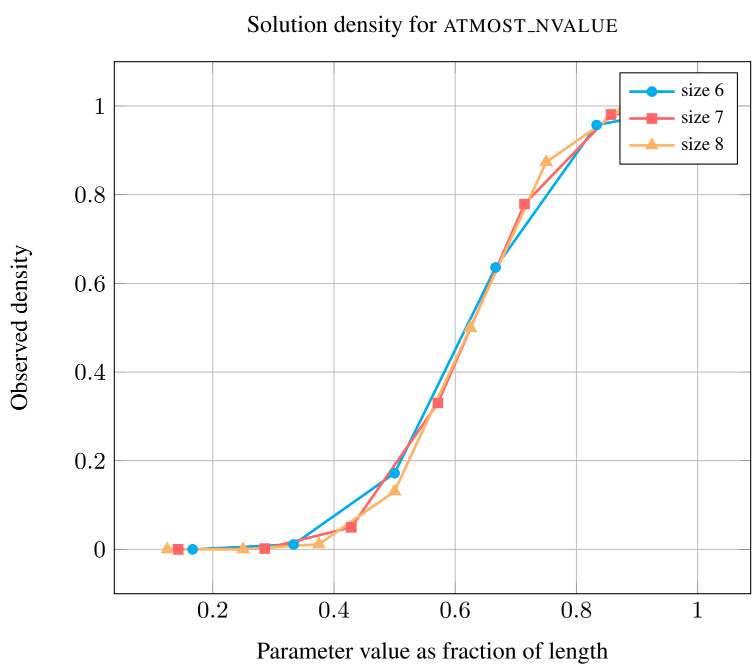




Length (n)		2	3	4	5	6	7	8
Total		12	108	1280	18750	326592	6588344	150994944
Parameter value	1	3	4	5	6	7	8	9
	2	9	40	145	456	1309	3536	9153
	3	-	64	505	3456	20209	104672	496017
	4	-	-	625	7056	74809	692672	5639841
	5	-	-	-	7776	112609	1633472	21515841
	6	-	-	-	-	117649	2056832	37603521
	7	-	-	-	-	-	2097152	42683841
	8	-	-	-	-	-	-	43046721

Solution count for ATMOST_NVALUE: domains $0..n$





Systems ATMOSTNVALUE in **Choco**.

See also **comparison swapped:** ATLEAST_NVALUE.

implied by: NVALUE *($\leq$ `NVAL` replaced by $=$ `NVAL`)*.

related: SOFT_ALL_EQUAL_MAX_VAR, SOFT_ALL_EQUAL_MIN_CTR, SOFT_ALL_EQUAL_MIN_VAR, SOFT_ALLDIFFERENT_CTR, SOFT_ALLDIFFERENT_VAR.

Keywords

complexity: 3-SAT.

constraint type: counting constraint, value partitioning constraint.

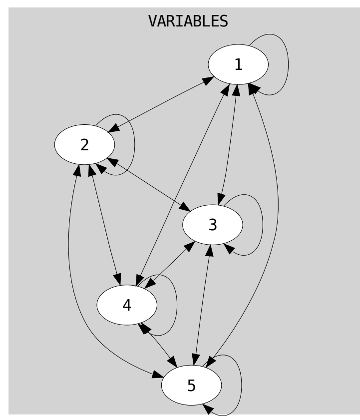
filtering: bound-consistency.

final graph structure: strongly connected component, equivalence.

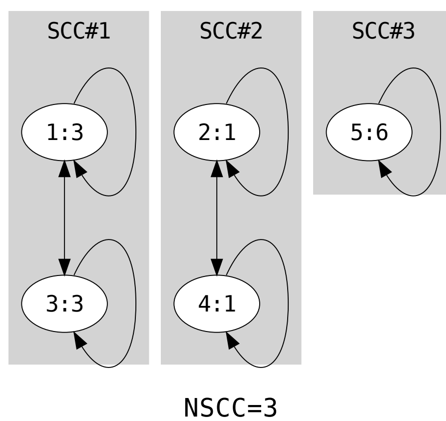
modelling: number of distinct equivalence classes, number of distinct values.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSCC** $\leq$ NVAL
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.117 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a specific value that is assigned to some variables of the VARIABLES collection. The 3 following values 1, 3 and 6 are used by the variables of the VARIABLES collection.



(A)



(B)

Figure 5.117: Initial and final graph of the ATMOST_NVALUE constraint

5.44 ATMOST_NVECTOR

► ▷ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from NVECTOR

Constraint ATMOST_NVECTOR(NVEC, VECTORS)

Type

VECTOR : collection(var−dvar)

Arguments

NVEC : dvar
VECTORS : collection(vec − VECTOR)

Restrictions

$|\text{VECTOR}| \geq 1$
$\text{NVEC} \geq \min(1, |\text{VECTORS}|)$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose

The number of distinct tuples of values taken by the vectors of the collection VECTORS is less than or equal to NVEC. Two tuples of values $\langle A_1, A_2, \ldots, A_m\rangle$ and $\langle B_1, B_2, \ldots, B_m\rangle$ are *distinct* if and only if there exist an integer $i \in [1, m]$ such that $A_i \neq B_i$.

Example

$$\left(3, \left\langle \begin{array}{l} \text{vec} - \langle 5, 6\rangle, \\ \text{vec} - \langle 5, 6\rangle, \\ \text{vec} - \langle 9, 3\rangle, \\ \text{vec} - \langle 5, 6\rangle, \\ \text{vec} - \langle 9, 3\rangle \end{array} \right\rangle \right)$$

The ATMOST_NVECTOR constraint holds since the collection VECTORS involves at most 3 distinct tuples of values (i.e., in fact the 2 distinct tuples $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$).

Typical

$|\text{VECTOR}| > 1$
$\text{NVEC} > 1$
$\text{NVEC} < |\text{VECTORS}|$
$|\text{VECTORS}| > 1$

Symmetries

- NVEC can be increased.
- Items of VECTORS are permutable.
- Items of VECTORS.vec are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values of VECTORS.vec can be swapped; all occurrences of a tuple of values of VECTORS.vec can be renamed to any unused tuple of values.

Arg. properties Contractible wrt. VECTORS.

Reformulation By introducing an extra variable NV $\in$ $[0, |\mathtt{VECTORS}|]$, the ATMOST_NVECTOR(NV, VECTORS) constraint can be expressed in term of an NVECTOR(NV, VECTORS) constraint and of an inequality constraint NV $\leq$ NVEC.

See also **comparison swapped:** ATLEAST_NVECTOR.

implied by: NVECTOR *($\leq$ NVEC replaced by $=$ NVEC)*, ORDERED_ATMOST_NVECTOR.

used in graph description: LEX_EQUAL.

Keywords **characteristic of a constraint:** vector.

constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes.

problems: domination.

Arc input(s)	VECTORS
Arc generator	$CLIQUE \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_EQUAL(vectors1.vec, vectors2.vec)
Graph property(ies)	**NSCC** $\leq$ NVEC
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.118 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 2 following tuple of values $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$ are used by the vectors of the VECTORS collection.

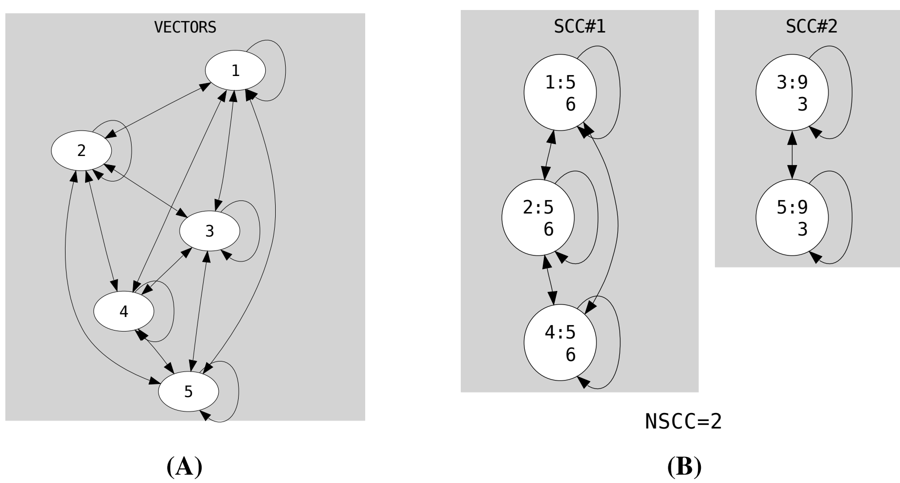


Figure 5.118: Initial and final graph of the ATMOST_NVECTOR constraint

5.45 BALANCE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint BALANCE(BALANCE, VARIABLES)

Arguments

```
BALANCE   : dvar
VARIABLES : collection(var-dvar)
```

Restrictions

BALANCE ≥ 0
BALANCE $\leq$ max$(0, |$VARIABLES$| - 2)$
required(VARIABLES, var)

Purpose

BALANCE is equal to the difference between the number of occurrence of the value that occurs the most and the value that occurs the least within the collection of variables VARIABLES.

Example

$(2, \langle 3, 1, 7, 1, 1\rangle)$
$(0, \langle 3, 3, 1, 1, 1, 3\rangle)$
$(4, \langle 3, 1, 1, 1, 1, 1\rangle)$

In the first example, values $1, 3$ and 7 are respectively used $3, 1$ and 1 times. The corresponding BALANCE constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., $3 - 1$). Figure 5.119 shows the solution associated with this first example.

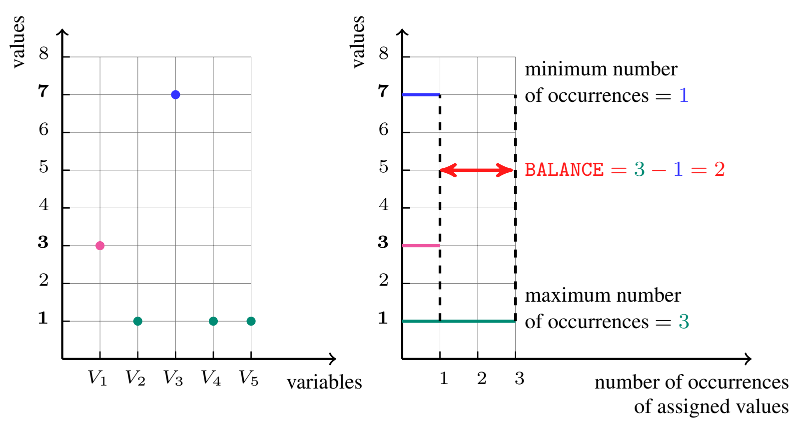


Figure 5.119: Illustration of the first example of the **Example** slot: five variables V_1, V_2, V_3, V_4, V_5 respectively fixed to values 3, 1, 7, 1 and 1, and the corresponding value of BALANCE $= 2$

All solutions Figure 5.120 gives all solutions to the following non ground instance of the BALANCE constraint: $\mathtt{BALANCE} \in [\mathbf{2}, \mathbf{3}]$, $\mathtt{V}_1 \in [0,5]$, $\mathtt{V}_2 \in [2,6]$, $\mathtt{V}_3 \in [0,1]$, $\mathtt{V}_4 \in [1,2]$, BALANCE$(\mathtt{BALANCE}, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4 \rangle)$.

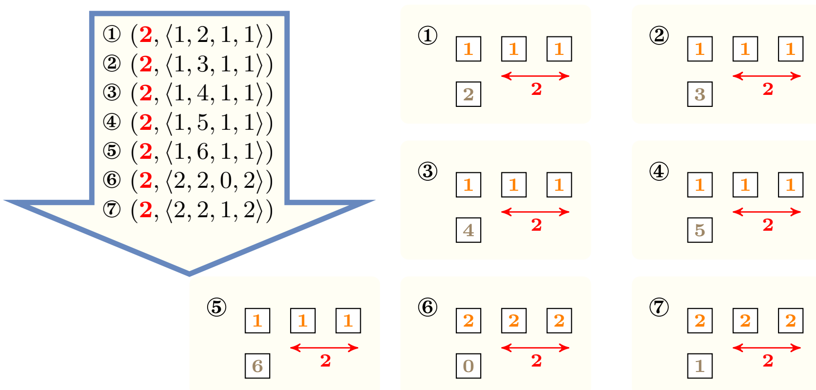


Figure 5.120: All solutions corresponding to the non ground example of the BALANCE constraint of the **All solutions** slot

Typical

$\mathtt{BALANCE} \leq 2 + |\mathtt{VARIABLES}|/10$
$|\mathtt{VARIABLES}| > 2$

Typical model

$\mathtt{nval}(\mathtt{VARIABLES.var}) > 2$

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties

Functional dependency: BALANCE determined by VARIABLES.

Usage An application of the BALANCE constraint is to enforce a *balanced assignment* of values, no matter how many distinct values will be used. In this case one will *push down* the maximum value of the first argument of the BALANCE constraint.

- On the one hand the BALANCE constraint should be used on problems for which, due to some constraints, not all consecutive values will be assigned. This is the case, for example, for some frequency assignment problems where, due to some interference, all used values are not necessarily consecutive.
- On the other hand if only consecutive values are used then one should rather employ the ALL_BALANCE constraint. This is the case, for example, in some resource assignment problems where we know in advance that all resources will be used.

Remark If we do not want to use an automaton with an array of counters a first possible reformulation of the BALANCE constraint introduced in [44, pages 1860–1861] can be achieved in the following way. We use a SORT constraint in order to reorder the variables of the collection

VARIABLES and compute the difference between the longest and the smallest sequences of consecutive values.

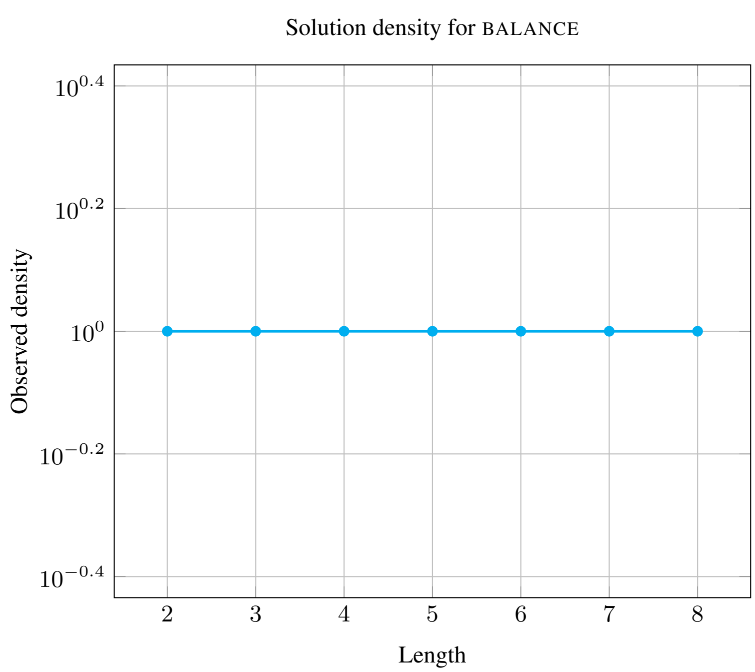
A more efficient reformulation introduced in [45, page 2071] uses a GLOBAL_CARDINALITY constraint for exposing the occurrence variables attached to each value that can possibly be assigned to the variables of the BALANCE constraint. Then the BALANCE argument is set to the difference between the maximum value of the occurrence variables and the minimum value of the occurrence variables that are different from 0.

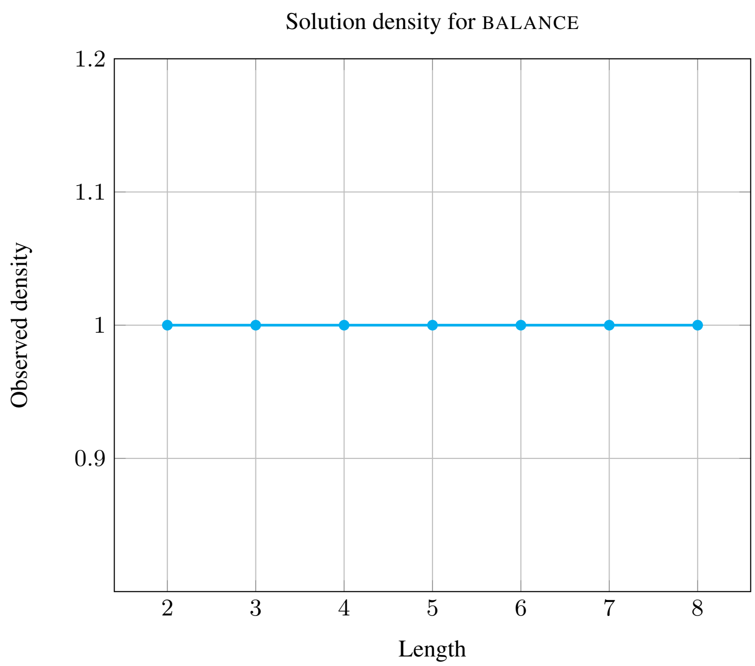
Algorithm It was shown in [77] that achieving arc-consistency on the BALANCE constraint is NP-hard.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

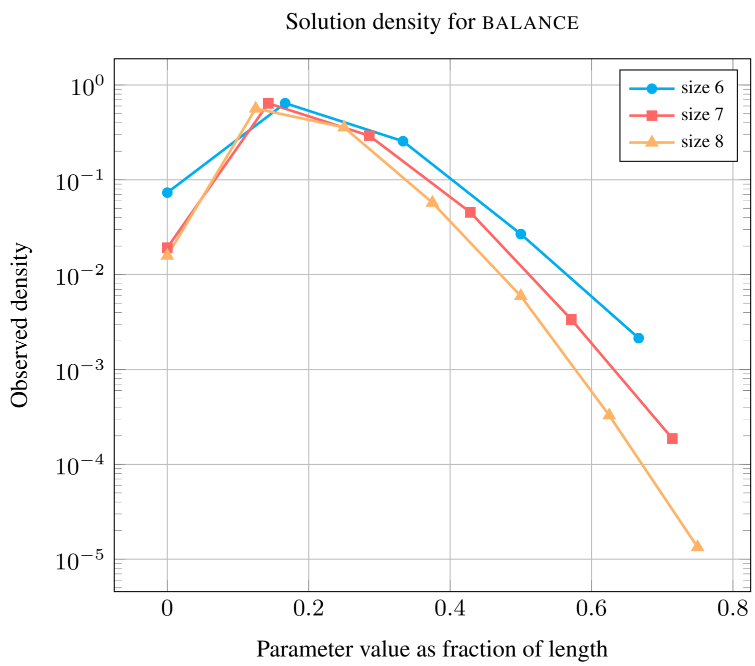
Number of solutions for BALANCE: domains $0..n$

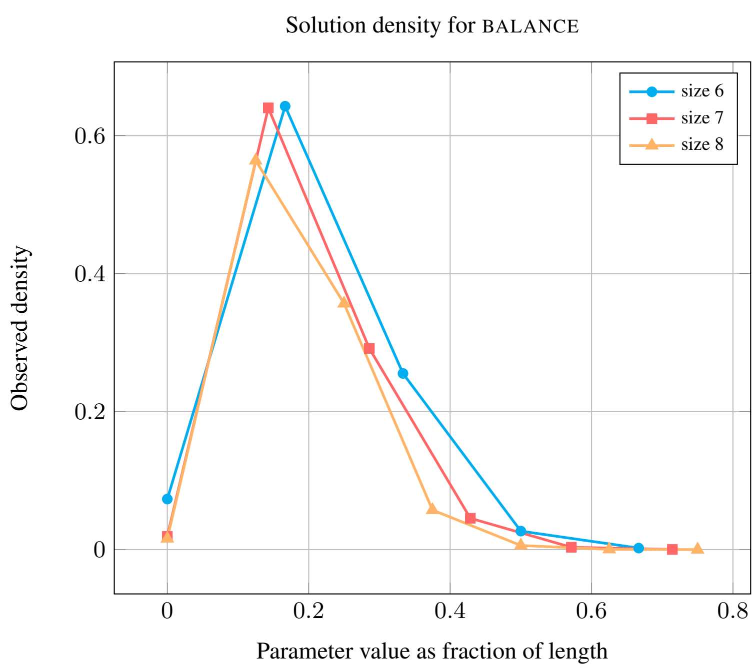




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	9	28	185	726	8617	40328	682929
	1	-	36	360	5700	75600	1342600	24272640
	2	-	-	80	1200	30030	611520	15350832
	3	-	-	-	150	3150	95256	2469600
	4	-	-	-	-	252	7056	256032
	5	-	-	-	-	-	392	14112
	6	-	-	-	-	-	-	576

Solution count for BALANCE: domains $0..n$





See also

generalisation: BALANCE_INTERVAL (`variable` *replaced by* `variable`/`constant`), BALANCE_MODULO (`variable` *replaced by* `variable` mod `constant`), BALANCE_PARTITION (`variable` *replaced by* `variable` $\in$ `partition`).

implies: SOFT_ALL_EQUAL_MIN_CTR.

related: ALL_BALANCE *(take into account unused values)*, BALANCE_CYCLE *(balanced assignment versus graph partitionning with balanced cycles)*, BALANCE_PATH *(balanced assignment versus graph partitionning with balanced paths)*, BALANCE_TREE *(balanced assignment versus graph partitionning with balanced trees)*, NVALUE *(no restriction on how balanced an assignment is)*, TREE_RANGE *(balanced assignment versus balanced tree)*.

shift of concept: EQUILIBRIUM.

Keywords

application area: assignment.

characteristic of a constraint: automaton, automaton with array of counters.

constraint arguments: pure functional dependency.

constraint type: value constraint.

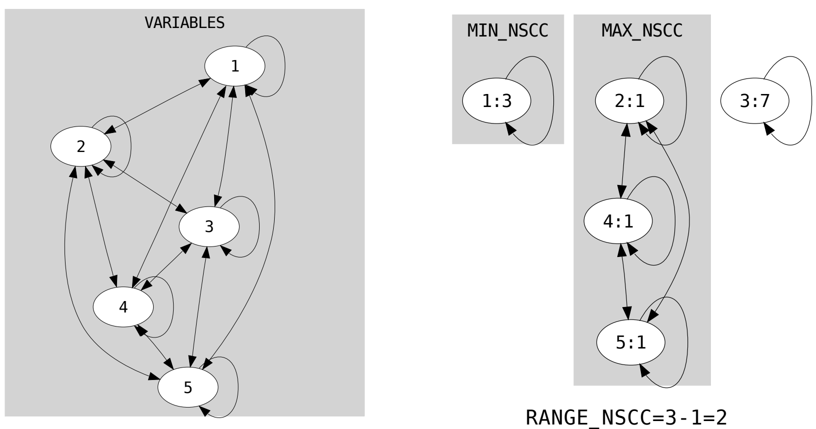
final graph structure: equivalence.

modelling: balanced assignment, functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**RANGE_NSCC** = BALANCE
Graph class	EQUIVALENCE

Graph model The graph property **RANGE_NSCC** constraints the difference between the sizes of the largest and smallest strongly connected components.

Parts (A) and (B) of Figure 5.121 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **RANGE_NSCC** graph property, we show the largest and smallest strongly connected components of the final graph.



(A) **(B)**

Figure 5.121: Initial and final graph of the BALANCE constraint

Automaton Figure 5.122 depicts the automaton associated with the BALANCE constraint. To each item of the collection VARIABLES corresponds a signature variable S_i that is equal to 1.

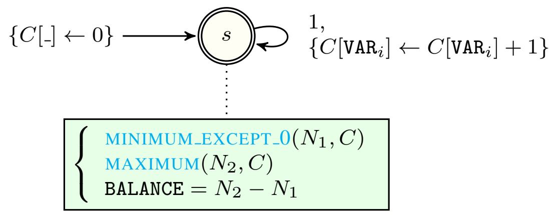


Figure 5.122: Automaton of the BALANCE constraint

5.46 BALANCE_CYCLE

Origin

derived from BALANCE and CYCLE

Constraint

BALANCE_CYCLE(BALANCE, NODES)

Arguments

BALANCE : dvar
NODES : collection(index−int, succ−dvar)

Restrictions

$\texttt{BALANCE} \geq 0$
$\texttt{BALANCE} \leq \texttt{max}(0, |\texttt{NODES}| - 2)$
$\texttt{required}(\texttt{NODES}, [\texttt{index}, \texttt{succ}])$
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
$\texttt{distinct}(\texttt{NODES}, \texttt{index})$
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$

Purpose

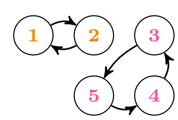
Consider a digraph G described by the NODES collection. Partition G into a set of vertex disjoint circuits in such a way that each vertex of G belongs to a single circuit. BALANCE is equal to the difference between the number of vertices of the largest circuit and the number of vertices of the smallest circuit.

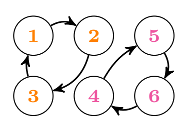
Example

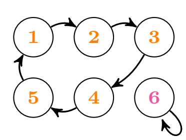
$$\left(1, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-2, \\ \texttt{index}-2 & \texttt{succ}-1, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-3, \\ \texttt{index}-5 & \texttt{succ}-4 \end{array} \right\rangle\right)$$

$$\left(0, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-2, \\ \texttt{index}-2 & \texttt{succ}-3, \\ \texttt{index}-3 & \texttt{succ}-1, \\ \texttt{index}-4 & \texttt{succ}-5, \\ \texttt{index}-5 & \texttt{succ}-6, \\ \texttt{index}-6 & \texttt{succ}-4 \end{array} \right\rangle\right)$$

$$\left(4, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-2, \\ \texttt{index}-2 & \texttt{succ}-3, \\ \texttt{index}-3 & \texttt{succ}-4, \\ \texttt{index}-4 & \texttt{succ}-5, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-6 \end{array} \right\rangle\right)$$







In the first example we have the following two circuits: $1 \rightarrow 2 \rightarrow 1$ and $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$. Since $\texttt{BALANCE} = 1$ is the difference between the number of vertices of the largest circuit (i.e., 3) and the number of vertices of the smallest circuit (i.e., 2) the corresponding BALANCE_CYCLE constraint holds.

All solutions

Figure 5.123 gives all solutions to the following non ground instance of the BALANCE_CYCLE constraint: BALANCE $\in [0,1]$, $S_1 \in [1,2]$, $S_2 \in [1,3]$, $S_3 \in [3,5]$, $S_4 \in [3,4]$, $S_5 \in [2,5]$, BALANCE_CYCLE(BALANCE, $\langle 1\ S_1, 2\ S_2, 3\ S_3, 4\ S_4, 5\ S_5\rangle$).

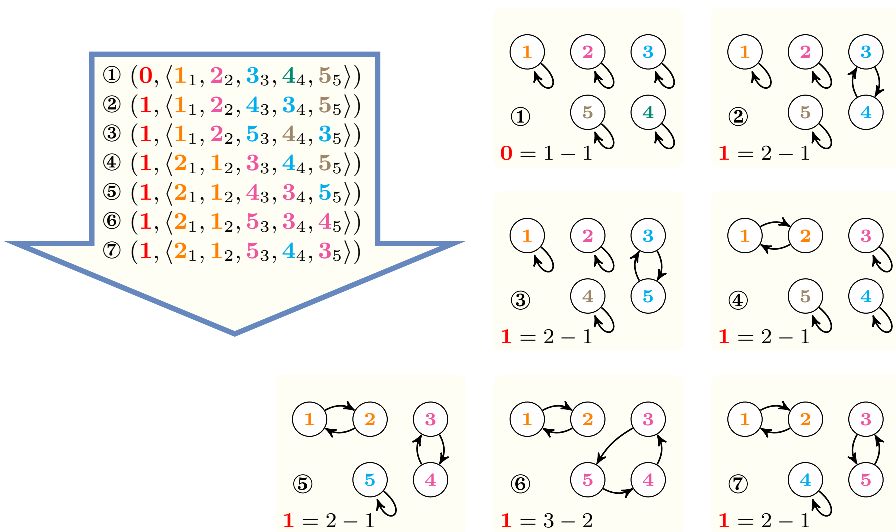


Figure 5.123: All solutions corresponding to the non ground example of the BALANCE_CYCLE constraint of the **All solutions** slot; the index attribute is displayed as indices of the succ attribute, and all vertices of a same cycle are coloured by the same colour; the bottom left part of each subfigure shows how the BALANCE argument (in red) is related to the largest and to the smallest cycles.

Typical

$|\text{NODES}| > 2$

Symmetry

Items of NODES are permutable.

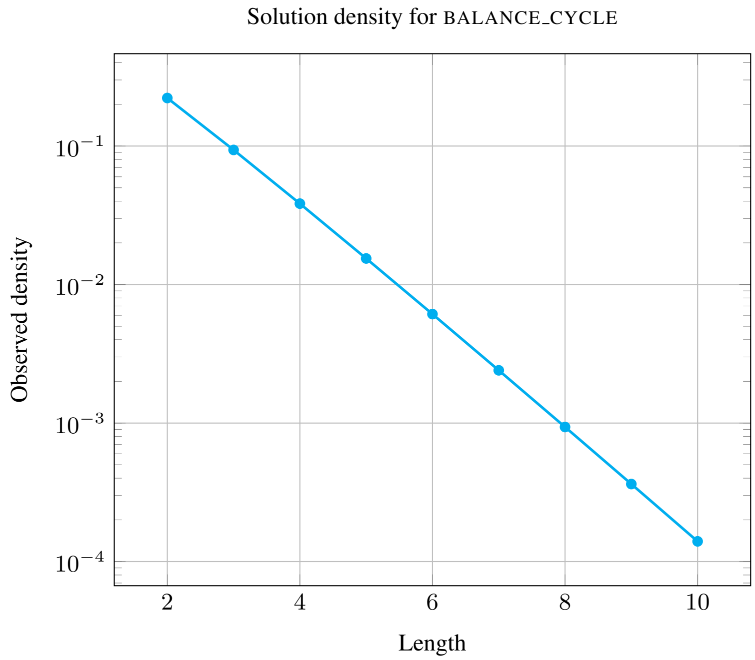
Arg. properties

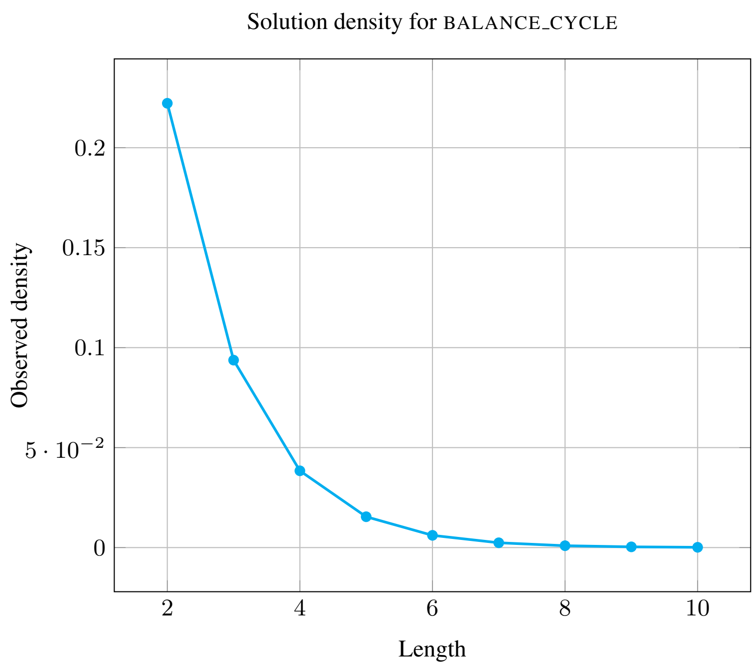
Functional dependency: BALANCE determined by NODES.

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	2	6	24	120	720	5040	40320	362880	3628800

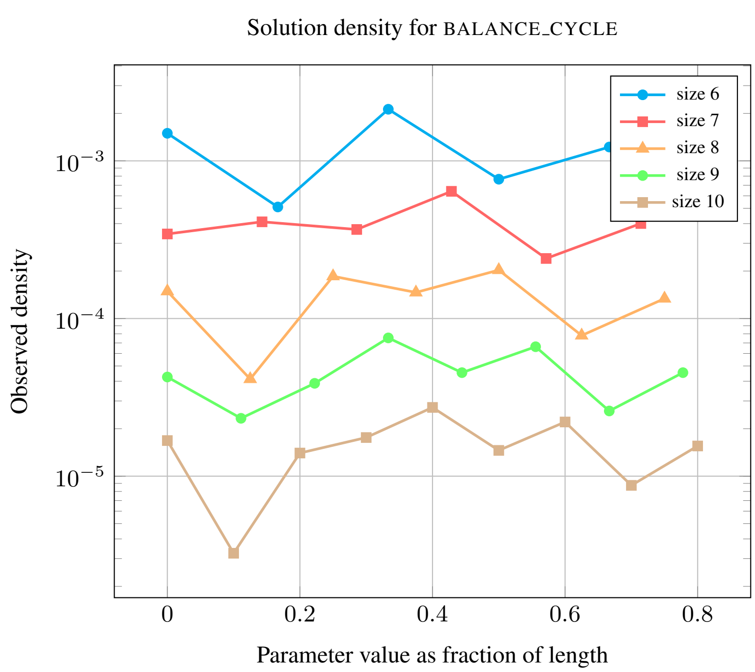
Number of solutions for BALANCE_CYCLE: domains $0..n$

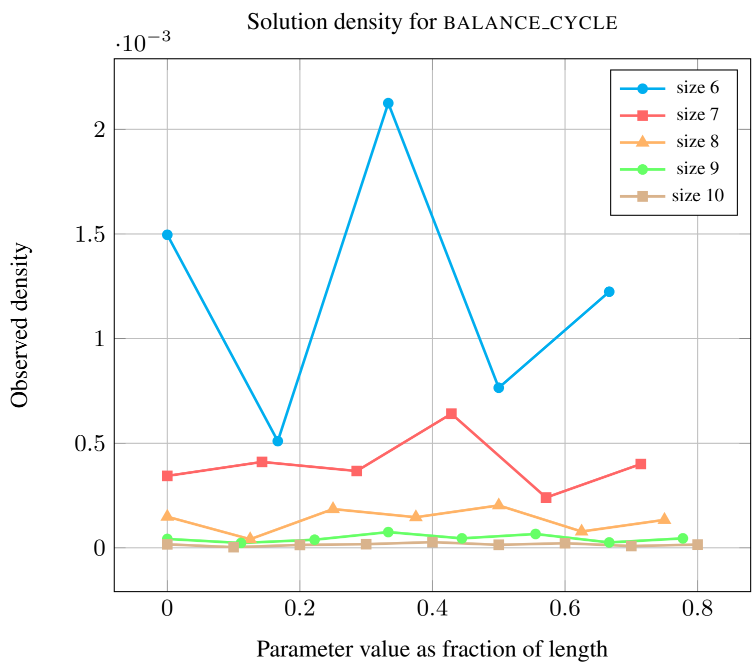

Solution density for BALANCE_CYCLE
Observed density
10^{-1}
10^{-2}
10^{-3}
10^{-4}
2
4
6
8
10
Length


Solution density for BALANCE_CYCLE
Observed density
0.2
0.15
0.1
$5 \cdot 10^{-2}$
0
2
4
6
8
10
Length

Length (n)		2	3	4	5	6	7	8	9	10
Total		2	6	24	120	720	5040	40320	362880	3628800
Parameter value	0	2	3	10	25	176	721	6406	42561	436402
	1	-	3	6	45	60	861	1778	23283	84150
	2	-	-	8	20	250	770	7980	38808	363680
	3	-	-	-	30	90	1344	6300	75348	456120
	4	-	-	-	-	144	504	8736	45360	708048
	5	-	-	-	-	-	840	3360	66240	378000
	6	-	-	-	-	-	-	5760	25920	572400
	7	-	-	-	-	-	-	-	45360	226800
	8	-	-	-	-	-	-	-	-	403200

Solution count for BALANCE_CYCLE: domains $0..n$





See also **related:** BALANCE *(equivalence classes correspond to vertices in same cycle rather than variables assigned to the same value)*, CYCLE *(do not care how many cycles but how balanced the cycles are)*.

Keywords **combinatorial object:** permutation.

constraint type: graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck.

final graph structure: circuit, connected component, strongly connected component, one_succ.

modelling: cycle, functional dependency.

Cond. implications

- BALANCE_CYCLE(BALANCE, NODES)
  with BALANCE > 0
  and BALANCE ≤ 2
  implies ALL_DIFFER_FROM_AT_LEAST_K_POS(K : BALANCE, VECTORS : NODES).

- BALANCE_CYCLE(BALANCE, NODES)
  implies PERMUTATION(VARIABLES : NODES).

Arc input(s)	NODES
Arc generator	*CLIQUE* $\mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **NTREE** = 0 • **RANGE_NCC** = BALANCE
Graph class	ONE_SUCC

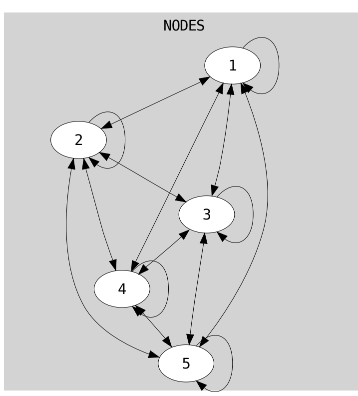
Graph model From the restrictions and from the arc constraint, we deduce that we have a bijection from the successor variables to the values of interval $[1, |\text{NODES}|]$. With no explicit restrictions it would have been impossible to derive this property.

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the BALANCE_CYCLE constraint considers objects that have two attributes:

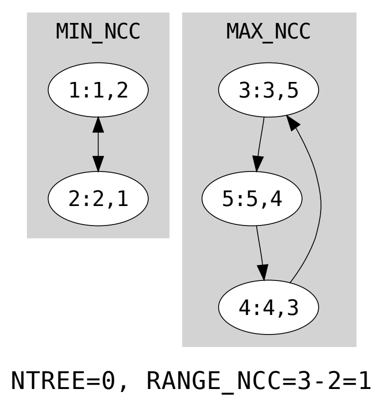
- One fixed attribute index that is the identifier of the vertex,
- One variable attribute succ that is the successor of the vertex.

The graph property **NTREE** = 0 is used in order to avoid having vertices that both do not belong to a circuit and have at least one successor located on a circuit. This concretely means that all vertices of the final graph should belong to a circuit.

Parts (A) and (B) of Figure 5.124 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **RANGE_NCC** graph property, we show the connected components of the final graph. The constraint holds since all the vertices belong to a circuit (i.e., **NTREE** = 0) and since BALANCE = **RANGE_NCC** = 1.



(A)



(B)

Figure 5.124: Initial and final graph of the BALANCE_CYCLE constraint

5.47 BALANCE_INTERVAL

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from BALANCE.

Constraint BALANCE_INTERVAL(BALANCE, VARIABLES, SIZE_INTERVAL)

Arguments

BALANCE	:	dvar
VARIABLES	:	collection(var−dvar)
SIZE_INTERVAL	:	int

Restrictions

BALANCE ≥ 0
BALANCE $\leq$ max$(0, |$VARIABLES$| - 2)$
required(VARIABLES, var)
SIZE_INTERVAL > 0

Purpose

Consider the largest set $\mathcal{S}_1$ (respectively the smallest set $\mathcal{S}_2$) of variables of the collection VARIABLES that take their values in a same interval [SIZE_INTERVAL $\cdot$ k, SIZE_INTERVAL $\cdot$ k + SIZE_INTERVAL $- 1$], where k is an integer. BALANCE is equal to the difference between the cardinality of $\mathcal{S}_2$ and the cardinality of $\mathcal{S}_1$.

Example

$(3, \langle 6, 4, 3, 3, 4 \rangle, 3)$

In the example, the third argument SIZE_INTERVAL $=$ 3 defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Values 6,4,3,3 and 4 are respectively located within intervals $[6, 8]$, $[3, 5]$, $[3, 5]$, $[3, 5]$ and $[3, 5]$. Therefore intervals $[6, 8]$ and $[3, 5]$ are respectively used 1 and 4 times. The BALANCE_INTERVAL constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., $4 - 1$).

Typical

$|$VARIABLES$| > 2$
SIZE_INTERVAL > 1
SIZE_INTERVAL $<$range(VARIABLES.var)

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties Functional dependency: BALANCE determined by VARIABLES and SIZE_INTERVAL.

Usage An application of the BALANCE_INTERVAL constraint is to enforce a *balanced assignment* of interval of values, no matter how many distinct interval of values will be used. In this case one will *push down* the maximum value of the first argument of the BALANCE_INTERVAL constraint.

See also **specialisation:** BALANCE *(*`variable`/`constant` *replaced by* `variable`*)*.

Keywords **application area:** assignment.

characteristic of a constraint: automaton, automaton with array of counters.

constraint arguments: pure functional dependency.

constraint type: value constraint.

final graph structure: equivalence.

modelling: interval, balanced assignment, functional dependency.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL
Graph property(ies)	**RANGE_NSCC** = BALANCE
Graph class	EQUIVALENCE

Graph model The graph property **RANGE_NSCC** constraints the difference between the sizes of the largest and smallest strongly connected components.

Parts (A) and (B) of Figure 5.125 respectively show the initial and final graph associated with the **Example** slot. Since we use the **RANGE_NSCC** graph property, we show the largest and smallest strongly connected components of the final graph.

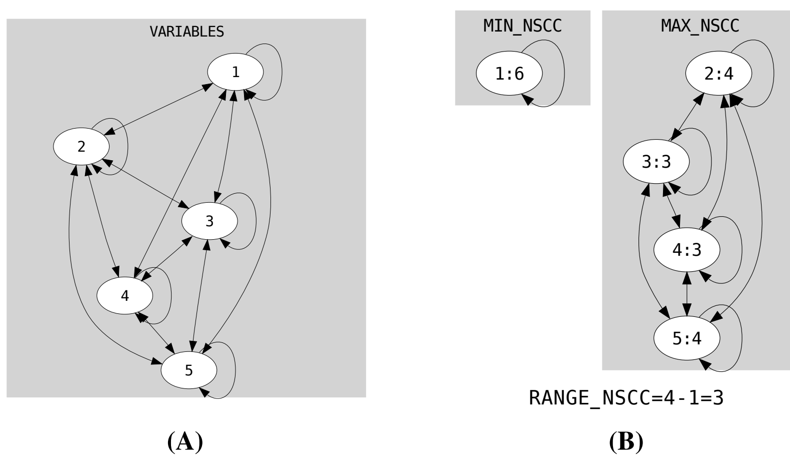


Figure 5.125: Initial and final graph of the BALANCE_INTERVAL constraint

Automaton Figure 5.126 depicts the automaton associated with the BALANCE_INTERVAL constraint. To each item of the collection VARIABLES corresponds a signature variable S_i that is equal to 1.

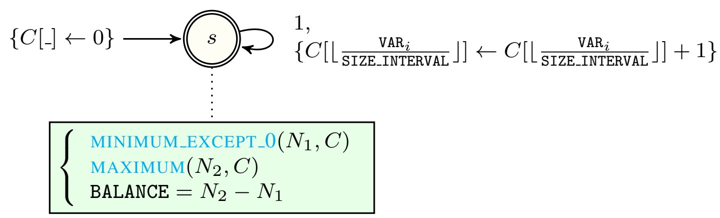


Figure 5.126: Automaton of the BALANCE_INTERVAL constraint

5.48 BALANCE_MODULO

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from BALANCE.

Constraint BALANCE_MODULO(BALANCE, VARIABLES, M)

Arguments

BALANCE	:	dvar
VARIABLES	:	collection(var−dvar)
M	:	int

Restrictions

BALANCE ≥ 0
BALANCE $\leq$ max(0, |VARIABLES| $-$ 2)
required(VARIABLES, var)
M > 0

Purpose Consider the largest set $\mathcal{S}_1$ (respectively the smallest set $\mathcal{S}_2$) of variables of the collection VARIABLES that have the same remainder when divided by M. BALANCE is equal to the difference between the cardinality of $\mathcal{S}_2$ and the cardinality of $\mathcal{S}_1$.

Example $(2, \langle 6, 1, 7, 1, 5\rangle, 3)$

In this example values 6, 1, 7, 1, 5 are respectively associated with the equivalence classes $6 \bmod 3 = 0$, $1 \bmod 3 = 1$, $7 \bmod 3 = 1$, $1 \bmod 3 = 1$, $5 \bmod 3 = 2$. Therefore the equivalence classes 0, 1 and 2 are respectively used 1, 3 and 1 times. The BALANCE_MODULO constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., $3 - 1$).

Typical

|VARIABLES| > 2
M > 1
M $<$ maxval(VARIABLES.var)

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value u of VARIABLES.var can be replaced by any other value v such that v is congruent to u modulo M.

Arg. properties Functional dependency: BALANCE determined by VARIABLES and M.

Usage An application of the BALANCE_MODULO constraint is to enforce a *balanced assignment* of values, no matter how many distinct equivalence classes will be used. In this case one will *push down* the maximum value of the first argument of the BALANCE_MODULO constraint.

See also **specialisation:** BALANCE (variable mod constant *replaced by* variable).

Keywords

application area: assignment.

characteristic of a constraint: modulo, automaton, automaton with array of counters.

constraint arguments: pure functional dependency.

constraint type: value constraint.

final graph structure: equivalence.

modelling: balanced assignment, functional dependency.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*↦collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var mod M = variables2.var mod M
Graph property(ies)	**RANGE_NSCC**= BALANCE
Graph class	EQUIVALENCE

Graph model The graph property **RANGE_NSCC** constraints the difference between the sizes of the largest and smallest strongly connected components.

Parts (A) and (B) of Figure 5.127 respectively show the initial and final graph associated with the **Example** slot. Since we use the **RANGE_NSCC** graph property, we show the largest and smallest strongly connected components of the final graph.

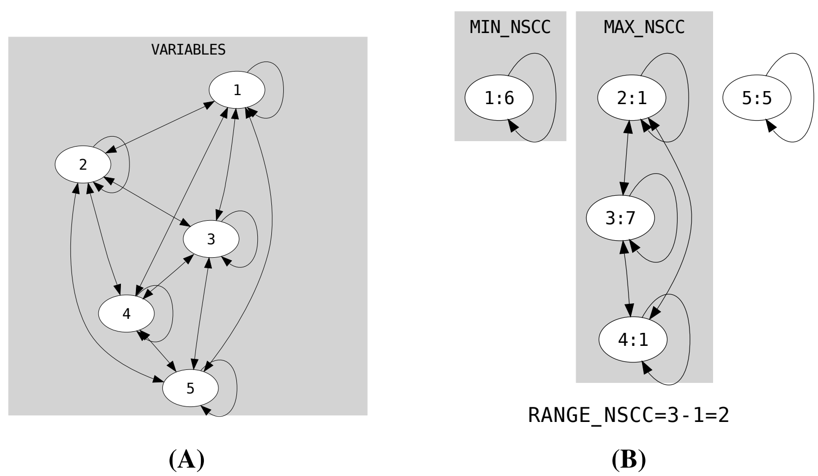


Figure 5.127: Initial and final graph of the BALANCE_MODULO constraint

Automaton Figure 5.128 depicts the automaton associated with the BALANCE_MODULO constraint. To each item of the collection `VARIABLES` corresponds a signature variable S_i that is equal to 1.

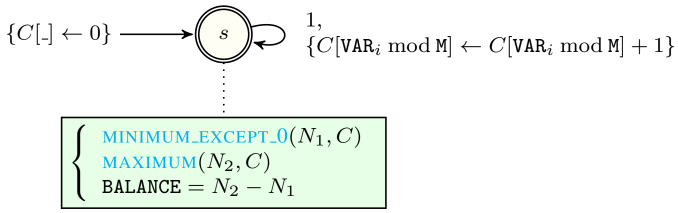


Figure 5.128: Automaton of the BALANCE_MODULO constraint

5.49 BALANCE_PARTITION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from BALANCE.

Constraint BALANCE_PARTITION(BALANCE, VARIABLES, PARTITIONS)

Type

VALUES : collection(val−int)

Arguments

BALANCE : dvar
VARIABLES : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

$|\mathtt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)
$\mathtt{BALANCE} \geq 0$
$\mathtt{BALANCE} \leq \mathtt{max}(0, |\mathtt{VARIABLES}| - 2)$
required(VARIABLES, var)
required(PARTITIONS, p)
$|\mathtt{PARTITIONS}| \geq 2$

Purpose

Consider the largest set $\mathcal{S}_1$ (respectively the smallest set $\mathcal{S}_2$) of variables of the collection VARIABLES that take their values in the same partition of the collection PARTITIONS.BALANCE is equal to the difference between the cardinality of $\mathcal{S}_2$ and the cardinality of $\mathcal{S}_1$.

Example

$$(1, \langle 6, 2, 6, 4, 4\rangle, \langle \mathtt{p} - \langle 1, 3\rangle, \mathtt{p} - \langle 4\rangle, \mathtt{p} - \langle 2, 6\rangle\rangle)$$

In this example values $6, 2, 6, 4, 4$ are respectively associated with the partitions $\mathtt{p} - \langle 2, 6\rangle$ and $\mathtt{p} - \langle 4\rangle$. Partitions $\mathtt{p} - \langle 4\rangle$ and $\mathtt{p} - \langle 2, 6\rangle$ are respectively used 2 and 3 times. The BALANCE_PARTITION constraint holds since its first argument BALANCE is assigned to the difference between the maximum and minimum number of the previous occurrences (i.e., $3 - 2$). Note that we do not consider those partitions that are not used at all.

Typical

$|\mathtt{VARIABLES}| > 2$
$|\mathtt{VARIABLES}| > |\mathtt{PARTITIONS}|$

Symmetries

- Items of VARIABLES are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Arg. properties Functional dependency: `BALANCE` determined by `VARIABLES` and `PARTITIONS`.

Usage An application of the BALANCE_PARTITION is to enforce a *balanced assignment* of values, no matter how many distinct partitions will be used. In this case one will *push down* the maximum value of the first argument of the BALANCE_PARTITION constraint.

See also **specialisation:** BALANCE *(*`variable` $\in$ `partition` *replaced by* `variable`*)*.

used in graph description: IN_SAME_PARTITION.

Keywords **application area:** assignment.

characteristic of a constraint: partition.

constraint arguments: pure functional dependency.

constraint type: value constraint.

final graph structure: equivalence.

modelling: balanced assignment, functional dependency.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	**RANGE_NSCC** = BALANCE
Graph class	EQUIVALENCE

Graph model The graph property **RANGE_NSCC** constraints the difference between the sizes of the largest and smallest strongly connected components.

Parts (A) and (B) of Figure 5.129 respectively show the initial and final graph associated with the **Example** slot. Since we use the **RANGE_NSCC** graph property, we show the largest and smallest strongly connected components of the final graph.

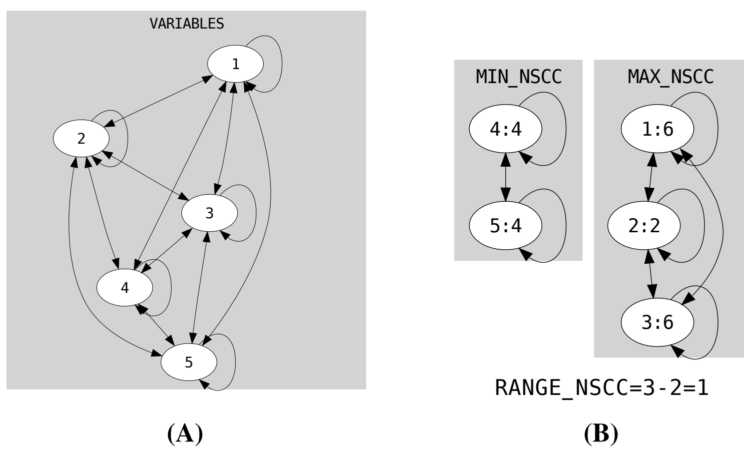


Figure 5.129: Initial and final graph of the BALANCE_PARTITION constraint

5.50 BALANCE_PATH

▶ ▷ ◁ ◀

DESCRIPTION LINKS GRAPH

Origin derived from BALANCE and PATH

Constraint BALANCE_PATH(BALANCE, NODES)

Arguments

BALANCE	:	dvar
NODES	:	collection(index−int, succ−dvar)

Restrictions

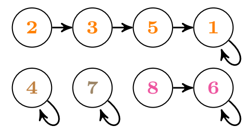
BALANCE ≥ 0
BALANCE $\leq$ max$(0, |$NODES$| - 2)$
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq |$NODES$|$

Purpose

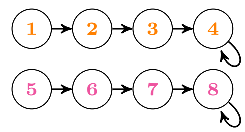
Consider a digraph G described by the NODES collection. Partition G into a set of vertex disjoint paths in such a way that each vertex of G belongs to a single path. BALANCE is equal to the difference between the number of vertices of the largest path and the number of vertices of the smallest path.

Example

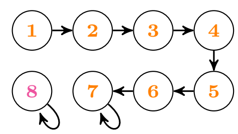
$$\left(\begin{array}{l}3,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{succ}-3,\\ \texttt{index}-3 & \texttt{succ}-5,\\ \texttt{index}-4 & \texttt{succ}-4,\\ \texttt{index}-5 & \texttt{succ}-1,\\ \texttt{index}-6 & \texttt{succ}-6,\\ \texttt{index}-7 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{succ}-6\end{array}\right\rangle\end{array}\right)$$



$$\left(\begin{array}{l}0,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-2,\\ \texttt{index}-2 & \texttt{succ}-3,\\ \texttt{index}-3 & \texttt{succ}-4,\\ \texttt{index}-4 & \texttt{succ}-4,\\ \texttt{index}-5 & \texttt{succ}-6,\\ \texttt{index}-6 & \texttt{succ}-7,\\ \texttt{index}-7 & \texttt{succ}-8,\\ \texttt{index}-8 & \texttt{succ}-8\end{array}\right\rangle\end{array}\right)$$



$$\left(\begin{array}{l}6,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-2,\\ \texttt{index}-2 & \texttt{succ}-3,\\ \texttt{index}-3 & \texttt{succ}-4,\\ \texttt{index}-4 & \texttt{succ}-5,\\ \texttt{index}-5 & \texttt{succ}-6,\\ \texttt{index}-6 & \texttt{succ}-7,\\ \texttt{index}-7 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{succ}-8\end{array}\right\rangle\end{array}\right)$$



In the first example we have the following four paths: $2 \rightarrow 3 \rightarrow 5 \rightarrow 1$, $8 \rightarrow 6$, 4, and 7. Since $\texttt{BALANCE} = 3$ is the difference between the number of vertices of the largest path (i.e., 4) and the number of vertices of the smallest path (i.e., 1) the corresponding BALANCE_PATH constraint holds.

All solutions

Figure 5.130 gives all solutions to the following non ground instance of the BALANCE_PATH constraint: $\texttt{BALANCE} = 0$, $\texttt{S}_1 \in [1,2]$, $\texttt{S}_2 \in [1,3]$, $\texttt{S}_3 \in [3,5]$, $\texttt{S}_4 \in [3,4]$, $\texttt{S}_5 \in [2,5]$, $\texttt{S}_6 \in [5,6]$, BALANCE_PATH$(\texttt{BALANCE}, \langle 1\ \texttt{S}_1, 2\ \texttt{S}_2, 3\ \texttt{S}_3, 4\ \texttt{S}_4, 5\ \texttt{S}_5, 6\ \texttt{S}_6\rangle)$.

Typical

$|\texttt{NODES}| > 2$

Symmetry

Items of NODES are permutable.

Arg. properties

Functional dependency: BALANCE determined by NODES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	13	73	501	4051	37633	394353

Number of solutions for BALANCE_PATH: domains $0..n$

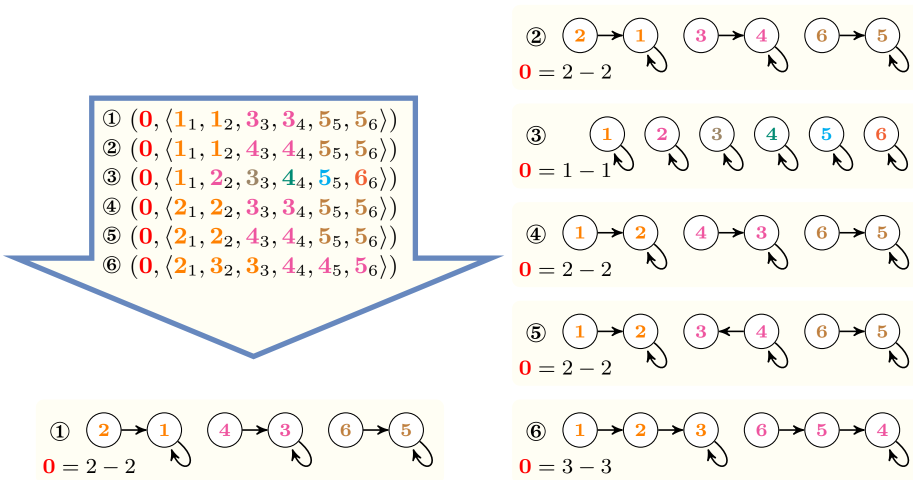
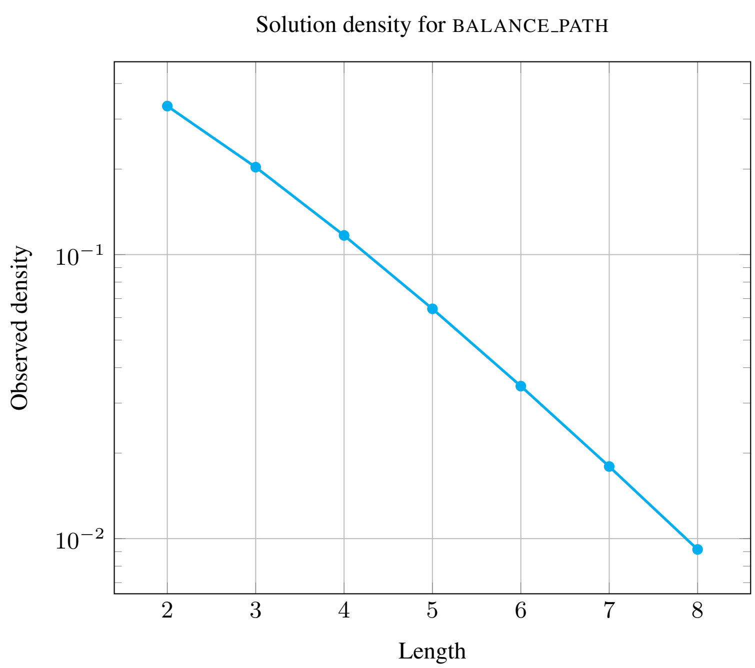
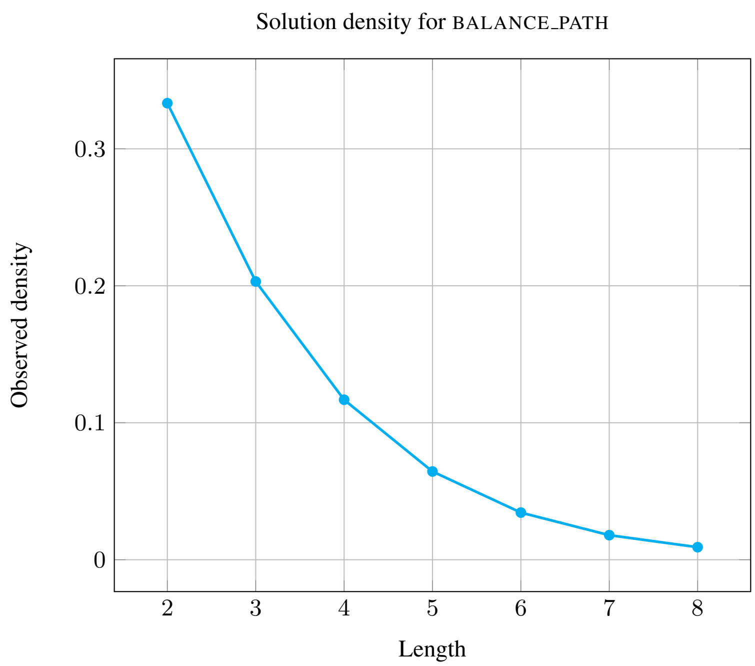


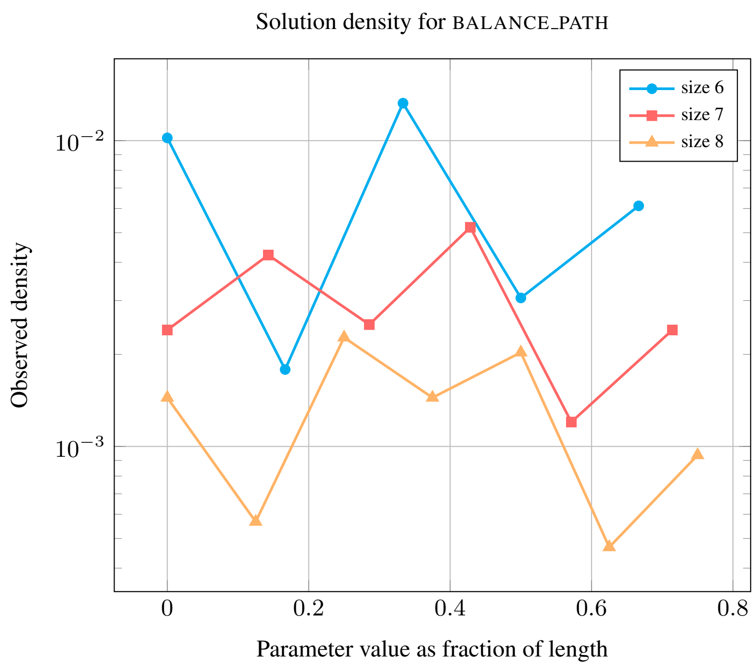
Figure 5.130: All solutions corresponding to the non ground example of the BALANCE_PATH constraint of the **All solutions** slot; the `index` attribute is displayed as indices of the `succ` attribute and all vertices of a same path are coloured by the same colour; the bottom left part of each subfigure shows how the `BALANCE` argument (in red) is related to the largest and to the smallest paths.

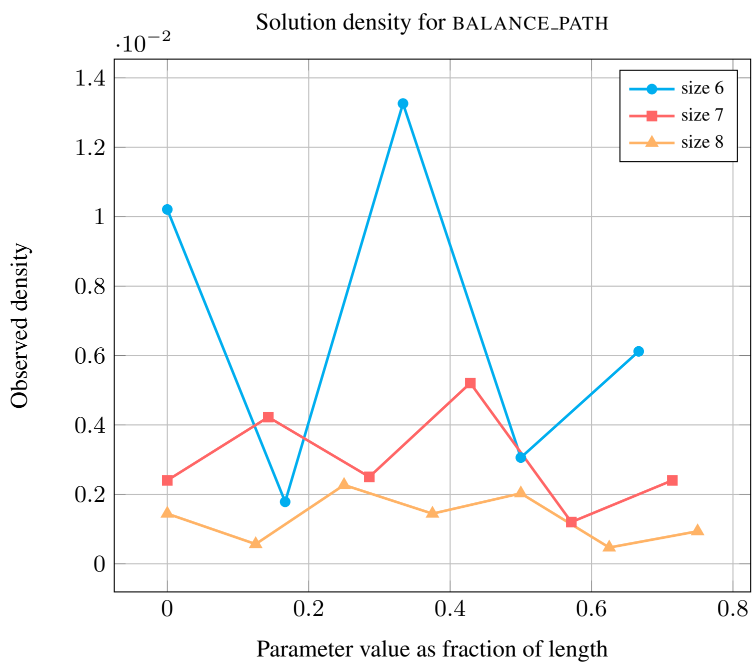




Length (n)		2	3	4	5	6	7	8
Total		3	13	73	501	4051	37633	394353
Parameter value	0	3	7	37	121	1201	5041	62161
	1	-	6	12	200	210	8862	24416
	2	-	-	24	60	1560	5250	97776
	3	-	-	-	120	360	10920	62160
	4	-	-	-	-	720	2520	87360
	5	-	-	-	-	-	5040	20160
	6	-	-	-	-	-	-	40320

Solution count for BALANCE_PATH: domains $0..n$





See also

implies: BALANCE_TREE.

related: BALANCE *(equivalence classes correspond to vertices in same path rather than variables assigned to the same value)*, PATH *(do not care how many paths but how balanced* *the paths are).*

Keywords

combinatorial object: path.

constraint type: graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck.

final graph structure: connected component, tree, one_succ.

modelling: functional dependency.

Arc input(s)	NODES
Arc generator	*CLIQUE* $\mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **MAX_ID** ≤ 1 • **RANGE_NCC** = BALANCE
Graph class	ONE_SUCC

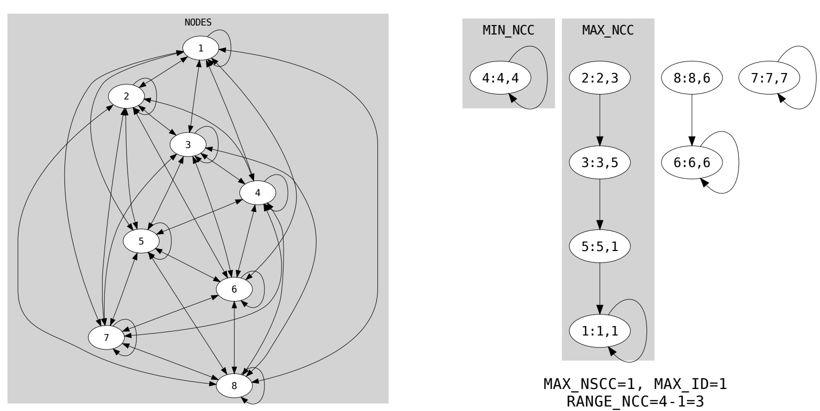
Graph model

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the BALANCE_PATH constraint considers objects that have two attributes:

- One fixed attribute index that is the identifier of the vertex,
- One variable attribute succ that is the successor of the vertex.

We use the graph property **MAX_NSCC** ≤ 1 in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with a single vertex. The graph property **MAX_ID** ≤ 1 constraints the maximum in-degree of the final graph to not exceed 1. **MAX_ID** does not consider loops: This is why we do not have any problem with the final node of each path.

Parts (A) and (B) of Figure 5.131 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **RANGE_NCC** graph property, we show the connected components of the final graph. The constraint holds since all the vertices belong to a path and since BALANCE = **RANGE_NCC** = 3.



(A) **(B)**

Figure 5.131: Initial and final graph of the BALANCE_PATH constraint

5.51 BALANCE_TREE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

derived from BALANCE and TREE

Constraint

BALANCE_TREE(BALANCE, NODES)

Arguments

BALANCE	:	dvar
NODES	:	collection(index−int, succ−dvar)

Restrictions

BALANCE ≥ 0
BALANCE $\leq$ max$(0, |$NODES$| - 2)$
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq |$NODES$|$

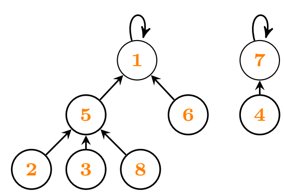
Purpose

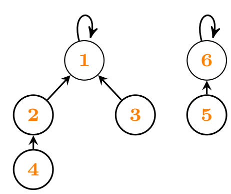
Consider a digraph G described by the NODES collection. Partition G into a set of vertex disjoint trees in such a way that each vertex of G belongs to a single tree. BALANCE is equal to the difference between the number of vertices of the largest tree and the number of vertices of the smallest tree.

Example

$$\left(\begin{array}{l} 4, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-5, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-7, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-1, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-5 \end{array} \right\rangle \end{array}\right)$$

$$\left(\begin{array}{l} 2, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-1, \\ \texttt{index}-3 & \texttt{succ}-1, \\ \texttt{index}-4 & \texttt{succ}-2, \\ \texttt{index}-5 & \texttt{succ}-6, \\ \texttt{index}-6 & \texttt{succ}-6 \end{array} \right\rangle \end{array}\right)$$





In the first example we have two trees involving respectively the set of vertices $\{1, 2, 3, 5, 6, 8\}$ and the set $\{4, 7\}$. They are depicted by Figure 5.132. Since BALANCE $= 6 - 2 = 4$ is the difference between the number of vertices of the largest tree (i.e., 6) and the number of vertices of the smallest tree (i.e., 2) the corresponding BALANCE_TREE constraint holds.

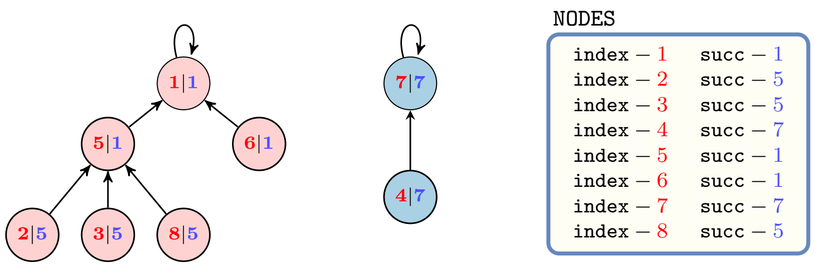


Figure 5.132: The two trees associated with the first example of the **Example** slot, respectively containing 6 and 2 vertices, therefore $\mathtt{BALANCE} = 6 - 2 = 4$; each vertex contains the information index|succ where succ is the index of its father in the tree (by convention the father of the root is the root itself).

All solutions

Figure 5.133 gives all solutions to the following non ground instance of the BALANCE_TREE constraint: $\mathtt{BALANCE} = 0$, $\mathtt{S}_1 \in [1,2]$, $\mathtt{S}_2 \in [1,2]$, $\mathtt{S}_3 \in [4,5]$, $\mathtt{S}_4 \in [2,4]$, $\mathtt{S}_5 \in [4,5]$, $\mathtt{S}_6 \in [5,6]$, BALANCE_TREE$(\mathtt{BALANCE}, \langle 1\,\mathtt{S}_1, 2\,\mathtt{S}_2, 3\,\mathtt{S}_3, 4\,\mathtt{S}_4, 5\,\mathtt{S}_5, 6\,\mathtt{S}_6\rangle)$.

Typical

$|\mathtt{NODES}| > 2$

Symmetry

Items of NODES are permutable.

Arg. properties

Functional dependency: BALANCE determined by NODES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	16	125	1296	16807	262144	4782969

Number of solutions for BALANCE_TREE: domains $0..n$

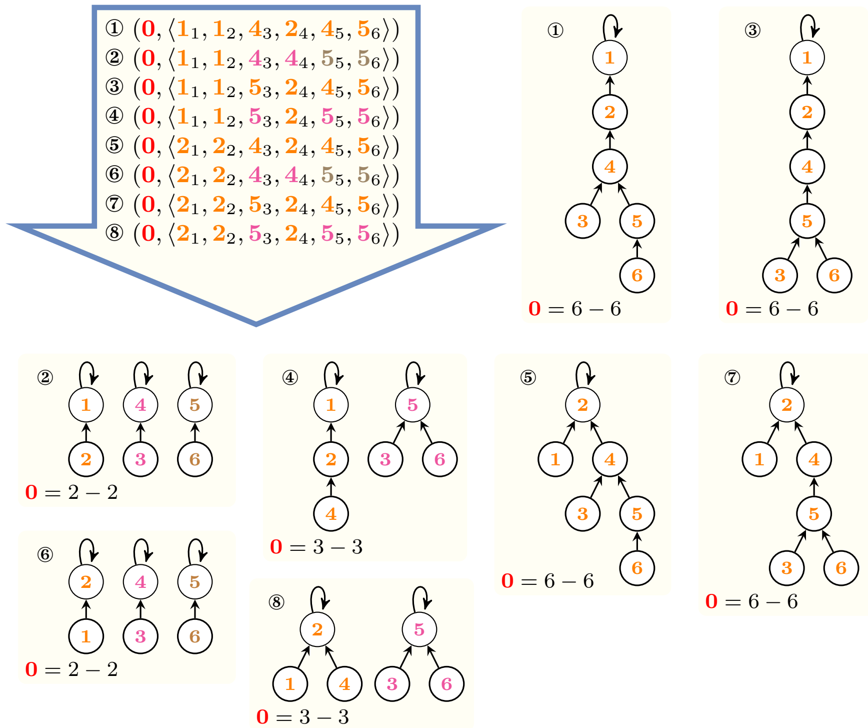
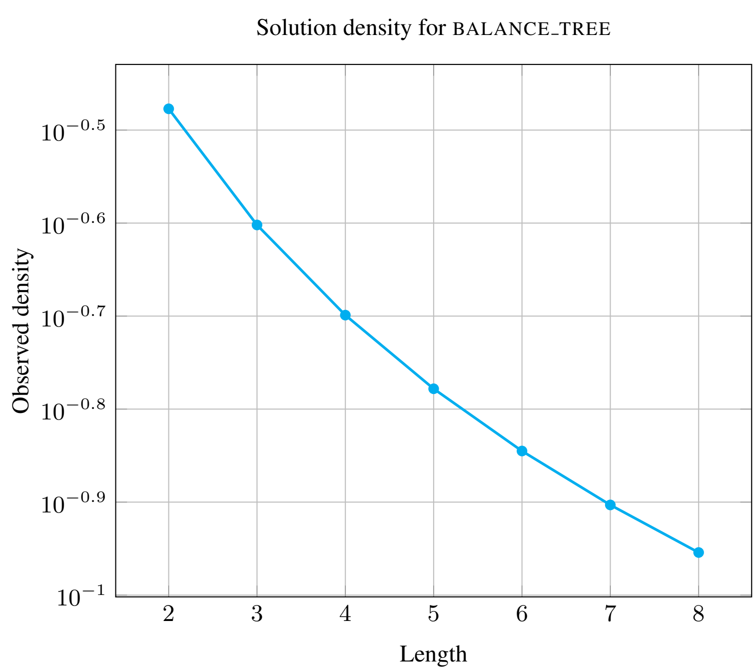
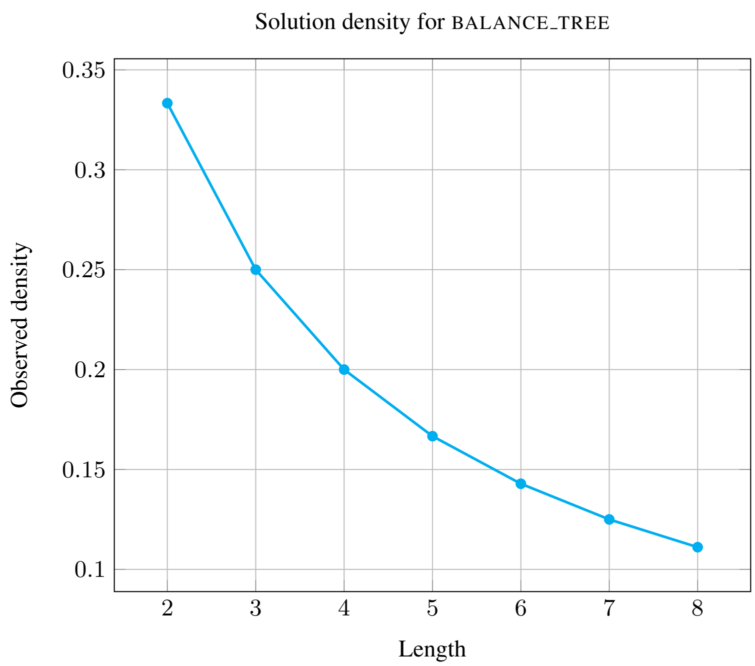


Figure 5.133: All solutions corresponding to the non ground example of the BALANCE_TREE constraint of the **All solutions** slot; the `index` attribute is displayed as indices of the `succ` attribute and all vertices of a same tree are coloured by the same colour; the bottom left part of each subfigure shows how the `BALANCE` argument (in red) is related to the largest and to the smallest trees.

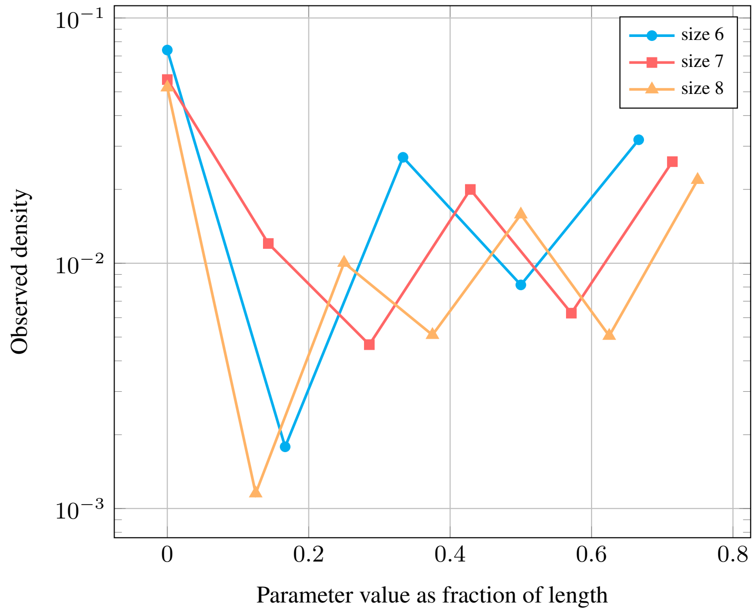




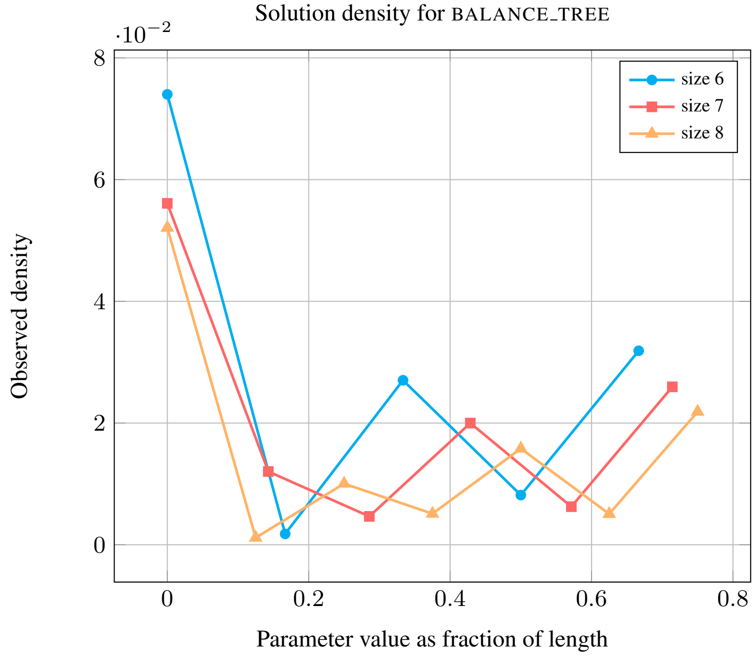
Length (n)		2	3	4	5	6	7	8
Total		3	16	125	1296	16807	262144	4782969
Parameter value	0	3	10	77	626	8707	117650	2242193
	1	-	6	12	260	210	25242	49616
	2	-	-	36	90	3180	9765	432264
	3	-	-	-	320	960	41930	219520
	4	-	-	-	-	3750	13125	680456
	5	-	-	-	-	-	54432	217728
	6	-	-	-	-	-	-	941192

Solution count for BALANCE_TREE: domains $0..n$

Solution density for BALANCE_TREE



Solution density for BALANCE_TREE



See also

implied by: BALANCE_PATH.

related: BALANCE *(equivalence classes correspond to vertices in same tree rather than variables assigned to the same value)*, TREE *(do not care how many trees but how balanced* *the trees are).*

Keywords

constraint type: graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck.

final graph structure: connected component, tree, one_succ.

modelling: functional dependency.

Cond. implications

BALANCE_TREE($\mathtt{BALANCE}, \mathtt{NODES}$)
with $\mathtt{BALANCE} > 0$
and $\mathtt{BALANCE} \leq |\mathtt{NODES}|$
implies ORDERED_ATLEAST_NVECTOR($\mathtt{NVEC} : \mathtt{BALANCE}, \mathtt{VECTORS} : \mathtt{NODES}$).

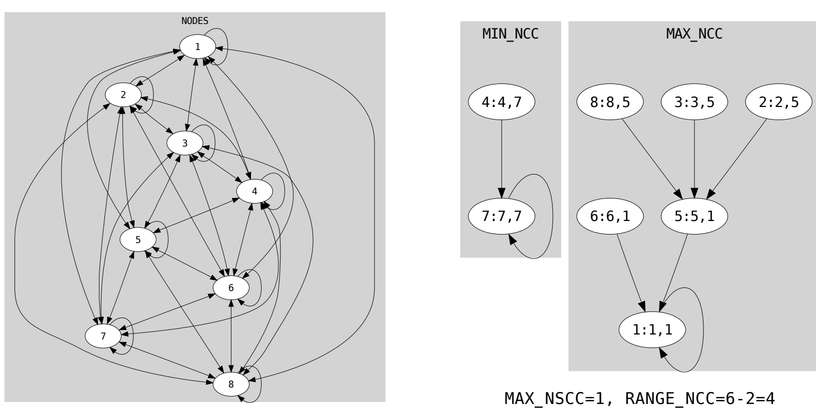
Arc input(s)	NODES
Arc generator	*CLIQUE* $\mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **RANGE_NCC** = BALANCE

Graph model In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the BALANCE_TREE constraint considers objects that have two attributes:

- One fixed attribute index that is the identifier of the vertex,
- One variable attribute succ that is the successor of the vertex.

We use the graph property **MAX_NSCC** ≤ 1 in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with a single vertex.

Parts (A) and (B) of Figure 5.134 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **RANGE_NCC** graph property, we show the connected components of the final graph. The constraint holds since all the vertices belong to a tree and since BALANCE = **RANGE_NCC**$6 - 2 = 4$.



(A) **(B)**

Figure 5.134: Initial and final graph of the BALANCE_TREE constraint

5.52 BETWEEN_MIN_MAX

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Used for defining CUMULATIVE_CONVEX.

Constraint BETWEEN_MIN_MAX(VAR, VARIABLES)

Arguments

VAR : dvar
VARIABLES : collection(var−dvar)

Restrictions

required(VARIABLES, var)
$|\texttt{VARIABLES}| > 0$

Purpose VAR is greater than or equal to at least one variable of the collection VARIABLES and less than or equal to at least one variable of the collection VARIABLES.

Example

$(3, \langle 1, 1, 4, 8\rangle)$
$(1, \langle 1, 1, 4, 8\rangle)$
$(8, \langle 1, 1, 4, 8\rangle)$

The first BETWEEN_MIN_MAX constraint holds since its first argument 3 is greater than or equal to the minimum value of the values of the collection $\langle 1, 1, 4, 8\rangle$ and less than or equal to the maximum value of $\langle 1, 1, 4, 8\rangle$.

Typical

$|\texttt{VARIABLES}| > 1$
range(VARIABLES.var) > 1

Symmetries

- Items of VARIABLES are permutable.
- VAR can be set to any value of VARIABLES.var.

Arg. properties Extensible wrt. VARIABLES.

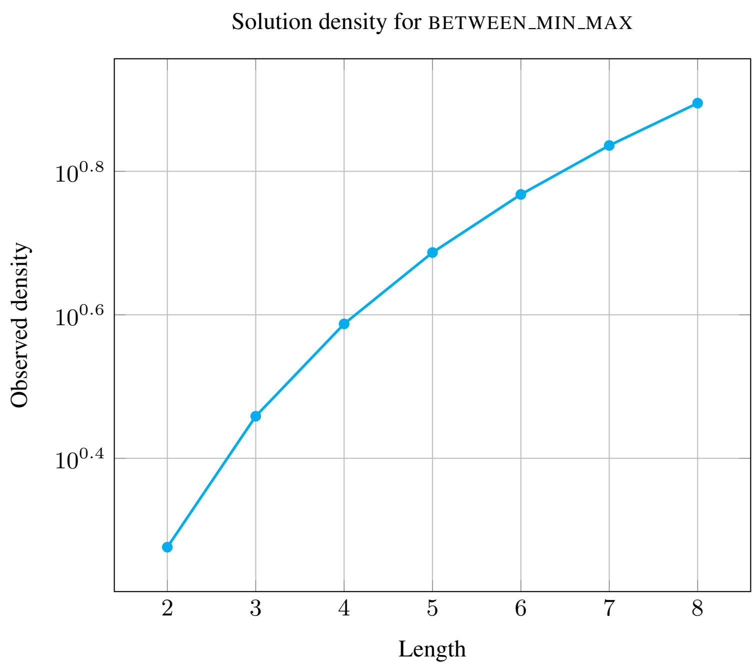
Reformulation By introducing two extra variables MIN and MAX, the BETWEEN_MIN_MAX(VAR, VARIABLES) constraint can be expressed in term of the following conjunction of constraints:

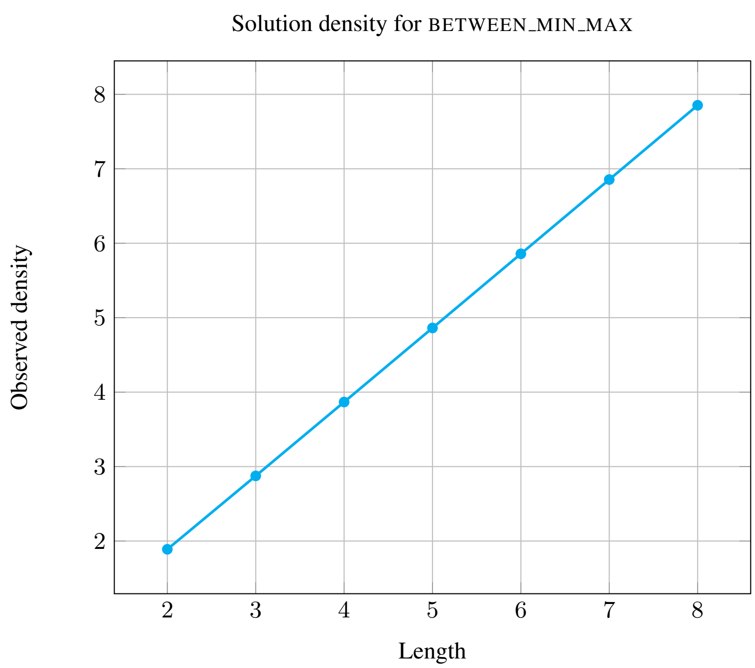
MINIMUM(MIN, VARIABLES),
MAXIMUM(MAX, VARIABLES),
VAR $\geq$ MIN,
VAR $\leq$ MAX.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	17	184	2417	37806	689201	14376608	338051265

Number of solutions for BETWEEN_MIN_MAX: domains $0..n$

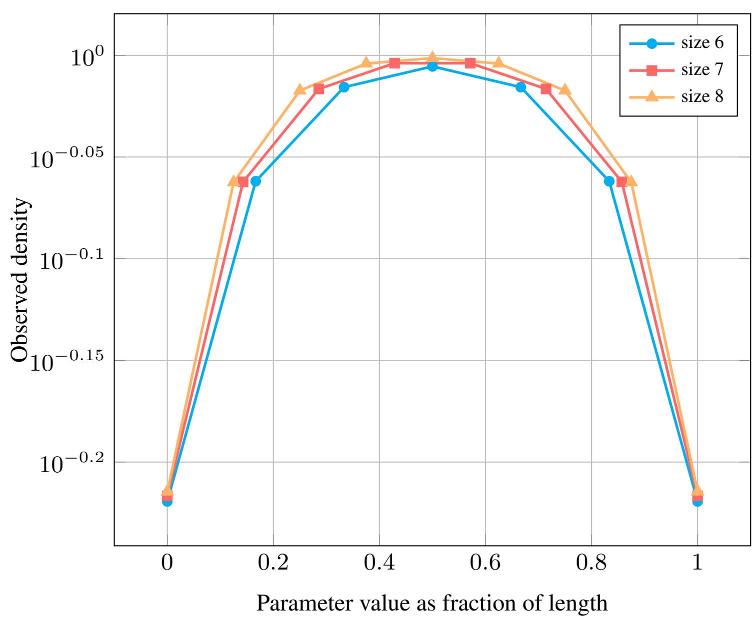

Solution density for BETWEEN_MIN_MAX
Observed density
$10^{0.8}$
$10^{0.6}$
$10^{0.4}$
2
3
4
5
6
7
8
Length


Solution density for BETWEEN_MIN_MAX
Observed density
8
7
6
5
4
3
2
2
3
4
5
6
7
8
Length

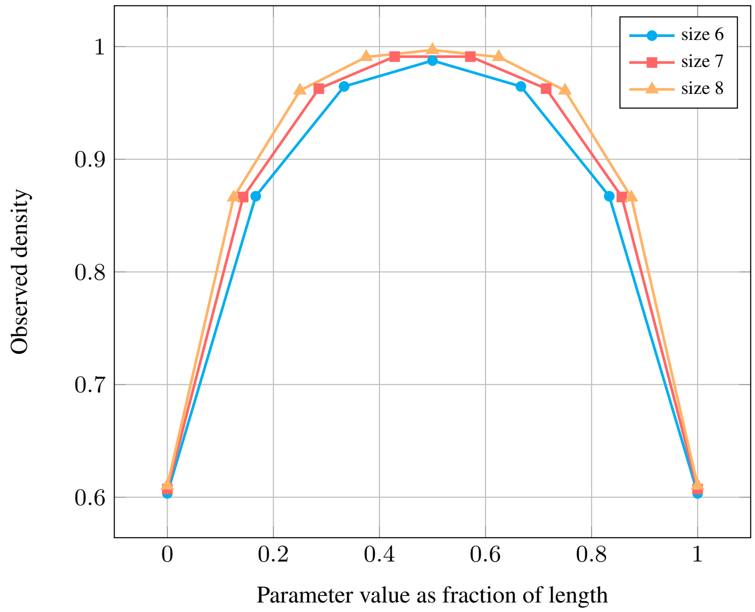
Length (n)		2	3	4	5	6	7	8
Total		17	184	2417	37806	689201	14376608	338051265
Parameter value	0	5	37	369	4651	70993	1273609	26269505
	1	7	55	543	6751	102023	1817215	37281919
	2	5	55	593	7501	113489	2018899	41366849
	3	-	37	543	7501	116191	2078581	42649535
	4	-	-	369	6751	113489	2078581	42915649
	5	-	-	-	4651	102023	2018899	42649535
	6	-	-	-	-	70993	1817215	41366849
	7	-	-	-	-	-	1273609	37281919
	8	-	-	-	-	-	-	26269505

Solution count for BETWEEN_MIN_MAX: domains $0..n$

Solution density for BETWEEN_MIN_MAX



Solution density for BETWEEN_MIN_MAX



Used in CUMULATIVE_CONVEX.

See also **implied by:** AND, DEEPEST_VALLEY, FIRST_VALUE_DIFF_0, HIGHEST_PEAK, IN, MAXIMUM, MINIMUM.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint network structure: centered cyclic(1) constraint network(1).

Derived Collection $\texttt{col}(\texttt{ITEM}-\texttt{collection}(\texttt{var}-\texttt{dvar}), [\texttt{item}(\texttt{var}-\texttt{VAR})])$

Arc input(s)	ITEM VARIABLES
Arc generator	$PRODUCT \mapsto \texttt{collection}(\texttt{item}, \texttt{variables})$
Arc arity	2
Arc constraint(s)	$\texttt{item.var} \geq \texttt{variables.var}$
Graph property(ies)	$\mathbf{NARC} \geq 1$
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Arc input(s)	ITEM VARIABLES
Arc generator	$PRODUCT \mapsto \texttt{collection}(\texttt{item}, \texttt{variables})$
Arc arity	2
Arc constraint(s)	$\texttt{item.var} \leq \texttt{variables.var}$
Graph property(ies)	$\mathbf{NARC} \geq 1$
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.135 respectively show the initial and final graph associated with the second graph constraint of the first example of the **Example** slot. Since we use the **NARC** graph property, the two arcs of the final graph are stressed in bold. The constraint holds since 3 is greater than 1 and since 3 is less than 8.

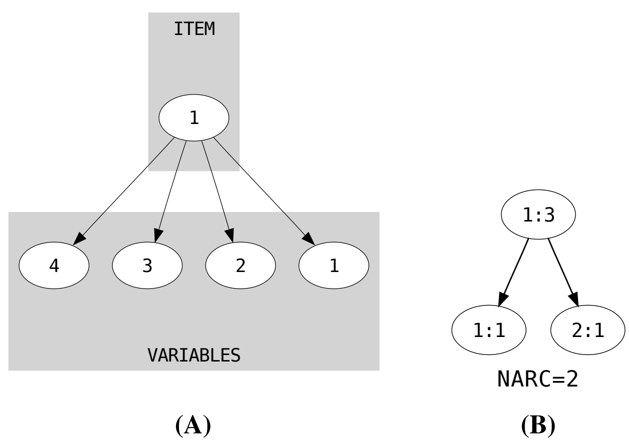


Figure 5.135: Initial and final graph of the BETWEEN_MIN_MAX constraint

Automaton Figure 5.136 depicts the automaton associated with the BETWEEN_MIN_MAX constraint. To each pair $(\mathtt{VAR}, \mathtt{VAR}_i)$, where $\mathtt{VAR}_i$ is a variable of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links VAR, $\mathtt{VAR}_i$ and S_i: $(\mathtt{VAR} < \mathtt{VAR}_i \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR} = \mathtt{VAR}_i \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR} > \mathtt{VAR}_i \Leftrightarrow S_i = 2)$.

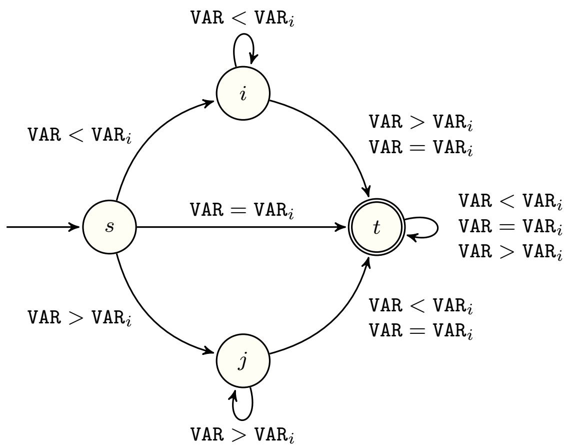


Figure 5.136: Automaton of the BETWEEN_MIN_MAX constraint

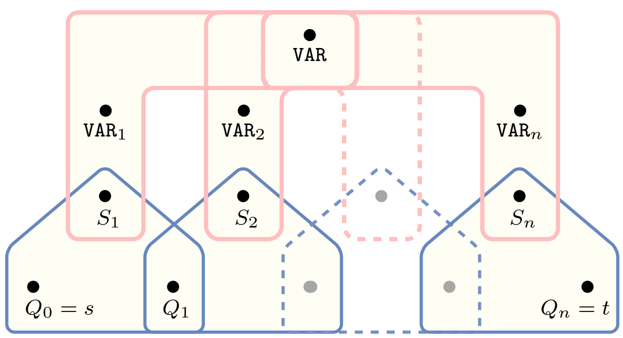


Figure 5.137: Hypergraph of the reformulation corresponding to the automaton of the BETWEEN_MIN_MAX constraint

5.53 BIG_PEAK

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin

Derived from PEAK.

Constraint

BIG_PEAK(N, VARIABLES, TOLERANCE)

Arguments

N	:	dvar
VARIABLES	:	collection(var−dvar)
TOLERANCE	:	int

Restrictions

$\texttt{N} \geq 0$
$2 * \texttt{N} \leq \texttt{max}(|\texttt{VARIABLES}| - 1, 0)$
required(VARIABLES, var)
$\texttt{TOLERANCE} \geq 0$

Purpose

A variable V_p $(1 < p < m)$ of the sequence of variables $\texttt{VARIABLES} = V_1, \ldots, V_m$ is a *peak* if and only if there exists an i $(1 < i \leq p)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \cdots = V_p$ and $V_p > V_{p+1}$. Similarly a variable V_v $(1 < k < m)$ is a *valley* if and only if there exists an i $(1 < i \leq v)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_v$ and $V_v < V_{v+1}$. A peak variable V_p $(1 < p < m)$ is a *potential big peak* wrt a non-negative integer TOLERANCE if and only if:

1. V_p is a peak,
2. $\exists i, j \in [1, m] \mid i < p < j$, V_i is a valley (or $i = 1$ if there is no valley before position p), V_j is a valley (or $i = m$ if there is no valley after position p), $V_p - V_i > \texttt{TOLERANCE}$, and $V_p - V_j > \texttt{TOLERANCE}$.

Let i_p and j_p be the largest i and the smallest j satisfying condition 2. Now a potential big peak V_p $(1 < p < m)$ is a *big peak* if and only if the interval $[i, j]$ does not contain any potential big peak that is strictly higher than V_p. The constraint BIG_PEAK holds if and only if N is the total number of big peaks of the sequence of variables VARIABLES.

Example

$$(7, \langle 4, 2, 2, 4, 3, 8, 6, 7, 7, 9, 5, 6, 3, 12, 12, 6, 6, 8, 4, 5, 1\rangle, 0)$$
$$(4, \langle 4, 2, 2, 4, 3, 8, 6, 7, 7, 9, 5, 6, 3, 12, 12, 6, 6, 8, 4, 5, 1\rangle, 1)$$

As shown part Part (A) of Figure 5.138, the first BIG_PEAK constraint holds since the sequence 4 2 2 4 3 8 6 7 7 9 5 6 3 12 12 6 6 8 4 5 1 contains seven big peaks wrt a tolerance of 0 (i.e., we consider standard peaks).

As shown part Part (B) of Figure 5.138, the second BIG_PEAK constraint holds since the same sequence 4 2 2 4 3 8 6 7 7 9 5 6 3 12 12 6 6 8 4 5 1 contains only four big peaks wrt a tolerance of 1.

Typical

$\texttt{N} \geq 1$
$|\texttt{VARIABLES}| > 6$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$\texttt{TOLERANCE} > 1$

Typical model

nval(VARIABLES.var) > 2
range(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: N determined by VARIABLES and TOLERANCE.
- Contractible wrt. VARIABLES when $\mathtt{N} = 0$ and $\mathtt{TOLERANCE} = 0$.

Usage

Useful for constraining the number of *big peaks* of a sequence of domain variables, by ignoring too small valleys that artificially create small peaks wrt TOLERANCE.

See also

specialisation: PEAK *(the tolerance is set to* 0 *and removed)*.

Keywords

characteristic of a constraint: automaton, automaton with counters.

combinatorial object: sequence.

constraint arguments: pure functional dependency.

modelling: functional dependency.

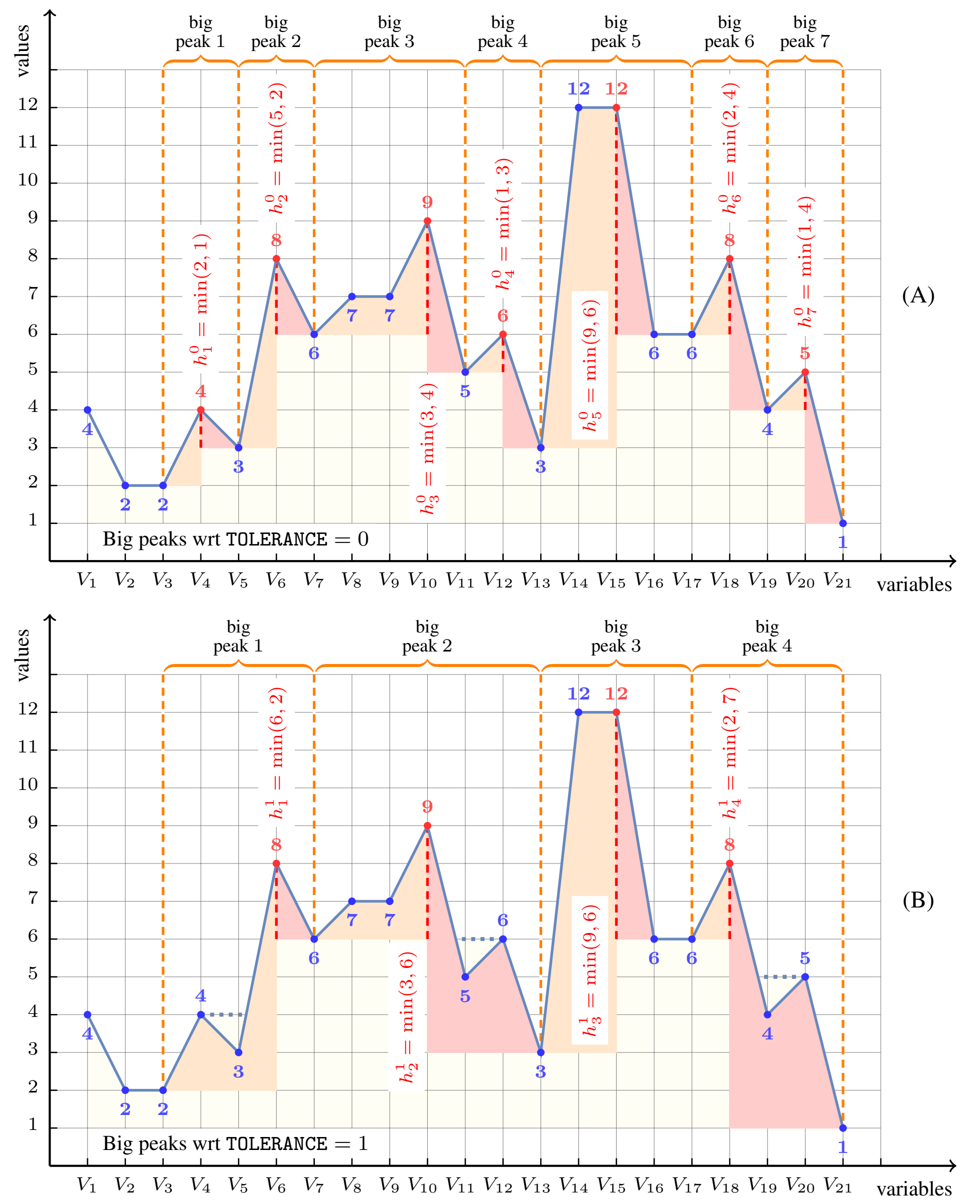


Figure 5.138: Illustration of the **Example** slot: Part (A) a sequence of 21 variables V_1, $V_2, \ldots, V_{21}$ respectively fixed to values 4, 2, 2, 4, 3, 8, 6, 7, 7, 9, 5, 6, 3, 12, 12, 6, 6, 8, 4, 5, 1 and its corresponding 7 peaks (`TOLERANCE` $= 0$ corresponds to standard peaks) with their respective heights $h_1^0 = 1$, $h_2^0 = 2$, $h_3^0 = 3$, $h_4^0 = 1$, $h_5^0 = 6$, $h_6^0 = 2$, $h_7^0 = 1$ (the left and right hand sides of each peak are coloured in light orange and light red) Part (B) the same sequence of variables and its 4 big peaks when `TOLERANCE` $= 1$ with their respective heights $h_1^1 = 2$, $h_2^1 = 3$, $h_3^1 = 6$, $h_4^1 = 2$

Automaton Figure 5.139 depicts the automaton associated with the BIG_PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

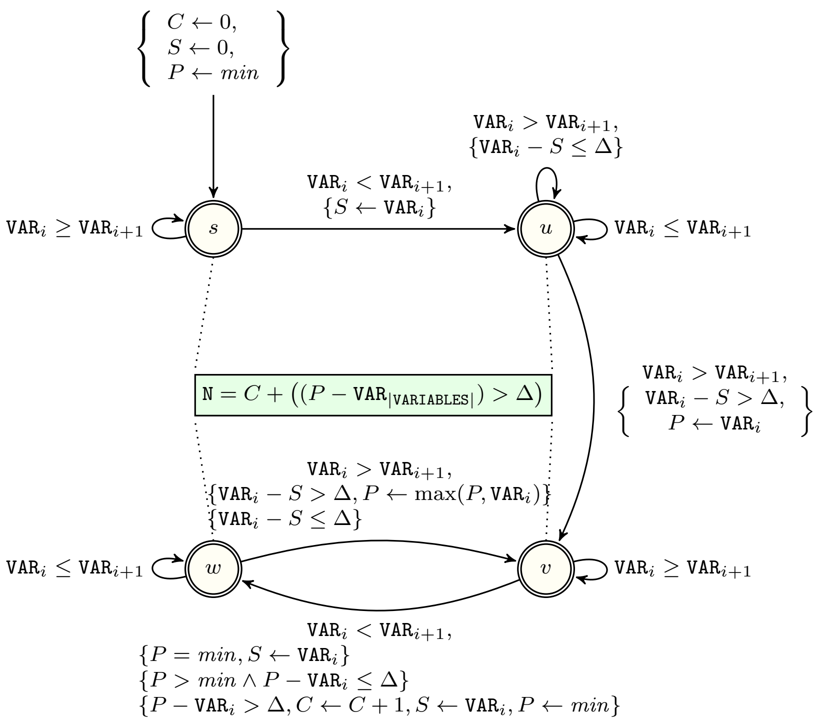


Figure 5.139: Automaton for the BIG_PEAK constraint where C, S, P, min and Δ respectively stand for the number of big peaks already encountered, the altitude at the start of the current potential big peak, the altitude of the current potential big peak, the smallest value that can be assigned to a variable of VARIABLES, the TOLERANCE parameter

5.54 BIG_VALLEY

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Derived from VALLEY.

Constraint BIG_VALLEY(N, VARIABLES, TOLERANCE)

Arguments

N	:	dvar
VARIABLES	:	collection(var−dvar)
TOLERANCE	:	int

Restrictions

$\texttt{N} \geq 0$
$2 * \texttt{N} \leq \texttt{max}(|\texttt{VARIABLES}| - 1, 0)$
required(VARIABLES, var)
$\texttt{TOLERANCE} \geq 0$

Purpose

A variable V_v $(1 < k < m)$ is a *valley* if and only if there exists an i $(1 < i \leq v)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_v$ and $V_v < V_{v+1}$. Similarly a variable V_p $(1 < p < m)$ of the sequence of variables $\texttt{VARIABLES} = V_1, \ldots, V_m$ is a *peak* if and only if there exists an i $(1 < i \leq p)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \cdots = V_p$ and $V_p > V_{p+1}$. A valley variable V_v $(1 < v < m)$ is a *potential big valley* wrt a non-negative integer TOLERANCE if and only if:

1. V_v is a valley,
2. $\exists i, j \in [1, m] \mid i < v < j$, V_i is a peak (or $i = 1$ if there is no peak before position p), V_j is a peak (or $i = m$ if there is no peak after position p), $V_i - V_v > \texttt{TOLERANCE}$, and $V_j - V_v > \texttt{TOLERANCE}$.

Let i_v and j_v be the largest i and the smallest j satisfying condition 2. Now a potential big valley V_v $(1 < v < m)$ is a *big valley* if and only if the interval $[i, j]$ does not contain any potential big valley that is strictly less than V_v. The constraint BIG_VALLEY holds if and only if N is the total number of big valleys of the sequence of variables VARIABLES.

Example

$(7, \langle 9, 11, 11, 9, 10, 5, 7, 6, 6, 4, 8, 7, 10, 1, 1, 7, 7, 5, 9, 8, 12\rangle, 0)$
$(4, \langle 9, 11, 11, 9, 10, 5, 7, 6, 6, 4, 8, 7, 10, 1, 1, 7, 7, 5, 9, 8, 12\rangle, 1)$

As shown part Part (A) of Figure 5.140, the first BIG_VALLEY constraint holds since the sequence 9 11 11 9 10 5 7 6 6 4 8 7 10 1 1 7 7 5 9 8 12 contains seven big valleys wrt a tolerance of 0 (i.e., we consider standard valleys).

As shown part Part (B) of Figure 5.140, the second BIG_VALLEY constraint holds since the same sequence 9 11 11 9 10 5 7 6 6 4 8 7 10 1 1 7 7 5 9 8 12 contains only four big valleys wrt a tolerance of 1.

Typical

$\texttt{N} \geq 1$
$|\texttt{VARIABLES}| > 6$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$\texttt{TOLERANCE} > 1$

Typical model

nval(VARIABLES.var) > 2
range(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: N determined by VARIABLES and TOLERANCE.
- Contractible wrt. VARIABLES when $\mathtt{N} = 0$ and $\mathtt{TOLERANCE} = 0$.

Usage

Useful for constraining the number of *big valleys* of a sequence of domain variables, by ignoring too small peaks that artificially create small valleys wrt TOLERANCE.

See also

specialisation: VALLEY *(the tolerance is set to* 0 *and removed)*.

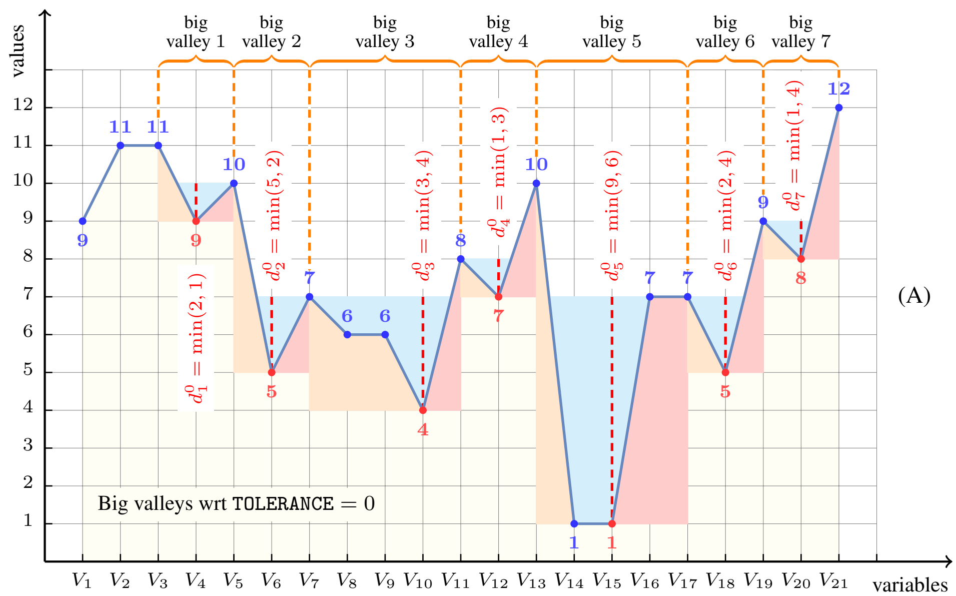
Keywords

characteristic of a constraint: automaton, automaton with counters.

combinatorial object: sequence.

constraint arguments: pure functional dependency.

modelling: functional dependency.



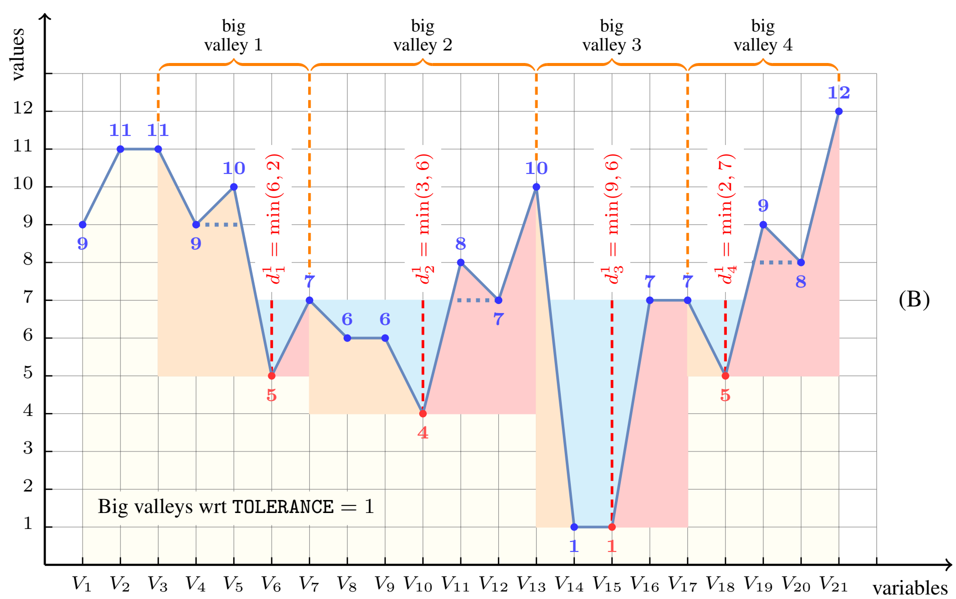


Figure 5.140: Illustration of the **Example** slot: Part (A) a sequence of 21 variables $V_1, V_2, \ldots, V_{21}$ respectively fixed to values 9, 11, 11, 9, 10, 5, 7, 6, 6, 4, 8, 7, 10, 1, 1, 7, 7, 5, 9, 8, 12 and its corresponding 7 valleys (`TOLERANCE` $= 0$ corresponds to standard valleys) with their respective depths $d_1^0 = 1$, $d_2^0 = 2$, $d_3^0 = 3$, $d_4^0 = 1$, $d_5^0 = 6$, $d_6^0 = 2$, $d_7^0 = 1$ (the left and right hand sides of each valley are coloured in light orange and light red) Part (B) the same sequence of variables and its 4 big valleys when `TOLERANCE` $= 1$ with their respective depths $d_1^1 = 2$, $d_2^1 = 3$, $d_3^1 = 6$, $d_4^1 = 2$

Automaton Figure 5.141 depicts the automaton associated with the BIG_VALLEY constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

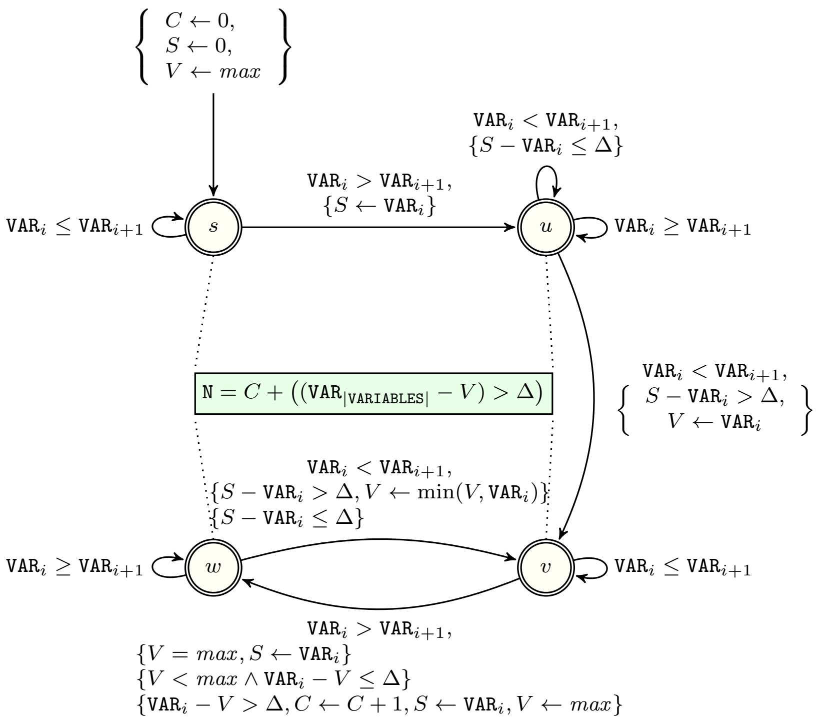


Figure 5.141: Automaton for the BIG_VALLEY where C, S, V, max and Δ respectively stand for the number of big valleys already encountered, the altitude at the start of the current potential big valley, the altitude of the current potential big valley, the largest value that can be assigned to a variable of VARIABLES, the TOLERANCE parameter

5.55 BIN_PACKING

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from CUMULATIVE.

Constraint BIN_PACKING(CAPACITY, ITEMS)

Arguments

CAPACITY	:	int
ITEMS	:	collection(bin−dvar, weight−int)

Restrictions

CAPACITY ≥ 0
required(ITEMS, [bin, weight])
ITEMS.weight ≥ 0
ITEMS.weight $\leq$ CAPACITY

Purpose

Given several items of the collection ITEMS (each of them having a specific weight), and different bins of a fixed capacity, assign each item to a bin so that the total weight of the items in each bin does not exceed CAPACITY.

Example

$$\left(5, \left\langle \begin{array}{ll} \texttt{bin}-3 & \texttt{weight}-4, \\ \texttt{bin}-1 & \texttt{weight}-3, \\ \texttt{bin}-3 & \texttt{weight}-1 \end{array} \right\rangle \right)$$

The BIN_PACKING constraint holds since the sum of the height of items that are assigned to bins 1 and 3 is respectively equal to 3 and 5. The previous quantities are both less than or equal to the maximum CAPACITY 5. Figure 5.142 shows the solution associated with the example.

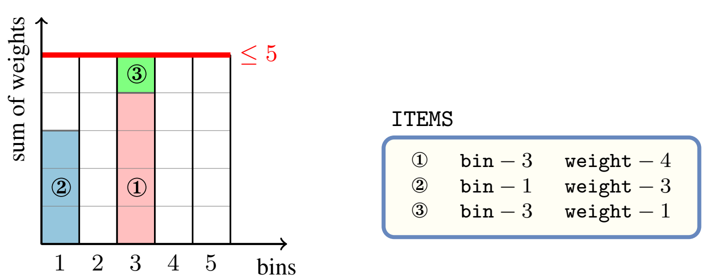


Figure 5.142: Bin-packing solution to the **Example** slot

Typical

CAPACITY $>$maxval(ITEMS.weight)
CAPACITY $\leq$sum(ITEMS.weight)
$|$ITEMS$| > 1$
range(ITEMS.bin) > 1
range(ITEMS.weight) > 1
ITEMS.bin ≥ 0
ITEMS.weight > 0

Symmetries

- CAPACITY can be increased.
- Items of ITEMS are permutable.
- ITEMS.weight can be decreased to any value ≥ 0.
- All occurrences of two distinct values of ITEMS.bin can be swapped; all occurrences of a value of ITEMS.bin can be renamed to any unused value.

Arg. properties

Contractible wrt. ITEMS.

Remark

Note the difference from the *classical* bin-packing problem [286, page 221] where one wants to find solutions that minimise the number of bins. In our case each item may be assigned only to specific bins (i.e., the different values of the bin variable) and the goal is to find a feasible solution. This constraint can be seen as a special case of the CUMULATIVE constraint [1], where all task durations are equal to 1.

In [390] the CAPACITY parameter of the BIN_PACKING constraint is replaced by a collection of domain variables representing the *load* of each bin (i.e., the sum of the weights of the items assigned to a bin). This allows representing problems where a minimum level has to be reached in each bin.

Coffman *and al.* give in [128] the worst case bounds of different list algorithms for the bin packing problem (i.e., given a positive integer CAPACITY and a list L of integer sizes $\mathtt{weight}_1, \mathtt{weight}_2, \ldots, \mathtt{weight}_n$ ($0 \leq \mathtt{weight}_i \leq$ CAPACITY), what is the smallest integer m such that there is a partition $L = L_1 \cup L_2 \cup \cdots \cup L_m$ satisfying $\sum_{\mathtt{weight}_i \in L_j} \mathtt{weight}_i \leq$ CAPACITY for all $j \in [1, m]$?).

Algorithm

Initial filtering algorithms are described in [302, 299, 300, 301, 390]. More recently, linear continuous relaxations based on the graph associated with the dynamic programming approach for knapsack by Trick [419], and on the more compact model introduced by Carvalho [110, 111] are presented in [98].

Systems

PACK in **Choco**, BINPACKING in **Gecode**, BIN_PACKING in **MiniZinc**.

See also

generalisation: BIN_PACKING_CAPA *(fixed overall capacity replaced by non-fixed capacity)*, CUMULATIVE *(task of* duration 1 *replaced by task of given* duration*)*, CUMULATIVE_TWO_D *(*task *of* duration 1 *replaced by* square *of size* 1 *with a* height*)*, INDEXED_SUM *(negative contribution also allowed, fixed capacity replaced by a set of variables)*.

used in graph description: SUM_CTR.

Keywords

application area: assignment.

characteristic of a constraint: automaton, automaton with array of counters.

constraint type: resource constraint.

final graph structure: acyclic, bipartite, no loop.

modelling: assignment dimension, assignment to the same set of values.

modelling exercises: assignment to the same set of values.

Cond. implications

BIN_PACKING($\texttt{CAPACITY}, \texttt{ITEMS}$)
with $\texttt{CAPACITY} \geq |\texttt{ITEMS}|$
implies ATMOST_NVECTOR($\texttt{NVEC} : \texttt{CAPACITY}, \texttt{VECTORS} : \texttt{ITEMS}$).

Arc input(s)	ITEMS ITEMS
Arc generator	*PRODUCT* $\mapsto$ collection(items1, items2)
Arc arity	2
Arc constraint(s)	items1.bin = items2.bin
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP
Sets	SUCC $\mapsto$ [source, variables − col (VARIABLES−collection(var−dvar), [item(var − ITEMS.weight)])]
Constraint(s) on sets	SUM_CTR(variables, ≤, CAPACITY)

Graph model We enforce the SUM_CTR constraint on the weight of the items that are assigned to the same bin.

Parts (A) and (B) of Figure 5.143 respectively show the initial and final graph associated with the **Example** slot. Each connected component of the final graph corresponds to the items that are all assigned to the same bin.

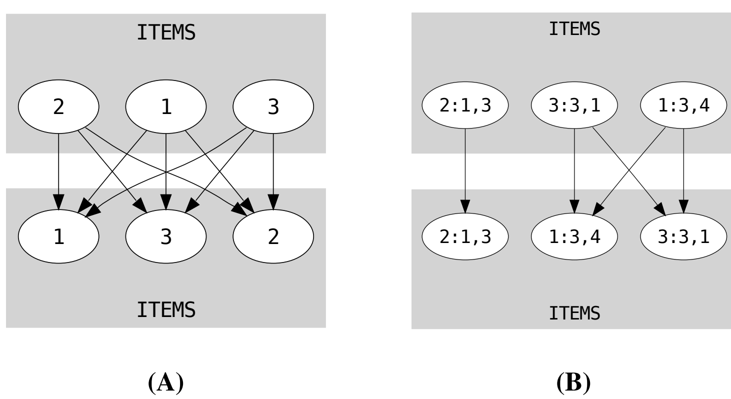


Figure 5.143: Initial and final graph of the BIN_PACKING constraint

Automaton Figure 5.144 depicts the automaton associated with the BIN_PACKING constraint. To each item of the collection ITEMS corresponds a signature variable S_i that is equal to 1.

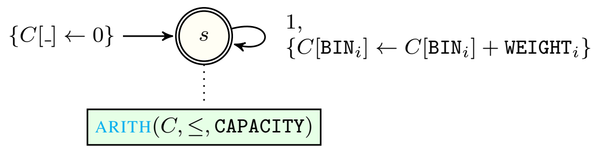


Figure 5.144: Automaton of the BIN_PACKING constraint

5.56 BIN_PACKING_CAPA

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Derived from BIN_PACKING.

Constraint BIN_PACKING_CAPA(BINS, ITEMS)

Arguments

$$\begin{array}{lll} \texttt{BINS} & : & \texttt{collection}(\texttt{id}-\texttt{int}, \texttt{capa}-\texttt{int}) \\ \texttt{ITEMS} & : & \texttt{collection}(\texttt{bin}-\texttt{dvar}, \texttt{weight}-\texttt{int}) \end{array}$$

Restrictions

$|\texttt{BINS}| > 0$
$\texttt{required}(\texttt{BINS}, [\texttt{id}, \texttt{capa}])$
$\texttt{distinct}(\texttt{BINS}, \texttt{id})$
$\texttt{BINS.id} \geq 1$
$\texttt{BINS.id} \leq |\texttt{BINS}|$
$\texttt{BINS.capa} \geq 0$
$\texttt{required}(\texttt{ITEMS}, [\texttt{bin}, \texttt{weight}])$
$\texttt{in\_attr}(\texttt{ITEMS}, \texttt{bin}, \texttt{BINS}, \texttt{id})$
$\texttt{ITEMS.weight} \geq 0$

Purpose

Given several items of the collection ITEMS (each of them having a specific weight), and different bins described the the items of collection BINS (each of them having a specific capacity capa), assign each item to a bin so that the total weight of the items in each bin does not exceed the capacity of the bin.

Example

$$\left(\begin{array}{l} \left\langle\begin{array}{ll} \texttt{id}-1 & \texttt{capa}-4, \\ \texttt{id}-2 & \texttt{capa}-3, \\ \texttt{id}-3 & \texttt{capa}-5, \\ \texttt{id}-4 & \texttt{capa}-3, \\ \texttt{id}-5 & \texttt{capa}-3 \end{array}\right\rangle, \\ \left\langle\begin{array}{ll} \texttt{bin}-3 & \texttt{weight}-4, \\ \texttt{bin}-1 & \texttt{weight}-3, \\ \texttt{bin}-3 & \texttt{weight}-1 \end{array}\right\rangle \end{array}\right)$$

The BIN_PACKING_CAPA constraint holds since the sum of the height of items that are assigned to bins 1 and 3 is respectively equal to 3 and 5. The previous quantities are respectively less than or equal to the maximum capacities 4 and 5 of bins 1 and 3. Figure 5.145 shows the solution associated with the example.

Typical

$|\texttt{BINS}| > 1$
$\texttt{range}(\texttt{BINS.capa}) > 1$
$\texttt{BINS.capa} > \texttt{maxval}(\texttt{ITEMS.weight})$
$\texttt{BINS.capa} \leq \texttt{sum}(\texttt{ITEMS.weight})$
$|\texttt{ITEMS}| > 1$
$\texttt{range}(\texttt{ITEMS.bin}) > 1$
$\texttt{range}(\texttt{ITEMS.weight}) > 1$
$\texttt{ITEMS.weight} > 0$

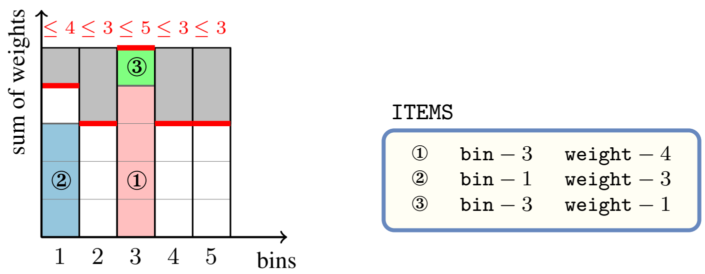


Figure 5.145: Bin-packing solution to the **Example** slot

Symmetries

- Items of BINS are permutable.
- Items of ITEMS are permutable.
- BINS.capa can be increased.
- ITEMS.weight can be decreased to any value ≥ 0.
- All occurrences of two distinct values in BINS.id or ITEMS.bin can be swapped; all occurrences of a value in BINS.id or ITEMS.bin can be renamed to any unused value.

Arg. properties

Contractible wrt. ITEMS.

Remark

In **MiniZinc** (http://www.minizinc.org/) there is also a constraint called BIN_PACKING_LOAD which, for each bin has a domain variable that is equal to the sum of the weights assigned to the corresponding bin.

Systems

PACK in **Choco**, BINPACKING in **Gecode**, BIN_PACKING_CAPA in **MiniZinc**.

See also

generalisation: INDEXED_SUM *(negative contribution also allowed)*.

specialisation: BIN_PACKING *(non-fixed capacity replaced by fixed overall capacity)*.

Keywords

application area: assignment.

constraint type: predefined constraint, resource constraint.

modelling: assignment dimension, assignment to the same set of values.

modelling exercises: assignment to the same set of values.

5.57 BINARY_TREE

▶ ▷ ◁ ◀

DESCRIPTION LINKS GRAPH

Origin Derived from TREE.

Constraint BINARY_TREE(NTREES, NODES)

Arguments

NTREES : dvar
NODES : collection(index−int, succ−dvar)

Restrictions

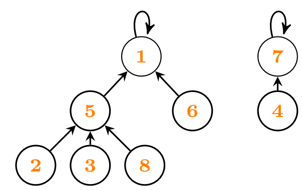
NTREES ≥ 0
NTREES $\leq$ |NODES|
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

Purpose

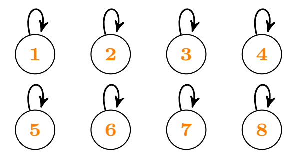
Cover the digraph G described by the NODES collection with NTREES binary trees in such a way that each vertex of G belongs to exactly one binary tree (i.e., each vertex of G has at most two children). The edges of the binary trees are directed from their leaves to their respective roots.

Example

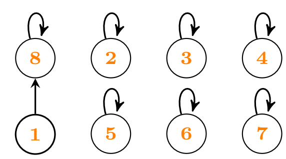
$$\left(\begin{array}{l} 2, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-3, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-7, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-1, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-5 \end{array} \right\rangle \end{array}\right)$$



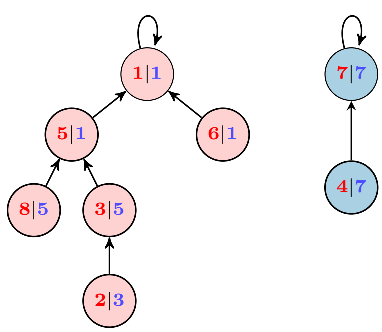
$$\left(\begin{array}{l} 8, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-2, \\ \texttt{index}-3 & \texttt{succ}-3, \\ \texttt{index}-4 & \texttt{succ}-4, \\ \texttt{index}-5 & \texttt{succ}-5, \\ \texttt{index}-6 & \texttt{succ}-6, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-8 \end{array} \right\rangle \end{array}\right)$$



$$\left(\begin{array}{l} 7, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-8, \\ \texttt{index}-2 & \texttt{succ}-2, \\ \texttt{index}-3 & \texttt{succ}-3, \\ \texttt{index}-4 & \texttt{succ}-4, \\ \texttt{index}-5 & \texttt{succ}-5, \\ \texttt{index}-6 & \texttt{succ}-6, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-8 \end{array} \right\rangle \end{array}\right)$$



The first BINARY_TREE constraint holds since its second argument corresponds to the 2 (i.e., the first argument of the first BINARY_TREE constraint) binary trees depicted by Figure 5.146.



NODES

index − 1	succ − 1
index − 2	succ − 3
index − 3	succ − 5
index − 4	succ − 7
index − 5	succ − 1
index − 6	succ − 1
index − 7	succ − 7
index − 8	succ − 5

Figure 5.146: The two binary trees corresponding to the first example of the **Example** slot; each vertex contains the information `index`|`succ` where `succ` is the index of its father in the tree (by convention the father of the root is the root itself).

All solutions

Figure 5.147 gives all solutions to the following non ground instance of the BINARY_TREE constraint: $\texttt{NTREES} \in \{1,4\}$, $\texttt{S}_1 \in [1,2]$, $\texttt{S}_2 \in [1,3]$, $\texttt{S}_3 \in [3,4]$, $\texttt{S}_4 \in [3,4]$, $\texttt{S}_5 \in [2,3]$, BINARY_TREE$(\texttt{NTREES}, \langle 1\ \texttt{S}_1, 2\ \texttt{S}_2, 3\ \texttt{S}_3, 4\ \texttt{S}_4, 5\ \texttt{S}_5\rangle)$.

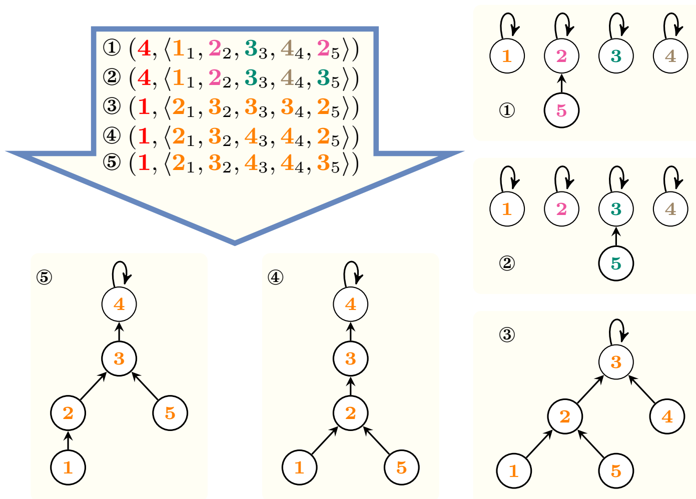


Figure 5.147: All solutions corresponding to the non ground example of the BINARY_TREE constraint of the **All solutions** slot, where all vertices of a same tree are coloured by the same colour; in the left-hand side the `index` attributes ares displayed as indices of the `succ` attribute, while in the right-hand side they are directly displayed within each node.

Typical

$\texttt{NTREES} > 0$
$\texttt{NTREES} < |\texttt{NODES}|$
$|\texttt{NODES}| > 2$

Symmetry

Items of NODES are permutable.

Arg. properties

Functional dependency: NTREES determined by NODES.

Reformulation

The BINARY_TREE constraint can be expressed in term of (1) a set of $|\texttt{NODES}|^2$ reified constraints for avoiding circuit between more than one node and of (2) $|\texttt{NODES}|$ reified constraints and of one sum constraint for counting the trees and of (3) a set of $|\texttt{NODES}|^2$ reified constraints and of $|\texttt{NODES}|$ inequalities constraints for enforcing the fact that each vertex has at most two children.

1. For each vertex $\texttt{NODES}[i]$ $(i \in [1, |\texttt{NODES}|])$ of the NODES collection we create a variable R_i that takes its value within interval $[1, |\texttt{NODES}|]$. This variable represents the *rank* of vertex $\texttt{NODES}[i]$ within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices $\texttt{NODES}[i], \texttt{NODES}[j]$ $(i, j \in [1, |\texttt{NODES}|])$ of the NODES collection we create a reified constraint of the form $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Rightarrow R_i < R_j$. The purpose of this constraint is to express the fact that, if there is an arc from vertex $\texttt{NODES}[i]$ to another vertex $\texttt{NODES}[j]$, then R_i should be strictly less than R_j.
2. For each vertex $\texttt{NODES}[i]$ $(i \in [1, |\texttt{NODES}|])$ of the NODES collection we create a 0-1 variable B_i and state the following reified constraint $\texttt{NODES}[i].\texttt{succ} =$

$\texttt{NODES}[i].\texttt{index} \Leftrightarrow B_i$ in order to force variable B_i to be set to value 1 if and only if there is a loop on vertex $\texttt{NODES}[i]$. Finally we create a constraint $\texttt{NTREES} = B_1 + B_2 + \cdots + B_{|\texttt{NODES}|}$ for stating the fact that the number of trees is equal to the number of loops of the graph.

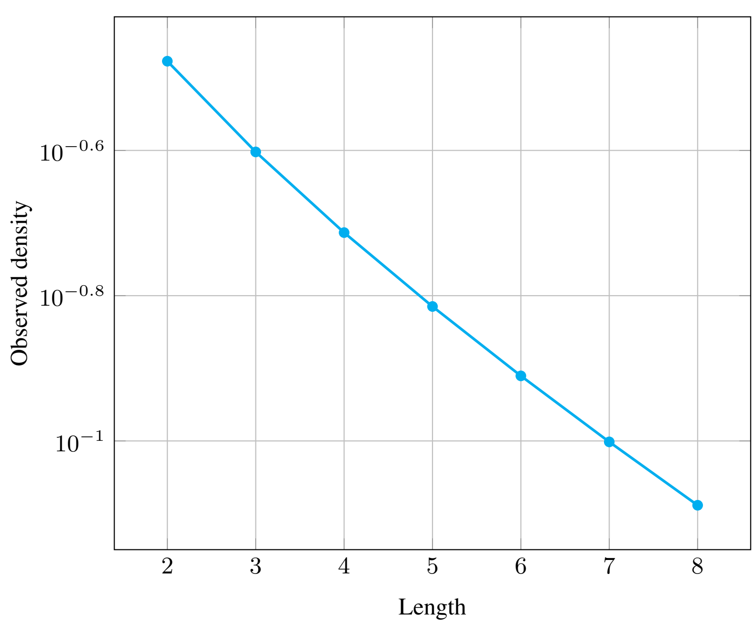
3. For each pair of vertices $\texttt{NODES}[i], \texttt{NODES}[j]$ $(i, j \in [1, |\texttt{NODES}|])$ of the NODES collection we create a 0-1 variable B_{ij} and state the following reified constraint $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Leftrightarrow B_{ij}$. Variable B_{ij} is set to value 1 if and only if there is an arc from $\texttt{NODES}[i]$ to $\texttt{NODES}[j]$. Then for each vertex $\texttt{NODES}[j]$ $(j \in [1, |\texttt{NODES}|])$ we create a constraint of the form $B_{1j} + B_{2j} + \cdots + B_{|\texttt{NODES}|j} \leq 2$.

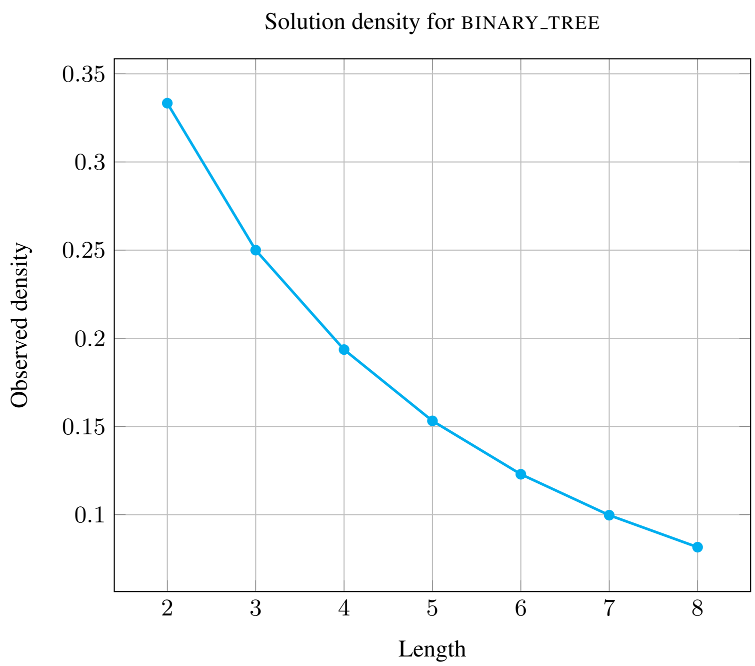
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	16	121	1191	14461	209098	3510921

Number of solutions for BINARY_TREE: domains $0..n$

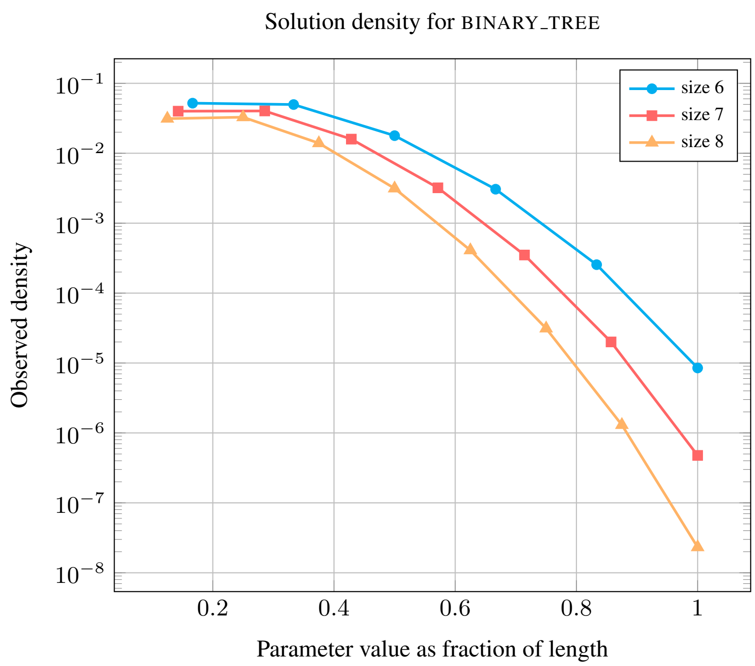
Solution density for BINARY_TREE

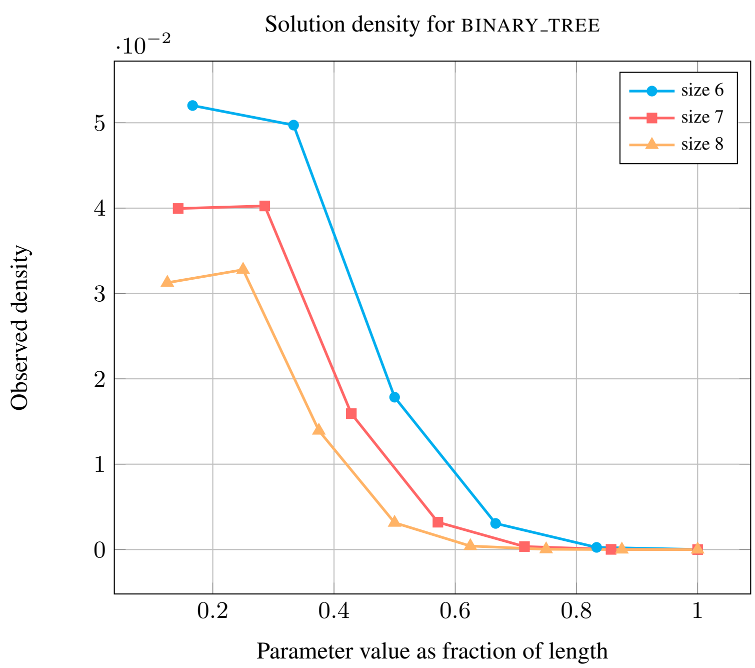




Length (n)		2	3	4	5	6	7	8
Total		3	16	121	1191	14461	209098	3510921
Parameter value	1	2	9	60	540	6120	83790	1345680
	2	1	6	48	480	5850	84420	1411200
	3	-	1	12	150	2100	33390	599760
	4	-	-	1	20	360	6720	135240
	5	-	-	-	1	30	735	17640
	6	-	-	-	-	1	42	1344
	7	-	-	-	-	-	1	56
	8	-	-	-	-	-	-	1

Solution count for BINARY_TREE: domains $0..n$





See also **generalisation:** TREE *(at most two childrens replaced by no restriction on maximum number of childrens)*.

implied by: PATH.

implies: TREE.

implies (items to collection): ATLEAST_NVECTOR.

specialisation: PATH *(at most two childrens replaced by at most one child)*.

Keywords

constraint type: graph constraint, graph partitioning constraint.

final graph structure: connected component, tree, one_succ.

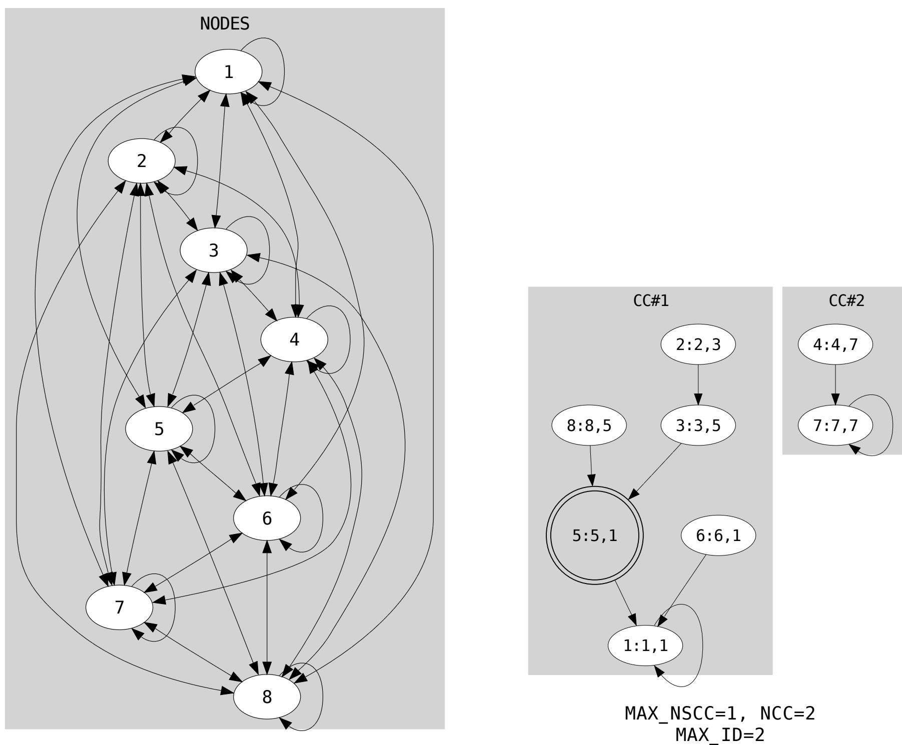
modelling: functional dependency.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NCC** = NTREES • **MAX_ID** ≤ 2
Graph class	ONE_SUCC

Graph model We use the same graph constraint as for the TREE constraint, except that we add the graph property **MAX_ID** ≤ 2, which constraints the maximum in-degree of the final graph to not exceed 2. **MAX_ID** does not consider loops: This is why we do not have any problem with the root of each tree.

Parts (A) and (B) of Figure 5.148 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NCC** graph property, we display the two connected components of the final graph. Each of them corresponds to a binary tree. Since we use the **MAX_IN_DEGREE** graph property, we also show with a double circle a vertex that has a maximum number of predecessors.

The BINARY_TREE constraint holds since all strongly connected components of the final graph have no more than one vertex, since NTREES = **NCC** = 2 and since **MAX_ID** = 2.



(A) **(B)**

Figure 5.148: Initial and final graph of the BINARY_TREE constraint

5.58 BIPARTITE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [151]

Constraint BIPARTITE(NODES)

Argument NODES : collection(index−int, succ−svar)

Restrictions
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq |$NODES$|$

Purpose Consider a digraph G described by the NODES collection. Select a subset of arcs of G so that the corresponding graph is symmetric (i.e., if there is an arc from i to j, there is also an arc from j to i) and bipartite (i.e., there is no cycle involving an odd number of vertices).

Example
$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\{2,3\},\\ \texttt{index}-2 & \texttt{succ}-\{1,4\},\\ \texttt{index}-3 & \texttt{succ}-\{1,4,5\},\\ \texttt{index}-4 & \texttt{succ}-\{2,3,6\},\\ \texttt{index}-5 & \texttt{succ}-\{3,6\},\\ \texttt{index}-6 & \texttt{succ}-\{4,5\}\end{array}\right\rangle\right)$$

The BIPARTITE constraint holds since the NODES collection depicts a symmetric graph with no cycle involving an odd number of vertices. The corresponding graph is depicted by Figure 5.149.

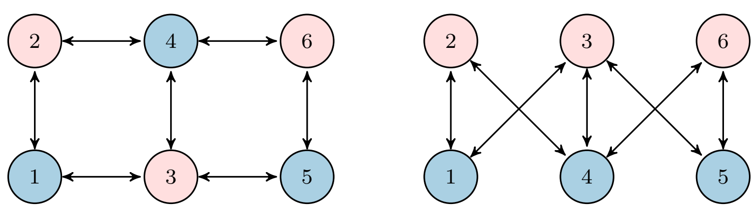


Figure 5.149: Two ways of looking at the bipartite graph given in the **Example** slot

Typical $|$NODES$| > 2$

Symmetry Items of NODES are permutable.

Algorithm The sketch of a filtering algorithm for the BIPARTITE constraint is given in [151, page 91]. Beside enforcing the fact that the graph is symmetric, it checks that the subset of mandatory vertices and arcs is bipartite and removes all potential arcs that would make the previous graph non-bipartite.

See also **used in graph description:** IN_SET.

Keywords **constraint arguments:** constraint involving set variables.

constraint type: graph constraint.

filtering: DFS-bottleneck.

final graph structure: bipartite, symmetric.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.succ)
Graph class	• SYMMETRIC • BIPARTITE

Graph model Part (A) of Figure 5.150 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.150 gives the final graph associated with the **Example** slot.

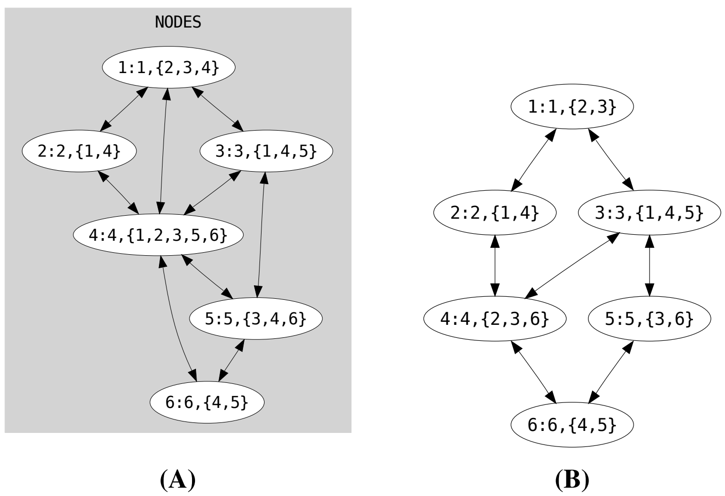


Figure 5.150: Initial and final graph of the BIPARTITE set constraint

5.59 CALENDAR

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin [28]

Constraint $\text{CALENDAR}(\texttt{INSTANTS}, \texttt{MACHINES})$

Type $\texttt{UNAVAILABILITIES} : \texttt{collection}(\texttt{low}-\texttt{int}, \texttt{up}-\texttt{int})$

Arguments

$$\begin{array}{lll} \texttt{INSTANTS} & : & \texttt{collection}\begin{pmatrix} \texttt{machine}-\texttt{dvar}, \\ \texttt{virtual}-\texttt{dvar}, \\ \texttt{ireal}-\texttt{dvar}, \\ \texttt{flagend}-\texttt{int} \end{pmatrix} \\ \texttt{MACHINES} & : & \texttt{collection}(\texttt{id}-\texttt{int}, \texttt{cal}-\texttt{UNAVAILABILITIES}) \end{array}$$

Restrictions

$\texttt{required}(\texttt{UNAVAILABILITIES}, [\texttt{low}, \texttt{up}])$
$\texttt{UNAVAILABILITIES.low} \leq \texttt{UNAVAILABILITIES.up}$
$\texttt{required}(\texttt{INSTANTS}, [\texttt{machine}, \texttt{virtual}, \texttt{ireal}, \texttt{flagend}])$
$\texttt{in\_attr}(\texttt{INSTANTS}, \texttt{machine}, \texttt{MACHINES}, \texttt{id})$
$\texttt{INSTANTS.flagend} \geq 0$
$\texttt{INSTANTS.flagend} \leq 1$
$|\texttt{MACHINES}| > 0$
$\texttt{required}(\texttt{MACHINES}, [\texttt{id}, \texttt{cal}])$
$\texttt{distinct}(\texttt{MACHINES}, \texttt{id})$

Purpose

Makes the link between an universal calendar and resource dependent calendars. Given a collection of machines MACHINES where each machine is defined by its identifier and its unavailability periods the CALENDAR constraint maps items of real and virtual dates depending on the machine assignment as well as of the fact that we consider start ($\texttt{flagend} = 0$) or end ($\texttt{flagend} = 1$) times. Virtual dates on a given machine m do not consider the unavailability periods on m, while real dates consider all time points.

Example

$$\left(\begin{array}{l} \left\langle\begin{array}{llll} \texttt{machine}-1 & \texttt{virtual}-2 & \texttt{ireal}-3 & \texttt{flagend}-0, \\ \texttt{machine}-1 & \texttt{virtual}-5 & \texttt{ireal}-6 & \texttt{flagend}-1, \\ \texttt{machine}-2 & \texttt{virtual}-4 & \texttt{ireal}-5 & \texttt{flagend}-0, \\ \texttt{machine}-2 & \texttt{virtual}-6 & \texttt{ireal}-9 & \texttt{flagend}-1, \\ \texttt{machine}-3 & \texttt{virtual}-2 & \texttt{ireal}-2 & \texttt{flagend}-0, \\ \texttt{machine}-3 & \texttt{virtual}-5 & \texttt{ireal}-5 & \texttt{flagend}-1, \\ \texttt{machine}-4 & \texttt{virtual}-2 & \texttt{ireal}-2 & \texttt{flagend}-0, \\ \texttt{machine}-4 & \texttt{virtual}-7 & \texttt{ireal}-9 & \texttt{flagend}-1 \end{array}\right\rangle, \\ \left\langle\begin{array}{ll} \texttt{id}-1 & \texttt{cal}-\langle\texttt{low}-2\ \texttt{up}-2, \texttt{low}-6\ \texttt{up}-7\rangle, \\ \texttt{id}-2 & \texttt{cal}-\langle\texttt{low}-2\ \texttt{up}-2, \texttt{low}-6\ \texttt{up}-7\rangle, \\ \texttt{id}-3 & \texttt{cal}-[\,], \\ \texttt{id}-4 & \texttt{cal}-\langle\texttt{low}-3\ \texttt{up}-4\rangle \end{array}\right\rangle \end{array}\right)$$

Figure 5.151 illustrates the example. It present four machines with their respective unavailability periods (in grey) as well as four tasks (in blue and pink). Each item of the INSTANTS collection corresponds to the start or to the end of one of the previous four tasks. The CALENDAR constraint holds since:

- The real date 3 (INSTANTS$[1]$.ireal $=$ 3) associated with the start (INSTANTS$[1]$.flagend $= 0$) of task (a) in the universal time corresponds to the virtual date 2 (INSTANTS$[1]$.virtual $= 2$) on machine 1 (INSTANTS$[1]$.machine $=$ 1).
- The real date 6 (INSTANTS$[2]$.ireal $=$ 6) associated with the end (INSTANTS$[2]$.flagend $= 1$) of task (a) in the universal time corresponds to the virtual date 5 (INSTANTS$[2]$.virtual $= 5$) on machine 1 (INSTANTS$[2]$.machine $=$ 1).
- The real date 5 (INSTANTS$[3]$.ireal $=$ 5) associated with the start (INSTANTS$[3]$.flagend $= 0$) of task (b) in the universal time corresponds to the virtual date 4 (INSTANTS$[3]$.virtual $= 4$) on machine 2 (INSTANTS$[3]$.machine $=$ 2).
- The real date 9 (INSTANTS$[4]$.ireal $=$ 9) associated with the end (INSTANTS$[4]$.flagend $= 1$) of task (b) in the universal time corresponds to the virtual date 6 (INSTANTS$[4]$.virtual $= 6$) on machine 2 (INSTANTS$[4]$.machine $=$ 2).
- The real date 2 (INSTANTS$[5]$.ireal $=$ 2) associated with the start (INSTANTS$[5]$.flagend $= 0$) of task (c) in the universal time corresponds to the virtual date 2 (INSTANTS$[5]$.virtual $= 2$) on machine 3 (INSTANTS$[5]$.machine $=$ 3).
- The real date 5 (INSTANTS$[6]$.ireal $=$ 5) associated with the end (INSTANTS$[6]$.flagend $= 1$) of task (c) in the universal time corresponds to the virtual date 5 (INSTANTS$[6]$.virtual $= 5$) on machine 3 (INSTANTS$[6]$.machine $=$ 3).
- The real date 2 (INSTANTS$[7]$.ireal $=$ 2) associated with the start (INSTANTS$[7]$.flagend $= 0$) of task (d) in the universal time corresponds to the virtual date 2 (INSTANTS$[7]$.virtual $= 2$) on machine 4 (INSTANTS$[7]$.machine $=$ 4).
- The real date 9 (INSTANTS$[8]$.ireal $=$ 9) associated with the end (INSTANTS$[8]$.flagend $= 1$) of task (d) in the universal time corresponds to the virtual date 7 (INSTANTS$[8]$.virtual $= 7$) on machine 4 (INSTANTS$[8]$.machine $=$ 4).

Typical

|INSTANTS| > 1
|MACHINES| > 1

Symmetries

- Items of INSTANTS are permutable.
- Items of MACHINES are permutable.

Arg. properties

Contractible wrt. INSTANTS.

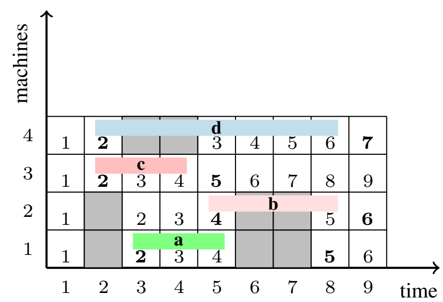


Figure 5.151: Four machines with their unavailability periods as well as four tasks assigned to these machines (virtual dates mentioned in the **Example** slot use a bold font)

Usage The CALENDAR constraint is used as a channelling constraint in resource scheduling problems where resources have unavailability periods that can preempt the execution of a task. In this context two time coordinates systems are used:

- A first coordinate system, so called the *virtual coordinate system*, ignores all unavailability periods on the different resources. All resource constraints are stated within this virtual coordinate system.
- A second coordinate system, so called the *real coordinate system*, corresponds to the real time. All temporal constraints (e.g., precedence constraints) are stated within this real coordinate system.

In this context, each task has a *virtual origin*, a *virtual duration*, a *virtual end*, a *real origin*, a *real duration*, a *real end* and the CALENDAR constraint links together the virtual origin and the real origin as well as the virtual end and the real end. The virtual duration (i.e., the real duration plus the sum of the unavailability periods crossed by the task) is linked to the virtual end and the virtual origin through an equality constraint on the difference between the virtual end and the virtual origin. The real duration is linked in a similar way to the real end and the real origin. The keyword scheduling with machine choice, calendars and preemption provides a concrete example of resource scheduling problem using the CALENDAR constraint.

Reformulation The CALENDAR constraint can be reformulated into two generalised CASE constraints (i.e., two CASE constraints augmented with linear constraints). Part (A) (respectively Part (B)) of Figure 5.152 provides the directed acyclic graph that allows mapping the virtual start and real start (respectively the virtual end and real end) of a task. This directed acyclic graph can be computed directly from the arguments of the CALENDAR constraint:

1. We create an initial root node labelled by m and we partition the set of machines into classes of consecutive machines that all share exactly the same unavailability periods. For each such class we create an arc from the root node to a new node vs labelled by the corresponding interval of consecutive machines identifiers. In Part (A) this corresponds to node m and its three outgoing arcs respectively labelled by intervals $[1, 2]$, $[3, 3]$ and $[4, 4]$.

2. For each class of consecutive machines found previously, we label in increasing order each timepoint that is not part of an unavailability period. We create an arc from the corresponding node vs for each maximum interval of available timepoints to a new node labelled by rs. In Part (A) this translate to:
   - For the class corresponding to machines 1 and 2 we create three outgoing arcs labelled by the time intervals $[1,1]$, $[2,4]$ and $[5,6]$.
   - For the class corresponding to machine 3 we create the outgoing arc labelled by time interval $[1,9]$.
   - For the class corresponding to machine 4 we create the two outgoing arcs labelled by the time intervals $[1,2]$ and $[3,7]$.
3. For each class of consecutive machines and for each maximum interval $[i,j]$ of available timepoints previously computed, we find out the number of unavailable timepoints b_i on the same class of machines that are located before the virtual date i. We create an outgoing arc from the corresponding node rs to a new node labelled by true (there is a single true node for the full directed acyclic graph). This arc is labelled by the interval $[i+b_i, j+b_i]$ and by the linear constraint $r_s = v_s + b_i$. In Part (A) this translate to:
   - For the class corresponding to machines 1 and 2 and for each r_s node associated with the time intervals $[1,1]$, $[2,4]$ and $[5,6]$ we respectively create an outgoing arc labelled by intervals $[1,1]$, $[3,5]$ and $[8,9]$. To each of these arcs we also respectively associate the linear constraints $r_s = v_s + 0$ ($+0$ since on machines 1 and 2 there is no unavailability period before the virtual date 1), $r_s = v_s + 1$ ($+1$ since on machines 1 and 2 there is a single unavailable timepoint before the virtual date 2) and $r_s = v_s + 3$ ($+3$ since on machines 1 and 2 there is three unavailable timepoints before the virtual date 5).
   - For the class corresponding to machine 3 and for the r_s node associated with the time interval $[1,9]$ we create the outgoing arc labelled by time interval $[1,9]$ and by $r_s = v_s + 0$ (i.e., since there is no unavailability period at all on machine 3).
   - For the class corresponding to machine 4 and for each r_s node associated with the time intervals $[1,2]$ and $[3,7]$ we respectively create an outgoing arc labelled by $[1,2]$ and $[5,9]$. To each of these arcs we also respectively associate the linear constraints $r_s = v_s + 0$ ($+0$ since on machine 4 there is no unavailability period before the virtual date 1) and $r_s = v_s + 2$ ($+2$ since on machine 4 there is two unavailable timepoints before the virtual date 3).

The CALENDAR constraint can also be reformulated into a conjunction of reified constraints. This is done by generating, for each pair of items $(\mathcal{I}, \mathcal{M})$ of the `INSTANTS` and `MACHINES` collections, a set of reified constraints expressing:

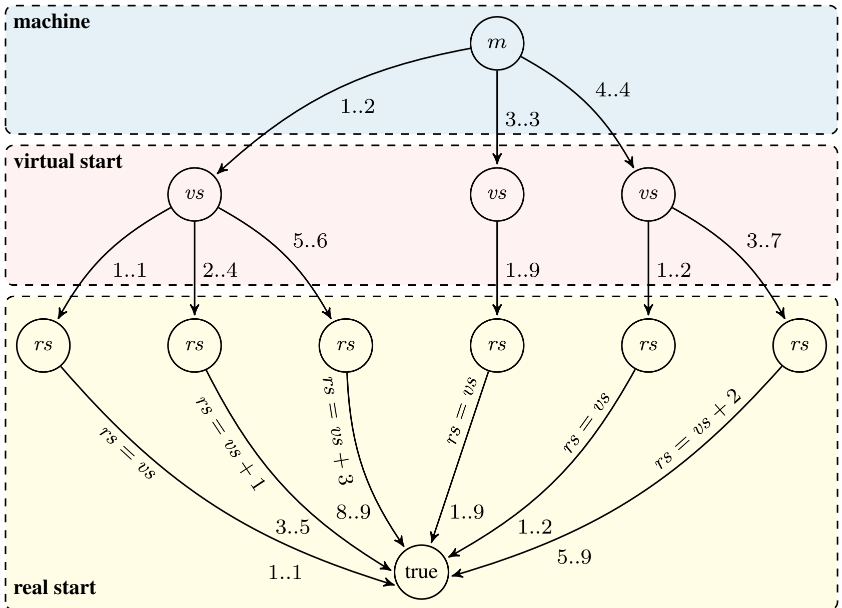
- The link between the real and the virtual dates under the hypothesis that the `machine` attribute of item $\mathcal{I}$ is assigned to the value of the `id` attribute of item $\mathcal{M}$. More precisely, we generate one reified constraint for each available time interval on machine `id`.
- The fact that a real date should not be located within an unavailability period of its corresponding machine.

Operationally, this leads to the following cases:

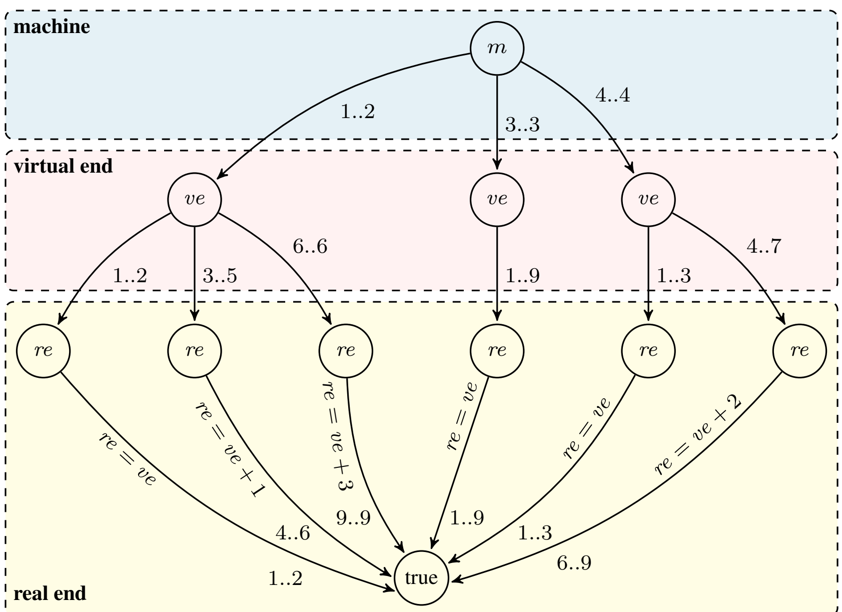
1. When machine `id` has *no unavailability at all* we state an equality constraint between the real and virtual dates.
2. When the real date is located *before the first unavailability period* we also state an equality constraint between the real and virtual dates.
3. When the real date is located *between two consecutive unavailability periods* we state:
   - An equality constraint between the real date and the virtual date plus the sum of all unavailabilities located before the real date.
   - An implication between the fact that the real date belongs to the first unavailability period (among the two consecutive unavailability periods) and the fact that the real date is not assigned to the machine that contains the unavailability period.
4. When the real date is located *after the last unavailability period* we state:
   - An equality constraint between the real date and the virtual date plus the sum of all unavailabilities.
   - An implication between the fact that the real date belongs to the last unavailability period and the fact that the real date is not assigned to the machine that contains the unavailability period.

As an example consider again consider the instance given in the **Example** slot. For the *start of task* a (i.e., the first item $\langle \texttt{machine}-1\ \texttt{virtual}-2\ \texttt{ireal}-2\ \texttt{flagend}-0\rangle$ of collection `INSTANTS`), we generate the following reified constraints, where equivalences of the form $\mathtt{true} \Leftrightarrow \mathtt{true}$ are shown in bold:

- (*if task* a *is assigned on machine 1*)
  ⋆ before $[2,2]$: $1=1 \wedge 3<2 \Leftrightarrow 1=1 \wedge 3=2$
  ⋆ between $[2,2]$ and $[6,7]$: $\mathbf{1=1 \wedge 3>2 \wedge 3<6 \Leftrightarrow 1=1 \wedge 3=2+1}$
  ⋆ after $[6,7]$: $1=1 \wedge 3>7 \Leftrightarrow 1=1 \wedge 3=2+3$
  ⋆ do not cross $[2,2]$, $[6,7]$: $3\in[2,2] \Rightarrow 1\neq 1, 3\in[6,7] \Leftrightarrow 1\neq 1$
- (*if task* a *is assigned on machine 2*)
  ⋆ before $[2,2]$: $1=2 \wedge 3<2 \Leftrightarrow 1=2 \wedge 3=2$
  ⋆ between $[2,2]$ and $[6,7]$: $1=2 \wedge 3>2 \wedge 3<6 \Leftrightarrow 1=2 \wedge 3=2+1$
  ⋆ after $[6,7]$: $1=2 \wedge 3>7 \Leftrightarrow 1=2 \wedge 3=2+3$
  ⋆ do not cross $[2,2]$, $[6,7]$: $3\in[2,2] \Rightarrow 1\neq 2, 3\in[6,7] \Leftrightarrow 1\neq 2$
- (*if task* a *is assigned on machine 3*)
  ⋆ no unavailability: $1=3 \Leftrightarrow 1=3 \wedge 3=2$
- (*if task* a *is assigned on machine 4*)
  ⋆ before $[3,4]$: $1=4 \wedge 3<3 \Leftrightarrow 1=4 \wedge 3=2$
  ⋆ after $[3,4]$: $1=4 \wedge 3>4 \Leftrightarrow 1=4 \wedge 3=2+2$
  ⋆ do not cross $[3,4]$: $3\in[3,4] \Rightarrow 1\neq 4$



(A) directed acyclic graph linking the machine attribute m, the virtual start vs and the real start rs of a task



(B) directed acyclic graph linking the machine attribute m, the virtual end ve and the real end re of a task

Figure 5.152: The two generalised CASE constraints for respectively mapping (A) the virtual start and real start of a task corresponding to the **Example** slot as well as (B) the virtual end and real end; the directed acyclic graphs were generated under the hypothesis that the virtual and real dates are located in $[1, 9]$.

For the *end of task* a (i.e., the second item $\langle$machine -1 virtual -5 ireal -6 flagend $-1\rangle$ of collection INSTANTS), we generate the following reified constraints:

- (*if task* a *is assigned on machine 1*)
  - ⋆ before $[2,2]$: $1=1 \wedge 6<3 \Leftrightarrow 1=1 \wedge 6=5$
  - ⋆ between $[2,2]$ and $[6,7]$: $\mathbf{1=1 \wedge 6>3 \wedge 6<7 \Leftrightarrow 1=1 \wedge 6=5+1}$
  - ⋆ after $[6,7]$: $1=1 \wedge 6>8 \Leftrightarrow 1=1 \wedge 6=5+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $6 \in [3,3] \Rightarrow 1 \neq 1$, $6 \in [7,8] \Rightarrow 1 \neq 1$
- (*if task* a *is assigned on machine 2*)
  - ⋆ before $[2,2]$: $1=2 \wedge 6<3 \Leftrightarrow 1=2 \wedge 6=5$
  - ⋆ between $[2,2]$ and $[6,7]$: $1=2 \wedge 6>3 \wedge 6<7 \Leftrightarrow 1=2 \wedge 6=5+1$
  - ⋆ after $[6,7]$: $1=2 \wedge 6>8 \Leftrightarrow 1=2 \wedge 6=5+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $6 \in [3,3] \Rightarrow 1 \neq 2$, $6 \in [7,8] \Rightarrow 1 \neq 2$
- (*if task* a *is assigned on machine 3*)
  - ⋆ no unavailability: $1=3 \Leftrightarrow 1=3 \wedge 6=5$
- (*if task* a *is assigned on machine 4*)
  - ⋆ before $[3,4]$: $1=4 \wedge 6<4 \Leftrightarrow 1=4 \wedge 6=5$
  - ⋆ after $[3,4]$: $1=4 \wedge 6>5 \Leftrightarrow 1=4 \wedge 6=5+2$
  - ⋆ do not cross $[3,4]$: $6 \in [4,5] \Rightarrow 1 \neq 4$

For the *start of task* b (i.e., the third item $\langle$machine -2 virtual -4 ireal -5 flagend $-0\rangle$ of collection INSTANTS), we generate the following reified constraints:

- (*if task* b *is assigned on machine 1*)
  - ⋆ before $[2,2]$: $2=1 \wedge 5<2 \Leftrightarrow 2=1 \wedge 5=4$
  - ⋆ between $[2,2]$ and $[6,7]$: $2=1 \wedge 5>2 \wedge 5<6 \Leftrightarrow 2=1 \wedge 5=4+1$
  - ⋆ after $[6,7]$: $2=1 \wedge 5>7 \Leftrightarrow 2=1 \wedge 5=4+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $5 \in [2,2] \Rightarrow 2 \neq 1$, $5 \in [6,7] \Rightarrow 2 \neq 1$
- (*if task* b *is assigned on machine 2*)
  - ⋆ before $[2,2]$: $2=2 \wedge 5<2 \Leftrightarrow 2=2 \wedge 5=4$
  - ⋆ between $[2,2]$ and $[6,7]$: $\mathbf{2=2 \wedge 5>2 \wedge 5<6 \Leftrightarrow 2=2 \wedge 5=4+1}$
  - ⋆ after $[6,7]$: $2=2 \wedge 5>7 \Leftrightarrow 2=2 \wedge 5=4+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $5 \in [2,2] \Rightarrow 2 \neq 2$, $5 \in [6,7] \Rightarrow 2 \neq 2$
- (*if task* b *is assigned on machine 3*)
  - ⋆ no unavailability: $2=3 \Leftrightarrow 2=3 \wedge 5=4$
- (*if task* b *is assigned on machine 4*)
  - ⋆ before $[3,4]$: $2=4 \wedge 5<3 \Leftrightarrow 2=4 \wedge 5=4$
  - ⋆ after $[3,4]$: $2=4 \wedge 5>4 \Leftrightarrow 2=4 \wedge 5=4+2$
  - ⋆ do not cross $[3,4]$: $5 \in [3,4] \Rightarrow 2 \neq 4$

For the *end of task* b (i.e., the fourth item $\langle$`machine`-2 `virtual`-6 `ireal`-9 `flagend`$-1\rangle$ of collection `INSTANTS`), we generate the following reified constraints:

- (*if task* b *is assigned on machine 1*)
  - ⋆ before $[2,2]$: $2=1 \wedge 9<3 \Leftrightarrow 2=1 \wedge 9=6$
  - ⋆ between $[2,2]$ and $[6,7]$: $2=1 \wedge 9>3 \wedge 9<7 \Leftrightarrow 2=1 \wedge 9=6+1$
  - ⋆ after $[6,7]$: $2=1 \wedge 9>8 \Leftrightarrow 2=1 \wedge 9=6+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $9 \in [3,3] \Rightarrow 2 \neq 1$, $9 \in [7,8] \Rightarrow 2 \neq 1$
- (*if task* b *is assigned on machine 2*)
  - ⋆ before $[2,2]$: $2=2 \wedge 9<3 \Leftrightarrow 2=2 \wedge 9=6$
  - ⋆ between $[2,2]$ and $[6,7]$: $2=2 \wedge 9>3 \wedge 9<7 \Leftrightarrow 2=2 \wedge 9=6+1$
  - ⋆ after $[6,7]$: $\mathbf{2=2 \wedge 9>8 \Leftrightarrow 2=2 \wedge 9=6+3}$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $9 \in [3,3] \Rightarrow 2 \neq 2$, $9 \in [7,8] \Rightarrow 2 \neq 2$
- (*if task* b *is assigned on machine 3*)
  - ⋆ no unavailability: $2=3 \Leftrightarrow 2=3 \wedge 9=6$
- (*if task* b *is assigned on machine 4*)
  - ⋆ before $[3,4]$: $2=4 \wedge 9<4 \Leftrightarrow 2=4 \wedge 9=6$
  - ⋆ after $[3,4]$: $2=4 \wedge 9>5 \Leftrightarrow 2=4 \wedge 9=6+2$
  - ⋆ do not cross $[3,4]$: $9 \in [4,5] \Rightarrow 2 \neq 4$

For the *start of task* c (i.e., the fifth item $\langle$`machine`-3 `virtual`-2 `ireal`-2 `flagend`$-0\rangle$ of collection `INSTANTS`), we generate the following reified constraints:

- (*if task* c *is assigned on machine 1*)
  - ⋆ before $[2,2]$: $3=1 \wedge 2<2 \Leftrightarrow 3=1 \wedge 2=2$
  - ⋆ between $[2,2]$ and $[6,7]$: $3=1 \wedge 2>2 \wedge 2<6 \Leftrightarrow 3=1 \wedge 2=2+1$
  - ⋆ after $[6,7]$: $3=1 \wedge 2>7 \Leftrightarrow 3=1 \wedge 2=2+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $2 \in [2,2] \Rightarrow 3 \neq 1$, $2 \in [6,7] \Rightarrow 3 \neq 1$
- (*if task* c *is assigned on machine 2*)
  - ⋆ before $[2,2]$: $3=2 \wedge 2<2 \Leftrightarrow 3=2 \wedge 2=2$
  - ⋆ between $[2,2]$ and $[6,7]$: $3=2 \wedge 2>2 \wedge 2<6 \Leftrightarrow 3=2 \wedge 2=2+1$
  - ⋆ after $[6,7]$: $3=2 \wedge 2>7 \Leftrightarrow 3=2 \wedge 2=2+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $2 \in [2,2] \Rightarrow 3 \neq 2$, $2 \in [6,7] \Rightarrow 3 \neq 2$
- (*if task* c *is assigned on machine 3*)
  - ⋆ no unavailability: $\mathbf{3=3 \Leftrightarrow 3=3 \wedge 2=2}$
- (*if task* c *is assigned on machine 4*)
  - ⋆ before $[3,4]$: $3=4 \wedge 2<3 \Leftrightarrow 3=4 \wedge 2=2$
  - ⋆ after $[3,4]$: $3=4 \wedge 2>4 \Leftrightarrow 3=4 \wedge 2=2+2$
  - ⋆ do not cross $[3,4]$: $2 \in [3,4] \Rightarrow 3 \neq 4$

For the *end of task* c (i.e., the sixth item $\langle$`machine`-3 `virtual`-5 `ireal`-5 `flagend`$-1\rangle$ of collection `INSTANTS`), we generate the following reified constraints:

- (*if task* c *is assigned on machine 1*)
  - ⋆ before $[2,2]$: $3=1 \wedge 5<3 \Leftrightarrow 3=1 \wedge 5=5$
  - ⋆ between $[2,2]$ and $[6,7]$: $3=1 \wedge 5>3 \wedge 5<7 \Leftrightarrow 3=1 \wedge 5=5+1$
  - ⋆ after $[6,7]$: $3=1 \wedge 5>8 \Leftrightarrow 3=1 \wedge 5=5+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $5 \in [3..3] \Rightarrow 3 \neq 1$, $5 \in [7..8] \Rightarrow 3 \neq 1$
- (*if task* c *is assigned on machine 2*)
  - ⋆ before $[2,2]$: $3=2 \wedge 5<3 \Leftrightarrow 3=2 \wedge 5=5$
  - ⋆ between $[2,2]$ and $[6,7]$: $3=2 \wedge 5>3 \wedge 5<7 \Leftrightarrow 3=2 \wedge 5=5+1$
  - ⋆ after $[6,7]$: $3=2 \wedge 5>8 \Leftrightarrow 3=2 \wedge 5=5+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $5 \in [3..3] \Rightarrow 3 \neq 2$, $5 \in [7..8] \Rightarrow 3 \neq 2$
- (*if task* c *is assigned on machine 3*)
  - ⋆ no unavailability: $\mathbf{3=3 \Leftrightarrow 3=3 \wedge 5=5}$
- (*if task* c *is assigned on machine 4*)
  - ⋆ before $[3,4]$: $3=4 \wedge 5<4 \Leftrightarrow 3=4 \wedge 5=5$
  - ⋆ after $[3,4]$: $3=4 \wedge 5>5 \Leftrightarrow 3=4 \wedge 5=5+2$
  - ⋆ do not cross $[3,4]$: $5 \in [4..5] \Rightarrow 3 \neq 4$

For the *start of task* d (i.e., the seventh item $\langle$`machine` $-$ 4 `virtual` $-$ 2 `ireal` $-$ 2 `flagend` $-$ 0$\rangle$ of collection `INSTANTS`), we generate the following reified constraints:

- (*if task* d *is assigned on machine 1*)
  - ⋆ before $[2,2]$: $4=1 \wedge 2<2 \Leftrightarrow 4=1 \wedge 2=2$
  - ⋆ between $[2,2]$ and $[6,7]$: $4=1 \wedge 2>2 \wedge 2<6 \Leftrightarrow 4=1 \wedge 2=2+1$
  - ⋆ after $[6,7]$: $4=1 \wedge 2>7 \Leftrightarrow 4=1 \wedge 2=2+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $2 \in [2,2] \Rightarrow 4 \neq 1$, $2 \in [6,7] \Rightarrow 4 \neq 1$
- (*if task* d *is assigned on machine 2*)
  - ⋆ before $[2,2]$: $4=2 \wedge 2<2 \Leftrightarrow 4=2 \wedge 2=2$
  - ⋆ between $[2,2]$ and $[6,7]$: $4=2 \wedge 2>2 \wedge 2<6 \Leftrightarrow 4=2 \wedge 2=2+1$
  - ⋆ after $[6,7]$: $4=2 \wedge 2>7 \Leftrightarrow 4=2 \wedge 2=2+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $2 \in [2,2] \Rightarrow 4 \neq 2$, $2 \in [6,7] \Rightarrow 4 \neq 2$
- (*if task* d *is assigned on machine 3*)
  - ⋆ no unavailability: $4=3 \Leftrightarrow 4=3 \wedge 2=2$
- (*if task* d *assigned on machine 4*)
  - ⋆ before $[3,4]$: $\mathbf{4=4 \wedge 2<3 \Leftrightarrow 4=4 \wedge 2=2}$
  - ⋆ after $[3,4]$: $4=4 \wedge 2>4 \Leftrightarrow 4=4 \wedge 2=2+2$
  - ⋆ do not cross $[3,4]$: $2 \in [3,4] \Rightarrow 4 \neq 4$

For the *end of task* d (i.e., the eighth item $\langle$`machine` -4 `virtual` -7 `ireal` -9 `flagend` $-1\rangle$ of collection `INSTANTS`), we generate the following reified constraints:

- (*if task* d *is assigned on machine 1*)
  - ⋆ before $[2,2]$: $4=1 \wedge 9<3 \Leftrightarrow 4=1 \wedge 9=7$
  - ⋆ between $[2,2]$ and $[6,7]$: $4=1 \wedge 9>3 \wedge 9<7 \Leftrightarrow 4=1 \wedge 9=7+1$
  - ⋆ after $[6,7]$: $4=1 \wedge 9>8 \Leftrightarrow 4=1 \wedge 9=7+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $9 \in [3,3] \Rightarrow 4 \neq 1, 9 \in [7,8] \Rightarrow 4 \neq 1$
- (*if task* d *is assigned on machine 2*)
  - ⋆ before $[2,2]$: $4=2 \wedge 9<3 \Leftrightarrow 4=2 \wedge 9=7$
  - ⋆ between $[2,2]$ and $[6,7]$: $4=2 \wedge 9>3 \wedge 9<7 \Leftrightarrow 4=2 \wedge 9=7+1$
  - ⋆ after $[6,7]$: $4=2 \wedge 9>8 \Leftrightarrow 4=2 \wedge 9=7+3$
  - ⋆ do not cross $[2,2]$, $[6,7]$: $9 \in [3,3] \Rightarrow 4 \neq 2, 9 \in [7,8] \Rightarrow 4 \neq 2$
- (*if task* d *is assigned on machine 3*)
  - ⋆ no unavailability: $4=3 \Leftrightarrow 4=3 \wedge 9=7$
- (*if task* d *is assigned on machine 4*)
  - ⋆ before $[3,4]$: $4=4 \wedge 9<4 \Leftrightarrow 4=4 \wedge 9=7$
  - ⋆ after $[3,4]$: $\mathbf{4=4 \wedge 9>5 \Leftrightarrow 4=4 \wedge 9=7+2}$
  - ⋆ do not cross $[3,4]$: $9 \in [4,5] \Rightarrow 4 \neq 4$

See also **common keyword:** CUMULATIVE *(scheduling constraint)*, CUMULATIVES *(scheduling with machine choice, calendars and preemption)*, DIFFN *(multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption)*, DISJUNCTIVE *(scheduling constraint)*, GEOST *(multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption)*.

Keywords **constraint type:** predefined constraint, temporal constraint, scheduling constraint.

modelling: channelling constraint, multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption, assignment dimension.

modelling exercises: multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption.

5.60 CARDINALITY_ATLEAST

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from GLOBAL_CARDINALITY.

Constraint CARDINALITY_ATLEAST(ATLEAST, VARIABLES, VALUES)

Arguments

ATLEAST	:	dvar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int)

Restrictions

ATLEAST ≥ 0
ATLEAST $\leq$ |VARIABLES|
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose ATLEAST is the minimum number of time that a value of VALUES is taken by the variables of the collection VARIABLES.

Example $(1, \langle 3, 3, 8\rangle, \langle 3, 8\rangle)$

In this example, values 3 and 8 are respectively used 2, and 1 times. The CARDINALITY_ATLEAST constraint holds since its first argument ATLEAST $= 1$ is assigned to the minimum number of time that values 3 and 8 occur in the collection $\langle 3, 3, 8\rangle$.

Typical

ATLEAST > 0
ATLEAST $<$ |VARIABLES|
|VARIABLES| > 1
|VALUES| > 0
|VARIABLES| $>$ |VALUES|

Symmetries

- Items of VARIABLES are permutable.
- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Arg. properties Functional dependency: ATLEAST determined by VARIABLES and VALUES.

Usage An application of the CARDINALITY_ATLEAST constraint is to enforce a minimum use of values.

Remark This is a restricted form of a variant of an AMONG constraint and of the GLOBAL_CARDINALITY constraint. In the original GLOBAL_CARDINALITY constraint, one specifies for each value its minimum and maximum number of occurrences.

Algorithm See GLOBAL_CARDINALITY [353].

See also **generalisation:** GLOBAL_CARDINALITY *(single* `count variable` *replaced by an individual* `count variable` *for each value)*.

Keywords **application area:** assignment.

characteristic of a constraint: automaton, automaton with array of counters.

constraint arguments: pure functional dependency.

constraint type: value constraint.

filtering: arc-consistency.

final graph structure: acyclic, bipartite, no loop.

modelling: functional dependency, at least.

Arc input(s) VARIABLES VALUES

Arc generator $PRODUCT \mapsto$ `collection`(`variables`, `values`)

Arc arity 2

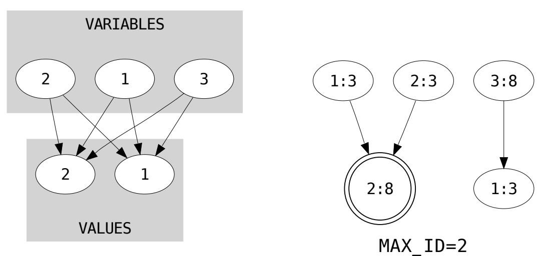
Arc constraint(s) `variables.var` $\neq$ `values.val`

Graph property(ies) **MAX_ID**= |VARIABLES| − ATLEAST

Graph class
- ACYCLIC
- BIPARTITE
- NO_LOOP

Graph model Using directly the graph property **MIN_ID** = ATLEAST, and replacing the disequality of the arc constraint by an equality does not work since it ignores values that are not assigned to any variable. This comes from the fact that isolated vertices are removed from the final graph.

Parts (A) and (B) of Figure 5.153 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_ID** graph property, the vertex with the maximum number of predecessor (i.e., namely two predecessors) is stressed with a double circle. As a consequence the first argument ATLEAST of the CARDINALITY_ATLEAST constraint is assigned to the total number of variables 3 minus 2.



(A) **(B)**

Figure 5.153: Initial and final graph of the CARDINALITY_ATLEAST constraint

Automaton Figure 5.154 depicts the automaton associated with the CARDINALITY_ATLEAST constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i \in \mathtt{VALUES} \Leftrightarrow S_i$.

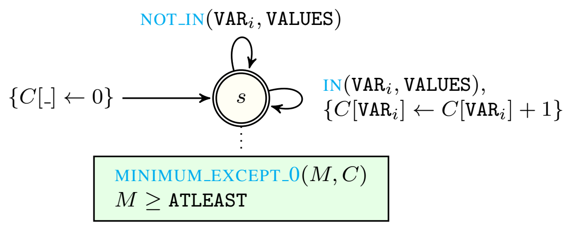


Figure 5.154: Automaton of the CARDINALITY_ATLEAST constraint

5.61 CARDINALITY_ATMOST

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from GLOBAL_CARDINALITY.

Constraint CARDINALITY_ATMOST(ATMOST, VARIABLES, VALUES)

Arguments

```
ATMOST    : dvar
VARIABLES : collection(var−dvar)
VALUES    : collection(val−int)
```

Restrictions

ATMOST ≥ 0
ATMOST $\leq$ |VARIABLES|
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose ATMOST is the maximum number of occurrences of each value of VALUES within the variables of the collection VARIABLES.

Example $(2, \langle 2, 1, 7, 1, 2\rangle, \langle 5, 7, 2, 9\rangle)$

In this example, values 5, 7, 2 and 9 occur respectively 0, 1, 2 and 0 times within the collection $\langle 2, 1, 7, 1, 2\rangle$. As a consequence, the CARDINALITY_ATMOST constraint holds since its first argument ATMOST is assigned to the maximum number of occurrences 2.

Typical

ATMOST > 0
ATMOST $<$ |VARIABLES|
|VARIABLES| > 1
|VALUES| > 0
|VARIABLES| $>$ |VALUES|

Symmetries

- Items of VARIABLES are permutable.
- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Arg. properties Functional dependency: ATMOST determined by VARIABLES and VALUES.

Usage An application of the CARDINALITY_ATMOST constraint is to enforce a maximum use of values.

Remark This is a restricted form of a variant of the AMONG constraint and of the GLOBAL_CARDINALITY constraint. In the original GLOBAL_CARDINALITY constraint, one specifies for each value its minimum and maximum number of occurrences.

Algorithm See GLOBAL_CARDINALITY [353].

See also **generalisation:** GLOBAL_CARDINALITY *(single* `count variable` *replaced by an individual* `count variable` *for each value)*, MULTI_INTER_DISTANCE *(window of size* 1 *replaced by window of* `DIST` *consecutive values)*.

implied by: AMONG.

Keywords **application area:** assignment.

characteristic of a constraint: automaton, automaton with array of counters.

constraint arguments: pure functional dependency.

constraint type: value constraint.

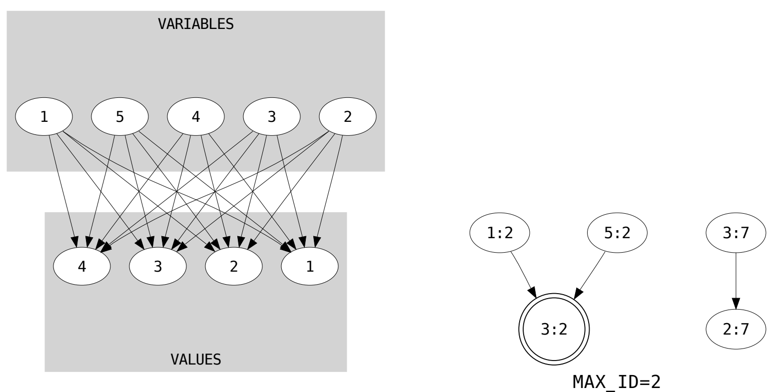
filtering: arc-consistency.

final graph structure: acyclic, bipartite, no loop.

modelling: at most, functional dependency.

Arc input(s)	VARIABLES VALUES
Arc generator	$PRODUCT \mapsto$ collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	**MAX_ID**= ATMOST
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.155 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_ID** graph property, the vertex that has the maximum number of predecessor is stressed with a double circle.



(A) **(B)**

Figure 5.155: Initial and final graph of the CARDINALITY_ATMOST constraint

Automaton Figure 5.156 depicts the automaton associated with the CARDINALITY_ATMOST constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i \in \mathtt{VALUES} \Leftrightarrow S_i$.

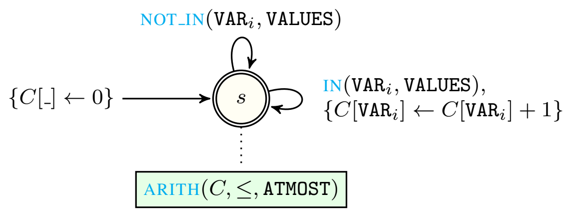


Figure 5.156: Automaton of the CARDINALITY_ATMOST constraint

5.62 CARDINALITY_ATMOST_PARTITION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from GLOBAL_CARDINALITY.

Constraint CARDINALITY_ATMOST_PARTITION(ATMOST, VARIABLES, PARTITIONS)

Type VALUES : collection(val−int)

Arguments

ATMOST : dvar
VARIABLES : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

|VALUES| ≥ 1
required(VALUES, val)
distinct(VALUES, val)
ATMOST ≥ 0
ATMOST $\leq$ |VARIABLES|
required(VARIABLES, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2

Purpose ATMOST is the maximum number of time that values of a same partition of PARTITIONS are taken by the variables of the collection VARIABLES.

Example $(2, \langle 2, 3, 7, 1, 6, 0\rangle, \langle \texttt{p} - \langle 1, 3\rangle, \texttt{p} - \langle 4\rangle, \texttt{p} - \langle 2, 6\rangle\rangle)$

In this example, two variables of the collection VARIABLES $= \langle 2, 3, 7, 1, 6, 0\rangle$ are assigned values of the first partition, no variable is assigned a value of the second partition, and finally two variables are assigned values of the last partition. As a consequence, the CARDINALITY_ATMOST_PARTITION constraint holds since its first argument ATMOST is assigned to the maximum number of occurrences 2.

Typical

ATMOST > 0
ATMOST $<$ |VARIABLES|
|VARIABLES| > 1
|VARIABLES| $>$ |PARTITIONS|

Symmetries

- Items of VARIABLES are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.

Arg. properties Functional dependency: ATMOST determined by VARIABLES and PARTITIONS.

See also

generalisation: GLOBAL_CARDINALITY *(single* `count` `variable` *replaced by an individual* `count` `variable` *for each value and* `variable` *replaced by* `variable` $\in$ `partition`*).*

used in graph description: IN.

Keywords

characteristic of a constraint: partition.

constraint arguments: pure functional dependency.

constraint type: value constraint.

filtering: arc-consistency.

final graph structure: acyclic, bipartite, no loop.

modelling: at most, functional dependency.

Arc input(s)	VARIABLES PARTITIONS
Arc generator	*PRODUCT* ↦ collection(variables, partitions)
Arc arity	2
Arc constraint(s)	IN(variables.var, partitions.p)
Graph property(ies)	**MAX_ID** = ATMOST
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.157 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_ID** graph property, a vertex with the maximum number of predecessor is stressed with a double circle.

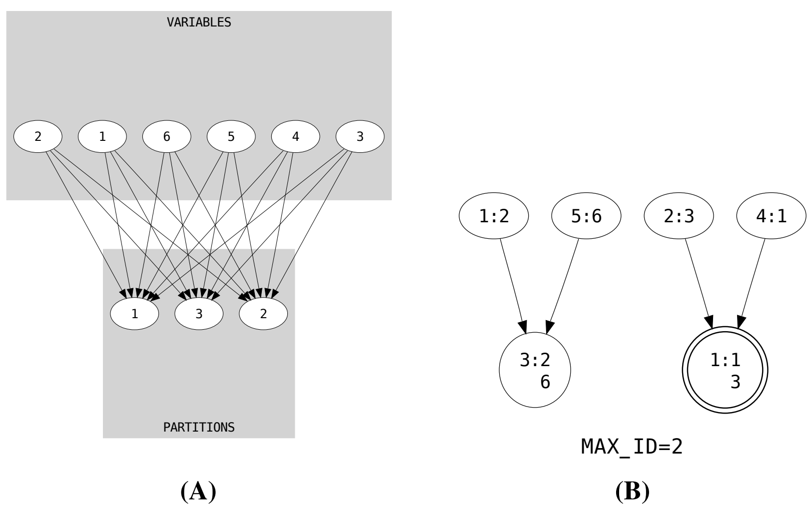


Figure 5.157: Initial and final graph of the CARDINALITY_ATMOST_PARTITION constraint

5.63 CHANGE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint CHANGE(NCHANGE, VARIABLES, CTR)

Synonyms NBCHANGES, SIMILARITY.

Arguments

NCHANGE	:	dvar
VARIABLES	:	collection(var−dvar)
CTR	:	atom

Restrictions

NCHANGE ≥ 0
NCHANGE $<$ |VARIABLES|
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose NCHANGE is the number of times that constraint CTR holds on consecutive variables of the collection VARIABLES.

Example

$(3, \langle 4, 4, 3, 4, 1\rangle, \neq)$
$(1, \langle 1, 2, 4, 3, 7\rangle, >)$

$4{=}4{\neq}3{\neq}4{\neq}1$
$1{\leq}2{\leq}4{>}3{\leq}7$

In the first example the changes are located between values 4 and 3, 3 and 4, 4 and 1. Consequently, the corresponding CHANGE constraint holds since its first argument NCHANGE is fixed to value 3.

In the second example the unique change occurs between values 4 and 3. Consequently, the corresponding CHANGE constraint holds since its first argument NCHANGE is fixed to 1.

All solutions Figure 5.158 gives all solutions to the following non ground instance of the CHANGE constraint: NCHANGE $\in [0, 1]$, $V_1 \in [2, 3]$, $V_2 \in [1, 2]$, $V_3 \in [4, 5]$, $V_4 \in [2, 4]$, CHANGE(NCHANGE, $\langle V_1, V_2, V_3, V_4\rangle, >$).

Typical

NCHANGE > 0
|VARIABLES| > 1
range(VARIABLES.var) > 1
CTR $\in [\neq]$

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: NCHANGE determined by VARIABLES and CTR.
- Contractible wrt. VARIABLES when CTR $\in [\neq, <, \geq, >, \leq]$ and NCHANGE $= 0$.
- Contractible wrt. VARIABLES when CTR $\in [=, <, \geq, >, \leq]$ and NCHANGE $=$ |VARIABLES| $- 1$.

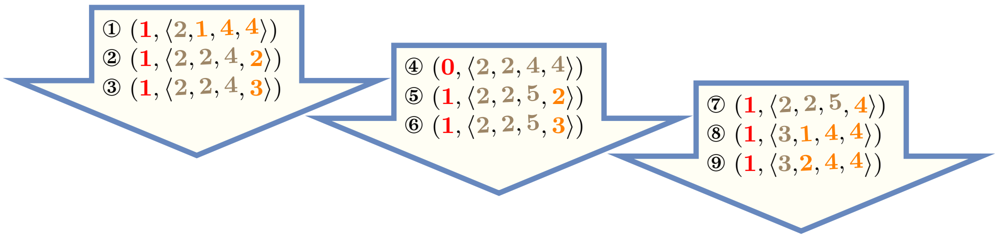


Figure 5.158: All solutions corresponding to the non ground example of the CHANGE constraint (with CTR set to $>$) of the **All solutions** slot; each change is shown by a color change between two consecutive values.

Usage This constraint can be used in the context of timetabling problems in order to put an upper limit on the number of changes of job types during a given period.

Remark A similar constraint appears in [314, page 338] under the name of SIMILARITY constraint. The difference consists of replacing the arithmetic constraint CTR by a binary constraint. When CTR is equal to $\neq$ this constraint is called NBCHANGES in [416].

Algorithm A first incomplete algorithm is described in [32]. The sketch of a filtering algorithm for the conjunction of the CHANGE and the STRETCH constraints based on dynamic programming achieving arc-consistency is mentioned by L. Hellsten in [219, page 56].

Reformulation The CHANGE constraint can be reformulated with the SEQ_BIN constraint [321] that we now introduce. Given N a domain variable, X a sequence of domain variables, and C and B two binary constraints, SEQ_BIN(N, X, C, B) holds if (1) N is equal to the number of C-stretches in the sequence X, and (2) B holds on any pair of consecutive variables in X. A C-stretch is a generalisation of the notion of stretch introduced by G. Pesant [316], where the equality constraint is made explicit by replacing it by a binary constraint C, i.e., a C-stretch is a maximal length subsequence of X for which the binary constraint C is satisfied on consecutive variables. CHANGE(NCHANGE, VARIABLES, CTR) can be reformulated as $\mathtt{N} = \mathtt{N1} - 1 \wedge$ SEQ_BIN(N1, X, $\neg$ CTR, true), where true is the universal constraint.

Used in PATTERN.

See also **common keyword:** CHANGE_PARTITION, CIRCULAR_CHANGE *(number of changes in a sequence of* variables *with respect to a* binary constraint*)*, CYCLIC_CHANGE, CYCLIC_CHANGE_JOKER *(number of changes)*, SMOOTH *(number of changes in a sequence of* variables *with respect to a* binary constraint*)*.

generalisation: CHANGE_PAIR *(*variable *replaced by* pair of variables*)*, CHANGE_VECTORS *(*variable *replaced by vector)*.

shift of concept: DISTANCE_CHANGE, LONGEST_CHANGE.

Keywords **characteristic of a constraint:** automaton, automaton with counters, non-deterministic automaton.

constraint arguments: pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2), Berge-acyclic constraint network.

constraint type: timetabling constraint.

filtering: dynamic programming.

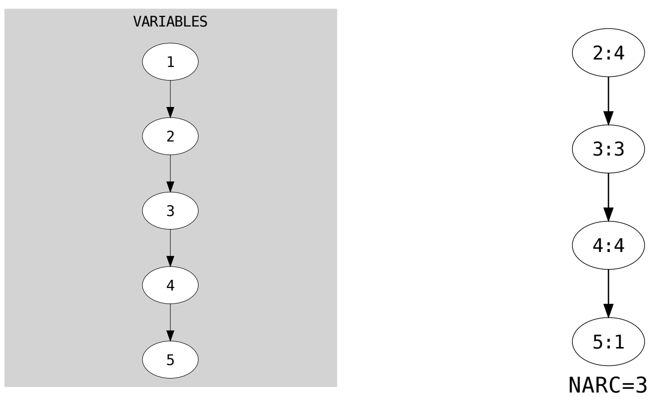
final graph structure: acyclic, bipartite, no loop.

modelling: number of changes, functional dependency.

Arc input(s)	VARIABLES
Arc generator	*PATH* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	**NARC** = NCHANGE
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Since we are only interested by the constraints linking two consecutive items of the collection VARIABLES we use *PATH* to generate the arcs of the initial graph.

Parts (A) and (B) of Figure 5.159 respectively show the initial and final graph of the first example of the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) (B)

Figure 5.159: Initial and final graph of the CHANGE constraint

Automaton Figure 5.160 depicts a first automaton that only accepts all the solutions to the CHANGE constraint. This automaton uses a counter in order to record the number of satisfied constraints of the form $\mathtt{VAR}_i$ CTR $\mathtt{VAR}_{i+1}$ already encountered. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $\mathtt{VAR}_i$ CTR $\mathtt{VAR}_{i+1} \Leftrightarrow S_i$.

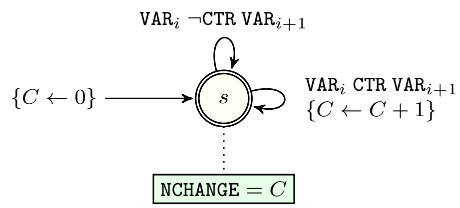


Figure 5.160: Automaton (with counter) of the CHANGE constraint

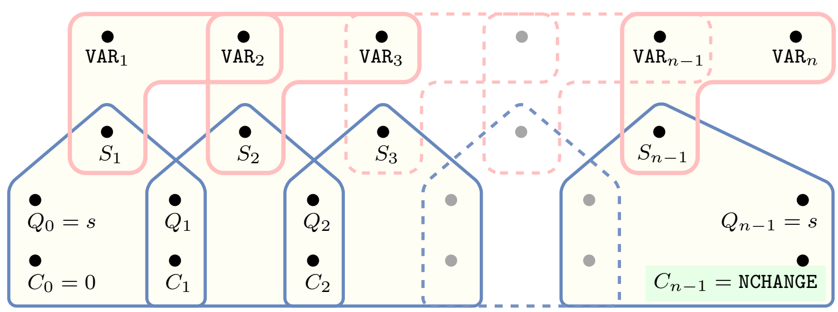


Figure 5.161: Hypergraph of the reformulation corresponding to the automaton (with counter) of the CHANGE constraint

Since the reformulation associated with the previous automaton is not Berge-acyclic, we now describe a second counter free automaton that also only accepts all the solutions to the CHANGE constraint. Without loss of generality, assume that the collection of variables VARIABLES contains at least two variables (i.e., $|\mathtt{VARIABLES}| \geq 2$). Let n and $\mathcal{D}$ respectively denote the number of variables of the collection VARIABLES, and the union of the domains of the variables of VARIABLES. Clearly, the maximum number of changes (i.e., the number of times the constraint $\mathtt{VAR}_i$ CTR $\mathtt{VAR}_{i+1}$ $(1 \leq i < n)$ holds) cannot exceed the quantity $m = \min(n-1, \overline{\mathtt{NCHANGE}})$. The $(m+1) \cdot |\mathcal{D}| + 2$ states of the automaton that only accepts all the solutions to the CHANGE constraint are defined in the following way:

- We have an initial state labelled by s_I.
- We have $m \cdot |\mathcal{D}|$ intermediate states labelled by s_{ij} $(i \in \mathcal{D}, j \in [0, m])$. The first subscript i of state s_{ij} corresponds to the value currently encountered. The second subscript j denotes the number of already encountered satisfied constraints of the form $\mathtt{VAR}_i$ CTR $\mathtt{VAR}_{i+1}$ from the initial state s_I to the state s_{ij}.
- We have an accepting state labelled by s_F.

Four classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by i from the initial state s_I to the state s_{i0}, $(i \in \mathcal{D})$.
2. There is a transition, labelled by j, from every state s_{ij}, $(i \in \mathcal{D}, j \in [0, m])$, to the accepting state s_F.
3. $\forall i \in \mathcal{D},\ \forall j \in [0, m],\ \forall k \in \mathcal{D} \cap \{k \mid i \neg$ `CTR` $k\}$ there is a transition labelled by k from s_{ij} to s_{kj} (i.e., the counter j does not change for values k such that constraint i `CTR` k does not hold).
4. $\forall i \in \mathcal{D},\ \forall j \in [0, m-1],\ \forall k \in \mathcal{D} \setminus \{k \mid i \neg$ `CTR` $k\}$ there is a transition labelled by k from s_{ij} to s_{kj+1} (i.e., the counter j is incremented by $+1$ for values k such that constraint i `CTR` k holds).

We have $|\mathcal{D}|$ transitions of type 1, $|\mathcal{D}| \cdot (m+1)$ transitions of type 2, and at least $|\mathcal{D}|^2 \cdot m$ transitions of types 3 and 4. Since the maximum value of m is equal to $n-1$, in the worst case we have at least $|\mathcal{D}|^2 \cdot (n-1)$ transitions. This leads to a worst case time complexity of $O(|\mathcal{D}|^2 \cdot n^2)$ if we use Pesant's algorithm for filtering the REGULAR constraint [317].

Figure 5.162 depicts the corresponding counter free non deterministic automaton associated with the CHANGE constraint under the hypothesis that (1) all variables of `VARIABLES` are assigned a value in $\{0, 1, 2, 3\}$, (2) $|$`VARIABLES`$|$ is equal to 4, and (3) `CTR` is equal to $\neq$.

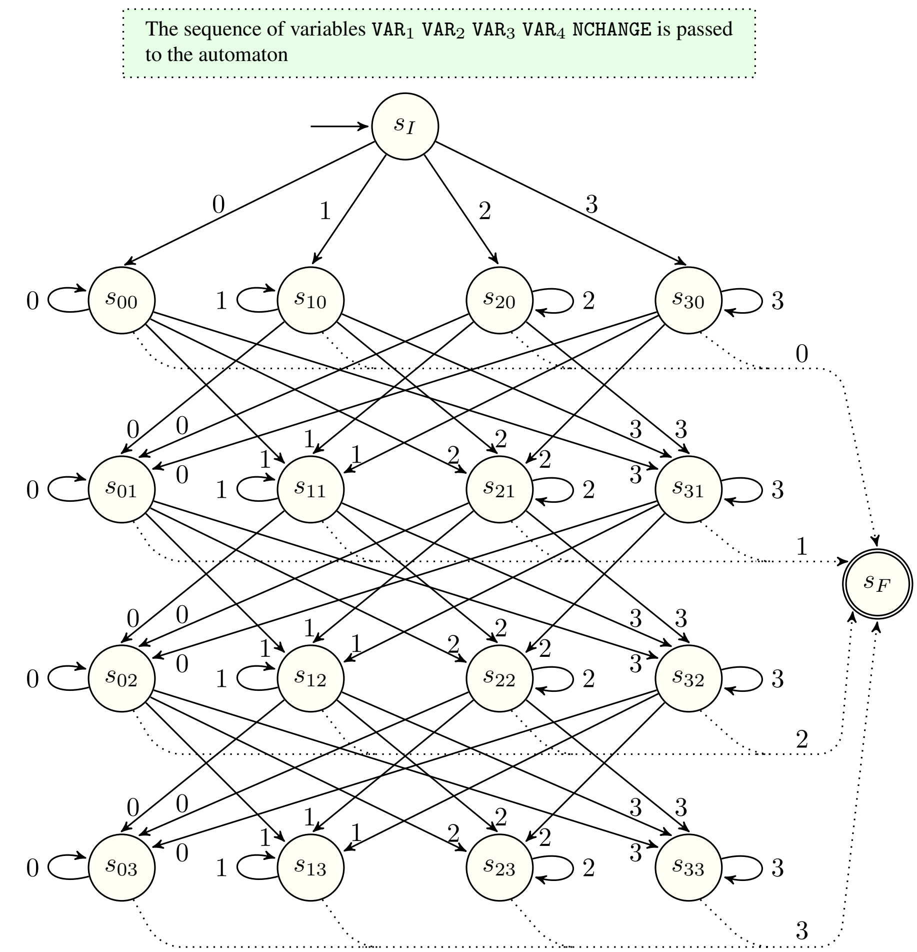


Figure 5.162: Counter free non deterministic automaton of the CHANGE(NCHANGE, $\langle$VAR$_1$, VAR$_2$, VAR$_3$, VAR$_4\rangle$, $\neq$) constraint assuming VAR$_i \in [0,3]$ $(1 \leq i \leq 3)$, with initial state s_I and accepting state s_F

5.64 CHANGE_CONTINUITY

▶ ▷ ◁ ◀

DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint

$$\text{CHANGE\_CONTINUITY}\left(\begin{array}{l}\text{NB\_PERIOD\_CHANGE},\\ \text{NB\_PERIOD\_CONTINUITY},\\ \text{MIN\_SIZE\_CHANGE},\\ \text{MAX\_SIZE\_CHANGE},\\ \text{MIN\_SIZE\_CONTINUITY},\\ \text{MAX\_SIZE\_CONTINUITY},\\ \text{NB\_CHANGE},\\ \text{NB\_CONTINUITY},\\ \text{VARIABLES},\\ \text{CTR}\end{array}\right)$$

Arguments

NB_PERIOD_CHANGE	:	dvar
NB_PERIOD_CONTINUITY	:	dvar
MIN_SIZE_CHANGE	:	dvar
MAX_SIZE_CHANGE	:	dvar
MIN_SIZE_CONTINUITY	:	dvar
MAX_SIZE_CONTINUITY	:	dvar
NB_CHANGE	:	dvar
NB_CONTINUITY	:	dvar
VARIABLES	:	collection(var−dvar)
CTR	:	atom

Restrictions

NB_PERIOD_CHANGE ≥ 0
NB_PERIOD_CONTINUITY ≥ 0
MIN_SIZE_CHANGE ≥ 0
MAX_SIZE_CHANGE $\geq$ MIN_SIZE_CHANGE
MIN_SIZE_CONTINUITY ≥ 0
MAX_SIZE_CONTINUITY $\geq$ MIN_SIZE_CONTINUITY
NB_CHANGE ≥ 0
NB_CONTINUITY ≥ 0
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

On the one hand a *change* is defined by the fact that constraint $\texttt{VARIABLES}[i].\texttt{var}$ CTR $\texttt{VARIABLES}[i+1].\texttt{var}$ holds.
On the other hand a *continuity* is defined by the fact that constraint $\texttt{VARIABLES}[i].\texttt{var}$ CTR $\texttt{VARIABLES}[i+1].\texttt{var}$ does not hold.
A *period of change* on variables

$$\texttt{VARIABLES}[i].\texttt{var}, \texttt{VARIABLES}[i+1].\texttt{var}, \ldots, \texttt{VARIABLES}[j].\texttt{var} \ (i < j)$$

is defined by the fact that all constraints $\texttt{VARIABLES}[k].\texttt{var}$ CTR $\texttt{VARIABLES}[k+1].\texttt{var}$ hold for $k \in [i, j-1]$.
A *period of continuity* on variables

$$\texttt{VARIABLES}[i].\texttt{var}, \texttt{VARIABLES}[i+1].\texttt{var}, \ldots, \texttt{VARIABLES}[j].\texttt{var} \ (i < j)$$

is defined by the fact that all constraints $\texttt{VARIABLES}[k].\texttt{var}$ CTR $\texttt{VARIABLES}[k+1].\texttt{var}$ do not hold for $k \in [i, j-1]$.

Purpose

The constraint CHANGE_CONTINUITY holds if and only if:

- NB_PERIOD_CHANGE is equal to the number of periods of change,
- NB_PERIOD_CONTINUITY is equal to the number of periods of continuity,
- MIN_SIZE_CHANGE is equal to the number of variables of the smallest period of change,
- MAX_SIZE_CHANGE is equal to the number of variables of the largest period of change,
- MIN_SIZE_CONTINUITY is equal to the number of variables of the smallest period of continuity,
- MAX_SIZE_CONTINUITY is equal to the number of variables of the largest period of continuity,
- NB_CHANGE is equal to the total number of changes,
- NB_CONTINUITY is equal to the total number of continuities.

Example

$$(3, 2, 2, 4, 2, 4, 6, 4, \langle 1, 3, 1, 8, 8, 4, 7, 7, 7, 7, 2\rangle, \neq)$$

Figure 5.163 makes clear the different parameters that are associated with the given example for the collection $\texttt{VARIABLES} = \langle 1, 3, 1, 8, 8, 4, 7, 7, 7, 7, 2\rangle$. We place character | for representing a change and a blank for a continuity. On top of the solution we represent the different periods of change, while below we show the different periods of continuity. The CHANGE_CONTINUITY constraint holds since:

- Its number of periods of change NB_PERIOD_CHANGE is equal to 3 (i.e., the 3 periods depicted on top of Figure 5.163),
- Its number of periods of continuity NB_PERIOD_CONTINUITY is equal to 2 (i.e., the 2 periods depicted below Figure 5.163),
- The number of variables of its smallest period of change MIN_SIZE_CHANGE is equal to 2 (i.e., the number of variables involved in the third period of change 7 2 depicted on top of Figure 5.163),

- The number of variables of the largest period of change MAX_SIZE_CHANGE is equal to 4 (i.e., the number of variables involved in the first period of change 1 3 1 8 depicted on top of Figure 5.163),

- The number of variables of the smallest period of continuity MIN_SIZE_CONTINUITY is equal to 2 (i.e., the number of variables involved in the first period 8 8 depicted below Figure 5.163),

- The number of variables of the largest period of continuity MAX_SIZE_CONTINUITY is equal to 4 (i.e., the number of variables involved in the second period 7 7 7 7 depicted below Figure 5.163),

- The total number of changes NB_CHANGE is equal to 6 (i.e., the number of occurrences of character | in Figure 5.163),

- The total number of continuities NB_CONTINUITY is equal to 4.

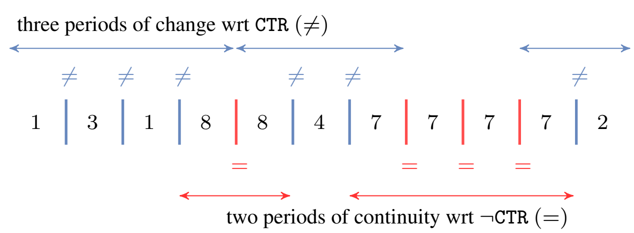


Figure 5.163: Illustration of the **Example** slot: periods of changes and periods of continuities wrt the constraint CTR equal to $\neq$

Typical

NB_PERIOD_CHANGE > 0
NB_PERIOD_CONTINUITY > 0
MIN_SIZE_CHANGE > 0
MIN_SIZE_CONTINUITY > 0
NB_CHANGE > 0
NB_CONTINUITY > 0
|VARIABLES| > 1
range(VARIABLES.var) > 1
CTR $\in [\neq]$

Symmetry

One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: NB_PERIOD_CHANGE determined by VARIABLES and CTR.
- Functional dependency: NB_PERIOD_CONTINUITY determined by VARIABLES and CTR.
- Functional dependency: MIN_SIZE_CHANGE determined by VARIABLES and CTR.
- Functional dependency: MAX_SIZE_CHANGE determined by VARIABLES and CTR.
- Functional dependency: MIN_SIZE_CONTINUITY determined by VARIABLES and CTR.
- Functional dependency: MAX_SIZE_CONTINUITY determined by VARIABLES and CTR.
- Functional dependency: NB_CHANGE determined by VARIABLES and CTR.
- Functional dependency: NB_CONTINUITY determined by VARIABLES and CTR.

Remark

If the variables of the collection VARIABLES have to take distinct values between 1 and the total number of variables, we have what is called a permutation. In this case, if we choose the binary constraint $<$, then MAX_SIZE_CHANGE gives the size of the longest run of the permutation; A *run* is a maximal increasing contiguous subsequence in a permutation.

See also

common keyword: GROUP, GROUP_SKIP_ISOLATED_ITEM, STRETCH_PATH *(timetabling constraint)*.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence, run of a permutation, permutation.

constraint arguments: reverse of a constraint.

constraint network structure: sliding cyclic(1) constraint network(2), sliding cyclic(1) constraint network(3).

constraint type: timetabling constraint.

filtering: glue matrix.

final graph structure: connected component, apartition, acyclic, bipartite, no loop.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	• **NCC**= NB_PERIOD_CHANGE • **MIN_NCC**= MIN_SIZE_CHANGE • **MAX_NCC**= MAX_SIZE_CHANGE • **NARC**= NB_CHANGE
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var$\neg$ CTR variables2.var
Graph property(ies)	• **NCC**= NB_PERIOD_CONTINUITY • **MIN_NCC**= MIN_SIZE_CONTINUITY • **MAX_NCC**= MAX_SIZE_CONTINUITY • **NARC**= NB_CONTINUITY
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model

We use two graph constraints to respectively catch the constraints on the period of changes and of the period of continuities. In both case each period corresponds to a connected component of the final graph.

Parts (A) and (B) of Figure 5.164 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot.

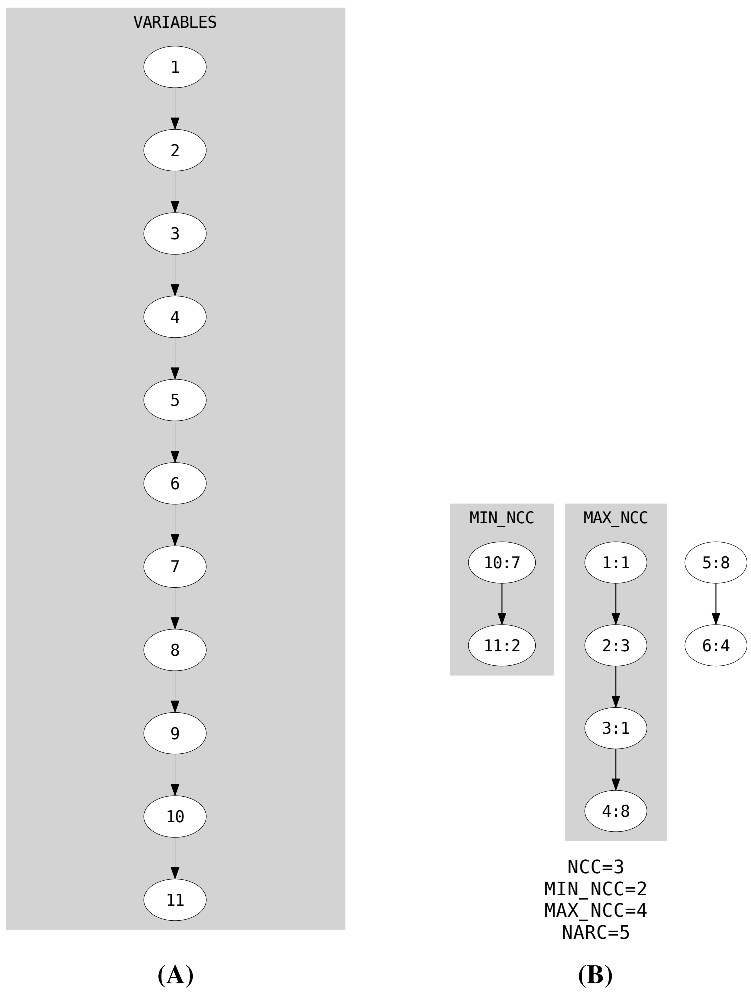
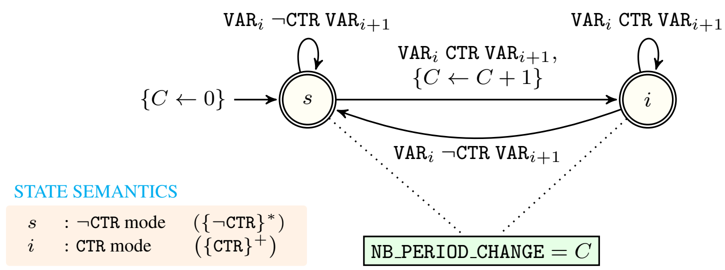


Figure 5.164: Initial and final graph of the CHANGE_CONTINUITY constraint

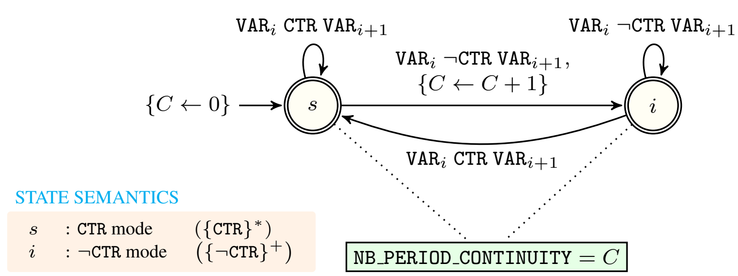
Automaton Figures 5.165 , 5.166 , 5.169 , 5.170 , 5.173 , 5.174 and 5.177 depict the automata associated with the different graph parameters of the CHANGE_CONTINUITY constraint. For the automata that respectively compute NB_PERIOD_CHANGE, NB_PERIOD_CONTINUITY MIN_SIZE_CHANGE, MIN_SIZE_CONTINUITY MAX_SIZE_CHANGE, MAX_SIZE_CONTINUITY NB_CHANGE and NB_CONTINUITY we have a 0-1 signature variable S_i for each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $\mathtt{VAR}_i$ CTR $\mathtt{VAR}_{i+1} \Leftrightarrow S_i$.



Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counters values C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	i
s	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}$
i	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}-1$

Figure 5.165: Automaton for the NB_PERIOD_CHANGE argument of the CHANGE_CONTINUITY constraint and its glue matrix; note that the reverse of CHANGE_CONTINUITY with CTR $\in \{=,\neq\}$ is the same constraint, while the reverse with CTR $\in \{<\}$ (resp. CTR $\in \{\leq\}$) is CTR $\in \{>\}$ (resp. CTR $\in \{\geq\}$).



Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counters values C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	i
s	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}$
i	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}-1$

Figure 5.166: Automaton for the NB_PERIOD_CONTINUITY argument of the CHANGE_CONTINUITY constraint and its glue matrix; note that the reverse of CHANGE_CONTINUITY with CTR $\in \{=,\neq\}$ is the same constraint, while the reverse with CTR $\in \{<\}$ (resp. CTR $\in \{\leq\}$) is CTR $\in \{>\}$ (resp. CTR $\in \{\geq\}$).

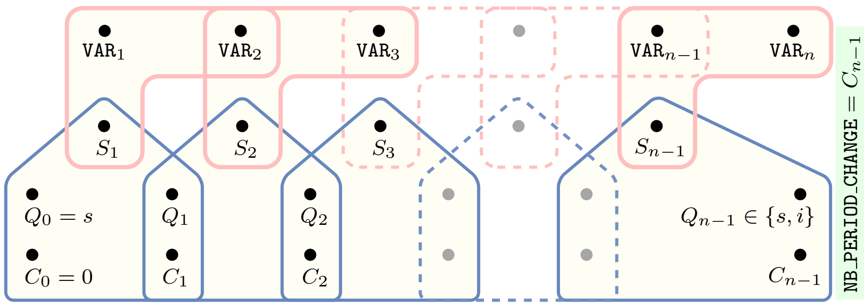


Figure 5.167: Hypergraph of the reformulation corresponding to the automaton of the NB_PERIOD_CHANGE argument of the CHANGE_CONTINUITY constraint

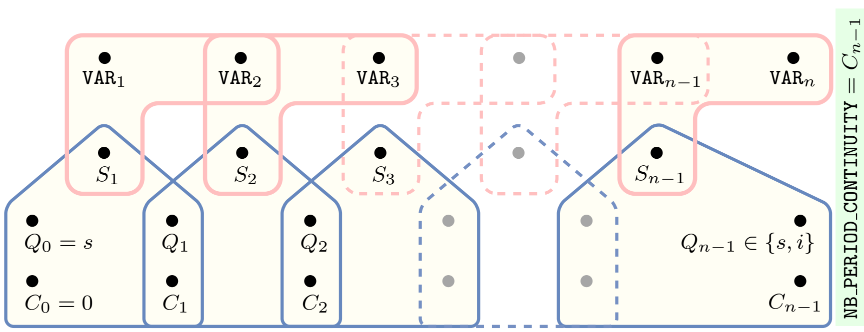
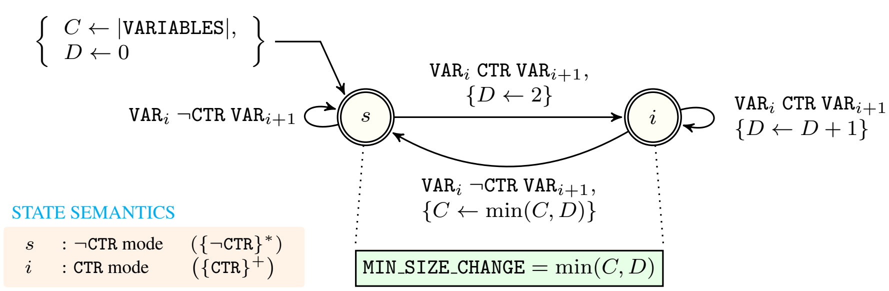


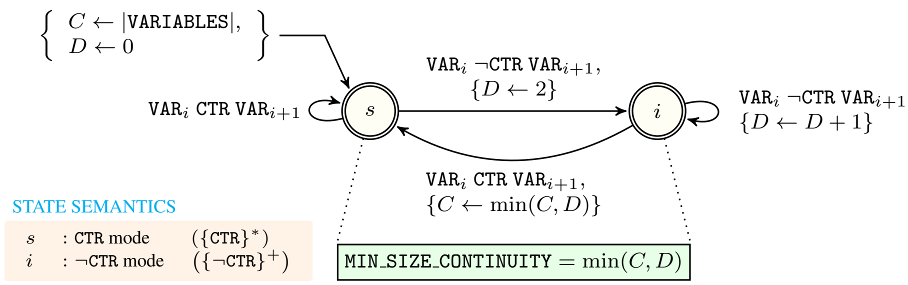
Figure 5.168: Hypergraph of the reformulation corresponding to the automaton of the NB_PERIOD_CONTINUITY argument of the CHANGE_CONTINUITY constraint



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	i
s	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overleftarrow{D}, \overleftarrow{C})$
i	$\min(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D} - 1, \overleftarrow{C})$

Figure 5.169: Automaton for the MIN_SIZE_CHANGE argument of the CHANGE_CONTINUITY constraint; its glue matrix when CTR $\in \{=, \neq\}$.



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	i
s	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overleftarrow{D}, \overleftarrow{C})$
i	$\min(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D} - 1, \overleftarrow{C})$

Figure 5.170: Automaton for the MIN_SIZE_CONTINUITY argument of the CHANGE_CONTINUITY constraint; its glue matrix when CTR $\in \{=, \neq\}$.

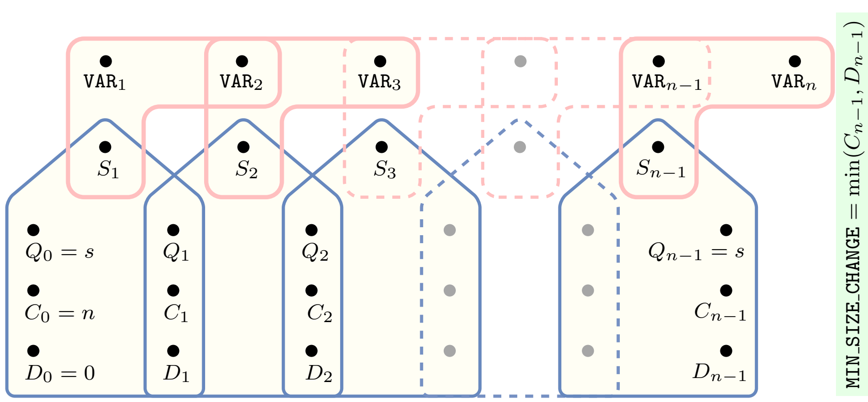


Figure 5.171: Hypergraph of the reformulation corresponding to the automaton of the MIN_SIZE_CHANGE argument of the CHANGE_CONTINUITY constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

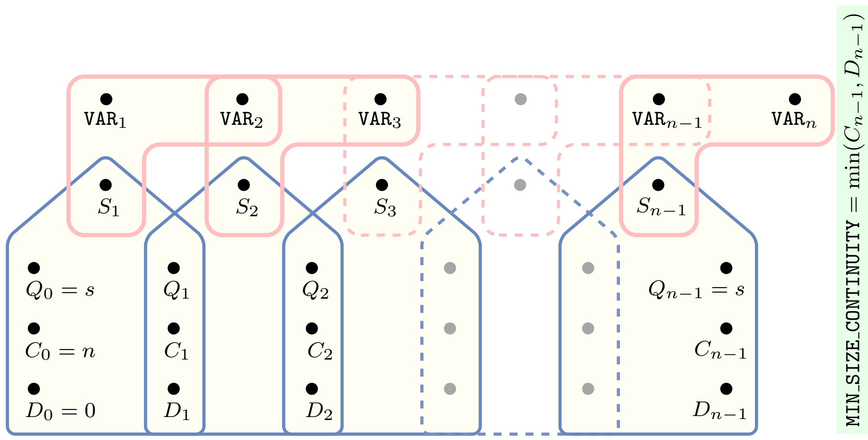
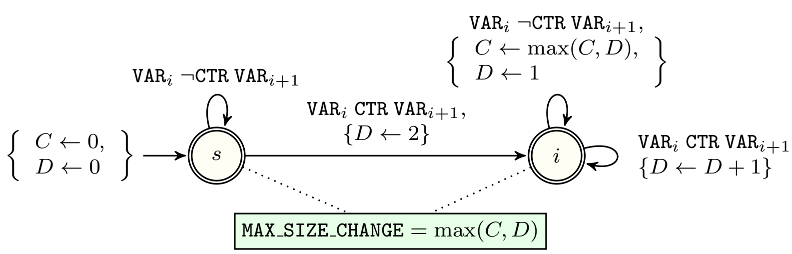


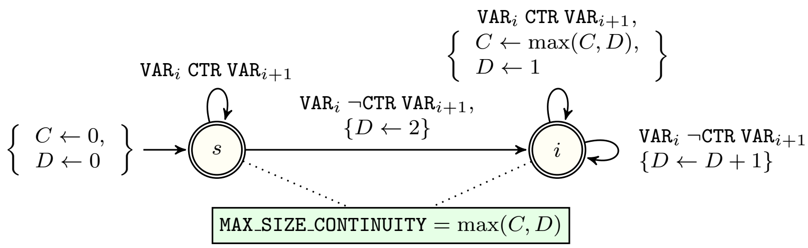
Figure 5.172: Hypergraph of the reformulation corresponding to the automaton of the MIN_SIZE_CONTINUITY argument of the CHANGE_CONTINUITY constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	i
s	0	$\max\left(\overleftarrow{D}, \overleftarrow{C}\right)$
i	$\max\left(\overrightarrow{C}, \overrightarrow{D}\right)$	$\max\left(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D} - 1, \overleftarrow{C}\right)$

Figure 5.173: Automaton for the MAX_SIZE_CHANGE argument of the CHANGE_CONTINUITY constraint; its glue matrix when $\mathtt{CTR} \in \{=, \neq\}$.



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	i
s	0	$\max\left(\overleftarrow{D}, \overleftarrow{C}\right)$
i	$\max\left(\overrightarrow{C}, \overrightarrow{D}\right)$	$\max\left(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D} - 1, \overleftarrow{C}\right)$

Figure 5.174: Automaton for the MAX_SIZE_CONTINUITY argument of the CHANGE_CONTINUITY constraint; its glue matrix when CTR $\in \{=, \neq\}$.

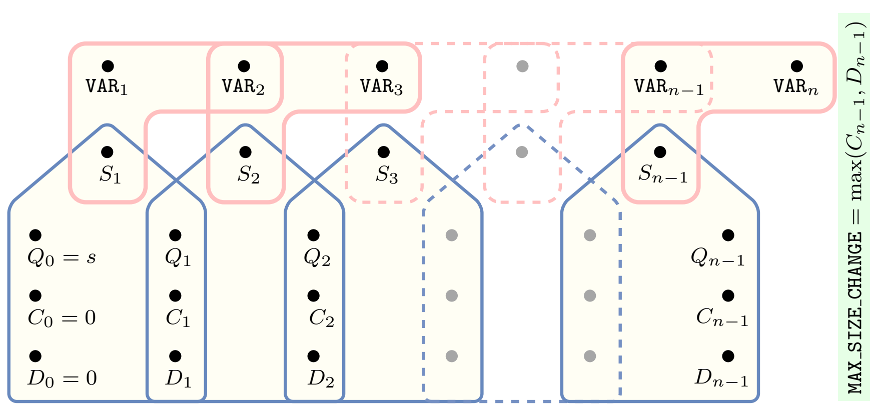


Figure 5.175: Hypergraph of the reformulation corresponding to the automaton of the MAX_SIZE_CHANGE argument of the CHANGE_CONTINUITY constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

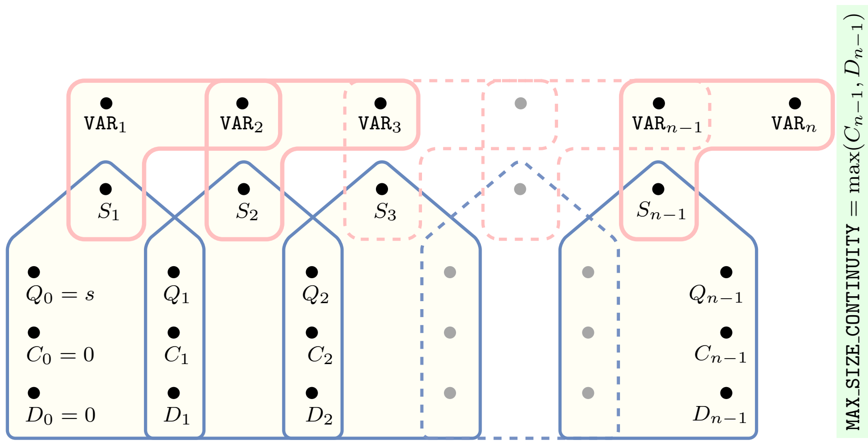


Figure 5.176: Hypergraph of the reformulation corresponding to the automaton of the MAX_SIZE_CONTINUITY argument of the CHANGE_CONTINUITY constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

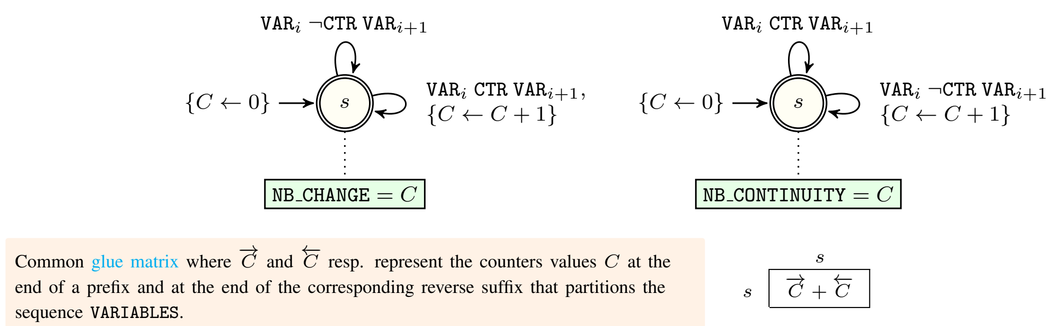


Figure 5.177: Automata for the NB_CHANGE and NB_CONTINUITY arguments of the CHANGE_CONTINUITY constraint; their common glue matrix when arg $CTR \in \{=, \neq\}$.

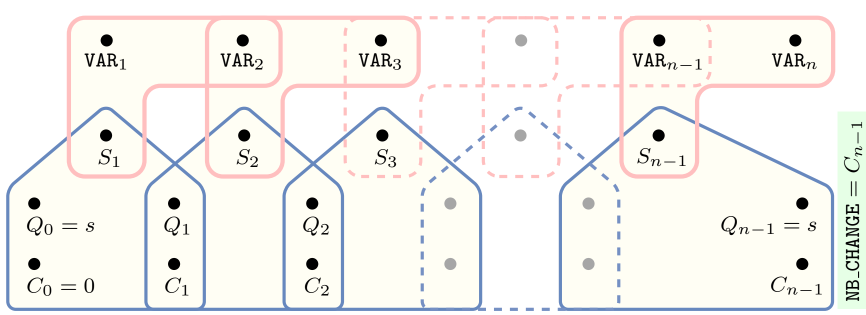


Figure 5.178: Hypergraph of the reformulation corresponding to the automaton of the NB_CHANGE argument of the CHANGE_CONTINUITY constraint

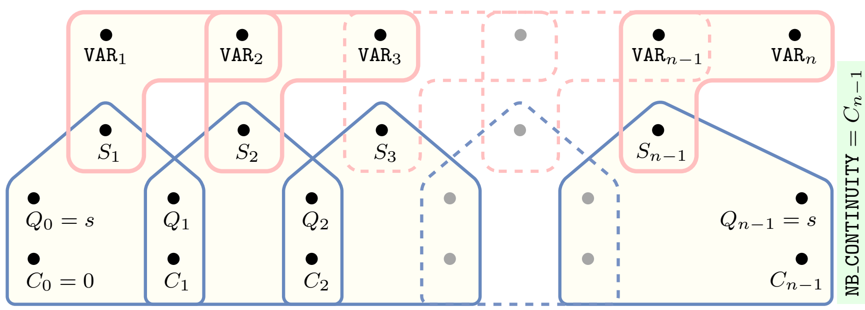


Figure 5.179: Hypergraph of the reformulation corresponding to the automaton of the NB_CONTINUITY argument of the CHANGE_CONTINUITY constraint

5.65 CHANGE_PAIR

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from CHANGE.

Constraint CHANGE_PAIR(NCHANGE, PAIRS, CTRX, CTRY)

Arguments

NCHANGE : dvar
PAIRS : collection(x−dvar, y−dvar)
CTRX : atom
CTRY : atom

Restrictions

NCHANGE ≥ 0
NCHANGE $<$ |PAIRS|
required(PAIRS, [x, y])
CTRX $\in [=, \neq, <, \geq, >, \leq]$
CTRY $\in [=, \neq, <, \geq, >, \leq]$

Purpose NCHANGE is the number of times that the following disjunction holds: $(X_1$ CTRX $X_2) \vee (Y_1$ CTRY $Y_2)$, where (X_1, Y_1) and (X_2, Y_2) correspond to consecutive pairs of variables of the collection PAIRS.

Example

$$\left(3, \left\langle \begin{array}{ll} \mathtt{x}-3 & \mathtt{y}-5, \\ \mathtt{x}-3 & \mathtt{y}-7, \\ \mathtt{x}-3 & \mathtt{y}-7, \\ \mathtt{x}-3 & \mathtt{y}-8, \\ \mathtt{x}-3 & \mathtt{y}-4, \\ \mathtt{x}-3 & \mathtt{y}-7, \\ \mathtt{x}-1 & \mathtt{y}-3, \\ \mathtt{x}-1 & \mathtt{y}-6, \\ \mathtt{x}-1 & \mathtt{y}-6, \\ \mathtt{x}-3 & \mathtt{y}-7 \end{array} \right\rangle, \neq, > \right)$$

3=3=3=3=3=3≠1=1=1≠3
5≤7≤7≤8>4≤7>3≤6≤6≤7

In the example we have the following 3 changes:

- One change between pairs x − 3 y − 8 and x − 3 y − 4 since $3 \neq 3 \vee 8 > 4$,
- One change between pairs x − 3 y − 7 and x − 1 y − 3 since $3 \neq 1 \vee 7 > 3$,
- One change between pairs x − 1 y − 6 and x − 3 y − 7 since $1 \neq 3 \vee 6 > 7$.

Consequently the CHANGE_PAIR constraint holds since its first argument NCHANGE is assigned value 3.

Typical

NCHANGE > 0
|PAIRS| > 1
range(PAIRS.x) > 1
range(PAIRS.y) > 1

Symmetries

- One and the same constant can be added to the x attribute of all items of PAIRS.
- One and the same constant can be added to the y attribute of all items of PAIRS.

Arg. properties

Functional dependency: NCHANGE determined by PAIRS, CTRX and CTRY.

Usage

Here is a typical example where this constraint is useful. Assume we have to produce a set of cables. A given quality and a given cross-section that respectively correspond to the x and y attributes of the previous pairs of variables characterise each cable. The problem is to sequence the different cables in order to minimise the number of times two consecutive wire cables C_1 and C_2 verify the following property: C_1 and C_2 do not have the same quality or the cross section of C_1 is greater than the cross section of C_2.

See also

generalisation: CHANGE_VECTORS *(*pair *of* variables *replaced by* vector*)*.

specialisation: CHANGE *(*pair *of* variables *replaced by* variable*)*.

Keywords

characteristic of a constraint: pair, automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: sliding cyclic(2) constraint network(2).

constraint type: timetabling constraint.

final graph structure: acyclic, bipartite, no loop.

modelling: number of changes, functional dependency.

Arc input(s)	PAIRS
Arc generator	$PATH \mapsto$ collection(pairs1, pairs2)
Arc arity	2
Arc constraint(s)	pairs1.x CTRX pairs2.x $\vee$ pairs1.y CTRY pairs2.y
Graph property(ies)	**NARC** = NCHANGE
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Same as CHANGE, except that each item has two attributes x and y.

Parts (A) and (B) of Figure 5.180 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

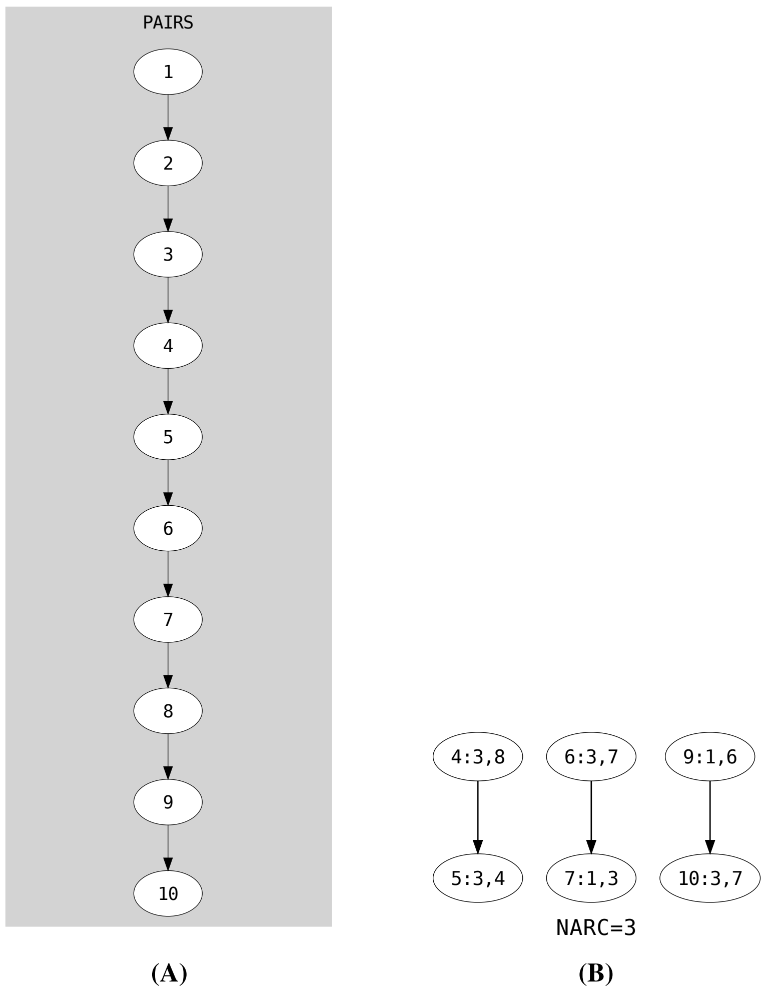


Figure 5.180: Initial and final graph of the CHANGE_PAIR constraint

Automaton Figure 5.181 depicts the automaton associated with the CHANGE_PAIR constraint. To each pair of consecutive pairs $((\mathtt{X}_i, \mathtt{Y}_i), (\mathtt{X}_{i+1}, \mathtt{Y}_{i+1}))$ of the collection PAIRS corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{X}_i$, $\mathtt{Y}_i$, $\mathtt{X}_{i+1}$, $\mathtt{Y}_{i+1}$ and S_i: $(\mathtt{X}_i\ \mathtt{CTRX}\ \mathtt{X}_{i+1}) \vee (\mathtt{Y}_i\ \mathtt{CTRY}\ \mathtt{Y}_{i+1}) \Leftrightarrow S_i$.

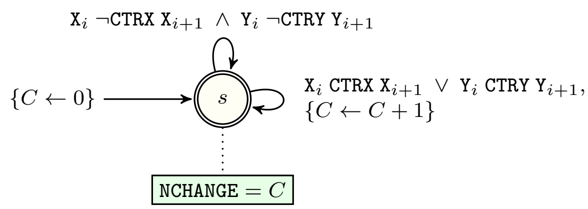


Figure 5.181: Automaton of the CHANGE_PAIR constraint

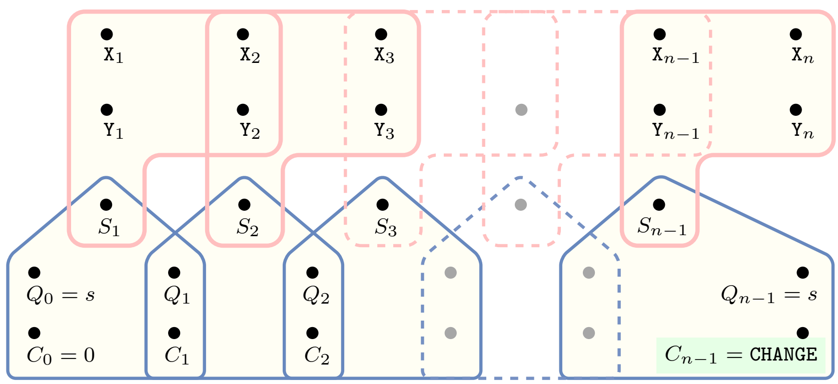


Figure 5.182: Hypergraph of the reformulation corresponding to the automaton of the CHANGE_PAIR constraint

5.66 CHANGE_PARTITION

DESCRIPTION LINKS GRAPH

Origin

Derived from CHANGE.

Constraint

CHANGE_PARTITION(NCHANGE, VARIABLES, PARTITIONS)

Type

VALUES : collection(val−int)

Arguments

NCHANGE : dvar
VARIABLES : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

|VALUES| ≥ 1
required(VALUES, val)
distinct(VALUES, val)
NCHANGE ≥ 0
NCHANGE < |VARIABLES|
required(VARIABLES, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2

Purpose

NCHANGE is the number of times that the following constraint holds: X and Y do not belong to the same partition of the collection PARTITIONS, where X and Y correspond to consecutive variables of the collection VARIABLES.

Example

$$\left(\begin{array}{l} 2, \langle 6,6,2,1,3,3,1,6,2,2,2\rangle, \\ \langle \mathtt{p}-\langle 1,3\rangle, \mathtt{p}-\langle 4\rangle, \mathtt{p}-\langle 2,6\rangle\rangle \end{array}\right)$$

In the example we have the following two changes:

- One change between values 2 and 1 (since 2 and 1 respectively belong to the third and the first partition),
- One change between values 1 and 6 (since 1 and 6 respectively belong to the first and the third partition).

Consequently the CHANGE_PARTITION constraint holds since its first argument NCHANGE is assigned to 2.

Typical

NCHANGE > 0
|VARIABLES| > 1
range(VARIABLES.var) > 1
|VARIABLES| > |PARTITIONS|

Symmetries

- Items of VARIABLES can be reversed.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Arg. properties

Functional dependency: NCHANGE determined by VARIABLES and PARTITIONS.

Usage

This constraint is useful for the following problem: Assume you have to produce a set of orders, each order belonging to a given family. In the context of the **Example** slot we have three families that respectively correspond to values $1, 3$, to value 4 and to values $2, 6$. We would like to sequence the orders in such a way that we minimise the number of times two consecutive orders do not belong to the same family.

Algorithm

[32].

See also

common keyword: CHANGE *(number of changes in a sequence of* `variables` *with respect to a* `binary constraint`*)*.

used in graph description: IN_SAME_PARTITION.

Keywords

characteristic of a constraint: partition.

constraint arguments: pure functional dependency.

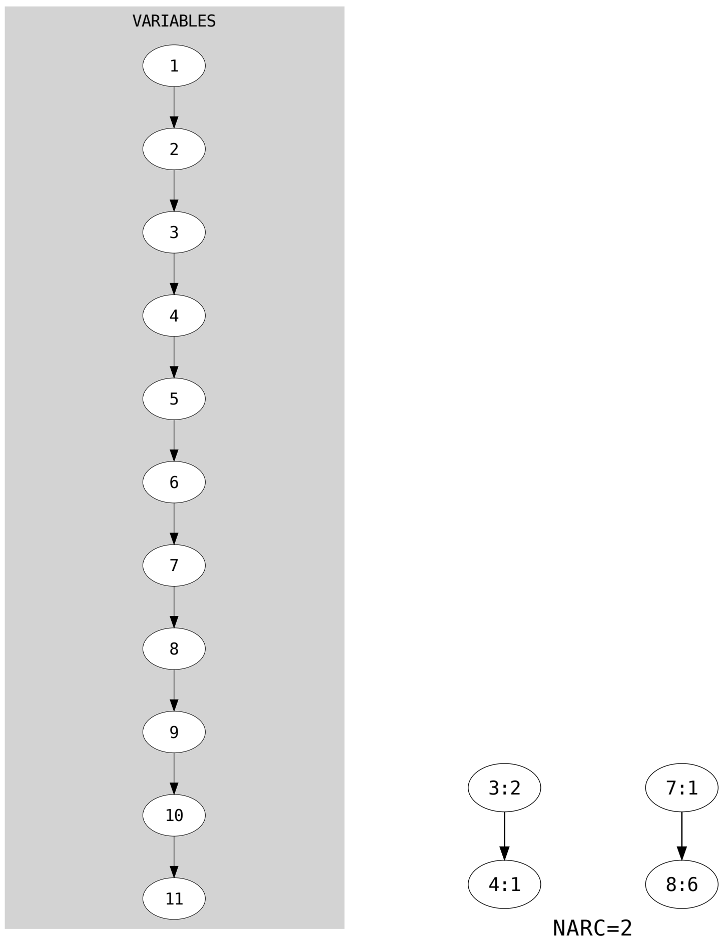
constraint type: timetabling constraint.

final graph structure: acyclic, bipartite, no loop.

modelling: number of changes, functional dependency.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	**NARC** = NCHANGE
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.183 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.183: Initial and final graph of the CHANGE_PARTITION constraint

5.67 CHANGE_VECTORS

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin

Derived from CHANGE

Constraint

CHANGE_VECTORS(NCHANGE, VECTORS, CTRS)

Types

VECTOR : collection(var−dvar)
CTR : atom

Arguments

NCHANGE : dvar
VECTORS : collection(vec − VECTOR)
CTRS : collection(ctr − CTR)

Restrictions

$|\texttt{VECTOR}| \geq 1$
required(VECTOR, var)
$\texttt{CTR} \in [=, \neq, <, \geq, >, \leq]$
$\texttt{NCHANGE} \geq 0$
$\texttt{NCHANGE} < |\texttt{VECTORS}|$
required(VECTORS, vec)
same_size(VECTORS, vec)
required(CTRS, ctr)
$|\texttt{CTRS}| = |\texttt{VECTOR}|$

Purpose

Let us note $\texttt{VECTOR}_1, \texttt{VECTOR}_2, \ldots, \texttt{VECTOR}_n$ the vectors of the VECTORS collection, and d the number of components of each vector (all vectors have the same size). NCHANGE is the number of times that the following disjunctions holds where $i \in [1, n-1]$

$$(\texttt{VECTOR}_i.\texttt{vec}[1]\ \texttt{CTRS}[1]\ \texttt{VECTOR}_{i+1}.\texttt{vec}[1]) \vee$$
$$(\texttt{VECTOR}_i.\texttt{vec}[2]\ \texttt{CTRS}[2]\ \texttt{VECTOR}_{i+1}.\texttt{vec}[2]) \vee$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \vee$$
$$(\texttt{VECTOR}_i.\texttt{vec}[d]\ \texttt{CTRS}[d]\ \texttt{VECTOR}_{i+1}.\texttt{vec}[d]).$$

Example

$$\left(\begin{array}{l} 3, \left\langle \begin{array}{l} \texttt{vec} - \langle 4,0\rangle, \\ \texttt{vec} - \langle 4,0\rangle, \\ \texttt{vec} - \langle 4,5\rangle, \\ \texttt{vec} - \langle 3,4\rangle, \\ \texttt{vec} - \langle 3,4\rangle, \\ \texttt{vec} - \langle 3,4\rangle, \\ \texttt{vec} - \langle 4,0\rangle \end{array} \right\rangle, \\ \langle \neq, \neq \rangle \end{array}\right)$$

In the example we have the following 3 changes:

- One change between $\langle 4, 0\rangle$ and $\langle 4, 5\rangle$ since $4 \neq 4 \vee 0 \neq 5$,
- One change between $\langle 4, 5\rangle$ and $\langle 3, 4\rangle$ since $4 \neq 3 \vee 5 \neq 4$,

- One change between $\langle 3, 4\rangle$ and $\langle 4, 0\rangle$ since $3 \neq 4 \vee 4 \neq 0$.

Consequently the CHANGE_VECTORS constraint holds since its first argument NCHANGE is assigned value 3.

Typical

$\mathtt{CTR} \in [\neq]$
$|\mathtt{VECTOR}| > 1$
$\mathtt{NCHANGE} > 0$
$|\mathtt{VECTORS}| > 1$

Arg. properties

Functional dependency: NCHANGE determined by VECTORS and CTRS.

See also

specialisation: CHANGE *(vector replaced by* variable*)*, CHANGE_PAIR *(vector replaced by* pair *of* variables*)*.

Keywords

characteristic of a constraint: automaton, automaton with counters, vector.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

modelling: number of changes, functional dependency.

5.68 CIRCUIT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [267]

Constraint CIRCUIT(NODES)

Synonyms ATOUR, CYCLE.

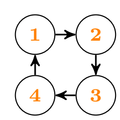
Argument NODES : collection(index−int, succ−dvar)

Restrictions
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |NODES|

Purpose Enforce to cover a digraph G described by the NODES collection with one circuit visiting once all vertices of G.

Example

$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-2,\\ \texttt{index}-2 & \texttt{succ}-3,\\ \texttt{index}-3 & \texttt{succ}-4,\\ \texttt{index}-4 & \texttt{succ}-1\end{array}\right\rangle\right)$$



The CIRCUIT constraint holds since its NODES argument depicts the following Hamiltonian circuit visiting successively the vertices 1, 2, 3, 4 and 1.

All solutions Figure 5.184 gives all solutions to the following non ground instance of the CIRCUIT constraint: $\mathtt{S}_1 \in [3,4]$, $\mathtt{S}_2 \in [1,2]$, $\mathtt{S}_3 \in [1,4]$, $\mathtt{S}_4 \in [2,4]$, CIRCUIT($\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4\rangle$).

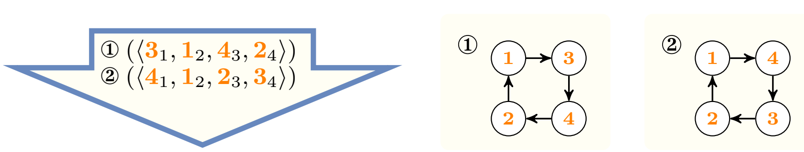


Figure 5.184: All solutions corresponding to the non ground example of the CIRCUIT constraint of the **All solutions** slot; in the left-hand side the index attributes are displayed as indices of the succ attribute, while in the right-hand side they are directly displayed within each node.

Typical |NODES| > 2

Symmetry Items of NODES are permutable.

Remark In the original CIRCUIT constraint of **CHIP** the `index` attribute was not explicitly present. It was implicitly defined as the position of a variable in a list.

Within the context of linear programming [6] this constraint was introduced under the name ATOUR. In the same context [226, page 380] provides continuous relaxations of the CIRCUIT constraint.

Within the KOALOG constraint system this constraint is called CYCLE.

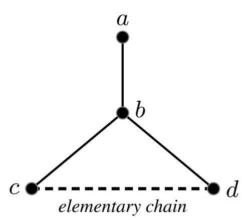
Algorithm Since all `succ` variables of the NODES collection have to take distinct values one can reuse the algorithms associated with the ALLDIFFERENT constraint. A second necessary condition is to have no more than one strongly connected component. Pruning for enforcing this condition can be done by forcing all strong bridges to belong to the final solution, since otherwise the strongly connected component would be broken apart. A third necessary condition is that, if the graph is bipartite then the number of vertices of each class should be identical. Consequently if the number of vertices is odd (i.e., |NODES| is odd) the graph should not be bipartite. Further necessary conditions (useful when the graph is sparse) combining the fact that we have a perfect matching and a single strongly connected component can be found in [392]. These conditions forget about the orientation of the arcs of the graph and characterise new required elementary chains. A typical pattern involving four vertices is depicted by Figure 5.185 where we assume that:

- There is an elementary chain between c and d (depicted by a dashed edge),
- b has exactly 3 neighbours.

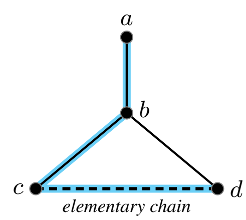
In this context the edge between a and b is mandatory in any covering (i.e., the arc from a to b or the arc from b to a) since otherwise a small circuit involving b, c and d would be created.

When the graph is planar [228][147] one can also use as a necessary condition discovered by Grinberg [210] for pruning.

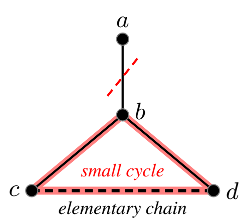
Finally, another approach based an the notion of 1-toughness [125] was proposed in [247] and evaluated for small graphs (i.e., graphs with up to 15 vertices).



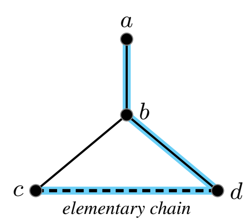
(A) Initial situation: a vertex b with 3 potential neighbours a, c, d with an elementary chain between c and d



(C) The first alternative: an elementary chain between a and d: (a, b) is kept



(B) Removing edge (a, b) leads to a contradiction: a small cycle that does not contain vertex a



(D) The second alternative: an elementary chain between a and c: (a, b) is kept

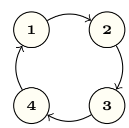
Figure 5.185: Reasoning about elementary chains and degrees: if we have an elementary chain between c and d and if b has 3 neighbours then the edge (a, b) is mandatory.

Reformulation

Let n and $s_1, s_2, \ldots, s_n$ respectively denotes the number of vertices (i.e., $|\texttt{NODES}|$) and the successor variables associated with vertices $1, 2, \ldots, n$. The CIRCUIT constraint can be reformulated as a conjunction of one DOMAIN constraint, two ALLDIFFERENT constraints, and n ELEMENT constraints.

- First, we state an ALLDIFFERENT($\langle s_1, s_2, \ldots, s_n\rangle$) constraint for enforcing distinct values to be assigned to the successor variables.
- Second, the key idea is, starting from vertex 1, to successively extract the vertices $t_1, t_2, \ldots, t_{n-1}$ of the circuit until we come back on vertex 1, where t_i (with $i \in [2, n-1]$) denotes the successor of t_{i-1} and t_1 the successor of vertex 1. Since we have a single circuit all the $t_1, t_2, \ldots, t_{n-1}$ should be different from 1. Consequently we state a DOMAIN($\langle t_1, t_2, \ldots, t_{n_1}\rangle, 2, n$) constraint for declaring their initial domains. To express the link between consecutive t_i we also state a conjunction of n ELEMENT constraints of the form:
  ELEMENT($1, \langle s_1, s_2, \ldots, s_n\rangle, t_1$),
  ELEMENT($t_1, \langle s_1, s_2, \ldots, s_n\rangle, t_2$),
  ....
  ELEMENT($t_{n-1}, \langle s_1, s_2, \ldots, s_n\rangle, 1$).
- Finally we add a redundant constraint for stating that all t_i (with $i \in [1, n-1]$) are distinct, i.e. ALLDIFFERENT($\langle t_1, t_2, \ldots, t_{n-1}\rangle$).

Illustration of the reformulation of CIRCUIT($\langle 1\ 2,\ 2\ 3,\ 3\ 4,\ 4\ 1\rangle$)



ALLDIFFERENT($\langle 2, 3, 4, 1\rangle$)
DOMAIN($\langle \mathbf{2}, \mathbf{3}, \mathbf{4}\rangle, 2, 4$)

ELEMENT($\mathbf{1}$, $\langle 2, 3, 4, 1\rangle$, $\mathbf{2}$)
ELEMENT($\mathbf{2}$, $\langle 2, 3, 4, 1\rangle$, $\mathbf{3}$)
ELEMENT($\mathbf{3}$, $\langle 2, 3, 4, 1\rangle$, $\mathbf{4}$)
ELEMENT($\mathbf{4}$, $\langle 2, 3, 4, 1\rangle$, $\mathbf{1}$)

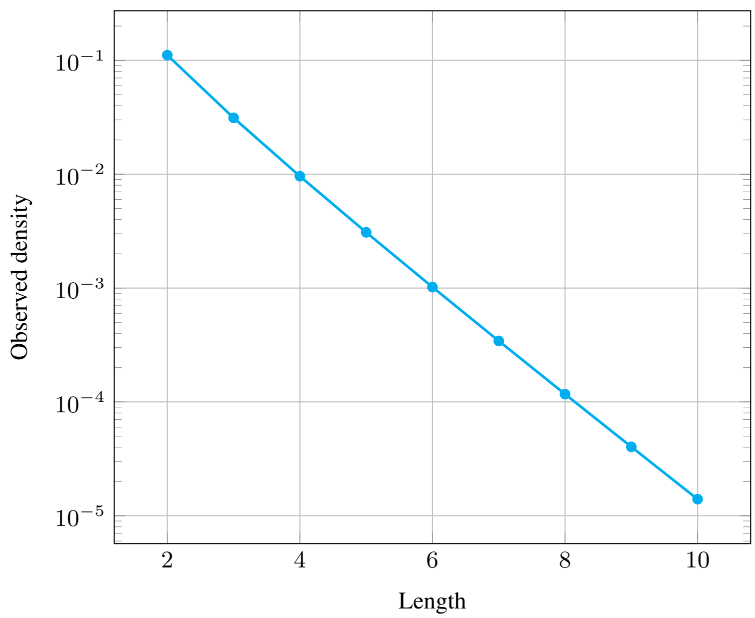
ALLDIFFERENT($\langle \mathbf{2}, \mathbf{3}, \mathbf{4}\rangle$)

Counting

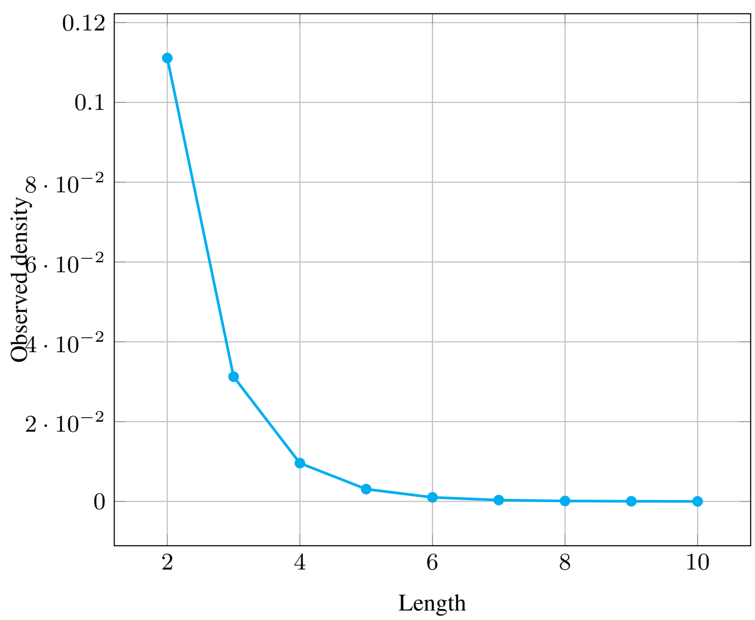
Length (n)	2	3	4	5	6	7	8	9	10
Solutions	1	2	6	24	120	720	5040	40320	362880

Number of solutions for CIRCUIT: domains $0..n$

Solution density for CIRCUIT



Solution density for CIRCUIT



Systems CIRCUIT in **Gecode**, CIRCUIT in **JaCoP**, CIRCUIT in **MiniZinc**, CIRCUIT in **SICStus**.

See also **common keyword:** ALLDIFFERENT *(permutation)*, CIRCUIT_CLUSTER *(graph* *constraint, one_succ*), PATH (*graph partitioning constraint, one_succ*), PROPER_CIRCUIT (*permutation, one_succ*), TOUR (*graph partitioning constraint, Hamiltonian*).

generalisation: CYCLE (*introduce a variable for the number of circuits*).

implies: ALLDIFFERENT, PROPER_CIRCUIT, TWIN.

implies (items to collection): LEX_ALLDIFFERENT.

related: STRONGLY_CONNECTED.

Keywords

combinatorial object: permutation.

constraint type: graph constraint, graph partitioning constraint.

filtering: linear programming, planarity test, strong bridge, DFS-bottleneck.

final graph structure: circuit, one_succ.

problems: Hamiltonian.

puzzles: Euler knight.

Cond. implications

- CIRCUIT(NODES)
  implies CYCLE(NCYCLE, NODES)
  when NCYCLE $= 1$.

- CIRCUIT(NODES)
  with $|$NODES$| > 1$
  implies DERANGEMENT(NODES).

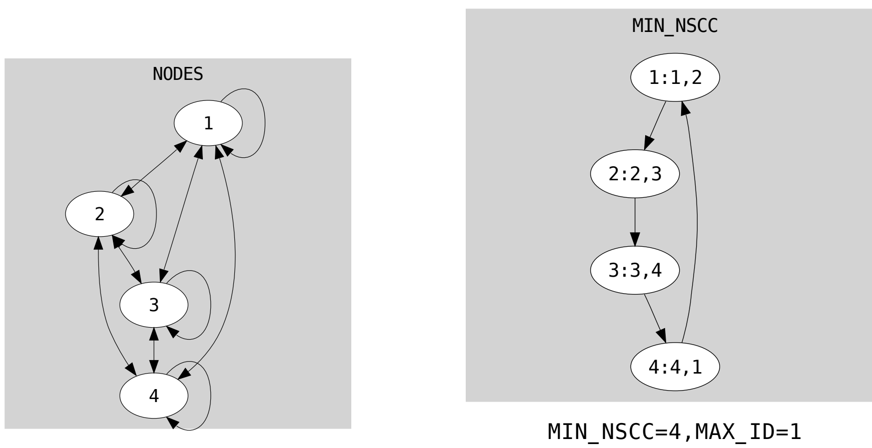
- CIRCUIT(NODES)
  with $|$NODES$| > 1$
  implies K_ALLDIFFERENT(VARS : NODES).

- CIRCUIT(NODES)
  implies PERMUTATION(VARIABLES : NODES).

Arc input(s)	`NODES`
Arc generator	*CLIQUE*$\mapsto$`collection`(`nodes1`, `nodes2`)
Arc arity	2
Arc constraint(s)	`nodes1.succ` = `nodes2.index`
Graph property(ies)	• **MIN_NSCC**= |`NODES`| • **MAX_ID**≤ 1
Graph class	`ONE_SUCC`

Graph model The first graph property enforces to have a single strongly connected component containing |`NODES`| vertices. The second graph property imposes to only have circuits. Since each vertex of the final graph has only one successor we do not need to use set variables for representing the successors of a vertex.

Parts (A) and (B) of Figure 5.186 respectively show the initial and final graph associated with the **Example** slot. The CIRCUIT constraint holds since the final graph consists of one circuit mentioning once every vertex of the initial graph.



(A) **(B)**

Figure 5.186: Initial and final graph of the CIRCUIT constraint

5.69 CIRCUIT_CLUSTER

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Inspired by [263].

Constraint CIRCUIT_CLUSTER(NCIRCUIT, NODES)

Arguments

NCIRCUIT : dvar
NODES : collection(index−int, cluster−int, succ−dvar)

Restrictions

NCIRCUIT ≥ 1
NCIRCUIT $\leq$ |NODES|
required(NODES, [index, cluster, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

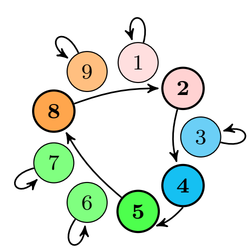
Purpose

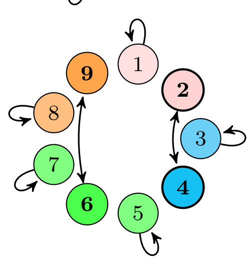
Consider a digraph G, described by the NODES collection, such that its vertices are partitioned among several clusters. NCIRCUIT is the number of circuits containing more than one vertex used for covering G in such a way that each cluster is visited by exactly one circuit of length greater than 1.

Example

$$\left(\begin{array}{l} 1, \left\langle\begin{array}{lll} \text{index}-1 & \text{cluster}-1 & \text{succ}-1, \\ \text{index}-2 & \text{cluster}-1 & \text{succ}-4, \\ \text{index}-3 & \text{cluster}-2 & \text{succ}-3, \\ \text{index}-4 & \text{cluster}-2 & \text{succ}-5, \\ \text{index}-5 & \text{cluster}-3 & \text{succ}-8, \\ \text{index}-6 & \text{cluster}-3 & \text{succ}-6, \\ \text{index}-7 & \text{cluster}-3 & \text{succ}-7, \\ \text{index}-8 & \text{cluster}-4 & \text{succ}-2, \\ \text{index}-9 & \text{cluster}-4 & \text{succ}-9 \end{array}\right\rangle \end{array}\right)$$

$$\left(\begin{array}{l} 2, \left\langle\begin{array}{lll} \text{index}-1 & \text{cluster}-1 & \text{succ}-1, \\ \text{index}-2 & \text{cluster}-1 & \text{succ}-4, \\ \text{index}-3 & \text{cluster}-2 & \text{succ}-3, \\ \text{index}-4 & \text{cluster}-2 & \text{succ}-2, \\ \text{index}-5 & \text{cluster}-3 & \text{succ}-5, \\ \text{index}-6 & \text{cluster}-3 & \text{succ}-9, \\ \text{index}-7 & \text{cluster}-3 & \text{succ}-7, \\ \text{index}-8 & \text{cluster}-4 & \text{succ}-8, \\ \text{index}-9 & \text{cluster}-4 & \text{succ}-6 \end{array}\right\rangle \end{array}\right)$$





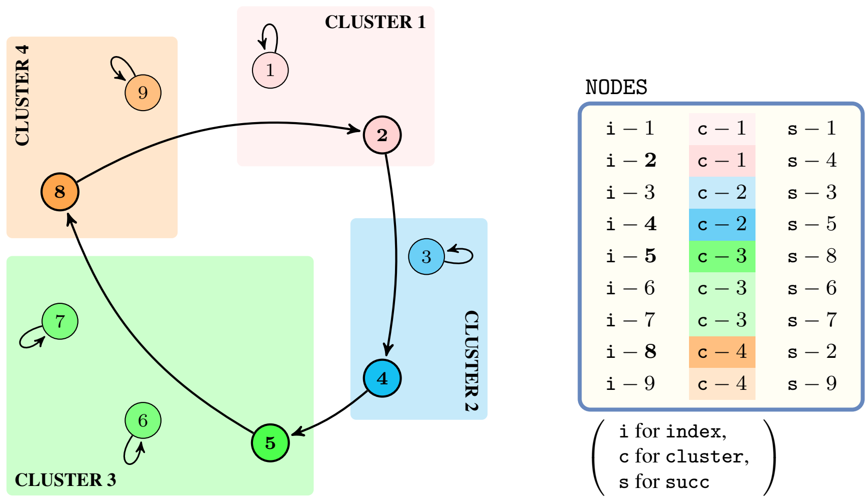


Figure 5.187: Four clusters and a covering with one circuit corresponding to the first example of the **Example** slot

Both examples involve 9 vertices $1, 2, \ldots, 9$ such that vertices 1 and 2 belong to cluster number 1, vertices 3 and 4 belong to cluster number 2, vertices 5, 6 and 7 belong to cluster number 3, and vertices 8 and 9 belong to cluster number 4.

The first example involves only a single circuit containing more than one vertex (i.e., see in Figure 5.187 the circuit $2 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 2$). The corresponding CIRCUIT_CLUSTER constraint holds since exactly one vertex of each cluster (i.e., vertex 2 for cluster 1, vertex 4 for cluster 2, vertex 5 for cluster 3, vertex 8 for cluster 4) belongs to this circuit.

The second example contains the two circuits $2 \rightarrow 4 \rightarrow 2$ and $6 \rightarrow 9 \rightarrow 6$ that both involve more than one vertex. The corresponding CIRCUIT_CLUSTER constraint holds since exactly one vertex of each cluster (i.e., see in Figure 5.188 vertex 2 in $2 \rightarrow 4 \rightarrow 2$ for cluster 1, vertex 4 in $2 \rightarrow 4 \rightarrow 2$ for cluster 2, vertex 6 in $6 \rightarrow 9 \rightarrow 6$ for cluster 3, vertex 9 in $6 \rightarrow 9 \rightarrow 6$ for cluster 4) belongs to these two circuits.

Typical

$\mathtt{NCIRCUIT} < |\mathtt{NODES}|$
$|\mathtt{NODES}| > 2$
$\mathrm{range}(\mathtt{NODES.cluster}) > 1$

Symmetry

Items of NODES are permutable.

Usage

A related abstraction in Operations Research was introduced in [263]. It was reported as the Generalised Travelling Salesman Problem (GTSP). The CIRCUIT_CLUSTER constraint differs from the GTSP because of the two following points:

- Each node of our graph belongs to a single cluster,
- We do not constrain the number of circuits to be equal to 1: The number of circuits should be equal to one of the values of the domain of the variable NCIRCUIT.

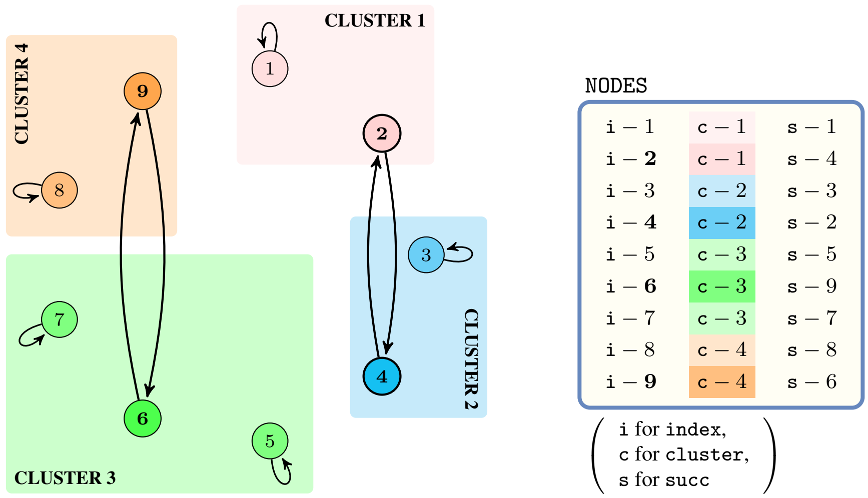


Figure 5.188: The same clusters as in the first example of the **Example** slot and a covering with two circuits corresponding to the second example of the **Example** slot

See also **common keyword:** ALLDIFFERENT *(permutation)*, CIRCUIT, CYCLE *(graph constraint, one_succ)*.

used in graph description: ALLDIFFERENT, NVALUES.

Keywords **combinatorial object:** permutation.

constraint type: graph constraint.

final graph structure: strongly connected component, one_succ.

modelling: cluster.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	• nodes1.succ $\neq$ nodes1.index • nodes1.succ = nodes2.index
Graph property(ies)	• **NTREE** = 0 • **NSCC** = NCIRCUIT
Graph class	ONE_SUCC
Sets	ALL_VERTICES $\mapsto$ $\left[\ \texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{NODES.cluster})]\end{array}\right)\ \right]$
Constraint(s) on sets	• ALLDIFFERENT(variables) • NVALUES(variables, =, size(NODES, cluster))

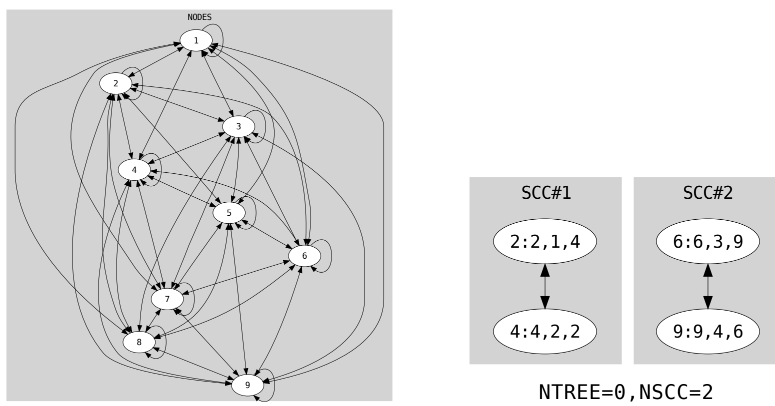
Graph model In order to express the binary constraint linking two vertices one has to make explicit the identifier of each vertex as well as the cluster to which belongs each vertex. This is why the CIRCUIT_CLUSTER constraint considers objects that have the following three attributes:

- The attribute `index` that is the identifier of a vertex.
- The attribute `cluster` that is the cluster to which belongs a vertex.
- The attribute `succ` that is the unique successor of a vertex.

The partitioning of the clusters by different circuits is expressed in the following way:

- First note the condition nodes1.succ $\neq$ nodes1.index prevents the final graph of containing any loop. Moreover the condition nodes1.succ = nodes2.index imposes no more than one successor for each vertex of the final graph.
- The graph property **NTREE** = 0 enforces that all vertices of the final graph belong to one circuit.
- The graph property **NSCC** = NCIRCUIT express the fact that the number of strongly connected components of the final graph is equal to NCIRCUIT.
- The constraint ALLDIFFERENT(variables) on the set ALL_VERTICES (i.e., all the vertices of the final graph) states that the cluster attributes of the vertices of the final graph should be pairwise distinct. This concretely means that no cluster should be visited more than once.
- The constraint NVALUES(variables, =, size(NODES, cluster)) on the set ALL_VERTICES conveys the fact that the number of distinct values of the cluster attribute of the vertices of the final graph should be equal to the total number of clusters. This implies that each cluster is visited at least one time.

Parts (A) and (B) of Figure 5.189 respectively show the initial and final graph associated with the second example of the **Example** slot. Since we use the **NSCC** graph property, we show the two strongly connected components of the final graph. They respectively correspond to the two circuits $2 \rightarrow 4 \rightarrow 2$ and $6 \rightarrow 9 \rightarrow 6$. Since all the vertices belongs to a circuit we have that **NTREE** = 0.



(A) **(B)**

Figure 5.189: Initial and final graph of the CIRCUIT_CLUSTER constraint

5.70 CIRCULAR_CHANGE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin

Derived from CHANGE.

Constraint

CIRCULAR_CHANGE(NCHANGE, VARIABLES, CTR)

Arguments

NCHANGE : dvar
VARIABLES : collection(var−dvar)
CTR : atom

Restrictions

NCHANGE ≥ 0
NCHANGE $\leq |$VARIABLES$|$
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

NCHANGE is the number of times that CTR holds on consecutive variables of the collection VARIABLES. The last and the first variables of the collection VARIABLES are also considered to be consecutive.

Example

$(4, \langle 4, 4, 3, 4, 1\rangle, \neq)$

In the example the changes within the VARIABLES $= \langle 4, 4, 3, 4, 1\rangle$ collection are located between values 4 and 3, 3 and 4, 4 and 1, and 1 and 4 (i.e., since the third argument CTR of the CIRCULAR_CHANGE constraint is set to $\neq$, we count one change for each disequality constraint between two consecutive variables that holds). Consequently, the corresponding CIRCULAR_CHANGE constraint holds since its first argument NCHANGE is fixed to 4.

All solutions

Figure 5.190 gives all solutions to the following non ground instance of the CIRCULAR_CHANGE constraint: NCHANGE $\in [0, 2]$, $V_1 \in [2, 3]$, $V_2 \in [1, 3]$, $V_3 \in [3, 4]$, $V_4 \in [2, 3]$, $V_5 \in [3, 4]$, CIRCULAR_CHANGE(NCHANGE, $\langle V_1, V_2, V_3, V_4, V_5\rangle, \neq$).

Typical

NCHANGE > 0
$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1
CTR $\in [\neq]$

Symmetries

- Items of VARIABLES can be shifted.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Functional dependency: NCHANGE determined by VARIABLES and CTR.

See also

common keyword: CHANGE (*number of changes*).

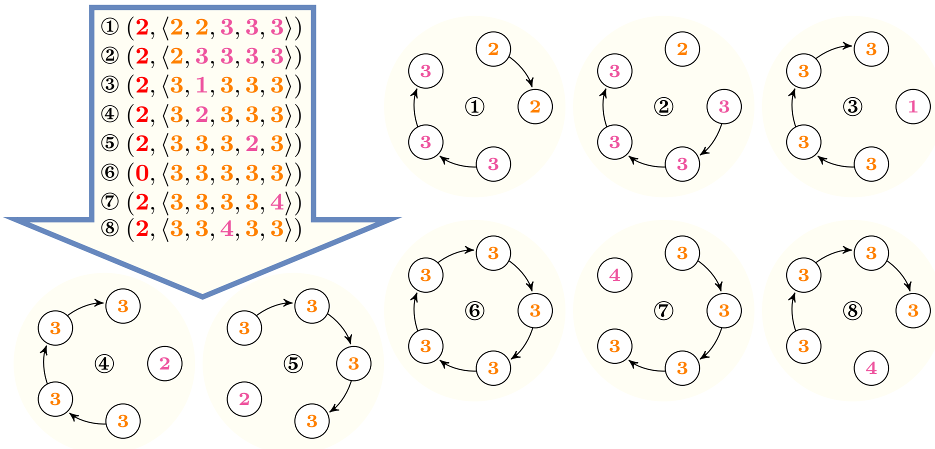


Figure 5.190: All solutions corresponding to the non ground example of the CIRCULAR_CHANGE constraint of the **All solutions** slot; a missing arc between two consecutive nodes represents a *change*, i.e. a disequality constraint that is satisfied between two consecutive variables (the last and the first variables of a sequence are also consecutive).

Keywords

characteristic of a constraint: cyclic, automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: circular sliding cyclic(1) constraint network(2).

constraint type: timetabling constraint.

modelling: number of changes, functional dependency.

Arc input(s)	VARIABLES
Arc generator	*CIRCUIT* ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	**NARC** = NCHANGE

Graph model Since we are also interested in the constraint that links the last and the first variable we use the arc generator *CIRCUIT* to produce the arcs of the initial graph.

Parts (A) and (B) of Figure 5.191 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

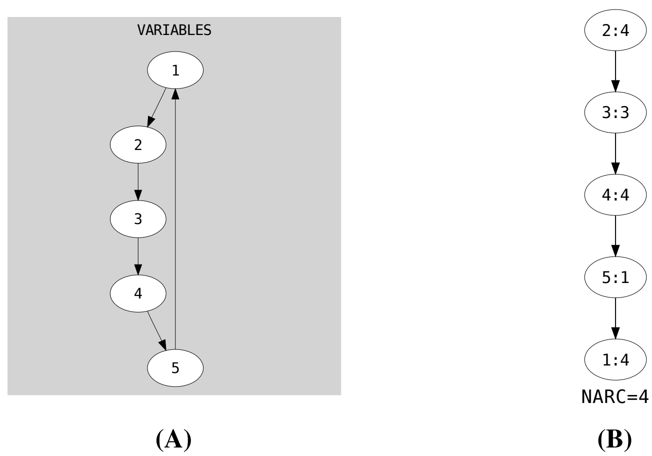


Figure 5.191: Initial and final graph of the CIRCULAR_CHANGE constraint

Automaton Figure 5.192 depicts the automaton associated with the CIRCULAR_CHANGE constraint. To each pair of consecutive variables $(\text{VAR}_i, \text{VAR}_{(i \bmod |\text{VARIABLES}|)+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links VAR_i, $\text{VAR}_{(i \bmod |\text{VARIABLES}|)+1}$ and S_i: $\text{VAR}_i \text{ CTR } \text{VAR}_{(i \bmod |\text{VARIABLES}|)+1} \Leftrightarrow S_i$.

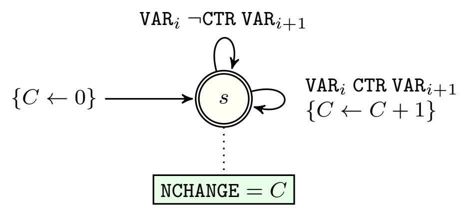


Figure 5.192: Automaton of the CIRCULAR_CHANGE constraint

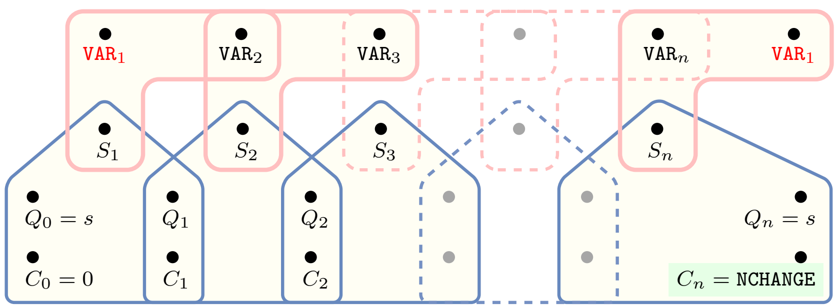


Figure 5.193: Hypergraph of the reformulation corresponding to the automaton of the CIRCULAR_CHANGE constraint

5.71 CLAUSE_AND

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint CLAUSE_AND(POSVARS, NEGVARS, VAR)

Synonym CLAUSE.

Arguments

POSVARS	:	collection(var−dvar)
NEGVARS	:	collection(var−dvar)
VAR	:	dvar

Restrictions

$|\text{POSVARS}| + |\text{NEGVARS}| > 0$
required(POSVARS, var)
POSVARS.var ≥ 0
POSVARS.var ≤ 1
required(NEGVARS, var)
NEGVARS.var ≥ 0
NEGVARS.var ≤ 1
VAR ≥ 0
VAR ≤ 1

Purpose Given a first collection of 0-1 variables POSVARS $= U_1, U_2, \ldots, U_p$, a second collection of 0-1 variables NEGVARS $= V_1, V_2, \ldots, V_n$, and a variable VAR, enforce VAR $= (U_1 \wedge U_2 \wedge \cdots \wedge U_p) \wedge (\neg V_1 \wedge \neg V_2 \wedge \cdots \wedge \neg V_n)$.

Example $(\langle 1, 0\rangle, \langle 0\rangle, 0)$

Typical $|\text{POSVARS}| + |\text{NEGVARS}| > 1$

Symmetries

- Items of POSVARS are permutable.
- Items of NEGVARS are permutable.

Arg. properties

- Extensible wrt. POSVARS when VAR $= 0$.
- Extensible wrt. NEGVARS when VAR $= 0$.

Remark The CLAUSE_OR constraint is called CLAUSE in **Gecode** (http://www.gecode.org/).

Systems REIFIEDAND in **Choco**, CLAUSE in **Choco**, CLAUSE in **Gecode**.

See also **common keyword:** AND, CLAUSE_OR (*Boolean constraint*).

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

Automaton Figure 5.194 depicts the automaton associated with the CLAUSE_AND constraint:

- To the argument VAR of the CLAUSE_AND constraint corresponds the first signature variable.
- To each variable of the argument POSVARS corresponds a signature variable.
- Finally, to each variable $\mathtt{VAR}_i$ of the argument NEGVARS corresponds a signature variable that is the negation of $\mathtt{VAR}_i$.

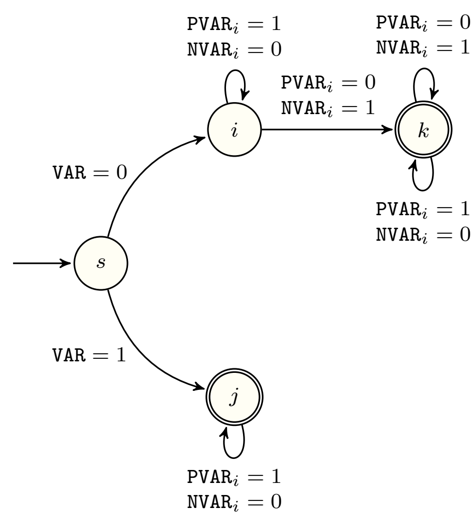


Figure 5.194: Automaton of the CLAUSE_AND constraint ($\mathtt{PVAR}_i$ and $\mathtt{NVAR}_i$ respectively denote variables of POSVARS and NEGVARS)

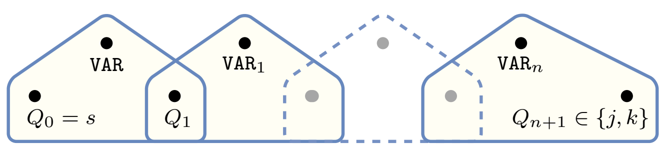


Figure 5.195: Hypergraph of the reformulation corresponding to the automaton of the CLAUSE_AND constraint ($\mathtt{VAR}_1, \ldots, \mathtt{VAR}_n$ denotes $\mathtt{PVAR}_1, \ldots, \mathtt{PVAR}_{|\mathtt{POSVARS}|}, 1 - \mathtt{NVAR}_1, \ldots, 1 - \mathtt{NVAR}_{|\mathtt{NEGVARS}|}$)

5.72 CLAUSE_OR

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint CLAUSE_OR(POSVARS, NEGVARS, VAR)

Synonym CLAUSE.

Arguments

POSVARS	:	collection(var−dvar)
NEGVARS	:	collection(var−dvar)
VAR	:	dvar

Restrictions

$|\mathtt{POSVARS}| + |\mathtt{NEGVARS}| > 0$
required(POSVARS, var)
POSVARS.var ≥ 0
POSVARS.var ≤ 1
required(NEGVARS, var)
NEGVARS.var ≥ 0
NEGVARS.var ≤ 1
VAR ≥ 0
VAR ≤ 1

Purpose Given a first collection of 0-1 variables POSVARS $= U_1, U_2, \ldots, U_p$, a second collection of 0-1 variables NEGVARS $= V_1, V_2, \ldots, V_n$, and a variable VAR, enforce VAR $= (U_1 \vee U_2 \vee \cdots \vee U_p) \vee (\neg V_1 \vee \neg V_2 \vee \cdots \vee \neg V_n)$.

Example $(\langle 0, 0\rangle, \langle 0\rangle, 1)$

Typical $|\mathtt{POSVARS}| + |\mathtt{NEGVARS}| > 1$

Symmetries

- Items of POSVARS are permutable.
- Items of NEGVARS are permutable.

Arg. properties

- Extensible wrt. POSVARS when VAR $= 1$.
- Extensible wrt. NEGVARS when VAR $= 1$.

Remark The CLAUSE_OR constraint is called CLAUSE in **Gecode** (http://www.gecode.org/).

Systems REIFIEDOR in **Choco**, CLAUSE in **Choco**, CLAUSE in **Gecode**.

See also **common keyword:** CLAUSE_AND, OR (*Boolean constraint*).

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: disjunction.

Automaton Figure 5.196 depicts the automaton associated with the CLAUSE_OR constraint:

- To the argument VAR of the CLAUSE_OR constraint corresponds the first signature variable.
- To each variable of the argument POSVARS corresponds a signature variable.
- Finally, to each variable $\mathtt{VAR}_i$ of the argument NEGVARS corresponds a signature variable that is the negation of $\mathtt{VAR}_i$.

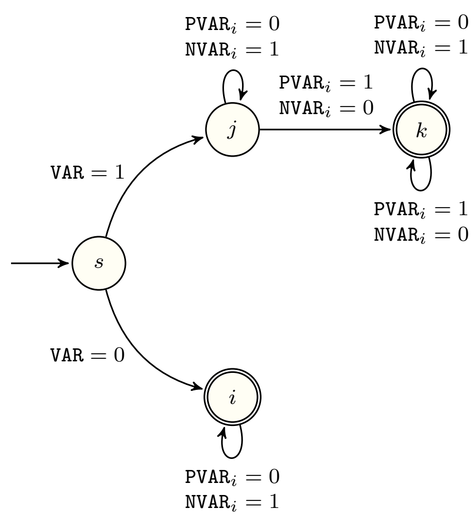


Figure 5.196: Automaton of the CLAUSE_OR constraint ($\mathtt{PVAR}_i$ and $\mathtt{NVAR}_i$ respectively denote variables of POSVARS and NEGVARS)

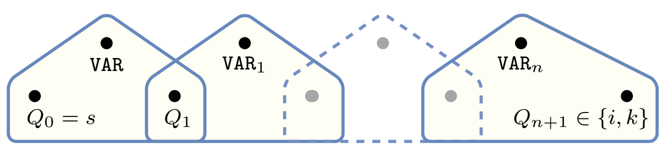


Figure 5.197: Hypergraph of the reformulation corresponding to the automaton of the CLAUSE_OR constraint ($\mathtt{VAR}_1, \ldots, \mathtt{VAR}_n$ denotes $\mathtt{PVAR}_1, \ldots, \mathtt{PVAR}_{|\mathtt{POSVARS}|}, 1 - \mathtt{NVAR}_1, \ldots, 1 - \mathtt{NVAR}_{|\mathtt{NEGVARS}|}$)

5.73 CLIQUE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [170]

Constraint CLIQUE(SIZE_CLIQUE, NODES)

Arguments

SIZE_CLIQUE : dvar
NODES : collection(index−int, succ−svar)

Restrictions

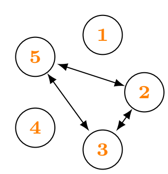
SIZE_CLIQUE ≥ 0
SIZE_CLIQUE $\leq$ |NODES|
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

Purpose

Consider a digraph G described by the NODES collection: to the i^{th} item of the NODES collection corresponds the i^{th} vertex of G; To each value j of the i^{th} succ variable corresponds an arc from the i^{th} vertex to the j^{th} vertex. Select a subset $\mathcal{S}$ of the vertices of G that forms a clique of size SIZE_CLIQUE (i.e., there is an arc between each pair of distinct vertices of $\mathcal{S}$).

Example

$$\left(3, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\emptyset, \\ \texttt{index}-2 & \texttt{succ}-\{3,5\}, \\ \texttt{index}-3 & \texttt{succ}-\{2,5\}, \\ \texttt{index}-4 & \texttt{succ}-\emptyset, \\ \texttt{index}-5 & \texttt{succ}-\{2,3\} \end{array} \right\rangle \right)$$



The CLIQUE constraint holds since the NODES collection depicts a clique involving 3 vertices (namely vertices 2, 3 and 5) and since its first argument SIZE_CLIQUE is set to the number of vertices of this clique.

Typical

SIZE_CLIQUE ≥ 2
SIZE_CLIQUE $<$ |NODES|
|NODES| > 2

Symmetry Items of NODES are permutable.

Arg. properties Functional dependency: SIZE_CLIQUE determined by NODES.

Algorithm [170], [358, 359]. The algorithm for finding maximum cliques in an undirected graph of C. Bron and J. Kerbosch [97] was adapted by J.-C. Régin to the context of constraint programming in his papers.

See also

common keyword: LINK_SET_TO_BOOLEANS *(constraint involving set variables, can be used for channelling)*.

used in graph description: IN_SET.

Keywords

constraint arguments: constraint involving set variables.

constraint type: graph constraint.

final graph structure: symmetric.

modelling: functional dependency.

problems: maximum clique.

Arc input(s)	NODES
Arc generator	$CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.succ)
Graph property(ies)	• **NARC** = SIZE_CLIQUE * SIZE_CLIQUE − SIZE_CLIQUE • **NVERTEX** = SIZE_CLIQUE
Graph class	SYMMETRIC

Graph model

Note the use of *set variables* for modelling the fact that the vertices of the final graph have more than one successor: The successor variable associated with each vertex contains the successors of the corresponding vertex.

Part (A) of Figure 5.198 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.198 gives the final graph associated with the **Example** slot. Since we both use the **NARC** and **NVERTEX** graph properties, the arcs and the vertices of the final graph are stressed in bold. The final graph corresponds to a clique containing three vertices.

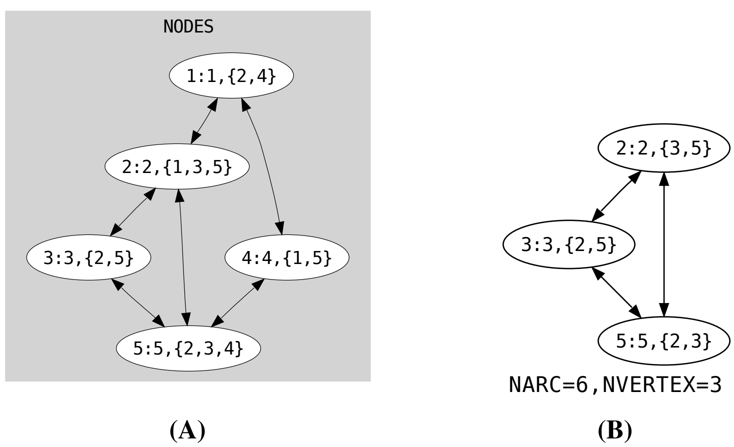


Figure 5.198: Initial and final graph of the CLIQUE set constraint

5.74 COLORED_MATRIX

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin KOALOG

Constraint COLORED_MATRIX(C, L, K, MATRIX, CPROJ, LPROJ)

Synonyms COLOURED_MATRIX, CARDINALITY_MATRIX, CARD_MATRIX.

Arguments

C	:	int
L	:	int
K	:	int
MATRIX	:	collection(column−int, line−int, var−dvar)
CPROJ	:	collection(column−int, val−int, nocc−dvar)
LPROJ	:	collection(line−int, val−int, nocc−dvar)

Restrictions

C ≥ 0
L ≥ 0
K ≥ 0
required(MATRIX, [column, line, var])
increasing_seq(MATRIX, [column, line])
|MATRIX| = C ∗ L + C + L + 1
MATRIX.column ≥ 0
MATRIX.column ≤ C
MATRIX.line ≥ 0
MATRIX.line ≤ L
MATRIX.var ≥ 0
MATRIX.var ≤ K
required(CPROJ, [column, val, nocc])
increasing_seq(CPROJ, [column, val])
|CPROJ| = C ∗ K + C + K + 1
CPROJ.column ≥ 0
CPROJ.column ≤ C
CPROJ.val ≥ 0
CPROJ.val ≤ K
required(LPROJ, [line, val, nocc])
increasing_seq(LPROJ, [line, val])
|LPROJ| = L ∗ K + L + K + 1
LPROJ.line ≥ 0
LPROJ.line ≤ L
LPROJ.val ≥ 0
LPROJ.val ≤ K

Purpose Given a matrix of domain variables, imposes a GLOBAL_CARDINALITY constraint involving cardinality variables on each column and each row of the matrix.

Example

$$\left(\begin{array}{l}
1,2,4,\left\langle\begin{array}{lll}
\texttt{column}-0 & \texttt{line}-0 & \texttt{var}-3,\\
\texttt{column}-0 & \texttt{line}-1 & \texttt{var}-1,\\
\texttt{column}-0 & \texttt{line}-2 & \texttt{var}-3,\\
\texttt{column}-1 & \texttt{line}-0 & \texttt{var}-4,\\
\texttt{column}-1 & \texttt{line}-1 & \texttt{var}-4,\\
\texttt{column}-1 & \texttt{line}-2 & \texttt{var}-3
\end{array}\right\rangle,\\
\left\langle\begin{array}{lll}
\texttt{column}-0 & \texttt{val}-0 & \texttt{nocc}-0,\\
\texttt{column}-0 & \texttt{val}-1 & \texttt{nocc}-1,\\
\texttt{column}-0 & \texttt{val}-2 & \texttt{nocc}-0,\\
\texttt{column}-0 & \texttt{val}-3 & \texttt{nocc}-2,\\
\texttt{column}-0 & \texttt{val}-4 & \texttt{nocc}-0,\\
\texttt{column}-1 & \texttt{val}-0 & \texttt{nocc}-0,\\
\texttt{column}-1 & \texttt{val}-1 & \texttt{nocc}-0,\\
\texttt{column}-1 & \texttt{val}-2 & \texttt{nocc}-0,\\
\texttt{column}-1 & \texttt{val}-3 & \texttt{nocc}-1,\\
\texttt{column}-1 & \texttt{val}-4 & \texttt{nocc}-2
\end{array}\right\rangle,\\
\left\langle\begin{array}{lll}
\texttt{line}-0 & \texttt{val}-0 & \texttt{nocc}-0,\\
\texttt{line}-0 & \texttt{val}-1 & \texttt{nocc}-0,\\
\texttt{line}-0 & \texttt{val}-2 & \texttt{nocc}-0,\\
\texttt{line}-0 & \texttt{val}-3 & \texttt{nocc}-1,\\
\texttt{line}-0 & \texttt{val}-4 & \texttt{nocc}-1,\\
\texttt{line}-1 & \texttt{val}-0 & \texttt{nocc}-0,\\
\texttt{line}-1 & \texttt{val}-1 & \texttt{nocc}-1,\\
\texttt{line}-1 & \texttt{val}-2 & \texttt{nocc}-0,\\
\texttt{line}-1 & \texttt{val}-3 & \texttt{nocc}-0,\\
\texttt{line}-1 & \texttt{val}-4 & \texttt{nocc}-1,\\
\texttt{line}-2 & \texttt{val}-0 & \texttt{nocc}-0,\\
\texttt{line}-2 & \texttt{val}-1 & \texttt{nocc}-0,\\
\texttt{line}-2 & \texttt{val}-2 & \texttt{nocc}-0,\\
\texttt{line}-2 & \texttt{val}-3 & \texttt{nocc}-2,\\
\texttt{line}-2 & \texttt{val}-4 & \texttt{nocc}-0
\end{array}\right\rangle
\end{array}\right)$$

Typical

$\texttt{C} \geq 1$
$\texttt{L} \geq 1$
$\texttt{K} \geq 1$
$\texttt{range}(\texttt{MATRIX.var}) > 1$

Arg. properties

- Functional dependency: CPROJ.nocc determined by C, L and K.
- Functional dependency: LPROJ.nocc determined by C, L and K.

Remark

Within [361] the COLORED_MATRIX constraint is called CARDINALITY_MATRIX.

Algorithm

The filtering algorithm described in [361] is based on network flow and does not achieve arc-consistency in general. However, when the number of values is restricted to two, the algorithm [361] achieves arc-consistency on the variables of the matrix. This corresponds in fact to a generalisation of the problem called ”Matrices composed of 0’s and 1’s” presented by Ford and Fulkerson [238].

See also

common keyword: K_ALLDIFFERENT *(system of constraints)*.

part of system of constraints: GLOBAL_CARDINALITY.

related to a common problem: SAME *(matrix reconstruction problem)*.

Keywords

constraint arguments: pure functional dependency.

constraint type: system of constraints, predefined constraint, timetabling constraint.

modelling: functional dependency, matrix, matrix model.

5.75 COLOURED_CUMULATIVE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

Derived from CUMULATIVE and NVALUES.

Constraint

COLOURED_CUMULATIVE(TASKS, LIMIT)

Synonym

COLORED_CUMULATIVE.

Arguments

$$\begin{array}{lll} \texttt{TASKS} & : & \texttt{collection}\left(\begin{array}{l}\texttt{origin}-\texttt{dvar},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar},\\ \texttt{colour}-\texttt{dvar}\end{array}\right) \\ \texttt{LIMIT} & : & \texttt{int} \end{array}$$

Restrictions

$\texttt{require\_at\_least}(2, \texttt{TASKS}, [\texttt{origin}, \texttt{duration}, \texttt{end}])$
$\texttt{required}(\texttt{TASKS}, \texttt{colour})$
$\texttt{TASKS.duration} \geq 0$
$\texttt{TASKS.origin} \leq \texttt{TASKS.end}$
$\texttt{LIMIT} \geq 0$

Purpose

Consider the set $\mathcal{T}$ of tasks described by the TASKS collection. The COLOURED_CUMULATIVE constraint forces that, at each point in time, the number of distinct colours of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i. For each task of $\mathcal{T}$ it also imposes the constraint $\texttt{origin} + \texttt{duration} = \texttt{end}$.

Example

$$\left(\left\langle\begin{array}{llll} \texttt{origin}-1 & \texttt{duration}-2 & \texttt{end}-3 & \texttt{colour}-1,\\ \texttt{origin}-2 & \texttt{duration}-9 & \texttt{end}-11 & \texttt{colour}-2,\\ \texttt{origin}-3 & \texttt{duration}-10 & \texttt{end}-13 & \texttt{colour}-3,\\ \texttt{origin}-6 & \texttt{duration}-6 & \texttt{end}-12 & \texttt{colour}-2,\\ \texttt{origin}-7 & \texttt{duration}-2 & \texttt{end}-9 & \texttt{colour}-3 \end{array}\right\rangle, 2\right)$$

Figure 5.199 shows the solution associated with the example. Each rectangle of the figure corresponds to a task of the COLOURED_CUMULATIVE constraint. Tasks that have their colour attributes set to 1, 2 and 3 are respectively coloured in yellow, blue and pink. The COLOURED_CUMULATIVE constraint holds since at each point in time we do not have more than $\texttt{LIMIT} = 2$ distinct colours.

Typical

$|\texttt{TASKS}| > 1$
$\texttt{range}(\texttt{TASKS.origin}) > 1$
$\texttt{range}(\texttt{TASKS.duration}) > 1$
$\texttt{range}(\texttt{TASKS.end}) > 1$
$\texttt{range}(\texttt{TASKS.colour}) > 1$
$\texttt{LIMIT} < \texttt{nval}(\texttt{TASKS.colour})$

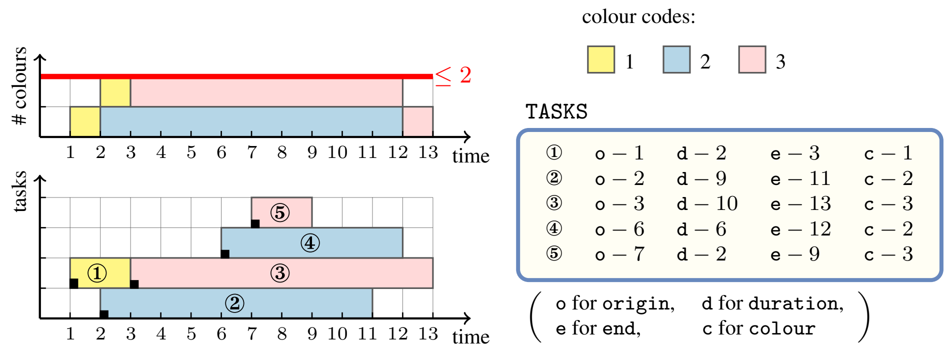


Figure 5.199: The coloured cumulative solution to the **Example** slot with at most two distinct colours in parallel

Symmetries

- Items of TASKS are permutable.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.
- All occurrences of two distinct values of TASKS.colour can be swapped; all occurrences of a value of TASKS.colour can be renamed to any unused value.
- LIMIT can be increased.

Arg. properties

Contractible wrt. TASKS.

Usage

Useful for scheduling problems where a machine can only proceed in parallel a maximum number of tasks of distinct type. This condition cannot be modelled by the classical CUMULATIVE constraint. Also useful for coloured bin packing problems (i.e., duration $= 1$) where each item has a colour and no bin contains items with more than LIMIT distinct colours [141, 197, 217].

Reformulation

The COLOURED_CUMULATIVE constraint can be expressed in term of a set of reified constraints and of $|$TASKS$|$ NVALUE constraints:

1. For each pair of tasks TASKS$[i]$, TASKS$[j]$ $(i, j \in [1, |\text{TASKS}|])$ of the TASKS collection we create a variable C_{ij} which is set to the colour of task TASKS$[j]$ if task TASKS$[j]$ overlaps the origin attribute of task TASKS$[i]$, and to the colour of task TASKS$[i]$ otherwise:
   - If $i = j$:
     - $C_{ij} = \text{TASKS}[i].\text{colour}$.
   - If $i \neq j$:
     - $C_{ij} = \text{TASKS}[i].\text{colour} \vee C_{ij} = \text{TASKS}[j].\text{colour}$.
     - $((\text{TASKS}[j].\text{origin} \leq \text{TASKS}[i].\text{origin} \wedge$
       $\text{TASKS}[j].\text{end} > \text{TASKS}[i].\text{origin}) \wedge (C_{ij} = \text{TASKS}[j].\text{colour})) \vee$
       $((\text{TASKS}[j].\text{origin} > \text{TASKS}[i].\text{origin} \vee$
       $\text{TASKS}[j].\text{end} \leq \text{TASKS}[i].\text{origin}) \wedge (C_{ij} = \text{TASKS}[i].\text{colour}))$

2. For each task $\texttt{TASKS}[i]$ $(i \in [1, |\texttt{TASKS}|])$ we create a variable N_i which gives the number of distinct colours associated with the tasks that overlap the origin of task $\texttt{TASKS}[i]$ ($\texttt{TASKS}[i]$ overlaps its own origin) and we impose N_i to not exceed the maximum number of distinct colours LIMIT allowed at each instant:
   - $N_i \geq 1 \wedge N_i \leq \texttt{LIMIT}$.
   - NVALUE$(N_i, \langle C_{i1}, C_{i2}, \ldots, C_{i|\texttt{TASKS}|}\rangle)$.

See also

assignment dimension added: COLOURED_CUMULATIVES.

common keyword: CUMULATIVE, TRACK *(resource constraint)*.

related: NVALUE.

specialisation: DISJOINT_TASKS *(a colour is assigned to each collection of tasks of constraint* DISJOINT_TASKS *and a limit of one single colour is enforced)*.

used in graph description: NVALUES.

Keywords

characteristic of a constraint: coloured.

constraint type: scheduling constraint, resource constraint, temporal constraint.

filtering: compulsory part, minimum task duration.

modelling: number of distinct values, zero-duration task.

Arc input(s)	TASKS
Arc generator	$SELF \mapsto$ collection(tasks)
Arc arity	1
Arc constraint(s)	tasks.origin + tasks.duration = tasks.end
Graph property(ies)	**NARC** $=$ |TASKS|

Arc input(s)	TASKS TASKS
Arc generator	$PRODUCT \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.duration > 0 • tasks2.origin $\leq$ tasks1.origin • tasks1.origin $<$ tasks2.end
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP
Sets	SUCC $\mapsto$ [source, variables − col(VARIABLES−collection(var−dvar), [item(var − TASKS.colour)])]
Constraint(s) on sets	NVALUES(variables, $\leq$, LIMIT)

Graph model Same as CUMULATIVE, except that we use another constraint for computing the resource consumption at each time point.

Parts (A) and (B) of Figure 5.200 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks that overlap that time point. The COLOURED_CUMULATIVE constraint holds since for each successor set $\mathcal{S}$ of the final graph the number of distinct colours of the tasks in $\mathcal{S}$ does not exceed the LIMIT 2.

Signature Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite **NARC** $=$ |TASKS| to **NARC** $\geq$ |TASKS|. This leads to simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

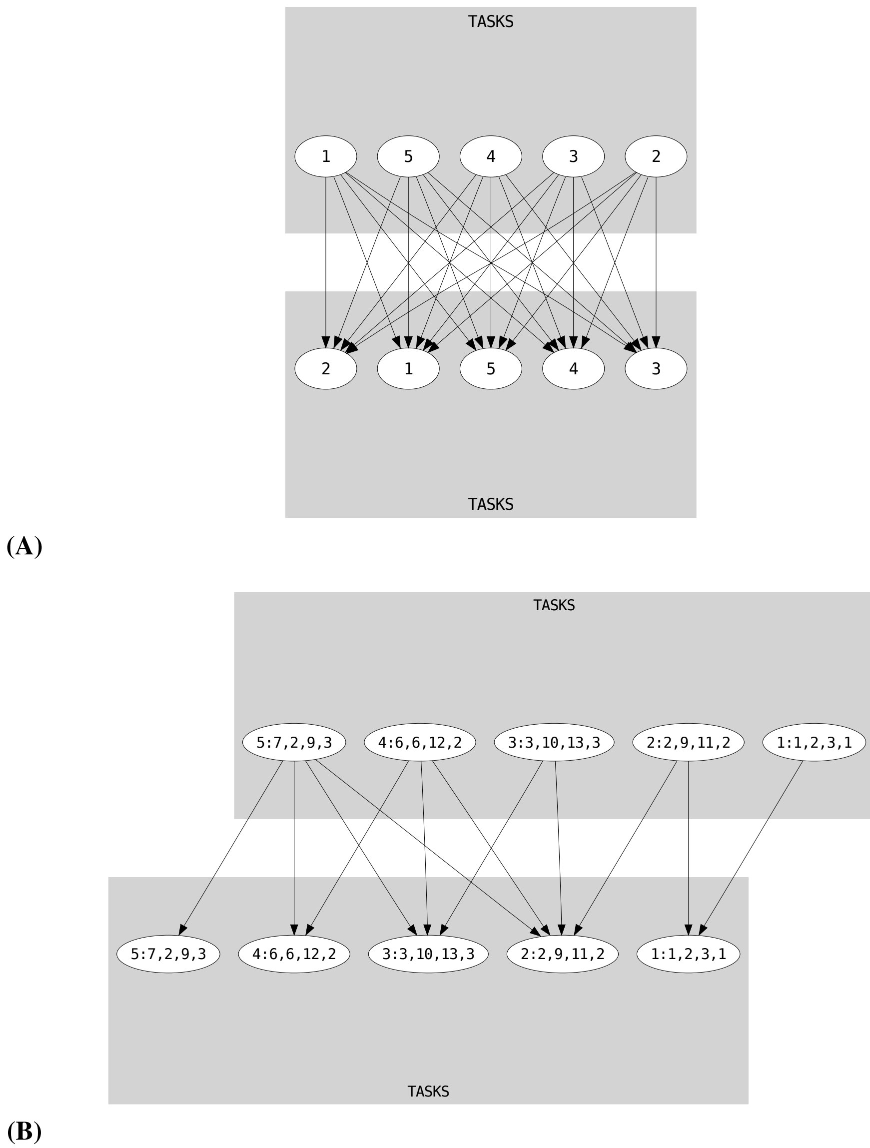


Figure 5.200: Initial and final graph of the COLOURED_CUMULATIVE constraint

5.76 COLOURED_CUMULATIVES

DESCRIPTION LINKS GRAPH

Origin Derived from CUMULATIVES and NVALUES.

Constraint COLOURED_CUMULATIVES(TASKS, MACHINES)

Synonym COLORED_CUMULATIVES.

Arguments

$$\begin{array}{lll} \texttt{TASKS} & : & \texttt{collection}\left(\begin{array}{l}\texttt{machine}-\texttt{dvar},\\ \texttt{origin}-\texttt{dvar},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar},\\ \texttt{colour}-\texttt{dvar}\end{array}\right)\\ \texttt{MACHINES} & : & \texttt{collection}(\texttt{id}-\texttt{int}, \texttt{capacity}-\texttt{int}) \end{array}$$

Restrictions

$\texttt{required}(\texttt{TASKS}, [\texttt{machine}, \texttt{colour}])$
$\texttt{require\_at\_least}(2, \texttt{TASKS}, [\texttt{origin}, \texttt{duration}, \texttt{end}])$
$\texttt{TASKS.duration} \geq 0$
$\texttt{TASKS.origin} \leq \texttt{TASKS.end}$
$\texttt{required}(\texttt{MACHINES}, [\texttt{id}, \texttt{capacity}])$
$\texttt{distinct}(\texttt{MACHINES}, \texttt{id})$
$\texttt{MACHINES.capacity} \geq 0$

Purpose

Consider a set $\mathcal{T}$ of tasks described by the TASKS collection. The COLOURED_CUMULATIVES constraint forces for each machine m of the MACHINES collection the following condition: at each point in time p, the numbers of distinct colours of the set of tasks that both overlap that point p and are assigned to machine m does not exceed the capacity of machine m. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i. It also imposes for each task of $\mathcal{T}$ the constraint $\texttt{origin} + \texttt{duration} = \texttt{end}$.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{lllll}\texttt{m}-1 & \texttt{origin}-6 & \texttt{dur}-6 & \texttt{end}-12 & \texttt{colour}-2,\\ \texttt{m}-1 & \texttt{origin}-2 & \texttt{dur}-9 & \texttt{end}-11 & \texttt{colour}-3,\\ \texttt{m}-2 & \texttt{origin}-7 & \texttt{dur}-3 & \texttt{end}-10 & \texttt{colour}-3,\\ \texttt{m}-1 & \texttt{origin}-1 & \texttt{dur}-2 & \texttt{end}-3 & \texttt{colour}-1,\\ \texttt{m}-2 & \texttt{origin}-4 & \texttt{dur}-5 & \texttt{end}-9 & \texttt{colour}-3,\\ \texttt{m}-1 & \texttt{origin}-3 & \texttt{dur}-10 & \texttt{end}-13 & \texttt{colour}-2\end{array}\right\rangle,\\ \langle\texttt{id}-1\ \texttt{capacity}-2, \texttt{id}-2\ \texttt{capacity}-1\rangle\end{array}\right)$$

(m for machine, dur for duration)

Figure 5.201 shows the solution associated with the example. Each rectangle of the figure corresponds to a task of the COLOURED_CUMULATIVES constraint. Tasks that have their colour attributes set to 1 and 2 are respectively coloured in blue and pink. The COLOURED_CUMULATIVES constraint holds since for machine 1 we have at most two distinct colours in parallel (which is the maximum capacity for machine 1), while for machine 2 we have no more than a single colour in parallel (which is actually the maximum capacity for machine 2).

TASKS

①	m − 1	o − 6	d − 6	e − 12	c − 2
②	m − 1	o − 2	d − 9	e − 11	c − 3
③	m − 2	o − 7	d − 3	e − 10	c − 3
④	m − 1	o − 1	d − 2	e − 3	c − 1
⑤	m − 2	o − 4	d − 5	e − 9	c − 3
⑥	m − 1	o − 3	d − 10	e − 13	c − 2

(m for machine, o for origin, d for duration, e for end, c for colour)

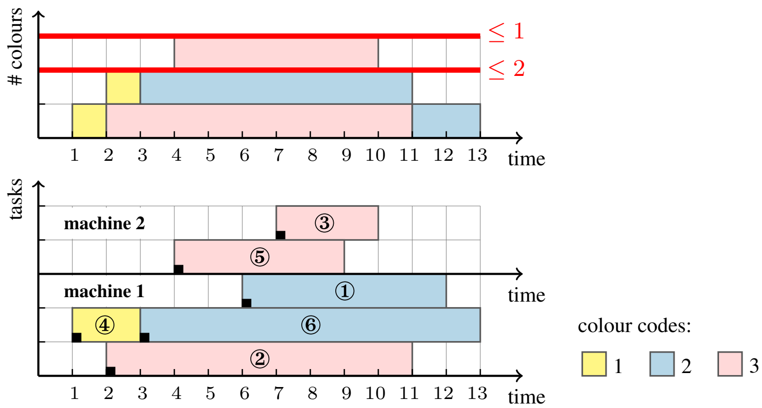


Figure 5.201: The coloured cumulative solution to the **Example** slot with at most two distinct colours in parallel on machine 1 and at most one distinct colour in parallel on machine 2

Typical

$|$TASKS$| > 1$
range(TASKS.machine) > 1
range(TASKS.origin) > 1
range(TASKS.duration) > 1
range(TASKS.end) > 1
range(TASKS.colour) > 1
TASKS.duration > 0
$|$MACHINES$| > 1$
MACHINES.capacity > 0
MACHINES.capacity $<$nval(TASKS.colour)
$|$TASKS$| > |$MACHINES$|$

Symmetries

- Items of TASKS are permutable.
- Items of MACHINES are permutable.
- MACHINES.capacity can be increased.
- All occurrences of two distinct values in TASKS.machine or MACHINES.id can be swapped; all occurrences of a value in TASKS.machine or MACHINES.id can be renamed to any unused value.

Arg. properties

Contractible wrt. TASKS.

Usage

Useful for scheduling problems where several machines are available and where you have to assign each task to a specific machine. In addition each machine can only proceed in parallel a maximum number of tasks of distinct types.

Reformulation

The COLOURED_CUMULATIVES constraint can be expressed in term of a set of reified constraints and of $|\texttt{TASKS}|$ NVALUE constraints:

1. For each pair of tasks $\texttt{TASKS}[i], \texttt{TASKS}[j]$ $(i, j \in [1, |\texttt{TASKS}|])$ of the TASKS collection we create a variable C_{ij} which is set to the colour of task $\texttt{TASKS}[j]$ if both tasks are assigned to the same machine and if task $\texttt{TASKS}[j]$ overlaps the origin attribute of task $\texttt{TASKS}[i]$, and to the colour of task $\texttt{TASKS}[i]$ otherwise:
   - If $i = j$:
     - $C_{ij} = \texttt{TASKS}[i].\texttt{colour}$.
   - If $i \neq j$:
     - $C_{ij} = \texttt{TASKS}[i].\texttt{colour} \vee C_{ij} = \texttt{TASKS}[j].\texttt{colour}$.
     - $((\texttt{TASKS}[j].\texttt{machine} = \texttt{TASKS}[i].\texttt{machine} \wedge$
       $\texttt{TASKS}[j].\texttt{origin} \leq \texttt{TASKS}[i].\texttt{origin} \wedge$
       $\texttt{TASKS}[j].\texttt{end} > \texttt{TASKS}[i].\texttt{origin}) \wedge (C_{ij} = \texttt{TASKS}[j].\texttt{colour})) \vee$
       $((\texttt{TASKS}[j].\texttt{machine} \neq \texttt{TASKS}[i].\texttt{machine} \vee$
       $\texttt{TASKS}[j].\texttt{origin} > \texttt{TASKS}[i].\texttt{origin} \vee$
       $\texttt{TASKS}[j].\texttt{end} \leq \texttt{TASKS}[i].\texttt{origin}) \wedge (C_{ij} = \texttt{TASKS}[i].\texttt{colour}))$
2. For each task $\texttt{TASKS}[i]$ $(i \in [1, |\texttt{TASKS}|])$ we create a variable N_i which gives the number of distinct colours associated with the tasks that both are assigned to the same machine as task $\texttt{TASKS}[i]$ and overlap the origin of task $\texttt{TASKS}[i]$ ($\texttt{TASKS}[i]$ overlaps its own origin) and we impose N_i to not exceed the maximum number of distinct colours LIMIT allowed at each instant:
   - $N_i \geq 1 \wedge N_i \leq \texttt{LIMIT}$.
   - NVALUE$(N_i, \langle C_{i1}, C_{i2}, \dots, C_{i|\texttt{TASKS}|}\rangle)$.

See also

assignment dimension removed: COLOURED_CUMULATIVE *(*machine *attribute removed)*, CUMULATIVE *(*machine *attribute removed and number of distinct* colours *replaced by sum of* task heights*)*.

common keyword: CUMULATIVE, CUMULATIVES *(resource constraint)*.

related: NVALUE.

used in graph description: NVALUES.

Keywords

characteristic of a constraint: coloured.

constraint type: scheduling constraint, resource constraint, temporal constraint.

filtering: compulsory part, minimum task duration.

modelling: number of distinct values, assignment dimension, zero-duration task.

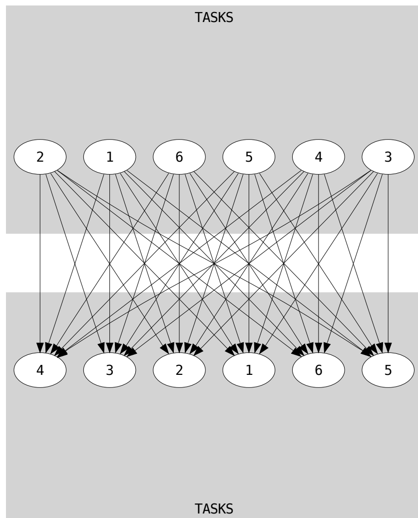
Arc input(s)	TASKS
Arc generator	$SELF \mapsto$ collection(tasks)
Arc arity	1
Arc constraint(s)	tasks.origin + tasks.duration = tasks.end
Graph property(ies)	**NARC**= |TASKS|

For all items of MACHINES:

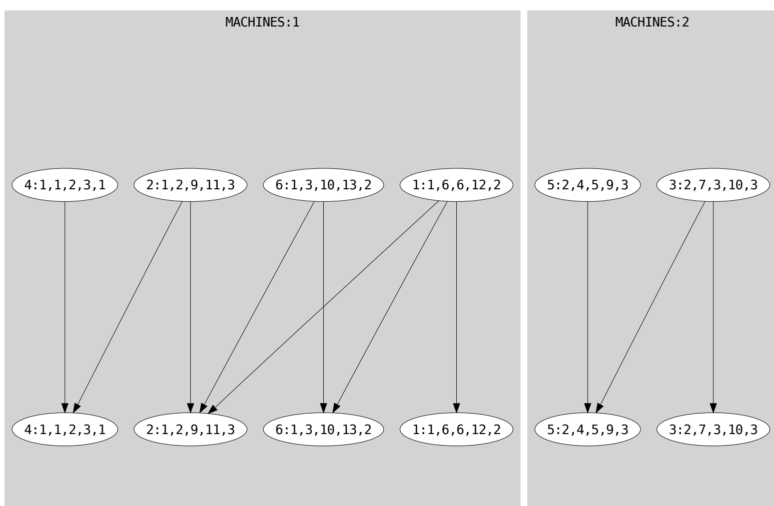
Arc input(s)	TASKS TASKS
Arc generator	$PRODUCT \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.machine = MACHINES.id • tasks1.machine = tasks2.machine • tasks1.duration > 0 • tasks2.origin ≤ tasks1.origin • tasks1.origin < tasks2.end
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP
Sets	$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\mathtt{source},\\ \mathtt{variables} - \mathtt{col}\left(\begin{array}{l}\mathtt{VARIABLES}-\mathtt{collection}(\mathtt{var}-\mathtt{dvar}),\\ [\mathtt{item}(\mathtt{var}-\mathtt{TASKS.colour})]\end{array}\right)\end{array}\right]$
Constraint(s) on sets	NVALUES(variables, ≤, MACHINES.capacity)

Graph model Parts (A) and (B) of Figure 5.202 respectively shows the initial and final graph associated with machines 1 and 2 involved in the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point p on a specific machine m. On the other hand the successors of a source vertex correspond to those tasks that both overlap that time point p and are assigned to machine m. The COLOURED_CUMULATIVES constraint holds since for each successor set $\mathcal{S}$ of the final graph the number of distinct colours in $\mathcal{S}$ does not exceed the capacity of the machine corresponding to the time point associated with $\mathcal{S}$.

Signature Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite **NARC** = |TASKS| to **NARC** ≥ |TASKS|. This leads to simplify $\underline{\overline{\textbf{NARC}}}$ to $\overline{\textbf{NARC}}$.



(A)



(B)

Figure 5.202: Initial and final graph of the COLOURED_CUMULATIVES constraint

5.77 COMMON

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint COMMON(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2)

Arguments

```
NCOMMON1   : dvar
NCOMMON2   : dvar
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
```

Restrictions

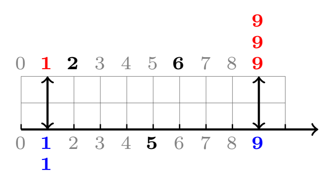
NCOMMON1 ≥ 0
NCOMMON1 $\leq$ |VARIABLES1|
NCOMMON2 ≥ 0
NCOMMON2 $\leq$ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose

NCOMMON1 is the number of variables of the collection of variables VARIABLES1 taking a value in VARIABLES2.
NCOMMON2 is the number of variables of the collection of variables VARIABLES2 taking a value in VARIABLES1.

Example

$(3, 4, \langle 1, 9, 1, 5\rangle, \langle 2, 1, 9, 9, 6, 9\rangle)$



The COMMON constraint holds since:

- Its first argument NCOMMON1 $= 3$ corresponds to the number of values of the collection $\langle 1, 9, 1, 5\rangle$ that occur within $\langle 2, 1, 9, 9, 6, 9\rangle$.
- Its second argument NCOMMON2 $= 4$ corresponds to the number of values of the collection $\langle 2, 1, 9, 9, 6, 9\rangle$ that occur within $\langle 1, 9, 1, 5\rangle$.

All solutions Figure 5.203 gives all solutions to the following non ground instance of the COMMON constraint: NCOMMON1 $\in [0, 1]$, NCOMMON2 $\in [2, 3]$, $U_1 \in [1, 2]$, $U_2 \in [1, 2]$, $U_3 \in [0, 1]$, $U_4 \in [5, 6]$, $V_1 \in [5, 6]$, $V_2 \in [1, 2]$, $V_3 \in [0, 1]$, COMMON(NCOMMON1, NCOMMON2, $\langle U_1, U_2, U_3, U_4\rangle$, $\langle V_1, V_2, V_3\rangle$).

Typical

|VARIABLES1| > 1
range(VARIABLES1.var) > 1
|VARIABLES2| > 1
range(VARIABLES2.var) > 1

① $(1, 2, \langle 1, 2, 0, 5\rangle, \langle 6, 1, 1\rangle)$
② $(1, 2, \langle 1, 2, 0, 6\rangle, \langle 5, 1, 1\rangle)$
③ $(1, 2, \langle 2, 1, 0, 5\rangle, \langle 6, 1, 1\rangle)$
④ $(1, 2, \langle 2, 1, 0, 6\rangle, \langle 5, 1, 1\rangle)$
⑤ $(1, 2, \langle 2, 2, 1, 5\rangle, \langle 6, 1, 1\rangle)$
⑥ $(1, 2, \langle 2, 2, 1, 6\rangle, \langle 5, 1, 1\rangle)$

Figure 5.203: All solutions corresponding to the non ground example of the COMMON constraint of the **All solutions** slot

Symmetries

- Arguments are permutable w.r.t. permutation (NCOMMON1, NCOMMON2) (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Arg. properties

- Functional dependency: NCOMMON1 determined by VARIABLES1 and VARIABLES2.
- Functional dependency: NCOMMON2 determined by VARIABLES1 and VARIABLES2.

Remark

It was shown in [76] that, finding out whether the COMMON constraint has a solution or not is NP-hard. This was achieved by reduction from 3-SAT.

See also

common keyword: ALLDIFFERENT_ON_INTERSECTION, NVALUE_ON_INTERSECTION, SAME_INTERSECTION *(constraint on the intersection)*.

generalisation: COMMON_INTERVAL (variable *replaced by* variable/constant), COMMON_MODULO (variable *replaced by* variable mod constant), COMMON_PARTITION (variable *replaced by* variable $\in$ partition).

related: AMONG_VAR, ROOTS.

root concept: AMONG.

specialisation: USES (NCOMMON2=|VARIABLES2|).

Keywords

complexity: 3-SAT.

constraint arguments: constraint between two collections of variables, pure functional dependency.

constraint type: constraint on the intersection.

final graph structure: acyclic, bipartite, no loop.

modelling: functional dependency.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	• **NSOURCE**= NCOMMON1 • **NSINK**= NCOMMON2
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.204 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the final graph has only 3 sources and 4 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 4. Note that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.

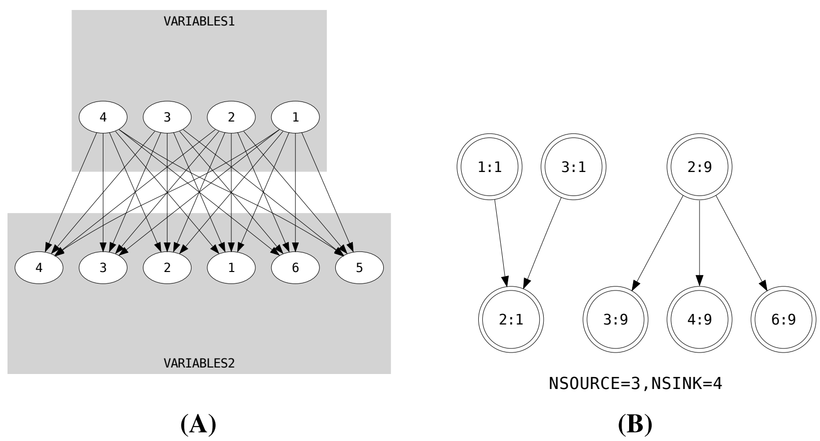


Figure 5.204: Initial and final graph of the COMMON constraint

5.78 COMMON_INTERVAL

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from COMMON.

Constraint

$$\text{COMMON\_INTERVAL}\left(\begin{array}{l}\texttt{NCOMMON1},\\ \texttt{NCOMMON2},\\ \texttt{VARIABLES1},\\ \texttt{VARIABLES2},\\ \texttt{SIZE\_INTERVAL}\end{array}\right)$$

Arguments

NCOMMON1	:	dvar
NCOMMON2	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
SIZE_INTERVAL	:	int

Restrictions

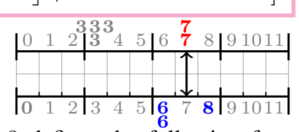
$\texttt{NCOMMON1} \geq 0$
$\texttt{NCOMMON1} \leq |\texttt{VARIABLES1}|$
$\texttt{NCOMMON2} \geq 0$
$\texttt{NCOMMON2} \leq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\texttt{SIZE\_INTERVAL} > 0$

Purpose

NCOMMON1 is the number of variables of the collection of variables VARIABLES1 taking a value in one of the intervals derived from the values assigned to the variables of the collection VARIABLES2: To each value v assigned to a variable of the collection VARIABLES2 we associate the interval $[\texttt{SIZE\_INTERVAL} \cdot \lfloor v/\texttt{SIZE\_INTERVAL}\rfloor, \texttt{SIZE\_INTERVAL} \cdot \lfloor v/\texttt{SIZE\_INTERVAL}\rfloor + \texttt{SIZE\_INTERVAL} - 1]$.
NCOMMON2 is the number of variables of the collection of variables VARIABLES2 taking a value in one of the intervals derived from the values assigned to the variables of the collection VARIABLES1: To each value v assigned to a variable of the collection VARIABLES1 we associate the interval $[\texttt{SIZE\_INTERVAL} \cdot \lfloor v/\texttt{SIZE\_INTERVAL}\rfloor, \texttt{SIZE\_INTERVAL} \cdot \lfloor v/\texttt{SIZE\_INTERVAL}\rfloor + \texttt{SIZE\_INTERVAL} - 1]$.

Example

$(3, 2, \langle 8, 6, 6, 0\rangle, \langle 7, 3, 3, 3, 3, 7\rangle, 3)$



In the example, the last argument $\texttt{SIZE\_INTERVAL} = 3$ defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. As a consequence the items of collection $\langle 8, 6, 6, 0\rangle$ respectively correspond to intervals $[6, 8]$, $[6, 8]$, $[6, 8]$ and $[0, 2]$. Similarly the items of collection $\langle 7, 3, 3, 3, 3, 7\rangle$ respectively correspond to intervals $[6, 8]$, $[3, 5]$, $[3, 5]$, $[3, 5]$, $[3, 5]$, $[6, 8]$. The COMMON_INTERVAL constraint holds since:

- Its first argument $\texttt{NCOMMON1} = 3$ is the number of intervals associated with the items of collection $\langle 8, 6, 6, 0\rangle$ that also correspond to intervals associated with $\langle 7, 3, 3, 3, 3, 7\rangle$.

- Its second argument $\texttt{NCOMMON2} = 2$ is the number of intervals associated with the items of collection $\langle 7, 3, 3, 3, 3, 7\rangle$ that also correspond to intervals associated with $\langle 8, 6, 6, 0\rangle$.

Typical

$|\texttt{VARIABLES1}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$|\texttt{VARIABLES2}| > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$\texttt{SIZE\_INTERVAL} > 1$
$\texttt{SIZE\_INTERVAL} < \texttt{range}(\texttt{VARIABLES1.var})$
$\texttt{SIZE\_INTERVAL} < \texttt{range}(\texttt{VARIABLES2.var})$

Symmetries

- Arguments are permutable w.r.t. permutation (NCOMMON1, NCOMMON2) (VARIABLES1, VARIABLES2) (SIZE_INTERVAL).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.
- An occurrence of a value of VARIABLES2.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties

- Functional dependency: NCOMMON1 determined by VARIABLES1, VARIABLES2 and SIZE_INTERVAL.
- Functional dependency: NCOMMON2 determined by VARIABLES1, VARIABLES2 and SIZE_INTERVAL.

See also

specialisation: COMMON (variable/constant *replaced by* variable).

Keywords

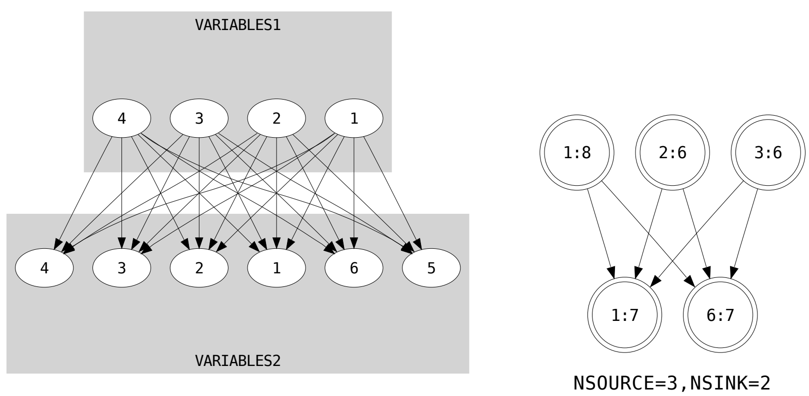
constraint arguments: constraint between two collections of variables, pure functional dependency.

final graph structure: acyclic, bipartite, no loop.

modelling: interval, functional dependency.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL
Graph property(ies)	• **NSOURCE**= NCOMMON1 • **NSINK**= NCOMMON2
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.205 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3 sources and 2 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 2. Note that the vertices corresponding to the variables that take values 0 or 3 were removed from the final graph since there is no arc for which the associated arc constraint holds.



(A) (B)

Figure 5.205: Initial and final graph of the COMMON_INTERVAL constraint

5.79 COMMON_MODULO

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from COMMON.

Constraint COMMON_MODULO(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2, M)

Arguments

NCOMMON1	:	dvar
NCOMMON2	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
M	:	int

Restrictions

NCOMMON1 ≥ 0
NCOMMON1 $\leq$ |VARIABLES1|
NCOMMON2 ≥ 0
NCOMMON2 $\leq$ |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
M > 0

Purpose

NCOMMON1 is the number of variables of the collection of variables VARIABLES1 taking a value situated in an equivalence class (congruence modulo a fixed number M) derived from the values assigned to the variables of VARIABLES2 and from M.
NCOMMON2 is the number of variables of the collection of variables VARIABLES2 taking a value situated in an equivalence class (congruence modulo a fixed number M) derived from the values assigned to the variables of VARIABLES1 and from M.

Example

$(3, 4, \langle 0, 4, 0, 8\rangle, \langle 7, 5, 4, 9, 2, 4\rangle, 5)$

In the example, the last argument M $= 5$ defines the equivalence classes $a \equiv 0 \pmod 5$, $a \equiv 1 \pmod 5$, $a \equiv 2 \pmod 5$, $a \equiv 3 \pmod 5$, and $a \equiv 4 \pmod 5$ where a is an integer. As a consequence the items of collection $\langle 0, 4, 0, 8\rangle$ respectively correspond to the equivalence classes $a \equiv 0 \pmod 5$, $a \equiv 4 \pmod 5$, $a \equiv 0 \pmod 5$, and $a \equiv 3 \pmod 5$. Similarly the items of collection $\langle 7, 5, 4, 9, 2, 4\rangle$ respectively correspond to the equivalence classes $a \equiv 2 \pmod 5$, $a \equiv 0 \pmod 5$, $a \equiv 4 \pmod 5$, $a \equiv 4 \pmod 5$, $a \equiv 2 \pmod 5$, and $a \equiv 4 \pmod 5$. The COMMON_MODULO constraint holds since:

- Its first argument NCOMMON1 $= 3$ is the number of equivalence classes associated with the items of collection $\langle 0, 4, 0, 8\rangle$ that also correspond to equivalence classes associated with $\langle 7, 5, 4, 9, 2, 4\rangle$.
- Its second argument NCOMMON2 $= 4$ is the number of equivalence classes associated with the items of collection $\langle 7, 5, 4, 9, 2, 4\rangle$ that also correspond to equivalence classes associated with $\langle 0, 4, 0, 8\rangle$.

Typical

$|\texttt{VARIABLES1}| > 1$
range$(\texttt{VARIABLES1.var}) > 1$
$|\texttt{VARIABLES2}| > 1$
range$(\texttt{VARIABLES2.var}) > 1$
$\texttt{M} > 1$
$\texttt{M} <$ maxval$(\texttt{VARIABLES1.var})$
$\texttt{M} <$ maxval$(\texttt{VARIABLES2.var})$

Symmetries

- Arguments are permutable w.r.t. permutation $(\texttt{NCOMMON1}, \texttt{NCOMMON2})$ $(\texttt{VARIABLES1}, \texttt{VARIABLES2})$ $(\texttt{M})$.
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value u of VARIABLES1.var can be replaced by any other value v such that v is congruent to u modulo M.
- An occurrence of a value u of VARIABLES2.var can be replaced by any other value v such that v is congruent to u modulo M.

Arg. properties

- Functional dependency: NCOMMON1 determined by VARIABLES1, VARIABLES2 and M.
- Functional dependency: NCOMMON2 determined by VARIABLES1, VARIABLES2 and M.

See also

specialisation: COMMON *(*variable mod constant *replaced by* variable*)*.

Keywords

characteristic of a constraint: modulo.

constraint arguments: constraint between two collections of variables, pure functional dependency.

final graph structure: acyclic, bipartite, no loop.

modelling: functional dependency.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* ↦collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var mod M = variables2.var mod M
Graph property(ies)	• **NSOURCE**= NCOMMON1 • **NSINK**= NCOMMON2
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.206 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3 sources and 4 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 4. Note that the vertices corresponding to the variables that take values 8, 7 or 2 were removed from the final graph since there is no arc for which the associated arc constraint holds.

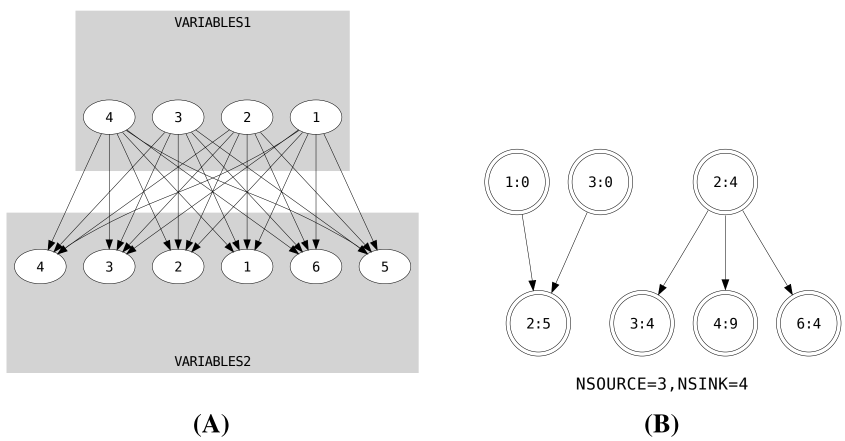


Figure 5.206: Initial and final graph of the COMMON_MODULO constraint

5.80 COMMON_PARTITION

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin

Derived from COMMON.

Constraint

$\text{COMMON\_PARTITION}\left(\begin{array}{l}\texttt{NCOMMON1},\\ \texttt{NCOMMON2},\\ \texttt{VARIABLES1},\\ \texttt{VARIABLES2},\\ \texttt{PARTITIONS}\end{array}\right)$

Type

VALUES : collection(val−int)

Arguments

NCOMMON1 : dvar
NCOMMON2 : dvar
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

$|\texttt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)
$\texttt{NCOMMON1} \geq 0$
$\texttt{NCOMMON1} \leq |\texttt{VARIABLES1}|$
$\texttt{NCOMMON2} \geq 0$
$\texttt{NCOMMON2} \leq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
$|\texttt{PARTITIONS}| \geq 2$

Purpose

NCOMMON1 is the number of variables of the VARIABLES1 collection taking a value in a partition derived from the values assigned to the variables of VARIABLES2 and from PARTITIONS.
NCOMMON2 is the number of variables of the VARIABLES2 collection taking a value in a partition derived from the values assigned to the variables of VARIABLES1 and from PARTITIONS.

Example

$$\left(\begin{array}{l}3, 4, \langle 2, 3, 6, 0\rangle,\\ \langle 0, 6, 3, 3, 7, 1\rangle,\\ \langle \texttt{p} - \langle 1, 3\rangle, \texttt{p} - \langle 4\rangle, \texttt{p} - \langle 2, 6\rangle\rangle\end{array}\right)$$

In the example, the last argument PARTITIONS defines the partitions $\texttt{p} - \langle 1, 3\rangle$, $\texttt{p} - \langle 4\rangle$ and $\texttt{p} - \langle 2, 6\rangle$. As a consequence the first three items of collection $\langle 2, 3, 6, 0\rangle$ respectively correspond to the partitions $\texttt{p} - \langle 2, 6\rangle$, $\texttt{p} - \langle 1, 3\rangle$, and $\texttt{p} - \langle 2, 6\rangle$. Similarly the items of collection $\langle 0, 6, 3, 3, 7, 1\rangle$ (from which we remove items 0 and 7 since they do not belong to any partition) respectively correspond to the partitions $\mathtt{p} - \langle 2, 6\rangle$, $\mathtt{p} - \langle 1, 3\rangle$, $\mathtt{p} - \langle 1, 3\rangle$, and $\mathtt{p} - \langle 1, 3\rangle$. The COMMON_PARTITION constraint holds since:

- Its first argument NCOMMON1 $= 3$ is the number of partitions associated with the items of collection $\langle 2, 3, 6, 0\rangle$ that also correspond to partitions associated with $\langle 0, 6, 3, 3, 7, 1\rangle$.
- Its second argument NCOMMON2 $= 4$ is the number of partitions associated with the items of collection $\langle 0, 6, 3, 3, 7, 1\rangle$ that also correspond to partitions associated with $\langle 2, 3, 6, 0\rangle$.

Typical

$|\mathtt{VARIABLES1}| > 1$
$\mathtt{range}(\mathtt{VARIABLES1.var}) > 1$
$|\mathtt{VARIABLES2}| > 1$
$\mathtt{range}(\mathtt{VARIABLES2.var}) > 1$
$|\mathtt{VARIABLES1}| > |\mathtt{PARTITIONS}|$
$|\mathtt{VARIABLES2}| > |\mathtt{PARTITIONS}|$

Symmetries

- Arguments are permutable w.r.t. permutation (NCOMMON1, NCOMMON2) (VARIABLES1, VARIABLES2) (PARTITIONS).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Arg. properties

- Functional dependency: NCOMMON1 determined by VARIABLES1, VARIABLES2 and PARTITIONS.
- Functional dependency: NCOMMON2 determined by VARIABLES1, VARIABLES2 and PARTITIONS.

See also

specialisation: COMMON (variable $\in$ partition *replaced by* variable).

used in graph description: IN_SAME_PARTITION.

Keywords

characteristic of a constraint: partition.

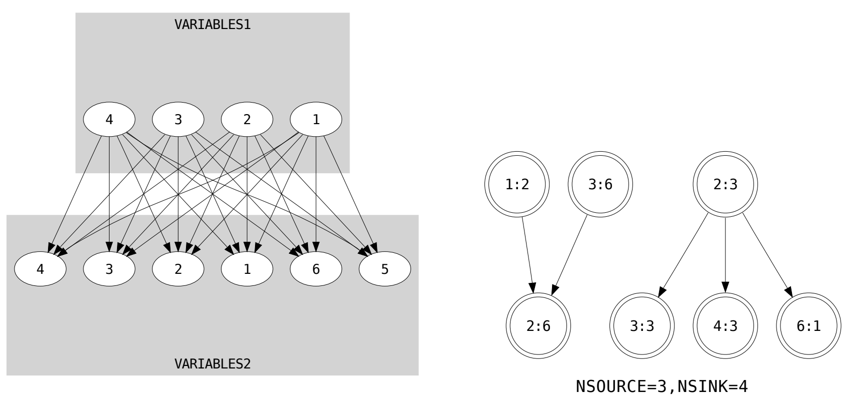
constraint arguments: constraint between two collections of variables, pure functional dependency.

final graph structure: acyclic, bipartite, no loop.

modelling: functional dependency.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	• **NSOURCE** = NCOMMON1 • **NSINK** = NCOMMON2
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.207 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since the graph has only 3 sources and 4 sinks the variables NCOMMON1 and NCOMMON2 are respectively equal to 3 and 4. Note that the vertices corresponding to the variables that take values 0 or 7 were removed from the final graph since there is no arc for which the associated IN_SAME_PARTITION constraint holds.



(A) (B)

Figure 5.207: Initial and final graph of the COMMON_PARTITION constraint

5.81 COMPARE_AND_COUNT

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Generalise DISCREPANCY

Constraint COMPARE_AND_COUNT(VARIABLES1, VARIABLES2, COMPARE, COUNT, LIMIT)

Arguments

VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
COMPARE	:	atom
COUNT	:	atom
LIMIT	:	dvar

Restrictions

$|\texttt{VARIABLES1}| = |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\texttt{COMPARE} \in [=, \neq, <, \geq, >, \leq]$
$\texttt{COUNT} \in [=, \neq, <, \geq, >, \leq]$
$\texttt{LIMIT} \geq 0$

Purpose

Enforce the condition
$\left(\sum_{i=1}^{|\texttt{VARIABLES1}|} \texttt{VARIABLES1}[i].\texttt{var}\ \texttt{COMPARE}\ \texttt{VARIABLES2}[i].\texttt{var}\right)\ \texttt{COUNT}\ \texttt{LIMIT}$.

Example

$(\langle 4, 5, 5, 4, 5\rangle, \langle 4, 2, 5, 1, 5\rangle, =, \leq, 3)$

The COMPARE_AND_COUNT constraint holds since no more than $\texttt{LIMIT} = 3$ pairs of variables are equal, i.e., the first, third and fifth pairs.

Typical

$|\texttt{VARIABLES1}| > 1$
$\text{range}(\texttt{VARIABLES1.var}) > 1$
$\text{range}(\texttt{VARIABLES2.var}) > 1$
$\texttt{COMPARE} \in [=]$
$\texttt{COUNT} \in [=, <, \geq, >, \leq]$
$\texttt{LIMIT} > 0$
$\texttt{LIMIT} < |\texttt{VARIABLES1}|$

Arg. properties

- Contractible wrt. VARIABLES1 and VARIABLES2 (*remove items from same position*) when $\texttt{COUNT} \in [<, \leq]$.
- Extensible wrt. VARIABLES1 and VARIABLES2 (*add items at same position*) when $\texttt{COUNT} \in [\geq, >]$.

See also **common keyword:** COUNT (*counting constraint*).

Keywords **constraint type:** predefined constraint, counting constraint.

5.82 COND_LEX_COST

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Inspired by [448].

Constraint COND_LEX_COST(VECTOR, PREFERENCE_TABLE, COST)

Type

TUPLE_OF_VALS : collection(val−int)

Arguments

VECTOR : collection(var−dvar)
PREFERENCE_TABLE : collection(tuple − TUPLE_OF_VALS)
COST : dvar

Restrictions

|TUPLE_OF_VALS| ≥ 1
required(TUPLE_OF_VALS, val)
required(VECTOR, var)
|VECTOR| = |TUPLE_OF_VALS|
required(PREFERENCE_TABLE, tuple)
same_size(PREFERENCE_TABLE, tuple)
distinct(PREFERENCE_TABLE, [])
IN_RELATION(VECTOR, PREFERENCE_TABLE)
COST ≥ 1
COST $\leq$ |PREFERENCE_TABLE|

Purpose VECTOR is assigned to the COSTth item of the collection PREFERENCE_TABLE.

Example

$$\left(\begin{array}{l}\langle 0,1\rangle, \\ \left\langle\begin{array}{l}\text{tuple} - \langle 1,0\rangle, \\ \text{tuple} - \langle 0,1\rangle, \\ \text{tuple} - \langle 0,0\rangle, \\ \text{tuple} - \langle 1,1\rangle\end{array}\right\rangle, 2\end{array}\right)$$

The COND_LEX_COST constraint holds since VECTOR is assigned to the second item of the collection PREFERENCE_TABLE.

Typical

|TUPLE_OF_VALS| > 1
|VECTOR| > 1
|PREFERENCE_TABLE| > 1

Symmetries

- Items of VECTOR and PREFERENCE_TABLE.tuple are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values in VECTOR or PREFERENCE_TABLE.tuple can be swapped; all occurrences of a tuple of values in VECTOR or PREFERENCE_TABLE.tuple can be renamed to any unused tuple of values.

Usage

We consider an example taken from [448] were a customer has to decide among vacations. There are two seasons when he can travel (`spring` and `summer`) and two locations `Naples` and `Helsinki`. Furthermore assume that location is more important than season and the preferred period of the year depends on the selected location. The travel preferences of a customer are explicitly defined by stating the preferences ordering among the possible tuples of values $\langle$`Naples`, `spring`$\rangle$, $\langle$`Naples`, `summer`$\rangle$, $\langle$`Helsinki`, `spring`$\rangle$ and $\langle$`Helsinki`, `summer`$\rangle$. For instance we may state within the preference table `PREFERENCE_TABLE` of the COND_LEX_COST constraint the preference ordering $\langle$`Naples`, `spring`$\rangle \succ \langle$`Helsinki`, `summer`$\rangle \succ \langle$`Helsinki`, `spring`$\rangle \succ \langle$`Naples`, `summer`$\rangle$, which denotes the fact that our customer prefers `Naples` in the `spring` and `Helsinki` in the `summer`, and a vacation in `spring` is preferred over `summer`. Finally a solution minimising the cost variable `COST` will match the preferences stated by our customer.

See also

attached to cost variant: IN_RELATION *(*`COST` *parameter removed)*.

common keyword: COND_LEX_GREATER, COND_LEX_GREATEREQ, COND_LEX_LESS, COND_LEX_LESSEQ *(preferences)*.

specialisation: ELEMENT *(*`tuple` *of* `variables` *replaced by single* `variable`*)*.

Keywords

characteristic of a constraint: vector, automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: arc-consistency, cost filtering constraint.

modelling: preferences.

symmetry: lexicographic order.

Automaton Figure 5.208 depicts the automaton associated with COND_LEX_LESSEQ constraint. Let $\mathtt{VAR}_k$ denote the `var` attribute of the k^{th} item of the `VECTOR` collection. Figure 5.209 depicts the reformulation of the COND_LEX_COST constraint.

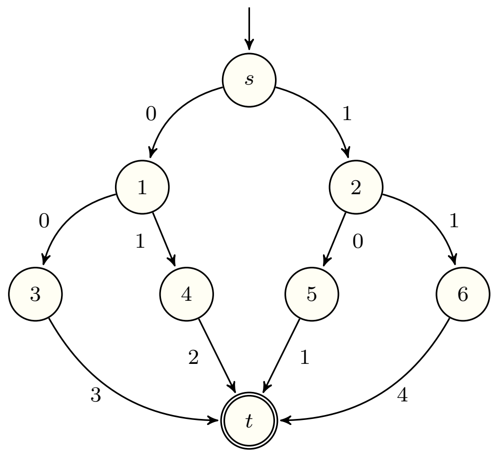


Figure 5.208: Automaton of the COND_LEX_COST constraint given in the **Example** slot

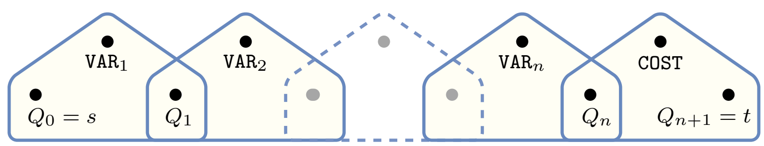


Figure 5.209: Hypergraph of the reformulation corresponding to the automaton of the COND_LEX_COST constraint

5.83 COND_LEX_GREATER

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Inspired by [448].

Constraint COND_LEX_GREATER(VECTOR1, VECTOR2, PREFERENCE_TABLE)

Type TUPLE_OF_VALS : collection(val−int)

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)
PREFERENCE_TABLE : collection(tuple − TUPLE_OF_VALS)

Restrictions

$|\text{TUPLE\_OF\_VALS}| \geq 1$
required(TUPLE_OF_VALS, val)
required(VECTOR1, var)
required(VECTOR2, var)
$|\text{VECTOR1}| = |\text{VECTOR2}|$
$|\text{VECTOR1}| = |\text{TUPLE\_OF\_VALS}|$
required(PREFERENCE_TABLE, tuple)
same_size(PREFERENCE_TABLE, tuple)
distinct(PREFERENCE_TABLE, [])
IN_RELATION(VECTOR1, PREFERENCE_TABLE)
IN_RELATION(VECTOR2, PREFERENCE_TABLE)

Purpose VECTOR1 and VECTOR2 are both assigned to the I^{th} and J^{th} items of the collection PREFERENCE_TABLE such that $\text{I} > \text{J}$.

Example

$$\left(\begin{array}{l}\langle 0,0\rangle, \\ \langle 1,0\rangle, \\ \left\langle\begin{array}{l}\texttt{tuple}-\langle 1,0\rangle, \\ \texttt{tuple}-\langle 0,1\rangle, \\ \texttt{tuple}-\langle 0,0\rangle, \\ \texttt{tuple}-\langle 1,1\rangle\end{array}\right\rangle\end{array}\right)$$

The COND_LEX_GREATER constraint holds since VECTOR1 and VECTOR2 are respectively assigned to the third and first items of the collection PREFERENCE_TABLE.

Typical

$|\text{TUPLE\_OF\_VALS}| > 1$
$|\text{VECTOR1}| > 1$
$|\text{VECTOR2}| > 1$
$|\text{PREFERENCE\_TABLE}| > 1$

Symmetries

- Items of VECTOR1, VECTOR2 and PREFERENCE_TABLE.tuple are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be swapped; all occurrences of a tuple of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be renamed to any unused tuple of values.

Usage

See COND_LEX_COST.

See also

common keyword: COND_LEX_COST, COND_LEX_GREATEREQ, COND_LEX_LESS, COND_LEX_LESSEQ *(preferences)*, LEX_GREATER *(lexicographic order)*.

implies: COND_LEX_GREATEREQ.

Keywords

characteristic of a constraint: vector, automaton.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: arc-consistency.

modelling: preferences.

symmetry: lexicographic order.

Automaton

Figure 5.210 depicts the automaton associated with the preference table of the COND_LEX_GREATER constraint given in the example. Let $\mathtt{VAR1}_k$ and $\mathtt{VAR2}_k$ respectively be the `var` attributes of the k^{th} items of the `VECTOR1` and the `VECTOR2` collections. Figure 5.211 depicts the reformulation of the COND_LEX_GREATER constraint. This reformulation uses:

- Two occurrences of the automaton depicted by Figure 5.210 for computing the positions `I` and `J` within the preference table corresponding to `VECTOR1` and `VECTOR2`.
- The binary constraint $\mathtt{I} > \mathtt{J}$.

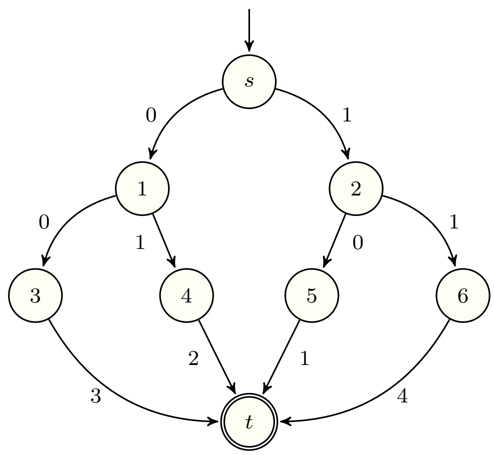


Figure 5.210: Automaton associated with the preference table of the COND_LEX_GREATER constraint given in the **Example** slot

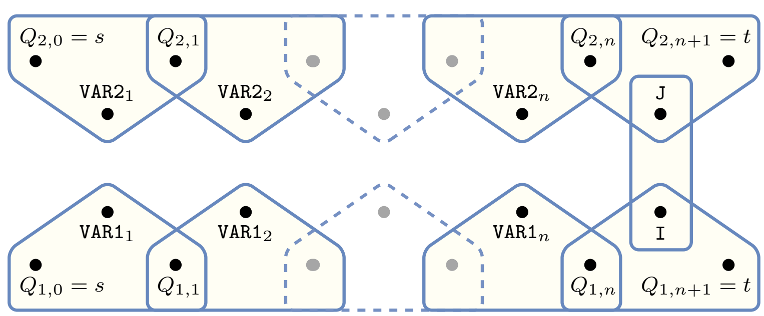


Figure 5.211: Hypergraph of the reformulation corresponding to the COND_LEX_GREATER constraint: it uses two occurrences of the automaton of Figure 5.210 and the constraint $\mathtt{I} > \mathtt{J}$

5.84 COND_LEX_GREATEREQ

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Inspired by [448].

Constraint COND_LEX_GREATEREQ(VECTOR1, VECTOR2, PREFERENCE_TABLE)

Type TUPLE_OF_VALS : collection(val−int)

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)
PREFERENCE_TABLE : collection(tuple − TUPLE_OF_VALS)

Restrictions

$|\text{TUPLE\_OF\_VALS}| \geq 1$
required(TUPLE_OF_VALS, val)
required(VECTOR1, var)
required(VECTOR2, var)
$|\text{VECTOR1}| = |\text{VECTOR2}|$
$|\text{VECTOR1}| = |\text{TUPLE\_OF\_VALS}|$
required(PREFERENCE_TABLE, tuple)
same_size(PREFERENCE_TABLE, tuple)
distinct(PREFERENCE_TABLE, [])
IN_RELATION(VECTOR1, PREFERENCE_TABLE)
IN_RELATION(VECTOR2, PREFERENCE_TABLE)

Purpose

VECTOR1 and VECTOR2 are both assigned to the I^{th} and J^{th} items of the collection PREFERENCE_TABLE such that $\text{I} \geq \text{J}$.

Example

$$\left(\begin{array}{l}\langle 0, 0\rangle, \\ \langle 1, 0\rangle, \\ \left\langle\begin{array}{l}\text{tuple} - \langle 1, 0\rangle, \\ \text{tuple} - \langle 0, 1\rangle, \\ \text{tuple} - \langle 0, 0\rangle, \\ \text{tuple} - \langle 1, 1\rangle\end{array}\right\rangle\end{array}\right)$$

The COND_LEX_GREATEREQ constraint holds since VECTOR1 and VECTOR2 are respectively assigned to the third and first items of the collection PREFERENCE_TABLE.

Typical

$|\text{TUPLE\_OF\_VALS}| > 1$
$|\text{VECTOR1}| > 1$
$|\text{VECTOR2}| > 1$
$|\text{PREFERENCE\_TABLE}| > 1$

Symmetries

- Items of `VECTOR1`, `VECTOR2` and `PREFERENCE_TABLE.tuple` are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values in `VECTOR1`, `VECTOR2` or `PREFERENCE_TABLE.tuple` can be swapped; all occurrences of a tuple of values in `VECTOR1`, `VECTOR2` or `PREFERENCE_TABLE.tuple` can be renamed to any unused tuple of values.

Usage

See COND_LEX_COST.

See also

common keyword: COND_LEX_COST, COND_LEX_GREATER, COND_LEX_LESS, COND_LEX_LESSEQ *(preferences)*, LEX_GREATEREQ *(lexicographic order)*.

implied by: COND_LEX_GREATER.

Keywords

characteristic of a constraint: vector, automaton.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: arc-consistency.

modelling: preferences.

symmetry: lexicographic order.

Automaton Figure 5.212 depicts the automaton associated with the preference table of the COND_LEX_GREATEREQ constraint given in the example. Let $\mathtt{VAR1}_k$ and $\mathtt{VAR2}_k$ respectively be the `var` attributes of the k^{th} items of the `VECTOR1` and the `VECTOR2` collections. Figure 5.213 depicts the reformulation of the COND_LEX_GREATEREQ constraint. This reformulation uses:

- Two occurrences of the automaton depicted by Figure 5.212 for computing the positions `I` and `J` within the preference table corresponding to `VECTOR1` and `VECTOR2`.
- The binary constraint $\mathtt{I} \geq \mathtt{J}$.

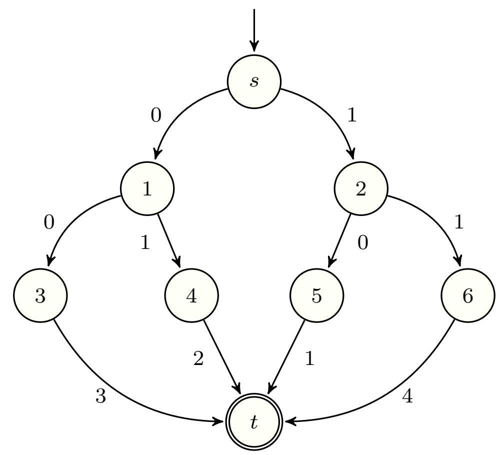


Figure 5.212: Automaton associated with the preference table of the COND_LEX_GREATEREQ constraint given in the **Example** slot

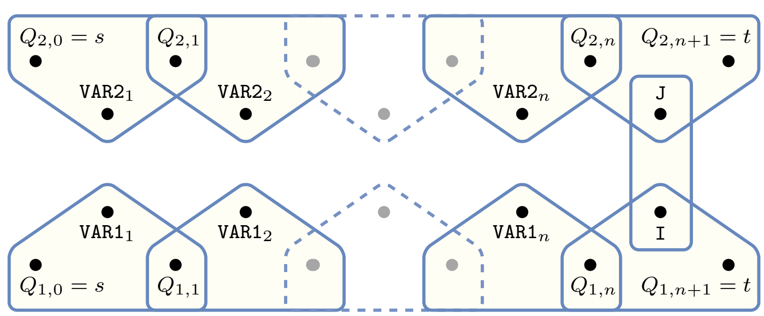


Figure 5.213: Hypergraph of the reformulation corresponding to the COND_LEX_GREATEREQ constraint: it uses two occurrences of the automaton of Figure 5.212 and the constraint $\mathtt{I} \geq \mathtt{J}$

5.85 COND_LEX_LESS

DESCRIPTION LINKS AUTOMATON

Origin

Inspired by [448].

Constraint

COND_LEX_LESS(VECTOR1, VECTOR2, PREFERENCE_TABLE)

Type

TUPLE_OF_VALS : collection(val−int)

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)
PREFERENCE_TABLE : collection(tuple − TUPLE_OF_VALS)

Restrictions

|TUPLE_OF_VALS| ≥ 1
required(TUPLE_OF_VALS, val)
required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|
|VECTOR1| = |TUPLE_OF_VALS|
required(PREFERENCE_TABLE, tuple)
same_size(PREFERENCE_TABLE, tuple)
distinct(PREFERENCE_TABLE, [])
IN_RELATION(VECTOR1, PREFERENCE_TABLE)
IN_RELATION(VECTOR2, PREFERENCE_TABLE)

Purpose

VECTOR1 and VECTOR2 are both assigned to the $\mathtt{I}^{th}$ and $\mathtt{J}^{th}$ items of the collection PREFERENCE_TABLE such that I < J.

Example

$$\left(\begin{array}{l}\langle 1,0\rangle,\\ \langle 0,0\rangle,\\ \left\langle\begin{array}{l}\texttt{tuple}-\langle 1,0\rangle,\\ \texttt{tuple}-\langle 0,1\rangle,\\ \texttt{tuple}-\langle 0,0\rangle,\\ \texttt{tuple}-\langle 1,1\rangle\end{array}\right\rangle\end{array}\right)$$

The COND_LEX_LESS constraint holds since VECTOR1 and VECTOR2 are respectively assigned to the first and third items of the collection PREFERENCE_TABLE.

Typical

|TUPLE_OF_VALS| > 1
|VECTOR1| > 1
|VECTOR2| > 1
|PREFERENCE_TABLE| > 1

Symmetries

- Items of `VECTOR1`, `VECTOR2` and `PREFERENCE_TABLE.tuple` are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values in `VECTOR1`, `VECTOR2` or `PREFERENCE_TABLE.tuple` can be swapped; all occurrences of a tuple of values in `VECTOR1`, `VECTOR2` or `PREFERENCE_TABLE.tuple` can be renamed to any unused tuple of values.

Usage

See COND_LEX_COST.

See also

common keyword: COND_LEX_COST, COND_LEX_GREATER, COND_LEX_GREATEREQ, COND_LEX_LESSEQ *(preferences)*, LEX_LESS *(lexicographic order)*.

implies: COND_LEX_LESSEQ.

Keywords

characteristic of a constraint: vector, automaton.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: arc-consistency.

modelling: preferences.

symmetry: lexicographic order.

Automaton Figure 5.214 depicts the automaton associated with the preference table of the COND_LEX_LESS constraint given in the example. Let $\texttt{VAR1}_k$ and $\texttt{VAR2}_k$ respectively be the `var` attributes of the k^{th} items of the VECTOR1 and the VECTOR2 collections. Figure 5.215 depicts the reformulation of the COND_LEX_LESS constraint. This reformulation uses:

- Two occurrences of the automaton depicted by Figure 5.214 for computing the positions I and J within the preference table corresponding to VECTOR1 and VECTOR2.
- The binary constraint I $<$ J.

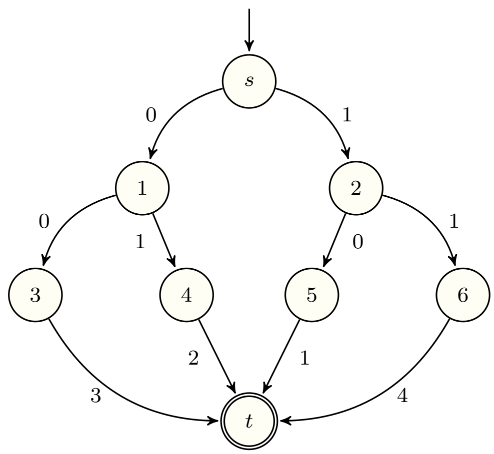


Figure 5.214: Automaton associated with the preference table of the COND_LEX_LESS constraint given in the **Example** slot

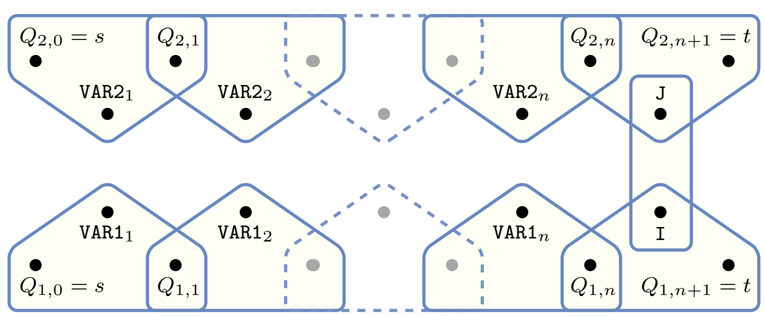


Figure 5.215: Hypergraph of the reformulation corresponding to the COND_LEX_LESS constraint: it uses two occurrences of the automaton of Figure 5.214 and the constraint I $<$ J

5.86 COND_LEX_LESSEQ

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Inspired by [448].

Constraint COND_LEX_LESSEQ(VECTOR1, VECTOR2, PREFERENCE_TABLE)

Type TUPLE_OF_VALS : collection(val−int)

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)
PREFERENCE_TABLE : collection(tuple − TUPLE_OF_VALS)

Restrictions

|TUPLE_OF_VALS| ≥ 1
required(TUPLE_OF_VALS, val)
required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|
|VECTOR1| = |TUPLE_OF_VALS|
required(PREFERENCE_TABLE, tuple)
same_size(PREFERENCE_TABLE, tuple)
distinct(PREFERENCE_TABLE, [])
IN_RELATION(VECTOR1, PREFERENCE_TABLE)
IN_RELATION(VECTOR2, PREFERENCE_TABLE)

Purpose VECTOR1 and VECTOR2 are both assigned to the $\mathtt{I}^{th}$ and $\mathtt{J}^{th}$ items of the collection PREFERENCE_TABLE such that $\mathtt{I} \leq \mathtt{J}$.

Example

$$\left(\begin{array}{l}\langle 1,0\rangle, \\ \langle 0,0\rangle, \\ \left\langle\begin{array}{l}\texttt{tuple}-\langle 1,0\rangle, \\ \texttt{tuple}-\langle 0,1\rangle, \\ \texttt{tuple}-\langle 0,0\rangle, \\ \texttt{tuple}-\langle 1,1\rangle\end{array}\right\rangle\end{array}\right)$$

The COND_LEX_LESSEQ constraint holds since VECTOR1 and VECTOR2 are respectively assigned to the first and third items of the collection PREFERENCE_TABLE.

Typical

|TUPLE_OF_VALS| > 1
|VECTOR1| > 1
|VECTOR2| > 1
|PREFERENCE_TABLE| > 1

Symmetries

- Items of VECTOR1, VECTOR2 and PREFERENCE_TABLE.tuple are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be swapped; all occurrences of a tuple of values in VECTOR1, VECTOR2 or PREFERENCE_TABLE.tuple can be renamed to any unused tuple of values.

Usage

See COND_LEX_COST.

See also

common keyword: COND_LEX_COST, COND_LEX_GREATER, COND_LEX_GREATEREQ, COND_LEX_LESS *(preferences)*, LEX_LESSEQ *(lexicographic order)*.

implied by: COND_LEX_LESS.

Keywords

characteristic of a constraint: vector, automaton.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: arc-consistency.

modelling: preferences.

symmetry: lexicographic order.

Automaton Figure 5.216 depicts the automaton associated with the preference table of the COND_LEX_LESSEQ constraint given in the example. Let $\mathtt{VAR1}_k$ and $\mathtt{VAR2}_k$ respectively be the `var` attributes of the k^{th} items of the `VECTOR1` and the `VECTOR2` collections. Figure 5.217 depicts the reformulation of the COND_LEX_LESSEQ constraint. This reformulation uses:

- Two occurrences of the automaton depicted by Figure 5.216 for computing the positions `I` and `J` within the preference table corresponding to `VECTOR1` and `VECTOR2`.
- The binary constraint $\mathtt{I} \leq \mathtt{J}$.

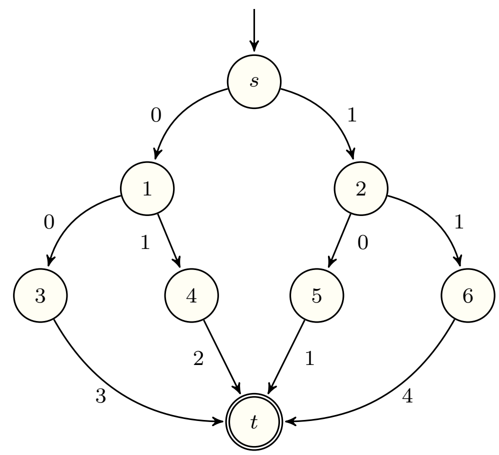


Figure 5.216: Automaton associated with the preference table of the COND_LEX_LESSEQ constraint given in the **Example** slot

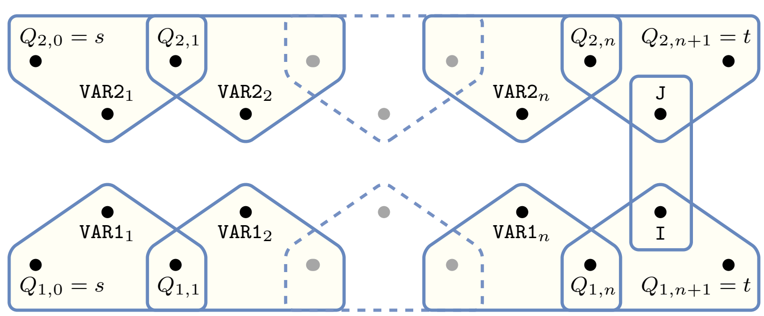


Figure 5.217: Hypergraph of the reformulation corresponding to the COND_LEX_LESSEQ constraint: it uses two occurrences of the automaton of Figure 5.216 and the constraint $\mathtt{I} \leq \mathtt{J}$

5.87 CONNECT_POINTS

▶ ▷ ◁ ◀

DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint CONNECT_POINTS(SIZE1, SIZE2, SIZE3, NGROUP, POINTS)

Arguments

SIZE1	:	int
SIZE2	:	int
SIZE3	:	int
NGROUP	:	dvar
POINTS	:	collection(p−dvar)

Restrictions

$\mathtt{SIZE1} > 0$
$\mathtt{SIZE2} > 0$
$\mathtt{SIZE3} > 0$
$\mathtt{NGROUP} \geq 0$
$\mathtt{NGROUP} \leq |\mathtt{POINTS}|$
$\mathtt{SIZE1} * \mathtt{SIZE2} * \mathtt{SIZE3} = |\mathtt{POINTS}|$
required(POINTS, p)

Purpose On a 3-dimensional grid of variables, number of groups, where a group consists of a connected set of variables that all have a same value distinct from 0.

Example

$$
\left(8,4,2,2,\left\langle\begin{array}{l}
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-1,\mathtt{p}-1,\\
\mathtt{p}-0,\mathtt{p}-2,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-1,\\
\mathtt{p}-0,\mathtt{p}-2,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-1,\\
\mathtt{p}-1,\mathtt{p}-1,\\
\mathtt{p}-1,\mathtt{p}-1,\\
\mathtt{p}-0,\mathtt{p}-2,\\
\mathtt{p}-0,\mathtt{p}-1,\\
\mathtt{p}-0,\mathtt{p}-2,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-2,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-2,\\
\mathtt{p}-2,\mathtt{p}-2,\\
\mathtt{p}-2,\mathtt{p}-2,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-2,\\
\mathtt{p}-0,\mathtt{p}-0,\\
\mathtt{p}-0,\mathtt{p}-2,\\
\mathtt{p}-0,\mathtt{p}-0
\end{array}\right\rangle\right)
$$

Figure 5.218 corresponds to the solution where we describe separately each layer of the grid. The CONNECT_POINTS constraint holds since we have two groups (NGROUP $=2$): a first one for the variables of the POINTS collection assigned to value 1, and a second one for the variables assigned to value 2.

0	0	**1**	**1**	0	**2**	0	0
0	0	0	**1**	0	**2**	0	0
0	0	0	**1**	**1**	**1**	**1**	**1**
0	**2**	0	**1**	0	**2**	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	**2**	0	0
0	**2**	**2**	**2**	**2**	**2**	0	0
0	**2**	0	0	0	**2**	0	0

Figure 5.218: The two layers of the solution

Typical

SIZE1 > 1
SIZE2 > 1
NGROUP > 0
NGROUP $<$ |POINTS|
|POINTS| > 3

Symmetry

All occurrences of two distinct values of POINTS.p that are both different from 0 can be swapped; all occurrences of a value of POINTS.p that is different from 0 can be renamed to any unused value that is also different from 0.

Arg. properties

Functional dependency: NGROUP determined by SIZE1, SIZE2, SIZE3 and POINTS.

Usage

Wiring problems [393], [462].

Algorithm

Since the graph corresponding to the 3-dimensional grid is symmetric one could certainly use as a starting point the filtering algorithm associated with the *number of connected components* graph property described in [58] (see the paragraphs "*Estimating* $\underline{\mathbf{NCC}}$" and "*Estimating* $\overline{\mathbf{NCC}}$"). One may also try to take advantage of the fact that the considered initial graph is a grid in order to simplify the previous filtering algorithm.

Keywords

characteristic of a constraint: joker value.

final graph structure: strongly connected component, symmetric.

geometry: geometrical constraint.

modelling: functional dependency.

problems: channel routing.

Arc input(s)	POINTS
Arc generator	*GRID*([SIZE1, SIZE2, SIZE3]) $\mapsto$collection(points1, points2)
Arc arity	2
Arc constraint(s)	• points1.p $\neq 0$ • points1.p = points2.p
Graph property(ies)	**NSCC**= NGROUP
Graph class	SYMMETRIC

Graph model Figure 5.219 gives the initial graph constructed by the *GRID* arc generator associated with the **Example** slot.

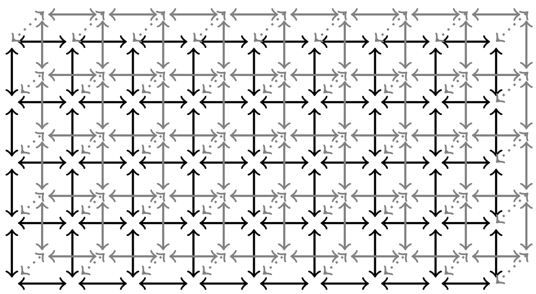


Figure 5.219: Graph generated by *GRID*([8,4,2])

5.88 CONNECTED

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [151]

Constraint CONNECTED(NODES)

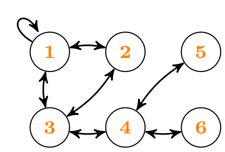
Argument NODES : collection(index−int, succ−svar)

Restrictions
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq |$NODES$|$

Purpose Consider a digraph G described by the NODES collection. Select a subset of arcs of G so that the corresponding graph is symmetric (i.e., if there is an arc from i to j, there is also an arc from j to i) and connected (i.e., there is a path between any pair of vertices of G).

Example

$$\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\{1,2,3\}, \\ \texttt{index}-2 & \texttt{succ}-\{1,3\}, \\ \texttt{index}-3 & \texttt{succ}-\{1,2,4\}, \\ \texttt{index}-4 & \texttt{succ}-\{3,5,6\}, \\ \texttt{index}-5 & \texttt{succ}-\{4\}, \\ \texttt{index}-6 & \texttt{succ}-\{4\} \end{array}\right\rangle\right)$$



The CONNECTED constraint holds since the NODES collection depicts a symmetric graph involving a single connected component.

Typical $|$NODES$| > 1$

Symmetry Items of NODES are permutable.

Algorithm A filtering algorithm for the CONNECTED constraint is sketched in [151, page 88]. Beside the pruning associated with the fact that the final graph is symmetric, it is based on the fact that all bridges and cut vertices on a path between two vertices that should for sure belong to the final graph should also belong to the final graph.

See also
common keyword: SYMMETRIC *(symmetric)*.
implies: STRONGLY_CONNECTED.
used in graph description: IN_SET.

Keywords
constraint arguments: constraint involving set variables.
constraint type: graph constraint.
filtering: DFS-bottleneck.
final graph structure: connected component, symmetric.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.succ)
Graph property(ies)	**NCC**$= 1$
Graph class	SYMMETRIC

Graph model Part (A) of Figure 5.220 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.220 gives the final graph associated with the **Example** slot.

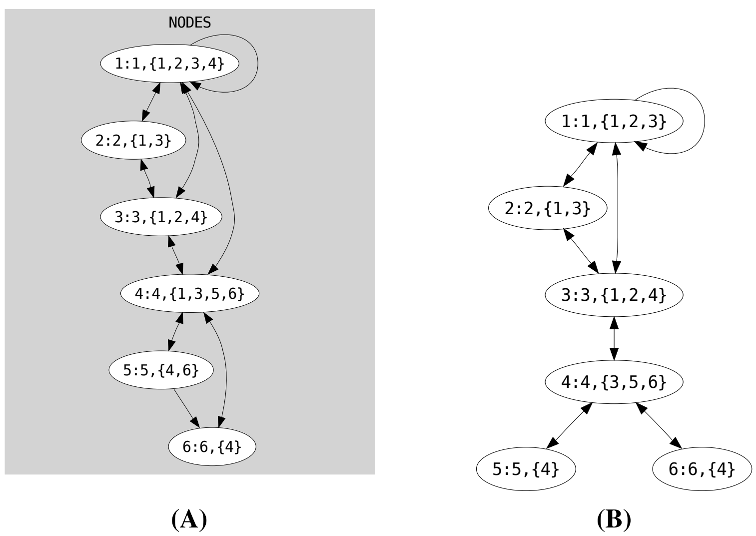


Figure 5.220: Initial and final graph of the CONNECTED set constraint

5.89 CONSECUTIVE_GROUPS_OF_ONES

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin

Derived from GROUP

Constraint

CONSECUTIVE_GROUPS_OF_ONES(GROUP_SIZES, VARIABLES)

Arguments

GROUP_SIZES	:	collection(nb−int)
VARIABLES	:	collection(var−dvar)

Restrictions

required(GROUP_SIZES, nb)
|GROUP_SIZES| ≥ 1
GROUP_SIZES.nb ≥ 1
GROUP_SIZES.nb $\leq$ |VARIABLES|
required(VARIABLES, var)
|VARIABLES| $\geq 2 *$ |GROUP_SIZES| $- 1$
|VARIABLES| $\geq$sum(GROUP_SIZES.nb) + |GROUP_SIZES| $- 1$
VARIABLES.var ≥ 0
VARIABLES.var ≤ 1

Purpose

In order to define the meaning of the CONSECUTIVE_GROUPS_OF_ONES constraint, we first introduce the notions of *stretch* and *span*. Let n be the number of variables of the collection VARIABLES and let m be the number of items of the collection GROUP_SIZES. Let $X_i, \ldots, X_j$ $(1 \leq i \leq j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ are assigned value 1,
- $i = 1$ or $X_{i-1} \neq 1$,
- $j = n$ or $X_{j+1} \neq 1$.

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $j - i + 1$. We now define the condition enforced by the CONSECUTIVE_GROUPS_OF_ONES constraint.

All variables of the VARIABLES collection should be assigned value 0 or 1. In addition there is |GROUP_SIZES| successive stretches of respective span GROUP_SIZES[1].nb, GROUP_SIZES[2].nb, ..., GROUP_SIZES[m].nb.

Example

$(\langle 2, 1\rangle, \langle 1, 1, 0, 0, 0, 1, 0\rangle)$

The CONSECUTIVE_GROUPS_OF_ONES constraint holds since the sequence 1 1 0 0 0 1 0 contains a first stretch (i.e., a maximum sequence of 1) of span 2 and a second stretch of span 1.

Typical

|VARIABLES| > 1
range(VARIABLES.var) > 1

Typical model $\text{sum}(\texttt{VARIABLES.var}) > 2$

Symmetry Items of GROUP_SIZES and VARIABLES are simultaneously reversable.

Usage The CONSECUTIVE_GROUPS_OF_ONES constraint can be used in order to model the logigraphe problem.

See also **root concept:** GROUP.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

filtering: arc-consistency.

modelling exercises: logigraphe.

puzzles: logigraphe.

Automaton Figure 5.221 depicts the automaton associated with the CONSECUTIVE_GROUPS_OF_ONES constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a signature variable that is equal to $\mathtt{VAR}_i$. There is no signature constraint.

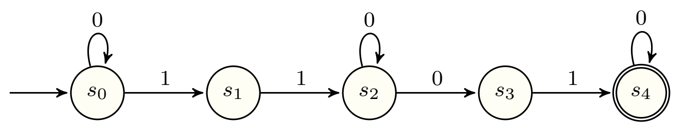


Figure 5.221: Non deterministic automaton of the CONSECUTIVE_GROUPS_OF_ONES constraint of the **Example** slot (a stretch of two 1 followed by a stretch of a single 1)

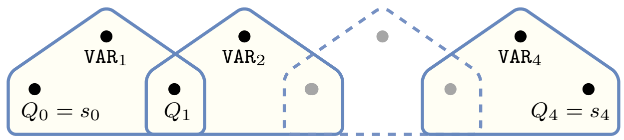


Figure 5.222: Hypergraph of the reformulation corresponding to the automaton of the CONSECUTIVE_GROUPS_OF_ONES constraint of the **Example** slot

5.90 CONSECUTIVE_VALUES

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin

Derived from ALLDIFFERENT_CONSECUTIVE_VALUES.

Constraint

CONSECUTIVE_VALUES(VARIABLES)

Argument

VARIABLES : collection(var−dvar)

Restriction

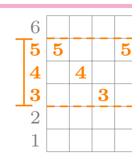
required(VARIABLES, var)

Purpose

Constraint the difference between the largest and the smallest values of the VARIABLES collection to be equal to the number of distinct values assigned to the variables of the VARIABLES collection minus one (i.e., there is no holes at all within the used values).

Example

$(\langle 5, 4, 3, 5\rangle)$



The CONSECUTIVE_VALUES constraint holds since all values between value 3 and value 5 are actually used.

Typical

$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1

Typical model

nval(VARIABLES.var) > 2

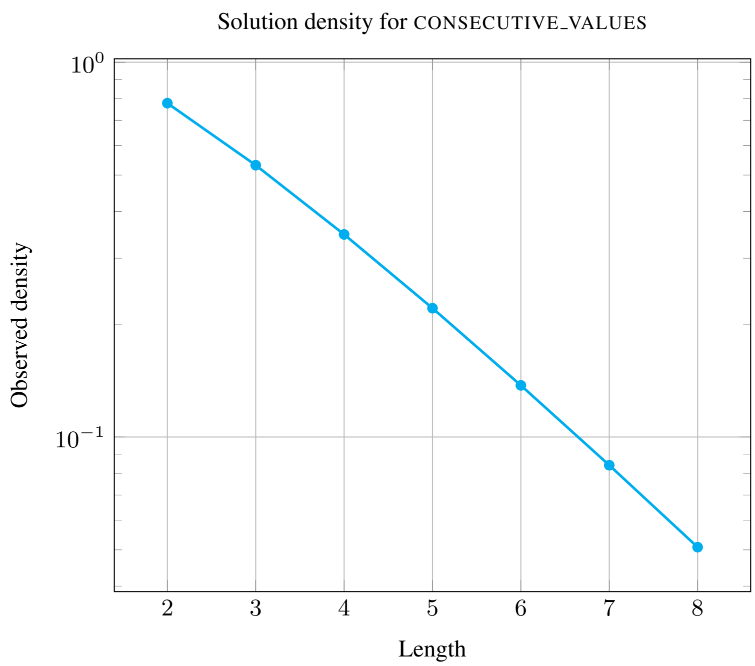
Symmetries

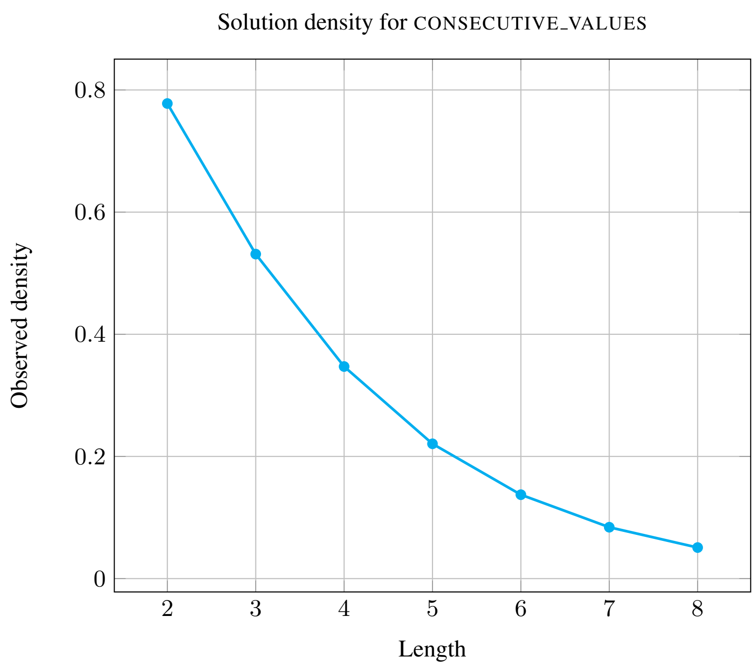
- Items of VARIABLES are permutable.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	7	34	217	1716	16159	176366	2187637

Number of solutions for CONSECUTIVE_VALUES: domains $0..n$





See also **implied by:** ALL_EQUAL, ALLDIFFERENT_CONSECUTIVE_VALUES, GLOBAL_CONTIGUITY.

used in reformulation: NVALUE.

Keywords **characteristic of a constraint:** sort based reformulation.

constraint type: value constraint, predefined constraint.

Cond. implications CONSECUTIVE_VALUES(VARIABLES)
with $|\texttt{VARIABLES}| >$ range(VARIABLES.var)
implies SOME_EQUAL(VARIABLES).

5.91 CONTAINS_SBOXES

DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [349]

Constraint CONTAINS_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonym CONTAINS.

Types

VARIABLES	:	collection(v−dvar)
INTEGERS	:	collection(v−int)
POSITIVES	:	collection(v−int)

Arguments

K	:	int
DIMS	:	sint
OBJECTS	:	collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES	:	collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

|VARIABLES| ≥ 1
|INTEGERS| ≥ 1
|POSITIVES| ≥ 1
required(VARIABLES, v)
|VARIABLES| = K
required(INTEGERS, v)
|INTEGERS| = K
required(POSITIVES, v)
|POSITIVES| = K
POSITIVES.v > 0
K > 0
DIMS ≥ 0
DIMS < K
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
OBJECTS.oid ≥ 1
OBJECTS.oid ≤ |OBJECTS|
OBJECTS.sid ≥ 1
OBJECTS.sid ≤ |SBOXES|
|SBOXES| ≥ 1
required(SBOXES, [sid, t, l])
SBOXES.sid ≥ 1
SBOXES.sid ≤ |SBOXES|
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i < j$, O_i contains O_j with respect to a set of dimensions depicted by `DIMS`. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a `K`-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id `sid`, shift offset `t`, and sizes `l`. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier `oid`, shape id `sid` and origin `x`.

An object O_i *contains* an object O_j with respect to a set of dimensions depicted by `DIMS` if and only if, for all shifted boxes s_j associated with O_j, there exists a shifted box s_i of O_i such that s_i contains s_j. A shifted box s_i *contains* a shifted box s_j if and only if, for all dimensions $d \in$ `DIMS`, (1) the start of s_i in dimension d is strictly less than the start of s_j in dimension d and (2) the end of s_j in dimension d is strictly less than the end of s_i in dimension d.

Example

$$\left(\begin{array}{l} 2, \{0,1\}, \\ \left\langle \begin{array}{lll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 2,2\rangle, \\ \texttt{oid}-3 & \texttt{sid}-3 & \texttt{x}-\langle 3,3\rangle \end{array} \right\rangle, \\ \left\langle \begin{array}{lll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 5,5\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,3\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle \end{array} \right\rangle \end{array}\right)$$

Figure 5.223 shows the objects of the example. Since O_1 contains both O_2 and O_3, and since O_2 contains O_3, the CONTAINS_SBOXES constraint holds.

Typical

$|\texttt{OBJECTS}| > 1$

Symmetries

- Items of `SBOXES` are permutable.
- Items of `OBJECTS.x`, `SBOXES.t` and `SBOXES.l` are permutable (*same permutation used*).

Arg. properties

Suffix-contractible wrt. `OBJECTS`.

Remark

One of the eight relations of the *Region Connection Calculus* [349]. The constraint CONTAINS_SBOXES is a restriction of the original relation since it requires that each shifted box of an object is contained by one shifted box of the other object.

See also

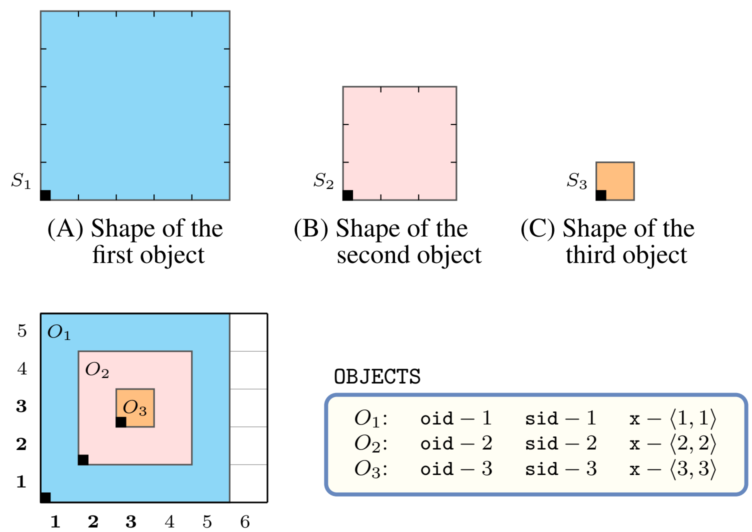
common keyword: COVEREDBY_SBOXES, COVERS_SBOXES, DISJOINT_SBOXES, EQUAL_SBOXES, INSIDE_SBOXES, MEET_SBOXES *(rcc8)*, NON_OVERLAP_SBOXES *(geometrical constraint,logic)*, OVERLAP_SBOXES *(rcc8)*.

Keywords

constraint type: logic.

geometry: geometrical constraint, rcc8.

miscellaneous: obscure.



(A) Shape of the first object

(B) Shape of the second object

(C) Shape of the third object

(D) Three objects O_1, O_2, O_3, where O_1 both contains O_2 and O_3 and where O_2 contains O_3

Figure 5.223: (D) the three nested objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_1, S_2, S_3; (A), (B), (C) shapes S_1, S_2 and S_3 are made up from a single shifted box.

Logic

- $\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D})$
- $\texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D}) + \texttt{S1.l}(\texttt{D})$
- $$\begin{array}{l} \texttt{contains\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=} \\ \quad \forall \texttt{D} \in \texttt{Dims} \\ \quad \bigwedge \left(\begin{array}{l} \texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) < \\ \texttt{origin}(\texttt{O2},\texttt{S2},\texttt{D}) \end{array}, \begin{array}{l} \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) < \\ \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \end{array} \right) \end{array}$$
- $$\begin{array}{l} \texttt{contains\_objects}(\texttt{Dims},\texttt{O1},\texttt{O2}) \stackrel{\text{def}}{=} \\ \quad \forall \texttt{S1} \in \texttt{sboxes}([\texttt{O1.sid}]) \\ \quad\; \exists \texttt{S2} \in \texttt{sboxes}\left(\left[\ \texttt{O2.sid}\ \right]\right) \\ \quad\; \texttt{contains\_sboxes}\left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{S1}, \\ \texttt{O2}, \\ \texttt{S2} \end{array} \right) \end{array}$$
- $$\begin{array}{l} \texttt{all\_contains}(\texttt{Dims},\texttt{OIDS}) \stackrel{\text{def}}{=} \\ \quad \forall \texttt{O1} \in \texttt{objects}(\texttt{OIDS}) \\ \quad\; \forall \texttt{O2} \in \texttt{objects}(\texttt{OIDS}) \\ \quad\quad \begin{array}{l} \texttt{O1.oid} < \\ \texttt{O2.oid} \end{array} \Rightarrow \\ \quad\quad \texttt{contains\_objects}\left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{O2} \end{array} \right) \end{array}$$
- $\texttt{all\_contains}(\texttt{DIMENSIONS},\texttt{OIDS})$

5.92 CORRESPONDENCE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

Derived from SORT_PERMUTATION by removing the sorting condition.

Constraint

CORRESPONDENCE(FROM, PERMUTATION, TO)

Arguments

FROM	:	collection(from−dvar)
PERMUTATION	:	collection(var−dvar)
TO	:	collection(tvar−dvar)

Restrictions

$|\texttt{PERMUTATION}| = |\texttt{FROM}|$
$|\texttt{PERMUTATION}| = |\texttt{TO}|$
$\texttt{PERMUTATION.var} \geq 1$
$\texttt{PERMUTATION.var} \leq |\texttt{PERMUTATION}|$
ALLDIFFERENT(PERMUTATION)
required(FROM, from)
required(PERMUTATION, var)
required(TO, tvar)

Purpose

The variables of collection FROM correspond to the variables of collection TO according to the permutation PERMUTATION (i.e., $\texttt{FROM}[i].\texttt{from} = \texttt{TO}[\texttt{PERMUTATION}[i].\texttt{var}].\texttt{tvar}$).

Example

$$(\langle 1, 9, 1, 5, 2, 1\rangle, \langle 6, 1, 3, 5, 4, 2\rangle, \langle 9, 1, 1, 2, 5, 1\rangle)$$

As illustrated by Figure 5.224, the CORRESPONDENCE constraint holds since:

- The first item $\texttt{FROM}[1].\texttt{from} = 1$ of collection FROM corresponds to the $\texttt{PERMUTATION}[1].\texttt{var} = 6^{th}$ item of collection TO.
- The second item $\texttt{FROM}[2].\texttt{from} = 9$ of collection FROM corresponds to the $\texttt{PERMUTATION}[2].\texttt{var} = 1^{th}$ item of collection TO.
- The third item $\texttt{FROM}[3].\texttt{from} = 1$ of collection FROM corresponds to the $\texttt{PERMUTATION}[3].\texttt{var} = 3^{th}$ item of collection TO.
- The fourth item $\texttt{FROM}[4].\texttt{from} = 5$ of collection FROM corresponds to the $\texttt{PERMUTATION}[4].\texttt{var} = 5^{th}$ item of collection TO.
- The fifth item $\texttt{FROM}[5].\texttt{from} = 2$ of collection FROM corresponds to the $\texttt{PERMUTATION}[5].\texttt{var} = 4^{th}$ item of collection TO.
- The sixth item $\texttt{FROM}[6].\texttt{from} = 1$ of collection FROM corresponds to the $\texttt{PERMUTATION}[6].\texttt{var} = 2^{th}$ item of collection TO.

Typical

$|\texttt{FROM}| > 1$
$\texttt{range}(\texttt{FROM.from}) > 1$

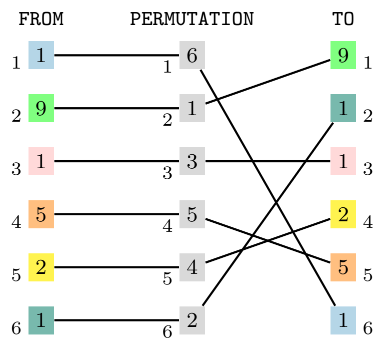


Figure 5.224: Illustration of the correspondence between the items of the FROM and the TO collections according to the permutation defined by the items of the PERMUTATION collection of the **Example** slot

Symmetry

All occurrences of two distinct values in FROM.from or TO.tvar can be swapped; all occurrences of a value in FROM.from or TO.tvar can be renamed to any unused value.

Remark

Similar to the SAME constraint except that we also provide the permutation that allows one to go from the items of collection FROM to the items of collection TO.

Algorithm

An arc-consistency filtering algorithm for the CORRESPONDENCE constraint is described in [138, 139]. The algorithm is based on the following ideas:

- First, one can map solutions to the CORRESPONDENCE constraint to perfect matchings in a bipartite graph derived from the domain of the variables of the constraint in the following way: to each variable of the FROM collection there is a *from* vertex; similarly, to each variable of the TO collection there is a *to* vertex; finally, there is an edge between the i^{th} from vertex and the j^{th} to vertex if and only if the corresponding domains intersect and if j belongs to the domain of the i^{th} permutation variable.
- Second, Dulmage-Mendelsohn decomposition [157] is used to characterise all edges that do not belong to any perfect matching, and therefore prune the corresponding variables.

See also

implied by: SORT_PERMUTATION.

specialisation: SAME *(*PERMUTATION *parameter removed)*.

Keywords

characteristic of a constraint: derived collection.

combinatorial object: permutation.

constraint arguments: constraint between three collections of variables.

filtering: bipartite matching.

final graph structure: acyclic, bipartite, no loop.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{FROM\_PERMUTATION}-\texttt{collection}(\texttt{from}-\texttt{dvar},\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{from}-\texttt{FROM.from},\texttt{var}-\texttt{PERMUTATION.var})]\end{array}\right)$$

Arc input(s) FROM_PERMUTATION TO

Arc generator $PRODUCT \mapsto$ collection(from_permutation, to)

Arc arity 2

Arc constraint(s)

- from_permutation.from = to.tvar
- from_permutation.var = to.key

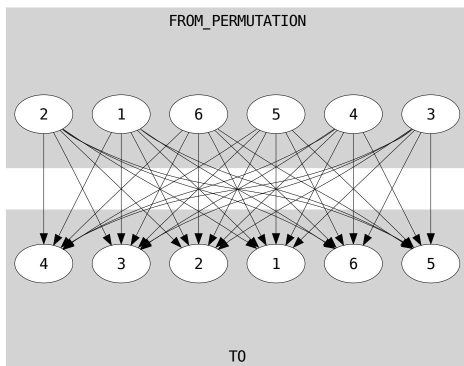
Graph property(ies) **NARC** = |PERMUTATION|

Graph class

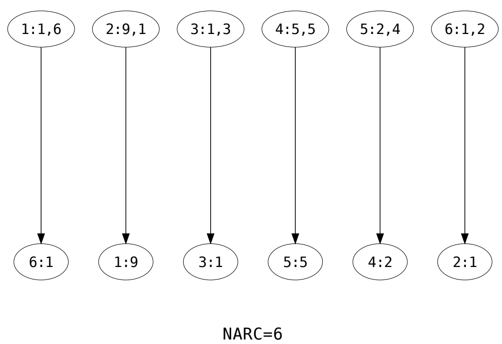
- ACYCLIC
- BIPARTITE
- NO_LOOP

Graph model Parts (A) and (B) of Figure 5.225 respectively show the initial and final graph associated with the **Example** slot. In both graphs the source vertices correspond to the derived collection FROM_PERMUTATION, while the sink vertices correspond to the collection TO. Since the final graph contains exactly |PERMUTATION| arcs the CORRESPONDENCE constraint holds. As we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

Signature Because of the second condition from_permutation.var = to.key of the arc constraint and since both, the var attributes of the collection FROM_PERMUTATION and the key attributes of the collection TO are all-distinct, the final graph contains at most |PERMUTATION| arcs. Therefore we can rewrite the graph property **NARC** = |PERMUTATION| to **NARC** $\geq$ |PERMUTATION|. This leads to simplify $\overline{\textbf{NARC}}$ to $\overline{\textbf{NARC}}$.



(A)



(B)

Figure 5.225: Initial and final graph of the CORRESPONDENCE constraint

5.93 COUNT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [108]

Constraint COUNT(VALUE, VARIABLES, RELOP, LIMIT)

Synonyms OCCURENCEMAX, OCCURENCEMIN, OCCURRENCE.

Arguments

VALUE	:	int
VARIABLES	:	collection(var−dvar)
RELOP	:	atom
LIMIT	:	dvar

Restrictions

required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Let N be the number of variables of the VARIABLES collection assigned to value VALUE; Enforce condition N RELOP LIMIT to hold.

Example

$(5, \langle 4, 5, 5, 4, 5\rangle, \geq, 2)$

The COUNT constraint holds since value VALUE $=$ 5 occurs 3 times within the items of the collection VARIABLES $= \langle 4, 5, 5, 4, 5\rangle$, which is greater than or equal to (RELOP is set to $\geq$) LIMIT $= 2$.

Typical

|VARIABLES| > 1
range(VARIABLES.var) > 1
RELOP $\in [=, <, \geq, >, \leq]$
LIMIT > 0
LIMIT $<$ |VARIABLES|

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that is different from VALUE can be replaced by any other value that is also different from VALUE.

Arg. properties

- Contractible wrt. VARIABLES when RELOP $\in [<, \leq]$.
- Extensible wrt. VARIABLES when RELOP $\in [\geq, >]$.
- Aggregate: VALUE(id), VARIABLES(union), RELOP(id), LIMIT(+) when RELOP $\in [<, \leq, \geq, >]$.

Remark

Similar to the AMONG constraint. Both, in **JaCoP** (http://www.jacop.eu/) and in **MiniZinc** (http://www.minizinc.org/) RELOP is implicitly set to $=$.

Reformulation The COUNT(VALUE, VARIABLES, RELOP , LIMIT) constraint can be expressed in term of the conjunction AMONG(N, VARIABLES, ⟨VALUE⟩) $\wedge$ N RELOP LIMIT.

Systems OCCURENCE in **Choco**, COUNT in **Gecode**, COUNT in **JaCoP**, COUNT_EQ in **MiniZinc**, COUNT_GEQ in **MiniZinc**, COUNT_GT in **MiniZinc**, COUNT_LEQ in **MiniZinc**, COUNT_LT in **MiniZinc**, COUNT_NEQ in **MiniZinc**, COUNT in **SICStus**.

See also **assignment dimension added:** ASSIGN_AND_COUNTS *(*variable=VALUE *replaced by* variable $\in$ VALUES *and assignment dimension introduced)*.

common keyword: AMONG *(value constraint,counting constraint)*, ARITH *(value constraint)*, COMPARE_AND_COUNT *(counting constraint)*, GLOBAL_CARDINALITY, MAX_NVALUE, MIN_NVALUE *(value constraint,counting constraint)*, NVALUE *(counting constraint)*.

generalisation: COUNTS *(*variable=VALUE *replaced by* variable $\in$ VALUES*)*.

related: ROOTS.

used in reformulation: AMONG.

Keywords **characteristic of a constraint:** automaton, automaton with counters.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

filtering: arc-consistency.

Arc input(s)	VARIABLES
Arc generator	$SELF \mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUE
Graph property(ies)	**NARC** RELOP LIMIT

Graph model Parts (A) and (B) of Figure 5.226 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

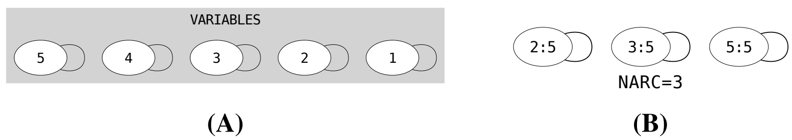


Figure 5.226: Initial and final graph of the COUNT constraint

Automaton Figure 5.227 depicts the automaton associated with the COUNT constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i = \mathtt{VALUE} \Leftrightarrow S_i$.

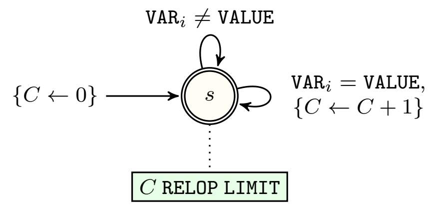


Figure 5.227: Automaton of the COUNT constraint

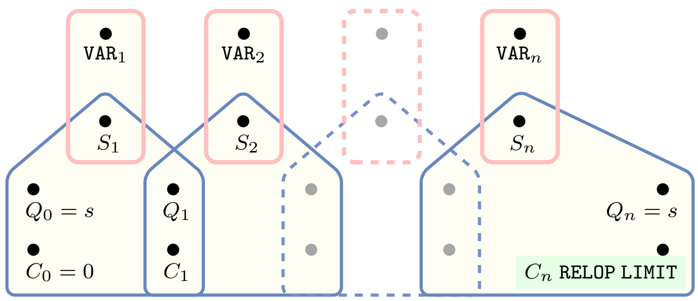


Figure 5.228: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the COUNT constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.94 COUNTS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from COUNT.

Constraint COUNTS(VALUES, VARIABLES, RELOP, LIMIT)

Arguments

VALUES	:	collection(val−int)
VARIABLES	:	collection(var−dvar)
RELOP	:	atom
LIMIT	:	dvar

Restrictions

required(VALUES, val)
distinct(VALUES, val)
required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Let N be the number of variables of the VARIABLES collection assigned to a value of the VALUES collection. Enforce condition N RELOP LIMIT to hold.

Example

$$(\langle 1, 3, 4, 9\rangle, \langle 4, 5, 5, 4, 1, 5\rangle, =, 3)$$

Values 1, 3, 4 and 9 of the VALUES collection are assigned to 3 items of the VARIABLES $= \langle 4, 5, 5, 4, 1, 5\rangle$ collection. The COUNTS constraint holds since this number is in fact equal (RELOP is set to $=$) to the last argument of the COUNTS constraint.

Typical

$|\text{VALUES}| > 1$
$|\text{VARIABLES}| > 1$
range(VARIABLES.var) > 1
$|\text{VARIABLES}| > |\text{VALUES}|$
RELOP $\in [=, <, \geq, >, \leq]$
LIMIT > 0
LIMIT $< |\text{VARIABLES}|$

Symmetries

- Items of VALUES are permutable.
- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).

Arg. properties

- Contractible wrt. VARIABLES when RELOP $\in [<, \leq]$.
- Extensible wrt. VARIABLES when RELOP $\in [\geq, >]$.
- Aggregate: VALUES(sunion), VARIABLES(union), RELOP(id), LIMIT(+) when RELOP $\in [<, \leq, \geq, >]$.

Usage Used in the **Constraint(s) on sets** slot for defining some constraints like ASSIGN_AND_COUNTS.

Reformulation The COUNT(VALUES, VARIABLES, RELOP , LIMIT) constraint can be expressed in term of the conjunction AMONG(N, VARIABLES, VALUES) $\wedge$ N RELOP LIMIT.

Systems COUNT in **Gecode**.

Used in ASSIGN_AND_COUNTS.

See also **assignment dimension added:** ASSIGN_AND_COUNTS *(assignment dimension introduced)*.

common keyword: AMONG *(value constraint,counting constraint)*.

specialisation: COUNT *(*variable $\in$ VALUES *replaced by* variable=VALUE*)*.

Keywords **characteristic of a constraint:** automaton, automaton with counters.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

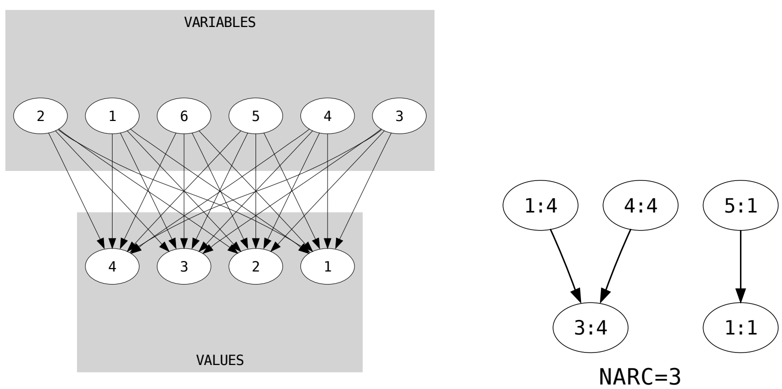
filtering: arc-consistency.

final graph structure: acyclic, bipartite, no loop.

Arc input(s)	VARIABLES VALUES
Arc generator	*PRODUCT* ↦ collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	**NARC** RELOP LIMIT
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Because of the arc constraint variables.var = values.val and since each domain variable can take at most one value, **NARC** is the number of variables taking a value in the VALUES collection.

Parts (A) and (B) of Figure 5.229 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) (B)

Figure 5.229: Initial and final graph of the COUNTS constraint

Automaton Figure 5.230 depicts the automaton associated with the COUNTS constraint. To each variable $\texttt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\texttt{VAR}_i$ and S_i: $\texttt{VAR}_i \in \texttt{VALUES} \Leftrightarrow S_i$.

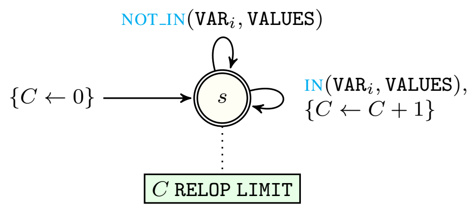


Figure 5.230: Automaton of the COUNTS constraint

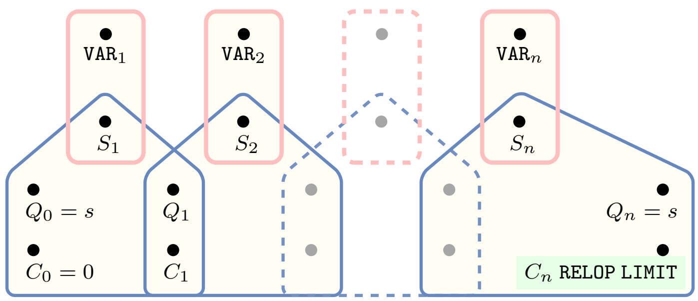


Figure 5.231: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the COUNTS constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.95 COVEREDBY_SBOXES

▶ ▷ ◁ ◀ DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [349]

Constraint COVEREDBY_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonym COVEREDBY.

Types

VARIABLES : collection(v−dvar)
INTEGERS : collection(v−int)
POSITIVES : collection(v−int)

Arguments

K : int
DIMS : sint
OBJECTS : collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES : collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

$|\texttt{VARIABLES}| \geq 1$
$|\texttt{INTEGERS}| \geq 1$
$|\texttt{POSITIVES}| \geq 1$
required(VARIABLES, v)
$|\texttt{VARIABLES}| = \texttt{K}$
required(INTEGERS, v)
$|\texttt{INTEGERS}| = \texttt{K}$
required(POSITIVES, v)
$|\texttt{POSITIVES}| = \texttt{K}$
$\texttt{POSITIVES.v} > 0$
$\texttt{K} > 0$
$\texttt{DIMS} \geq 0$
$\texttt{DIMS} < \texttt{K}$
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
$\texttt{OBJECTS.oid} \geq 1$
$\texttt{OBJECTS.oid} \leq |\texttt{OBJECTS}|$
$\texttt{OBJECTS.sid} \geq 1$
$\texttt{OBJECTS.sid} \leq |\texttt{SBOXES}|$
required(SBOXES, [sid, t, l])
$|\texttt{SBOXES}| \geq 1$
$\texttt{SBOXES.sid} \geq 1$
$\texttt{SBOXES.sid} \leq |\texttt{SBOXES}|$
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i < j$, O_i is covered by O_j with respect to a set of dimensions depicted by DIMS. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a K-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id sid, shift offset t, and sizes l. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier oid, shape id sid and origin x.
An object O_i *is covered by* an object O_j with respect to a set of dimensions depicted by DIMS if and only if, for all shifted box s_i of O_i, there exists a shifted box s_j of O_j such that:

- For all dimensions $d \in$ DIMS, (1) the start of s_j in dimension d is less than or equal to the start of s_i in dimension d, and (2) the end of s_i in dimension d is less than or equal to the end of s_j in dimension d.
- There exists a dimension d where, (1) the start of s_j in dimension d coincide with the start of s_i in dimension d, or (2) the end of s_j in dimension d coincide with the end of s_i in dimension d.

Example

$$\left(\begin{array}{l} 2, \{0,1\}, \\ \left\langle\begin{array}{lll} \texttt{oid}-1 & \texttt{sid}-4 & \texttt{x}-\langle 2,3\rangle, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 2,2\rangle, \\ \texttt{oid}-3 & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,3\rangle, \\ \texttt{sid}-1 & \texttt{t}-\langle 3,0\rangle & \texttt{l}-\langle 2,2\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,2\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 2,0\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,2\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-4 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle \end{array}\right\rangle \end{array}\right)$$

Figure 5.232 shows the objects of the example. Since O_1 is covered by both O_2 and O_3, and since O_2 is covered by O_3, the COVEREDBY_SBOXES constraint holds.

Typical

$|\texttt{OBJECTS}| > 1$

Symmetries

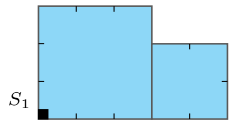
- Items of SBOXES are permutable.
- Items of OBJECTS.x, SBOXES.t and SBOXES.l are permutable (*same permutation used*).

Remark

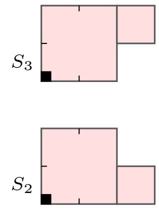
One of the eight relations of the *Region Connection Calculus* [349]. The constraint COVEREDBY_SBOXES is a restriction of the original relation since it requires that each shifted box of an object is covered by one shifted box of the other object.

See also

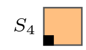
common keyword: CONTAINS_SBOXES, COVERS_SBOXES, DISJOINT_SBOXES, EQUAL_SBOXES, INSIDE_SBOXES, MEET_SBOXES *(rcc8)*, NON_OVERLAP_SBOXES *(geometrical constraint,logic)*, OVERLAP_SBOXES *(rcc8)*.



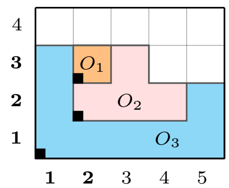
(A) Shape of the third object



(B) Shapes of the second object



(C) Shape of the first object



OBJECTS

O_1:	oid $-$ 1	sid $-$ 4	x $- \langle 2, 3\rangle$
O_2:	oid $-$ 2	sid $-$ 2	x $- \langle 2, 2\rangle$
O_3:	oid $-$ 3	sid $-$ 1	x $- \langle 1, 1\rangle$

(D) Three objects O_1, O_2, O_3, where O_1 is covered by both O_2 and O_3 and where O_2 is covered by O_3

Figure 5.232: (D) the three objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_4, S_2, S_1; (A), (B), (C) shapes S_1, S_2, S_3 and S_4 are respectively made up from 2, 2, 2 and 1 single shifted box.

Keywords

constraint type: logic.

geometry: geometrical constraint, rcc8.

miscellaneous: obscure.

Logic

- $\mathtt{origin}(\mathtt{O1},\mathtt{S1},\mathtt{D}) \stackrel{\text{def}}{=} \mathtt{O1.x}(\mathtt{D}) + \mathtt{S1.t}(\mathtt{D})$
- $\mathtt{end}(\mathtt{O1},\mathtt{S1},\mathtt{D}) \stackrel{\text{def}}{=} \mathtt{O1.x}(\mathtt{D}) + \mathtt{S1.t}(\mathtt{D}) + \mathtt{S1.l}(\mathtt{D})$
- $\mathtt{coveredby\_sboxes}(\mathtt{Dims},\mathtt{O1},\mathtt{S1},\mathtt{O2},\mathtt{S2}) \stackrel{\text{def}}{=}$

$$\bigwedge\left(\begin{array}{l}\forall \mathtt{D} \in \mathtt{Dims} \\ \bigwedge\left(\begin{array}{l}\mathtt{origin}\left(\begin{array}{l}\mathtt{O2},\\ \mathtt{S2},\\ \mathtt{D}\end{array}\right) \leq \\ \mathtt{origin}\left(\begin{array}{l}\mathtt{O1},\\ \mathtt{S1},\\ \mathtt{D}\end{array}\right)\end{array}, \begin{array}{l}\mathtt{end}(\mathtt{O1},\mathtt{S1},\mathtt{D}) \leq \\ \mathtt{end}(\mathtt{O2},\mathtt{S2},\mathtt{D})\end{array}\right), \\ \exists \mathtt{D} \in \mathtt{Dims} \\ \bigvee\left(\begin{array}{l}\mathtt{origin}\left(\begin{array}{l}\mathtt{O2},\\ \mathtt{S2},\\ \mathtt{D}\end{array}\right) = \\ \mathtt{origin}\left(\begin{array}{l}\mathtt{O1},\\ \mathtt{S1},\\ \mathtt{D}\end{array}\right)\end{array}, \begin{array}{l}\mathtt{end}(\mathtt{O1},\mathtt{S1},\mathtt{D}) = \\ \mathtt{end}(\mathtt{O2},\mathtt{S2},\mathtt{D})\end{array}\right)\end{array}\right)$$

- $\mathtt{coveredby\_objects}(\mathtt{Dims},\mathtt{O1},\mathtt{O2}) \stackrel{\text{def}}{=}$
  $\forall \mathtt{S1} \in \mathtt{sboxes}([\mathtt{O1.sid}])$
  $\exists \mathtt{S2} \in \mathtt{sboxes}\left(\left[\ \mathtt{O2.sid}\ \right]\right)$
  $\mathtt{coveredby\_sboxes}\left(\begin{array}{l}\mathtt{Dims},\\ \mathtt{O1},\\ \mathtt{S1},\\ \mathtt{O2},\\ \mathtt{S2}\end{array}\right)$
- $\mathtt{all\_coveredby}(\mathtt{Dims},\mathtt{OIDS}) \stackrel{\text{def}}{=}$
  $\forall \mathtt{O1} \in \mathtt{objects}(\mathtt{OIDS})$
  $\forall \mathtt{O2} \in \mathtt{objects}(\mathtt{OIDS})$
  $\begin{array}{l}\mathtt{O1.oid} < \\ \mathtt{O2.oid}\end{array} \Rightarrow$
  $\mathtt{coveredby\_objects}\left(\begin{array}{l}\mathtt{Dims},\\ \mathtt{O1},\\ \mathtt{O2}\end{array}\right)$
- $\mathtt{all\_coveredby}(\mathtt{DIMENSIONS},\mathtt{OIDS})$

5.96 COVERS_SBOXES

▶ ▷ ◁ ◀ DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [349]

Constraint COVERS_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonym COVERS.

Types

VARIABLES : collection(v−dvar)
INTEGERS : collection(v−int)
POSITIVES : collection(v−int)

Arguments

K : int
DIMS : sint
OBJECTS : collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES : collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

$|$VARIABLES$| \geq 1$
$|$INTEGERS$| \geq 1$
$|$POSITIVES$| \geq 1$
required(VARIABLES, v)
$|$VARIABLES$| =$ K
required(INTEGERS, v)
$|$INTEGERS$| =$ K
required(POSITIVES, v)
$|$POSITIVES$| =$ K
POSITIVES.v > 0
K > 0
DIMS ≥ 0
DIMS $<$ K
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
OBJECTS.oid ≥ 1
OBJECTS.oid $\leq |$OBJECTS$|$
OBJECTS.sid ≥ 1
OBJECTS.sid $\leq |$SBOXES$|$
$|$SBOXES$| \geq 1$
required(SBOXES, [sid, t, l])
SBOXES.sid ≥ 1
SBOXES.sid $\leq |$SBOXES$|$
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i < j$, O_i covers O_j with respect to a set of dimensions depicted by `DIMS`. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a `K`-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id `sid`, shift offset `t`, and sizes `l`. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier `oid`, shape id `sid` and origin `x`.
An object O_i *covers* an object O_j with respect to a set of dimensions depicted by `DIMS` if and only if, for all shifted box s_j of O_j, there exists a shifted box s_i of O_i such that:

- For all dimensions $d \in$ `DIMS`, (1) the start of s_i in dimension d is less than or equal to the start of s_j in dimension d, and (2) the end of s_j in dimension d is less than or equal to the end of s_i in dimension d.
- There exists a dimension d where, (1) the start of s_i in dimension d coincide with the start of s_j in dimension d, or (2) the end of s_i in dimension d coincide with the end of s_j in dimension d.

Example

$$\left(\begin{array}{l} 2, \{0,1\}, \\ \left\langle\begin{array}{lll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 2,2\rangle, \\ \texttt{oid}-3 & \texttt{sid}-4 & \texttt{x}-\langle 2,3\rangle \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,3\rangle, \\ \texttt{sid}-1 & \texttt{t}-\langle 3,0\rangle & \texttt{l}-\langle 2,2\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,2\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 2,0\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,2\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-4 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle \end{array}\right\rangle \end{array}\right)$$

Figure 5.233 shows the objects of the example. Since O_1 covers both O_2 and O_3, and since O_2 covers O_3, the COVERS_SBOXES constraint holds.

Typical

$|$`OBJECTS`$| > 1$

Symmetries

- Items of `SBOXES` are permutable.
- Items of `OBJECTS.x`, `SBOXES.t` and `SBOXES.l` are permutable (*same permutation used*).

Arg. properties

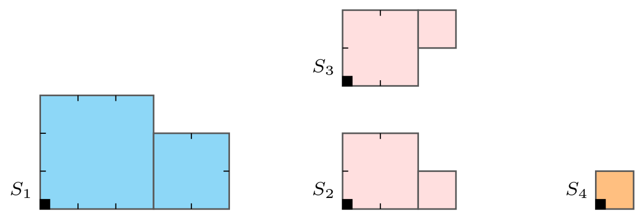
Suffix-contractible wrt. `OBJECTS`.

Remark

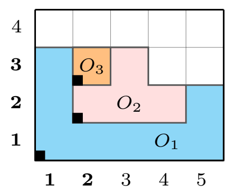
One of the eight relations of the *Region Connection Calculus* [349]. The constraint COVERS_SBOXES is a relaxation of the original relation since it requires that each shifted box of an object is covered by one shifted box of the other object.

See also

common keyword: CONTAINS_SBOXES, COVEREDBY_SBOXES, DISJOINT_SBOXES, EQUAL_SBOXES, INSIDE_SBOXES, MEET_SBOXES (*rcc8*), NON_OVERLAP_SBOXES (*geometrical constraint*,*logic*), OVERLAP_SBOXES (*rcc8*).



(A) Shape of the first object (B) Shapes of the second object (C) Shape of the third object



OBJECTS

O_1:	oid − 1	sid − 1	x − $\langle 1,1\rangle$
O_2:	oid − 2	sid − 2	x − $\langle 2,2\rangle$
O_3:	oid − 3	sid − 4	x − $\langle 2,3\rangle$

(D) Three objects O_1, O_2, O_3, where O_1 covers both O_2 and O_3 and where O_2 covers O_3

Figure 5.233: (D) the three objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_1, S_2, S_4; (A), (B), (C) shapes S_1, S_2, S_3 and S_4 are respectively made up from 2, 2, 2 and 1 single shifted box.

Keywords

constraint type: logic.

geometry: geometrical constraint, rcc8.

miscellaneous: obscure.

Logic

- $\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D})$
- $\texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D}) + \texttt{S1.l}(\texttt{D})$
- $\texttt{covers\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=}$
$$\bigwedge\left(\begin{array}{l}\forall\texttt{D}\in\texttt{Dims}\\ \bigwedge\left(\begin{array}{ll}\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \leq & ,\\ \texttt{origin}(\texttt{O2},\texttt{S2},\texttt{D}) & \\ \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) \leq & \\ \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) & \end{array}\right),\\ \exists\texttt{D}\in\texttt{Dims}\\ \bigvee\left(\begin{array}{ll}\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) = & ,\\ \texttt{origin}(\texttt{O2},\texttt{S2},\texttt{D}) & \\ \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) = & \\ \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) & \end{array}\right)\end{array}\right)$$
- $\texttt{covers\_objects}(\texttt{Dims},\texttt{O1},\texttt{O2}) \stackrel{\text{def}}{=}$
$$\begin{array}{l}\forall\texttt{S2}\in\texttt{sboxes}([\texttt{O2.sid}])\\ \quad\exists\texttt{S1}\in\texttt{sboxes}\left(\left[\ \texttt{O1.sid}\ \right]\right)\\ \quad\texttt{covers\_sboxes}\left(\begin{array}{l}\texttt{Dims},\\ \texttt{O1},\\ \texttt{S1},\\ \texttt{O2},\\ \texttt{S2}\end{array}\right)\end{array}$$
- $\texttt{all\_covers}(\texttt{Dims},\texttt{OIDS}) \stackrel{\text{def}}{=}$
$$\begin{array}{l}\forall\texttt{O1}\in\texttt{objects}(\texttt{OIDS})\\ \quad\forall\texttt{O2}\in\texttt{objects}(\texttt{OIDS})\\ \qquad\begin{array}{l}\texttt{O1.oid} <\\ \texttt{O2.oid}\end{array}\Rightarrow\\ \qquad\texttt{covers\_objects}\left(\begin{array}{l}\texttt{Dims},\\ \texttt{O1},\\ \texttt{O2}\end{array}\right)\end{array}$$
- $\texttt{all\_covers}(\texttt{DIMENSIONS},\texttt{OIDS})$

5.97 CROSSING

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Inspired by [131].

Constraint CROSSING(NCROSS, SEGMENTS)

Arguments

NCROSS : dvar
SEGMENTS : collection(ox−dvar, oy−dvar, ex−dvar, ey−dvar)

Restrictions

NCROSS ≥ 0
NCROSS $\leq$ (|SEGMENTS| $*$ |SEGMENTS| $-$ |SEGMENTS|)/2
required(SEGMENTS, [ox, oy, ex, ey])

Purpose NCROSS is the number of line segments intersections between the line segments defined by the SEGMENTS collection. Each line segment is defined by the coordinates (ox, oy) and (ex, ey) of its two extremities.

Example

$$\left(3, \left\langle \begin{array}{llll} \texttt{ox}-1 & \texttt{oy}-4 & \texttt{ex}-9 & \texttt{ey}-2, \\ \texttt{ox}-1 & \texttt{oy}-1 & \texttt{ex}-3 & \texttt{ey}-5, \\ \texttt{ox}-3 & \texttt{oy}-2 & \texttt{ex}-7 & \texttt{ey}-4, \\ \texttt{ox}-9 & \texttt{oy}-1 & \texttt{ex}-9 & \texttt{ey}-4 \end{array} \right\rangle \right)$$

Figure 5.234 provides a picture of the example with the corresponding four line segments of the SEGMENTS collection. The CROSSING constraint holds since its first argument NCROSS is set to 3, which is actually the number of line segments intersections.

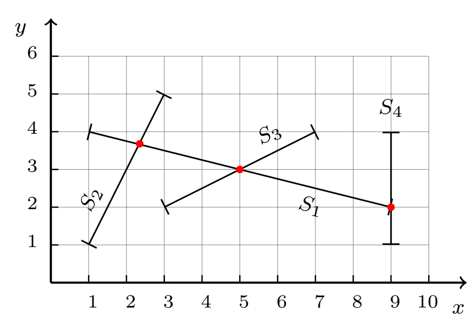


Figure 5.234: Illustration of the **Example** slot: intersection, in red, between the four line segments S_1, S_2, S_3 and S_4 (NCROSS $= 3$)

Typical |SEGMENTS| > 1

Symmetries

- Items of SEGMENTS are permutable.
- Attributes of SEGMENTS are permutable w.r.t. permutation (ox, oy) (ex, ey) (*permutation applied to all items*).
- One and the same constant can be added to the ox and ex attributes of all items of SEGMENTS.
- One and the same constant can be added to the oy and ey attributes of all items of SEGMENTS.

Arg. properties

Functional dependency: NCROSS determined by SEGMENTS.

See also

common keyword: GRAPH_CROSSING, TWO_LAYER_EDGE_CROSSING *(line segments intersection)*.

Keywords

constraint arguments: pure functional dependency.

final graph structure: acyclic, no loop.

geometry: geometrical constraint, line segments intersection.

modelling: functional dependency.

Arc input(s) SEGMENTS

Arc generator $CLIQUE(<) \mapsto$ collection(s1, s2)

Arc arity 2

Arc constraint(s)

- $\texttt{max}(\texttt{s1.ox}, \texttt{s1.ex}) \geq \texttt{min}(\texttt{s2.ox}, \texttt{s2.ex})$
- $\texttt{max}(\texttt{s2.ox}, \texttt{s2.ex}) \geq \texttt{min}(\texttt{s1.ox}, \texttt{s1.ex})$
- $\texttt{max}(\texttt{s1.oy}, \texttt{s1.ey}) \geq \texttt{min}(\texttt{s2.oy}, \texttt{s2.ey})$
- $\texttt{max}(\texttt{s2.oy}, \texttt{s2.ey}) \geq \texttt{min}(\texttt{s1.oy}, \texttt{s1.ey})$
- $\bigvee \left(\begin{array}{l} \begin{array}{l} (\texttt{s2.ox} - \texttt{s1.ex}) * (\texttt{s1.ey} - \texttt{s1.oy}) - \\ (\texttt{s1.ex} - \texttt{s1.ox}) * (\texttt{s2.oy} - \texttt{s1.ey}) \end{array} = 0, \\ \begin{array}{l} (\texttt{s2.ex} - \texttt{s1.ex}) * (\texttt{s2.oy} - \texttt{s1.oy}) - \\ (\texttt{s2.ox} - \texttt{s1.ox}) * (\texttt{s2.ey} - \texttt{s1.ey}) \end{array} = 0, \\ \texttt{sign} \left(\begin{array}{l} (\texttt{s2.ox} - \texttt{s1.ex}) * (\texttt{s1.ey} - \texttt{s1.oy}) - \\ (\texttt{s1.ex} - \texttt{s1.ox}) * (\texttt{s2.oy} - \texttt{s1.ey}) \end{array} \right) \neq \\ \texttt{sign} \left(\begin{array}{l} (\texttt{s2.ex} - \texttt{s1.ex}) * (\texttt{s2.oy} - \texttt{s1.oy}) - \\ (\texttt{s2.ox} - \texttt{s1.ox}) * (\texttt{s2.ey} - \texttt{s1.ey}) \end{array} \right) \end{array} \right)$

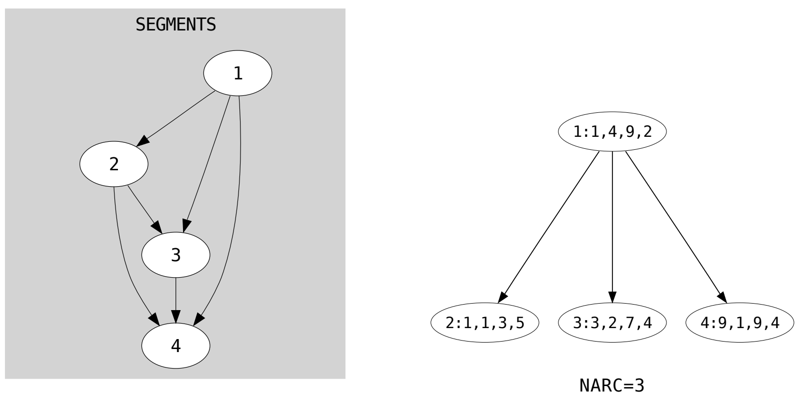
Graph property(ies) **NARC** = NCROSS

Graph class

- ACYCLIC
- NO_LOOP

Graph model Each line segment is described by the x and y coordinates of its two extremities. In the arc generator we use the restriction $<$ in order to generate a single arc for each pair of segments. This is required, since otherwise we would count more than once a given line segments intersection.

Parts (A) and (B) of Figure 5.235 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. An arc constraint expresses the fact the two line segments intersect. It is taken from [131, page 889]. Each arc of the final graph corresponds to a line segments intersection.



(A) **(B)**

Figure 5.235: Initial and final graph of the CROSSING constraint

5.98 CUMULATIVE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [1]

Constraint CUMULATIVE(TASKS, LIMIT)

Synonym CUMULATIVE_MAX.

Arguments

$$\begin{array}{lll} \text{TASKS} & : & \text{collection}\begin{pmatrix} \text{origin}-\text{dvar}, \\ \text{duration}-\text{dvar}, \\ \text{end}-\text{dvar}, \\ \text{height}-\text{dvar} \end{pmatrix} \\ \text{LIMIT} & : & \text{int} \end{array}$$

Restrictions

require_at_least(2, TASKS, [origin, duration, end])
required(TASKS, height)
TASKS.duration ≥ 0
TASKS.origin $\leq$ TASKS.end
TASKS.height ≥ 0
LIMIT ≥ 0

Purpose

Cumulative scheduling constraint or scheduling under resource constraints. Consider a set $\mathcal{T}$ of tasks described by the TASKS collection. The CUMULATIVE constraint enforces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i. It also imposes for each task of $\mathcal{T}$ the constraint origin + duration = end.

Example

$$\left(\left\langle\begin{array}{llll} \text{origin}-1 & \text{duration}-3 & \text{end}-4 & \text{height}-1, \\ \text{origin}-2 & \text{duration}-9 & \text{end}-11 & \text{height}-2, \\ \text{origin}-3 & \text{duration}-10 & \text{end}-13 & \text{height}-1, \\ \text{origin}-6 & \text{duration}-6 & \text{end}-12 & \text{height}-1, \\ \text{origin}-7 & \text{duration}-2 & \text{end}-9 & \text{height}-3 \end{array}\right\rangle, 8\right)$$

Figure 5.236 shows the cumulated profile associated with the example. To each task of the CUMULATIVE constraint, i.e. each line of the example, corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. The CUMULATIVE constraint holds since at each point in time we do not have a cumulated resource consumption strictly greater than the upper limit 8 enforced by the last argument of the CUMULATIVE constraint.

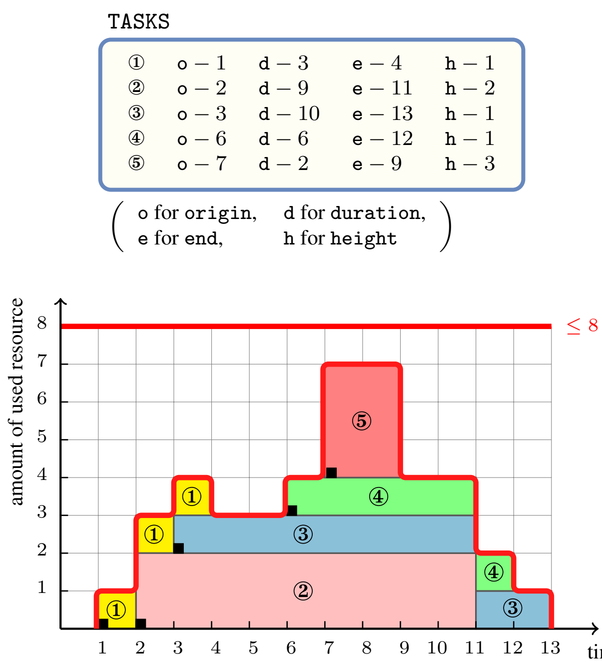


Figure 5.236: Resource consumption profile corresponding to the five tasks of the **Example** slot (note that the vertical position of a task does not really matter but is only used for displaying the contribution of a task to the resource consumption profile)

All solutions

Figure 5.237 gives all solutions to the following non ground instance of the CUMULATIVE constraint:
$\mathtt{O}_1 \in [1,5]$, $\mathtt{D}_1 \in [4,4]$, $\mathtt{E}_1 \in [1,9]$, $\mathtt{H}_1 \in [2,6]$,
$\mathtt{O}_2 \in [2,7]$, $\mathtt{D}_2 \in [6,6]$, $\mathtt{E}_2 \in [1,9]$, $\mathtt{H}_2 \in [3,3]$,
$\mathtt{O}_3 \in [3,6]$, $\mathtt{D}_3 \in [3,6]$, $\mathtt{E}_3 \in [1,9]$, $\mathtt{H}_3 \in [1,2]$,
$\mathtt{O}_4 \in [1,8]$, $\mathtt{D}_4 \in [2,3]$, $\mathtt{E}_4 \in [1,9]$, $\mathtt{H}_4 \in [3,4]$,
CUMULATIVE$(\langle \mathtt{O}_1\ \mathtt{D}_1\ \mathtt{E}_1\ \mathtt{H}_1\ 1,\ \mathtt{O}_2\ \mathtt{D}_2\ \mathtt{E}_2\ \mathtt{H}_2\ 2,\ \mathtt{O}_3\ \mathtt{D}_3\ \mathtt{E}_3\ \mathtt{H}_3\ 3,\ \mathtt{O}_4\ \mathtt{D}_4\ \mathtt{E}_4\ \mathtt{H}_4\ 4\rangle, 5)$.

Typical

$|\mathtt{TASKS}| > 1$
$\mathrm{range}(\mathtt{TASKS.origin}) > 1$
$\mathrm{range}(\mathtt{TASKS.duration}) > 1$
$\mathrm{range}(\mathtt{TASKS.end}) > 1$
$\mathrm{range}(\mathtt{TASKS.height}) > 1$
$\mathtt{TASKS.duration} > 0$
$\mathtt{TASKS.height} > 0$
$\mathtt{LIMIT} < \mathrm{sum}(\mathtt{TASKS.height})$

① (⟨1 4 5 2, 3 6 9 3, 5 3 8 1, 1 2 3 3⟩)
② (⟨1 4 5 2, 3 6 9 3, 5 3 8 2, 1 2 3 3⟩)
③ (⟨1 4 5 2, 3 6 9 3, 5 4 9 1, 1 2 3 3⟩)
④ (⟨1 4 5 2, 3 6 9 3, 5 4 9 2, 1 2 3 3⟩)
⑤ (⟨1 4 5 2, 3 6 9 3, 6 3 9 1, 1 2 3 3⟩)
⑥ (⟨1 4 5 2, 3 6 9 3, 6 3 9 2, 1 2 3 3⟩)
⑦ (⟨2 4 6 2, 3 6 9 3, 6 3 9 1, 1 2 3 3⟩)
⑧ (⟨2 4 6 2, 3 6 9 3, 6 3 9 2, 1 2 3 3⟩)

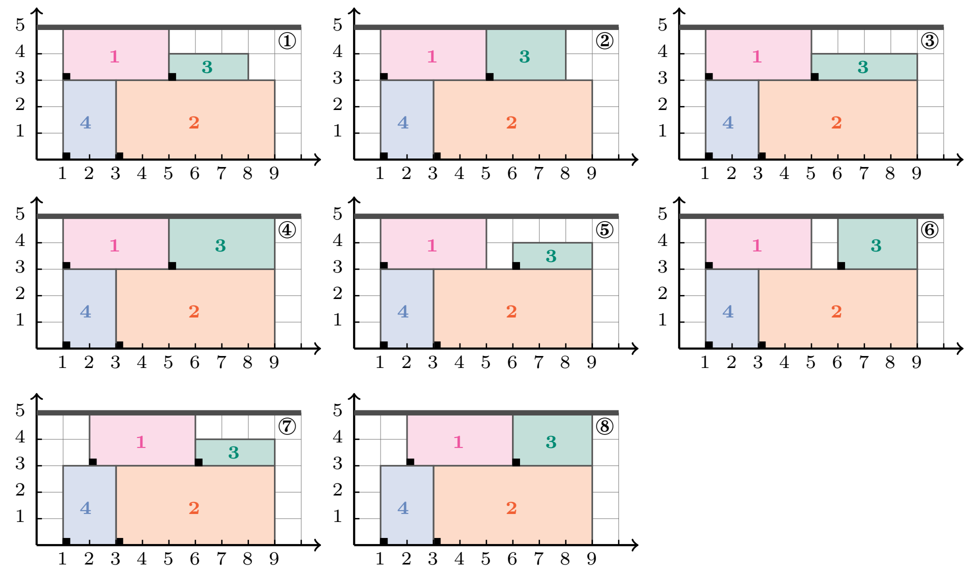


Figure 5.237: All solutions corresponding to the non ground example of the CUMULATIVE constraint of the **All solutions** slot

Symmetries

- Items of `TASKS` are permutable.
- `TASKS.duration` can be decreased to any value ≥ 0.
- `TASKS.height` can be decreased to any value ≥ 0.
- One and the same constant can be added to the `origin` and `end` attributes of all items of `TASKS`.
- `LIMIT` can be increased.

Arg. properties

Contractible wrt. `TASKS`.

Usage

The CUMULATIVE constraint occurs in most resource scheduling problems where one has to deal with renewable and/or non-renewable resources:

- *Renewable resources* typically correspond to machines or persons, and tasks require such resources during all their executions (i.e. a resource starts to be used at the beginning of the task and is released at the end of the task). This means that, once a task has finished its work, the resource it was using is available for other tasks. Tasks are defined by their *origins*, *durations*, *ends* and *resource consumptions* and can not be interrupted. When the duration and resource consumption are not fixed tasks can be defined by their *loads*, i.e. the product of their durations and resource consumptions. To express the dependency between a non-fixed duration/resource consumption of a task with another decision variable (e.g., to state that the duration of a task depends on its start) one can use the ELEMENT constraint where the decision variable corresponds to the index argument of the ELEMENT constraint.
- *Non renewable resources* typically correspond to stock or money, i.e., resources that do not come back when a task finishes to use them. In this context the CUMULATIVE constraint is used for modelling producer-consumer problems, i.e. problems where a first set of tasks produces a non-renewable resource, while a second set of tasks consumes this resource so that a limit on the minimum or the maximum stock at each instant is imposed.

The CUMULATIVE constraint is also used as a necessary condition for non-overlapping rectangles (see the DIFFN constraint).

Remark

In the original CUMULATIVE constraint of **CHIP** the `LIMIT` parameter was a domain variable corresponding to the *maximum peak of the resource consumption profile*. Given a fixed time frame, this variable could be used as a cost in order to directly minimise the maximum resource consumption peak. Fixing this variable is potentially dangerous since it imposes the maximum peak *to be equal to a given target value*.

Some systems like Ilog CP Optimizer also assume that a zero-duration task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is greater than or equal to i. Under this definition, the height of a zero-duration task is also taken into account in the resource consumption profile.

Note that the concept of cumulative is *different* from the concept of rectangles non-overlapping even though, most of the time, each task of a ground solution to a CUMULATIVE constraint is simply drawn as a single rectangle. As illustrated by Figure 5.291, this is in fact not always possible (i.e., some rectangles may need to be broken apart). In fact the CUMULATIVE constraint is only a necessary condition for rectangles non-overlapping (see Figure 5.290 and the corresponding explanation in the **Algorithm** slot of the DIFFN constraint).

In **MiniZinc** (`http://www.minizinc.org/`) the tasks of a CUMULATIVE constraint have no `end` attribute.

Algorithm

The first filtering algorithms were related to the notion of compulsory part of a task [261]. They compute a cumulated resource profile of all the compulsory parts of the tasks and prune the origins of the tasks with respect to this profile in order to not exceed the resource capacity. These methods are sometimes called *time tabling*. Even though these methods are quite local, i.e., a task has a non-empty compulsory part only when the difference between its latest start and its earliest start is strictly less than its duration, it scales well and is therefore widely used. Later on, more global algorithms[6] based on the resource consumption of

[6] Even though these more global algorithms usually can prune more early in the search tree, these algorithms do not catch all deductions derived from the cumulated resource profile of compulsory parts.

the tasks on specific intervals were introduced [163, 112, 276]. A popular variant, called *edge finding*, considers only specific intervals [294]. An efficient implementation of edge finding in $O(kn \log n)$, where k is the number of distinct task heights and n is the number of tasks, based on a specific data structure, so called a *cumulative Φ-tree* [446], is provided in [445]. When the number of distinct task heights k is not small, a usually almost faster implementation in $O(n^2)$ is described in [241]. A $O(n^2 \log n)$ filtering algorithm based on tasks that can not be the earliest (or not be the latest) is described in [386].

Within the context of linear programming, the reference [227] provides a relaxation of the CUMULATIVE constraint.

A necessary condition for the CUMULATIVE constraint is obtained by stating a DISJUNCTIVE constraint on a subset of tasks $\mathcal{T}$ such that, for each pair of tasks of $\mathcal{T}$, the sum of the two corresponding minimum heights is strictly greater than LIMIT. This can be done by applying the following procedure:

- Let h be the smallest minimum height strictly greater than $\lfloor \frac{\texttt{LIMIT}}{2} \rfloor$ of the tasks of the CUMULATIVE constraint. If no such task exists then the procedure is stopped without stating any DISJUNCTIVE constraint.
- Let $\mathcal{T}_h$ denote the set of tasks of the CUMULATIVE constraint for which the minimum height is greater than or equal to h. By construction, the tasks of $\mathcal{T}_h$ cannot overlap. But we can possibly add one more task as shown by the next step.
- When it exists, we can add one task that does not belong to $\mathcal{T}_h$ and such that its minimum height is strictly greater than $\texttt{LIMIT} - h$. Again, by construction, this task cannot overlap all the tasks of $\mathcal{T}_h$.

When the tasks are involved in several CUMULATIVE constraints more sophisticated methods are available for extracting DISJUNCTIVE constraints [18, 17].

In the context where, both the duration and height of all the tasks are fixed, [41] provides two kinds of additional filtering algorithms that are especially useful when the slack σ (i.e., the difference between the available space and the sum of the surfaces of the tasks) is very small:

- The first one introduces bounds for the so called cumulative longest hole problem. Given an integer ϵ that does not exceed the resource limit, and a subset of tasks $\mathcal{T}'$ for which the resource consumption is a most ϵ, the *cumulative longest hole problem* is to find the largest integer $lmax_\sigma^\epsilon(\mathcal{T}')$ such that there is a cumulative placement of maximum height ϵ involving a subset of tasks of $\mathcal{T}'$ where, on one interval $[i, i + lmax_\sigma^\epsilon(\mathcal{T}') - 1]$ of the cumulative profile, the area of the empty space does not exceed σ.
- The second one used dynamic programming for filtering so called *balancing knapsack constraints*. When the slack is 0, such constraints express that the total height of tasks ending at instant i must equal the total height of tasks starting at instant i. Such constraints can be generalised to non-zero slack.

Bound-consistency algorithms are available for the following relaxations of the CUMULATIVE constraint:

- When the durations and the resource consumptions are all equal to 1, one can use the bound-consistency filtering algorithm of the GLOBAL_CARDINALITY_LOW_UP constraint.

- When the durations are all equal, one can use the bound-consistency filtering algorithm of the MULTI_INTER_DISTANCE constraint.

Systems CUMULATIVEMAX in **Choco**, CUMULATIVE in **Gecode**, CUMULATIVE in **JaCoP**, CUMULATIVE in **MiniZinc**, CUMULATIVE in **SICStus**.

See also **assignment dimension added:** COLOURED_CUMULATIVES *(sum of* `task heights` *replaced by number of distinct* `colours`*, assignment dimension added)*, CUMULATIVES *(negative* `heights` *allowed and assignment dimension added)*.

common keyword: CALENDAR *(scheduling constraint)*, COLOURED_CUMULATIVE *(resource constraint, sum of* `task heights` *replaced by number of distinct values)*, COLOURED_CUMULATIVES *(resource constraint)*, CUMULATIVE_CONVEX *(resource constraint,* `task` *defined by a set of* `points`*)*, CUMULATIVE_PRODUCT *(resource constraint, sum of* `task heights` *replaced by product of* `task heights`*)*, CUMULATIVE_WITH_LEVEL_OF_PRIORITY *(resource constraint, a* CUMULATIVE *constraint for each set of* `tasks` *having a priority less than or equal to a given threshold)*.

generalisation: CUMULATIVE_TWO_D *(*`task` *replaced by* `rectangle` *with a* `height`*)*.

implied by: DIFFN *(*CUMULATIVE *is a neccessary condition for each dimension of the* DIFFN *constraint)*.

related: LEX_CHAIN_LESS, LEX_CHAIN_LESSEQ *(lexicographic ordering on the origins of* `tasks`, `rectangles`, *...)*, ORDERED_GLOBAL_CARDINALITY *(controlling the shape of the cumulative profile for breaking symmetry)*.

soft variant: SOFT_CUMULATIVE.

specialisation: ATMOST *(*`task` *replaced by* `variable`*)*, BIN_PACKING *(all* `tasks` *have a* `duration` *of* 1 *and a fixed* `height`*)*, DISJUNCTIVE *(all* `tasks` *have a* `height` *of* 1*)*, GLOBAL_CARDINALITY_LOW_UP *(all* `tasks` *have the same* `duration` *and* `height` *of* 1*)*, MULTI_INTER_DISTANCE *(all* `tasks` *have the same* `duration` *equal to* `DIST` *and the same* `height` *of* 1*)*.

used in graph description: SUM_CTR.

Keywords **characteristic of a constraint:** core, automaton, automaton with array of counters.

complexity: sequencing with release times and deadlines.

constraint type: scheduling constraint, resource constraint, temporal constraint.

filtering: linear programming, dynamic programming, compulsory part, cumulative longest hole problems, Phi-tree, minimum task duration.

modelling: zero-duration task.

problems: producer-consumer.

puzzles: squared squares.

Cond. implications CUMULATIVE$(\mathtt{TASKS}, \mathtt{LIMIT})$
with $\mathtt{TASKS.height} > 0$
implies COLOURED_CUMULATIVE$(\mathtt{TASKS} : \mathtt{TASKS}, \mathtt{LIMIT} : \mathtt{LIMIT})$.

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) NARC = |TASKS|

Arc input(s) TASKS TASKS

Arc generator $PRODUCT \mapsto$ collection(tasks1, tasks2)

Arc arity 2

Arc constraint(s)
- tasks1.duration > 0
- tasks2.origin $\leq$ tasks1.origin
- tasks1.origin $<$ tasks2.end

Graph class
- ACYCLIC
- BIPARTITE
- NO_LOOP

Sets

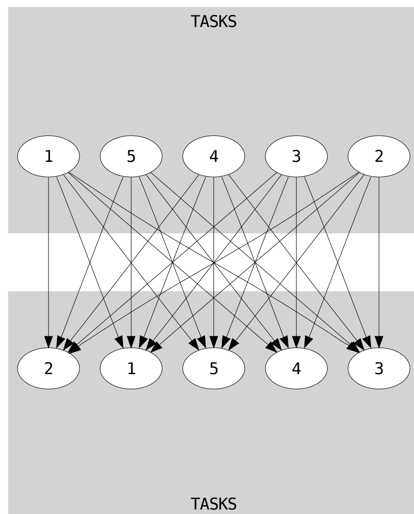
$$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables}-\texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{TASKS.height})]\end{array}\right)\end{array}\right]$$

Constraint(s) on sets SUM_CTR(variables, $\leq$, LIMIT)

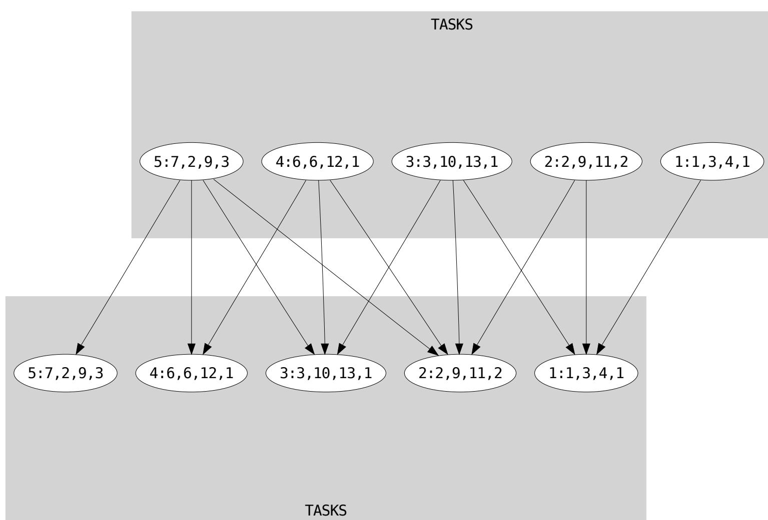
Graph model The first graph constraint forces for each task the link between its origin, its duration and its end. The second graph constraint makes sure, for each time point t corresponding to the start of a task, that the cumulated heights of the tasks that overlap t does not exceed the limit of the resource.

Parts (A) and (B) of Figure 5.238 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks that overlap that time point. The CUMULATIVE constraint holds since for each successor set $\mathcal{S}$ of the final graph the sum of the heights of the tasks in $\mathcal{S}$ does not exceed the limit LIMIT $= 8$.

Signature Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite NARC = |TASKS| to NARC $\geq$ |TASKS|. This leads to simplify $\overline{\text{NARC}}$ to $\overline{\text{NARC}}$.



(A)



(B)

Figure 5.238: Initial and final graph of the CUMULATIVE constraint

Automaton Figure ?? depicts the automaton associated with the CUMULATIVE constraint. To each item of the collection TASKS corresponds a signature variable S_i that is equal to 1.

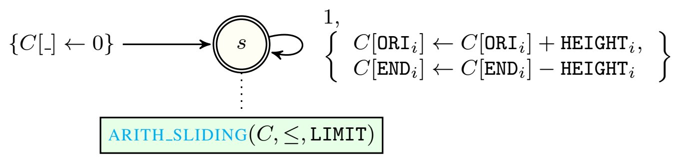


Figure 5.239: Automaton of the CUMULATIVE constraint

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint CUMULATIVE$(\langle 1\ 2\ 3\ 3, 2\ 2\ 4\ 2, 4\ 1\ 5\ 1\rangle, 4)$ hold?

B. Does the constraint CUMULATIVE$(\langle 1\ 2\ 3\ 1, 4\ 1\ 5\ 2\rangle, 1)$ hold?

C. Does the constraint CUMULATIVE$(\langle 1\ 2\ 3\ 0, 1\ 2\ 3\ 4, 4\ 1\ 6\ 1\rangle, 4)$ hold?

[a] Hint: go back to the definition of CUMULATIVE.

EXERCISE 2 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\left\{\begin{array}{l} O_1 \in [1,9],\ O_2 \in [1,9],\ O_3 \in [1,9],\ O_4 \in [1,9], \\ E_1 \in [1,8],\ E_2 \in [1,8],\ E_3 \in [1,8],\ E_4 \in [1,8], \\ \text{CUMULATIVE}(\langle O_1\ 1\ E_1\ 1, O_2\ 2\ E_2\ 2, O_3\ 3\ E_3\ 5, O_4\ 4\ E_4\ 7\rangle, 7). \end{array}\right.$$

[a] Hint: reason first on the two highest tasks and then on the other tasks.

EXERCISE 3 (switching time and resource and breaking symmetry)

Consider a set of 20 tasks for which the duration is given by Table 1, as well as five machines. Each task has to be assigned to one of the five machines in such a way that:

- preemption is not allowed,
- two tasks assigned to a same machine should not overlap.

The goal is to find a schedule that minimises the latest completion time of the tasks.

44	670	949	969	851	573	361	118	309	43
309	929	385	879	704	374	996	631	343	920

Table 1: Durations of the tasks

A. Model the problem with a single CUMULATIVE constraint and show how to use an extra dummy task to handle the fact that the optimal makespan is initially unknown. Make sure to model the machine to which a task is assigned.

B. Provide a search heuristic that takes into account the durations of the tasks.

C. Show how to modify the previous search heuristic in order to break the symmetry related to the fact that all machines are equivalent.

D. Rather than handling symmetry breaking inside the search heuristic use a symmetry-breaking constraint.

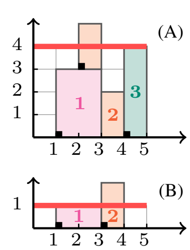
E. Use your favourite solver to find the minimal makespan.

SOLUTION TO EXERCISE 1

A. *No, since the first and second tasks overlap at time point* 2 *and use up to* $3+2$ *resource units which exceeds the resource capacity* 4.

B. *No, since the second task uses* 2 *resource units, while the resource capacity is* 1.

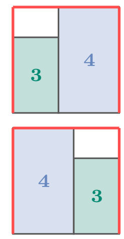
C. *No, since for the third task the origin plus the duration is different from the end* $(4+1\neq 6)$.



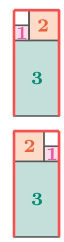
SOLUTION TO EXERCISE 2

(*nested disjunctions*)

1. *Since we have a resource limit of* 7 *the third task (of height* 5*) cannot overlap the fourth task (of height* 7*). Since there is no slack on the time axis (i.e., the difference between the latest end of the third and fourth tasks and their earliest starts is equal to the sum of their durations,* $8-1=3+4$*), this leads to the two configurations shown on the right.*



2. *Since there is no available space on top of the fourth task, the first and second tasks have to be put on top of the third task. Since on top of the third task we only have a capacity of* 2 *the first and second tasks cannot overlap. Since there is no remaining slack on the time axis this leads to the two configurations shown on the right.*



3. *Combining the two previous observations together leads to the four solutions shown below.*

the four solutions

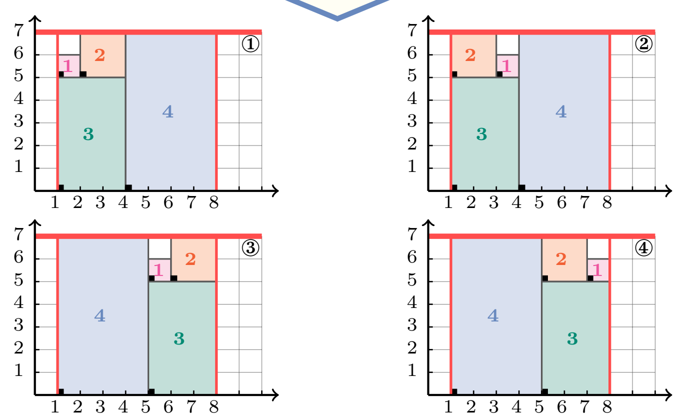
$\langle O_1 D_1 E_1 H_1, O_2 D_2 E_2 H_2, O_3 D_3 E_3 H_3, O_4 D_4 E_4 H_4 \rangle$

① $(\langle 1\ 1\ 2\ 1,\ \ 2\ 2\ 4\ 2,\ \ 1\ 3\ 4\ 5,\ \ 4\ 4\ 8\ 7 \rangle)$

② $(\langle 3\ 1\ 4\ 1,\ \ 1\ 2\ 3\ 2,\ \ 1\ 3\ 4\ 5,\ \ 4\ 4\ 8\ 7 \rangle)$

③ $(\langle 5\ 1\ 6\ 1,\ \ 6\ 2\ 8\ 2,\ \ 5\ 3\ 8\ 5,\ \ 1\ 4\ 5\ 7 \rangle)$

④ $(\langle 7\ 1\ 8\ 1,\ \ 5\ 2\ 7\ 2,\ \ 5\ 3\ 8\ 5,\ \ 1\ 4\ 5\ 7 \rangle)$



SOLUTION TO EXERCISE 3

A. *Since there are no temporal constraints, the problem is modelled as a bin-packing problem where each bin represents a machine.*

(a) Consider a task to schedule and its counterpart within the CUMULATIVE *constraint that models the problem. The origin, the height and the duration of a task in the* CUMULATIVE *constraint respectively correspond to the machine where the task will be assigned, to its duration and to* 1*. We set the* `limit` *argument of the* CUMULATIVE *constraint to an upper bound on the makespan, e.g., the sum* ℓ *of the durations of the tasks to schedule.*

(b) Finally, to link the makespan to the machines where the tasks are assigned, we create a dummy task of variable height h *over all the five bins, i.e., starting at* 1 *and ending at* 6*. Note that the need for creating this dummy task explains why we use a* CUMULATIVE *constraint rather than a* BIN_PACKING *constraint, for which the maximum capacity is usually fixed.*

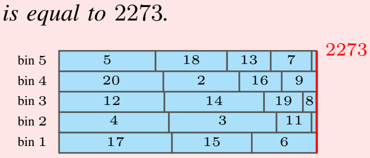
This leads to the following CUMULATIVE *constraint, where w.l.o.g. we omit the end attribute of each task:*
$\forall i \in [1, 20] : m_i \in [1, 5]$,
$\ell = 44 + 670 + \cdots + 920$,
$h \in [0, \ell]$, *makespan* $\in [0, \ell]$, *makespan* $= \ell - h + 1$,
CUMULATIVE$(\langle m_1\ 1\ 44, m_2\ 1\ 670, \ldots, m_{20}\ 1\ 920, 1\ 5\ h\rangle, \ell)$.

B. *To reduce the search space, order the variables* $m_1, m_2, \ldots, m_{20}$ *by decreasing task durations before fixing them by trying successively all values* $1, 2, \ldots, 5$*. Let* $m_{d_1}, m_{d_2}, \ldots, m_{d_{20}}$ *denote the corresponding ordered variables.*

C. *Rather than trying* all *values* $1, 2, \ldots, 5$ *when fixing a variable* m_{d_i}*, first try all values that are already assigned to the previously fixed variables, and then try only one of the not yet used values, i.e. the smallest one. This amounts to, for example, fixing the first variable* m_{d_1} *to the first machine, and upon backtracking not trying all other values, between* 2 *and* 5.

D. *We use the constraint* INT_VALUE_PRECEDE_CHAIN$(\langle 1, 2, \ldots, 5\rangle, \langle m_{d_1}, m_{d_2}, \ldots, m_{d_{20}}\rangle)$ *to state that the first occurrence of value* i *within the sequence of variables* $m_{d_1}, m_{d_2}, \ldots, m_{d_{20}}$ *is located before the first occurrence of value* $i+1$ *in the same sequence (with* $i \in [1, 4]$*).*

E. *By using the previous model together with the symmetry-breaking constraint we find a minimal value of* 2274 *shown below. In other words the resource consumption peak is equal to* 2273.



An optimal solution where the numbers in the rectangles denote the task identifiers

5.99 CUMULATIVE_CONVEX

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from CUMULATIVE

Constraint CUMULATIVE_CONVEX(TASKS, LIMIT)

Type POINTS : collection(var−dvar)

Arguments

TASKS : collection(points − POINTS, height−dvar)
LIMIT : int

Restrictions

required(POINTS, var)
$|\texttt{POINTS}| > 0$
required(TASKS, [points, height])
$\texttt{TASKS.height} \geq 0$
$\texttt{LIMIT} \geq 0$

Purpose

Cumulative scheduling constraint or scheduling under resource constraints. Consider a set $\mathcal{T}$ of tasks described by the TASKS collection where each task is defined by:

- A *set of distinct points* depicting the time interval where the task is actually running: the smallest and largest coordinates of these points respectively give the first and last instant of that time interval.
- A *height* that depicts the resource consumption used by the task from its first instant to its last instant.

The CUMULATIVE_CONVEX constraint enforces that, at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i.

Example

$$\left(\left\langle\begin{array}{ll}\texttt{points}-\langle 2,1,5\rangle & \texttt{height}-1,\\ \texttt{points}-\langle 4,5,7\rangle & \texttt{height}-2,\\ \texttt{points}-\langle 14,13,9,11,10\rangle & \texttt{height}-2\end{array}\right\rangle, 3\right)$$

Figure 5.240 shows the cumulated profile associated with the example. To each set of points defining a task corresponds a rectangle. The height of each rectangle represents the resource consumption of the associated task. The CUMULATIVE_CONVEX constraint holds since at each point in time we do not have a cumulated resource consumption strictly greater than the upper limit 3 enforced by the last argument of the CUMULATIVE_CONVEX constraint.

Typical

$|\texttt{TASKS}| > 1$
$\texttt{TASKS.height} > 0$
$\texttt{LIMIT} < \text{sum}(\texttt{TASKS.height})$

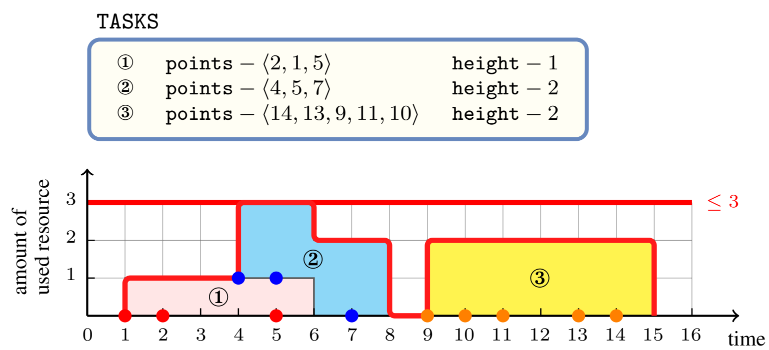


Figure 5.240: Points defining the three tasks of the **Example** slot and corresponding resource consumption profile (note that the vertical position of a task does not really matter but is only used for displaying the contribution of a task to the resource consumption profile)

Symmetries

- Items of TASKS are permutable.
- Items of TASKS.points are permutable.
- TASKS.height can be decreased to any value ≥ 0.
- LIMIT can be increased.

Arg. properties

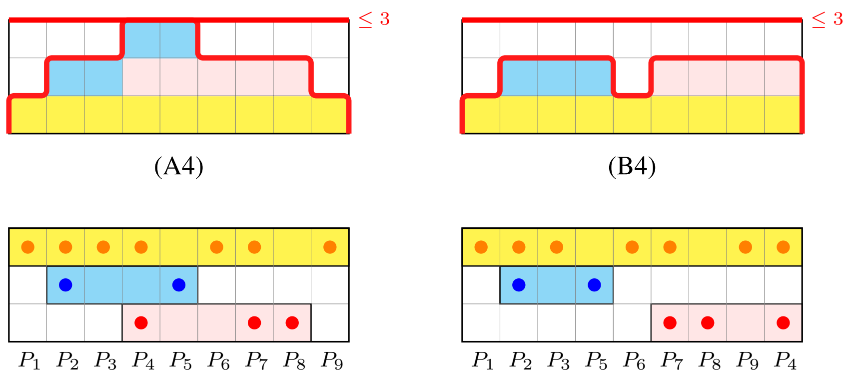
Contractible wrt. TASKS.

Usage

A natural use of the CUMULATIVE_CONVEX constraint corresponds to problems where a task is defined as the convex hull of a set of distinct points $P_1, \ldots, P_n$ that are not initially fixed. Note that, by explicitly introducing a start S and an end E variables, and by using a MINIMUM$(S, \langle \texttt{var}-P_1, \ldots, \texttt{var}-P_n \rangle)$ and a MAXIMUM$(E, \langle \texttt{var}-P_1, \ldots, \texttt{var}-P_n \rangle)$ constraints, one could replace the CUMULATIVE_CONVEX constraint by a CUMULATIVE constraint. However this hinders propagation.

As a concrete example of use of the CUMULATIVE_CONVEX constraint we present a constraint model for a well-known pattern-sequencing problem [177] (also known to be equivalent to the graph path-width [274] problem) that is based on a single CUMULATIVE_CONVEX constraint. The *pattern sequencing problem* can be described as follows: Given a 0-1 matrix in which each column j $(1 \leq j \leq p)$ corresponds to a product required by the customers and each row i $(1 \leq i \leq c)$ corresponds to the order of a particular customer (The entry c_{ij} is equal to 1 if and only if customer i has ordered some quantity of product j.), the objective is to find a permutation of the products such that the maximum number of open orders at any point in the sequence is minimised. Order i is *open* at point k in the production sequence if there is a product required in order i that appears at or before position k in the sequence and also a product that appears at or after position k in the sequence.

Before giving the constraint model, let us first provide an instance of the pattern-sequencing problem. Consider the matrix $\mathcal{M}_1$ depicted by part (A1) of Fig. 5.241. Part (A2) gives its corresponding *cumulated* matrix $\mathcal{M}_2$ obtained by setting to 1 each 0 of $\mathcal{M}_1$ that is both preceded and followed by a 1. Part (A3) depicts the corresponding solution in term of



(A4) (B4)

(A3) (B3)

1	1	1	1	1	1	1	1	1
0	1	1	1	1	0	0	0	0
0	0	0	1	1	1	1	1	0
P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9

(A2)

1	1	1	1	1	1	1	1	1
0	1	1	1	0	0	0	0	0
0	0	0	0	0	1	1	1	0
P_1	P_2	P_3	P_5	P_6	P_7	P_8	P_9	P_4

(B2)

1	1	1	1	0	1	1	0	1
0	1	0	0	1	0	0	0	0
0	0	0	1	0	0	1	1	0
P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9

(A1)

1	1	1	0	1	1	0	1	1
0	1	0	1	0	0	0	0	0
0	0	0	0	0	1	1	0	1
P_1	P_2	P_3	P_5	P_6	P_7	P_8	P_9	P_4

(B1)

Figure 5.241: An input matrix for the pattern sequencing problem (A1), its corresponding cumulated matrix (A2), a view in term of tasks (A3) and the corresponding cumulative profile (A4); a second matrix (B1) where column 4 of (A1) is put at the rightmost position.

the CUMULATIVE_CONVEX constraint: to each row of the matrix $\mathcal{M}_1$ corresponds a task defined as the convex hull of the different 1 located on that row. Finally part (A4) gives the cumulated profile associated with part (A3), namely the number of 1 in each column of $\mathcal{M}_2$. The cost 3 of this solution is equal to the maximum number of 1 in the columns of the *cumulated* matrix $\mathcal{M}_2$. As shown by parts (B1-B4), we can get a lower cost of 2 by pushing the fourth column to the rightmost position.

The idea of the model is to associate to each row (i.e., customer) i of the *cumulated* matrix a *stack task* that starts at the first 1 on row i and ends at the last 1 of row i (i.e., the task corresponds to the convex hull of the different 1 located on row i). Then the cost of a solution is simply the maximum height on the corresponding cumulated profile.

For each column j of the 0-1 matrix initially given there is a variable V_j ranging from 1 to the number of columns p. The value of V_j gives the position of column j in a solution. We put all the stack tasks in a CUMULATIVE_CONVEX constraint, telling that each stack task uses one unit of the resource during all it execution. Since we want to have the same model for different limits on the maximum number of open stacks, and since all variables $V_1, V_2, \ldots, V_p$ have to be distinct, we have an extra dummy task characterised as the convex hull of $V_1, V_2, \ldots, V_p$. This extra dummy task has a height H that has to be maximised. For the matrix depicted by (A1) of Fig. 5.241 we pass to the CUMULATIVE_CONVEX constraint the following collection of tasks:

$$\left\langle \begin{array}{ll} \texttt{points} - \langle P_1, P_2, P_3, P_4, P_6, P_7, P_9 \rangle & \texttt{height} - 1, \\ \texttt{points} - \langle P_2, P_5 \rangle & \texttt{height} - 1, \\ \texttt{points} - \langle P_4, P_7, P_8 \rangle & \texttt{height} - 1, \\ \texttt{points} - \langle P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9 \rangle & \texttt{height} - 0 \end{array} \right\rangle$$

Algorithm

A first natural way to handle the CUMULATIVE_CONVEX constraint is to accumulate the compulsory part [261] of the different tasks in a profile and to prune according to this profile. We give the main ideas for computing the compulsory part of a task and for pruning a task according to the profile of compulsory parts.

Compulsory part of a task Given a task T characterised as the convex hull of a set of distinct points $P_1, P_2, \ldots, P_k$ the compulsory part of T corresponds to the, possibly empty, interval $[s_T, e_T]$ where:

- s_T is the largest value v such that, when all variables $P_1, P_2, \ldots, P_k$ are greater than or equal to v, all variables $P_1, P_2, \ldots, P_k$ can still take distinct values.
- e_T is the smallest value v such that, when all variables $P_1, P_2, \ldots, P_k$ are less than or equal to v, all variables $P_1, P_2, \ldots, P_k$ can still take distinct values.

Pruning according to the profile of compulsory parts Given two instants i and j $(i < j)$ and a task T characterised as the convex hull of a set of distinct points $P_1, P_2, \ldots, P_k$, assume that T cannot overlap i and j since this would lead exceeding `LIMIT`, the second argument of the CUMULATIVE_CONVEX constraint. Furthermore assume that, when all variables $P_1, P_2, \ldots, P_k$ are both greater than i and less than j, all variables $P_1, P_2, \ldots, P_k$ cannot take distinct values. Then all values of $[i+1, j-1]$ can be removed from variables $P_1, P_2, \ldots, P_k$.

See also

common keyword: CUMULATIVE *(resource constraint)*.

used in graph description: ALLDIFFERENT, BETWEEN_MIN_MAX, SUM_CTR.

Keywords

characteristic of a constraint: convex.

constraint type: scheduling constraint, resource constraint, temporal constraint.

filtering: compulsory part.

problems: pattern sequencing.

Derived Collection	$\texttt{col}\left(\begin{array}{l}\texttt{INSTANTS}-\texttt{collection}(\texttt{instant}-\texttt{dvar}),\\ [\texttt{item}(\texttt{instant}-\texttt{TASKS.points.var})]\end{array}\right)$

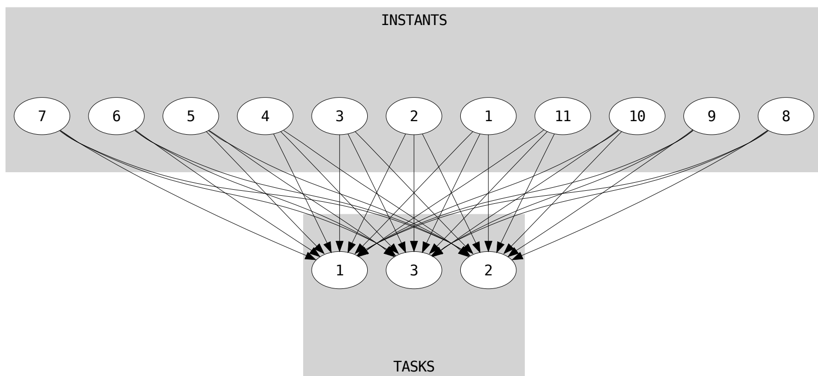
Arc input(s)	TASKS
Arc generator	$SELF \mapsto \texttt{collection}(\texttt{tasks})$
Arc arity	1
Arc constraint(s)	ALLDIFFERENT(tasks.points)
Graph property(ies)	**NARC** $= \vert\texttt{TASKS}\vert$

Arc input(s)	INSTANTS TASKS
Arc generator	$PRODUCT \mapsto \texttt{collection}(\texttt{instants}, \texttt{tasks})$
Arc arity	2
Arc constraint(s)	BETWEEN_MIN_MAX(instants.instant, tasks.points)
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP
Sets	$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables}-\texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{TASKS.height})]\end{array}\right)\end{array}\right]$
Constraint(s) on sets	SUM_CTR(variables, $\leq$, LIMIT)

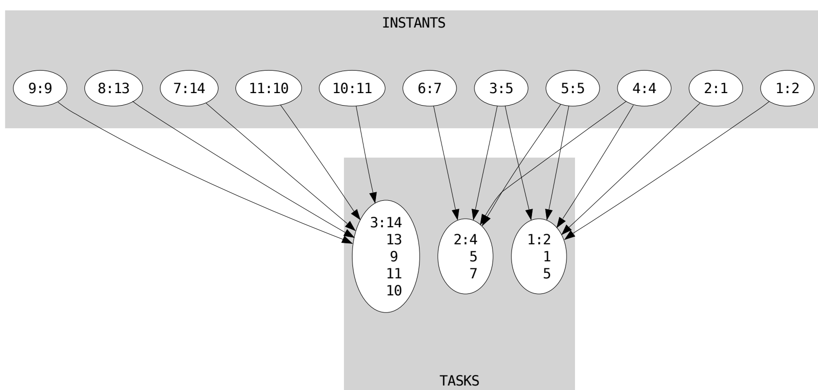
Graph model

The first graph constraint forces for each task that the set of points defining its time interval are all distinct. The second graph constraint makes sure for each time point t, that the cumulated heights of the tasks that overlap t does not exceed the limit of the resource.

Parts (A) and (B) of Figure 5.242 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point corresponding to a point used in the definitions of the different tasks. On the other hand, the successors of a source vertex correspond to those tasks that overlap a given time point. The CUMULATIVE_CONVEX constraint holds since, for each successor set $\mathcal{S}$ of the final graph, the sum of the heights of the tasks in $\mathcal{S}$ does not exceed the limit LIMIT $= 3$.



(A)



(B)

Figure 5.242: Initial and final graph of the CUMULATIVE_CONVEX constraint

5.100 CUMULATIVE_PRODUCT

DESCRIPTION LINKS GRAPH

Origin Derived from CUMULATIVE.

Constraint CUMULATIVE_PRODUCT(TASKS, LIMIT)

Arguments

$$\begin{array}{lll} \texttt{TASKS} & : & \texttt{collection}\left(\begin{array}{l}\texttt{origin}-\texttt{dvar},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar},\\ \texttt{height}-\texttt{dvar}\end{array}\right)\\ \texttt{LIMIT} & : & \texttt{int}\end{array}$$

Restrictions

$\texttt{require\_at\_least}(2, \texttt{TASKS}, [\texttt{origin}, \texttt{duration}, \texttt{end}])$
$\texttt{required}(\texttt{TASKS}, \texttt{height})$
$\texttt{TASKS.duration} \geq 0$
$\texttt{TASKS.origin} \leq \texttt{TASKS.end}$
$\texttt{TASKS.height} \geq 1$
$\texttt{LIMIT} \geq 0$

Purpose

Consider a set $\mathcal{T}$ of tasks described by the TASKS collection. The CUMULATIVE_PRODUCT constraint forces that at each point in time, the product of the heights of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i. It also imposes for each task of $\mathcal{T}$ the constraint $\texttt{origin} + \texttt{duration} = \texttt{end}$.

Example

$$\left(\left\langle\begin{array}{llll}\texttt{origin}-1 & \texttt{duration}-3 & \texttt{end}-4 & \texttt{height}-1,\\ \texttt{origin}-2 & \texttt{duration}-9 & \texttt{end}-11 & \texttt{height}-2,\\ \texttt{origin}-3 & \texttt{duration}-10 & \texttt{end}-13 & \texttt{height}-1,\\ \texttt{origin}-6 & \texttt{duration}-6 & \texttt{end}-12 & \texttt{height}-1,\\ \texttt{origin}-7 & \texttt{duration}-2 & \texttt{end}-9 & \texttt{height}-3\end{array}\right\rangle, 6\right)$$

Figure 5.243 shows the solution associated with the example. To each task of the CUMULATIVE_PRODUCT constraint corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the height of the task. The profile corresponding to the product of the heights of the tasks that overlap a given point is depicted by a thick red line. The CUMULATIVE_PRODUCT constraint holds since at each point in time the product of the heights of the tasks that overlap that point is not strictly greater than the upper limit 6 enforced by the last argument of the CUMULATIVE_PRODUCT constraint.

TASKS

①	o − 1	d − 3	e − 4	h − 1
②	o − 2	d − 9	e − 11	h − 2
③	o − 3	d − 10	e − 13	h − 1
④	o − 6	d − 6	e − 12	h − 1
⑤	o − 7	d − 2	e − 9	h − 3

(o for origin, d for duration, e for end, h for height)

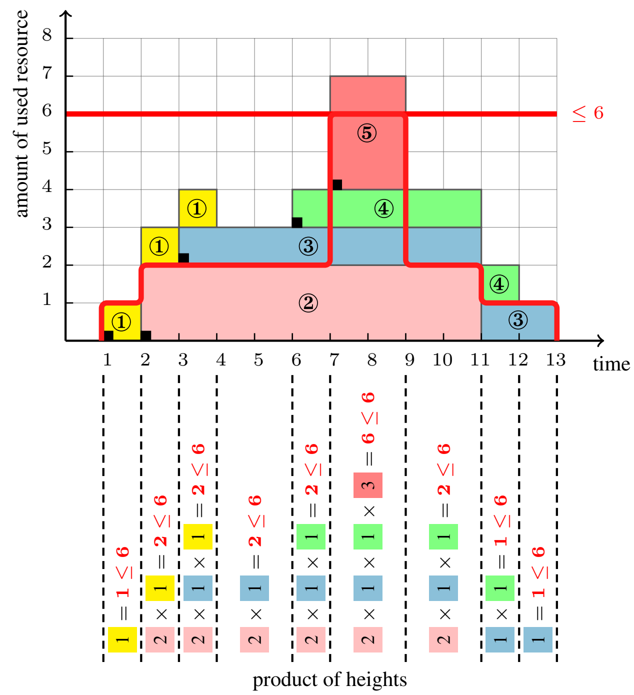


Figure 5.243: Resource consumption profile in red corresponding to the product of the heights of the five tasks of the **Example** slot

Typical

$|\texttt{TASKS}| > 1$
range(TASKS.origin) > 1
range(TASKS.duration) > 1
range(TASKS.end) > 1
range(TASKS.height) > 1
TASKS.duration > 0
LIMIT $<$prod(TASKS.height)

Symmetries

- Items of TASKS are permutable.
- TASKS.height can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.
- LIMIT can be increased.

Arg. properties

Contractible wrt. TASKS.

Reformulation

The CUMULATIVE_PRODUCT constraint can be expressed in term of a set of reified constraints and of |TASKS| constraints of the form $h_1 \cdot h_2 \cdot \dots \cdot h_{|\mathtt{TASKS}|} \leq l$:

1. For each pair of tasks $\mathtt{TASKS}[i], \mathtt{TASKS}[j]$ $(i, j \in [1, |\mathtt{TASKS}|])$ of the TASKS collection we create a variable H_{ij} which is set to the height of task $\mathtt{TASKS}[j]$ if task $\mathtt{TASKS}[j]$ overlaps the origin attribute of task $\mathtt{TASKS}[i]$, and to 1 otherwise:
   - If $i = j$:
     - $H_{ij} = \mathtt{TASKS}[i].\mathtt{height}$.
   - If $i \neq j$:
     - $H_{ij} = \mathtt{TASKS}[j].\mathtt{height} \vee H_{ij} = 1$.
     - $((\mathtt{TASKS}[j].\mathtt{origin} \leq \mathtt{TASKS}[i].\mathtt{origin} \wedge$ $\mathtt{TASKS}[j].\mathtt{end} > \mathtt{TASKS}[i].\mathtt{origin}) \wedge (H_{ij} = \mathtt{TASKS}[j].\mathtt{height})) \vee$ $((\mathtt{TASKS}[j].\mathtt{origin} > \mathtt{TASKS}[i].\mathtt{origin} \vee$ $\mathtt{TASKS}[j].\mathtt{end} \leq \mathtt{TASKS}[i].\mathtt{origin}) \wedge (H_{ij} = 1))$
2. For each task $\mathtt{TASKS}[i]$ $(i \in [1, |\mathtt{TASKS}|])$ we impose a constraint of the form $H_{i1} \cdot H_{i2} \cdot \dots \cdot H_{i|\mathtt{TASKS}|} \leq \mathtt{LIMIT}$.

See also

common keyword: CUMULATIVE *(resource constraint)*.

used in graph description: PRODUCT_CTR.

Keywords

characteristic of a constraint: product.

constraint type: scheduling constraint, resource constraint, temporal constraint.

filtering: compulsory part, minimum task duration.

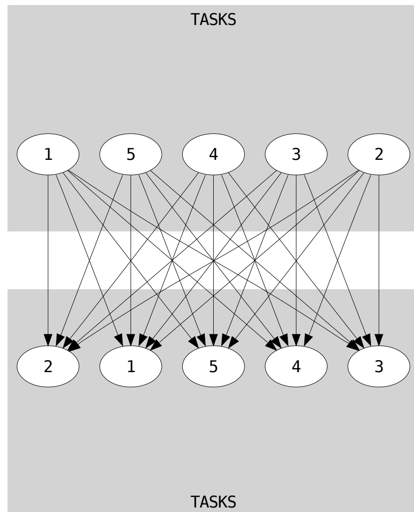
modelling: zero-duration task.

Arc input(s)	TASKS
Arc generator	*SELF* $\mapsto$ collection(tasks)
Arc arity	1
Arc constraint(s)	tasks.origin + tasks.duration = tasks.end
Graph property(ies)	**NARC** = |TASKS|

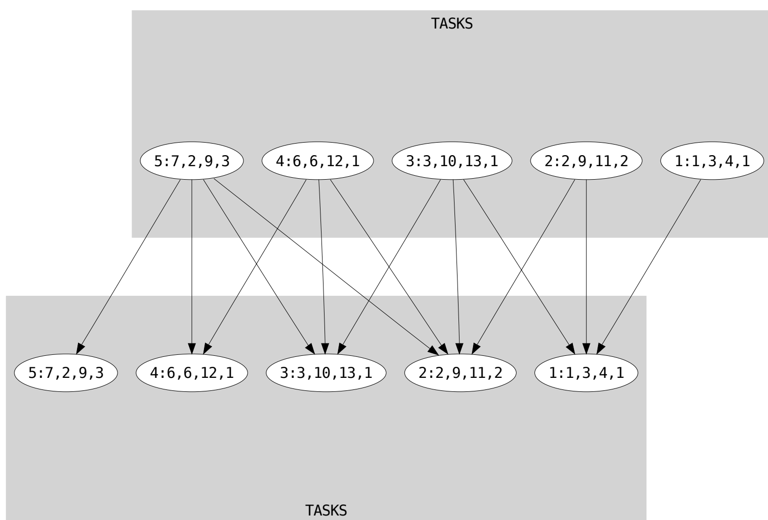
Arc input(s)	TASKS TASKS
Arc generator	*PRODUCT* $\mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.duration > 0 • tasks2.origin ≤ tasks1.origin • tasks1.origin < tasks2.end
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP
Sets	$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{ITEMS.height})]\end{array}\right)\end{array}\right]$
Constraint(s) on sets	PRODUCT_CTR(variables, ≤, LIMIT)

Graph model Parts (A) and (B) of Figure 5.244 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. On the one hand, each source vertex of the final graph can be interpreted as a time point. On the other hand the successors of a source vertex correspond to those tasks that overlap that time point. The CUMULATIVE_PRODUCT constraint holds since for each successor set $\mathcal{S}$ of the final graph the product of the heights of the tasks in $\mathcal{S}$ does not exceed the limit LIMIT $= 6$.

Signature Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite **NARC** = |TASKS| to **NARC** ≥ |TASKS|. This leads to simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.



(A)



(B)

Figure 5.244: Initial and final graph of the CUMULATIVE_PRODUCT constraint

5.101 CUMULATIVE_TWO_D

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin

Inspired by CUMULATIVE and DIFFN.

Constraint

CUMULATIVE_TWO_D(RECTANGLES, LIMIT)

Arguments

$$\begin{array}{lll} \texttt{RECTANGLES} & : & \texttt{collection}\begin{pmatrix} \texttt{start1}-\texttt{dvar}, \\ \texttt{size1}-\texttt{dvar}, \\ \texttt{last1}-\texttt{dvar}, \\ \texttt{start2}-\texttt{dvar}, \\ \texttt{size2}-\texttt{dvar}, \\ \texttt{last2}-\texttt{dvar}, \\ \texttt{height}-\texttt{dvar} \end{pmatrix} \\ \texttt{LIMIT} & : & \texttt{int} \end{array}$$

Restrictions

$\texttt{require\_at\_least}(2, \texttt{RECTANGLES}, [\texttt{start1}, \texttt{size1}, \texttt{last1}])$
$\texttt{require\_at\_least}(2, \texttt{RECTANGLES}, [\texttt{start2}, \texttt{size2}, \texttt{last2}])$
$\texttt{required}(\texttt{RECTANGLES}, \texttt{height})$
$\texttt{RECTANGLES.size1} \geq 0$
$\texttt{RECTANGLES.size2} \geq 0$
$\texttt{RECTANGLES.height} \geq 0$
$\texttt{LIMIT} \geq 0$

Purpose

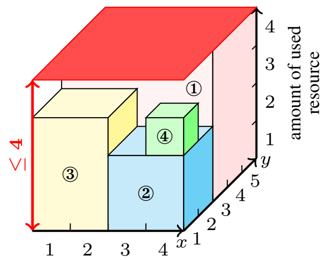
Consider a set $\mathcal{R}$ of rectangles described by the RECTANGLES collection. Enforces that at each point of the plane, the cumulated height of the set of rectangles that overlap that point, does not exceed a given limit.

Example

$$\left(\left\langle\begin{array}{lllllll} \texttt{s1}-1 & \texttt{d1}-4 & \texttt{e1}-4 & \texttt{s2}-3 & \texttt{d2}-3 & \texttt{e2}-5 & \texttt{h}-4, \\ \texttt{s1}-3 & \texttt{d1}-2 & \texttt{e1}-4 & \texttt{s2}-1 & \texttt{d2}-2 & \texttt{e2}-2 & \texttt{h}-2, \\ \texttt{s1}-1 & \texttt{d1}-2 & \texttt{e1}-2 & \texttt{s2}-1 & \texttt{d2}-2 & \texttt{e2}-2 & \texttt{h}-3, \\ \texttt{s1}-4 & \texttt{d1}-1 & \texttt{e1}-4 & \texttt{s2}-1 & \texttt{d2}-1 & \texttt{e2}-1 & \texttt{h}-1 \end{array}\right\rangle, 4\right)$$

$$\begin{pmatrix} \texttt{s1} \text{ for } \texttt{start1}, \ \texttt{d1} \text{ for } \texttt{size1}, \ \texttt{e1} \text{ for } \texttt{last1} \\ \texttt{s2} \text{ for } \texttt{start2}, \ \texttt{d2} \text{ for } \texttt{size2}, \ \texttt{e2} \text{ for } \texttt{last2} \\ \texttt{h} \text{ for } \texttt{height} \end{pmatrix}$$

Part (A) of Figure 5.245 shows the 4 parallelepipeds of height 4, 2, 3 and 1 associated with the items of the RECTANGLES collection (parallelepipeds since each rectangle also has a height). Part (B) gives the corresponding cumulated 2-dimensional profile, where each number is the cumulated height of all the rectangles that contain the corresponding region. The CUMULATIVE_TWO_D constraint holds since the highest peak of the cumulated 2-dimensional profile does not exceed the upper limit 4 imposed by the last argument of the CUMULATIVE_TWO_D constraint.



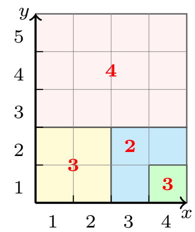


Figure 5.245: Two representations of a 2-dimensional cumulative profile of the **Example** slot (where the profile provides for each point of coordinates (c_x, c_y) the corresponding sum of the heights of the items intersecting that point): (A) a three dimensional representation and (B) a two dimensional representation from above with the height of the profile in red; as for the CUMULATIVE constraint the position of an item on the z axis does not matter, i.e. only its height matters.

Typical

|RECTANGLES| > 1
RECTANGLES.size1 > 0
RECTANGLES.size2 > 0
RECTANGLES.height > 0
LIMIT $<$sum(RECTANGLES.height)

Symmetries

- Items of RECTANGLES are permutable.
- Attributes of RECTANGLES are permutable w.r.t. permutation (start1, start2) (size1, size2) (last1, last2) (height) (*permutation applied to all items*).
- RECTANGLES.height can be decreased to any value ≥ 0.
- One and the same constant can be added to the start1 and last1 attributes of all items of RECTANGLES.
- One and the same constant can be added to the start2 and last2 attributes of all items of RECTANGLES.
- LIMIT can be increased.

Arg. properties

Contractible wrt. RECTANGLES.

Usage

The CUMULATIVE_TWO_D constraint is a necessary condition for the DIFFN constraint in 3 dimensions (i.e., the placement of parallelepipeds in such a way that they do not pairwise overlap and that each parallelepiped has his sides parallel to the sides of the placement space).

Algorithm

A first natural way to handle this constraint would be to accumulate the compulsory part [261] of the different rectangles in a quadtree [378]. To each leave of the quadtree we associate the cumulated height of the rectangles containing the corresponding region.

Systems

GEOST in **Choco**.

See also

related: DIFFN *(*CUMULATIVE_TWO_D *is a necessary condition for* DIFFN*: forget one dimension when the number of dimensions is equal to* 3*)*.

specialisation: BIN_PACKING *(*`square` *of size* 1 *with a* `height` *replaced by* `task` *of* `duration` 1*)*, CUMULATIVE *(*`rectangle` *with a* `height` *replaced by* `task` *with same* `height`*)*.

Keywords

characteristic of a constraint: derived collection.

constraint type: predefined constraint.

filtering: quadtree, compulsory part.

geometry: geometrical constraint.

5.102 CUMULATIVE_WITH_LEVEL_OF_PRIORITY

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin H. Simonis

Constraint CUMULATIVE_WITH_LEVEL_OF_PRIORITY(TASKS, PRIORITIES)

Arguments

$$\begin{array}{lll} \texttt{TASKS} & : & \texttt{collection}\begin{pmatrix} \texttt{priority}-\texttt{int}, \\ \texttt{origin}-\texttt{dvar}, \\ \texttt{duration}-\texttt{dvar}, \\ \texttt{end}-\texttt{dvar}, \\ \texttt{height}-\texttt{dvar} \end{pmatrix} \\ \texttt{PRIORITIES} & : & \texttt{collection}(\texttt{id}-\texttt{int}, \texttt{capacity}-\texttt{int}) \end{array}$$

Restrictions

required(TASKS, [priority, height])
require_at_least(2, TASKS, [origin, duration, end])
TASKS.priority ≥ 1
TASKS.priority $\leq$ |PRIORITIES|
TASKS.duration ≥ 0
TASKS.origin $\leq$ TASKS.end
TASKS.height ≥ 0
required(PRIORITIES, [id, capacity])
PRIORITIES.id ≥ 1
PRIORITIES.id $\leq$ |PRIORITIES|
increasing_seq(PRIORITIES, id)
increasing_seq(PRIORITIES, capacity)

Purpose

Consider a set $\mathcal{T}$ of tasks described by the TASKS collection where each task has a given priority chosen in the range [1, PRIORITIES]. Let $\mathcal{T}_i$ denote the subset of tasks of $\mathcal{T}$ that all have a priority less than or equal to i. For each set $\mathcal{T}_i$, the CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint forces that at each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i. Finally, it also imposes for each task of $\mathcal{T}$ the constraint origin + duration = end.

Example

$$\left(\begin{array}{l} \left\langle\begin{array}{lllll} \texttt{p}-1 & \texttt{origin}-1 & \texttt{dur}-2 & \texttt{end}-3 & \texttt{height}-1, \\ \texttt{p}-1 & \texttt{origin}-2 & \texttt{dur}-3 & \texttt{end}-5 & \texttt{height}-1, \\ \texttt{p}-1 & \texttt{origin}-5 & \texttt{dur}-2 & \texttt{end}-7 & \texttt{height}-2, \\ \texttt{p}-2 & \texttt{origin}-3 & \texttt{dur}-2 & \texttt{end}-5 & \texttt{height}-2, \\ \texttt{p}-2 & \texttt{origin}-6 & \texttt{dur}-3 & \texttt{end}-9 & \texttt{height}-1 \end{array}\right\rangle, \\ \langle \texttt{id}-1\ \texttt{capacity}-2, \texttt{id}-2\ \texttt{capacity}-3\rangle \end{array}\right)$$

(p for priority, dur for duration)

Figure 5.246 shows the cumulated profile associated with both levels of priority.

To each task of the CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint corresponds a set of rectangles containing the same number (i.e., the position of the task within the TASKS collection): the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. Tasks that have a priority of 1 are coloured in pink, while tasks that have a priority of 2 are coloured in blue. The CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint holds since:

- At each point in time the cumulated resource consumption profile of the tasks of priority 1 does not exceed the upper capacity 2 enforced by the first item of the PRIORITIES collection.
- At each point in time the cumulated resource consumption profile of the tasks of priority 1 and 2 does not exceed the upper capacity 3 enforced by the second item of the PRIORITIES collection.

Typical

$|\texttt{TASKS}| > 1$
$\texttt{range}(\texttt{TASKS.priority}) > 1$
$\texttt{range}(\texttt{TASKS.origin}) > 1$
$\texttt{range}(\texttt{TASKS.duration}) > 1$
$\texttt{range}(\texttt{TASKS.end}) > 1$
$\texttt{range}(\texttt{TASKS.height}) > 1$
$\texttt{TASKS.duration} > 0$
$\texttt{TASKS.height} > 0$
$|\texttt{PRIORITIES}| > 1$
$\texttt{PRIORITIES.capacity} > 0$
$\texttt{PRIORITIES.capacity} < \texttt{sum}(\texttt{TASKS.height})$
$|\texttt{TASKS}| > |\texttt{PRIORITIES}|$

Symmetries

- Items of TASKS are permutable.
- TASKS.priority can be increased to any value $\leq |\texttt{PRIORITIES}|$.
- TASKS.height can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.
- PRIORITIES.capacity can be increased.

Arg. properties

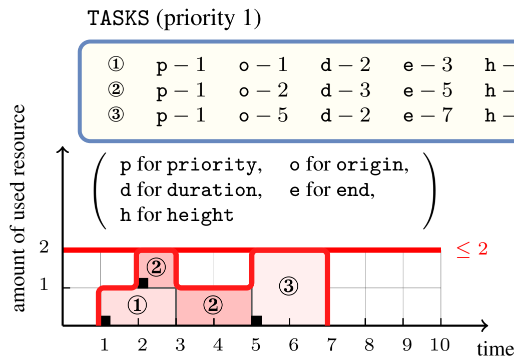
Contractible wrt. TASKS.

Usage

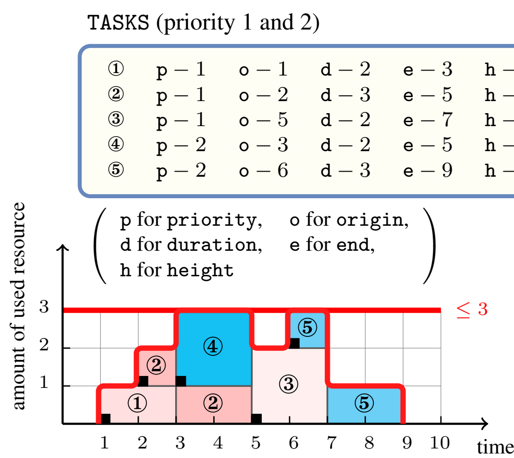
The CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint was suggested by problems from the telecommunication area where one has to ensure different levels of quality of service. For this purpose the capacity of a transmission link is split so that a given percentage is reserved to each level. In addition we have that, if the capacities allocated to levels $1, 2, \ldots, i$ is not completely used, then level $i+1$ can use the corresponding spare capacity.

Remark

The CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint can be modelled by a conjunction of CUMULATIVE constraints. As shown by the next example, the consistency for all variables of the CUMULATIVE constraints does not implies consistency for the corresponding CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint. The following CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint



(A) Profile of tasks of priority 1



(B) Profile of tasks of priority 1 and 2

Figure 5.246: Resource consumption profiles according to both levels of priority for the tasks of the **Example** slot

$$\left(\begin{array}{l}\left\langle\begin{array}{llll}\texttt{priority}-1 & \texttt{origin}-o_1 & \texttt{duration}-2 & \texttt{height}-2,\\ \texttt{priority}-1 & \texttt{origin}-o_2 & \texttt{duration}-2 & \texttt{height}-1,\\ \texttt{priority}-2 & \texttt{origin}-o_3 & \texttt{duration}-1 & \texttt{height}-3\end{array}\right\rangle,\\ \left\langle\begin{array}{ll}\texttt{id}-1 & \texttt{capacity}-2,\\ \texttt{id}-2 & \texttt{capacity}-3\end{array}\right\rangle\end{array}\right)$$

where the domains of o_1, o_2 and o_3 are respectively equal to $\{1,2,3\}$, $\{1,2,3\}$ and $\{1,2,3,4\}$ corresponds to the following conjunction of CUMULATIVE constraints

$$\texttt{cumulative}\left(\left\langle\begin{array}{lll}\texttt{origin}-o_1 & \texttt{duration}-2 & \texttt{height}-2,\\ \texttt{origin}-o_2 & \texttt{duration}-2 & \texttt{height}-1\end{array}\right\rangle, 2\right)$$

$$\texttt{cumulative}\left(\left\langle\begin{array}{lll}\texttt{origin}-o_1 & \texttt{duration}-2 & \texttt{height}-2,\\ \texttt{origin}-o_2 & \texttt{duration}-2 & \texttt{height}-1,\\ \texttt{origin}-o_3 & \texttt{duration}-1 & \texttt{height}-3\end{array}\right\rangle,3\right)$$

Even if the CUMULATIVE constraint could achieve arc-consistency, the previous conjunction of CUMULATIVE constraints would not detect the fact that there is no solution.

See also

common keyword: CUMULATIVE *(resource constraint)*.

used in graph description: SUM_CTR.

Keywords

characteristic of a constraint: derived collection.

constraint type: scheduling constraint, resource constraint, temporal constraint.

modelling: zero-duration task.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{TIME\_POINTS}-\texttt{collection}\left(\begin{array}{l}\texttt{idp}-\texttt{int},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{point}-\texttt{dvar}\end{array}\right),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{idp}-\texttt{TASKS.priority},\\ \texttt{duration}-\texttt{TASKS.duration},\\ \texttt{point}-\texttt{TASKS.origin}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{idp}-\texttt{TASKS.priority},\\ \texttt{duration}-\texttt{TASKS.duration},\\ \texttt{point}-\texttt{TASKS.end}\end{array}\right)\end{array}\right]\end{array}\right)$$

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) **NARC** = |TASKS|

For all items of PRIORITIES:

Arc input(s) TIME_POINTS TASKS

Arc generator $PRODUCT \mapsto$ collection(time_points, tasks)

Arc arity 2

Arc constraint(s)

- time_points.idp = PRIORITIES.id
- time_points.idp $\geq$ tasks.priority
- time_points.duration > 0
- tasks.origin $\leq$ time_points.point
- time_points.point $<$ tasks.end

Graph class

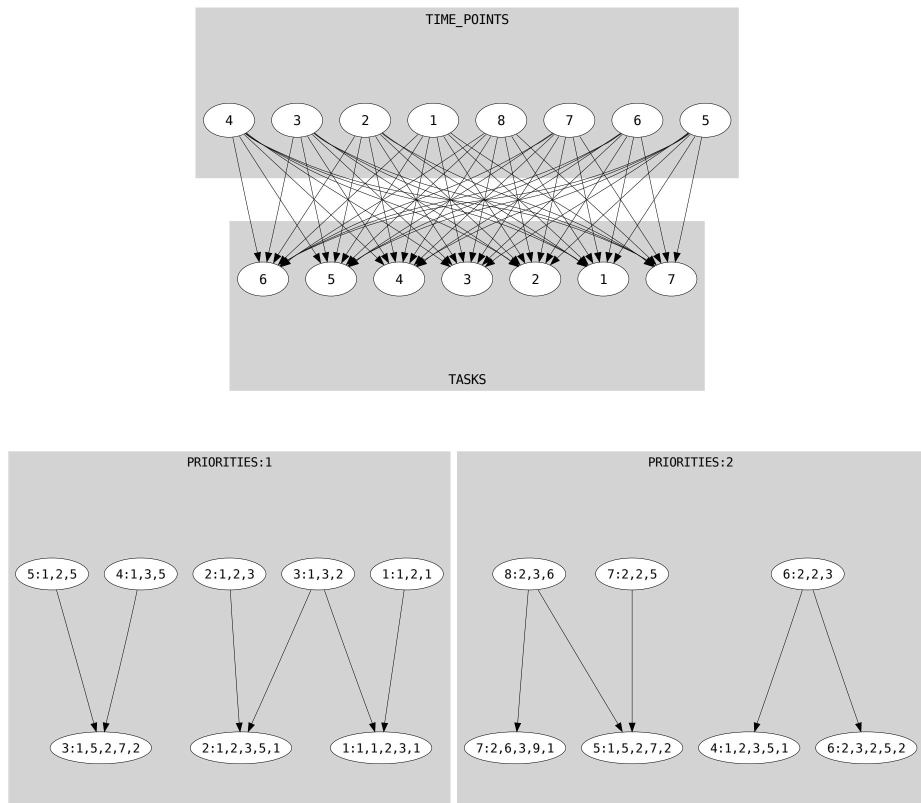
- ACYCLIC
- BIPARTITE
- NO_LOOP

Sets

$$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables}-\texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{TASKS.height})]\end{array}\right)\end{array}\right]$$

Constraint(s) on sets SUM_CTR(variables, $\leq$, PRIORITIES.capacity)

Graph model Within the context of the second graph constraint, part (A) of Figure 5.247 shows the initial graphs associated with priorities 1 and 2 of the **Example** slot. Part (B) of Figure 5.247 shows the corresponding final graphs associated with priorities 1 and 2. On the one hand, each source vertex of the final graph can be interpreted as a time point p. On the other hand the successors of a source vertex correspond to those tasks that both overlap that time point p and have a priority less than or equal to a given level. The CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint holds since for each successor set $\mathcal{S}$ of the final graph the sum of the height of the tasks in $\mathcal{S}$ is less than or equal to the capacity associated with a given level of priority.



(A)

(B)

Figure 5.247: Initial and final graph of the CUMULATIVE_WITH_LEVEL_OF_PRIORITY constraint

Signature Since TASKS is the maximum number of vertices of the final graph of the first graph constraint we can rewrite $\mathbf{NARC} = |\mathtt{TASKS}|$ to $\mathbf{NARC} \geq |\mathtt{TASKS}|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.103 CUMULATIVES

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [35]

Constraint CUMULATIVES(TASKS, MACHINES, CTR)

Arguments

$$\begin{array}{lll} \texttt{TASKS} & : & \texttt{collection}\begin{pmatrix} \texttt{machine}-\texttt{dvar}, \\ \texttt{origin}-\texttt{dvar}, \\ \texttt{duration}-\texttt{dvar}, \\ \texttt{end}-\texttt{dvar}, \\ \texttt{height}-\texttt{dvar} \end{pmatrix} \\ \texttt{MACHINES} & : & \texttt{collection}(\texttt{id}-\texttt{int}, \texttt{capacity}-\texttt{int}) \\ \texttt{CTR} & : & \texttt{atom} \end{array}$$

Restrictions

required(TASKS, [machine, height])
require_at_least(2, TASKS, [origin, duration, end])
in_attr(TASKS, machine, MACHINES, id)
TASKS.duration ≥ 0
TASKS.origin $\leq$ TASKS.end
$|$MACHINES$| > 0$
required(MACHINES, [id, capacity])
distinct(MACHINES, id)
CTR $\in [\leq, \geq]$

Purpose

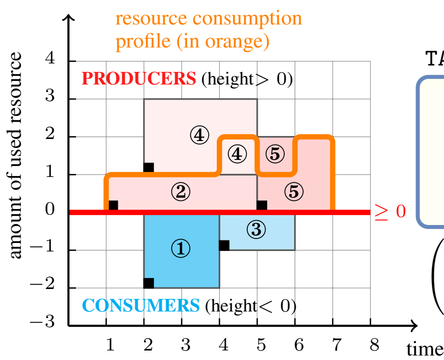
Consider a set $\mathcal{T}$ of tasks described by the TASKS collection. When CTR is equal to $\leq$ (respectively $\geq$), the CUMULATIVES constraint forces the following condition for each machine m: At each point in time, where at least one task assigned on machine m is present, the cumulated height of the set of tasks that both overlap that point and are assigned to machine m should be less than or equal to (respectively greater than or equal to) the capacity associated with machine m. A task overlaps a point i if and only if (1) its origin is less than or equal to i, and (2) its end is strictly greater than i. It also imposes for each task of $\mathcal{T}$ the constraint origin + duration = end.

Example

$$\left(\begin{array}{l} \left\langle\begin{array}{lllll} \texttt{m}-1 & \texttt{origin}-2 & \texttt{dur}-2 & \texttt{end}-4 & \texttt{height}--2, \\ \texttt{m}-1 & \texttt{origin}-1 & \texttt{dur}-4 & \texttt{end}-5 & \texttt{height}-1, \\ \texttt{m}-1 & \texttt{origin}-4 & \texttt{dur}-2 & \texttt{end}-6 & \texttt{height}--1, \\ \texttt{m}-1 & \texttt{origin}-2 & \texttt{dur}-3 & \texttt{end}-5 & \texttt{height}-2, \\ \texttt{m}-1 & \texttt{origin}-5 & \texttt{dur}-2 & \texttt{end}-7 & \texttt{height}-2, \\ \texttt{m}-2 & \texttt{origin}-3 & \texttt{dur}-2 & \texttt{end}-5 & \texttt{height}--1, \\ \texttt{m}-2 & \texttt{origin}-1 & \texttt{dur}-4 & \texttt{end}-5 & \texttt{height}-1 \end{array}\right\rangle, \\ \langle \texttt{id}-1\ \texttt{capacity}-0, \texttt{id}-2\ \texttt{capacity}-0\rangle, \geq \end{array}\right)$$

(m for machine, dur for duration)

Figure 5.248 shows with a thick line the cumulated profile on the two machines described by the `MACHINES` collection. Within this profile a task with a positive (respectively negative) height is represented by a pink (respectively blue) rectangle, where the length of the rectangle corresponds to the duration of the task. The CUMULATIVES constraint holds since, both on machines 1 and 2, we have that at each point in time the cumulated resource consumption is greater than or equal to the limit 0 enforced by the last argument (i.e., the attribute `capacity` of the items of the `MACHINES` collection) of the CUMULATIVES constraint (i.e., we have a limit of 0 both on machines 1 and 2).

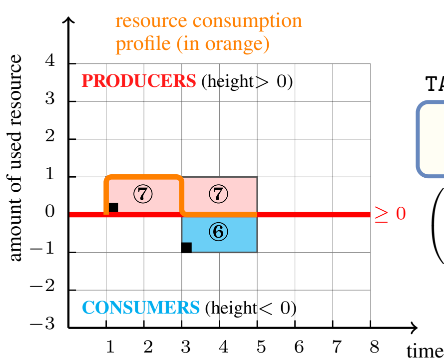


TASKS

①	m − 1	o − 2	d − 2	e − 4	h − −2
②	m − 1	o − 1	d − 4	e − 5	h − 1
③	m − 1	o − 4	d − 2	e − 6	h − −1
④	m − 1	o − 2	d − 3	e − 5	h − 2
⑤	m − 1	o − 5	d − 2	e − 7	h − 2

(`m` for `machine`, `o` for `origin`, `d` for `duration`, `e` for `end`, `h` for `height`)

(A) tasks on machine 1



TASKS

⑥	m − 2	o − 3	d − 2	e − 5	h − −1
⑦	m − 2	o − 1	d − 4	e − 5	h − 1

(`m` for `machine`, `o` for `origin`, `d` for `duration`, `e` for `end`, `h` for `height`)

(B) tasks on machine 2

Figure 5.248: Resource consumption profiles on the different machines for the tasks of the **Example** slot

Typical

$|\mathtt{TASKS}| > 1$
$\mathtt{range}(\mathtt{TASKS.machine}) > 1$
$\mathtt{range}(\mathtt{TASKS.origin}) > 1$
$\mathtt{range}(\mathtt{TASKS.duration}) > 1$
$\mathtt{range}(\mathtt{TASKS.end}) > 1$
$\mathtt{range}(\mathtt{TASKS.height}) > 1$
$\mathtt{TASKS.duration} > 0$
$\mathtt{TASKS.height} \neq 0$
$|\mathtt{MACHINES}| > 1$
$\mathtt{MACHINES.capacity} < \mathtt{sum}(\mathtt{TASKS.height})$
$|\mathtt{TASKS}| > |\mathtt{MACHINES}|$

Symmetries

- Items of TASKS are permutable.
- Items of MACHINES are permutable.
- All occurrences of two distinct values in TASKS.machine or MACHINES.id can be swapped; all occurrences of a value in TASKS.machine or MACHINES.id can be renamed to any unused value.

Arg. properties

Contractible wrt. TASKS when RELOP $\in [\leq]$ and $\mathtt{minval}(\mathtt{TASKS.height}) \geq 0$.

Usage

As shown in the **Example** slot, the CUMULATIVES constraint is useful for covering problems where different demand profiles have to be covered by a set of tasks. This is modelled in the following way:

- To each demand profile is associated a given machine m and a set of tasks for which all attributes (machine, origin, duration, end, height) are fixed; moreover the machine attribute is fixed to m and the height attribute is strictly negative. For each machine m the cumulated profile of all the previous tasks constitutes the demand profile to cover.
- To each task that can be used to cover the demand is associated a task for which the height attribute is a positive integer; the height attribute describes the amount of demand that can be covered by the task at each instant during its execution (between its origin and its end) on the demand profile associated with the machine attribute.
- In order to express the fact that each demand profile should completely be covered, we set the capacity attribute of each machine to 0. We can also relax the constraint by setting the capacity attribute to a negative number that specifies the maximum allowed uncovered demand at each instant.

The demand profiles might also not be completely fixed in advance.

When all the heights of the tasks are non-negative, one other possible use of the CUMULATIVES constraint is to enforce to reach a minimum level of resource consumption. This is imposed on those time points that are overlapped by at least one task.

By introducing a dummy task of height 0, of origin the minimum origin of all the tasks and of end the maximum end of all the tasks, this can also be imposed between the first and the last utilisation of the resource.

Finally the CUMULATIVES constraint is also useful for scheduling problems where several *cumulative* machines are available and where you have to assign each task on a specific machine.

Algorithm Three filtering algorithms for this constraint are described in [35].

Systems CUMULATIVES in **Gecode**, CUMULATIVES in **SICStus**.

See also **assignment dimension removed:** CUMULATIVE *(negative* `heights` *not allowed)*.

common keyword: CALENDAR *(scheduling constraint)*, COLOURED_CUMULATIVES *(resource constraint)*.

generalisation: DIFFN *(*`task` *with* `machine assignment` *and* `origin` *attributes replaced by orthotope)*.

used in graph description: SUM_CTR.

Keywords **application area:** workload covering.

characteristic of a constraint: derived collection.

complexity: sequencing with release times and deadlines.

constraint type: scheduling constraint, resource constraint, temporal constraint, timetabling constraint.

filtering: compulsory part, sweep, minimum task duration.

modelling: assignment dimension, assignment to the same set of values, scheduling with machine choice, calendars and preemption, zero-duration task.

modelling exercises: assignment to the same set of values, scheduling with machine choice, calendars and preemption.

problems: producer-consumer, demand profile.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{TIME\_POINTS}-\texttt{collection}\left(\begin{array}{l}\texttt{idm}-\texttt{int},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{point}-\texttt{dvar}\end{array}\right),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{idm}-\texttt{TASKS.machine},\\ \texttt{duration}-\texttt{TASKS.duration},\\ \texttt{point}-\texttt{TASKS.origin}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{idm}-\texttt{TASKS.machine},\\ \texttt{duration}-\texttt{TASKS.duration},\\ \texttt{point}-\texttt{TASKS.end}\end{array}\right)\end{array}\right]\end{array}\right)$$

Arc input(s) TASKS

Arc generator $SELF \mapsto$ collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin + tasks.duration = tasks.end

Graph property(ies) **NARC** = |TASKS|

For all items of MACHINES:

Arc input(s) TIME_POINTS TASKS

Arc generator $PRODUCT \mapsto$ collection(time_points, tasks)

Arc arity 2

Arc constraint(s)
- time_points.idm = MACHINES.id
- time_points.idm = tasks.machine
- time_points.duration > 0
- tasks.origin ≤ time_points.point
- time_points.point < tasks.end

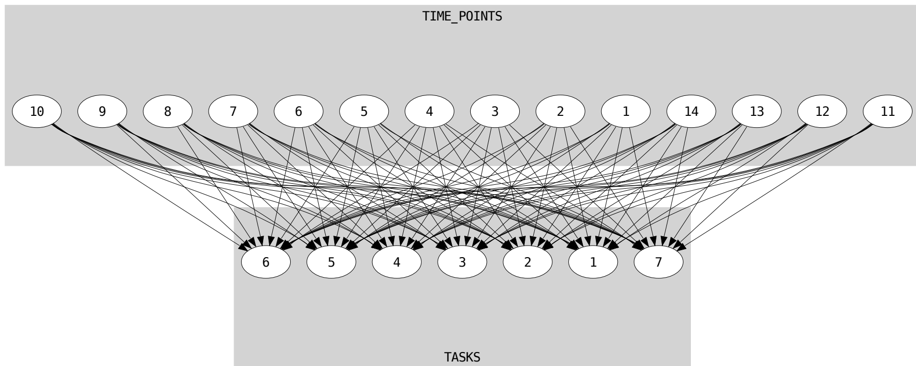
Graph class
- ACYCLIC
- BIPARTITE
- NO_LOOP

Sets

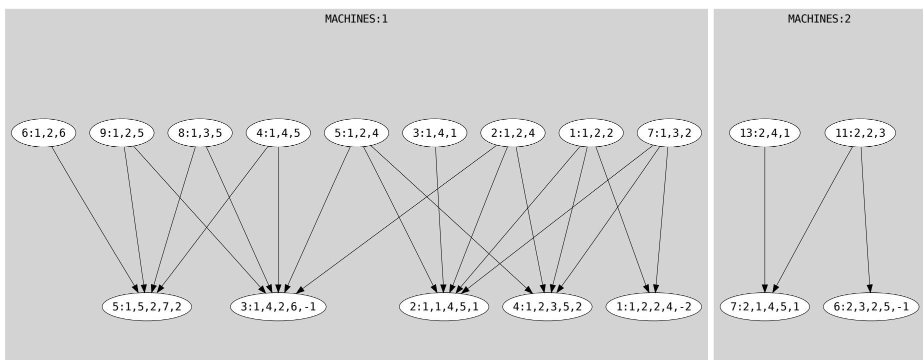
$$\mathsf{SUCC}\mapsto\left[\begin{array}{l}\texttt{source},\\ \texttt{variables}-\texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ \left[\texttt{item}(\texttt{var}-\texttt{TASKS.height})\right]\end{array}\right)\end{array}\right]$$

Constraint(s) on sets SUM_CTR(variables, CTR, MACHINES.capacity)

Graph model Within the context of the second graph constraint, part (A) of Figure 5.249 shows the initial graphs associated with machines 1 and 2 of the **Example** slot. Part (B) of Figure 5.249 shows the corresponding final graphs associated with machines 1 and 2. On the one hand, each source vertex of the final graph can be interpreted as a time point p on a specific machine m. On the other hand the successors of a source vertex correspond to those tasks that both overlap that time point p and are assigned to machine m. Since they do not have any successors we have eliminated those vertices corresponding to the end of the last three tasks of the TASKS collection. The CUMULATIVES constraint holds since for each successor set $\mathcal{S}$ of the final graph the sum of the height of the tasks in $\mathcal{S}$ is greater than or equal to the capacity of the machine corresponding to the time point associated with $\mathcal{S}$.



(A)



(B)

Figure 5.249: Initial and final graph of the CUMULATIVES constraint

Signature Since **NARC** is the maximum number of vertices of the final graph of the first graph constraint we can rewrite **NARC** $= |$TASKS$|$ to **NARC** $\geq |$TASKS$|$. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.104 CUTSET

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [165]

Constraint CUTSET(SIZE_CUTSET, NODES)

Arguments

SIZE_CUTSET : dvar
NODES : collection(index−int, succ−sint, bool−dvar)

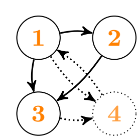
Restrictions

SIZE_CUTSET ≥ 0
SIZE_CUTSET $\leq$ |NODES|
required(NODES, [index, succ, bool])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.bool ≥ 0
NODES.bool ≤ 1

Purpose Consider a digraph G with n vertices described by the NODES collection. Enforces that the subset of kept vertices of cardinality $n -$ SIZE_CUTSET and their corresponding arcs form a graph without circuit.

Example

$$\left(1, \left\langle \begin{array}{lll} \texttt{index}-1 & \texttt{succ}-\{2,3,4\} & \texttt{bool}-1, \\ \texttt{index}-2 & \texttt{succ}-\{3\} & \texttt{bool}-1, \\ \texttt{index}-3 & \texttt{succ}-\{4\} & \texttt{bool}-1, \\ \texttt{index}-4 & \texttt{succ}-\{1\} & \texttt{bool}-0 \end{array} \right\rangle \right)$$



The CUTSET constraint holds since the vertices of the NODES collection for which the bool attribute is set to 1 correspond to a graph without circuit and since exactly one (SIZE_CUTSET $= 1$) vertex has its bool attribute set to 0.

Typical

SIZE_CUTSET > 0
SIZE_CUTSET $\leq$ |NODES|
|NODES| > 1

Symmetry Items of NODES are permutable.

Usage The article [165] introducing the CUTSET constraint mentions applications from various areas such that deadlock breaking or program verification.

Remark The undirected version of the CUTSET constraint corresponds to the minimum feedback vertex set problem.

Algorithm The filtering algorithm presented in [165] uses graph reduction techniques inspired from Levy and Low [271] as well as from Lloyd, Soffa and Wang [275].

Keywords

application area: deadlock breaking, program verification.

constraint type: graph constraint.

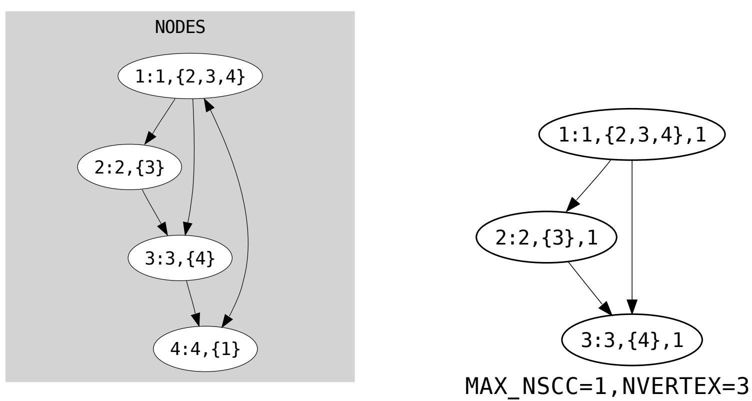
final graph structure: circuit, directed acyclic graph, acyclic, no loop.

problems: minimum feedback vertex set.

Arc input(s)	NODES
Arc generator	*CLIQUE*$\mapsto$collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	• IN_SET(nodes2.index, nodes1.succ) • nodes1.bool $=1$ • nodes2.bool $=1$
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NVERTEX** $=$ |NODES| $-$ SIZE_CUTSET
Graph class	• ACYCLIC • NO_LOOP

Graph model We use a set of integers for representing the successors of each vertex. Because of the arc constraint, all arcs such that the bool attribute of one extremity is equal to 0 are eliminated; Therefore all vertices for which the bool attribute is equal to 0 are also eliminated (since they will correspond to isolated vertices). The graph property **MAX_NSCC** ≤ 1 enforces the size of the largest strongly connected component to not exceed 1; Therefore, the final graph cannot contain any circuit.

Part (A) of Figure 5.250 shows the initial graph from which we have chosen to start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.250 gives the final graph associated with the **Example** slot. Since we use the **NVERTEX** graph property, the vertices of the final graph are stressed in bold. The CUTSET constraint holds since the final graph does not contain any circuit and since the number of removed vertices SIZE_CUTSET is equal to 1.



(A) **(B)**

Figure 5.250: Initial and final graph of the CUTSET set constraint

5.105 CYCLE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [47]

Constraint CYCLE(NCYCLE, NODES)

Arguments

NCYCLE : dvar
NODES : collection(index−int, succ−dvar)

Restrictions

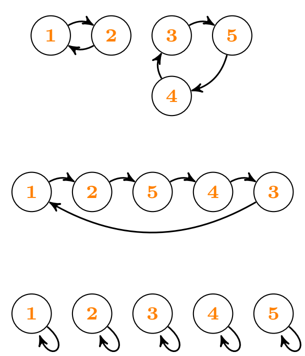
$\texttt{NCYCLE} \geq 1$
$\texttt{NCYCLE} \leq |\texttt{NODES}|$
required(NODES, [index, succ])
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
distinct(NODES, index)
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$

Purpose

Consider a digraph G described by the NODES collection. NCYCLE is equal to the number of circuits for covering G in such a way that each vertex of G belongs to a single circuit. NCYCLE can also be interpreted as the number of cycles of the permutation associated with the successor variables of the NODES collection.

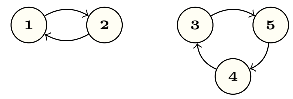
Example

$$\left(\begin{array}{l}2, \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-2,\\ \texttt{index}-2 & \texttt{succ}-1,\\ \texttt{index}-3 & \texttt{succ}-5,\\ \texttt{index}-4 & \texttt{succ}-3,\\ \texttt{index}-5 & \texttt{succ}-4\end{array}\right\rangle\end{array}\right)$$
$$\left(\begin{array}{l}1, \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-2,\\ \texttt{index}-2 & \texttt{succ}-5,\\ \texttt{index}-3 & \texttt{succ}-1,\\ \texttt{index}-4 & \texttt{succ}-3,\\ \texttt{index}-5 & \texttt{succ}-4\end{array}\right\rangle\end{array}\right)$$
$$\left(\begin{array}{l}5, \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{succ}-2,\\ \texttt{index}-3 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{succ}-4,\\ \texttt{index}-5 & \texttt{succ}-5\end{array}\right\rangle\end{array}\right)$$



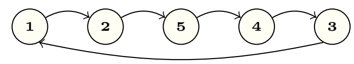
- In the first example we have the following 2 (NCYCLE $=2$) cycles: $1 \mapsto 2 \mapsto 1$ and $3 \mapsto 5 \mapsto 4 \mapsto 3$. Consequently, the corresponding CYCLE constraint holds.

CYCLE($\mathbf{2}, \langle 1\,2,\ 2\,1,\ 3\,5,\ 4\,3,\ 5\,4\rangle$)



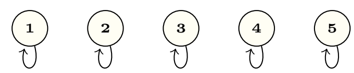
- In the second example we have 1 (NCYCLE $=1$) cycle: $1 \mapsto 2 \mapsto 5 \mapsto 4 \mapsto 3 \mapsto 1$.

CYCLE($\mathbf{1}, \langle 1\,2,\ 2\,5,\ 3\,1,\ 4\,3,\ 5\,4\rangle$)



- In the third example we have the following 5 (NCYCLE $=5$) cycles: $1 \mapsto 1$, $2 \mapsto 2$, $3 \mapsto 3$, $4 \mapsto 4$ and $5 \mapsto 5$.

CYCLE($\mathbf{5}, \langle 1\,1,\ 2\,2,\ 3\,3,\ 4\,4,\ 5\,5\rangle$)



All solutions Figure 5.251 gives all solutions to the following non ground instance of the CYCLE constraint: $\mathtt{N} \in [1,2]$, $\mathtt{V}_1 \in [2,4]$, $\mathtt{V}_2 \in [2,3]$, $\mathtt{V}_3 \in [1,6]$, $\mathtt{V}_4 \in [2,5]$, $\mathtt{V}_5 \in [2,3]$, $\mathtt{V}_6 \in [1,6]$, CYCLE($\mathtt{N}, \langle 1\,\mathtt{V}_1,\ 2\,\mathtt{V}_2,\ 3\,\mathtt{V}_3,\ 4\,\mathtt{V}_4,\ 5\,\mathtt{V}_5,\ 6\,\mathtt{V}_6\rangle$).

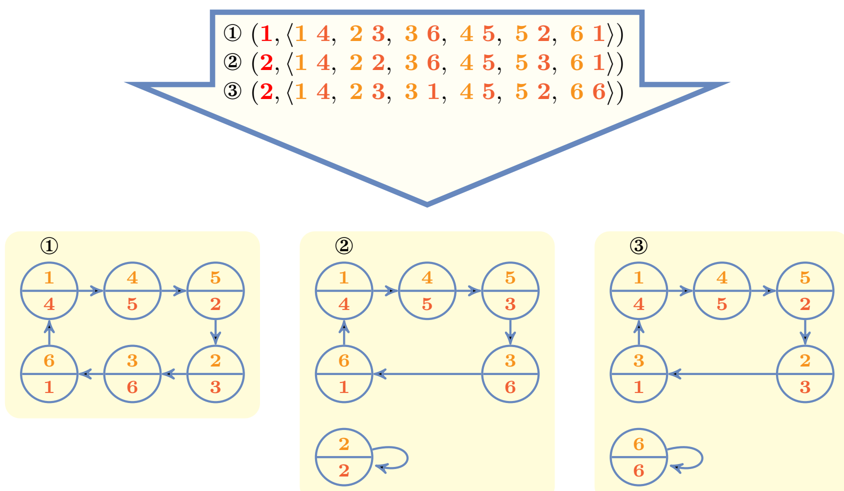


Figure 5.251: All solutions corresponding to the non ground example of the CYCLE constraint of the **All solutions** slot

Typical

$\mathtt{NCYCLE} < |\mathtt{NODES}|$
$|\mathtt{NODES}| > 2$

Symmetries

- Items of NODES are permutable.
- Attributes of NODES are permutable w.r.t. permutation (index, succ) (*permutation applied to all items*).

Arg. properties Functional dependency: NCYCLE determined by NODES.

Usage The PhD thesis of Éric Bourreau [93] mentions the following applications of extensions of the CYCLE constraint:

- The balanced Euler knight problem where one tries to cover a rectangular chessboard of size $N \cdot M$ by C knights that all have to visit between $2 \cdot \lfloor\lfloor(N \cdot M)/C\rfloor/2\rfloor$ and $2 \cdot \lceil\lceil(N \cdot M)/C\rceil/2\rceil$ distinct locations. For some values of N, M and C there does not exist any solution to the previous problem. This is the case, for example, when $N = M = C = 6$. Figure 5.252 depicts the graph associated with the 6×6 chessboard as well as examples of balanced solutions with respectively 1, 2, 3, 4 and 5 knights.
- Some pick-up delivery problems where a fleet of vehicles has to transport a set of orders. Each order is characterised by its initial location, its final destination and its weight. In addition one also has to take into account the capacity of the different vehicles.

Remark In the original CYCLE constraint of **CHIP** the `index` attribute was not explicitly present. It was implicitly defined as the position of a variable in a list.

In an early version of the **CHIP** there was a constraint named CIRCUIT that, from a declarative point of view, was equivalent to cycle(1, NODES). In ALICE [267] the CIRCUIT constraint was also present.

Given a complete digraph of n vertices as well as an unrestricted number of circuits NCYCLE, the total number of solutions to the corresponding CYCLE constraint corresponds to the sequence A000142 of the On-Line Encyclopaedia of Integer Sequences [403]. Given a complete digraph of n vertices as well as a fixed number of circuits NCYCLE between 1 and n, the total number of solutions to the corresponding CYCLE constraint corresponds to the so called *Stirling number of first kind*.

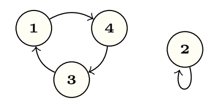
Algorithm Since all `succ` variables have to take distinct values one can reuse the algorithms associated with the ALLDIFFERENT constraint. A second necessary condition is to have no more than $\overline{\text{NCYCLE}}$ strongly connected components. Pruning for enforcing this condition, as soon as we have $\overline{\text{NCYCLE}}$ strongly connected components, can be done by forcing all strong bridges to belong to the final solution, since otherwise we would have more than $\overline{\text{NCYCLE}}$ strongly connected components. Since all the vertices of a circuit belong to the same strongly connected component an arc going from one strongly connected component to another strongly connected component has to be removed.

Reformulation

Let n and $s_1, s_2, \ldots, s_n$ respectively denotes the number of vertices (i.e., |NODES|) and the successor variables associated with vertices $1, 2, \ldots, n$. The CYCLE constraint can be reformulated as a conjunction of one ALLDIFFERENT constraint, $n \cdot (n-1)$ ELEMENT constraints, n MINIMUM constraints, and one NVALUE constraint.

- First, we state an ALLDIFFERENT$(\langle s_1, s_2, \ldots, s_n\rangle)$ constraint for enforcing distinct values to be assigned to the successor variables.
- Second, the key idea is to extract for each vertex i (with $i \in [1, n]$) all the vertices that belong to the same cycle. This is done by stating a conjunction of $n-1$ ELEMENT constraints of the form:
  ELEMENT$(i, \langle s_1, s_2, \ldots, s_n\rangle, s_{i,1})$,
  ELEMENT$(s_{i,1}, \langle s_1, s_2, \ldots, s_n\rangle, s_{i,2})$,
  ....
  ELEMENT$(s_{i,n-2}, \langle s_1, s_2, \ldots, s_n\rangle, s_{i,n-1})$.
  Then, using a MINIMUM$(m_i, \langle i, s_{i,1}, s_{i,2}, \ldots, s_{i,n-1}\rangle)$ constraint, we get a unique representative for the cycle containing vertex i.
- Third, using a NVALUE(NCYCLE, $\langle m_1, m_2, \ldots, m_n\rangle$) constraint, we get the number of distinct cycles.

Illustration of the reformulation of
CYCLE$(\mathbf{2}, \langle 1\ 4,\ 2\ 2,\ 3\ 1,\ 4\ 3\rangle)$



ALLDIFFERENT$(\langle 4, 2, 1, 3\rangle)$

ELEMENT$(\mathbf{1}, \langle 4, 2, 1, 3\rangle, \boxed{4})$
ELEMENT$(4, \langle 4, 2, 1, 3\rangle, \boxed{3})$
ELEMENT$(3, \langle 4, 2, 1, 3\rangle, \boxed{1})$
(*representative of the cycle containing vertex* **1**) min $=1$

ELEMENT$(\mathbf{2}, \langle 4, 2, 1, 3\rangle, \boxed{2})$
ELEMENT$(2, \langle 4, 2, 1, 3\rangle, \boxed{2})$
ELEMENT$(2, \langle 4, 2, 1, 3\rangle, \boxed{2})$
(*representative of the cycle containing vertex* **2**) min $=2$

ELEMENT$(\mathbf{3}, \langle 4, 2, 1, 3\rangle, \boxed{1})$
ELEMENT$(1, \langle 4, 2, 1, 3\rangle, \boxed{4})$
ELEMENT$(4, \langle 4, 2, 1, 3\rangle, \boxed{3})$
(*representative of the cycle containing vertex* **3**) min $=1$

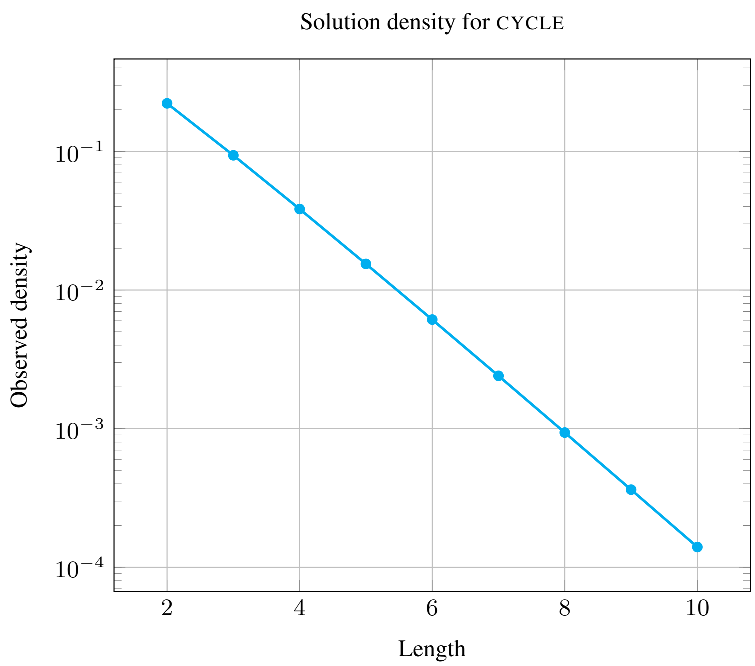
ELEMENT$(\mathbf{4}, \langle 4, 2, 1, 3\rangle, \boxed{3})$
ELEMENT$(3, \langle 4, 2, 1, 3\rangle, \boxed{1})$
ELEMENT$(1, \langle 4, 2, 1, 3\rangle, \boxed{4})$
(*representative of the cycle containing vertex* **4**) min $=1$

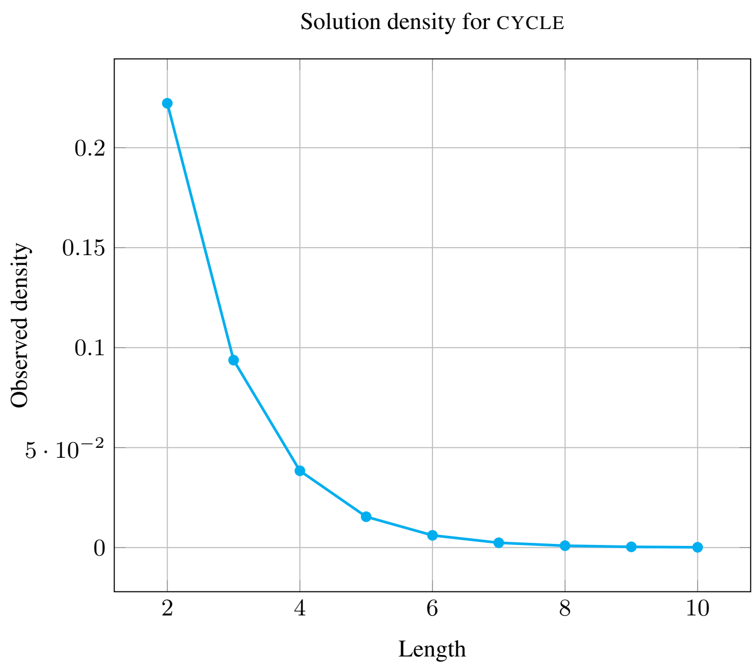
NVALUE$=\mathbf{2}$

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	2	6	24	120	720	5040	40320	362880	3628800

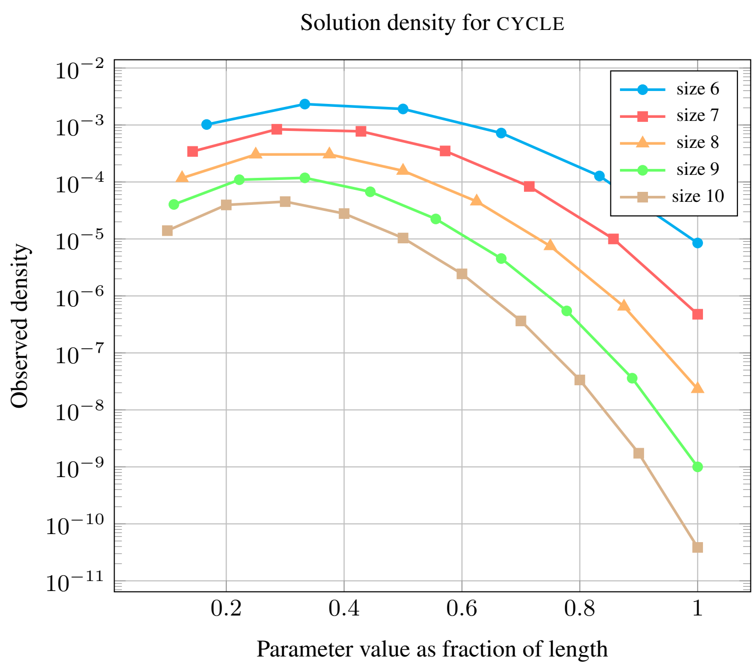
Number of solutions for CYCLE: domains $0..n$

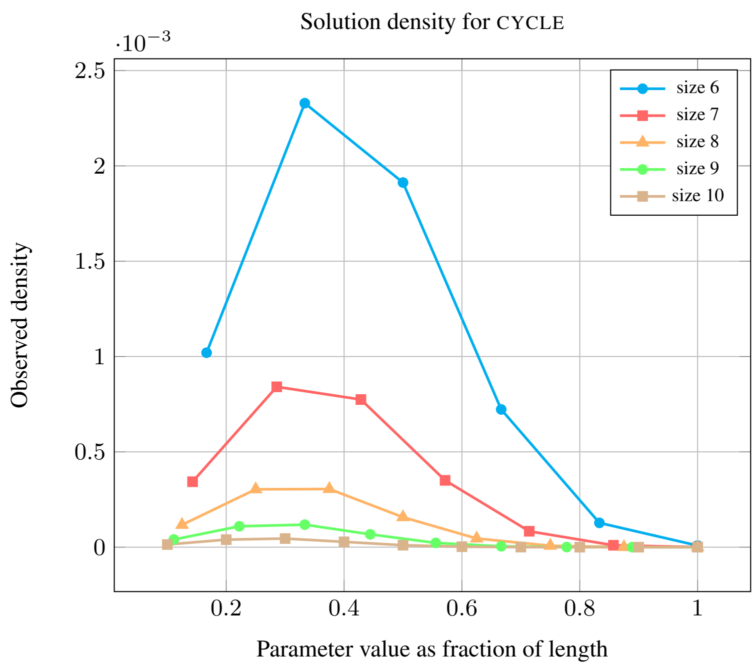

Solution density for CYCLE
Observed density
10^{-1}
10^{-2}
10^{-3}
10^{-4}
2
4
6
8
10
Length


Solution density for CYCLE
Observed density
0.2
0.15
0.1
$5 \cdot 10^{-2}$
0
2
4
6
8
10
Length

Length (n)		2	3	4	5	6	7	8	9	10
Total		2	6	24	120	720	5040	40320	362880	3628800
Parameter value	1	1	2	6	24	120	720	5040	40320	362880
	2	1	3	11	50	274	1764	13068	109584	1026576
	3	-	1	6	35	225	1624	13132	118124	1172700
	4	-	-	1	10	85	735	6769	67284	723680
	5	-	-	-	1	15	175	1960	22449	269325
	6	-	-	-	-	1	21	322	4536	63273
	7	-	-	-	-	-	1	28	546	9450
	8	-	-	-	-	-	-	1	36	870
	9	-	-	-	-	-	-	-	1	45
	10	-	-	-	-	-	-	-	-	1

Solution count for CYCLE: domains $0..n$





See also

common keyword: ALLDIFFERENT *(permutation)*,
CIRCUIT_CLUSTER *(graph constraint, one_succ)*,
CYCLE_CARD_ON_PATH *(permutation,graph partitioning constraint)*,
CYCLE_OR_ACCESSIBILITY *(graph constraint)*,
CYCLE_RESOURCE *(graph partitioning constraint)*,
DERANGEMENT *(permutation)*,
GRAPH_CROSSING *(graph constraint,graph partitioning constraint)*,
INVERSE *(permutation)*,
MAP *(graph partitioning constraint)*,
SYMMETRIC_ALLDIFFERENT *(permutation)*,
TOUR *(graph constraint)*,
TREE *(graph partitioning constraint)*.

implies: ALLDIFFERENT.

implies (items to collection): ATLEAST_NVECTOR.

related: BALANCE_CYCLE *(counting number of cycles versus controlling how balanced the cycles are)*.

specialisation: CIRCUIT (`NCYCLE` *set to* 1).

used in reformulation: ALLDIFFERENT, ELEMENT, MINIMUM, NVALUE.

Keywords

characteristic of a constraint: core.

combinatorial object: permutation.

constraint arguments: business rules.

constraint type: graph constraint, graph partitioning constraint.

filtering: strong bridge, DFS-bottleneck.

final graph structure: circuit, connected component, strongly connected component, one_succ.

modelling: cycle, functional dependency.

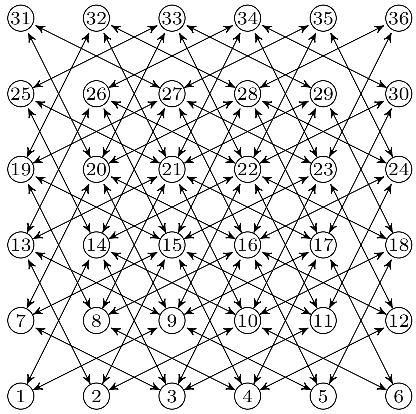
problems: pick-up delivery.

puzzles: Euler knight.

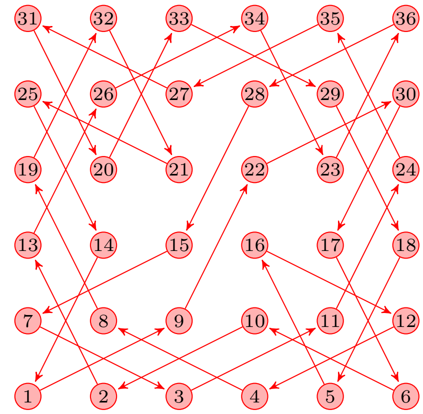
Cond. implications

- CYCLE(NCYCLE, NODES)
  with NCYCLE $= 1$
  implies BALANCE_CYCLE(BALANCE, NODES)
  when BALANCE $= 0$.

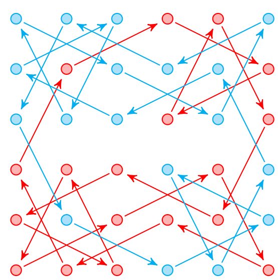
- CYCLE(NCYCLE, NODES)
  implies PERMUTATION(VARIABLES : NODES).



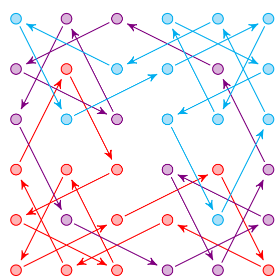
Graph of potential moves
of a 6×6 chessboard



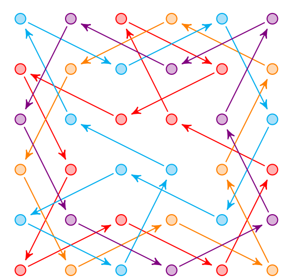
1 knight
(**36** moves)



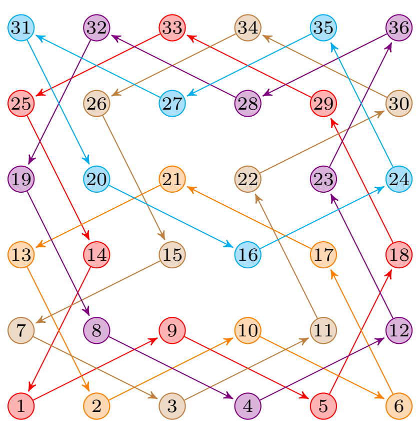
2 knights
(**18** and **18** moves)



3 knights
(**12**, **12** and **12** moves)



4 knights
(**8**, **8**, **10** and **10** moves)



5 knights
(**6**, **6**, **8**, **8** and **8** moves)

`NODES` (`i` for `index`, `s` for `succ`)

`i` − **1**	`s` − 9,	`i` − **19**	`s` − 8,
`i` − **2**	`s` − 10,	`i` − **20**	`s` − 16,
`i` − **3**	`s` − 11,	`i` − **21**	`s` − 13,
`i` − **4**	`s` − 12,	`i` − **22**	`s` − 30,
`i` − **5**	`s` − 18,	`i` − **23**	`s` − 36,
`i` − **6**	`s` − 17,	`i` − **24**	`s` − 35,
`i` − **7**	`s` − 3,	`i` − **25**	`s` − 14,
`i` − **8**	`s` − 4,	`i` − **26**	`s` − 15,
`i` − **9**	`s` − 5,	`i` − **27**	`s` − 31,
`i` − **10**	`s` − 6,	`i` − **28**	`s` − 32,
`i` − **11**	`s` − 22,	`i` − **29**	`s` − 33,
`i` − **12**	`s` − 23,	`i` − **30**	`s` − 34,
`i` − **13**	`s` − 2,	`i` − **31**	`s` − 20,
`i` − **14**	`s` − 1,	`i` − **32**	`s` − 19,
`i` − **15**	`s` − 7,	`i` − **33**	`s` − 25,
`i` − **16**	`s` − 24,	`i` − **34**	`s` − 26,
`i` − **17**	`s` − 21,	`i` − **35**	`s` − 27,
`i` − **18**	`s` − 29,	`i` − **36**	`s` − 28

Figure 5.252: Graph of potential moves of a 6×6 chessboard, corresponding balanced knight's tours with 1 up to 5 knights, and collection of nodes passed to the CYCLE constraint corresponding to the solution with 5 knights; note that there is no balanced knight's tour on a 6×6 chessboard where each knight exactly performs 6 moves.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **NTREE** = 0 • **NCC** = NCYCLE
Graph class	ONE_SUCC

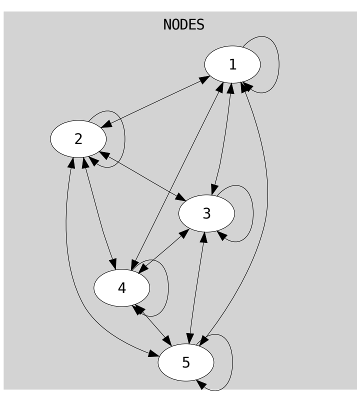
Graph model From the restrictions and from the arc constraint, we deduce that we have a bijection from the successor variables to the values of interval $[1, |\text{NODES}|]$. With no explicit restrictions it would have been impossible to derive this property.

In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the CYCLE constraint considers objects that have two attributes:

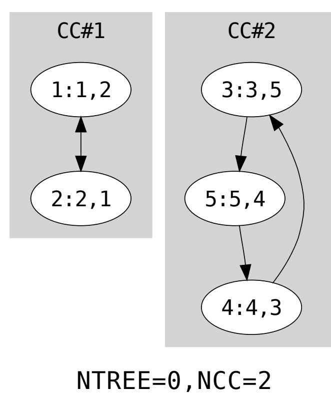
- One fixed attribute index that is the identifier of the vertex,
- One variable attribute succ that is the successor of the vertex.

The graph property **NTREE** = 0 is used in order to avoid having vertices that both do not belong to a circuit and have at least one successor located on a circuit. This concretely means that all vertices of the final graph should belong to a circuit.

Parts (A) and (B) of Figure 5.253 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NCC** graph property, we show the two connected components of the final graph. The constraint holds since all the vertices belong to a circuit (i.e., **NTREE** = 0) and since NCYCLE = **NCC** = 2.



(A)



(B)

Figure 5.253: Initial and final graph of the CYCLE constraint

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint CYCLE$(1, \langle 1\ 2, 2\ 1, 3\ 2\rangle)$ hold?

B. Does the constraint CYCLE$(2, \langle 1\ 3, 2\ 2, 3\ 1\rangle)$ hold?

C. Does the constraint CYCLE$(3, \langle 1\ 1, 2\ 2, 3\ 3\rangle)$ hold?

D. Does the constraint CYCLE$(2, \langle 1\ 5, 2\ 4, 3\ 3, 4\ 2, 5\ 1\rangle)$ hold?

[a] Hint: go back to the definition of CYCLE.

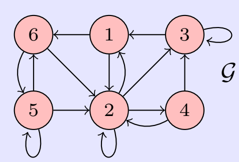
EXERCISE 2 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\begin{cases} N \in \{2,4\}, \\ V_1 \in \{1,3,4,5\}, \quad V_2 \in \{3,4\}, \quad V_3 \in \{2,3,5,6\}, \\ V_4 \in \{1,4,6\}, \quad V_5 \in \{2,6\}, \quad V_6 \in \{3,4,6\}, \\ \text{CYCLE}(N, \langle 1\ V_1,\ 2\ V_2,\ 3\ V_3,\ 4\ V_4,\ 5\ V_5,\ 6\ V_6\rangle). \end{cases}$$

[a] Hint: follow the order induced by the functional dependency between the arguments of CYCLE, start with variables that have the smallest domain.

EXERCISE 3 (identifying infeasible values)[a]



A. Describe the following digraph $\mathcal{G}$ in terms of successor variables and their corresponding domains. Give the implicit assumption behind this description.

B. Model with a single CYCLE constraint the problem of finding a Hamiltonian cycle[b] in the graph $\mathcal{G}$.

C. Identify variable-value pairs that do not belong to any solution to the CYCLE constraint stated in the previous question.

[a] Hint: make a link between the successor variables and the arcs of the graph, identify the basic constraint on the successor variables, make a what-if reasoning with respect to the arcs and the strongly connected components.

[b] Given a digraph $\mathcal{G}$ with p vertices, a *Hamiltonian cycle* of $\mathcal{G}$ is a succession of arcs $v_1 \mapsto v_2, v_2 \mapsto v_3, \cdots, v_{p-1} \mapsto v_p, v_p \mapsto v_1$ of $\mathcal{G}$ such that the vertices $v_1, v_2, \cdots, v_p$ are all distinct.

EXERCISE 4 (variable-based degree of violation)[a]

A. Compute the variable-based degree of violation[b] of the following constraints:

(a) $\textsc{cycle}(4, \langle 1\ 2, 2\ 3, 3\ 1, 4\ 4\rangle)$,

(b) $\textsc{cycle}(1, \langle 1\ 3, 2\ 4, 3\ 3, 4\ 4\rangle)$,

(c) $\textsc{cycle}(6, \langle 1\ 2, 2\ 2, 3\ 4, 4\ 4, 5\ 6, 6\ 5\rangle)$.

B. Give a formula for evaluating the variable-based degree of violation of any ground instance of the CYCLE constraint.

[a]Hint: focus first on the basic constraint on the successor variables, then on the first argument of CYCLE.

[b]Given a constraint for which all variables are fixed, the *variable-based degree of violation* is the minimum number of variables to assign differently in order to satisfy the constraint.

EXERCISE 5 (De Bruijn sequence)[a]

Given an alphabet $A = \{0, 1, \ldots, n-1\}$ and an integer $m > 0$ the corresponding *De Bruijn digraph* $\mathcal{G}_m^n = (V, E)$ of order m is defined as follows:

- The set of vertices V consist of every potential word of length m over the alphabet A.
- The set E contains all arcs $w_1 \mapsto w_2$ where w_1 and w_2 are words of length m over the alphabet A such that the last $m-1$ letters of w_1 coincide with the first $m-1$ first letters of w_2.

Given an alphabet $A = \{0, 1, \ldots, n-1\}$ and an integer $m > 0$ a *De Bruijn sequence* s_m^n of order m is a word over the alphabet A such that every word of length m over the alphabet A occurs[b] exactly once in s.

A. Given an alphabet $A = \{0, 1, \ldots, n-1\}$ define a De Bruijn sequence of order m wrt the ***De Bruijn digraph*** of order m defined on the same alphabet A. Illustrate this link on the De Bruijn sequence $0\ 1\ 0\ 1\ 1\ 1\ 0\ 0$ when $n = 2$, $m = 3$ and $A = \{0, 1\}$.

B. Based on the previous correspondence give a compact model for De Bruijn sequences of order m that uses a single CYCLE constraint.

[a]Hint: define the vertices of the De Bruijn digraph $\mathcal{G}_3^2$, define the arcs of $\mathcal{G}_3^2$, search a pattern on $\mathcal{G}_3^2$ corresponding to a De Bruijn sequence.

[b]A word $w = w_0 w_1 \cdots w_{m-1}$ *occurs* in a sequence $s = s_0 s_1 \cdots s_{p-1}$ ($p \geq m$) if there exists a position i ($0 \leq i < p$) such that $w_0 = s_i, w_1 = s_{(i+1) \bmod p}, \cdots, w_{m-1} = s_{(i+m-1) \bmod p}$.

SOLUTION TO EXERCISE 1

A. *No, since the successor attributes* 2, 1, 2 *are not all different.*

B. *Yes, since we have two cycles namely* $1 \mapsto 3 \mapsto 1$ *and* $2 \mapsto 2$.

C. *Yes, since we have three cycles namely* $1 \mapsto 1$, $2 \mapsto 2$ *and* $3 \mapsto 3$.

D. *No, since we have three cycles namely* $1 \mapsto 5 \mapsto 1$, $2 \mapsto 4 \mapsto 2$ *and* $3 \mapsto 3$, *rather than two cycles as stated by the first argument of the* CYCLE *constraint.*

SOLUTION TO EXERCISE 2
(*variables of a same cycle are coloured with the same colour*)

the five solutions

$N, \langle 1\ V_1, 2\ V_2, 3\ V_3, 4\ V_4, 5\ V_5, 6\ V_6\rangle$

① $(\mathbf{2}, \langle 1\ \mathbf{1}, 2\ \mathbf{4}, 3\ \mathbf{5}, 4\ \mathbf{6}, 5\ \mathbf{2}, 6\ \mathbf{3}\rangle)$
$\mathbf{1} \mapsto \mathbf{1}$, $\mathbf{2} \mapsto \mathbf{4} \mapsto \mathbf{6} \mapsto \mathbf{3} \mapsto \mathbf{5} \mapsto \mathbf{2}$

② $(\mathbf{2}, \langle 1\ \mathbf{3}, 2\ \mathbf{4}, 3\ \mathbf{5}, 4\ \mathbf{1}, 5\ \mathbf{2}, 6\ \mathbf{6}\rangle)$
$\mathbf{1} \mapsto \mathbf{3} \mapsto \mathbf{5} \mapsto \mathbf{2} \mapsto \mathbf{4} \mapsto \mathbf{1}$, $\mathbf{6} \mapsto \mathbf{6}$

③ $(\mathbf{2}, \langle 1\ \mathbf{5}, 2\ \mathbf{3}, 3\ \mathbf{2}, 4\ \mathbf{1}, 5\ \mathbf{6}, 6\ \mathbf{4}\rangle)$
$\mathbf{1} \mapsto \mathbf{5} \mapsto \mathbf{6} \mapsto \mathbf{4} \mapsto \mathbf{1}$, $\mathbf{2} \mapsto \mathbf{3} \mapsto \mathbf{2}$

④ $(\mathbf{2}, \langle 1\ \mathbf{5}, 2\ \mathbf{4}, 3\ \mathbf{6}, 4\ \mathbf{1}, 5\ \mathbf{2}, 6\ \mathbf{3}\rangle)$
$\mathbf{1} \mapsto \mathbf{5} \mapsto \mathbf{2} \mapsto \mathbf{4} \mapsto \mathbf{1}$, $\mathbf{3} \mapsto \mathbf{6} \mapsto \mathbf{3}$

⑤ $(\mathbf{4}, \langle 1\ \mathbf{1}, 2\ \mathbf{3}, 3\ \mathbf{5}, 4\ \mathbf{4}, 5\ \mathbf{2}, 6\ \mathbf{6}\rangle)$
$\mathbf{1} \mapsto \mathbf{1}$, $\mathbf{2} \mapsto \mathbf{3} \mapsto \mathbf{5} \mapsto \mathbf{2}$, $\mathbf{4} \mapsto \mathbf{4}$, $\mathbf{6} \mapsto \mathbf{6}$

SOLUTION TO EXERCISE 3

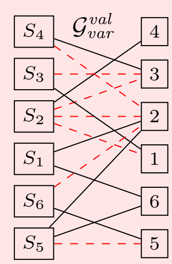
A. *To each vertex v of $\mathcal{G}$ we associate a* successor *variable S_v whose initial domain is set to the labels of the successors of v. Thus we have:*

$$\left\{\begin{array}{lll} S_1 \in \{2,6\}, & S_2 \in \{1,2,3,4\}, & S_3 \in \{1,3\}, \\ S_4 \in \{2,3\}, & S_5 \in \{2,5,6\}, & S_6 \in \{2,5\}. \end{array}\right.$$

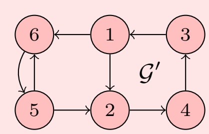
The implicit hypothesis is that, in solutions to the modelled problem, each vertex of the corresponding induced subgraph of $\mathcal{G}$ has exactly one successor.

B. *Since we were asked to have a single cycle we set the first argument of* CYCLE *to* 1 *and obtain* CYCLE$(1, \langle 1\ S_1, 2\ S_2, 3\ S_3, 4\ S_4, 5\ S_5, 6\ S_6\rangle)$.

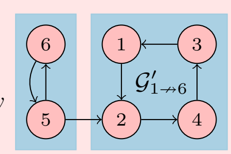
C. *Since there is a single cycle, $S_i \neq i$ (with $i \in [1,6]$). A necessary condition for the* CYCLE *constraint is that all its successor variables are assigned distinct values, i.e. each vertex has exactly one predecessor in a ground solution. Consequently, infeasible variable-value pairs for* ALLDIFFERENT *are also infeasible for* CYCLE*. Any edge that does not belong to a matching of cardinality* 6 *in the corresponding variable-value graph $\mathcal{G}_{var}^{val}$ given on the right can not be part of a solution. As a result $\mathcal{G}'$ is shown below on the right.*



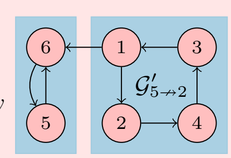
We now deal with the fact that we should have a single cycle. A necessary condition is that the graph $\mathcal{G}'$ consists of a single strongly connected component. We identify the arcs $u \mapsto v$ of $\mathcal{G}'$ such that, if they were removed, the number of strongly connected components of $\mathcal{G}'$ would be greater than one. For such arcs $u \mapsto v$ we remove all arcs $w \mapsto v$ (with $w \neq u$).



- *If we remove $1 \mapsto 6$ from $\mathcal{G}'$ we obtain $\mathcal{G}'_{1 \not\mapsto 6}$, which has the two strongly connected components depicted by the two blue rectangles. Consequently* the arc $5 \mapsto 6$ *is forbidden.*



- *If we remove $5 \mapsto 2$ from $\mathcal{G}'$ we obtain $\mathcal{G}'_{5 \not\mapsto 2}$, which has the two strongly connected components depicted by the two blue rectangles. Consequently the arc $1 \mapsto 2$ is forbidden.*



As a consequence we have a unique solution $S_1 = 6$, $S_2 = 4$, $S_3 = 1$, $S_4 = 3$, $S_5 = 2$, $S_6 = 5$ corresponding to the Hamiltonian cycle $1 \mapsto 6 \mapsto 5 \mapsto 2 \mapsto 4 \mapsto 3 \mapsto 1$.

SOLUTION TO EXERCISE 4

A. (a) *The variable-based degree of violation is equal to* 1 *since the* ALLDIFFERENT *constraint holds and since we just have to correct the number of cycles (we have the two cycles* $1 \mapsto 2 \mapsto 3 \mapsto 1$ *and* $4 \mapsto 4$ *rather than one cycle). Therefore we only need to set the first argument of the* CYCLE *constraint to* 2.

$$\text{CYCLE}(\overset{2}{\mathbf{4}}, \langle 1\ 2, 2\ 3, 3\ 1, 4\ 4\rangle)$$

(b) *Since we have two occurrences of* 3 *and two occurrences of* 4 *in the successor variables the variable-based degree of violation is at least equal to* 2. *Since, as shown below, it is possible the building of a single cycle* $1 \mapsto 3 \mapsto 2 \mapsto 4 \mapsto 1$ *by just changing the assignment of two variables, the variable-based degree of violation is equal to* 2.

$$\text{CYCLE}(1, \langle 1\ 3, 2\ 4, 3\ \overset{2}{\mathbf{3}}, 4\ \overset{1}{\mathbf{4}}\rangle)$$

(c) *Since we have two occurrences of* 2 *and two occurrences of* 4 *in the successor variables the variable-based degree of violation is at least equal to* 2. *Since just changing the values of two successor variables does not allow the building of* 6 *cycles the variable-based degree of violation is at least equal to* 3. *It is equal to* 3 *as shown by the following assignment that corresponds to the three cycles* $1 \mapsto 2 \mapsto 1$, $3 \mapsto 4 \mapsto 3$, $5 \mapsto 6 \mapsto 5$.

$$\text{CYCLE}(\overset{3}{\mathbf{6}}, \langle 1\ 2, 2\ \overset{1}{\mathbf{2}}, 3\ 4, 4\ \overset{3}{\mathbf{4}}, 5\ 6, 6\ 5\rangle)$$

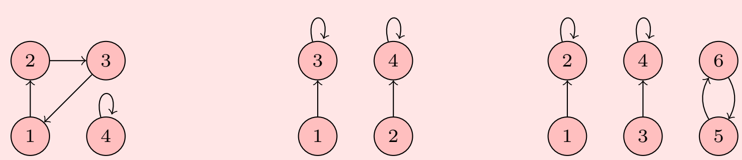
B. *Within the graph associated with the* CYCLE *constraint let* $ncycle$, $nmap$ *and* $nsource$ *respectively denote the number of connected components corresponding to a single cycle, the number of connected components with at least one source, and the number of sources.*

Given `N` *the first argument of the* CYCLE *constraint the variable-based degree of violation is equal to* $nsource + \delta$ *where* δ *is equal to* 0 *if* $\mathtt{N} \in [ncycle + (nmap > 0), ncycle + nmap]$ *and* 1 *otherwise. The idea is that we have to change at least* $nsource$ *successor variables to fulfil the* ALLDIFFERENT *constraint, and possibly the first argument* `N` *if we can not reach* `N` *cycles by just changing* $nsource$ *successor variables. The figures below illustrate the formula for the three examples of the previous question:*

(a) *We have* $0 + 4 \notin [2 + (0 > 0), 2 + 0] = 1$,

(b) *We have* $2 + 1 \notin [0 + (2 > 0), 0 + 2] = 2$,

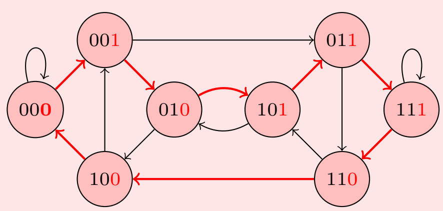
(c) *We have* $2 + 6 \notin [1 + (2 > 0), 1 + 2] = 3$.



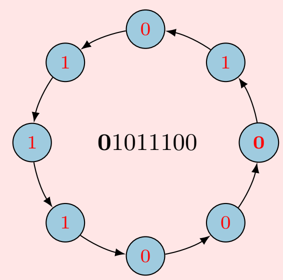
(a)	(b)	(c)
$ncycle = 2$	$ncycle = 0$	$ncycle = 1$
$nmap = 0$	$nmap = 2$	$nmap = 2$
$source = 0$	$source = 2$	$source = 2$

SOLUTION TO EXERCISE 5

A. *A De Bruijn sequence of order m over an alphabet $A = \{0, 1, \ldots, n-1\}$ can be seen as a Hamiltonian cycle*[a] *on the De Bruijn digraph of order m defined over the same alphabet A, where the sequence of letters corresponds to the sequence of last letters of the words associated with the successive vertices of the cycle. Visiting once each vertex of the digraph allows the corresponding cyclic sequence to contain exactly once each word of length m of the alphabet A.*



De Bruijn digraph of order 3 over $A = \{0, 1\}$



A De Bruijn sequence of order 3 over $A = \{0, 1\}$

B. *Each vertex of the De Bruijn graph associated with a word w is labelled by the decimal number plus one*[b] *corresponding to w. Then to each vertex of the De Bruijn graph corresponds a successor variable whose initial domain is set to the labels of the successors of v. Finally a* CYCLE *constraint with one cycle is posted.*

$$\left\{\begin{array}{llll} S_1 \in \{1,2\}, & S_2 \in \{3,4\}, & S_3 \in \{5,6\}, & S_4 \in \{7,8\}, \\ S_5 \in \{1,2\}, & S_6 \in \{3,4\}, & S_7 \in \{5,6\}, & S_8 \in \{7,8\}, \\ \multicolumn{4}{l}{\textsc{cycle}(1, \langle 1\ S_1, 2\ S_2, 3\ S_3, 4\ S_4, 5\ S_5, 6\ S_6, 7\ S_7, 8\ S_8 \rangle).} \end{array}\right.$$

A solution corresponds to the sequence $(S_1 - 1) \bmod n$, $(S_2 - 1) \bmod n$, $\cdots$, $(S_8 - 1) \bmod n$.

---

[a] Given a digraph $\mathcal{G}$ with p vertices, a *Hamiltonian cycle* of $\mathcal{G}$ is a succession of arcs $v_1 \mapsto v_2, v_2 \mapsto v_3, \cdots, v_{p-1} \mapsto v_p, v_p \mapsto v_1$ of $\mathcal{G}$ such that the vertices $v_1, v_2, \cdots, v_p$ are all distinct.

[b] $+1$ since, within the CYCLE constraint, vertices are labelled from 1 up to the total number of vertices.

5.106 CYCLE_CARD_ON_PATH

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin CHIP

Constraint $\text{CYCLE\_CARD\_ON\_PATH}(\texttt{NCYCLE}, \texttt{NODES}, \texttt{ATLEAST}, \texttt{ATMOST}, \texttt{PATH\_LEN}, \texttt{VALUES})$

Arguments

NCYCLE	:	dvar
NODES	:	collection(index−int, succ−dvar, colour−dvar)
ATLEAST	:	int
ATMOST	:	int
PATH_LEN	:	int
VALUES	:	collection(val−int)

Restrictions

$\texttt{NCYCLE} \geq 1$
$\texttt{NCYCLE} \leq |\texttt{NODES}|$
required(NODES, [index, succ, colour])
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
distinct(NODES, index)
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$
$\texttt{ATLEAST} \geq 0$
$\texttt{ATLEAST} \leq \texttt{PATH\_LEN}$
$\texttt{ATMOST} \geq \texttt{ATLEAST}$
$\texttt{PATH\_LEN} \geq 0$
$|\texttt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)

Purpose

Consider a digraph G described by the NODES collection. NCYCLE is the number of circuits for covering G in such a way that each vertex belongs to a single circuit. In addition the following constraint must also hold: on each set of PATH_LEN consecutive distinct vertices of each final circuit, the number of vertices for which the attribute colour takes his value in the collection of values VALUES should be located within the range [ATLEAST, ATMOST].

Example

$$\left(\begin{array}{l} 2, \left\langle\begin{array}{lll} \texttt{index}-1 & \texttt{succ}-7 & \texttt{colour}-2, \\ \texttt{index}-2 & \texttt{succ}-4 & \texttt{colour}-3, \\ \texttt{index}-3 & \texttt{succ}-8 & \texttt{colour}-2, \\ \texttt{index}-4 & \texttt{succ}-9 & \texttt{colour}-1, \\ \texttt{index}-5 & \texttt{succ}-1 & \texttt{colour}-2, \\ \texttt{index}-6 & \texttt{succ}-2 & \texttt{colour}-1, \\ \texttt{index}-7 & \texttt{succ}-5 & \texttt{colour}-1, \\ \texttt{index}-8 & \texttt{succ}-6 & \texttt{colour}-1, \\ \texttt{index}-9 & \texttt{succ}-3 & \texttt{colour}-1 \end{array}\right\rangle, 1, 2, 3, \\ \langle 1 \rangle \end{array}\right)$$

Figure 5.254 illustrates the example with its corresponding circuits and all their sliding sequences of three consecutive vertices. The constraint CYCLE_CARD_ON_PATH holds since the vertices of the NODES collection correspond to a set of disjoint circuits and since, for each set of 3 (i.e., PATH_LEN $= 3$) consecutive vertices, colour 1 (i.e., the value provided by the VALUES collection) occurs at least once (i.e., ATLEAST $= 1$) and at most twice (i.e., ATMOST $= 2$).

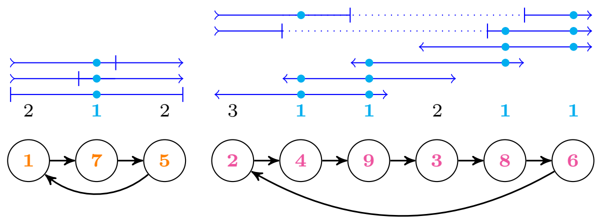


Figure 5.254: The two circuits (bottom) and all the corresponding sliding sequences of three consecutive vertices, where an occurrence of colour 1 is represented by a tiny circle (top) of the **Example** slot

Typical

$|$NODES$| > 2$
NCYCLE $< |$NODES$|$
ATLEAST $<$ PATH_LEN
ATMOST > 0
PATH_LEN > 1
$|$NODES$| > |$VALUES$|$
ATLEAST $> 0 \vee$ ATMOST $<$ PATH_LEN

Symmetries

- Items of NODES are permutable.
- An occurrence of a value of NODES.colour that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).
- ATLEAST can be decreased to any value ≥ 0.
- ATMOST can be increased.
- Items of VALUES are permutable.

Usage

Assume that the vertices of G are partitioned into the following two categories:

- Clients to visit.
- Depots where one can reload a vehicle.

Using the CYCLE_CARD_ON_PATH constraint we can express a constraint like: after visiting three consecutive clients we should visit a depot. This is typically not possible with the ATMOST constraint since we do not know in advance the set of variables involved in the ATMOST constraint.

Remark

This constraint is a special case of the sequence parameter of the CYCLE constraint of **CHIP** [93, pages 121–128].

See also

common keyword: CYCLE *(graph partitioning constraint)*.

used in graph description: AMONG_LOW_UP.

Keywords

characteristic of a constraint: coloured.

combinatorial object: sequence.

constraint type: graph constraint, graph partitioning constraint, sliding sequence constraint.

final graph structure: connected component, one_succ.

Arc input(s)	NODES
Arc generator	*CLIQUE*$\mapsto$collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **NTREE**= 0 • **NCC**= NCYCLE
Graph class	ONE_SUCC
Sets	PATH_LENGTH(PATH_LEN) $\mapsto$ $\left[\ \text{variables} - \text{col}\left(\begin{array}{l}\text{VARIABLES}-\text{collection}(\text{var}-\text{dvar}),\\ [\text{item}(\text{var}-\text{NODES.colour})]\end{array}\right)\ \right]$
Constraint(s) on sets	AMONG_LOW_UP(ATLEAST, ATMOST, variables, VALUES)

Graph model Parts (A) and (B) of Figure 5.255 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NCC** graph property, we show the two connected components of the final graph. The constraint CYCLE_CARD_ON_PATH holds since all the vertices belong to a circuit (i.e., **NTREE** = 0) and since for each set of three consecutive vertices, colour 1 occurs at least once and at most twice (i.e., the AMONG_LOW_UP constraint holds).

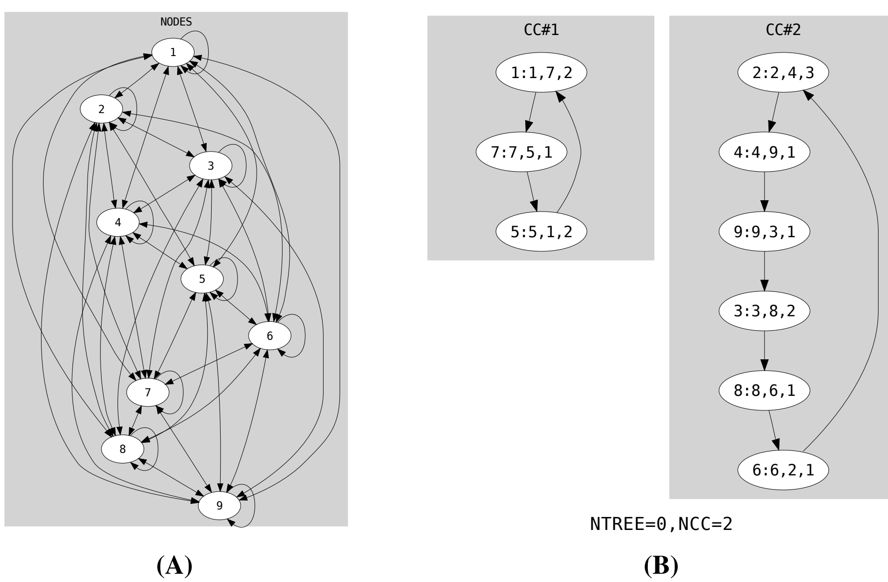


Figure 5.255: Initial and final graph of the CYCLE_CARD_ON_PATH constraint

5.107 CYCLE_OR_ACCESSIBILITY

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Inspired by [255].

Constraint CYCLE_OR_ACCESSIBILITY(MAXDIST, NCYCLE, NODES)

Arguments

MAXDIST	:	int
NCYCLE	:	dvar
NODES	:	collection(index−int, succ−dvar, x−int, y−int)

Restrictions

MAXDIST ≥ 0
NCYCLE ≥ 1
NCYCLE $\leq$ |NODES|
required(NODES, [index, succ, x, y])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 0
NODES.succ $\leq$ |NODES|
NODES.x ≥ 0
NODES.y ≥ 0

Purpose Consider a digraph G described by the NODES collection. Cover a subset of the vertices of G by a set of vertex-disjoint circuits in such a way that the following property holds: for each uncovered vertex v_1 of G there exists at least one covered vertex v_2 of G such that the Manhattan distance between v_1 and v_2 is less than or equal to MAXDIST.

Example

$$\left(3, 2, \left\langle \begin{array}{llll} \texttt{index}-1 & \texttt{succ}-6 & \texttt{x}-4 & \texttt{y}-5, \\ \texttt{index}-2 & \texttt{succ}-0 & \texttt{x}-9 & \texttt{y}-1, \\ \texttt{index}-3 & \texttt{succ}-0 & \texttt{x}-2 & \texttt{y}-4, \\ \texttt{index}-4 & \texttt{succ}-1 & \texttt{x}-2 & \texttt{y}-6, \\ \texttt{index}-5 & \texttt{succ}-5 & \texttt{x}-7 & \texttt{y}-2, \\ \texttt{index}-6 & \texttt{succ}-4 & \texttt{x}-4 & \texttt{y}-7, \\ \texttt{index}-7 & \texttt{succ}-0 & \texttt{x}-6 & \texttt{y}-4 \end{array} \right\rangle \right)$$

Figure 5.256 represents the solution associated with the example. The covered vertices are coloured in blue, while the links starting from the uncovered vertices are dashed. The CYCLE_OR_ACCESSIBILITY constraint holds since:

- In the solution we have NCYCLE $= 2$ disjoint circuits.
- All the 3 uncovered nodes are located at a distance that does not exceed MAXDIST $= 3$ from at least one covered node.

Typical

MAXDIST > 0
NCYCLE $<$ |NODES|
|NODES| > 2

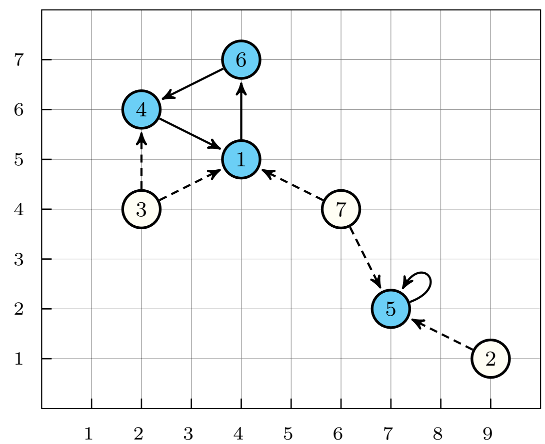


Figure 5.256: Final graph associated with the facilities location problem

Symmetries

- Items of `NODES` are permutable.
- Attributes of `NODES` are permutable w.r.t. permutation (`index`) (`succ`) (`x`, `y`) (*permutation applied to all items*).
- One and the same constant can be added to the `x` attribute of all items of `NODES`.
- One and the same constant can be added to the `y` attribute of all items of `NODES`.

Arg. properties

Functional dependency: `NCYCLE` determined by `NODES`.

Remark

This kind of facilities location problem is described in [255, pages 187–189] pages. In addition to our example they also mention the cost problem that is usually a trade-off between the vertices that are directly covered by circuits and the others.

See also

common keyword: CYCLE *(graph constraint)*.

used in graph description: NVALUES_EXCEPT_0.

Keywords

constraint type: graph constraint.

final graph structure: strongly connected component.

geometry: geometrical constraint.

modelling: functional dependency.

problems: facilities location problem.

Arc input(s) NODES

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{nodes1}, \texttt{nodes2})$

Arc arity 2

Arc constraint(s) $\texttt{nodes1.succ} = \texttt{nodes2.index}$

Graph property(ies)
- **NTREE** $= 0$
- **NCC** $=$ NCYCLE

Arc input(s) NODES

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{nodes1}, \texttt{nodes2})$

Arc arity 2

Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\texttt{nodes1.succ} = \texttt{nodes2.index},\\ \bigwedge\left(\begin{array}{l}\texttt{nodes1.succ} = 0,\\ \texttt{nodes2.succ} \neq 0,\\ \texttt{abs}(\texttt{nodes1.x} - \texttt{nodes2.x}) + \texttt{abs}(\texttt{nodes1.y} - \texttt{nodes2.y}) \leq \texttt{MAXDIST}\end{array}\right)\end{array}\right)$$

Graph property(ies) **NVERTEX** $= |\texttt{NODES}|$

Sets

$$\texttt{PRED} \mapsto \left[\begin{array}{l}\texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var} - \texttt{NODES.succ})]\end{array}\right),\\ \texttt{destination}\end{array}\right]$$

Constraint(s) on sets $\text{NVALUES\_EXCEPT\_0}(\texttt{variables}, =, 1)$

Graph model For each vertex v we have introduced the following attributes:

- index: the label associated with v,
- succ: if v is not covered by a circuit then 0; If v is covered by a circuit then index of the successor of v.
- x: the x-coordinate of v,
- y: the y-coordinate of v.

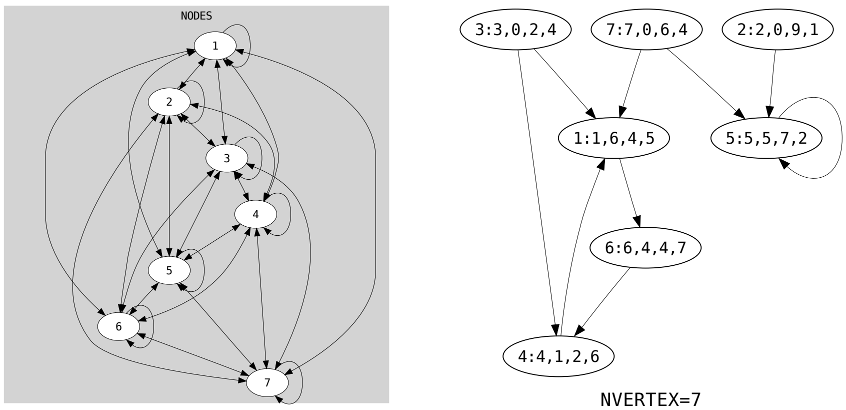
The first graph constraint forces all vertices, which have a non-zero successor, to form a set of NCYCLE vertex-disjoint circuits.

The final graph associated with the second graph constraint contains two types of arcs:

- The arcs belonging to one circuit (i.e., $\texttt{nodes1.succ} = \texttt{nodes2.index}$),
- The arcs between one vertex v_1 that does not belong to any circuit (i.e., $\texttt{nodes1.succ} = 0$) and one vertex v_2 located on a circuit (i.e., $\texttt{nodes2.succ} \neq 0$) such that the Manhattan distance between v_1 and v_2 is less than or equal to MAXDIST.

In order to specify the fact that each vertex is involved in at least one arc we use the graph property **NVERTEX** $= |\texttt{NODES}|$. Finally the dynamic constraint $\texttt{nvalues\_except\_0}(\texttt{variables}, =, 1)$ expresses the fact that, for each vertex v, there is exactly one predecessor of v that belongs to a circuit.

Parts (A) and (B) of Figure 5.257 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot.



(A) **(B)**

Figure 5.257: Initial and final graph of the CYCLE_OR_ACCESSIBILITY constraint

Signature Since |NODES| is the maximum number of vertices of the final graph associated with the second graph constraint we can rewrite $\mathbf{NVERTEX} = |\texttt{NODES}|$ to $\mathbf{NVERTEX} \geq |\texttt{NODES}|$. This leads to simplify $\underline{\mathbf{NVERTEX}}$ to $\overline{\mathbf{NVERTEX}}$.

5.108 CYCLE_RESOURCE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin CHIP

Constraint CYCLE_RESOURCE(RESOURCE, TASK)

Arguments

RESOURCE : collection(id−int, first_task−dvar, nb_task−dvar)
TASK : collection(id−int, next_task−dvar, resource−dvar)

Restrictions

required(RESOURCE, [id, first_task, nb_task])
RESOURCE.id ≥ 1
RESOURCE.id $\leq$ |RESOURCE|
distinct(RESOURCE, id)
RESOURCE.first_task ≥ 1
RESOURCE.first_task $\leq$ |RESOURCE| + |TASK|
RESOURCE.nb_task ≥ 0
RESOURCE.nb_task $\leq$ |TASK|
required(TASK, [id, next_task, resource])
TASK.id > |RESOURCE|
TASK.id $\leq$ |RESOURCE| + |TASK|
distinct(TASK, id)
TASK.next_task ≥ 1
TASK.next_task $\leq$ |RESOURCE| + |TASK|
TASK.resource ≥ 1
TASK.resource $\leq$ |RESOURCE|

Purpose

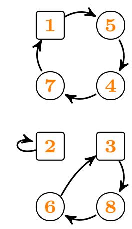
Consider a digraph G defined as follows:

- To each item of the RESOURCE and TASK collections corresponds one vertex of G. A vertex that was generated from an item of the RESOURCE (respectively TASK) collection is called a *resource* vertex (respectively *task* vertex).
- There is an arc from a resource vertex r to a task vertex t if $t \in$ RESOURCE[r].first_task.
- There is an arc from a task vertex t to a resource vertex r if $r \in$ TASK[t].next_task.
- There is an arc from a task vertex t_1 to a task vertex t_2 if $t_2 \in$ TASK[t_1].next_task.
- There is no arc between two resource vertices.

Enforce to cover G in such a way that each vertex belongs to a single circuit. Each circuit is made up from a single *resource* vertex and zero, one or more *task* vertices. For each resource-vertex a domain variable indicates how many task-vertices belong to the corresponding circuit. For each task a domain variable provides the identifier of the resource that can effectively handle that task.

Example

$$\left(\begin{array}{l} \left\langle\begin{array}{lll} \texttt{id}-1 & \texttt{first\_task}-5 & \texttt{nb\_task}-3, \\ \texttt{id}-2 & \texttt{first\_task}-2 & \texttt{nb\_task}-0, \\ \texttt{id}-3 & \texttt{first\_task}-8 & \texttt{nb\_task}-2 \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \texttt{id}-4 & \texttt{next\_task}-7 & \texttt{resource}-1, \\ \texttt{id}-5 & \texttt{next\_task}-4 & \texttt{resource}-1, \\ \texttt{id}-6 & \texttt{next\_task}-3 & \texttt{resource}-3, \\ \texttt{id}-7 & \texttt{next\_task}-1 & \texttt{resource}-1, \\ \texttt{id}-8 & \texttt{next\_task}-6 & \texttt{resource}-3 \end{array}\right\rangle \end{array}\right)$$



The CYCLE_RESOURCE constraint holds since the graph corresponding to the vertices described by its arguments consists of the following 3 disjoint circuits:

- The first circuit involves the *resource* vertex 1 as well as the *task* vertices 5, 4 and 7.
- The second circuit is limited to the *resource* vertex 2.
- Finally the third circuit is made up from the remaining vertices, namely the *resource* vertex 3 and the *task* vertices 8 and 6.

Typical

$|\texttt{RESOURCE}| > 1$
$|\texttt{TASK}| > 1$
$|\texttt{TASK}| > |\texttt{RESOURCE}|$

Symmetries

- Items of RESOURCE are permutable.
- Items of TASK are permutable.
- All occurrences of two distinct values in RESOURCE.id or TASK.resource can be swapped.

Usage

This constraint is useful for some vehicles routing problem where the number of locations to visit depends of the vehicle type that is actually used. The resource attribute allows expressing various constraints such as:

- The compatibility or incompatibility between tasks and vehicles,
- The fact that certain tasks should be performed by the same vehicle,
- The pre-assignment of certain tasks to a given vehicle.

Remark

This constraint could be expressed with the CYCLE constraint of **CHIP** by using the following optional parameters:

- The *resource node* parameter [93, page 97],
- The *circuit weight* parameter [93, page 101],
- The *name* parameter [93, page 104].

See also

common keyword: CYCLE *(graph partitioning constraint)*.

Keywords

characteristic of a constraint: derived collection.

constraint type: graph constraint, resource constraint, graph partitioning constraint.

final graph structure: connected component, strongly connected component.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{RESOURCE\_TASK}-\texttt{collection}\left(\begin{array}{l}\texttt{index}-\texttt{int},\\ \texttt{succ}-\texttt{dvar},\\ \texttt{name}-\texttt{dvar}\end{array}\right),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{RESOURCE.id},\\ \texttt{succ}-\texttt{RESOURCE.first\_task},\\ \texttt{name}-\texttt{RESOURCE.id}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{TASK.id},\\ \texttt{succ}-\texttt{TASK.next\_task},\\ \texttt{name}-\texttt{TASK.resource}\end{array}\right)\end{array}\right]\end{array}\right)$$

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

Arc arity 2

Arc constraint(s)
- resource_task1.succ = resource_task2.index
- resource_task1.name = resource_task2.name

Graph property(ies)
- **NTREE**$= 0$
- **NCC**$= |\texttt{RESOURCE}|$
- **NVERTEX**$= |\texttt{RESOURCE}| + |\texttt{TASK}|$

Graph class ONE_SUCC

For all items of RESOURCE:

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

Arc arity 2

Arc constraint(s)
- resource_task1.succ = resource_task2.index
- resource_task1.name = resource_task2.name
- resource_task1.name = RESOURCE.id

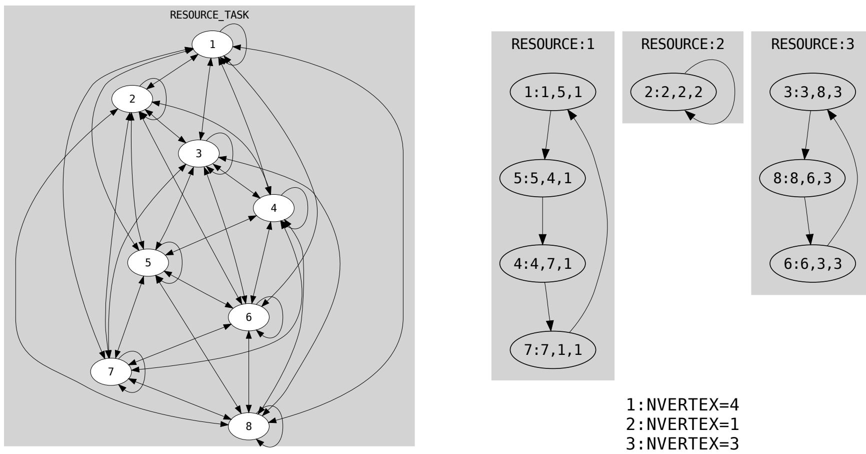
Graph property(ies) **NVERTEX**$= \texttt{RESOURCE.nb\_task} + 1$

Graph model The graph model of the CYCLE_RESOURCE constraint illustrates the following points:

- How to differentiate the constraint on the length of a circuit according to a resource that is assigned to a circuit? This is achieved by introducing a collection of resources and by asking a different graph property for each item of that collection.
- How to introduce the concept of name that corresponds to the resource that handles a given task? This is done by adding to the arc constraint associated with the CYCLE constraint the condition that the name variables of two consecutive vertices should be equal.

Part (A) of Figure 5.258 shows the initial graphs (of the second graph constraint) associated with resources 1, 2 and 3 of the **Example** slot. Part (B) of Figure 5.258 shows the corresponding final graphs (of the second graph constraint) associated with resources 1, 2 and 3.

Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold. To each resource corresponds a circuit of respectively 3, 0 and 2 task-vertices.



(A) **(B)**

Figure 5.258: Initial and final graph of the CYCLE_RESOURCE constraint

Signature Since the initial graph of the first graph constraint contains $|\texttt{RESOURCE}| + |\texttt{TASK}|$ vertices, the corresponding final graph cannot have more than $|\texttt{RESOURCE}| + |\texttt{TASK}|$ vertices. Therefore we can rewrite the graph property **NVERTEX** $= |\texttt{RESOURCE}| + |\texttt{TASK}|$ to **NVERTEX** $\geq |\texttt{RESOURCE}| + |\texttt{TASK}|$ and simplify $\underline{\overline{\mathbf{NVERTEX}}}$ to $\overline{\mathbf{NVERTEX}}$.

5.109 CYCLIC_CHANGE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from CHANGE.

Constraint CYCLIC_CHANGE(NCHANGE, CYCLE_LENGTH, VARIABLES, CTR)

Arguments

NCHANGE	:	dvar
CYCLE_LENGTH	:	int
VARIABLES	:	collection(var−dvar)
CTR	:	atom

Restrictions

NCHANGE ≥ 0
NCHANGE $<$ |VARIABLES|
CYCLE_LENGTH > 0
required(VARIABLES, var)
VARIABLES.var ≥ 0
VARIABLES.var $<$ CYCLE_LENGTH
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose NCHANGE is the number of times that constraint $((X + 1) \bmod \text{CYCLE\_LENGTH})$ CTR Y holds; X and Y correspond to consecutive variables of the collection VARIABLES.

Example $(2, 4, \langle 3, 0, 2, 3, 1\rangle, \neq)$

Since CTR is set to $\neq$ and since CYCLE_LENGTH is set to 4, a change between two consecutive items X and Y of the VARIABLES collection corresponds to the fact that the condition $((X + 1) \bmod 4) \neq Y$ holds. Consequently, the CYCLIC_CHANGE constraint holds since we have the two following changes (i.e., NCHANGE $= 2$) within $\langle 3, 0, 2, 3, 1\rangle$:

- A first change between the consecutive values 0 and 2,
- A second change between the consecutive values 3 and 1.

However, the sequence 3 0 does not correspond to a change since $(3 + 1) \bmod 4$ is equal to 0.

Typical

NCHANGE > 0
|VARIABLES| > 1
range(VARIABLES.var) > 1
CTR $\in [\neq]$

Symmetry Items of VARIABLES can be shifted.

Arg. properties Functional dependency: NCHANGE determined by CYCLE_LENGTH, VARIABLES and CTR.

Usage

This constraint may be used for personnel cyclic timetabling problems where each person has to work according to cycles. In this context each variable of the VARIABLES collection corresponds to the type of work a person performs on a specific day. Because of some perturbation (e.g., illness, unavailability, variation of the workload) it is in practice not reasonable to ask for perfect cyclic solutions. One alternative is to use the CYCLIC_CHANGE constraint and to ask for solutions where one tries to minimise the number of cycle breaks (i.e., the variable NCHANGE).

See also

common keyword: CHANGE, CYCLIC_CHANGE_JOKER *(number of changes)*.

implies: CYCLIC_CHANGE_JOKER.

Keywords

characteristic of a constraint: cyclic, automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

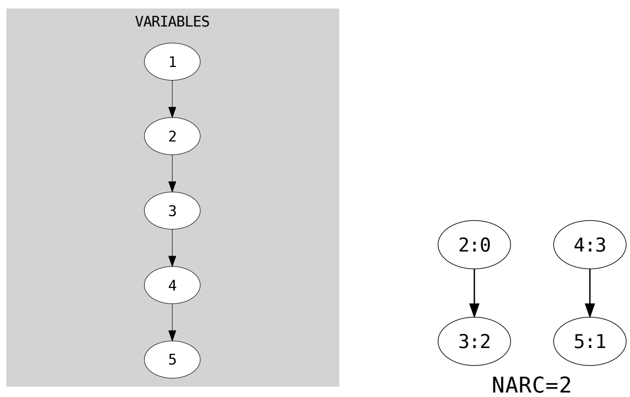
constraint type: timetabling constraint.

final graph structure: acyclic, bipartite, no loop.

modelling: number of changes, functional dependency.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	(variables1.var + 1) mod CYCLE_LENGTH CTR variables2.var
Graph property(ies)	**NARC** = NCHANGE
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.259 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.259: Initial and final graph of the CYCLIC_CHANGE constraint

Automaton Figure 5.260 depicts the automaton associated with the CYCLIC_CHANGE constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $((\mathtt{VAR}_i + 1) \bmod \mathtt{CYCLE\_LENGTH})\ \mathtt{CTR}\ \mathtt{VAR}_{i+1} \Leftrightarrow S_i$.

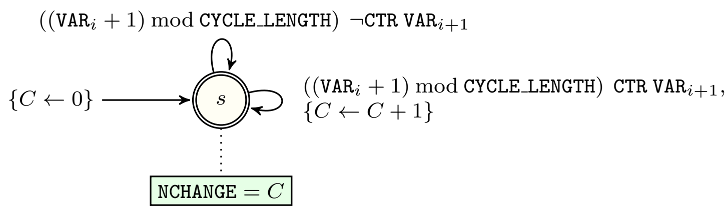


Figure 5.260: Automaton of the CYCLIC_CHANGE constraint

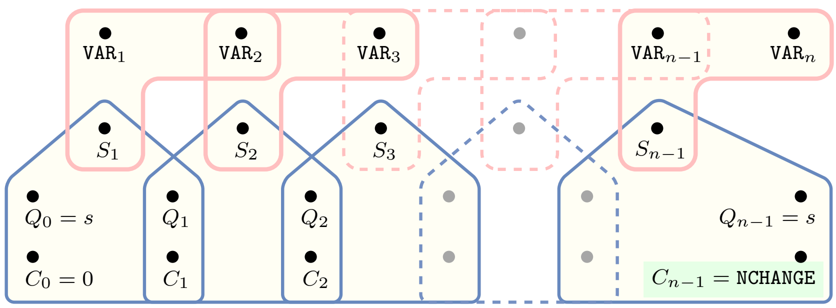


Figure 5.261: Hypergraph of the reformulation corresponding to the automaton of the CYCLIC_CHANGE constraint

5.110 CYCLIC_CHANGE_JOKER

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from CYCLIC_CHANGE.

Constraint CYCLIC_CHANGE_JOKER(NCHANGE, CYCLE_LENGTH, VARIABLES, CTR)

Arguments

NCHANGE	:	dvar
CYCLE_LENGTH	:	int
VARIABLES	:	collection(var−dvar)
CTR	:	atom

Restrictions

NCHANGE ≥ 0
NCHANGE $<$ |VARIABLES|
CYCLE_LENGTH > 0
required(VARIABLES, var)
VARIABLES.var ≥ 0
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

NCHANGE is the number of times that the following constraint holds:

$((X+1) \bmod \texttt{CYCLE\_LENGTH})\ \texttt{CTR}\ Y \wedge X < \texttt{CYCLE\_LENGTH} \wedge Y < \texttt{CYCLE\_LENGTH}$

X and Y correspond to consecutive variables of the collection VARIABLES.

Example

$(2, 4, \langle 3, 0, 2, 4, 4, 4, 3, 1, 4\rangle, \neq)$

Since CTR is set to $\neq$ and since CYCLE_LENGTH is set to 4, a change between two consecutive items X and Y of the VARIABLES collection corresponds to the fact that the condition $((X+1) \bmod 4) \neq Y \wedge X < 4 \wedge Y < 4$ holds. Consequently, the CYCLIC_CHANGE_JOKER constraint holds since we have the two following changes (i.e., NCHANGE $= 2$) within $\langle 3, 0, 2, 4, 4, 4, 3, 1, 4\rangle$:

- A first change between 0 and 2,
- A second change between 3 and 1.

But when the joker value 4 is involved, there is no change. This is why no change is counted between values 2 and 4, between 4 and 4 and between 1 and 4.

Typical

NCHANGE > 0
CYCLE_LENGTH > 1
|VARIABLES| > 1
range(VARIABLES.var) > 1
maxval(VARIABLES.var) $\geq$ CYCLE_LENGTH
CTR $\in [\neq]$

Typical model ATLEAST$(2,$ VARIABLES$, 0)$

Symmetry Items of VARIABLES can be shifted.

Arg. properties Functional dependency: NCHANGE determined by CYCLE_LENGTH, VARIABLES and CTR.

Usage The CYCLIC_CHANGE_JOKER constraint can be used in the same context as the CYCLIC_CHANGE constraint with the additional feature: in our example codes 0 to 3 correspond to different type of activities (i.e., working the morning, the afternoon or the night) and code 4 represents a holiday. We want to express the fact that we do not count any change for two consecutive days d_1, d_2 such that d_1 or d_2 is a holiday.

See also **common keyword:** CHANGE, CYCLIC_CHANGE *(number of changes)*.

implied by: CYCLIC_CHANGE.

Keywords **characteristic of a constraint:** cyclic, joker value, automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

constraint type: timetabling constraint.

final graph structure: acyclic, bipartite, no loop.

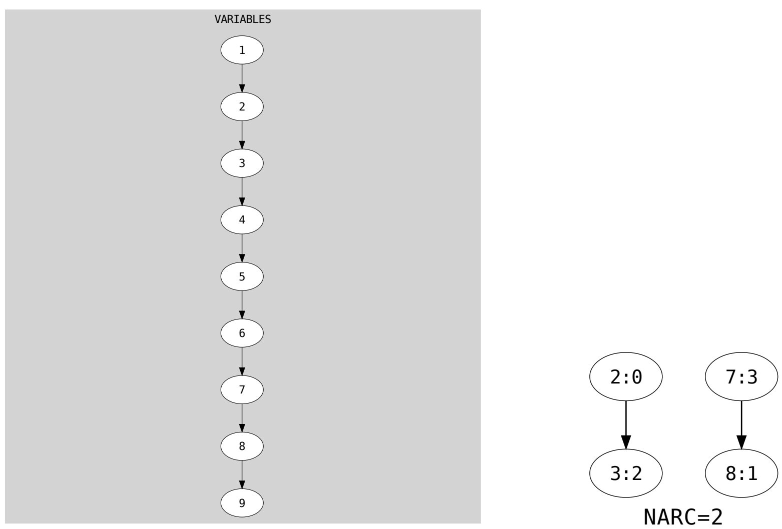
modelling: number of changes, functional dependency.

Arc input(s)	VARIABLES
Arc generator	*PATH* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• (variables1.var + 1) mod CYCLE_LENGTH CTR variables2.var • variables1.var < CYCLE_LENGTH • variables2.var < CYCLE_LENGTH
Graph property(ies)	**NARC** = NCHANGE
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model

The *joker values* are those values that are greater than or equal to CYCLE_LENGTH. We do not count any change for those arc constraints involving at least one variable taking a joker value.

Parts (A) and (B) of Figure 5.262 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) (B)

Figure 5.262: Initial and final graph of the CYCLIC_CHANGE_JOKER constraint

Automaton Figure 5.263 depicts the automaton associated with the CYCLIC_CHANGE_JOKER constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i:

$$(((\mathtt{VAR}_i + 1) \bmod \mathtt{CYCLE\_LENGTH})\ \mathtt{CTR}\ \mathtt{VAR}_{i+1} \wedge$$
$$(\mathtt{VAR}_i < \mathtt{CYCLE\_LENGTH}) \wedge (\mathtt{VAR}_{i+1} < \mathtt{CYCLE\_LENGTH})) \Leftrightarrow S_i.$$

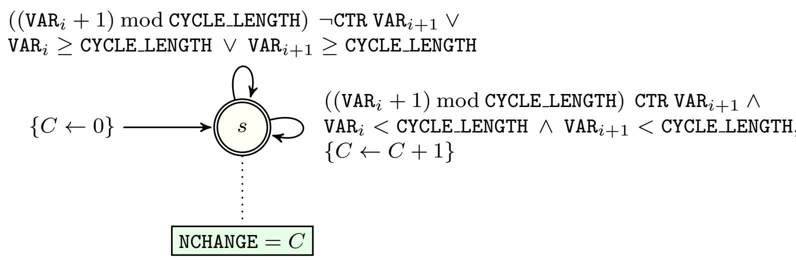


Figure 5.263: Automaton of the CYCLIC_CHANGE_JOKER constraint

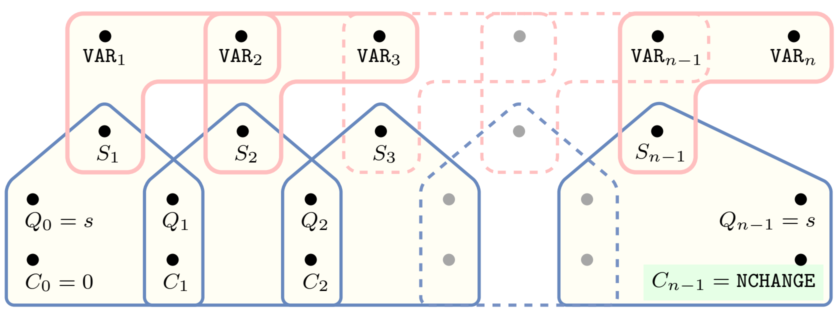


Figure 5.264: Hypergraph of the reformulation corresponding to the automaton of the CYCLIC_CHANGE_JOKER constraint

5.111 DAG

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [151]

Constraint $\text{DAG}(\texttt{NODES})$

Argument $\texttt{NODES} : \texttt{collection}(\texttt{index}-\texttt{int}, \texttt{succ}-\texttt{svar})$

Restrictions

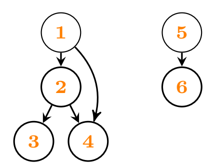
$\texttt{required}(\texttt{NODES}, [\texttt{index}, \texttt{succ}])$
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
$\texttt{distinct}(\texttt{NODES}, \texttt{index})$
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$

Purpose

Consider a digraph G described by the NODES collection. Select a subset of arcs of G so that the corresponding graph does not contain any circuit.

Example

$$\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\{2,4\}, \\ \texttt{index}-2 & \texttt{succ}-\{3,4\}, \\ \texttt{index}-3 & \texttt{succ}-\emptyset, \\ \texttt{index}-4 & \texttt{succ}-\emptyset, \\ \texttt{index}-5 & \texttt{succ}-\{6\}, \\ \texttt{index}-6 & \texttt{succ}-\emptyset \end{array}\right\rangle\right)$$



The DAG constraint holds since the NODES collection depicts a graph without circuit.

Typical $|\texttt{NODES}| > 2$

Symmetry Items of NODES are permutable.

Algorithm A filtering algorithm for the DAG constraint is given in [151, page 90]. It removes potential arcs that would create a circuit of mandatory arcs.

See also **used in graph description:** IN_SET.

Keywords **constraint arguments:** constraint involving set variables.

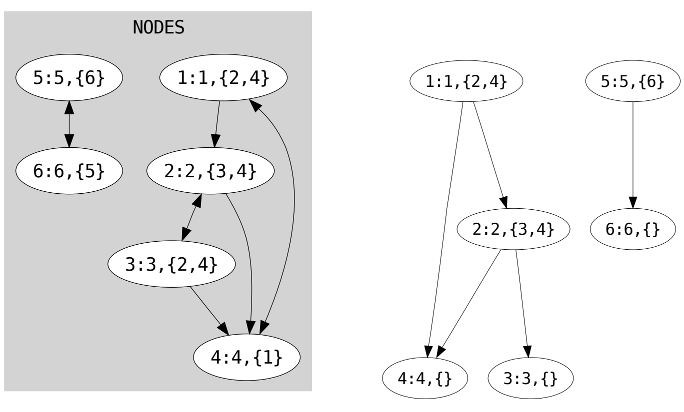
constraint type: graph constraint.

Arc input(s)	NODES
Arc generator	$SELF \mapsto$ collection(nodes)
Arc arity	1
Arc constraint(s)	IN_SET(nodes.key, nodes.succ)
Graph property(ies)	**NARC** $= 0$

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.succ)
Graph property(ies)	**MAX_NSCC** ≤ 1

Graph model The first graph constraint removes the loop of each vertex. The second graph constraint forbids the creation of circuits involving more than one vertex.

Part (A) of Figure 5.265 shows the initial graph associated with the second graph constraint of the **Example** slot. This initial graph from which we start is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.265 gives the final graph associated with the **Example** slot.



(A) **(B)**

Figure 5.265: Initial and final graph of the DAG set constraint

5.112 DECREASING

DESCRIPTION LINKS GRAPH AUTOMATON

Origin Inspired by INCREASING.

Constraint DECREASING(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restriction required(VARIABLES, var)

Purpose The variables of the collection VARIABLES are decreasing.

Example $(\langle 8, 4, 1, 1\rangle)$

The DECREASING constraint holds since $8 \geq 4 \geq 1 \geq 1$.

Typical $|\texttt{VARIABLES}| > 2$
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

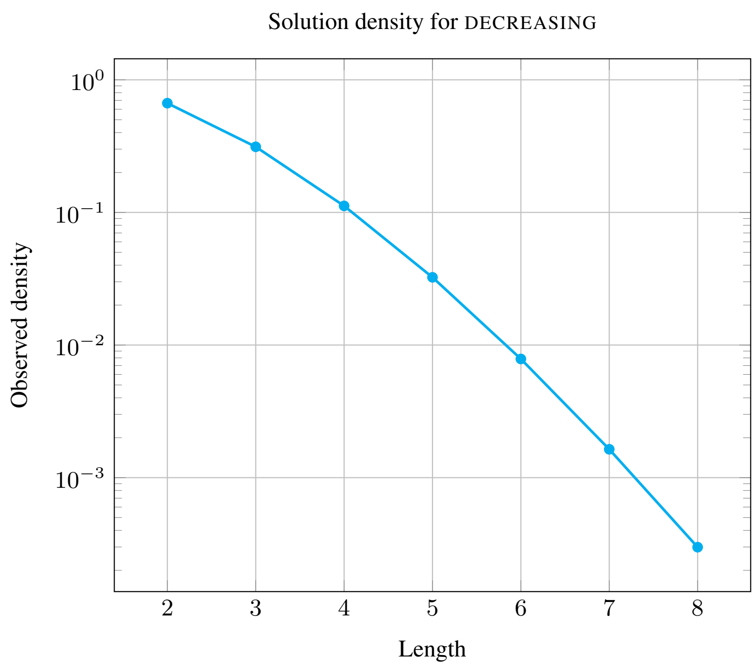
Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

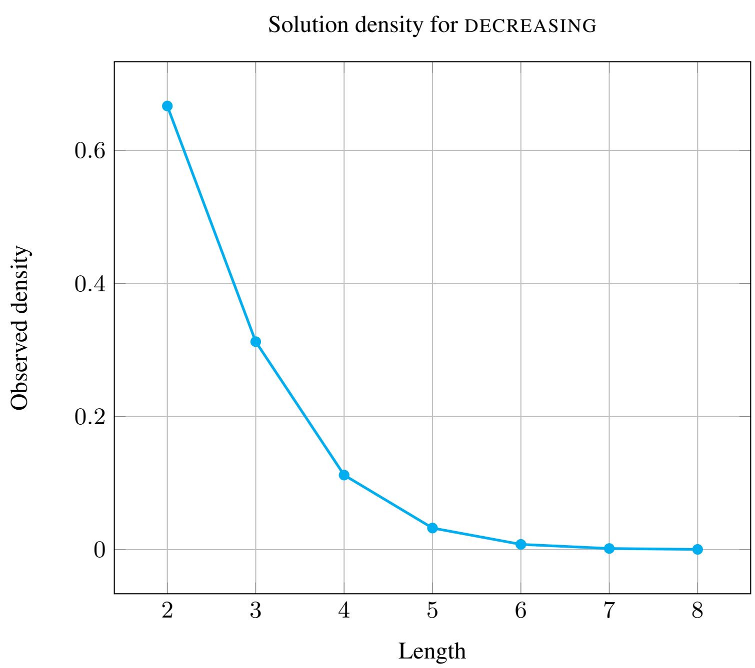
Arg. properties Contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	6	20	70	252	924	3432	12870

Number of solutions for DECREASING: domains $0..n$





Systems INCREASINGNVALUE in **Choco**, REL in **Gecode**, DECREASING in **MiniZinc**.

See also **common keyword:** STRICTLY_INCREASING *(order constraint)*.

comparison swapped: INCREASING.

implied by: ALL_EQUAL, STRICTLY_DECREASING.

implies: MULTI_GLOBAL_CONTIGUITY, NO_PEAK, NO_VALLEY.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: sliding cyclic(1) constraint network(1).

constraint type: decomposition, order constraint.

filtering: arc-consistency.

final graph structure: acyclic, bipartite, no loop.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var $\geq$ variables2.var
Graph property(ies)	**NARC**$=$ $\vert$VARIABLES$\vert - 1$
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.266 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

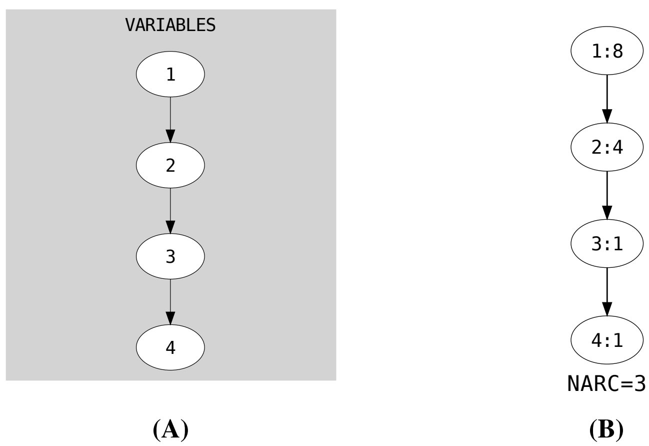


Figure 5.266: Initial and final graph of the DECREASING constraint

Automaton Figure 5.267 depicts the automaton associated with the DECREASING constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $\mathtt{VAR}_i \geq \mathtt{VAR}_{i+1} \Leftrightarrow S_i$.

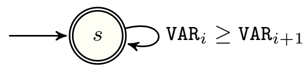


Figure 5.267: Automaton of the DECREASING constraint

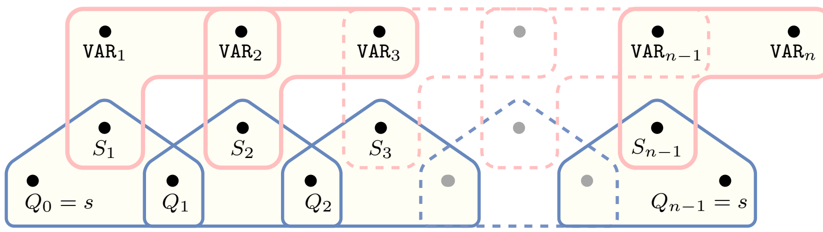


Figure 5.268: Hypergraph of the reformulation corresponding to the automaton of the DECREASING constraint

5.113 DECREASING_PEAK

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Derived from PEAK and DECREASING.

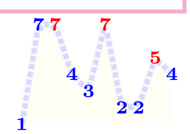
Constraint DECREASING_PEAK(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions $|\texttt{VARIABLES}| > 0$
required(VARIABLES, var)

Purpose A variable V_k $(1 < k < m)$ of the sequence of variables $\texttt{VARIABLES} = V_1, \ldots, V_m$ is a *peak* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k > V_{k+1}$.
When considering all the peaks of the sequence VARIABLES from left to right enforce all peaks to be decreasing, i.e. the altitude of each peak is less than or equal to the altitude of its preceding peak when it exists.

Example $(\langle 1, 7, 7, 4, 3, 7, 2, 2, 5, 4 \rangle)$



The DECREASING_PEAK constraint holds since the sequence 1 7 **7** 4 3 **7** 2 2 **5** 4 contains three peaks, in bold, that are decreasing.

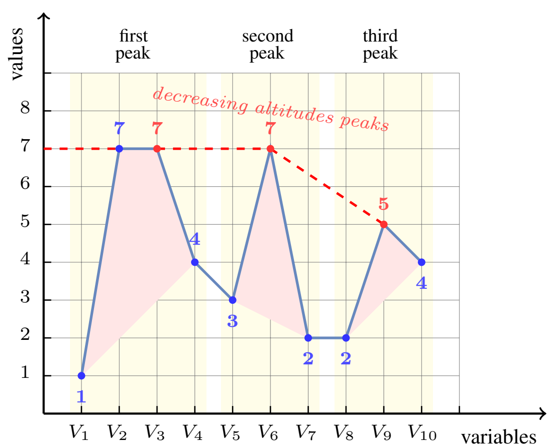


Figure 5.269: Illustration of the **Example** slot: a sequence of ten variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10} respectively fixed to values 1, 7, 7, 4, 3, 7, 2, 2, 5, 4 and its corresponding three peaks, in red, respectively located at altitudes 7, 7 and 5

Typical

$|$VARIABLES$| \geq 7$
range(VARIABLES.var) > 1
PEAK(VARIABLES.var) ≥ 3

Typical model

nval(VARIABLES.var) > 2

Symmetry

One and the same constant can be added to the var attribute of all items of VARIABLES.

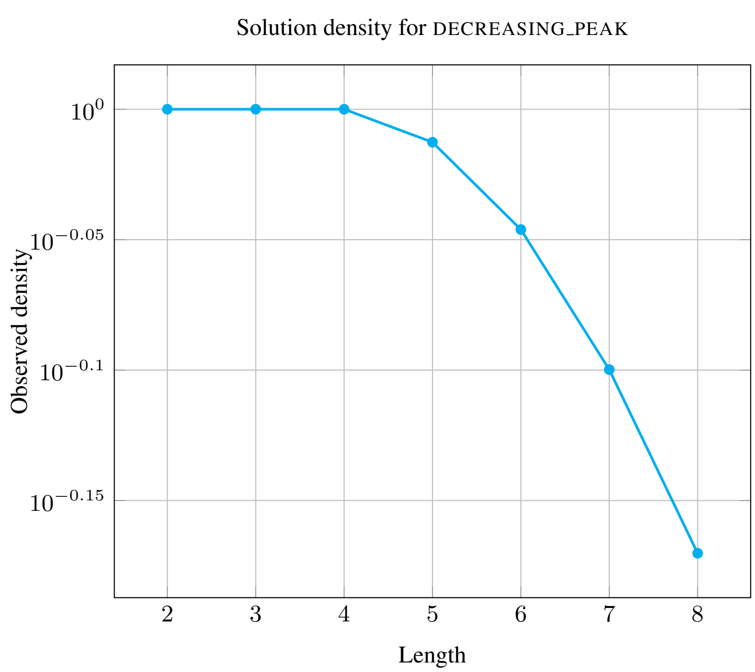
Arg. properties

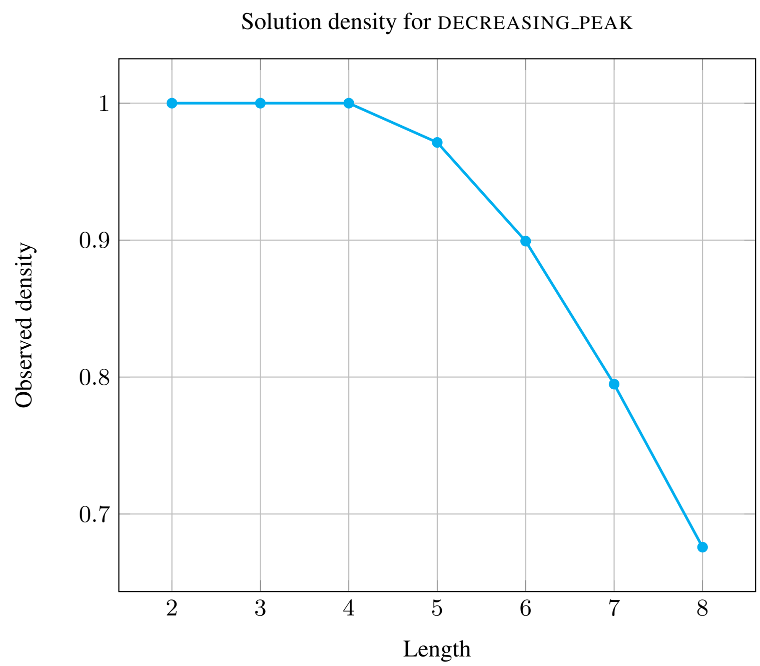
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7553	105798	1666878	29090469

Number of solutions for DECREASING_PEAK: domains $0..n$





See also **implied by:** ALL_EQUAL_PEAK.

related: INCREASING_PEAK, PEAK.

Keywords **characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

Cond. implications DECREASING_PEAK(VARIABLES)
with PEAK(VARIABLES.var) > 0
implies NOT_ALL_EQUAL(VARIABLES).

Automaton Figure 5.270 depicts the automaton associated with the DECREASING_PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: initial stationary or decreasing mode	$(\{= \mid >\}^*)$
u	: increasing (before first potential peak) mode	$(< \{< \mid =\}^*)$
v	: decreasing (after a peak) mode	$(> \{> \mid =\}^*)$
w	: increasing (after a peak) mode	$(< \{< \mid =\}^*)$

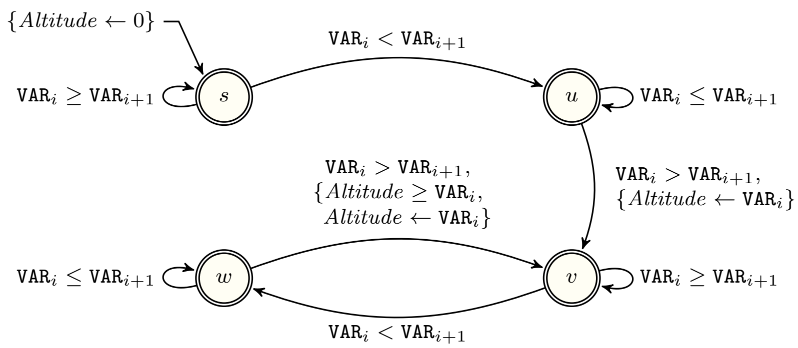


Figure 5.270: Automaton for the DECREASING_PEAK constraint (note the conditional transition from state w to state v testing that the counter $Altitude$ is greater than or equal to $\mathtt{VAR}_i$ for enforcing that all peaks from left to right are in decreasing altitude)

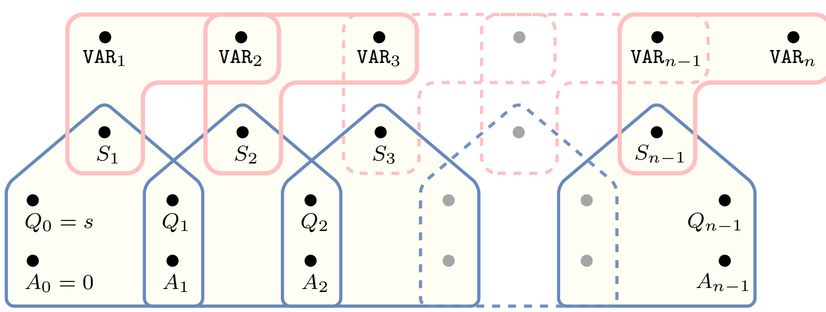


Figure 5.271: Hypergraph of the reformulation corresponding to the automaton of the DECREASING_PEAK constraint where A_i stands for the value of the counter $Altitude$ (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.114 DECREASING_VALLEY

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin

Derived from VALLEY and DECREASING.

Constraint

DECREASING_VALLEY(VARIABLES)

Argument

VARIABLES : collection(var−dvar)

Restrictions

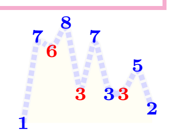
$|\text{VARIABLES}| > 0$
required(VARIABLES, var)

Purpose

A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \ldots, V_m$ is a *valley* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k < V_{k+1}$.
When considering all the valleys of the sequence VARIABLES from left to right enforce all valleys to be decreasing, i.e. the altitude of each valley is less than or equal to the altitude of its preceding valley when it exists.

Example

$(\langle 1, 7, 6, 8, 3, 7, 3, 3, 5, 4\rangle)$



The DECREASING_VALLEY constraint holds since the sequence 1 7 **6** 8 **3** 7 3 **3** 5 2 contains three valleys, in bold, that are decreasing.

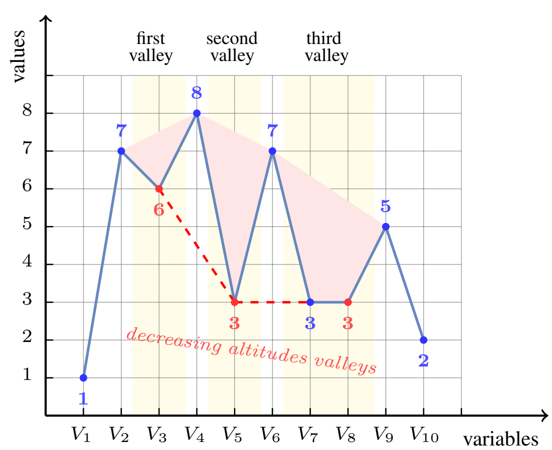


Figure 5.272: Illustration of the **Example** slot: a sequence of ten variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10} respectively fixed to values 1, 7, 6, 8, 3, 7, 3, 3, 5, 2 and its corresponding three valleys, in red, respectively located at altitudes 6, 3 and 3

Typical

$|\texttt{VARIABLES}| \geq 7$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$\text{VALLEY}(\texttt{VARIABLES.var}) \geq 3$

Typical model

$\texttt{nval}(\texttt{VARIABLES.var}) > 2$

Symmetry

One and the same constant can be added to the `var` attribute of all items of `VARIABLES`.

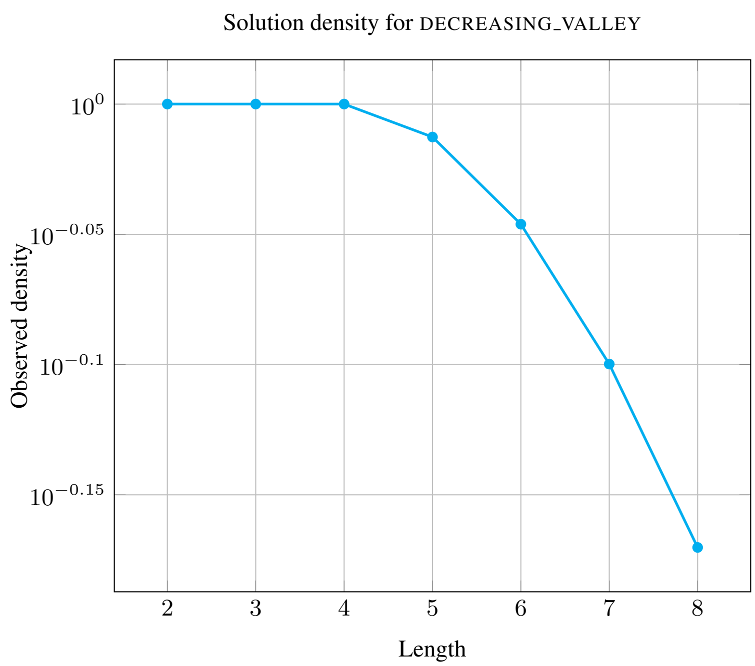
Arg. properties

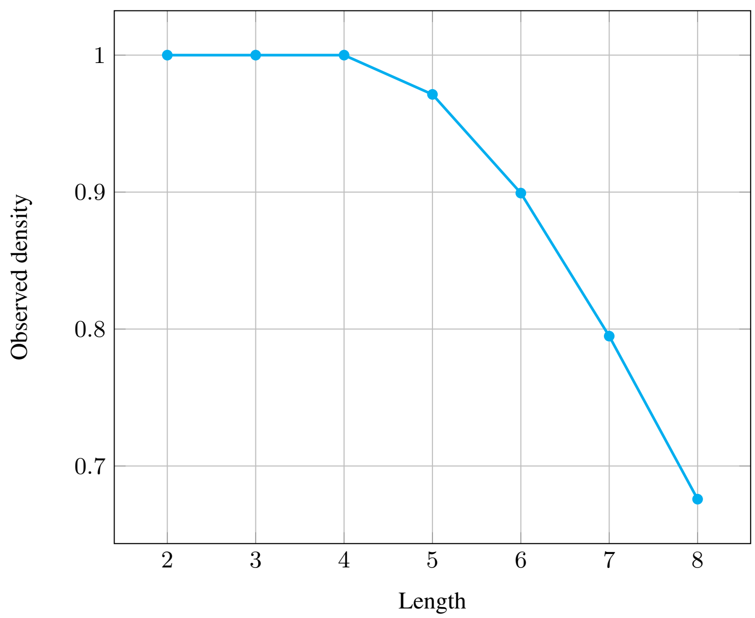
- Prefix-contractible wrt. `VARIABLES`.
- Suffix-contractible wrt. `VARIABLES`.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7553	105798	1666878	29090469

Number of solutions for DECREASING_VALLEY: domains $0..n$





See also **implied by:** ALL_EQUAL_VALLEY.

related: INCREASING_VALLEY, VALLEY.

Keywords **characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

Cond. implications DECREASING_VALLEY(VARIABLES)
with VALLEY(VARIABLES.var) > 0
implies NOT_ALL_EQUAL(VARIABLES).

Automaton Figure 5.273 depicts the automaton associated with the DECREASING_VALLEY constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: initial stationary or increasing mode	$(\{=\mid>\}^*)$
u	: decreasing (before first potential valley) mode	$(<\{<\mid=\}^*)$
v	: increasing (after a valley) mode	$(>\{>\mid=\}^*)$
w	: decreasing (after a valley) mode	$(<\{<\mid=\}^*)$

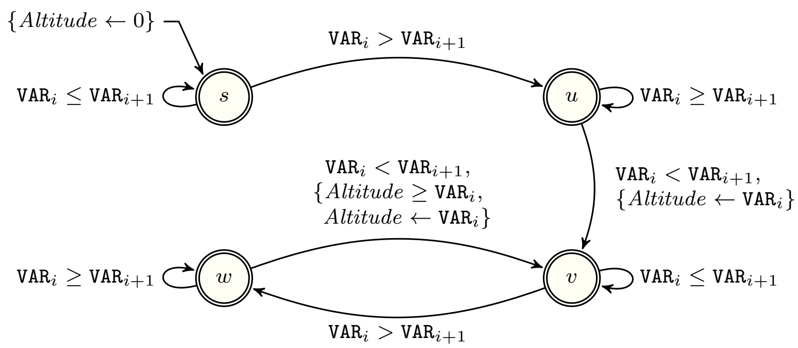


Figure 5.273: Automaton for the DECREASING_VALLEY constraint (note the conditional transition from state w to state v testing that the counter $Altitude$ is greater than or equal to $\mathtt{VAR}_i$ for enforcing that all valleys from left to right are in decreasing altitude)

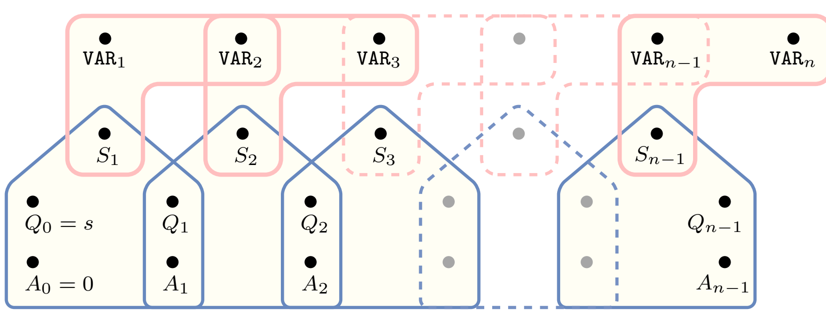


Figure 5.274: Hypergraph of the reformulation corresponding to the automaton of the DECREASING_VALLEY constraint where A_i stands for the value of the counter $Altitude$ (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.115 DEEPEST_VALLEY

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from VALLEY.

Constraint DEEPEST_VALLEY(DEPTH, VARIABLES)

Arguments

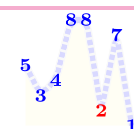
DEPTH	:	dvar
VARIABLES	:	collection(var−dvar)

Restriction required(VARIABLES, var)

Purpose A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \ldots, V_m$ is a *valley* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k < V_{k+1}$. DEPTH is the minimum value of the valley variables. If no such variable exists DEPTH is equal to the default value MAXINT.

Example

$(2, \langle 5, 3, 4, 8, 8, 2, 7, 1\rangle)$
$(7, \langle 1, 3, 4, 8, 8, 8, 7, 8\rangle)$



The first DEEPEST_VALLEY constraint holds since 2 is the deepest valley of the sequence 5 3 4 8 8 2 7 1.

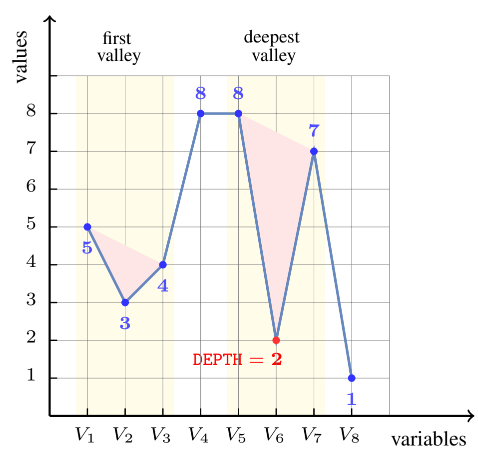


Figure 5.275: Illustration of the first example of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values 5, 3, 4, 8, 8, 2, 7, 1 and its corresponding deepest valley of depth 2

Typical

$|\mathtt{VARIABLES}| > 2$
range(VARIABLES.var) > 2
VALLEY(VARIABLES.var) > 0

Typical model nval(VARIABLES.var) > 2

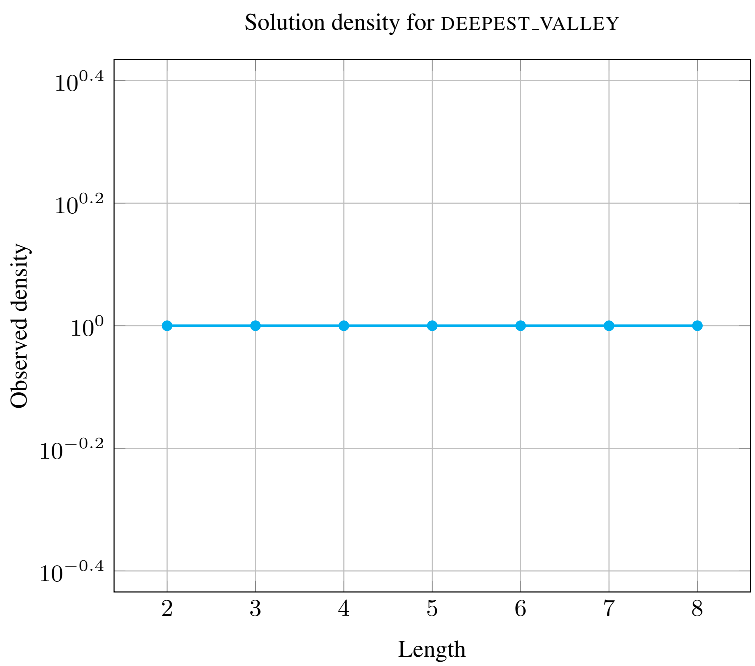
Symmetry Items of VARIABLES can be reversed.

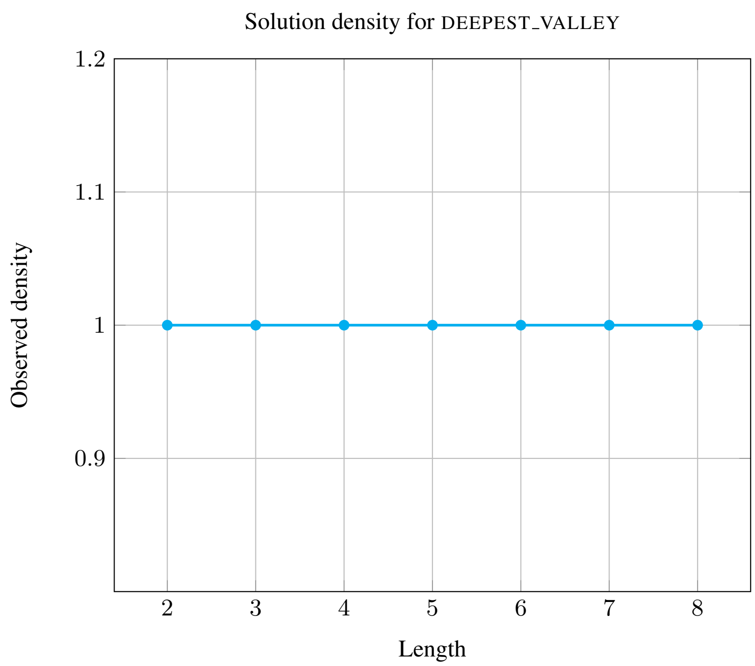
Arg. properties Functional dependency: DEPTH determined by VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for DEEPEST_VALLEY: domains $0..n$

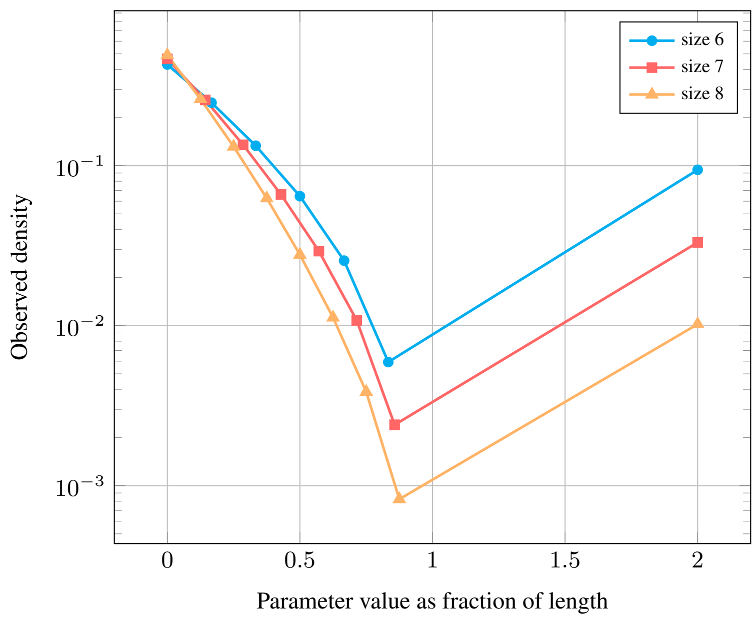




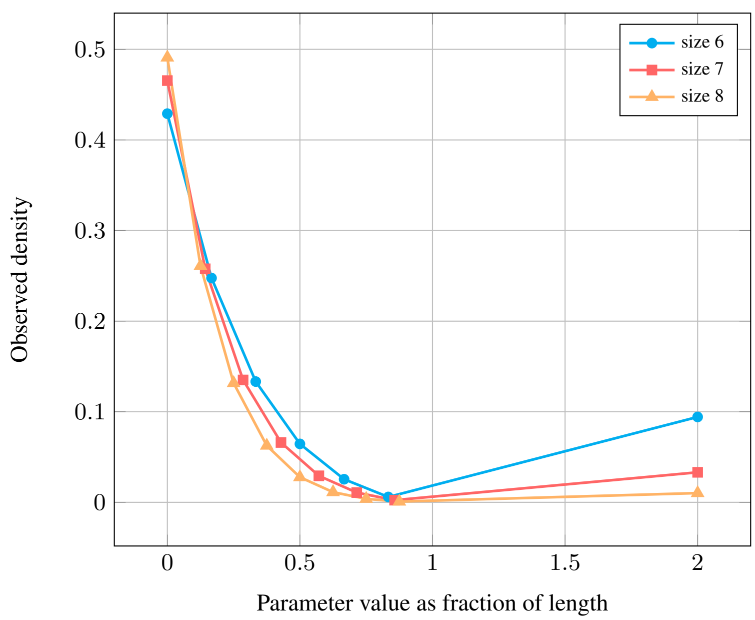
Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	-	9	176	2900	50472	976227	21133632
	1	-	4	99	1712	29125	540576	11233250
	2	-	1	44	900	15680	283250	5665896
	3	-	-	11	380	7587	138544	2693425
	4	-	-	-	92	3000	61389	1195056
	5	-	-	-	-	697	22632	484020
	6	-	-	-	-	-	5036	166208
	7	-	-	-	-	-	-	35443
	1000000	9	50	295	1792	11088	69498	439791

Solution count for DEEPEST_VALLEY: domains $0..n$

Solution density for DEEPEST_VALLEY



Solution density for DEEPEST_VALLEY



See also **common keyword:** HIGHEST_PEAK, VALLEY *(sequence)*.

implies: BETWEEN_MIN_MAX.

Keywords

characteristic of a constraint: maxint, automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

filtering: glue matrix.

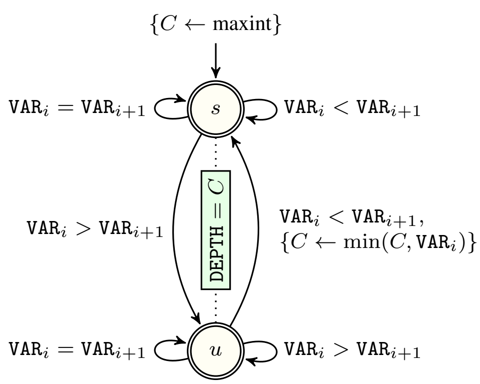
modelling: functional dependency.

Automaton Figure 5.276 depicts the automaton associated with the DEEPEST_VALLEY constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i:

$\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0 \wedge \mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1 \wedge \mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2$.

STATE SEMANTICS

s	: stationary/increasing mode	$(\{< \mid =\}^*)$
u	: decreasing mode	$(> \{> \mid =\}^*)$



Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counters values C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES; $\overrightarrow{X}$ denotes the last variable of the prefix.

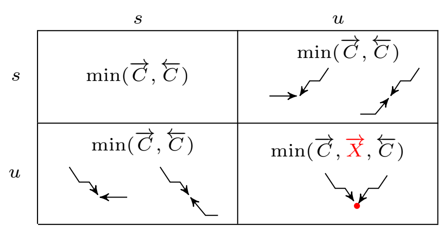


Figure 5.276: Automaton of the DEEPEST_VALLEY constraint and its glue matrix (state s means that we are in *increasing* or *stationary* mode, state u means that we are in *decreasing* mode, a new valley is detected each time we switch from decreasing to increasing mode and the counter C is updated accordingly); maxint is the largest integer that can be represented on a machine

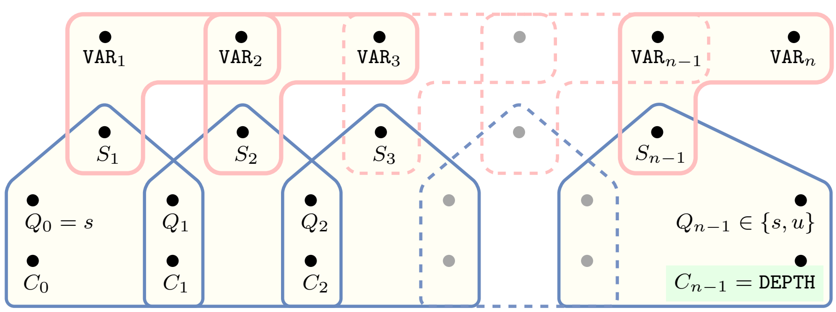


Figure 5.277: Hypergraph of the reformulation corresponding to the automaton of the DEEPEST_VALLEY constraint (C_0 is set to maxint the largest integer that can be represented on a machine)

5.116 DERANGEMENT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin

Derived from CYCLE.

Constraint

DERANGEMENT(NODES)

Argument

NODES : collection(index−int, succ−dvar)

Restrictions

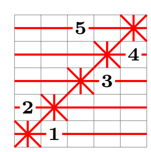
|NODES| > 1
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |NODES|

Purpose

Enforce to have a permutation with no cycle of length one. The permutation is depicted by the succ attribute of the NODES collection.

Example

$$\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-2, \\ \texttt{index}-2 & \texttt{succ}-1, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-3, \\ \texttt{index}-5 & \texttt{succ}-4 \end{array}\right\rangle\right)$$



In the permutation of the example we have the following 2 cycles: $1 \rightarrow 2 \rightarrow 1$ and $3 \rightarrow 5 \rightarrow 4 \rightarrow 3$. Since these cycles have both a length strictly greater than one the corresponding DERANGEMENT constraint holds.

All solutions

Figure 5.278 gives all solutions to the following non ground instance of the DERANGEMENT constraint: $\texttt{S}_1 \in [2,4]$, $\texttt{S}_2 \in [1,2]$, $\texttt{S}_3 \in [1,4]$, $\texttt{S}_4 \in [2,4]$, DERANGEMENT($\langle 1\ \texttt{S}_1, 2\ \texttt{S}_2, 3\ \texttt{S}_3, 4\ \texttt{S}_4\rangle$).

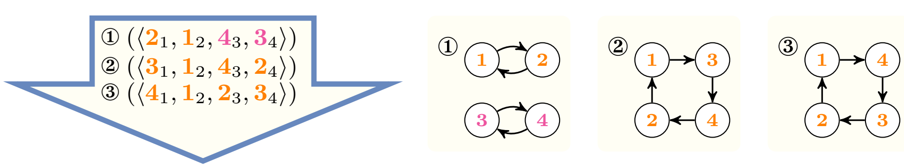


Figure 5.278: All solutions corresponding to the non ground example of the DERANGEMENT constraint of the **All solutions** slot; in the left-hand side the index attributes are displayed as indices of the succ attribute, while in the right-hand side they are directly displayed within each node.

Typical $|\mathtt{NODES}| > 2$

Symmetries

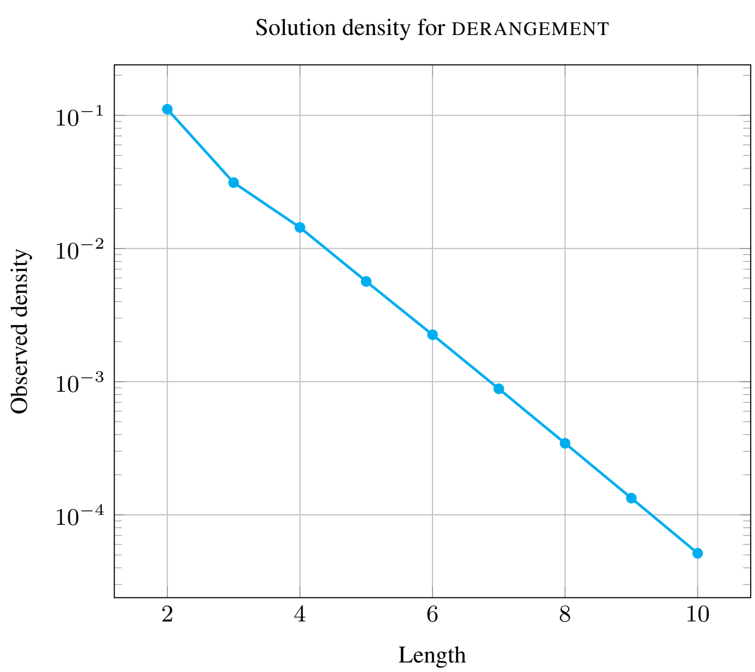
- Items of `NODES` are permutable.
- Attributes of `NODES` are permutable w.r.t. permutation $(\mathtt{index}, \mathtt{succ})$ (*permutation applied to all items*).

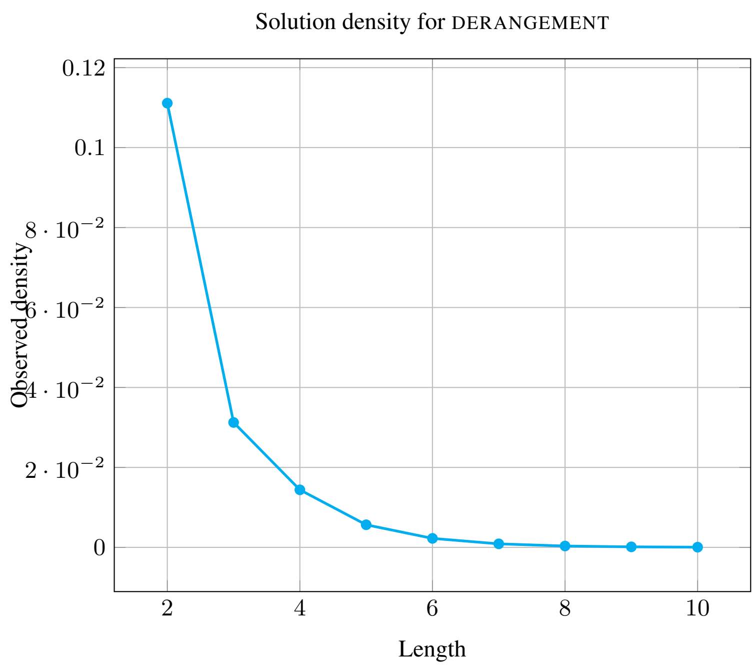
Remark A special case of the CYCLE [47] constraint.

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	1	2	9	44	265	1854	14833	133496	1334961

Number of solutions for DERANGEMENT: domains $0..n$





See also **common keyword:** ALLDIFFERENT, CYCLE *(permutation)*.

implied by: SYMMETRIC_ALLDIFFERENT.

implies: TWIN.

implies (items to collection): K_ALLDIFFERENT, LEX_ALLDIFFERENT.

Keywords **characteristic of a constraint:** sort based reformulation.

combinatorial object: permutation.

constraint type: graph constraint.

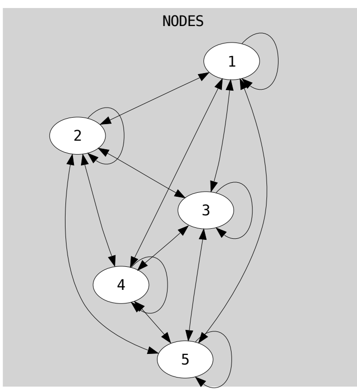
filtering: arc-consistency, DFS-bottleneck.

final graph structure: one_succ.

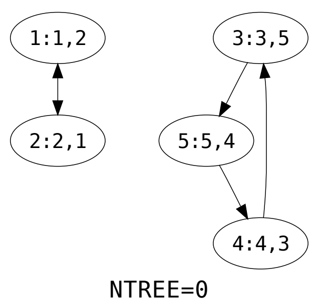
Cond. implications DERANGEMENT(NODES)
implies PERMUTATION(VARIABLES : NODES).

Arc input(s)	NODES
Arc generator	*CLIQUE* $\mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	• nodes1.succ = nodes2.index • nodes1.succ $\neq$ nodes1.index
Graph property(ies)	**NTREE** = 0
Graph class	ONE_SUCC

Graph model Parts (A) and (B) of Figure 5.279 respectively show the initial and final graph associated with the **Example** slot. The DERANGEMENT constraint holds since the final graph does not contain any vertex that does not belong to a circuit (i.e., **NTREE** = 0).



(A)



(B)

Figure 5.279: Initial and final graph of the DERANGEMENT constraint

In order to express the binary constraint that links two vertices of the NODES collection one has to make explicit the index value of the vertices. This is why the DERANGEMENT constraint considers objects that have two attributes:

- One fixed attribute index that is the identifier of the vertex,
- One variable attribute succ that is the successor of the vertex.

Forbidding cycles of length one is achieved by the second condition of the arc constraint.

Signature Since 0 is the smallest possible value of **NTREE** we can rewrite the graph property **NTREE** = 0 to **NTREE** $\leq$ 0. This leads to simplify $\overline{\textbf{NTREE}}$ to $\underline{\textbf{NTREE}}$.

5.117 DIFFER_FROM_AT_LEAST_K_POS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Inspired by [188].

Constraint DIFFER_FROM_AT_LEAST_K_POS(K, VECTOR1, VECTOR2)

Type VECTOR : collection(var−dvar)

Arguments

K	:	int
VECTOR1	:	VECTOR
VECTOR2	:	VECTOR

Restrictions

$|\texttt{VECTOR}| \geq 1$
required(VECTOR, var)
$\texttt{K} \geq 0$
$\texttt{K} \leq |\texttt{VECTOR1}|$
$|\texttt{VECTOR1}| = |\texttt{VECTOR2}|$

Purpose Enforce two vectors VECTOR1 and VECTOR2 to differ from at least K positions.

Example $(2, \langle 2,5,2,0\rangle, \langle 3,6,2,1\rangle)$

The DIFFER_FROM_AT_LEAST_K_POS constraint holds since the first and second vectors differ from 3 positions, which is greater than or equal to $\texttt{K} = 2$.

Typical

$\texttt{K} > 0$
$\texttt{K} < |\texttt{VECTOR1}|$
$|\texttt{VECTOR1}| > 1$

Symmetries

- Arguments are permutable w.r.t. permutation (K) (VECTOR1, VECTOR2).
- K can be decreased to any value ≥ 0.
- Items of VECTOR1 and VECTOR2 are permutable (*same permutation used*).

Arg. properties Extensible wrt. VARIABLES1 and VARIABLES2 (*add items at same position*).

Remark Used in the **Arc constraint(s)** slot of the ALL_DIFFER_FROM_AT_LEAST_K_POS constraint.

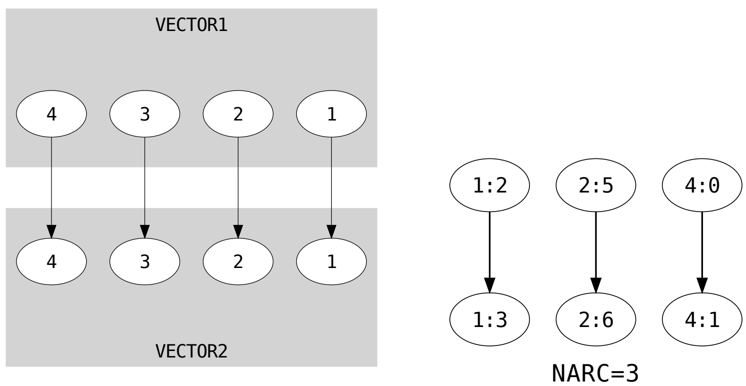
Used in ALL_DIFFER_FROM_AT_LEAST_K_POS.

See also **implied by:** DIFFER_FROM_EXACTLY_K_POS ($\geq$ K *replaced by* $=$ K).
system of constraints: ALL_DIFFER_FROM_AT_LEAST_K_POS.

Keywords **characteristic of a constraint:** vector, automaton, automaton with counters.
constraint network structure: alpha-acyclic constraint network(2).
constraint type: value constraint.

Arc input(s)	VECTOR1 VECTOR2
Arc generator	$PRODUCT(=) \mapsto$ collection(vector1, vector2)
Arc arity	2
Arc constraint(s)	vector1.var $\neq$ vector2.var
Graph property(ies)	**NARC** $\geq$ K

Graph model Parts (A) and (B) of Figure 5.280 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.280: Initial and final graph of the DIFFER_FROM_AT_LEAST_K_POS constraint

Automaton Figure 5.281 depicts the automaton associated with the DIFFER_FROM_AT_LEAST_K_POS constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ be the i^{th} variables of the VECTOR1 and VECTOR2 collections. To each pair of variables $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable S_i. The following signature constraint links $\texttt{VAR1}_i$, $\texttt{VAR2}_i$ and S_i: $\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow S_i$.

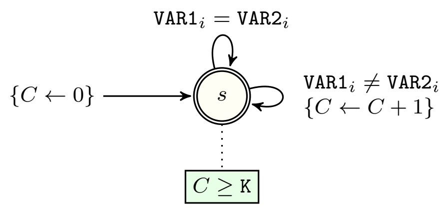


Figure 5.281: Automaton of the DIFFER_FROM_AT_LEAST_K_POS constraint

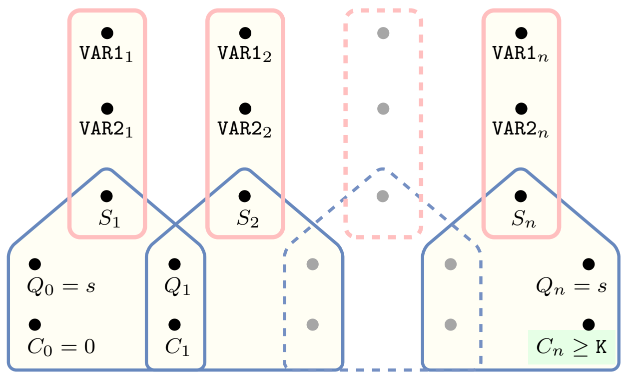


Figure 5.282: Hypergraph of the reformulation corresponding to the automaton of the DIFFER_FROM_AT_LEAST_K_POS constraint

5.118 DIFFER_FROM_AT_MOST_K_POS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Inspired by DIFFER_FROM_AT_LEAST_K_POS.

Constraint DIFFER_FROM_AT_MOST_K_POS(K, VECTOR1, VECTOR2)

Type

VECTOR : collection(var−dvar)

Arguments

K	:	int
VECTOR1	:	VECTOR
VECTOR2	:	VECTOR

Restrictions

$|\texttt{VECTOR}| \geq 1$
required(VECTOR, var)
$\texttt{K} \geq 0$
$\texttt{K} \leq |\texttt{VECTOR1}|$
$|\texttt{VECTOR1}| = |\texttt{VECTOR2}|$

Purpose Enforce two vectors VECTOR1 and VECTOR2 to differ from at most K positions.

Example $(3, \langle 2, 5, 2, 0\rangle, \langle 3, 6, 2, 0\rangle)$

The DIFFER_FROM_AT_MOST_K_POS constraint holds since the first and second vectors differ from 2 positions, which is less than or equal to $\texttt{K} = 3$.

Typical

$\texttt{K} > 0$
$\texttt{K} < |\texttt{VECTOR1}|$
$|\texttt{VECTOR1}| > 1$

Symmetries

- Arguments are permutable w.r.t. permutation (K) (VECTOR1, VECTOR2).
- K can be increased to any value $\leq |\texttt{VECTOR1}|$.
- Items of VECTOR1 and VECTOR2 are permutable (*same permutation used*).

Arg. properties Contractible wrt. VARIABLES1 and VARIABLES2 (*remove items from same position*).

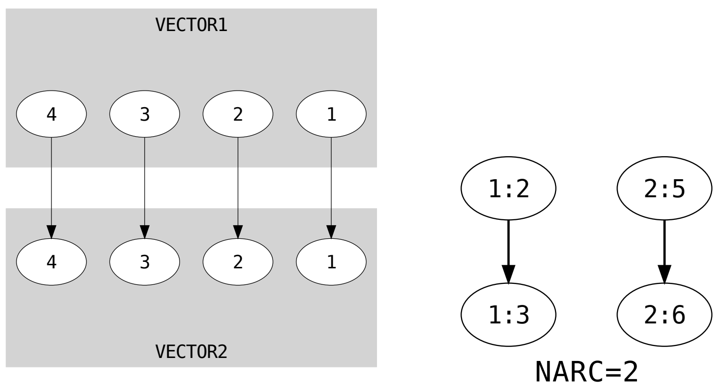
Used in ALL_DIFFER_FROM_AT_MOST_K_POS.

See also **implied by:** DIFFER_FROM_EXACTLY_K_POS ($\leq$ K *replaced by* = K).
system of constraints: ALL_DIFFER_FROM_AT_MOST_K_POS.

Keywords **characteristic of a constraint:** vector.
constraint type: value constraint.

Arc input(s)	VECTOR1 VECTOR2
Arc generator	$PRODUCT(=) \mapsto$ collection(vector1, vector2)
Arc arity	2
Arc constraint(s)	vector1.var $\neq$ vector2.var
Graph property(ies)	**NARC** $\leq$ K

Graph model Parts (A) and (B) of Figure 5.283 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.283: Initial and final graph of the DIFFER_FROM_AT_MOST_K_POS constraint

5.119 DIFFER_FROM_EXACTLY_K_POS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin
Inspired by DIFFER_FROM_AT_LEAST_K_POS.

Constraint
DIFFER_FROM_EXACTLY_K_POS(K, VECTOR1, VECTOR2)

Type
VECTOR : collection(var−dvar)

Arguments

K	:	int
VECTOR1	:	VECTOR
VECTOR2	:	VECTOR

Restrictions
$|\mathtt{VECTOR}| \geq 1$
required(VECTOR, var)
$\mathtt{K} \geq 0$
$\mathtt{K} \leq |\mathtt{VECTOR1}|$
$|\mathtt{VECTOR1}| = |\mathtt{VECTOR2}|$

Purpose
Enforce two vectors VECTOR1 and VECTOR2 to differ from exactly K positions.

Example
$(2, \langle 3, 0, 2, 0\rangle, \langle 3, 6, 2, 1\rangle)$

The DIFFER_FROM_EXACTLY_K_POS constraint holds since the first and second vectors differ from 2 positions, which is equal to $\mathtt{K} = 2$.

Typical
$\mathtt{K} > 0$
$\mathtt{K} \leq |\mathtt{VECTOR1}|$
$|\mathtt{VECTOR1}| > 1$

Symmetries
- Arguments are permutable w.r.t. permutation (K) (VECTOR1, VECTOR2).
- Items of VECTOR1 and VECTOR2 are permutable (*same permutation used*).

Arg. properties
Functional dependency: K determined by VECTOR1.

Used in
ALL_DIFFER_FROM_EXACTLY_K_POS.

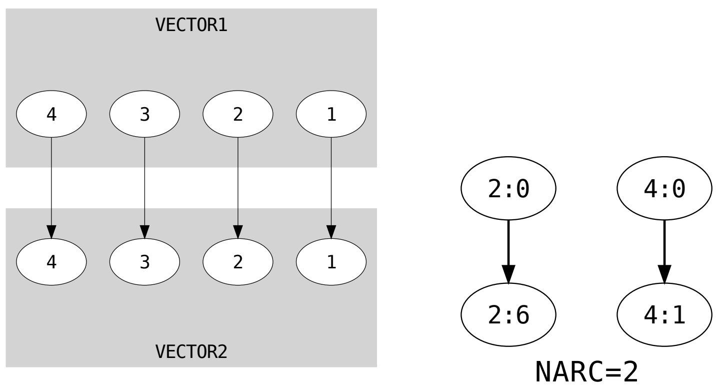
See also
implies: DIFFER_FROM_AT_LEAST_K_POS (= K *replaced by* $\geq$ K), DIFFER_FROM_AT_MOST_K_POS (= K *replaced by* $\leq$ K).

system of constraints: ALL_DIFFER_FROM_EXACTLY_K_POS.

Keywords
characteristic of a constraint: vector.
constraint arguments: pure functional dependency.
constraint type: value constraint.
modelling: functional dependency.

Arc input(s)	VECTOR1 VECTOR2
Arc generator	$PRODUCT(=) \mapsto$ collection(vector1, vector2)
Arc arity	2
Arc constraint(s)	vector1.var $\neq$ vector2.var
Graph property(ies)	**NARC** = K

Graph model Parts (A) and (B) of Figure 5.284 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.284: Initial and final graph of the DIFFER_FROM_EXACTLY_K_POS constraint

5.120 DIFFN

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [47]

Constraint DIFFN(ORTHOTOPES)

Synonyms DISJOINT, DISJOINT1, DISJOINT2, DIFF2.

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Argument ORTHOTOPES : collection(orth − ORTHOTOPE)

Restrictions
|ORTHOTOPE| >0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori $\leq$ ORTHOTOPE.end
required(ORTHOTOPES, orth)
same_size(ORTHOTOPES, orth)

Purpose Generalised multi-dimensional non-overlapping constraint: Holds if, for each pair of orthotopes (O_1, O_2), O_1 and O_2 do not overlap. Two orthotopes do not overlap if one of the orthotopes has zero size or if there exists at least one dimension where their projections do not overlap.

Example

$$\left(\left\langle\begin{array}{l}\texttt{orth}-\langle\texttt{ori}-2\ \texttt{siz}-2\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3\rangle,\\ \texttt{orth}-\langle\texttt{ori}-4\ \texttt{siz}-4\ \texttt{end}-8, \texttt{ori}-2\ \texttt{siz}-2\ \texttt{end}-4\rangle,\\ \texttt{orth}-\left\langle\begin{array}{lll}\texttt{ori}-6 & \texttt{siz}-5 & \texttt{end}-11,\\ \texttt{ori}-5 & \texttt{siz}-2 & \texttt{end}-7\end{array}\right\rangle\end{array}\right\rangle\right)$$

Figure 5.285 represents the position of the three rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The DIFFN constraint holds since the three rectangles do not overlap as explained in Figure 5.286.

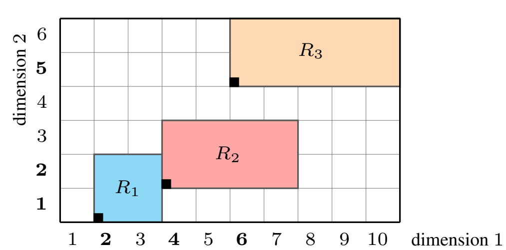


Figure 5.285: Illustration of the **Example** slot: the three rectangles

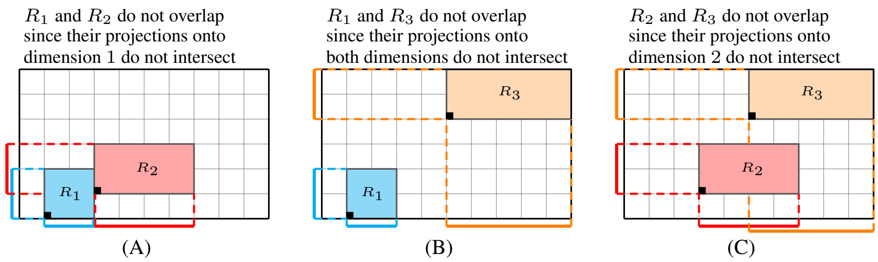


Figure 5.286: Illustration of the **Example** slot: the reasons (A), (B), (C) why the pairs of rectangles (R_1, R_2), (R_1, R_3), (R_2, R_3) do not overlap

All solutions

Figure 5.287 gives all solutions to the following non ground instance of the DIFFN constraint:
$\mathtt{X}_1 \in [1,3]$, $\mathtt{EX}_1 \in [1,9]$, $\mathtt{Y}_1 \in [1,3]$, $\mathtt{EY}_1 \in [1,9]$,
$\mathtt{X}_2 \in [1,3]$, $\mathtt{EX}_2 \in [1,9]$, $\mathtt{Y}_2 \in [2,3]$, $\mathtt{EY}_2 \in [1,9]$,
$\mathtt{X}_3 \in [1,2]$, $\mathtt{EX}_3 \in [1,9]$, $\mathtt{Y}_3 \in [1,4]$, $\mathtt{EY}_3 \in [1,9]$,
$\mathtt{X}_4 \in [1,3]$, $\mathtt{EX}_4 \in [1,9]$, $\mathtt{Y}_4 \in [1,3]$, $\mathtt{EY}_4 \in [1,9]$,
DIFFN$(\langle\langle \mathtt{X}_1\ 2\ \mathtt{EX}_1,\ \mathtt{Y}_1\ 3\ \mathtt{EY}_1\rangle,\ \langle \mathtt{X}_2\ 3\ \mathtt{EX}_2,\ \mathtt{Y}_2\ 2\ \mathtt{EY}_2\rangle,$
$\langle \mathtt{X}_3\ 1\ \mathtt{EX}_3,\ \mathtt{Y}_3\ 4\ \mathtt{EY}_3\rangle,\ \langle \mathtt{X}_4\ 4\ \mathtt{EX}_4,\ \mathtt{Y}_4\ 1\ \mathtt{EY}_4\rangle\rangle)$.

Typical

$|\mathtt{ORTHOTOPE}| > 1$
$\mathtt{ORTHOTOPE.siz} > 0$
$|\mathtt{ORTHOTOPES}| > 1$

Symmetries

- Items of ORTHOTOPES are permutable.
- Items of ORTHOTOPES.orth are permutable (*same permutation used*).
- ORTHOTOPES.orth.siz can be decreased to any value ≥ 0.
- One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPES.orth.

Arg. properties

Contractible wrt. ORTHOTOPES.

Usage

The DIFFN constraint occurs in placement and scheduling problems. It was used, for example, for scheduling problems where one has to both assign each non-preemptive task to a resource and fix its origin so that two tasks, which are assigned to the same resource, do not overlap. When the resource is a set of persons to which non-preemptive tasks have to be assigned this corresponds to so called *timetabling problems*. A second practical application from the area of the design of memory-dominated embedded systems [413] can be found in [414]. Together with arithmetic and CUMULATIVE constraints, the DIFFN constraint was used in [412] for packing more complex shapes such as angles. Figure 5.288 illustrates the angle packing problem on an instance involving 10 angles taken from [412].

One other packing problem attributed to S. Golomb is to find the smallest square that can contain the set of consecutive squares from 1×1 up to $n \times n$ so that these squares do not overlap each other (see the smallest rectangle area problem).

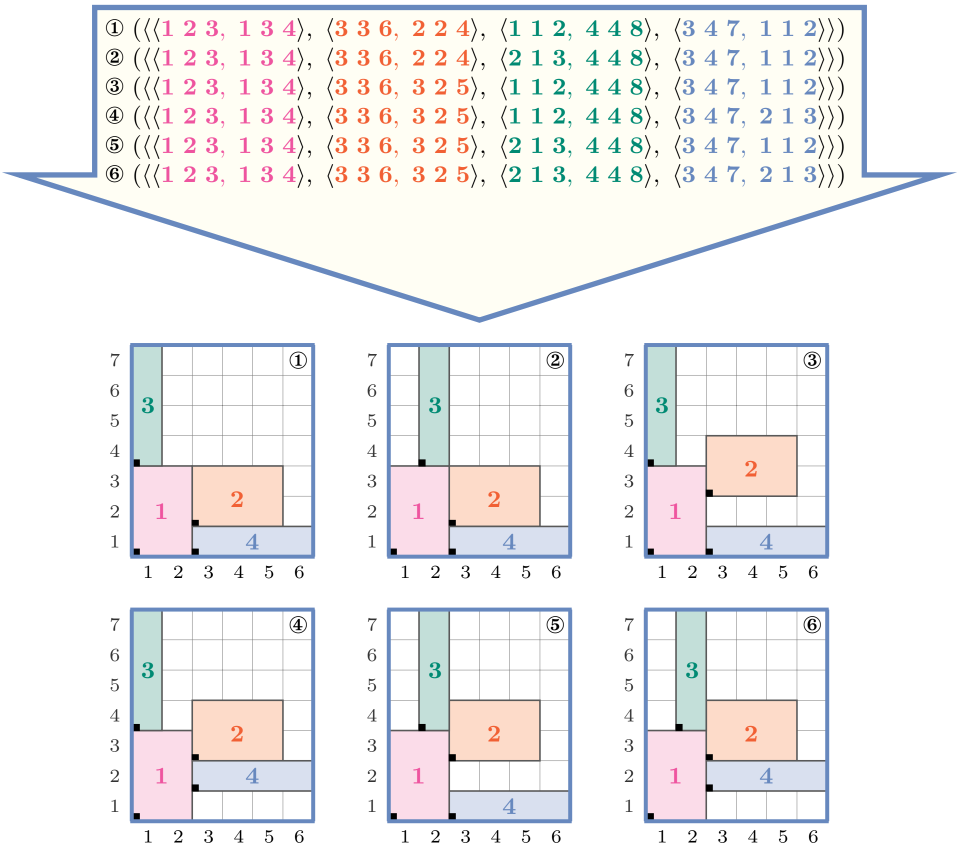


Figure 5.287: All solutions corresponding to the non ground example of the DIFFN constraint of the **All solutions** slot

Remark

Unlike the definition of the **Purpose** slot the original paper [47] introducing the DIFFN constraint imposes all orthotopes sizes to be different from 0. But it is convenient to allow variable sizes which can be assigned value 0 to model the fact that an orthotope can be skipped.

When we have segments (respectively rectangles) the DIFFN constraint is referenced under the name DISJOINT1 (respectively DISJOINT2) in **SICStus Prolog** [108]. When we have rectangles the DIFFN constraint is also called DIFF2 in **JaCoP**. In **MiniZinc** (`http://www.minizinc.org/`) the DIFFN constraint considers only rectangles.

It was shown in [417, page 137] that, finding out whether a non-overlapping constraint between a set of rectangles has a solution or not is NP-hard. This was achieved by reduction from sequencing with release times and deadlines.

In the two-dimensional case, when rectangles heights are all equal to one and when rectangles starts in the first dimension are all fixed, the DIFFN constraint can be rewritten as a K_ALLDIFFERENT constraint corresponding to a system of ALLDIFFERENT constraints derived from the maximum cliques of the corresponding interval graph.

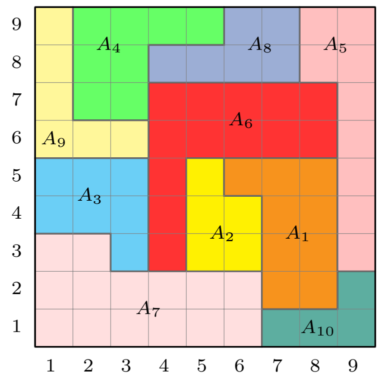


Figure 5.288: A solution for the angle packing problem of items $A_1 = [2,4,3,1]$, $A_2 = [2,2,1,3]$, $A_3 = [1,3,3,2]$, $A_4 = [2,1,4,3]$, $A_5 = [1,7,2,2]$, $A_6 = [1,2,5,5]$, $A_7 = [6,2,2,3]$, $A_8 = [4,2,2,1]$, $A_9 = [3,1,1,4]$, $A_{10} = [3,2,1,1]$.

Algorithm

Checking whether a DIFFN constraint for which all variables are fixed is satisfied or not is related to the Klee's measure problem: given a collection of axis-aligned multi-dimensional boxes, how quickly can one compute the volume of their union. Then the DIFFN constraint holds if the volume of the union is equal to the sum of the volumes of the different boxes.

A first possible method for filtering non zero size orthotopes is to use constructive disjunction. The idea is to try out each alternative of a disjunction (e.g., given two orthotopes o_1 and o_2 that should not overlap, we successively assume for each dimension that o_1 finishes before o_2, and that o_2 finishes before o_1) and to remove values that were pruned in all alternatives. For the two-dimensional case of DIFFN a second possible solution used in [372] is to represent explicitly the two-dimensional domain of the origin of each rectangle by a quadtree [378] and to accumulate all forbidden regions within this data structure. As for conventional domain variables, a failure occurs when a two-dimensional domain get empty. A third possible filtering algorithm based on sweep is described in [34].

The thesis of J. Nelissen [303] considers the case where all rectangles have the same size and can be rotated from 90 degrees (i.e., the pallet loading problem.). For the n-dimensional case of DIFFN a filtering algorithm handling the fact that two objects do not overlap is given in [50].

Extensions of the non-overlapping constraint to polygons and to more complex shapes are respectively described in [50] and in [367]. Specialised propagation algorithms for the squared squares problem [94] (based on the fact that no waste is permitted) are given in [192] and in [191].

The CUMULATIVE constraint can be used as a *necessary condition* for the DIFFN constraint. Figure 5.290 illustrates this point for the two-dimensional case. A first (respectively second) CUMULATIVE constraint is obtained by forgetting the y-coordinate (respectively the x-coordinate) of the origin of each rectangle occurring in a DIFFN constraint. Parts (B) and (C) respectively depict the cumulated profiles associated with the projection of the rectangles depicted by part (A) on the x and y axes.

The CUMULATIVE constraint is a necessary *but not sufficient condition* for the two-dimensional case of the DIFFN constraint. Figure 5.291 illustrates this point on an

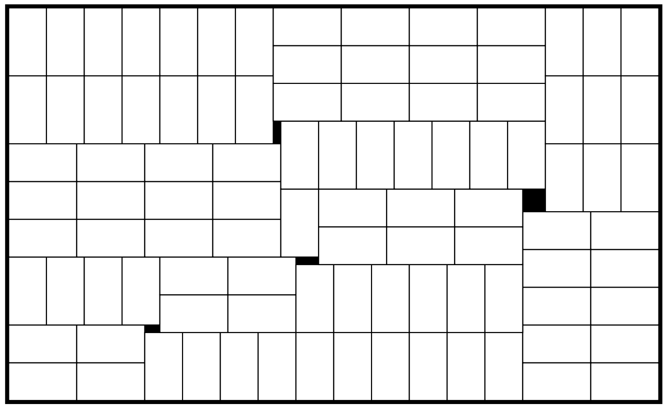


Figure 5.289: A hard instance from [303, page 165]: A solution for packing 99 rectangles of size 5×9 into a rectangle of size 86×52

example taken from [84] where one has to place the 8 rectangles R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8 of respective size 5×2, 8×2, 6×1, 5×1, 2×1, 3×1, 2×2 and 1×2 in a big rectangle of size 12×4. As shown by Figure 5.291 there is a cumulative solution where R_8 is split in two parts but M. Hujter proves in [232] that there is no solution where no rectangle is split.

In the context of n parallelepipeds that have to be packed [198, 272] within a box of sizes $X \times Y \times Z$ one can proceed as follows for stating three CUMULATIVE constraints. The i^{th} (with $i \in [1, n]$) parallelepiped is described by the following attributes:

- ox_i, oy_i, oz_i (with $i \in [1, n]$) the coordinates of its origin on the x, y and z-axes.
- sx_i, sy_i, sz_i (with $i \in [1, n]$) its sizes on the x, y and z-axes.
- px_i, py_i, pz_i (with $i \in [1, n]$) the surfaces of its projections onto the planes yz, xz, and xy respectively equal to $sy_i sz_i$, $sx_i sz_i$, and $sx_i sy_i$.
- v_i its volume (equal to $sx_i sy_i sz_i$).

For the placement of n parallelepipeds we get the following necessary conditions that respectively correspond to three CUMULATIVE constraints on the planes yz, xz, and xy:

$$\begin{cases} \forall i \in [1, X] : \sum_{j \mid ox_j \leq i \leq ox_j + sx_j - 1} px_j \leq YZ \\ \forall i \in [1, Y] : \sum_{j \mid oy_j \leq i \leq oy_j + sy_j - 1} py_j \leq XZ \\ \forall i \in [1, Z] : \sum_{j \mid oz_j \leq i \leq oz_j + sz_j - 1} pz_j \leq XY \end{cases}$$

Reformulation Based on the fact that two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap one can reformulate the DIFFN(ORTHOTOPES) constraint as a disjunction of inequalities between the origin and the end attributes. In addition one has to link the origin, the size and the end attributes of each orthotope in each dimension.

If we consider the example described in the **Example** slot we get the following reformulation:

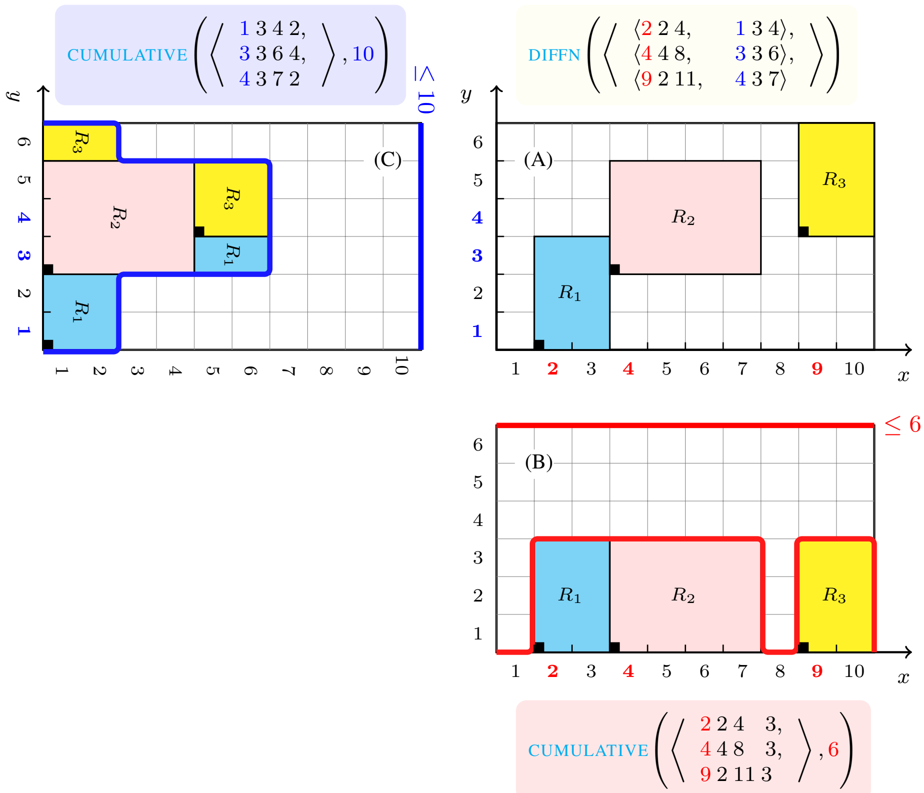


Figure 5.290: Looking from the perspective of the CUMULATIVE constraint in a two-dimensional rectangles placement problem: projecting the three rectangles of (A) on the x axis (B) and on the y axis (C)

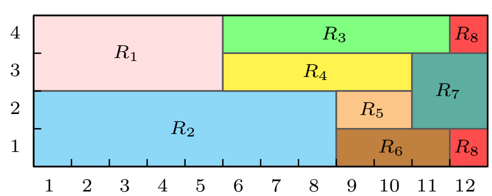


Figure 5.291: Illustrating the necessary but not sufficient placement condition: the rectangle R_8 is split in two parts

- $4 = 2+2$ (*link between the origin, size and end in dimension* 1 *of the first orthotope*),
- $4 = 1+3$ (*link between the origin, size and end in dimension* 2 *of the first orthotope*),
- $8 = 4 + 4$ (*link between the origin, size and end in dimension* 1 *of the second*

orthotope),

- $6 = 3 + 3$ (*link between the origin, size and end in dimension* 2 *of the second orthotope*),
- $11 = 9 + 2$ (*link between the origin, size and end in dimension* 1 *of the third orthotope*),
- $7 = 4+3$ (*link between the origin, size and end in dimension* 2 *of the third orthotope*),
- $4 \leq 4 \vee 8 \leq 2 \vee 4 \leq 3 \vee 6 \leq 1$ (*non-overlapping between the first and second orthotopes*),
- $4 \leq 9 \vee 11 \leq 2 \vee 4 \leq 4 \vee 7 \leq 1$ (*non-overlapping between the first and third orthotopes*),
- $8 \leq 9 \vee 11 \leq 4 \vee 6 \leq 4 \vee 7 \leq 3$ (*non-overlapping between the second and third orthotopes*).

Systems GEOST in **Choco**, NOOVERLAP in **Gecode**, DIFF2 in **JaCoP**, DIFF in **JaCoP**, DISJOINT in **JaCoP**, DISJOINTCONDITIONAL in **JaCoP**, DIFFN in **MiniZinc**.

Used in DIFFN_COLUMN, DIFFN_INCLUDE, PLACE_IN_PYRAMID.

See also **common keyword:** CALENDAR (*multi-site employee scheduling with calendar constraints*,
scheduling with machine choice, calendars and preemption), DIFFN_COLUMN, DIFFN_INCLUDE (*geometrical constraint,orthotope*), GEOST, GEOST_TIME, NON_OVERLAP_SBOXES (*geometrical constraint,non-overlapping*), VISIBLE (*geometrical constraint*).

implied by: ORTHS_ARE_CONNECTED.

implies: CUMULATIVE (*implies one* CUMULATIVE *constraint for each dimension*).

related: CUMULATIVE_TWO_D (CUMULATIVE_TWO_D *is a necessary condition for* DIFFN*: forget one dimension when the number of dimensions is equal to* 3), LEX_CHAIN_LESS, LEX_CHAIN_LESSEQ (*lexicographic ordering on the origins of* `tasks`, `rectangles`, *...*), TWO_ORTH_COLUMN, TWO_ORTH_INCLUDE.

specialisation: ALL_MIN_DIST (`orthotope` *replaced by* `line segment`, *of same length*), ALLDIFFERENT (`orthotope` *replaced by* `variable`), CUMULATIVES (`orthotope` *replaced by* `task` *with* `machine assignment` *and* `origin` *attributes*), DISJUNCTIVE (*orthotope replaced by* `task` *of* `heigth` 1), K_ALLDIFFERENT (*when rectangles heights are all equal to* 1 *and rectangles starts in the first dimension are all fixed*), LEX_ALLDIFFERENT (`orthotope` *replaced by* `vector`).

used in graph description: ORTH_LINK_ORI_SIZ_END, TWO_ORTH_DO_NOT_OVERLAP.

Keywords **application area:** floor planning problem.

characteristic of a constraint: core.

combinatorial object: pentomino.

complexity: sequencing with release times and deadlines.

constraint arguments: business rules.

constraint type: decomposition, timetabling constraint, relaxation.

filtering: Klee measure problem, sweep, quadtree, compulsory part, constructive disjunction, SAT.

geometry: geometrical constraint, orthotope, polygon, non-overlapping.

heuristics: heuristics for two-dimensional rectangle placement problems.

modelling: disjunction, assignment dimension, assignment to the same set of values, assigning and scheduling tasks that run in parallel, relaxation dimension, sequence dependent set-up, multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption.

modelling exercises: assignment to the same set of values, assigning and scheduling tasks that run in parallel, relaxation dimension, sequence dependent set-up, multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption.

problems: strip packing, two-dimensional orthogonal packing, pallet loading.

puzzles: squared squares, packing almost squares, Partridge, pentomino, Shikaku, smallest square for packing consecutive dominoes, smallest square for packing rectangles with distinct sizes, smallest rectangle area, Conway packing problem.

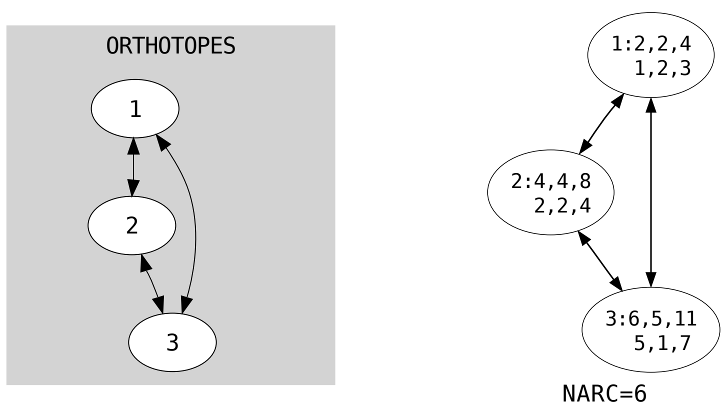
Arc input(s)	ORTHOTOPES
Arc generator	$SELF \mapsto$ collection(orthotopes)
Arc arity	1
Arc constraint(s)	ORTH_LINK_ORI_SIZ_END(orthotopes.orth)
Graph property(ies)	**NARC**$=$ |ORTHOTOPES|

Arc input(s)	ORTHOTOPES
Arc generator	$CLIQUE(\neq) \mapsto$ collection(orthotopes1, orthotopes2)
Arc arity	2
Arc constraint(s)	TWO_ORTH_DO_NOT_OVERLAP(orthotopes1.orth, orthotopes2.orth)
Graph property(ies)	**NARC**$=$ |ORTHOTOPES| $*$ |ORTHOTOPES| $-$ |ORTHOTOPES|

Graph model The DIFFN constraint is expressed by using two graph constraints:

- The first graph constraint forces for each dimension and for each orthotope the link between the corresponding `ori`, `siz` and `end` attributes.
- The second graph constraint imposes each pair of distinct orthotopes to not overlap.

Parts (A) and (B) of Figure 5.292 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.292: Initial and final graph of the DIFFN constraint

Signature Since |ORTHOTOPES| is the maximum number of vertices of the final graph of the first graph constraint we can rewrite **NARC** $=$ |ORTHOTOPES| to **NARC** $\geq$ |ORTHOTOPES|. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Since we use the $CLIQUE(\neq)$ arc generator on the ORTHOTOPES collection, $|\text{ORTHOTOPES}| \cdot |\text{ORTHOTOPES}| - |\text{ORTHOTOPES}|$ is the maximum number of vertices of the final graph of the second graph constraint. Therefore we can rewrite **NARC** $= |\text{ORTHOTOPES}| \cdot |\text{ORTHOTOPES}| - |\text{ORTHOTOPES}|$ to **NARC** $\geq |\text{ORTHOTOPES}| \cdot |\text{ORTHOTOPES}| - |\text{ORTHOTOPES}|$. Again, this leads to simplify $\underline{\overline{\textbf{NARC}}}$ to $\overline{\textbf{NARC}}$.

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint DIFFN$(\langle\langle 1\ 1\ 1, 2\ 2\ 4\rangle, \langle 5\ 1\ 6, 4\ 1\ 5\rangle\rangle)$ hold?

B. Does the constraint DIFFN$(\langle\langle 2\ 2\ 4, 3\ 2\ 5\rangle, \langle 4\ 3\ 7, 4\ 2\ 6\rangle, \langle 8\ 2\ 10, 2\ 3\ 5\rangle\rangle)$ hold?

C. Does the constraint DIFFN$(\langle\langle 2\ 2\ 4, 2\ 2\ 4\rangle, \langle 4\ 5\ 9, 4\ 2\ 6\rangle, \langle 8\ 2\ 10, 2\ 3\ 5\rangle\rangle)$ hold?

D. Does the constraint DIFFN$(\langle\langle 3\ 2\ 5\rangle, \langle 4\ 0\ 4\rangle, \langle 6\ 3\ 9\rangle\rangle)$ hold?

[a] Hint: go back to the definition of DIFFN.

EXERCISE 2 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\left\{\begin{array}{llll} OX_1 \in [1,5], & EX_1 \in [1,5], & OY_1 \in [1,5], & EY_1 \in [1,5], \\ OX_2 \in [1,5], & EX_2 \in [1,5], & OY_2 \in [1,5], & EY_2 \in [1,5], \\ OX_3 \in [1,5], & EX_3 \in [1,5], & OY_3 \in [1,5], & EY_3 \in [1,5], \\ \multicolumn{4}{l}{\text{DIFFN}\left(\left\langle\begin{array}{ll} \langle OX_1\ 1\ EX_1, & OY_1\ 3\ EY_1\rangle, \\ \langle OX_2\ 4\ EX_2, & OY_2\ 1\ EY_2\rangle, \\ \langle OX_3\ 3\ EX_3, & OY_3\ 3\ EY_3\rangle \end{array}\right\rangle\right).} \end{array}\right.$$

[a] Hint: consider rectangles by decreasing surface and focus on the coordinates of their origins (OX_3, OY_3), (OX_2, OY_2) and (OX_1, OY_1); enumerate solutions in lexicographic order of (OX_3, OY_3).

EXERCISE 3 (finding the unique solution)[a]

Find the unique solution to the constraint:

$$\left\{\begin{array}{l}
\begin{array}{llll}
OX_1 \in [1,8], & EX_1 \in [1,8], & OY_1 \in [1,8], & EY_1 \in [1,8], \\
OX_2 \in [1,8], & EX_2 \in [1,8], & OY_2 \in [1,8], & EY_2 \in [1,8], \\
OX_3 \in [1,8], & EX_3 \in [1,8], & OY_3 \in [1,8], & EY_3 \in [1,8], \\
OX_4 \in [1,8], & EX_4 \in [1,8], & OY_4 \in [1,8], & EY_4 \in [1,8], \\
OX_5 \in [1,8], & EX_5 \in [1,8], & OY_5 \in [1,8], & EY_5 \in [1,8],
\end{array} \\
\\
\textsc{diffn}\left(\left\langle\begin{array}{llllll}
\langle OX_1 & 2 & EX_1, & OY_1 & 5 & EY_1\rangle, \\
\langle OX_2 & 5 & EX_2, & OY_2 & 1 & EY_2\rangle, \\
\langle OX_3 & 2 & EX_3, & OY_3 & 4 & EY_3\rangle, \\
\langle OX_4 & 4 & EX_4, & OY_4 & 2 & EY_4\rangle, \\
\langle OX_5 & 3 & EX_5, & OY_5 & 3 & EY_5\rangle, \\
\langle 1 & 3 & 4, & 1 & 3 & 4\rangle
\end{array}\right\rangle\right).
\end{array}\right.$$

[a]Hint: reason on whose compulsory parts of the projections of the rectangles onto the x and y axes.

EXERCISE 4 (degrees of violation for non-overlapping)[a]

A. Give the variable-based degree of violation[b] of the constraint

$$\textsc{diffn}\left(\left\langle\begin{array}{l}
\langle 3\ 2\ 5, \quad 4\ 2\ 6\rangle, \\
\langle 4\ 5\ 9, \quad 5\ 2\ 7\rangle, \\
\langle 6\ 2\ 8, \quad 3\ 5\ 8\rangle, \\
\langle 7\ 1\ 8, \quad 2\ 4\ 6\rangle
\end{array}\right\rangle\right).$$

B. Give the decomposition-based degree of violation[c] of the same constraint.

C. In the decomposition-based degree of violation each violated binary constraint contributes $+1$ to the degree of violation, which does not consider how much two orthotopes overlap. For the same constraint give the *overlap decomposition-based degree of violation* where the degree of violation of a binary no-overlap constraint is equal to the overlap between the corresponding orthotopes.

D. Given two fixed orthotopes of the DIFFN constraint, propose a formula for computing how much these orthotopes overlap.

[a]Hint: focus on the rectangles that overlap; by changing one coordinate of a rectangle one can move it (A); count how many rectangles overlap (B); count how much each pair of rectangles overlaps (C).

[b]Given a constraint for which all variables are fixed, the *variable-based degree of violation* is the minimum number of variables to assign differently in order to satisfy the constraint.

[c]Given a constraint that can be decomposed in a conjunction of binary constraints, the *decomposition-based degree of violation* is the number of binary constraints that do not hold.

EXERCISE 5 (nasty pattern)[a]

Find a solution to the following constraint, or prove that no solution exists:

$$\left\{\begin{array}{l} \begin{array}{llll} OX_1 \in [0,2], & EX_1 \in [0,9], & OY_1 \in [0,2], & EY_1 \in [0,9], \\ OX_2 \in [1,3], & EX_2 \in [0,9], & OY_2 \in [1,3], & EY_2 \in [0,9], \\ OX_3 \in [3,4], & EX_3 \in [0,9], & OY_3 \in [3,4], & EY_3 \in [0,9], \\ OX_4 \in [3,5], & EX_4 \in [0,9], & OY_4 \in [3,5], & EY_4 \in [0,9], \\ OX_5 \in [4,6], & EX_5 \in [0,9], & OY_5 \in [4,6], & EY_5 \in [0,9], \end{array} \\ \\ \textsc{diffn}\left(\left\langle \begin{array}{ll} \langle OX_1\ 3\ EX_1, & OY_1\ 3\ EY_1\rangle, \\ \langle OX_2\ 3\ EX_2, & OY_2\ 3\ EY_2\rangle, \\ \langle OX_3\ 2\ EX_3, & OY_3\ 2\ EY_3\rangle, \\ \langle OX_4\ 3\ EX_4, & OY_4\ 3\ EY_4\rangle, \\ \langle OX_5\ 3\ EX_5, & OY_5\ 3\ EY_5\rangle \end{array} \right\rangle\right). \end{array}\right.$$

[a] Hint: focus on the compulsory part of the squares.

SOLUTION TO EXERCISE 1

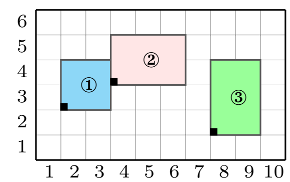
A. *No, since the first rectangle* $\langle 1\ 1\ 1, 2\ 2\ 4\rangle$ *is not well formed, i.e.,* $1 + 1 \neq 1$.

B. *Yes, since the three rectangles do not overlap:*

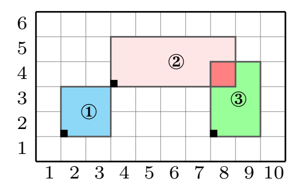
- *Rectangles ① and ② do not overlap since their projections onto the x axis do not overlap,*
- *Rectangles ② and ③ do not overlap since their projections onto the x axis do not overlap,*
- *Rectangles ① and ③ do not overlap since their projections onto the x axis do not overlap.*

C. *No, since rectangles ② and ③ overlap, i.e., their projections onto the x and y axes both overlap.*

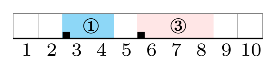
D. *Yes, since line segments ① and ③ do not overlap and since the size of line segment ② is zero.*



B.



C.



D.

SOLUTION TO EXERCISE 2

the four solutions *(once R_3 is fixed, R_2 and R_1 are also fixed)*

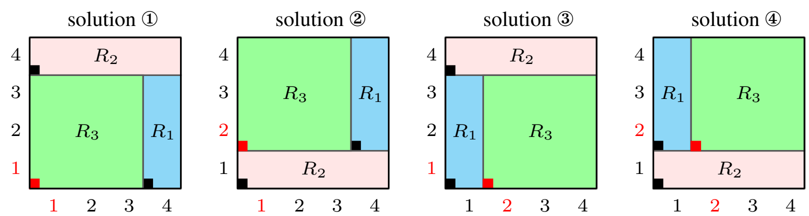
$\langle\langle OX_1 1 EX_1, OY_1 3 EY_1\rangle, \langle OX_2 4 EX_2, OY_2 1 EY_2\rangle, \langle OX_3 3 EX_3, OY_3 3 EY_3\rangle\rangle$

① $(\langle\langle 4\ 1\ 5, 1\ 3\ 4\rangle,\quad \langle 1\ 4\ 5, 4\ 1\ 5\rangle,\quad \langle 1\ 3\ 4, 1\ 3\ 4\rangle\rangle)$

② $(\langle\langle 4\ 1\ 5, 2\ 3\ 5\rangle,\quad \langle 1\ 4\ 5, 1\ 1\ 2\rangle,\quad \langle 1\ 3\ 4, 2\ 3\ 5\rangle\rangle)$

③ $(\langle\langle 1\ 1\ 2, 1\ 3\ 4\rangle,\quad \langle 1\ 4\ 5, 4\ 1\ 5\rangle,\quad \langle 2\ 3\ 5, 1\ 3\ 4\rangle\rangle)$

④ $(\langle\langle 1\ 1\ 2, 2\ 3\ 5\rangle,\quad \langle 1\ 4\ 5, 1\ 1\ 2\rangle,\quad \langle 2\ 3\ 5, 2\ 3\ 5\rangle\rangle)$

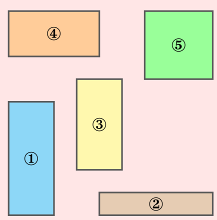


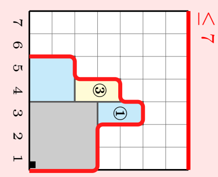
SOLUTION TO EXERCISE 3

We go through the following reasoning steps:

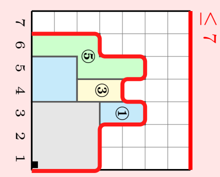
A. [*SELECTING THE x OR y AXIS FOR REASONING*]
Among the rectangles that have the largest side lengths, i.e., $R_1 : 2 \times 5$ and $R_2 : 5 \times 1$, we select the rectangle that has the largest surface, i.e., R_1. Since the largest size of R_1 is located in dimension y we first choose to reason on the compulsory parts of the projections of the rectangles onto the y axis.

B. [*REASONING ON THE PROJECTIONS ONTO THE y AXIS*]
We focus on the projections of the rectangles onto the y axis and gradually build the cumulated profile of their compulsory parts. Fixed (respectively not completely fixed) projections use a saturated (respectively unsaturated) colour. The gray square corresponds to an initially fixed square.

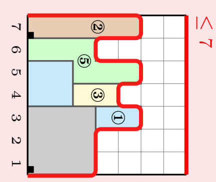




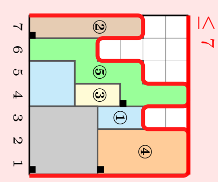
① overlaps $[3, 5]$,
③ overlaps $[4, 4]$,
⑤ cannot start in $[1, 3]$.



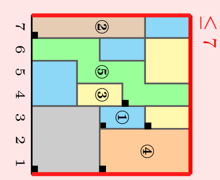
⑤ overlaps $[5, 6]$,
② cannot start in $[1, 6]$.



② overlaps $[7, 7]$:
$\Rightarrow$ $OY_2 = 7$,
⑤ cannot start in $[5, 5]$:
$\Rightarrow$ $OY_5 = 4$,
④ cannot start in $[2, 7]$:
$\Rightarrow$ $OY_4 = 1$.



① cannot start in $[1, 2]$:
$\Rightarrow$ $OY_1 = 3$,
③ cannot end in $[7, 7]$:
$\Rightarrow$ $OY_3 = 3$.



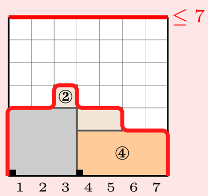
C. [*FILTERING A RECTANGLE WRT ONE OTHER RECTANGLE*]
A no-overlap constraint between two rectangles can be represented as a disjunction with four alternatives:

- *on the x axis the first rectangle ends before the start of the second rectangle,*
- *on the x axis the second rectangle ends before the start of the first rectangle,*
- *on the y axis the first rectangle ends before the start of the second rectangle,*
- *on the y axis the second rectangle ends before the start of the first rectangle.*

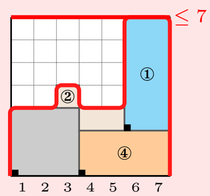
If we consider the fixed 3×3 gray square and the rectangle R_4 we have $4 \leq OX_4 \vee EX_4 \leq 1 \vee 4 \leq 1 \vee 3 \leq 1$. The part $4 \leq 1 \vee 3 \leq 1$ does not hold. Since the minimum value of EX_4 is equal to 5 the inequality $EX_4 \leq 1$ does also not hold. Consequently $4 \leq OX_4$ must hold and the minimum value of OX_4 is equal to 4. Moreover since the maximum value of OX_4 is 4 we have $OX_4 = 4$.

SOLUTION TO EXERCISE 3 (continued)

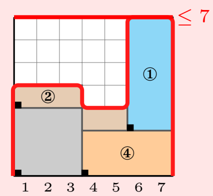
D. [REASONING ON THE PROJECTIONS ONTO THE x AXIS]
We focus on the projections of the rectangles onto the x axis and gradually build the cumulated profile of their compulsory parts. Fixed (respectively not completely fixed) projections use a saturated (respectively unsaturated) colour.



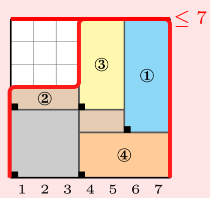
① cannot start in $[1, 5]$:
$\Rightarrow$ $OX_1 = 6$.



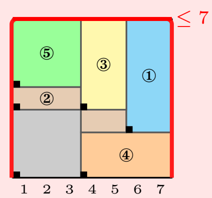
② cannot overlap $[6, 7]$:
$\Rightarrow$ $OX_2 = 1$.



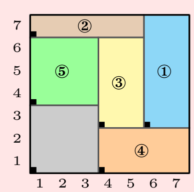
③ cannot overlap $[1, 3]$:
$\Rightarrow$ $OX_3 = 4$.



⑤ cannot overlap $[4, 7]$:
$\Rightarrow$ $OX_5 = 1$.

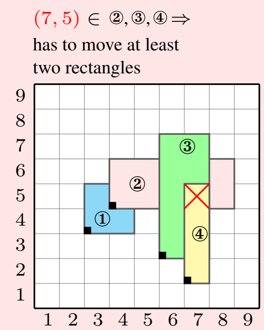


E. [PUTTING THINGS TOGETHER]
For any pair of distinct rectangles i, j we check $EX_i \leq OX_j \vee EX_j \leq OX_i \vee EY_i \leq OY_j \vee EY_j \leq OY_i$, i.e., there exists at least one dimension where the projections of the two rectangles do not overlap. We obtain the following unique solution.

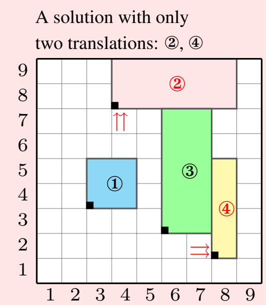


SOLUTION TO EXERCISE 4

A. *Since point* $(7, 5)$ *is included in the three rectangles ②, ③, ④, (see Figure* (A1)*), we need to modify the attributes of at least two rectangles. Figure* (A2) *shows a solution where only rectangles ② and ④ are translated. Therefore the variable-based degree of violation is equal to* 2*.*

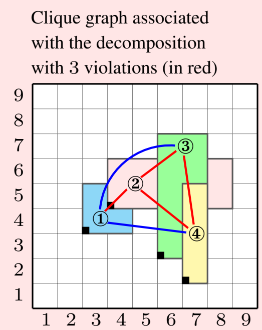


(A1)



(A2)

B. *Figure* (B) *shows the constraint network associated with the decomposition where each vertex corresponds to a rectangle and each edge to a binary no-overlap constraint between two rectangles. Edges where the corresponding binary constraint holds (respectively does not hold) are coloured in blue (in red). Consequently the decomposition-based degree of violation is equal to* 4 *(i.e., each pair (①,②), (②,③), (②,④), (③,④) corresponds to two overlapping rectangles).*

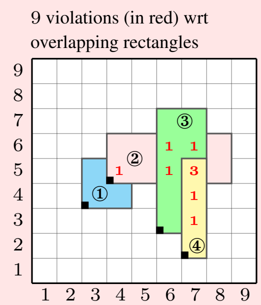


(B)

SOLUTION TO EXERCISE 4 (continued)

C. *As illustrated by Figure* (C), *the overlap decomposition-based degree of violation is equal to* 9 *since:*

- *the overlap between rectangles ① and ② is equal to* 1,
- *the overlap between rectangles ② and ③ is equal to* 4,
- *the overlap between rectangles ② and ④ is equal to* 1,
- *the overlap between rectangles ③ and ④ is equal to* 3,
- *the other pairs of rectangles do not overlap.*



(C)

D. *Given two orthotopes i and j, defined by their ends and their origins in each dimension, their overlap is defined by* $\prod_{d=1}^{|\texttt{ORTHOTOPE}|} \max(0, \min(\texttt{end}_{i,d}, \texttt{end}_{j,d}) - \max(\texttt{ori}_{i,d}, \texttt{ori}_{j,d}))$.

5.121 DIFFN_COLUMN

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

CHIP: option guillotine cut (column) of DIFFN.

Constraint

DIFFN_COLUMN(ORTHOTOPES, DIM)

Type

ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments

ORTHOTOPES : collection(orth − ORTHOTOPE)
DIM : int

Restrictions

$|\mathtt{ORTHOTOPE}| > 0$
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori $\leq$ ORTHOTOPE.end
required(ORTHOTOPES, orth)
same_size(ORTHOTOPES, orth)
DIM > 0
DIM $\leq |\mathtt{ORTHOTOPE}|$
DIFFN(ORTHOTOPES)

Purpose

Extension of the generalised multi-dimensional non-overlapping diffn constraint. Holds if, for each pair of orthotopes (O_1, O_2) the following conditions hold:

- O_1 and O_2 do not overlap. Two orthotopes do not overlap if one of the orthotopes has zero size or if there exists at least one dimension where their projections do not overlap.
- Let P_1 and P_2 respectively denote the projections of O_1 and O_2 onto dimension DIM. If P_1 and P_2 overlap then the size of their intersection is equal to the size of O_1 in dimension DIM, as well as to the size of O_2 in dimension DIM.

Example

$$\left(\left\langle\begin{array}{l}
\mathtt{orth}-\langle\mathtt{ori}-1\ \mathtt{siz}-3\ \mathtt{end}-4, \mathtt{ori}-3\ \mathtt{siz}-2\ \mathtt{end}-5\rangle,\\
\mathtt{orth}-\left\langle\begin{array}{lll}\mathtt{ori}-9 & \mathtt{siz}-1 & \mathtt{end}-10,\\ \mathtt{ori}-4 & \mathtt{siz}-3 & \mathtt{end}-7\end{array}\right\rangle,\\
\mathtt{orth}-\langle\mathtt{ori}-4\ \mathtt{siz}-2\ \mathtt{end}-6, \mathtt{ori}-3\ \mathtt{siz}-4\ \mathtt{end}-7\rangle,\\
\mathtt{orth}-\langle\mathtt{ori}-1\ \mathtt{siz}-3\ \mathtt{end}-4, \mathtt{ori}-6\ \mathtt{siz}-1\ \mathtt{end}-7\rangle,\\
\mathtt{orth}-\langle\mathtt{ori}-6\ \mathtt{siz}-2\ \mathtt{end}-8, \mathtt{ori}-1\ \mathtt{siz}-4\ \mathtt{end}-5\rangle,\\
\mathtt{orth}-\left\langle\begin{array}{lll}\mathtt{ori}-10 & \mathtt{siz}-1 & \mathtt{end}-11,\\ \mathtt{ori}-1 & \mathtt{siz}-1 & \mathtt{end}-2\end{array}\right\rangle,\\
\mathtt{orth}-\left\langle\begin{array}{lll}\mathtt{ori}-9 & \mathtt{siz}-1 & \mathtt{end}-10,\\ \mathtt{ori}-1 & \mathtt{siz}-1 & \mathtt{end}-2\end{array}\right\rangle,\\
\mathtt{orth}-\langle\mathtt{ori}-6\ \mathtt{siz}-2\ \mathtt{end}-8, \mathtt{ori}-6\ \mathtt{siz}-1\ \mathtt{end}-7\rangle
\end{array}\right\rangle, 1\right)$$

Figure 5.293 represents the respective position of the eight rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The DIFFN_COLUMN constraint holds since (1) the eight rectangles do not overlap and since (2) when their projection onto dimension $\texttt{DIM} = 1$ overlap the size of their intersection is equal to the size of the corresponding rectangles in dimension $\texttt{DIM} = 1$.

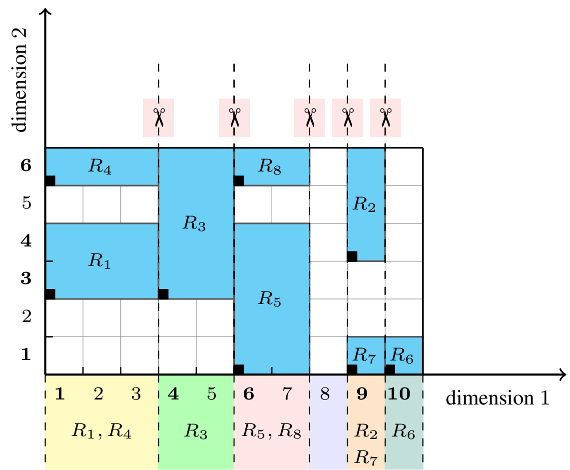


Figure 5.293: Illustration of the **Example** slot: eight non-overlapping rectangles such that, for each pair of rectangles R_i, R_j $(1 \leq i < j \leq 8)$, if the projections onto dimension 1 of rectangles R_i and R_j intersect then the size of their intersection is equal to the size of R_i in dimension 1 and to the size of R_j in dimension 1 (i.e. complete vertical strips along the border of any rectangle can be cut without crossing any rectangle)

Typical

$|\texttt{ORTHOTOPE}| > 1$
$\texttt{ORTHOTOPE.siz} > 0$
$|\texttt{ORTHOTOPES}| > 1$

Symmetries

- Items of ORTHOTOPES are permutable.
- One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPES.orth.

Arg. properties

Contractible wrt. ORTHOTOPES.

See also

common keyword: DIFFN *(geometrical constraint,orthotope)*, DIFFN_INCLUDE *(geometrical constraint,orthotope,positioning constraint)*.

implies: DIFFN_INCLUDE.

used in graph description: TWO_ORTH_COLUMN.

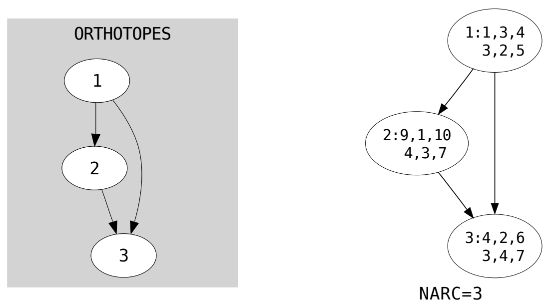
Keywords

constraint type: decomposition.

geometry: geometrical constraint, positioning constraint, orthotope, guillotine cut.

Arc input(s)	ORTHOTOPES
Arc generator	$CLIQUE(<) \mapsto$ collection(orthotopes1, orthotopes2)
Arc arity	2
Arc constraint(s)	TWO_ORTH_COLUMN(orthotopes1.orth, orthotopes2.orth, DIM)
Graph property(ies)	**NARC**$= \lvert$ORTHOTOPES$\rvert * (\lvert$ORTHOTOPES$\rvert - 1)/2$

Graph model Since showing all items produces too big graphs, parts (A) and (B) of Figure 5.294 respectively show the initial and final graph associated with the first three items of the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.294: Initial and final graph of the DIFFN_COLUMN constraint

5.122 DIFFN_INCLUDE

DESCRIPTION LINKS GRAPH

Origin CHIP: option guillotine cut (include) of DIFFN.

Constraint DIFFN_INCLUDE(ORTHOTOPES, DIM)

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments

ORTHOTOPES : collection(orth − ORTHOTOPE)
DIM : int

Restrictions

|ORTHOTOPE| $>$ 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz $\geq$ 0
ORTHOTOPE.ori $\leq$ ORTHOTOPE.end
required(ORTHOTOPES, orth)
same_size(ORTHOTOPES, orth)
DIM $>$ 0
DIM $\leq$ |ORTHOTOPE|
DIFFN(ORTHOTOPES)

Purpose

Extension of the generalised multi-dimensional non-overlapping diffn constraint. Holds if, for each pair of orthotopes (O_1, O_2) the following conditions hold:

- O_1 and O_2 do not overlap. Two orthotopes do not overlap if one of the orthotopes has zero size or if there exists at least one dimension where their projections do not overlap.
- Let P_1 and P_2 respectively denote the projections of O_1 and O_2 onto dimension DIM. If P_1 and P_2 overlap then, either P_1 is included in P_2, either P_2 is included in P_1.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{l}\texttt{orth}-\langle\texttt{ori}-8\ \texttt{siz}-1\ \texttt{end}-9,\texttt{ori}-4\ \texttt{siz}-1\ \texttt{end}-5\rangle,\\ \texttt{orth}-\left\langle\begin{array}{lll}\texttt{ori}-9 & \texttt{siz}-1 & \texttt{end}-10,\\ \texttt{ori}-4 & \texttt{siz}-3 & \texttt{end}-7\end{array}\right\rangle,\\ \texttt{orth}-\langle\texttt{ori}-6\ \texttt{siz}-3\ \texttt{end}-9,\texttt{ori}-5\ \texttt{siz}-2\ \texttt{end}-7\rangle,\\ \texttt{orth}-\langle\texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4,\texttt{ori}-6\ \texttt{siz}-1\ \texttt{end}-7\rangle,\\ \texttt{orth}-\langle\texttt{ori}-4\ \texttt{siz}-2\ \texttt{end}-6,\texttt{ori}-3\ \texttt{siz}-4\ \texttt{end}-7\rangle,\\ \texttt{orth}-\left\langle\begin{array}{lll}\texttt{ori}-6 & \texttt{siz}-4 & \texttt{end}-10,\\ \texttt{ori}-1 & \texttt{siz}-1 & \texttt{end}-2\end{array}\right\rangle,\\ \texttt{orth}-\left\langle\begin{array}{lll}\texttt{ori}-10 & \texttt{siz}-1 & \texttt{end}-11,\\ \texttt{ori}-1 & \texttt{siz}-1 & \texttt{end}-2\end{array}\right\rangle,\\ \texttt{orth}-\left\langle\begin{array}{lll}\texttt{ori}-6 & \texttt{siz}-5 & \texttt{end}-11,\\ \texttt{ori}-2 & \texttt{siz}-2 & \texttt{end}-4\end{array}\right\rangle,\\ \texttt{orth}-\langle\texttt{ori}-6\ \texttt{siz}-2\ \texttt{end}-8,\texttt{ori}-4\ \texttt{siz}-1\ \texttt{end}-5\rangle,\\ \texttt{orth}-\langle\texttt{ori}-1\ \texttt{siz}-5\ \texttt{end}-6,\texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3\rangle,\\ \texttt{orth}-\langle\texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4,\texttt{ori}-3\ \texttt{siz}-2\ \texttt{end}-5\rangle,\\ \texttt{orth}-\langle\texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3,\texttt{ori}-5\ \texttt{siz}-1\ \texttt{end}-6\rangle\end{array}\right\rangle,1\end{array}\right)$$

Figure 5.295 represents the respective position of the twelve rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The DIFFN_INCLUDE constraint holds since (1) the twelve rectangles do not overlap and since (2) when their projection onto dimension $\texttt{DIM} = 1$ overlap one of the projections is included within the other one.

Typical

$|\texttt{ORTHOTOPE}| > 1$
$\texttt{ORTHOTOPE.siz} > 0$
$|\texttt{ORTHOTOPES}| > 1$

Symmetries

- Items of ORTHOTOPES are permutable.
- One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPES.orth.

Arg. properties

Contractible wrt. ORTHOTOPES.

See also

common keyword: DIFFN *(geometrical constraint,orthotope)*, DIFFN_COLUMN *(geometrical constraint,orthotope,positioning constraint)*.

implied by: DIFFN_COLUMN.

used in graph description: TWO_ORTH_COLUMN.

Keywords

constraint type: decomposition.

geometry: geometrical constraint, positioning constraint, orthotope.

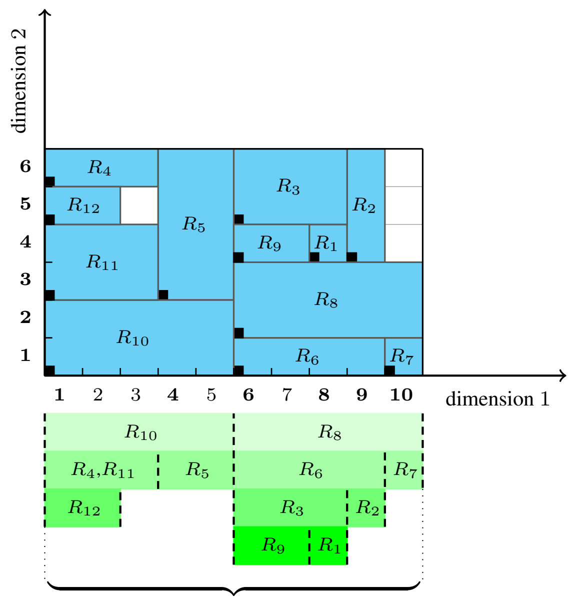
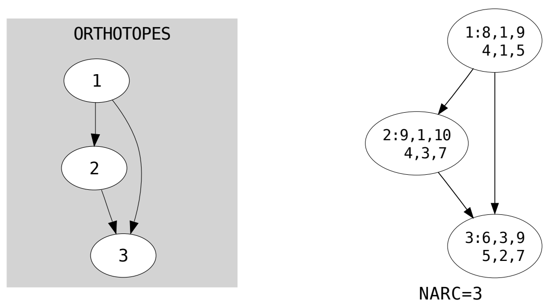


Figure 5.295: Illustration of the **Example** slot: twelve non-overlapping rectangles such that, for each pair of rectangles R_i, R_j $(1 \leq i < j \leq 12)$, if the projections onto dimension 1 of rectangles R_i and R_j intersect then one of the projections is included within the other projection

Arc input(s)	ORTHOTOPES
Arc generator	*CLIQUE*(<) ↦collection(orthotopes1, orthotopes2)
Arc arity	2
Arc constraint(s)	TWO_ORTH_INCLUDE(orthotopes1.orth, orthotopes2.orth, DIM)
Graph property(ies)	**NARC**= |ORTHOTOPES| ∗ (|ORTHOTOPES| − 1)/2

Graph model Since showing all items produces too big graphs, parts (A) and (B) of Figure 5.296 respectively show the initial and final graph associated with the first three items of the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.296: Initial and final graph of the DIFFN_INCLUDE constraint

5.123 DISCREPANCY

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [181] and [434]

Constraint DISCREPANCY(VARIABLES, K)

Arguments

VARIABLES	:	collection(var−dvar, bad−sint)
K	:	int

Restrictions

required(VARIABLES, var)
required(VARIABLES, bad)
$K \geq 0$
$K \leq |VARIABLES|$

Purpose K is the number of variables of the collection VARIABLES that take their values in their respective sets of bad values.

Example

$$\left(\left\langle\begin{array}{ll} \texttt{var}-4 & \texttt{bad}-\{1,4,6\}, \\ \texttt{var}-5 & \texttt{bad}-\{0,1\}, \\ \texttt{var}-5 & \texttt{bad}-\{1,6,9\}, \\ \texttt{var}-4 & \texttt{bad}-\{1,4\}, \\ \texttt{var}-1 & \texttt{bad}-\emptyset \end{array}\right\rangle, 2\right)$$

The DISCREPANCY constraint holds since exactly $K = 2$ variables (i.e., the first and fourth variables) of the VARIABLES collection take their values within their respective sets of bad values.

Typical

$|VARIABLES| > 1$
$K < |VARIABLES|$

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VARIABLES.bad can be swapped; all occurrences of a value in VARIABLES.var or VARIABLES.bad can be renamed to any unused value.

Arg. properties

- Functional dependency: K determined by VARIABLES.
- Aggregate: VARIABLES(union), K(+).

Remark Limited discrepancy search was first introduced by M. L. Ginsberg and W. D. Harvey as a search technique in [204]. Later on, discrepancy based filtering was presented in the PhD thesis of F. Focacci [181, pages 171–172]. Finally the DISCREPANCY constraint was explicitly defined in the PhD thesis of W.-J. van Hoeve [434, page 104].

See also

common keyword: AMONG *(counting constraint)*.

used in graph description: IN_SET.

Keywords

constraint arguments: pure functional dependency.

constraint type: value constraint, counting constraint.

filtering: arc-consistency.

heuristics: heuristics, limited discrepancy search.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	IN_SET(variables.var, variables.bad)
Graph property(ies)	**NARC** = K

Graph model The arc constraint corresponds to the constraint in_set(variables.var, variables.bad) defined in this catalogue. We employ the *SELF* arc generator in order to produce an initial graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.297 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

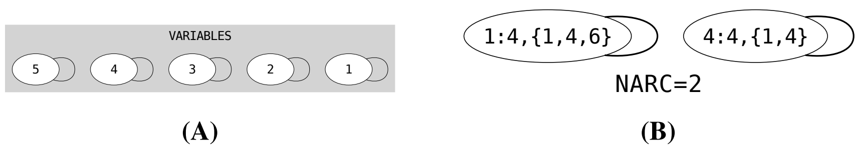


Figure 5.297: Initial and final graph of the DISCREPANCY constraint

5.124 DISJ

DESCRIPTION LINKS GRAPH

Origin [298]

Constraint DISJ(TASKS)

Argument

$$\texttt{TASKS} : \texttt{collection}\begin{pmatrix}\texttt{start}-\texttt{dvar},\\ \texttt{duration}-\texttt{dvar},\\ \texttt{before}-\texttt{svar},\\ \texttt{position}-\texttt{dvar}\end{pmatrix}$$

Restrictions

required(TASKS, [start, duration, before, position])
$\texttt{TASKS.duration} \geq 1$
$\texttt{TASKS.position} \geq 0$
$\texttt{TASKS.position} < |\texttt{TASKS}|$

Purpose

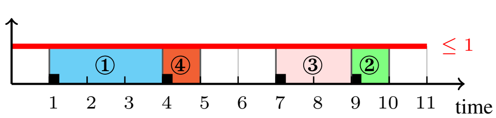
All the tasks of the collection TASKS should not overlap. For a given task t the attributes before and position respectively correspond to the set of tasks starting before task t (assuming that the first task is labelled by 1) and to the position of task t (assuming that the first task has position 0).

Example

$$\left(\left\langle\begin{array}{llll}\texttt{start}-1 & \texttt{duration}-3 & \texttt{before}-\emptyset & \texttt{p}-0,\\ \texttt{start}-9 & \texttt{duration}-1 & \texttt{before}-\{1,3,4\} & \texttt{p}-3,\\ \texttt{start}-7 & \texttt{duration}-2 & \texttt{before}-\{1,4\} & \texttt{p}-2,\\ \texttt{start}-4 & \texttt{duration}-1 & \texttt{before}-\{1\} & \texttt{p}-1\end{array}\right\rangle\right)$$

(p for position)

Figure 5.298 shows the tasks of the example. Since these tasks do not overlap the DISJ constraint holds.



TASKS

①	s − 1	d − 3	b − $\emptyset$	p − 0
②	s − 9	d − 1	b − $\{1,3,4\}$	p − 3
③	s − 7	d − 2	b − $\{1,4\}$	p − 2
④	s − 4	d − 1	b − $\{1\}$	p − 1

(s for start, d for duration, b for before, p for position)

Figure 5.298: Tasks corresponding to the **Example** slot

Typical $|\texttt{TASKS}| > 1$

Symmetries

- One and the same constant can be added to the `start` attribute of all items of `TASKS`.
- `TASKS.duration` can be decreased to any value ≥ 1.

Usage

The DISJ constraint was originally applied [298] to solve the *open-shop* problem.

Remark

This constraint is similar to the DISJUNCTIVE constraint. In addition to the `start` and the `duration` attributes of a task t, the DISJ constraint introduces a set variable `before` that represents the set of tasks that end before the start of task t as well as a domain variable `position` that gives the absolute order of task t in the resource. Since it assumes that the first task has position 0 we have that, for a given task t, the number of elements of its `before` attribute is equal to the value of its `position` attribute.

Algorithm

The main idea of the algorithm is to apply in a systematic way shaving on the `position` attribute of a task. It is implemented in **Gecode** [385].

See also

common keyword: DISJUNCTIVE *(scheduling constraint)*.

used in graph description: IN_SET.

Keywords

complexity: sequencing with release times and deadlines.

constraint arguments: constraint involving set variables.

constraint type: scheduling constraint, resource constraint, decomposition.

Arc input(s)	TASKS
Arc generator	$CLIQUE(\neq) \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• $\bigvee \left(\begin{array}{l} \texttt{tasks1.start} + \texttt{tasks1.duration} \leq \texttt{tasks2.start}, \\ \texttt{tasks2.start} + \texttt{tasks2.duration} \leq \texttt{tasks1.start} \end{array} \right)$ • $\texttt{tasks1.start} + \texttt{tasks1.duration} \leq \texttt{tasks2.start} \Leftrightarrow$ IN_SET(tasks1.key, tasks2.before) • $\texttt{tasks1.start} + \texttt{tasks1.duration} \leq \texttt{tasks2.start} \Leftrightarrow$ $\texttt{tasks1.position} < \texttt{tasks2.position}$
Graph property(ies)	**NARC**$= \lvert\texttt{TASKS}\rvert * \lvert\texttt{TASKS}\rvert - \lvert\texttt{TASKS}\rvert$

Graph model We generate a *clique* with a non-overlapping constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph. For two tasks t_1 and t_2, the three conditions of the arc constraint respectively correspond to:

- The fact that t_1 ends before the start of t_2 or that t_2 ends before the start of t_1.
- The equivalence between the fact that t_1 ends before the start of t_2 and the fact that the identifier of task t_1 belongs to the `before` attribute of task t_2.
- The equivalence between the fact that t_1 ends before the start of t_2 and the fact that the `position` attribute of task t_1 is strictly less than the `position` attribute of task t_2.

Parts (A) and (B) of Figure 5.299 respectively show the initial and final graph associated with the **Example** slot. The DISJ constraint holds since all the arcs of the initial graph belong to the final graph: all the non-overlapping constraints holds.

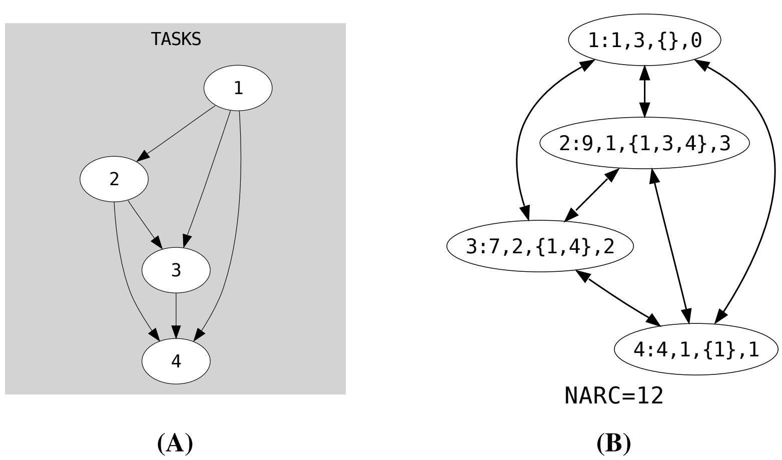


Figure 5.299: Initial and final graph of the DISJ constraint

5.125 DISJOINT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from ALLDIFFERENT.

Constraint DISJOINT(VARIABLES1, VARIABLES2)

Arguments

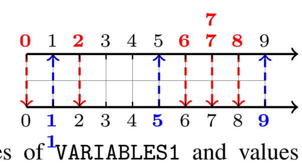
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

Restrictions

required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose Each variable of the collection VARIABLES1 should take a value that is distinct from all the values assigned to the variables of the collection VARIABLES2.

Example $(\langle 1, 9, 1, 5\rangle, \langle 2, 7, 7, 0, 6, 8\rangle)$



In this example, values $1, 5, 9$ are used by the variables of VARIABLES1 and values $0, 2, 6, 7, 8$ by the variables of VARIABLES2. Since there is no intersection between the two previous sets of values the DISJOINT constraint holds.

All solutions Figure 5.300 gives all solutions to the following non ground instance of the DISJOINT constraint: $U_1 \in [0, 2]$, $U_2 \in [1, 2]$, $U_3 \in [1, 2]$, $V_1 \in [0, 1]$, $V_2 \in [1, 2]$, DISJOINT$(\langle U_1, U_2, U_3\rangle, \langle V_1, V_2\rangle)$.

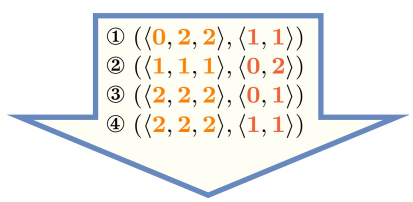


Figure 5.300: All solutions corresponding to the non ground example of the DISJOINT constraint of the **All solutions** slot

Typical

$|$VARIABLES1$| > 1$
$|$VARIABLES2$| > 1$

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any value of VARIABLES1.var.
- An occurrence of a value of VARIABLES2.var can be replaced by any value of VARIABLES2.var.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Arg. properties

- Contractible wrt. VARIABLES1.
- Contractible wrt. VARIABLES2.

Remark

Despite the fact that this is not an uncommon constraint, it can not be modelled in a compact way neither with a *disequality* constraint (i.e., two given variables have to take distinct values) nor with the ALLDIFFERENT constraint. The DISJOINT constraint can bee seen as a special case of the COMMON(NCOMMON1, NCOMMON2, VARIABLES1, VARIABLES2) constraint where NCOMMON1 and NCOMMON2 are both set to 0.

MiniZinc (http://www.minizinc.org/) has a DISJOINT constraint between two set variables rather than between two collections of variables.

Algorithm

Let us note:

- n_1 the minimum number of distinct values taken by the variables of the collection VARIABLES1.
- n_2 the minimum number of distinct values taken by the variables of the collection VARIABLES2.
- n_{12} the maximum number of distinct values taken by the union of the variables of VARIABLES1 and VARIABLES2.

One invariant to maintain for the DISJOINT constraint is $n_1 + n_2 \leq n_{12}$. A lower bound of n_1 and n_2 can be obtained by using the algorithms provided in [29, 46]. An exact upper bound of n_{12} can be computed by using a bipartite matching algorithm.

Systems

DISJOINT in **MiniZinc**.

Used in

K_DISJOINT.

See also

generalisation: DISJOINT_TASKS *(*variable *replaced by* task*)*.

implies: ALLDIFFERENT_ON_INTERSECTION, LEX_DIFFERENT.

system of constraints: K_DISJOINT.

Keywords

characteristic of a constraint: disequality, automaton, automaton with array of counters.

constraint arguments: constraint between two collections of variables.

constraint type: value constraint.

filtering: bipartite matching.

modelling: empty intersection.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NARC** = 0

Graph model *PRODUCT* is used in order to generate the arcs of the graph between all variables of VARIABLES1 and all variables of VARIABLES2. Since we use the graph property **NARC** = 0 the final graph will be empty. Figure 5.301 shows the initial graph associated with the **Example** slot. Since we use the **NARC** = 0 graph property the final graph is empty.

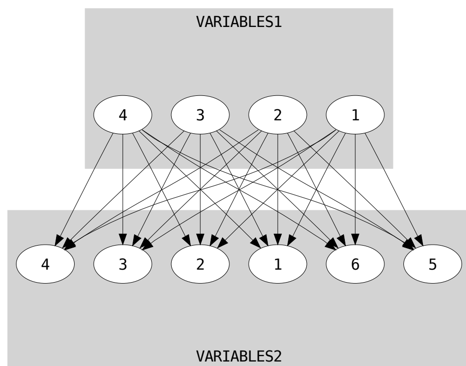


Figure 5.301: Initial graph of the DISJOINT constraint (the final graph is empty)

Signature Since 0 is the smallest number of arcs of the final graph we can rewrite **NARC** = 0 to **NARC** $\leq$ 0. This leads to simplify $\overline{\textbf{NARC}}$ to $\underline{\textbf{NARC}}$.

Automaton Figure 5.302 depicts the automaton associated with the DISJOINT constraint. To each variable $\texttt{VAR1}_i$ of the collection VARIABLES1 corresponds a signature variable S_i that is equal to 0. To each variable $\texttt{VAR2}_i$ of the collection VARIABLES2 corresponds a signature variable $S_{i+|\texttt{VARIABLES1}|}$ that is equal to 1.

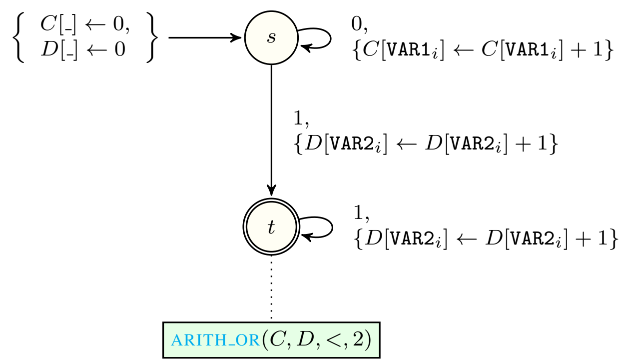


Figure 5.302: Automaton of the DISJOINT(VARIABLES1, VARIABLES2) constraint, where state s handles variables of the collection VARIABLES1 and state t handles variables of the collection VARIABLES2

5.126 DISJOINT_SBOXES

► ▷ ◁ ◄ DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [349]

Constraint DISJOINT_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonym DISJOINT.

Types

```
VARIABLES : collection(v-dvar)
INTEGERS  : collection(v-int)
POSITIVES : collection(v-int)
```

Arguments

```
K       : int
DIMS    : sint
OBJECTS : collection(oid-int, sid-dvar, x - VARIABLES)
SBOXES  : collection(sid-int, t - INTEGERS, l - POSITIVES)
```

Restrictions

$|\mathtt{VARIABLES}| \geq 1$
$|\mathtt{INTEGERS}| \geq 1$
$|\mathtt{POSITIVES}| \geq 1$
required(VARIABLES, v)
$|\mathtt{VARIABLES}| = \mathtt{K}$
required(INTEGERS, v)
$|\mathtt{INTEGERS}| = \mathtt{K}$
required(POSITIVES, v)
$|\mathtt{POSITIVES}| = \mathtt{K}$
$\mathtt{POSITIVES.v} > 0$
$\mathtt{K} > 0$
$\mathtt{DIMS} \geq 0$
$\mathtt{DIMS} < \mathtt{K}$
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
$\mathtt{OBJECTS.oid} \geq 1$
$\mathtt{OBJECTS.oid} \leq |\mathtt{OBJECTS}|$
$\mathtt{OBJECTS.sid} \geq 1$
$\mathtt{OBJECTS.sid} \leq |\mathtt{SBOXES}|$
$|\mathtt{SBOXES}| \geq 1$
required(SBOXES, [sid, t, l])
$\mathtt{SBOXES.sid} \geq 1$
$\mathtt{SBOXES.sid} \leq |\mathtt{SBOXES}|$
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i \neq j$, O_i and O_j are disjoint with respect to a set of dimensions depicted by `DIMS`. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a `K`-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id `sid`, shift offset `t`, and sizes `l`. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier `oid`, shape id `sid` and origin `x`.
Two objects O_i and object O_j are *disjoint* with respect to a set of dimensions depicted by `DIMS` if and only if for all shifted box s_i associated with O_i and for all shifted box s_j associated with O_j there exists at least one dimension $d \in$ `DIMS` such that (1) the origin of s_i in dimension d is strictly greater than the end of s_j in dimension d, or (2) the origin of s_j in dimension d is strictly greater than the end of s_i in dimension d.

Example

$$\left(\begin{array}{l} 2, \{0,1\}, \\ \left\langle\begin{array}{lll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,1\rangle, \\ \texttt{oid}-3 & \texttt{sid}-4 & \texttt{x}-\langle 2,4\rangle \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 1,0\rangle & \texttt{l}-\langle 1,3\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,2\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,1\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-4 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle \end{array}\right\rangle \end{array}\right)$$

Figure 5.303 shows the objects of the example. Since these objects are pairwise disjoint the DISJOINT_SBOXES constraint holds.

Typical

$|$`OBJECTS`$| > 1$

Symmetries

- Items of `OBJECTS` are permutable.
- Items of `SBOXES` are permutable.
- `SBOXES.l.v` can be decreased to any value ≥ 1.

Arg. properties

Suffix-contractible wrt. `OBJECTS`.

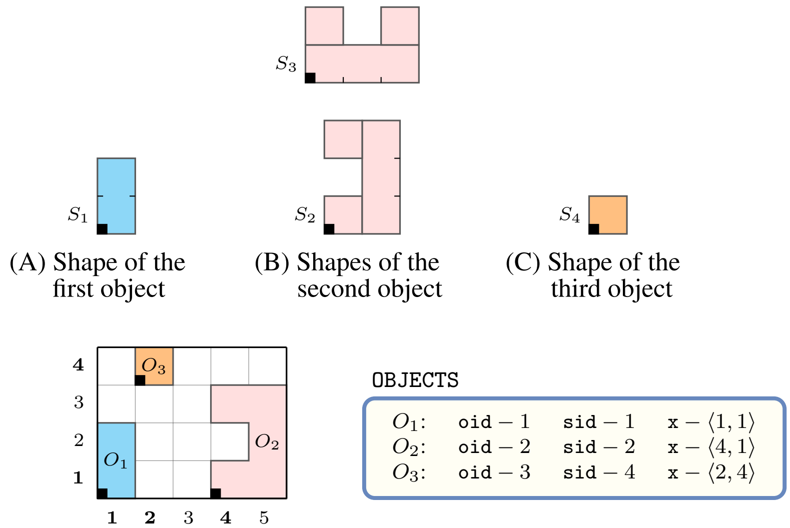
Remark

One of the eight relations of the *Region Connection Calculus* [349]. Unlike the NON_OVERLAP_SBOXES constraint, which just prevents objects from overlapping, the DISJOINT_SBOXES constraint in addition enforces that borders and corners of objects are not directly in contact.

See also

common keyword: CONTAINS_SBOXES, COVEREDBY_SBOXES, COVERS_SBOXES, EQUAL_SBOXES, INSIDE_SBOXES, MEET_SBOXES *(rcc8)*, NON_OVERLAP_SBOXES *(geometrical constraint,logic)*, OVERLAP_SBOXES *(rcc8)*.

implies: NON_OVERLAP_SBOXES.



(D) Three mutually disjoint objects

Figure 5.303: (D) the three mutually disjoint objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_1, S_2, S_4; (A), (B), (C) shapes S_1, S_2, S_3 and S_4 are respectively made up from 1, 3, 3 and 1 disjoint shifted box.

Keywords

constraint type: logic.

geometry: geometrical constraint, rcc8.

miscellaneous: obscure.

Logic

- $\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D})$
- $\texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D}) + \texttt{S1.l}(\texttt{D})$
- $\texttt{disjoint\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=}$
  $\exists \texttt{D} \in \texttt{Dims}$
  $\bigvee \left(\begin{array}{l} \texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) > \\ \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) \end{array}, \begin{array}{l} \texttt{origin}(\texttt{O2},\texttt{S2},\texttt{D}) > \\ \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \end{array} \right)$
- $\texttt{disjoint\_objects}(\texttt{Dims},\texttt{O1},\texttt{O2}) \stackrel{\text{def}}{=}$
  $\forall \texttt{S1} \in \texttt{sboxes}([\texttt{O1.sid}])$
  $\forall \texttt{S2} \in \texttt{sboxes}\left(\left[\ \texttt{O2.sid}\ \right]\right)$
  $\texttt{disjoint\_sboxes}\left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{S1}, \\ \texttt{O2}, \\ \texttt{S2} \end{array}\right)$
- $\texttt{all\_disjoint}(\texttt{Dims},\texttt{OIDS}) \stackrel{\text{def}}{=}$
  $\forall \texttt{O1} \in \texttt{objects}(\texttt{OIDS})$
  $\forall \texttt{O2} \in \texttt{objects}(\texttt{OIDS})$
  $\begin{array}{l} \texttt{O1.oid} < \\ \texttt{O2.oid} \end{array} \Rightarrow$
  $\texttt{disjoint\_objects}\left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{O2} \end{array}\right)$
- $\texttt{all\_disjoint}(\texttt{DIMENSIONS},\texttt{OIDS})$

5.127 DISJOINT_TASKS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

Derived from DISJOINT.

Constraint

DISJOINT_TASKS(TASKS1, TASKS2)

Arguments

TASKS1 : collection(origin−dvar, duration−dvar, end−dvar)
TASKS2 : collection(origin−dvar, duration−dvar, end−dvar)

Restrictions

require_at_least(2, TASKS1, [origin, duration, end])
TASKS1.duration ≥ 0
TASKS1.origin $\leq$ TASKS1.end
require_at_least(2, TASKS2, [origin, duration, end])
TASKS2.duration ≥ 0
TASKS2.origin $\leq$ TASKS2.end

Purpose

Each task of the collection TASKS1 should not overlap any task of the collection TASKS2. Two tasks overlap if they have an intersection that is strictly greater than zero.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{lll}\texttt{origin}-6 & \texttt{duration}-5 & \texttt{end}-11,\\ \texttt{origin}-8 & \texttt{duration}-2 & \texttt{end}-10\end{array}\right\rangle,\\ \left\langle\begin{array}{lll}\texttt{origin}-2 & \texttt{duration}-2 & \texttt{end}-4,\\ \texttt{origin}-3 & \texttt{duration}-3 & \texttt{end}-6,\\ \texttt{origin}-12 & \texttt{duration}-1 & \texttt{end}-13\end{array}\right\rangle\end{array}\right)$$

Figure 5.304 displays the two groups of tasks (i.e., the tasks of TASKS1 and the tasks of TASKS2). Since no task of the first group overlaps any task of the second group, the DISJOINT_TASKS constraint holds.

Typical

$|$TASKS1$| > 1$
TASKS1.duration > 0
$|$TASKS2$| > 1$
TASKS2.duration > 0

Symmetries

- Arguments are permutable w.r.t. permutation (TASKS1, TASKS2).
- Items of TASKS1 are permutable.
- Items of TASKS2 are permutable.
- One and the same constant can be added to the origin and end attributes of all items of TASKS1 and TASKS2.

Arg. properties

- Contractible wrt. TASKS1.
- Contractible wrt. TASKS2.

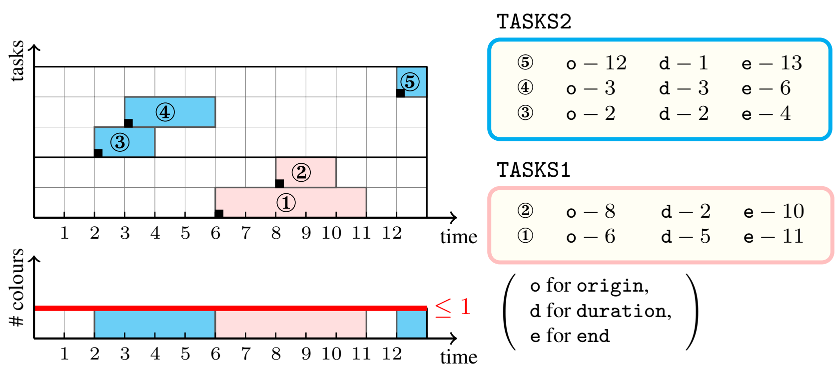


Figure 5.304: The DISJOINT_TASKS solution to the **Example** slot with at most one distinct colour in parallel (tasks in TASKS1 have the pink colour, while tasks in TASKS2 have the blue colour)

Remark Despite the fact that this is not an uncommon constraint, it cannot be modelled in a compact way with a single CUMULATIVE constraint. But it can be expressed by using the COLOURED_CUMULATIVE constraint: We assign a first colour to the tasks of TASKS1 as well as a second distinct colour to the tasks of TASKS2. Finally we set up a limit of 1 for the maximum number of distinct colours allowed at each time point.

Reformulation The DISJOINT_TASKS constraint can be expressed in term of $|\texttt{TASKS1}| \cdot |\texttt{TASKS2}|$ reified constraints. For each task $\texttt{TASKS1}[i]$ ($i \in [1, |\texttt{TASKS1}|]$) and for each task $\texttt{TASKS2}[j]$ ($j \in [1, |\texttt{TASKS2}|]$) we generate a reified constraint of the form $\texttt{TASKS1}[i].\texttt{end} \leq \texttt{TASKS2}[j].\texttt{origin} \vee \texttt{TASKS2}[j].\texttt{end} \leq \texttt{TASKS1}[i].\texttt{origin}$. In addition we also state for each task an arithmetic constraint that states that the end of a task is equal to the sum of its origin and its duration.

Systems DISJOINT in **Choco**.

See also **generalisation:** COLOURED_CUMULATIVE *(tasks colours and limit on maximum number of colours in parallel are explicitly given)*.

specialisation: DISJOINT *(*`task` *replaced by* `variable`*)*.

Keywords **constraint type:** scheduling constraint, temporal constraint.

geometry: non-overlapping.

Arc input(s)	TASKS1
Arc generator	$SELF$ $\mapsto$ collection(tasks1)
Arc arity	1
Arc constraint(s)	tasks1.origin + tasks1.duration = tasks1.end
Graph property(ies)	**NARC** $= \|$TASKS1$\|$

Arc input(s)	TASKS2
Arc generator	$SELF$ $\mapsto$ collection(tasks2)
Arc arity	1
Arc constraint(s)	tasks2.origin + tasks2.duration = tasks2.end
Graph property(ies)	**NARC** $= \|$TASKS2$\|$

Arc input(s)	TASKS1 TASKS2
Arc generator	$PRODUCT$ $\mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.duration > 0 • tasks2.duration > 0 • tasks1.origin $<$ tasks2.end • tasks2.origin $<$ tasks1.end
Graph property(ies)	**NARC** $= 0$

Graph model $PRODUCT$ is used in order to generate the arcs of the graph between all the tasks of the collection TASKS1 and all tasks of the collection TASKS2. The first two graph constraints respectively enforce for each task of TASKS1 and TASKS2 the fact that the end of a task is equal to the sum of its origin and its duration. The arc constraint of the third graph constraint depicts the fact that two tasks overlap. Therefore, since we use the graph property **NARC** $= 0$ the final graph associated with the third graph constraint will be empty and no task of TASKS1 will overlap any task of TASKS2. Figure 5.305 shows the initial graph of the third graph constraint associated with the **Example** slot. Because of the graph property **NARC** $= 0$ the corresponding final graph is empty.

Signature Since TASKS1 is the maximum number of arcs of the final graph associated with the first graph constraint we can rewrite **NARC** $= |$TASKS1$|$. This leads to simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

We can apply a similar remark for the second graph constraint.

Finally, since 0 is the smallest number of arcs of the final graph we can rewrite **NARC** $=$ 0 to **NARC** ≤ 0. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\underline{\mathbf{NARC}}$.

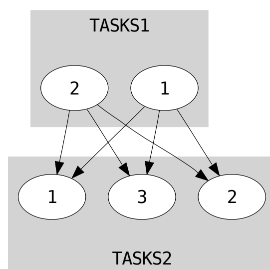


Figure 5.305: Initial graph of the DISJOINT_TASKS constraint (the final graph is empty)

5.128 DISJUNCTIVE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [100]

Constraint DISJUNCTIVE(TASKS)

Synonym ONE_MACHINE.

Argument TASKS : collection(origin−dvar, duration−dvar)

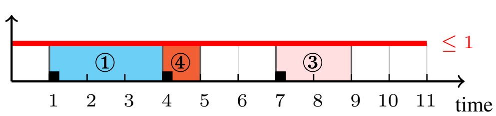
Restrictions required(TASKS, [origin, duration])
TASKS.duration ≥ 0

Purpose All the tasks of the collection TASKS that have a duration strictly greater than 0 should not overlap.

Example

$$\left(\left\langle\begin{array}{ll} \texttt{origin}-1 & \texttt{duration}-3, \\ \texttt{origin}-2 & \texttt{duration}-0, \\ \texttt{origin}-7 & \texttt{duration}-2, \\ \texttt{origin}-4 & \texttt{duration}-1 \end{array}\right\rangle\right)$$

Figure 5.306 shows the tasks with non-zero duration of the example. Since these tasks do not overlap the DISJUNCTIVE constraint holds.



TASKS

①	origin − 1	duration − 3
③	origin − 7	duration − 2
④	origin − 4	duration − 1

Figure 5.306: Tasks with non-zero duration of the **Example** slot

All solutions Figure 5.307 gives all solutions to the following non ground instance of the DISJUNCTIVE constraint: $O_1 \in [2,5]$, $D_1 \in [2,4]$, $O_2 \in [2,4]$, $D_2 \in [1,6]$, $O_3 \in [3,6]$, $D_3 \in [4,4]$, $O_4 \in [2,7]$, $D_4 \in [1,3]$, DISJUNCTIVE($\langle O_1\ D_1,\ O_2\ D_2,\ O_3\ D_3,\ O_4\ D_4\rangle$).

Typical $|\texttt{TASKS}| > 2$
TASKS.duration ≥ 1

Symmetries

- Items of TASKS are permutable.
- TASKS.duration can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin attribute of all items of TASKS.

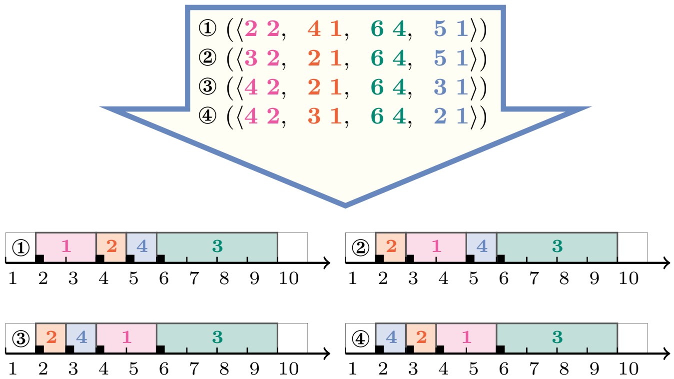


Figure 5.307: All solutions corresponding to the non ground example of the DISJUNCTIVE constraint of the **All solutions** slot

Arg. properties Contractible wrt. TASKS.

Usage The DISJUNCTIVE constraint occurs in many resource scheduling problems in order to model a resource that can not be shared. This means that tasks using this resource can not overlap in time. Quite often DISJUNCTIVE constraints are used together with precedence constraints. A precedence constraint between two tasks models the fact that the processing of a task has to be postponed until an other task is completed. Such mix of disjunctive and precedence constraints occurs, for example, in job-shop problems.

Remark Some systems like Ilog CP Optimizer also imposes that zero duration tasks do not overlap non-zero duration tasks.

A soft version of this constraint, under the hypothesis that all durations are fixed, was presented by P. Baptiste *et al.* in [19]. In this context the goal was to perform as many tasks as possible within their respective due-dates.

When all tasks have the same (fixed) duration the DISJUNCTIVE constraint can be reformulated as an ALL_MIN_DIST constraint for which a filtering algorithm achieving bound-consistency is available [12].

Within the context of linear programming [226, page 386] provides several relaxations of the DISJUNCTIVE constraint.

Some solvers use in a pre-processing phase, while stating precedence and cumulative constraints, *an algorithm for automatically extracting large cliques* [97] from a set of tasks that should not pairwise overlap (i.e., two tasks t_i and t_j can not overlap either, because t_i ends before the start of t_j, either because the sum of resource consumption of t_i and t_j exceeds the capacity of a cumulative resource that both tasks use) in order to state DISJUNCTIVE constraints.

Algorithm We have four main families of methods for handling the DISJUNCTIVE constraint:

- Methods based on the compulsory part [261] of the tasks (also called time-tabling methods). These methods determine the time slots which for sure are occupied by a given task, an propagate back this information to the attributes of each task (i.e., the origin and the duration). Because of their simplicities, these methods have been originally used for handling the DISJUNCTIVE constraint. Even if they propagate less than the other methods they can in practice handle a large number of tasks. To our best knowledge no efficient incremental algorithm devoted to this problem was published up to now (i.e., September 2006).

- Methods based on constructive disjunction. The idea is to try out each alternative of a disjunction (e.g., given two tasks t_1 and t_2 that should not overlap, we successively assume that t_1 finishes before t_2, and that t_2 finishes before t_1) and to remove values that were pruned in both alternatives.
- Methods based on *edge-finding*. Given a set of tasks $\mathcal{T}$, edge-finding determines that some task must, can, or cannot execute first or last in $\mathcal{T}$. Efficient edge-finding algorithms for handling the DISJUNCTIVE constraint were originally described in [101, 102] and more recently in [443, 315].
- Methods that, for any task t, consider the maximal number of tasks that can end up before the start of task t as well as the maximal number of tasks that can start after the end of task t [453].

All these methods are usually used for adjusting the minimum and maximum values of the variables of the DISJUNCTIVE constraint. However some systems use these methods for pruning the full domain of the variables. Finally, *Jackson priority rule* [236] provides a necessary condition [102] for the DISJUNCTIVE constraint. Given a set of tasks $\mathcal{T}$, it consists to progressively schedule all tasks of $\mathcal{T}$ in the following way:

- It assigns to the first possible time point (i.e., the earliest start of all tasks of $\mathcal{T}$) the available task with minimal latest end. In this context, available means a task for which the earliest start is less than or equal to the considered time point.
- It continues by considering the next time point until all the tasks are completely scheduled.

In 2014, assuming that the tasks are sorted, and using a union-find data structure, linear-time filtering algorithms [169] wrt the number of tasks were obtained for time-tabling, overload check, and detectable precedences.

Systems DISJUNCTIVE in **Choco**, UNARY in **Gecode**.

See also **common keyword:** CALENDAR, DISJ, DISJUNCTIVE_OR_SAME_END, DISJUNCTIVE_OR_SAME_START *(scheduling constraint)*.

generalisation: CUMULATIVE *(*`task heights` *and* `resource limit` *are not necessarly all equal to* 1*)*, DIFFN *(*`task` *of* `heigth` 1 *replaced by orthotope)*.

implied by: PRECEDENCE.

implies: DISJUNCTIVE_OR_SAME_END, DISJUNCTIVE_OR_SAME_START.

specialisation: ALL_MIN_DIST *(*`line segment` *replaced by* `line segment`*, of same length)*, ALLDIFFERENT *(*`task` *replaced by* `variable`*)*.

Keywords **characteristic of a constraint:** core, sort based reformulation.

complexity: sequencing with release times and deadlines.

constraint type: scheduling constraint, resource constraint, decomposition.

filtering: compulsory part, constructive disjunction, Phi-tree, minimum task duration.

modelling: disjunction, sequence dependent set-up, zero-duration task.

modelling exercises: sequence dependent set-up.

problems: maximum clique.

Cond. implications

- DISJUNCTIVE(TASKS)
  with minval(TASKS.duration) > 0
  implies ALLDIFFERENT(TASKS.origin).
- DISJUNCTIVE(TASKS)
  with minval(TASKS.duration) > 0
  implies ALLDIFFERENT_CST(VARIABLES : TASKS).

Arc input(s) TASKS

Arc generator $CLIQUE(<) \mapsto$ collection(tasks1, tasks2)

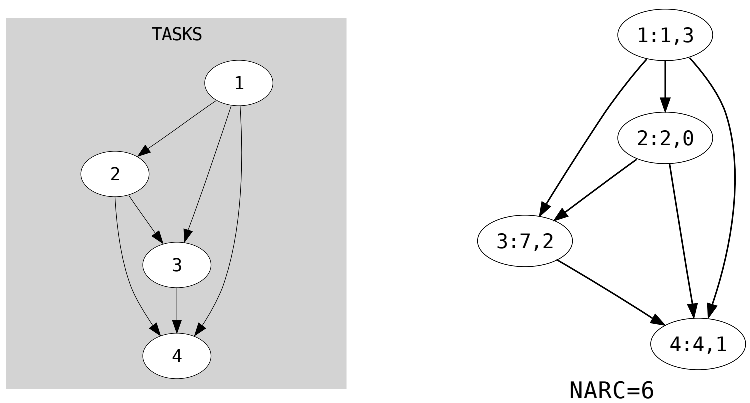
Arc arity 2

Arc constraint(s)
$$\bigvee \begin{pmatrix} \texttt{tasks1.duration} = 0, \\ \texttt{tasks2.duration} = 0, \\ \texttt{tasks1.origin} + \texttt{tasks1.duration} \leq \texttt{tasks2.origin}, \\ \texttt{tasks2.origin} + \texttt{tasks2.duration} \leq \texttt{tasks1.origin} \end{pmatrix}$$

Graph property(ies) $\mathbf{NARC} = |\texttt{TASKS}| * (|\texttt{TASKS}| - 1)/2$

Graph model We generate a *clique* with a non-overlapping constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Parts (A) and (B) of Figure 5.308 respectively show the initial and final graph associated with the **Example** slot. The DISJUNCTIVE constraint holds since all the arcs of the initial graph belong to the final graph: all the non-overlapping constraints holds.



(A) **(B)**

Figure 5.308: Initial and final graph of the DISJUNCTIVE constraint

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint DISJUNCTIVE($\langle\langle 2\ 2\rangle, \langle 4\ 5\rangle, \langle 8\ 2\rangle\rangle$) hold?

B. Does the constraint DISJUNCTIVE($\langle\langle 1\ 3\rangle, \langle 4\ 4\rangle, \langle 8\ 2\rangle\rangle$) hold?

C. Does the constraint DISJUNCTIVE($\langle\langle 3\ 2\rangle, \langle 4\ 0\rangle, \langle 7\ 3\rangle\rangle$) hold?

[a]Hint: go back to the definition of DISJUNCTIVE.

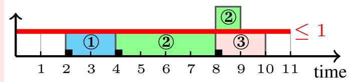
SOLUTION TO EXERCISE 1

A. *No, since tasks ② and ③ overlap at instant* 8.

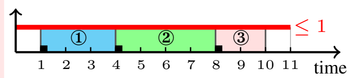
B. *Yes, since the three tasks do not overlap:*

- *Tasks ① and ② do not overlap since task ① ends before task ②,*
- *Tasks ① and ③ do not overlap since task ① ends before task ③,*
- *Tasks ② and ③ do not overlap since task ② ends before task ③.*

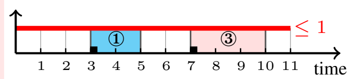
C. *Yes, since tasks ① and ③ do not overlap and since the duration of task ② is zero.*



(A)



(B)



(C)

5.129 DISJUNCTIVE_OR_SAME_END

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Scheduling.

Constraint DISJUNCTIVE_OR_SAME_END(TASKS)

Synonyms SAME_END_OR_DISJUNCTIVE, NON_OVERLAP_OR_SAME_END, SAME_END_OR_NON_OVERLAP.

Argument TASKS : collection(origin−dvar, duration−dvar)

Restrictions
required(TASKS, [origin, duration])
TASKS.duration ≥ 0

Purpose

All pairs of tasks of the collection TASKS that have a duration strictly greater than 0 should either not overlap either have the same end, i.e. $\forall i \in [1, |\mathtt{TASKS}|], \forall j \in [i+1, |\mathtt{TASKS}|] : \mathtt{TASKS}[i].\mathtt{duration} = 0 \vee \mathtt{TASKS}[j].\mathtt{duration} = 0 \vee \mathtt{TASKS}[i].\mathtt{origin} + \mathtt{TASKS}[i].\mathtt{duration} \leq \mathtt{TASKS}[j].\mathtt{origin} \vee \mathtt{TASKS}[j].\mathtt{origin} + \mathtt{TASKS}[j].\mathtt{duration} \leq \mathtt{TASKS}[i].\mathtt{origin} \vee \mathtt{TASKS}[i].\mathtt{origin} + \mathtt{TASKS}[i].\mathtt{duration} = \mathtt{TASKS}[j].\mathtt{origin} + \mathtt{TASKS}[j].\mathtt{duration}$.

Example

$$\left(\left\langle \begin{array}{ll} \mathtt{origin}-4 & \mathtt{duration}-3, \\ \mathtt{origin}-7 & \mathtt{duration}-2, \\ \mathtt{origin}-5 & \mathtt{duration}-2 \end{array} \right\rangle\right)$$

Since the ends of the first and third tasks coincide, and since the second task does neither overlap the first task nor the third task, the DISJUNCTIVE_OR_SAME_END constraint holds.

Typical
$|\mathtt{TASKS}| > 2$
TASKS.duration ≥ 1

Symmetries

- Items of TASKS are permutable.
- TASKS.duration can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin attribute of all items of TASKS.

Arg. properties Contractible wrt. TASKS.

Algorithm Let o_k and d_k (with $k \in [1, |\mathtt{TASKS}|]$) respectively denote the origin and the duration of the k^{th} task of the TASKS collection. The set of forbidden values for the origin of task j wrt task i (with $i, j \in [1, |\mathtt{TASKS}|], i \neq j$) is defined by the union $[\overline{o_i} - d_j + 1, \underline{o_i} + d_i - d_j - 1] \cup [\overline{o_i} + d_i - d_j + 1, \underline{o_i} + d_i - 1]$. One can use the sweep algorithm introduced in [42] to filter the origin of a task wrt all the sets of forbidden values issued from the other tasks.

See also **common keyword:** DISJUNCTIVE, DISJUNCTIVE_OR_SAME_START *(scheduling constraint)*.

implied by: DISJUNCTIVE.

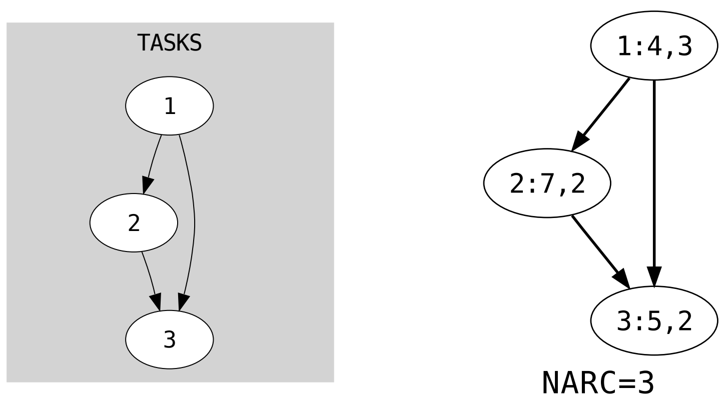
Keywords **constraint type:** scheduling constraint, resource constraint, decomposition.

modelling: disjunction, zero-duration task.

Arc input(s)	`TASKS`
Arc generator	$CLIQUE(<) \mapsto$ `collection`(`tasks1`, `tasks2`)
Arc arity	2
Arc constraint(s)	$\bigvee\left(\begin{array}{l}\texttt{tasks1.duration}=0,\\ \texttt{tasks2.duration}=0,\\ \texttt{tasks1.origin}+\texttt{tasks1.duration}\leq\texttt{tasks2.origin},\\ \texttt{tasks2.origin}+\texttt{tasks2.duration}\leq\texttt{tasks1.origin},\\ \quad\texttt{tasks1.origin}+\texttt{tasks1.duration}=\\ \quad\texttt{tasks2.origin}+\texttt{tasks2.duration}\end{array}\right)$
Graph property(ies)	**NARC**$=|\texttt{TASKS}|*(|\texttt{TASKS}|-1)/2$

Graph model We generate a *clique* with a non-overlapping constraint or a same end constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Parts (A) and (B) of Figure 5.309 respectively show the initial and final graph associated with the **Example** slot. The DISJUNCTIVE_OR_SAME_END constraint holds since all the arcs of the initial graph belong to the final graph.



(A) **(B)**

Figure 5.309: Initial and final graph of the DISJUNCTIVE_OR_SAME_END constraint

5.130 DISJUNCTIVE_OR_SAME_START

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Scheduling.

Constraint DISJUNCTIVE_OR_SAME_START(TASKS)

Synonyms SAME_START_OR_DISJUNCTIVE, NON_OVERLAP_OR_SAME_START, SAME_START_OR_NON_OVERLAP.

Argument TASKS : collection(origin−dvar, duration−dvar)

Restrictions
required(TASKS, [origin, duration])
TASKS.duration ≥ 0

Purpose All pairs of tasks of the collection TASKS that have a duration strictly greater than 0 should either not overlap either have the same start, i.e. $\forall i \in [1, |\text{TASKS}|], \forall j \in [i+1, |\text{TASKS}|]$: $\text{TASKS}[i].\text{duration} = 0 \vee \text{TASKS}[j].\text{duration} = 0 \vee \text{TASKS}[i].\text{origin} + \text{TASKS}[i].\text{duration} \leq \text{TASKS}[j].\text{origin} \vee \text{TASKS}[j].\text{origin} + \text{TASKS}[j].\text{duration} \leq \text{TASKS}[i].\text{origin} \vee \text{TASKS}[i].\text{origin} = \text{TASKS}[j].\text{origin}$.

Example
$$\left(\left\langle\begin{array}{ll}\text{origin}-4 & \text{duration}-3,\\ \text{origin}-7 & \text{duration}-2,\\ \text{origin}-4 & \text{duration}-1\end{array}\right\rangle\right)$$

Since the starts of the first and third tasks coincide, and since the second task does neither overlap the first task nor the third task, the DISJUNCTIVE_OR_SAME_START constraint holds.

Typical
$|\text{TASKS}| > 2$
TASKS.duration ≥ 1

Symmetries
- Items of TASKS are permutable.
- TASKS.duration can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin attribute of all items of TASKS.

Arg. properties Contractible wrt. TASKS.

Algorithm Let o_k and d_k (with $k \in [1, |\text{TASKS}|]$) respectively denote the origin and the duration of the k^{th} task of the TASKS collection. The set of forbidden values for the origin of task j wrt task i (with $i, j \in [1, |\text{TASKS}|], i \neq j$) is defined by the union $[\overline{o_i} - d_j + 1, \underline{o_i} - 1] \cup [\overline{o_i} + 1, \underline{o_i} + d_i - 1]$. One can use the sweep algorithm introduced in [42] to filter the origin of a task wrt all the sets of forbidden values issued from the other tasks.

See also **common keyword:** DISJUNCTIVE, DISJUNCTIVE_OR_SAME_END *(scheduling constraint)*.

implied by: DISJUNCTIVE.

Keywords **constraint type:** scheduling constraint, resource constraint, decomposition.

modelling: disjunction, zero-duration task.

Arc input(s)	TASKS
Arc generator	$CLIQUE(<) \mapsto \texttt{collection}(\texttt{tasks1}, \texttt{tasks2})$
Arc arity	2
Arc constraint(s)	$\bigvee\left(\begin{array}{l}\texttt{tasks1.duration} = 0,\\ \texttt{tasks2.duration} = 0,\\ \texttt{tasks1.origin} + \texttt{tasks1.duration} \leq \texttt{tasks2.origin},\\ \texttt{tasks2.origin} + \texttt{tasks2.duration} \leq \texttt{tasks1.origin},\\ \texttt{tasks1.origin} = \texttt{tasks2.origin}\end{array}\right)$
Graph property(ies)	$\mathbf{NARC} = \lvert\texttt{TASKS}\rvert * (\lvert\texttt{TASKS}\rvert - 1)/2$

Graph model We generate a *clique* with a non-overlapping constraint or a same start constraint between each pair of distinct tasks and state that the number of arcs of the final graph should be equal to the number of arcs of the initial graph.

Parts (A) and (B) of Figure 5.310 respectively show the initial and final graph associated with the **Example** slot. The DISJUNCTIVE_OR_SAME_START constraint holds since all the arcs of the initial graph belong to the final graph.

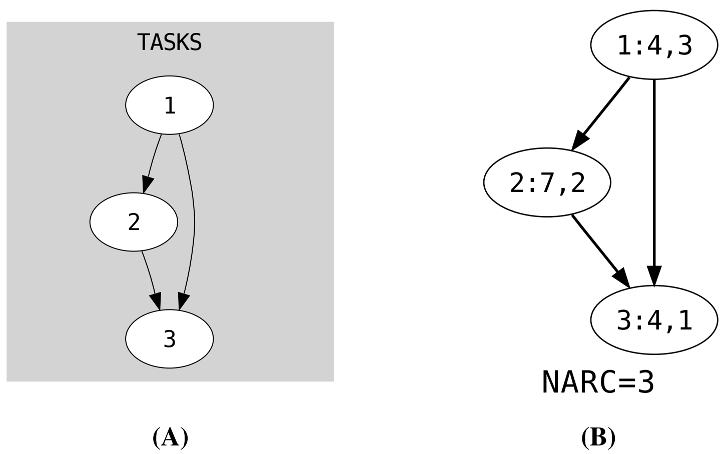


Figure 5.310: Initial and final graph of the DISJUNCTIVE_OR_SAME_START constraint

5.131 DISTANCE

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic constraint.

Constraint DISTANCE(X, Y, Z)

Arguments

X : dvar
Y : dvar
Z : dvar

Restriction $Z \geq 0$

Purpose Enforce the fact that Z is equal to $|X - Y|$.

Example $(5, 7, 2)$

The DISTANCE constraint holds since $2 = |5 - 7|$.

Typical $Z > 0$

Symmetry Arguments are permutable w.r.t. permutation (X, Y) (Z).

Arg. properties Functional dependency: Z determined by X and Y.

Systems DISTANCEEQ in **Choco**, DISTANCE in **JaCoP**, DISTANCE2 in **JaCoP**.

See also **implies:** LEQ_CST.

related: ALL_MIN_DIST *(fixed minimum distance between all pairs of variables of a collection of variables)*, SMOOTH.

Keywords **constraint arguments:** ternary constraint, pure functional dependency.

constraint type: arithmetic constraint, predefined constraint.

modelling: functional dependency.

5.132 DISTANCE_BETWEEN

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint DISTANCE_BETWEEN(DIST, VARIABLES1, VARIABLES2, CTR)

Synonym DISTANCE.

Arguments

```
DIST       : dvar
VARIABLES1 : collection(var-dvar)
VARIABLES2 : collection(var-dvar)
CTR        : atom
```

Restrictions

DIST ≥ 0
DIST $\leq$ |VARIABLES1| $*$ |VARIABLES2| $-$ |VARIABLES1|
required(VARIABLES1, var)
required(VARIABLES2, var)
|VARIABLES1| $=$ |VARIABLES2|
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Let U_i and V_i be respectively the i^{th} and j^{th} variables $(i \neq j)$ of the collection VARIABLES1. In a similar way, let X_i and Y_i be respectively the i^{th} and j^{th} variables $(i \neq j)$ of the collection VARIABLES2. DIST is equal to the number of times one of the following mutually incompatible conditions are true:

- U_i CTR V_i holds and X_i CTR Y_i does not hold,
- X_i CTR Y_i holds and U_i CTR V_i does not hold.

Example

$(2, \langle 3, 4, 6, 2, 4\rangle, \langle 2, 6, 9, 3, 6\rangle, <)$

The DISTANCE_BETWEEN constraint holds since the following DIST $= 2$ conditions are verified:

- VARIABLES1[4].var $= 2 <$ VARIABLES1[1].var $= 3 \wedge$
  VARIABLES2[4].var $= 3 \geq$ VARIABLES2[1].var $= 2$
- VARIABLES2[1].var $= 2 <$ VARIABLES2[4].var $= 3 \wedge$
  VARIABLES1[1].var $= 3 \geq$ VARIABLES1[4].var $= 2$

Typical

DIST > 0
DIST $<$ |VARIABLES1| $*$ |VARIABLES2| $-$ |VARIABLES1|
|VARIABLES1| > 1
CTR $\in [=, \neq]$

Symmetries

- Arguments are permutable w.r.t. permutation (DIST) (VARIABLES1, VARIABLES2) (CTR).
- Items of VARIABLES1 and VARIABLES2 are permutable (*same permutation used*).
- One and the same constant can be added to the var attribute of all items of VARIABLES1.
- One and the same constant can be added to the var attribute of all items of VARIABLES2.

Arg. properties

Functional dependency: DIST determined by VARIABLES1, VARIABLES2 and CTR.

Usage

Measure the distance between two sequences in term of the number of constraint changes. This should be put in contrast to the number of value changes that is sometimes superficial.

See also

common keyword: DISTANCE_CHANGE (*proximity constraint*).

Keywords

constraint arguments: pure functional dependency.

constraint type: proximity constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES1/ VARIABLES2
Arc generator	*CLIQUE*($\neq$) $\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	**DISTANCE**= DIST

Graph model Within the **Arc input(s)** slot, the character / indicates that we generate two distinct graphs. The graph property **DISTANCE** measures the distance between two digraphs G_1 and G_2. This distance is defined as the sum of the following quantities:

- The number of arcs of G_1 that do not belong to G_2,
- The number of arcs of G_2 that do not belong to G_1.

Part (A) of Figure 5.311 gives the final graph associated with the sequence var-3,var-4,var-6,var-2,var-4 (i.e., the second argument of the constraint of the **Example** slot), while part (B) shows the final graph corresponding to var-2,var-6,var-9,var-3,var-6 (i.e., the third argument of the constraint of the **Example** slot). The two arc constraints that differ from one graph to the other are marked by a dotted line. The DISTANCE_BETWEEN constraint holds since between sequence var-3,var-4,var-6,var-2,var-4 and sequence var-2,var-6,var-9,var-3,var-6 there are DIST $= 2$ changes that respectively correspond to:

- Within the final graph associated with sequence var-3,var-4,var-6,var-2,var-4 the arc $4 \rightarrow 1$ (i.e., values $2 \rightarrow 3$) does not occur in the final graph associated with var-2,var-6,var-9,var-3,var-6,
- Within the final graph associated with sequence var-2,var-6,var-9,var-3,var-6 the arc $1 \rightarrow 4$ (i.e., values $2 \rightarrow 3$) does not occur in the final graph associated with var-3,var-4,var-6,var-2,var-4.

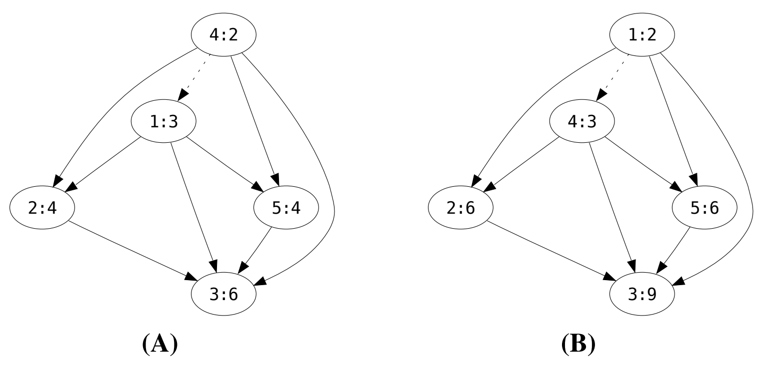


Figure 5.311: Final graphs of the DISTANCE_BETWEEN constraint

5.133 DISTANCE_CHANGE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from CHANGE.

Constraint DISTANCE_CHANGE(DIST, VARIABLES1, VARIABLES2, CTR)

Synonym DISTANCE.

Arguments

DIST	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
CTR	:	atom

Restrictions

DIST ≥ 0
DIST $<$ |VARIABLES1|
required(VARIABLES1, var)
required(VARIABLES2, var)
|VARIABLES1| $=$ |VARIABLES2|
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

DIST is equal to the number of times one of the following two conditions is true ($1 \leq i < n$):

- VARIABLES1$[i]$.var CTR VARIABLES1$[i+1]$.var holds and VARIABLES2$[i]$.var CTR VARIABLES2$[i+1]$.var does not hold,
- VARIABLES2$[i]$.var CTR VARIABLES2$[i+1]$.var holds and VARIABLES1$[i]$.var CTR VARIABLES1$[i+1]$.var does not hold.

Example

$(1, \langle 3, 3, 1, 2, 2\rangle, \langle 4, 4, 3, 3, 3\rangle, \neq)$

The DISTANCE_CHANGE constraint holds since the following condition (DIST $=$ 1) is verified: $\left\{\begin{array}{l}\texttt{VARIABLES1}[3].\texttt{var} = 1 \neq \texttt{VARIABLES1}[4].\texttt{var} = 2 \wedge \\ \texttt{VARIABLES2}[3].\texttt{var} = 3 = \texttt{VARIABLES1}[4].\texttt{var} = 3\end{array}\right.$.

Typical

DIST > 0
|VARIABLES1| > 1
CTR $\in [=, \neq]$

Symmetries

- Arguments are permutable w.r.t. permutation (DIST) (VARIABLES1, VARIABLES2) (CTR).
- One and the same constant can be added to the var attribute of all items of VARIABLES1.
- One and the same constant can be added to the var attribute of all items of VARIABLES2.

Arg. properties Functional dependency: DIST determined by VARIABLES1, VARIABLES2 and CTR.

Usage Measure the distance between two sequences according to the CHANGE constraint.

Remark We measure that distance with respect to a given constraint and not according to the fact that the variables are assigned distinct values.

See also **common keyword:** DISTANCE_BETWEEN *(proximity constraint)*.

root concept: CHANGE.

Keywords **characteristic of a constraint:** automaton, automaton with counters.

constraint arguments: pure functional dependency.

constraint network structure: sliding cyclic(2) constraint network(2).

constraint type: proximity constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES1/ VARIABLES2
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	**DISTANCE**= DIST

Graph model Within the **Arc input(s)** slot, the character / indicates that we generate two distinct graphs. The graph property DISTANCE measures the distance between two digraphs G_1 and G_2. This distance is defined as the sum of the following quantities:

- The number of arcs of G_1 that do not belong to G_2,
- The number of arcs of G_2 that do not belong to G_1.

Part (A) of Figure 5.312 gives the final graph associated with the sequence var-3,var-3,var-1,var-2,var-2 (i.e., the second argument of the constraint of the **Example** slot), while part (B) shows the final graph corresponding to var-4,var-4,var-3,var-3,var-3 (i.e., the third argument of the constraint of the **Example** slot). Since arc $3 \rightarrow 4$ belongs to the first final graph but not to the second one, the distance between the two final graphs is equal to 1.

(A) **(B)**

Figure 5.312: Final graphs of the DISTANCE_CHANGE constraint

Automaton Figure 5.313 depicts the automaton associated with the DISTANCE_CHANGE constraint. Let $(\mathtt{VAR1}_i, \mathtt{VAR1}_{i+1})$ and $(\mathtt{VAR2}_i, \mathtt{VAR2}_{i+1})$ respectively be the i^{th} pairs of consecutive variables of the collections VARIABLES1 and VARIABLES2. To each quadruple $(\mathtt{VAR1}_i, \mathtt{VAR1}_{i+1}, \mathtt{VAR2}_i, \mathtt{VAR2}_{i+1})$ corresponds a 0-1 signature variable S_i. The following signature constraint links these variables:

$((\mathtt{VAR1}_i = \mathtt{VAR1}_{i+1}) \wedge (\mathtt{VAR2}_i \neq \mathtt{VAR2}_{i+1})) \vee$

$((\mathtt{VAR1}_i \neq \mathtt{VAR1}_{i+1}) \wedge (\mathtt{VAR2}_i = \mathtt{VAR2}_{i+1})) \Leftrightarrow S_i$.

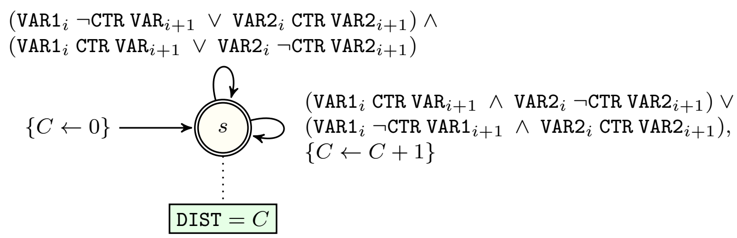


Figure 5.313: Automaton of the DISTANCE_CHANGE constraint

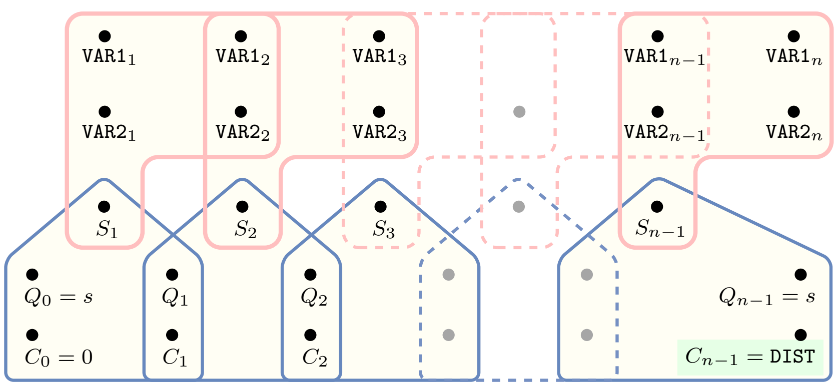


Figure 5.314: Hypergraph of the reformulation corresponding to the automaton of the DISTANCE_CHANGE constraint

5.134 DIVISIBLE

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint DIVISIBLE(Q, D)

Synonym DIV.

Arguments

Q : dvar
D : dvar

Restrictions

$Q \geq 0$
$D > 0$

Purpose Enforce the fact that the first variable Q is divisible by the second variable D.

Example $(12, 4)$

The DIVISIBLE constraint holds since 12 is divisible by 4.

Typical

$Q > 1$
$D < Q$

See also **implies:** DIVISIBLE_OR.

Keywords **constraint arguments:** binary constraint.
constraint type: predefined constraint, arithmetic constraint.
filtering: arc-consistency.

5.135 DIVISIBLE_OR

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint DIVISIBLE_OR(C, D)

Synonym DIV_OR.

Arguments

C	:	dvar
D	:	dvar

Restrictions

$\texttt{C} > 0$
$\texttt{D} > 0$

Purpose Enforce the fact that the first variable C is divisible by the second variable D, or that D is divisible by C.

Example $(4, 12)$

The DIVISIBLE_OR constraint holds since 12 is divisible by 4.

See also **implied by:** DIVISIBLE.

implies: SAME_SIGN, ZERO_OR_NOT_ZERO.

Keywords **constraint arguments:** binary constraint.

constraint type: predefined constraint, arithmetic constraint.

5.136 DOM_REACHABILITY

▶ ▷ ◁ ◀

DESCRIPTION LINKS

Origin [341]

Constraint

$$\text{DOM\_REACHABILITY}\left(\begin{array}{l}\texttt{SOURCE},\\ \texttt{FLOW\_GRAPH},\\ \texttt{DOMINATOR\_GRAPH},\\ \texttt{TRANSITIVE\_CLOSURE\_GRAPH}\end{array}\right)$$

Arguments

SOURCE	:	int
FLOW_GRAPH	:	collection(index−int, succ−svar)
DOMINATOR_GRAPH	:	collection(index−int, succ−sint)
TRANSITIVE_CLOSURE_GRAPH	:	collection(index−int, succ−svar)

Restrictions

SOURCE ≥ 1
SOURCE $\leq$ |FLOW_GRAPH|
required(FLOW_GRAPH, [index, succ])
FLOW_GRAPH.index ≥ 1
FLOW_GRAPH.index $\leq$ |FLOW_GRAPH|
FLOW_GRAPH.succ ≥ 1
FLOW_GRAPH.succ $\leq$ |FLOW_GRAPH|
distinct(FLOW_GRAPH, index)
required(DOMINATOR_GRAPH, [index, succ])
|DOMINATOR_GRAPH| = |FLOW_GRAPH|
DOMINATOR_GRAPH.index ≥ 1
DOMINATOR_GRAPH.index $\leq$ |DOMINATOR_GRAPH|
DOMINATOR_GRAPH.succ ≥ 1
DOMINATOR_GRAPH.succ $\leq$ |DOMINATOR_GRAPH|
distinct(DOMINATOR_GRAPH, index)
required(TRANSITIVE_CLOSURE_GRAPH, [index, succ])
|TRANSITIVE_CLOSURE_GRAPH| = |FLOW_GRAPH|
TRANSITIVE_CLOSURE_GRAPH.index ≥ 1
TRANSITIVE_CLOSURE_GRAPH.index $\leq$ |TRANSITIVE_CLOSURE_GRAPH|
TRANSITIVE_CLOSURE_GRAPH.succ ≥ 1
TRANSITIVE_CLOSURE_GRAPH.succ $\leq$ |TRANSITIVE_CLOSURE_GRAPH|
distinct(TRANSITIVE_CLOSURE_GRAPH, index)

Purpose

Let FLOW_GRAPH, DOMINATOR_GRAPH and TRANSITIVE_CLOSURE_GRAPH be three directed graphs respectively called the *flow graph*, the *dominance graph* and the *transitive closure graph* which all have the same vertices. In addition let SOURCE denote a vertex of the flow graph called the *source node* (not necessarily a vertex with no incoming arcs). The DOM_REACHABILITY constraint holds if and only if the flow graph (and its source node) verifies:

- The dominance relation expressed by the dominance graph (i.e., if there is an arc (i, j) in the dominance graph then, within the flow graph, all the paths from the source node to j contain i; note that when there is no path from the source node to j then any node dominates j).
- The transitive relation expressed by the transitive closure graph (i.e., if there is an arc (i, j) in the transitive closure graph then there is also a path from i to j in the flow graph).

Example

$$\left(\begin{array}{l} 1, \left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\{2\}, \\ \texttt{index}-2 & \texttt{succ}-\{3,4\}, \\ \texttt{index}-3 & \texttt{succ}-\emptyset, \\ \texttt{index}-4 & \texttt{succ}-\emptyset \end{array}\right\rangle, \\ \left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\{2,3,4\}, \\ \texttt{index}-2 & \texttt{succ}-\{3,4\}, \\ \texttt{index}-3 & \texttt{succ}-\emptyset, \\ \texttt{index}-4 & \texttt{succ}-\emptyset \end{array}\right\rangle, \\ \left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\{1,2,3,4\}, \\ \texttt{index}-2 & \texttt{succ}-\{2,3,4\}, \\ \texttt{index}-3 & \texttt{succ}-\{3\}, \\ \texttt{index}-4 & \texttt{succ}-\{4\} \end{array}\right\rangle \end{array}\right)$$

The flow graph, the dominance graph and the transitive closure graph corresponding to the second, third and fourth arguments of the DOM_REACHABILITY constraint are respectively depicted by parts (A), (B) and (C) of Figure 5.315. The DOM_REACHABILITY holds since the following conditions hold.

- The dominance relation expressed by the dominance graph is verified:
  - Since $(1, 2)$ belongs to the dominance graph all the paths from 1 to 2 in the flow graph pass through 1.
  - Since $(1, 3)$ belongs to the dominance graph all the paths from 1 to 3 in the flow graph pass through 1.
  - Since $(1, 4)$ belongs to the dominance graph all the paths from 1 to 4 in the flow graph pass through 1.
  - Since $(2, 3)$ belongs to the dominance graph all the paths from 1 to 3 in the flow graph pass through 2.
  - Since $(2, 4)$ belongs to the dominance graph all the paths from 1 to 4 in the flow graph pass through 2.
- The graph depicted by the fourth argument of the DOM_REACHABILITY constraint (i.e., TRANSITIVE_CLOSURE_GRAPH) is the transitive closure of the graph depicted by the second argument (i.e., FLOW_GRAPH).

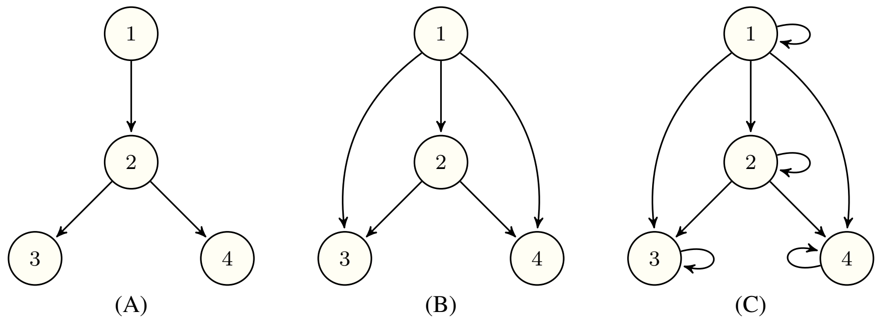


Figure 5.315: (A) Flow graph, (B) dominance graph and (C) transitive closure graph of the **Example** slot (taken from [339, page 40])

Typical $|\texttt{FLOW\_GRAPH}| > 2$

Symmetries

- Items of FLOW_GRAPH are permutable.
- Items of DOMINATOR_GRAPH are permutable.
- Items of TRANSITIVE_CLOSURE_GRAPH are permutable.

Usage The DOM_REACHABILITY constraint was introduced in order to solve reachability problems (e.g., disjoint paths, simple path with mandatory nodes).

Remark Within the name DOM_REACHABILITY, DOM stands for *domination*. In the context of path problems SOURCE refers to the start of the path we want to build.

Algorithm It was shown in [339] that, finding out wether a DOM_REACHABILITY constraint has a solution or not is NP-hard. This was achieved by reduction to *disjoint paths* problem [194].

The first implementation [340] of the DOM_REACHABILITY constraint was done in **Mozart** [130]. Later on, a second implemention [339] was done in **Gecode** [385]. Both implementations consist of the following two parts:

- Algorithms [373] for maintaining the lower bound of the transitive closure graph.
- Algorithms for maintaining the upper bound of the transitive closure graph, while respecting the dominance constraints [203].

See also **common keyword:** PATH, PATH_FROM_TO *(path)*.

Keywords **combinatorial object:** path.

constraint arguments: constraint involving set variables.

constraint type: predefined constraint, graph constraint.

5.137 DOMAIN

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Domain definition.

Constraint DOMAIN(VARIABLES, LOW, UP)

Synonym DOM.

Arguments

VARIABLES	:	collection(var−dvar)
LOW	:	int
UP	:	int

Restrictions

required(VARIABLES, var)
LOW $\leq$ UP

Purpose Enforce all the variables of the collection VARIABLES to take a value within the interval [LOW, UP].

Example $(\langle 2, 8, 2\rangle, 1, 9)$

The DOMAIN constraint holds since all the values 2, 8 and 2 of its first argument are greater than or equal to its second argument LOW $= 1$ and less than or equal to its third argument UP $= 9$.

Typical

$|\mathtt{VARIABLES}| > 1$
LOW $<$ UP

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any other value in [LOW, UP].
- LOW can be decreased.
- UP can be increased.
- One and the same constant can be added to the var attribute of all items of VARIABLES as well as to LOW and UP.

Arg. properties Contractible wrt. VARIABLES.

Remark The DOMAIN constraint is called DOM in **Gecode** (http://www.gecode.org/).

Reformulation The DOMAIN$(\langle \mathtt{var} - V_1, \mathtt{var} - V_2, \ldots, \mathtt{var} - V_{|\mathtt{VARIABLES}|}\rangle, \mathtt{LOW}, \mathtt{UP})$ constraint can be expressed in term of the conjunction

$V_1 \geq \mathtt{LOW} \wedge V_1 \leq \mathtt{UP}$,
$V_2 \geq \mathtt{LOW} \wedge V_2 \leq \mathtt{UP}$,
....
$V_{|\mathtt{VARIABLES}|} \geq \mathtt{LOW} \wedge V_{|\mathtt{VARIABLES}|} \leq \mathtt{UP}$.

Systems MEMBER in **Choco**, DOM in **Gecode**, DOMAIN in **SICStus**.

See also **common keyword:** IN, IN_INTERVAL *(domain definition)*.

uses in its reformulation: TREE_RANGE.

Keywords **constraint type:** predefined constraint, value constraint.

modelling: interval, domain definition.

5.138 DOMAIN_CONSTRAINT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [350]

Constraint DOMAIN_CONSTRAINT(VAR, VALUES)

Synonym DOMAIN.

Arguments

VAR	:	dvar
VALUES	:	collection(var01−dvar, value−int)

Restrictions

required(VALUES, [var01, value])
VALUES.var01 ≥ 0
VALUES.var01 ≤ 1
distinct(VALUES, value)

Purpose Make the link between a domain variable VAR and those 0-1 variables that are associated with each potential value of VAR: The 0-1 variable associated with the value that is taken by variable VAR is equal to 1, while the remaining 0-1 variables are all equal to 0.

Example

$$\left(5, \left\langle \begin{array}{ll} \texttt{var01}-0 & \texttt{value}-9, \\ \texttt{var01}-1 & \texttt{value}-5, \\ \texttt{var01}-0 & \texttt{value}-2, \\ \texttt{var01}-0 & \texttt{value}-7 \end{array} \right\rangle \right)$$

The DOMAIN_CONSTRAINT holds since VAR $= 5$ is set to the value corresponding to the 0-1 variable set to 1, while the other 0-1 variables are all set to 0.

Typical $|\texttt{VALUES}| > 1$

Symmetry Items of VALUES are permutable.

Usage This constraint is used in order to make the link between a formulation using finite domain constraints and a formulation exploiting 0-1 variables.

Reformulation The DOMAIN_CONSTRAINT(VAR,

$$\begin{array}{l} \langle \texttt{var01}-B_1\ \texttt{value}-v_1, \\ \ \texttt{var01}-B_2\ \texttt{value}-v_2, \\ \ \dots\dots\dots\dots\dots\dots \\ \ \texttt{var01}-B_{|\texttt{VALUES}|}\ \texttt{value}-v_{|\texttt{VALUES}|}\rangle) \end{array}$$

constraint can be expressed in term of the following reified constraint $(\texttt{VAR} = v_1 \wedge B_1 = 1) \vee (\texttt{VAR} = v_2 \wedge B_2 = 1) \vee \cdots \vee (\texttt{VAR} = v_{|\texttt{VALUES}|} \wedge B_{|\texttt{VALUES}|} = 1)$.

Systems DOMAINCHANNELING in **Choco**, CHANNEL in **Gecode**, IN in **SICStus**, IN_SET in **SICStus**.

See also

common keyword: LINK_SET_TO_BOOLEANS *(channelling constraint)*.

related: ROOTS.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, derived collection.

constraint network structure: centered cyclic(1) constraint network(1).

constraint type: decomposition.

filtering: linear programming, arc-consistency.

modelling: channelling constraint, domain channel, Boolean channel.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{VALUE}-\texttt{collection}(\texttt{var01}-\texttt{int},\texttt{value}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var01}-1,\texttt{value}-\texttt{VAR})]\end{array}\right)$$

Arc input(s)	VALUE VALUES
Arc generator	$PRODUCT \mapsto$ collection(value, values)
Arc arity	2
Arc constraint(s)	value.value = values.value $\Leftrightarrow$ values.var01 = 1
Graph property(ies)	**NARC**= |VALUES|

Graph model The DOMAIN_CONSTRAINT constraint is modelled with the following bipartite graph:

- The first class of vertices corresponds to a single vertex containing the domain variable.
- The second class of vertices contains one vertex for each item of the collection VALUES.

$PRODUCT$ is used in order to generate the arcs of the graph. In our context it takes a collection with a single item $\langle$var01 $-$ 1 value $-$ VAR$\rangle$ and the collection VALUES.

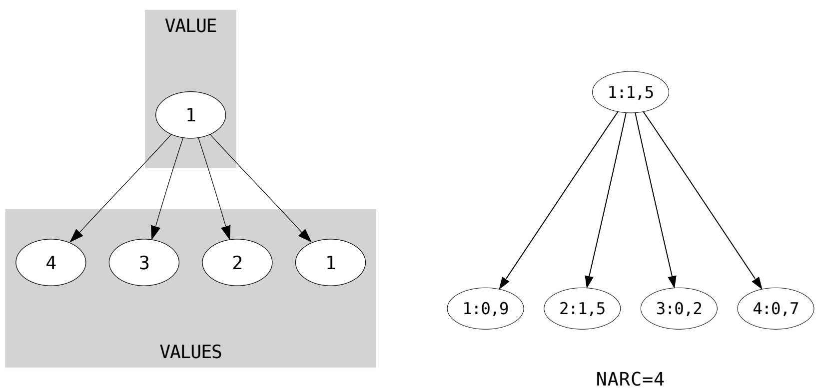
The arc constraint between the variable VAR and one potential value v expresses the following:

- If the 0-1 variable associated with v is equal to 1, VAR is equal to v.
- Otherwise, if the 0-1 variable associated with v is equal to 0, VAR is not equal to v.

Since all arc constraints should hold the final graph contains exactly |VALUES| arcs.

Parts (A) and (B) of Figure 5.316 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

Signature Since the number of arcs of the initial graph is equal to VALUES the maximum number of arcs of the final graph is also equal to VALUES. Therefore we can rewrite the graph property **NARC** $=$ |VALUES| to **NARC** $\geq$ |VALUES|. This leads to simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.



(A) **(B)**

Figure 5.316: Initial and final graph of the DOMAIN_CONSTRAINT constraint

Automaton Figure 5.317 depicts the automaton associated with the DOMAIN_CONSTRAINT constraint. Let $\texttt{VAR01}_i$ and $\texttt{VALUE}_i$ respectively be the `var01` and the `value` attributes of the i^{th} item of the `VALUES` collection. To each triple $(\texttt{VAR}, \texttt{VAR01}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((\texttt{VAR} = \texttt{VALUE}_i) \Leftrightarrow \texttt{VAR01}_i) \Leftrightarrow S_i$.

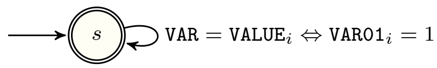


Figure 5.317: Automaton of the DOMAIN_CONSTRAINT constraint

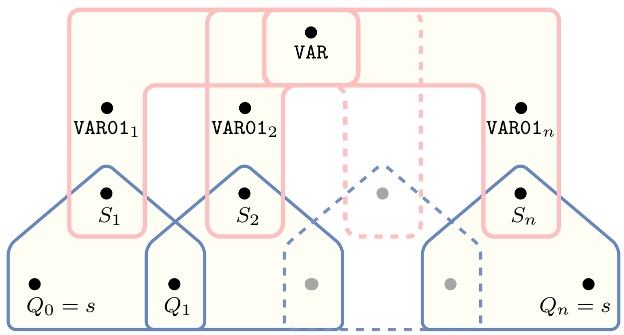


Figure 5.318: Hypergraph of the reformulation corresponding to the automaton of the DOMAIN_CONSTRAINT constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints involve only a single variable and the constraint network is Berge-acyclic

5.139 ELEM

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from ELEMENT.

Constraint ELEM(ITEM, TABLE)

Usual name ELEMENT

Synonyms NTH, ARRAY.

Arguments

ITEM	:	collection(index−dvar, value−dvar)
TABLE	:	collection(index−int, value−dvar)

Restrictions

required(ITEM, [index, value])
ITEM.index ≥ 1
ITEM.index $\leq |$TABLE$|$
$|$ITEM$| = 1$
$|$TABLE$| > 0$
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq |$TABLE$|$
distinct(TABLE, index)

Purpose ITEM is equal to one of the entries of the table TABLE.

Example

$$\left(\begin{array}{l}\langle \texttt{index}-3\ \texttt{value}-2\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\rangle\end{array}\right)$$

The ELEM constraint holds since its first argument ITEM corresponds to the third item of the TABLE collection.

Typical

$|$TABLE$| > 1$
range(TABLE.value) > 1

Symmetries

- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEM.value or TABLE.value can be swapped; all occurrences of a value in ITEM.value or TABLE.value can be renamed to any unused value.

Arg. properties Functional dependency: ITEM.value determined by ITEM.index and TABLE.

Usage

Makes the link between the discrete decision variable `INDEX` and the variable `VALUE` according to a given table of values `TABLE`. We now give five typical uses of the ELEM constraint.

1. In some problems we may have to *represent a function* $y = f(x)$ (with $x \in [1, m]$). In this context we generate the following ELEM constraint where `INDEX` is a domain variable taking its values in $\{1, 2, \ldots, m\}$:

$$\text{ELEM}\left(\begin{array}{l}\langle\ \texttt{index} - x \quad \texttt{value} - y\ \rangle,\\ \left\langle \begin{array}{ll} \texttt{index} - 1 & \texttt{value} - f(1), \\ \texttt{index} - 2 & \texttt{value} - f(2), \\ & \vdots \\ \texttt{index} - m & \texttt{value} - f(m) \end{array} \right\rangle \end{array}\right)$$

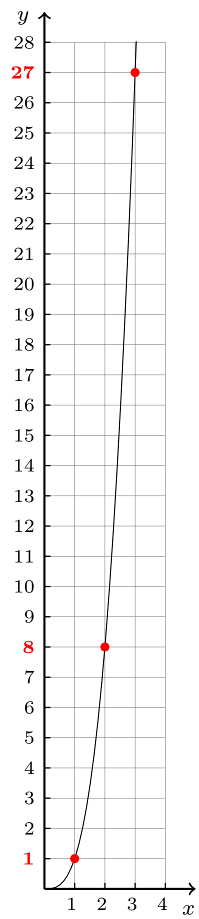


Figure 5.319: $y = x^3$ $(1 \leq x \leq 3)$

As an example, consider the problem of finding the smallest integer that can be decomposed in two different ways in the sum of two cubes [213]. The ELEM constraint can be used for representing the function $y = x^3$ (Figure 5.319). The unique solution $1729 = 12^3 + 1^3 = 10^3 + 9^3$ can be obtained by the following set of constraints:

$$\left\{\begin{array}{l}
\text{ELEM}(\langle \texttt{index} - x_1\ \texttt{value} - y_1\rangle,\\
\qquad \langle \texttt{index} - 1\ \texttt{value} - 1, \texttt{index} - 2\ \texttt{value} - 8, \ldots, \texttt{index} - 20\ \texttt{value} - 8000\rangle)\\
\text{ELEM}(\langle \texttt{index} - x_2\ \texttt{value} - y_2\rangle,\\
\qquad \langle \texttt{index} - 1\ \texttt{value} - 1, \texttt{index} - 2\ \texttt{value} - 8, \ldots, \texttt{index} - 20\ \texttt{value} - 8000\rangle)\\
\text{ELEM}(\langle \texttt{index} - x_3\ \texttt{value} - y_3\rangle,\\
\qquad \langle \texttt{index} - 1\ \texttt{value} - 1, \texttt{index} - 2\ \texttt{value} - 8, \ldots, \texttt{index} - 20\ \texttt{value} - 8000\rangle)\\
\text{ELEM}(\langle \texttt{index} - x_4\ \texttt{value} - y_4\rangle,\\
\qquad \langle \texttt{index} - 1\ \texttt{value} - 1, \texttt{index} - 2\ \texttt{value} - 8, \ldots, \texttt{index} - 20\ \texttt{value} - 8000\rangle)\\
y_1 + y_2 = y_3 + y_4\\
x_1 < x_2\\
x_3 < x_4\\
x_1 < x_3
\end{array}\right.$$

The last three inequalities constraints in the conjunction are used for breaking symmetries. The constraints $x_1 < x_2$ and $x_3 < x_4$ respectively order the pairs of variables (x_1, x_2) and (x_3, x_4) from which the sums $x_1^3 + x_2^3$ and $x_3^3 + x_4^3$ are generated. Finally the inequality $x_1 < x_3$ enforces a lexicographic ordering between the two pairs of variables (x_1, x_2) and (x_3, x_4).

2. In some optimisation problems a classical use of the ELEM constraint consists *expressing the link between a discrete choice and its corresponding cost*. For each discrete choice we create an ELEM constraint of the form:

$$\text{ELEM}\left(\begin{array}{l}
\langle\ \texttt{index} - \texttt{Choice}\quad \texttt{value} - \texttt{Cost}\ \rangle,\\
\left\langle\begin{array}{ll}
\texttt{index} - 1 & \texttt{value} - \texttt{Cost}_1,\\
\texttt{index} - 2 & \texttt{value} - \texttt{Cost}_2,\\
 & \vdots\\
\texttt{index} - m & \texttt{value} - \texttt{Cost}_m
\end{array}\right\rangle
\end{array}\right)$$

where:

- `Choice` is a domain variable that indicates which alternative will be finally selected,
- `Cost` is a domain variable that corresponds to the cost of the decision associated with the value of the `Choice` variable,
- $\texttt{Cost}_1, \texttt{Cost}_2, \ldots, \texttt{Cost}_m$ are the respective costs associated with the alternatives $1, 2, \ldots, m$.

3. In some problems we need to express a disjunction of the form $\texttt{VAR} = \texttt{VAR}_1 \vee \texttt{VAR} = \texttt{VAR}_2 \vee \cdots \vee \texttt{VAR} = \texttt{VAR}_n$. This can be directly reformulated as the following ELEM constraint, where `INDEX` is a domain variable taking its value in the finite set $\{1, 2, \ldots, n\}$ and where the `TABLE` argument corresponds to the domain variables $\texttt{VAR}_1, \texttt{VAR}_2, \ldots, \texttt{VAR}_n$:

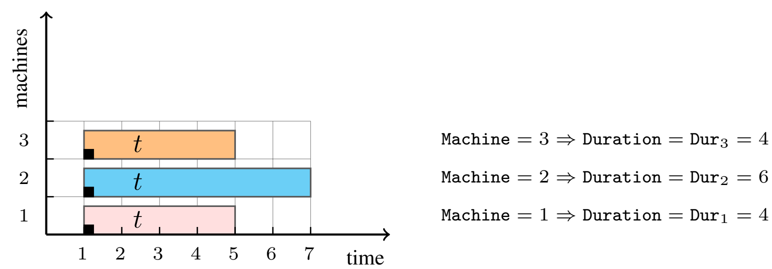
$$\text{ELEM}\left(\begin{array}{l}
\langle\ \texttt{index} - \texttt{INDEX}\quad \texttt{value} - \texttt{VAR}\ \rangle,\\
\left\langle\begin{array}{ll}
\texttt{index} - 1 & \texttt{value} - \texttt{VAR}_1,\\
\texttt{index} - 2 & \texttt{value} - \texttt{VAR}_2,\\
 & \vdots\\
\texttt{index} - n & \texttt{value} - \texttt{VAR}_n
\end{array}\right\rangle
\end{array}\right)$$

4. In some scheduling problems the duration of a task depends on the machine where the task will be assigned in final schedule. In this case we generate for each task an ELEM constraint of the following form:

$$\text{ELEM}\left(\begin{array}{l}\left\langle\ \texttt{index}-\texttt{Machine}\quad \texttt{value}-\texttt{Duration}\ \right\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-\texttt{Dur}_1,\\ \texttt{index}-2 & \texttt{value}-\texttt{Dur}_2,\\ & \vdots\\ \texttt{index}-m & \texttt{value}-\texttt{Dur}_m\end{array}\right\rangle\end{array}\right)$$

where:

- $\texttt{Machine}$ is a domain variable that indicates the resource to which the task will be assigned,
- $\texttt{Duration}$ is a domain variable that corresponds to the duration of the task,
- $\texttt{Dur}_1, \texttt{Dur}_2, \ldots, \texttt{Dur}_m$ are the respective duration of the task according to the hypothesis that it runs on machine $1, 2$ or m.



$$\text{ELEM}\left(\begin{array}{l}\langle\texttt{index}-\texttt{Machine}\ \texttt{value}-\texttt{Duration}\rangle,\\ \langle\texttt{index}-1\ \texttt{value}-4, \texttt{index}-2\ \texttt{value}-6, \texttt{index}-3\ \texttt{value}-4\rangle\end{array}\right)$$

Figure 5.320: A task t for which the duration depends on the machine 1, 2 or 3 to which it is assigned

Figure 5.320 illustrates this particular use of the ELEM constraint for modelling that a task has a duration of 4, 6 and 4 when we respectively assign it on machines 1, 2 and 3.

5. In some vehicle routing problems we typically use the ELEM constraint to express the distance between location i and the next location visited by a vehicle. For this purpose we generate for each location i an ELEM constraint of the form:

$$\text{ELEM}\left(\begin{array}{l}\left\langle\ \texttt{index}-\texttt{Next}_i\quad \texttt{value}-\texttt{distance}_i\ \right\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-\texttt{Dist}_{i_1},\\ \texttt{index}-2 & \texttt{value}-\texttt{Dist}_{i_2},\\ & \vdots\\ \texttt{index}-m & \texttt{value}-\texttt{Dist}_{i_m}\end{array}\right\rangle\end{array}\right)$$

where:

- $\texttt{Next}_i$ is a domain variable that gives the index of the location the vehicle will visit just after location i,

- $\texttt{distance}_i$ is a domain variable that corresponds to the distance between location i and the location the vehicle will visit just after,
- $\texttt{Dist}_{i_1}, \texttt{Dist}_{i_2}, \ldots, \texttt{Dist}_{i_m}$ are the respective distances between location i and locations $1, 2, \ldots, m$.

An other example where the table argument corresponds to domain variables is described in the keyword entry assignment to the same set of values.

Remark Originally, the parameters of the ELEM constraint had the form ELEMENT(INDEX, TABLE, VALUE), where INDEX and VALUE were two domain variables and TABLE was a list of non-negative integers.

Within some systems (e.g., **Gecode**), the index of the first entry of the table TABLE corresponds to 0 rather than to 1.

When the first entry of the table TABLE corresponds to a value p that is different from 1 we can still use the ELEM constraint. We use the reformulation $I = J - p + 1 \wedge$ ELEM($\langle$index $- I$ value $- V\rangle$, TABLE), where I and J are domain variables respectively ranging from 1 to $|\texttt{TABLE}|$ and from p to $p + |\texttt{TABLE}| - 1$.

Systems NTH in **Choco**, ELEMENT in **Gecode**, ELEMENT in **JaCoP**, ELEMENT in **SICStus**.

See also **common keyword:** ELEM_FROM_TO, ELEMENT_MATRIX, ELEMENT_PRODUCT, ELEMENT_SPARSE *(array constraint)*, ELEMENTS_SPARSE, STAGE_ELEMENT *(data constraint)*.

implied by: ELEMENT.

implies: ELEMENT *(single item replaced by two variables)*, ELEMENT_GREATEREQ, ELEMENT_LESSEQ, ELEMENTS.

system of constraints: ELEMENTS.

uses in its reformulation: ELEMENTS_ALLDIFFERENT.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: arc-consistency.

heuristics: labelling by increasing cost, regret based heuristics.

modelling: array constraint, table, functional dependency, variable indexing, variable subscript, disjunction, assignment to the same set of values, sequence dependent set-up.

modelling exercises: assignment to the same set of values, sequence dependent set-up, zebra puzzle.

puzzles: zebra puzzle.

Cond. implications ELEM(ITEM, TABLE)
with TABLE.value ≥ 0
implies BIN_PACKING_CAPA(TABLE, ITEM).

Arc input(s)	ITEM TABLE
Arc generator	*PRODUCT* $\mapsto$ collection(item, table)
Arc arity	2
Arc constraint(s)	• item.index = table.index • item.value = table.value
Graph property(ies)	**NARC** = 1

Graph model We regroup the INDEX and VALUE parameters of the original ELEMENT constraint ELEMENT(INDEX, TABLE, VALUE) into the parameter ITEM. We also make explicit the different indices of the table TABLE.

Parts (A) and (B) of Figure 5.321 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

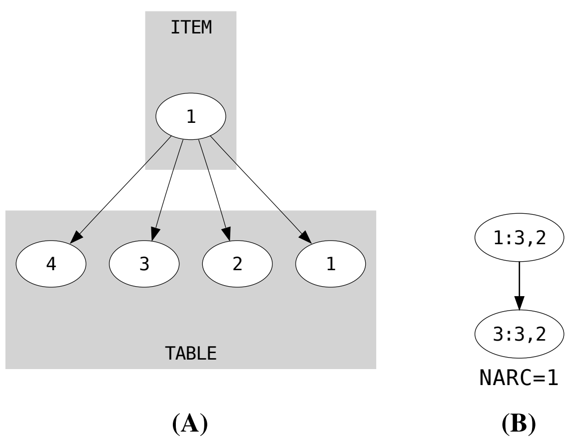


Figure 5.321: Initial and final graph of the ELEM constraint

Signature Since all the index attributes of TABLE are distinct and because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 5.322 depicts the automaton associated with the ELEM constraint. Let INDEX and VALUE respectively be the index and the value attributes of the unique item of the ITEM collection. Let $\mathtt{INDEX}_i$ and $\mathtt{VALUE}_i$ respectively be the index and the value attributes of item i of the TABLE collection. To each quadruple $(\mathtt{INDEX}, \mathtt{VALUE}, \mathtt{INDEX}_i, \mathtt{VALUE}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((\mathtt{INDEX} = \mathtt{INDEX}_i) \wedge (\mathtt{VALUE} = \mathtt{VALUE}_i)) \Leftrightarrow S_i$.

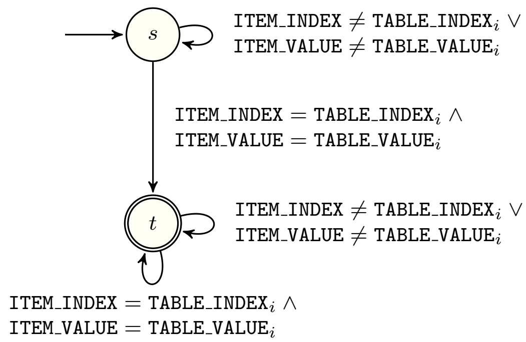


Figure 5.322: Automaton of the ELEM(ITEM, TABLE) constraint (once one finds the right item – index and value – in the table, one switches from the initial state s to the accepting state t)

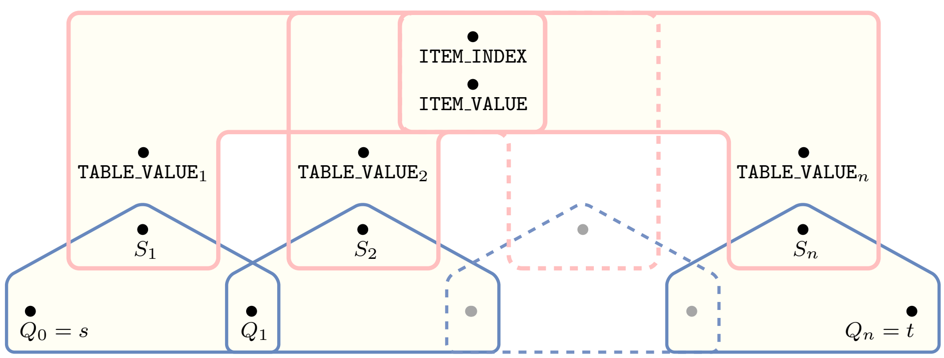


Figure 5.323: Hypergraph of the reformulation corresponding to the automaton of the ELEM constraint

5.140 ELEM_FROM_TO

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from ELEM.

Constraint ELEM_FROM_TO(ITEM, TABLE)

Synonym ELEMENT_FROM_TO.

Arguments

$$\begin{array}{lll} \texttt{ITEM} & : & \texttt{collection}\left(\begin{array}{l}\texttt{from}-\texttt{dvar},\\ \texttt{cst\_from}-\texttt{int},\\ \texttt{to}-\texttt{dvar},\\ \texttt{cst\_to}-\texttt{int},\\ \texttt{value}-\texttt{dvar}\end{array}\right)\\ \texttt{TABLE} & : & \texttt{collection}(\texttt{index}-\texttt{int},\texttt{value}-\texttt{dvar}) \end{array}$$

Restrictions

$\texttt{required}(\texttt{ITEM},[\texttt{from},\texttt{cst\_from},\texttt{to},\texttt{cst\_to},\texttt{value}])$
$\texttt{ITEM.from} \geq 1$
$\texttt{ITEM.from} \leq |\texttt{TABLE}|$
$\texttt{ITEM.to} \geq 1$
$\texttt{ITEM.to} \leq |\texttt{TABLE}|$
$\texttt{ITEM.from} \leq \texttt{ITEM.to}$
$|\texttt{ITEM}| = 1$
$\texttt{required}(\texttt{TABLE},[\texttt{index},\texttt{value}])$
$\texttt{TABLE.index} \geq 1$
$\texttt{TABLE.index} \leq |\texttt{TABLE}|$
$\texttt{increasing\_seq}(\texttt{TABLE},[\texttt{index}])$

Purpose

Let FROM, CST_FROM, TO, CST_TO, VALUE respectively denote the attributes ITEM[1].from, ITEM[1].cst_from, ITEM[1].to, ITEM[1].cst_to, ITEM[1].value of the unique item of the ITEM collection.

Beside imposing the fact that FROM $\leq$ TO and that both FROM and TO are assigned a value in $[1, |\texttt{TABLE}|]$, the ELEM_FROM_TO constraint forces the following condition: All entries of the TABLE collection from position $\max(1, \texttt{FROM} + \texttt{CST\_FROM})$ to position $\min(|\texttt{TABLE}|, \texttt{TO} + \texttt{CST\_TO})$ are equal to VALUE. When $\max(1, \texttt{FROM} + \texttt{CST\_FROM})$ is strictly greater than $\min(|\texttt{TABLE}|, \texttt{TO} + \texttt{CST\_TO})$ the constraint holds no matter what value is assigned to VALUE.

Example

$$\left(\begin{array}{l}\langle \texttt{from}-1\ \texttt{cst\_from}-1\ \texttt{to}-4\ \texttt{cst\_to}-{-1}\ \texttt{value}-2\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-2,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9,\\ \texttt{index}-5 & \texttt{value}-9\end{array}\right\rangle\end{array}\right)$$

The ELEM_FROM_TO constraint holds since all entries between position

$\max(1, \texttt{FROM} + \texttt{CST\_FROM}) = \max(1, 1+1) = 2$ and position $\min(|\texttt{TABLE}|, \texttt{TO} + \texttt{CST\_TO}) = \min(5, 4-1) = 3$ are equal to 2.

Typical

$\texttt{ITEM.cst\_from} \geq 0$
$\texttt{ITEM.cst\_from} \leq 1$
$\texttt{ITEM.cst\_to} \geq -1$
$\texttt{ITEM.cst\_to} \leq 1$
$|\texttt{TABLE}| > 1$
$\texttt{range}(\texttt{TABLE.value}) > 1$

Symmetry

All occurrences of two distinct values in ITEM.value or TABLE.value can be swapped; all occurrences of a value in ITEM.value or TABLE.value can be renamed to any unused value.

Usage

Given an array $t[1..n]$ of integers (i.e., an array of integers for which the entries are defined between 1 and n), the ELEM_FROM_TO constraint is useful, for example, for encoding expressions of the form $\exists i \in [1, n], \forall j \in [i+1, n] \mid t[i] = 0$. Note that, when the interval $[i+1, n]$ is empty, the condition $\forall j \in [i+1, n] \mid t[i] = 0$ is satisfied and i is equal to n. This example is encoded by using an ELEM_FROM_TO constraint and by respectively setting:

- FROM to i, where i is a variable that is assigned a value from interval $[1, n]$,
- CST_FROM to constant 1,
- TO to n, the index of the last entry of the array $t[1..n]$,
- CST_TO to constant 0,
- VALUE to 0, the value we are looking for.
- TABLE to the array of integers $t[1..n]$.

Finally, note that j is not used at all.

See also

common keyword: ELEM, ELEMENT *(array constraint)*.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint type: data constraint.

filtering: arc-consistency.

modelling: array constraint, table, variable indexing, variable subscript.

Automaton Figure 5.324 depicts the automaton associated with the ELEM_FROM_TO constraint. Let us first introduce some notations:

- Let n denote the number of items of the TABLE collection.
- Let $\texttt{INDEX}_i$ and $\texttt{VALUE}_i$ respectively be the index and the value attributes of the i^{th} item of the TABLE collection.
- Let FROM, CST_FROM, TO, CST_TO, VALUE respectively denote the attributes ITEM[1].from, ITEM[1].cst_from, ITEM[1].to, ITEM[1].cst_to, ITEM[1].value of the unique item of the ITEM collection.
- Let IN be a shortcut for condition $1 \leq \texttt{FROM} \wedge \texttt{FROM} \leq \texttt{TO} \wedge \texttt{TO} \leq n$.
- Let F and T respectively denote the quantities $\max(1, \texttt{FROM} + \texttt{CST\_FROM})$ and $\min(|\texttt{TABLE}|, \texttt{TO} + \texttt{CST\_TO})$.

To each septuple $(\texttt{FROM}, \texttt{TO}, \texttt{F}, \texttt{T}, \texttt{VALUE}, \texttt{INDEX}_i, \texttt{VALUE}_i)$ corresponds a signature variable S_i as well as the following signature constraint:

$$\begin{cases} (\texttt{IN} \wedge \texttt{F} > \texttt{T}) & \Leftrightarrow \; S_i = 0 \wedge \\ (\texttt{IN} \wedge \texttt{F} \leq \texttt{T} \wedge \texttt{F} > \texttt{INDEX}_i) & \Leftrightarrow \; S_i = 1 \wedge \\ (\texttt{IN} \wedge \texttt{F} \leq \texttt{T} \wedge \texttt{T} < \texttt{INDEX}_i) & \Leftrightarrow \; S_i = 2 \wedge \\ (\texttt{IN} \wedge \texttt{F} \leq \texttt{T} \wedge \texttt{F} \leq \texttt{INDEX}_i \wedge \texttt{INDEX}_i \leq \texttt{T} \wedge \texttt{VALUE} = \texttt{VALUE}_i) & \Leftrightarrow \; S_i = 3 \wedge \\ (\texttt{IN} \wedge \texttt{F} \leq \texttt{T} \wedge \texttt{F} \leq \texttt{INDEX}_i \wedge \texttt{INDEX}_i \leq \texttt{T} \wedge \texttt{VALUE} \neq \texttt{VALUE}_i) & \Leftrightarrow \; S_i = 4 \end{cases} .$$

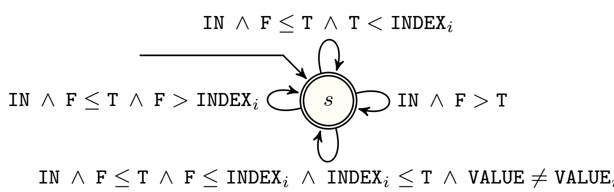


Figure 5.324: Automaton of the ELEM_FROM_TO constraint

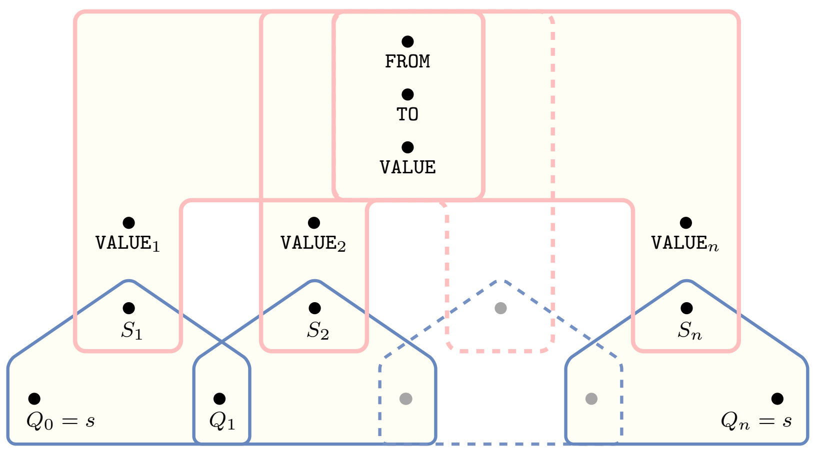


Figure 5.325: Hypergraph of the reformulation corresponding to the automaton of the ELEM_FROM_TO constraint

5.141 ELEMENT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [429]

Constraint ELEMENT(INDEX, TABLE, VALUE)

Synonyms NTH, ELEMENT_VAR, ARRAY.

Arguments

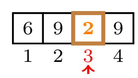
INDEX : dvar
TABLE : collection(value−dvar)
VALUE : dvar

Restrictions

$\texttt{INDEX} \geq 1$
$\texttt{INDEX} \leq |\texttt{TABLE}|$
$|\texttt{TABLE}| > 0$
required(TABLE, value)

Purpose VALUE is equal to the INDEXth item of TABLE, i.e. VALUE = TABLE[INDEX].

Example $(3, \langle 6, 9, 2, 9\rangle, 2)$



The ELEMENT constraint holds since its third argument VALUE = 2 is equal to the 3^{th} (INDEX = 3) item of the collection $\langle 6, 9, 2, 9\rangle$.

All solutions Figure 5.326 gives all solutions to the following non ground instance of the ELEMENT constraint: $\texttt{I} \in [3, 6]$, $\texttt{V} \in [1, 9]$, ELEMENT(I, $\langle 4, 8, 1, 0, 3, 3, 4, 3\rangle$, V).

① $(\mathbf{3}, \langle 4_1, 8_2, \mathbf{1_3}, 0_4, 3_4, 3_6, 7_7, 3_8\rangle, \mathbf{1})$
② $(\mathbf{5}, \langle 4_1, 8_2, 1_3, 0_4, \mathbf{3_5}, 3_6, 4_7, 3_8\rangle, \mathbf{3})$
③ $(\mathbf{6}, \langle 4_1, 8_2, 1_3, 0_4, 3_5, \mathbf{3_6}, 4_7, 3_8\rangle, \mathbf{3})$

Figure 5.326: All solutions corresponding to the non ground example of the ELEMENT constraint of the **All solutions** slot

Typical

$|\texttt{TABLE}| > 1$
range(TABLE.value) > 1

Symmetry All occurrences of two distinct values in TABLE.value or VALUE can be swapped; all occurrences of a value in TABLE.value or VALUE can be renamed to any unused value.

Arg. properties

- Functional dependency: VALUE determined by INDEX and TABLE.
- Suffix-extensible wrt. TABLE.

Usage

See **Usage** slot of ELEM.

Remark

In the original ELEMENT constraint of **CHIP** the index attribute was not explicitly present in the table of values. It was implicitly defined as the position of a value in the previous table.

Within some systems (e.g., **Gecode**), the index of the first entry of the table TABLE corresponds to 0 rather than to 1.

When the first entry of the table TABLE corresponds to a value p that is different from 1 we can still use the ELEMENT constraint. We use the reformulation $I = J - p + 1 \wedge$ ELEMENT$(I, \texttt{TABLE}, V)$, where I and J are domain variables respectively ranging from 1 to $|\texttt{TABLE}|$ and from p to $p + |\texttt{TABLE}| - 1$.

The ELEMENT constraint is called NTH in **Choco** (http://choco.sourceforge.net/). It is also sometimes called ELEMENT_VAR when the second argument corresponds to a table of variables.

The CASE constraint [108] is a generalisation of the ELEMENT constraint, where the table is replaced by a directed acyclic graph describing the set of solutions: there is a one to one correspondence between the solutions and the paths from the unique source of the dag to its leaves.

Systems

NTH in **Choco**, ELEMENT in **Gecode**, ELEMENT in **JaCoP**, ELEMENT in **MiniZinc**, ELEMENT in **SICStus**.

See also

common keyword: ELEM_FROM_TO, ELEMENT_GREATEREQ, ELEMENT_LESSEQ, ELEMENT_MATRIX, ELEMENT_PRODUCT, ELEMENT_SPARSE *(array constraint)*, ELEMENTN, ELEMENTS_SPARSE, IN_RELATION, STAGE_ELEMENT, SUM *(data constraint)*.

generalisation: COND_LEX_COST *(*variable *replaced by* tuple *of* variables*)*.

implied by: ELEM.

implies: ELEM.

related: TWIN *((pairs linked by an element with the same table))*.

system of constraints: ELEMENTS.

uses in its reformulation: CYCLE, ELEMENTS_ALLDIFFERENT, SORT_PERMUTATION, TREE_RANGE, TREE_RESOURCE.

Keywords

characteristic of a constraint: core, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: pure functional dependency.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: arc-consistency.

heuristics: labelling by increasing cost, regret based heuristics.

modelling: array constraint, table, functional dependency, variable indexing, variable subscript, disjunction, assignment to the same set of values, sequence dependent set-up.

modelling exercises: assignment to the same set of values, sequence dependent set-up, zebra puzzle.

puzzles: zebra puzzle.

Derived Collection

$\texttt{col}\left(\begin{array}{l}\texttt{ITEM}-\texttt{collection}(\texttt{index}-\texttt{dvar},\texttt{value}-\texttt{dvar}),\\ [\texttt{item}(\texttt{index}-\texttt{INDEX},\texttt{value}-\texttt{VALUE})]\end{array}\right)$

Arc input(s): ITEM TABLE

Arc generator: $PRODUCT \mapsto$ collection(item, table)

Arc arity: 2

Arc constraint(s):
- item.index = table.key
- item.value = table.value

Graph property(ies): **NARC** $= 1$

Graph model

The original ELEMENT constraint with three arguments. We use the derived collection ITEM for putting together the INDEX and VALUE parameters of the ELEMENT constraint. Within the arc constraint we use the implicit attribute key that associates to each item of a collection its position within the collection.

Parts (A) and (B) of Figure 5.327 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

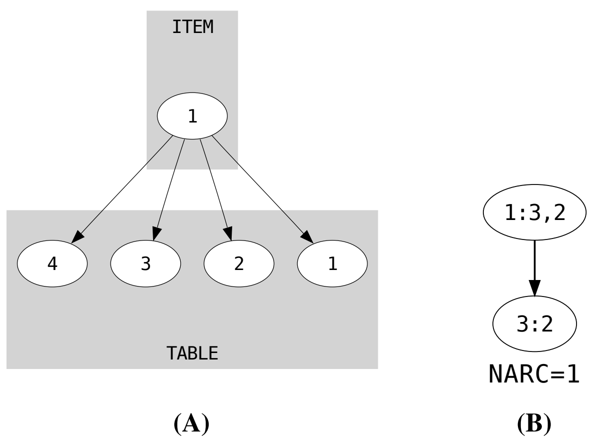


Figure 5.327: Initial and final graph of the ELEMENT constraint

Signature

Because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite **NARC** $= 1$ to **NARC** ≥ 1 and simplify $\overline{\mathbf{NARC}}$ to $\underline{\mathbf{NARC}}$.

Automaton Figure 5.328 depicts the automaton associated with the ELEMENT constraint. Let $\texttt{VALUE}_i$ be the `value` attribute of item i of the `TABLE` collection. To each triple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $(\texttt{INDEX} = i \wedge \texttt{VALUE} = \texttt{VALUE}_i) \Leftrightarrow S_i$.

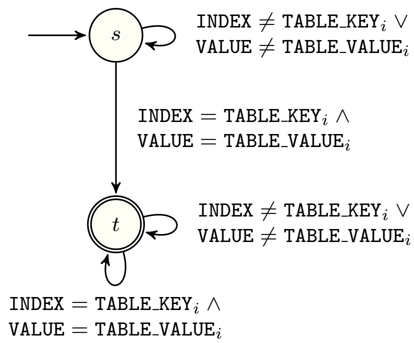


Figure 5.328: Automaton of the ELEMENT(INDEX, TABLE, VALUE) constraint (once one finds the right index and value in the table, one switches from the initial state s to the accepting state t)

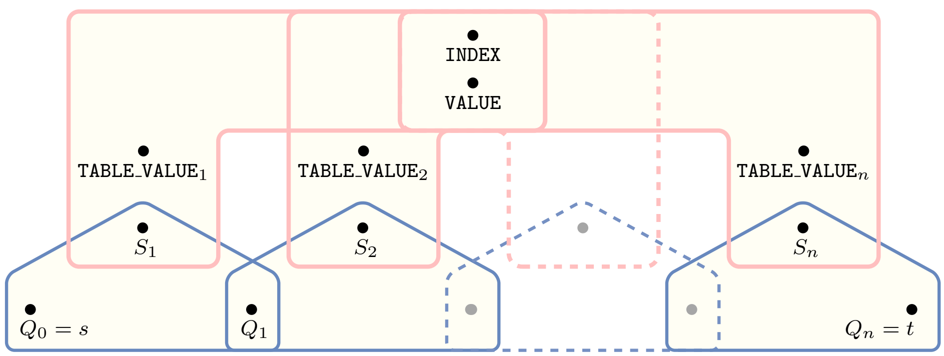


Figure 5.329: Hypergraph of the reformulation corresponding to the automaton of the ELEMENT constraint

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint $\text{ELEMENT}(0, \langle 5, 1, 4, 8, 1\rangle, 5)$ hold?

B. Does the constraint $\text{ELEMENT}(3, \langle 8, 2, 4, 3\rangle, 4)$ hold?

C. Does the constraint $\text{ELEMENT}(5, \langle 0, 1, 2, 3, 4, 5\rangle, 5)$ hold?

[a] Hint: go back to the definition of ELEMENT.

EXERCISE 2 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\begin{cases} I \in [2,6], \\ V \in [0,5], \\ \text{ELEMENT}(I, \langle 0,2,9,5,9,2,3,9\rangle, V). \end{cases}$$

[a]Hint: follow the order induced by the functional dependency between the arguments of ELEMENT, enumerate solutions in lexicographic order.

EXERCISE 3 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\begin{cases} I \in [2,3], \\ V_1 \in [5,5], V_2 \in [3,5], V_3 \in [0,3], \\ V \in [1,2], \\ \text{ELEMENT}(I, \langle V_1, V_2, V_3\rangle, V). \end{cases}$$

[a]Hint: first find the feasible values of the first argument, then enumerate solutions in lexicographic order.

EXERCISE 4 (identifying infeasible values)[a]

Identify all variable-value pairs (V_i, val) $(0 \leq i \leq 3)$, such that the following constraint has no solution when value val is assigned to variable V_i:

$$\begin{cases} V_0 \in [2,3],\ \ V_1 \in [2,4], \\ V_2 \in [0,4],\ \ V_3 \in [3,5], \\ \text{ELEMENT}(V_0, \langle V_1, 0, V_2, 6\rangle, V_3). \end{cases}$$

[a]Hint: first find the feasible values of the first argument, then filter the other variables.

EXERCISE 5 (variable-based degree of violation)[a]

Compute the variable-based degree of violation[b] of the following constraints:

A. ELEMENT$(0, \langle 2,2,2,2\rangle, 2)$,

B. ELEMENT$(3, \langle 3,1,5,2,7\rangle, 4)$,

C. ELEMENT$(8, \langle 5,5,8,5,5,0,7\rangle, 2)$.

[a]Hint: take advantage of the functional dependency.

[b]Given a constraint for which all variables are fixed, the *variable-based degree of violation* is the minimum number of variables to assign differently in order to satisfy the constraint.

EXERCISE 6 (using entailment for counting)[a]

A. Given an ELEMENT$(i, \langle t_1, t_2, \ldots, t_n\rangle, v)$ constraint where i, $t_1, t_2, \ldots, t_n$, v are variables, what is the minimum number of variables to fix in order to achieve entailment.[b] We assume that the constraint has at least one solution.

B. Exploit entailment in order to compute the number of solutions to the constraint $i \in [1, 3]$, $v_1 \in [0, 1]$, $v_2 \in [1, 9]$, $v_3 \in [3, 5]$, $v \in [2, 7]$, ELEMENT$(i, \langle v_1, v_2, v_3\rangle, v)$.

[a]Hint: take advantage of the functional dependency, use a case analysis on the first argument.

[b]A constraint is *entailed* if and only if it is for sure satisfied even though some of its variables are not fixed.

EXERCISE 7 (modelling with an unconstrained index)[a]

What does the ELEMENT constraint model when its first argument, the index, is unconstrained?

[a]Hint: how would one define the set of solutions of the third argument?

EXERCISE 8 (modelling an index starting at 0)[a]

Given a table t whose entries are indexed at $[0, n]$ model the requirement $v = t[i]$.

[a]Hint: make a shift.

EXERCISE 9 (modelling indirection)[a]

Given a table t whose entries vary between 1 and 9, model the requirement $v = t[t[i]]$ as one or several constraints. What is the implicit assumption we have on the entries of the table?

[a]Hint: use more than one constraint.

SOLUTION TO EXERCISE 1

A. *No, since value the first argument starts at index* 1.

B. *Yes, since the third entry of the table is equal to* 4.

C. *No, since the fifth entry is equal to* 4 *(and not to* 5*).*

SOLUTION TO EXERCISE 2

the three solutions

$I,\quad \langle 0,2,9,5,9,2,3,9\rangle,\quad V$

① $(\mathbf{2}, \langle 0_1, \mathbf{2_2}, 9_3, 5_4, 9_5, 2_6, 3_7, 9_8\rangle, \mathbf{2})$

② $(\mathbf{4}, \langle 0_1, 2_2, 9_3, \mathbf{5_4}, 9_5, 2_6, 3_7, 9_8\rangle, \mathbf{5})$

③ $(\mathbf{6}, \langle 0_1, 2_2, 9_3, 5_4, 9_5, \mathbf{2_6}, 3_7, 9_8\rangle, \mathbf{2})$

1. *The active entries of the table are located between index* 2 *and* 6, *as shown in bold by* $\langle 0_1, \mathbf{2_2}, \mathbf{9_3}, \mathbf{5_4}, \mathbf{9_5}, \mathbf{2_6}, 3_7, 9_8\rangle$.
2. *Among these entries we restrict ourselves to those entries for which the value is located in the domain of variable* V, *i.e. in interval* $[0, 5]$. *The remaining entries are shown in bold, i.e.* $\langle 0_1, \mathbf{2_2}, 9_3, \mathbf{5_4}, 9_5, \mathbf{2_6}, 3_7, 9_8\rangle$.
3. *This leads to the three solutions* $I = \mathbf{2}$ $V = \mathbf{2}$, $I = \mathbf{4}$ $V = \mathbf{5}$ *and* $I = \mathbf{6}$ $V = \mathbf{2}$.

SOLUTION TO EXERCISE 3

the six solutions

$I,\ \langle V_1, V_2, V_3\rangle,\ \ V$

① $(\mathbf{3}, \langle 5_1, 3_2, \mathbf{1_3}\rangle, \mathbf{1})$

② $(\mathbf{3}, \langle 5_1, 3_2, \mathbf{2_3}\rangle, \mathbf{2})$

③ $(\mathbf{3}, \langle 5_1, 4_2, \mathbf{1_3}\rangle, \mathbf{1})$

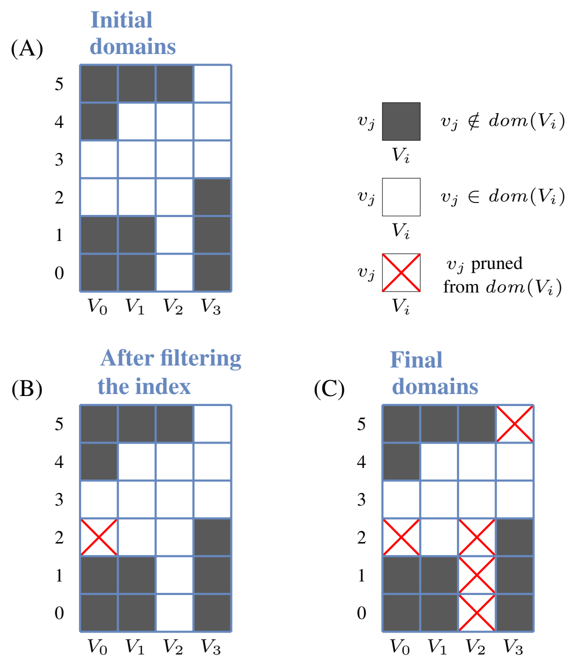
④ $(\mathbf{3}, \langle 5_1, 4_2, \mathbf{2_3}\rangle, \mathbf{2})$

⑤ $(\mathbf{3}, \langle 5_1, 5_2, \mathbf{1_3}\rangle, \mathbf{1})$

⑥ $(\mathbf{3}, \langle 5_1, 5_2, \mathbf{2_3}\rangle, \mathbf{2})$

1. *Since the domain of* V_2 *which is located at the second entry of the table does not intersect the domain of* V *(the third argument), the index variable* I *(the first argument) can not be assigned value* 2, *and is therefore fixed to* 3.
2. *Since* I *is fixed to* 3 *we have that* $V = V_3$. *Consequently* V *and* V_3 *are assigned a same value that belongs to the intersection of their respective domains, i.e.* $[0, 3] \cap [1, 2] = [1, 2]$.

SOLUTION TO EXERCISE 4



1. *In part* (A) *we give the initial domains of the index variable* (V_0), *of the first and third entries of the table* (V_1, V_2), *and of the third argument of the* ELEMENT *constraint* (V_3).
2. *In part* (B) *we prune the index variable* V_0. *On the one hand, it can not be assigned value* 2 *since the second entry of the table is set to* 0, *and* 0 *does not belong to the domain of* V_3, *see* $\times$. *On the other hand, it can be assigned value* 3 *since* $dom(V_2) \cap dom(V_3) \neq \emptyset$.
3. *Finally in part* (C) *we remove those values that contradict the fact that* $V_2 = V_3$, *see* $\times$.

SOLUTION TO EXERCISE 5

A. *The degree of violation is equal to* 1 *since we only need to change the index from* 0 *(because* 0 *is not an allowed value for the index) to any integer value in* $[1, 4]$.

$$\textsc{element}(\overset{1}{\mathbf{0}}, \langle 2, 2, 2, 2\rangle, 2)$$

B. *The degree of violation is equal to* 1 *since we only need to change the third entry of the table to* 4 *(or to switch the third argument from* 4 *to* 5*).*

$$\textsc{element}(3, \langle 3, 1, \overset{4}{\mathbf{5}}, 2, 7\rangle, 4)$$

C. *The degree of violation is equal to* 2 *since we need to change both the index (the table has only* 7 *entries) and the third argument (value* 2 *does not occur in the table). Rather than changing the third argument, we may change an entry of the table (e.g., if we set the index to* 3 *we set the third entry of the table to* 2*).*

$$\textsc{element}(\overset{1}{\mathbf{8}}, \langle 5, 5, 8, 5, 5, 0, 7\rangle, \overset{5}{\mathbf{2}})$$

SOLUTION TO EXERCISE 6

A. We need to fix 3 variables in the following way:

- **(i)** The first argument, the index i, is fixed to a value α $(1 \leq \alpha \leq n)$ such that $dom(t_\alpha) \cap dom(v) \neq \emptyset$.
- **(ii)** We fix the third argument v to a value β in $dom(t_\alpha) \cap dom(v)$.
- **(iii)** We fix t_α to β.

B. We have 90 solutions depending on whether $i = 1$, $i = 2$, $i = 3$ (and $v = v_i$):

- **(i)** $|dom(v) \cap dom(v_1)| \cdot |dom(v_2)| \cdot |dom(v_3)| = 0 \cdot 9 \cdot 3 = 0$,
- **(ii)** $|dom(v) \cap dom(v_2)| \cdot |dom(v_1)| \cdot |dom(v_3)| = 6 \cdot 2 \cdot 3 = 36$,
- **(iii)** $|dom(v) \cap dom(v_3)| \cdot |dom(v_1)| \cdot |dom(v_2)| = 3 \cdot 2 \cdot 9 = 54$.

SOLUTION TO EXERCISE 7

Given a table t *of* n *entries* $t[1], t[2], \ldots, t[n]$, ELEMENT *models a disjunction stating that the third argument* v *is equal to one of the values that can be assigned to one of the variables of the table, i.e.* $v = t[1] \vee v = t[2] \vee \cdots \vee v = t[n]$.

SOLUTION TO EXERCISE 8

The requirement $v = t[i]$ *can be modelled as the conjunction of the two constraints* $j = i + 1$, ELEMENT$(j, \langle t[0], t[1], \ldots, t[n]\rangle, v)$.

SOLUTION TO EXERCISE 9

The requirement $v = t[t[i]]$ *can be modelled as the conjunction of two* ELEMENT *constraints sharing the same table, namely:*

ELEMENT$(i, \langle t[1], t[2], \ldots, t[9]\rangle, j)$ $\leftarrow$ *inner indirection* $t[t[i]]$

ELEMENT$(j, \langle t[1], t[2], \ldots, t[9]\rangle, v)$ $\leftarrow$ *outer indirection* $t[t[i]]$

The second ELEMENT *constraint assumes that* j *corresponds to a valid index of the table, i.e. a value between* 1 *and* 9.

5.142 ELEMENT_GREATEREQ

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [312]

Constraint ELEMENT_GREATEREQ(ITEM, TABLE)

Arguments

ITEM : collection(index−dvar, value−dvar)
TABLE : collection(index−int, value−int)

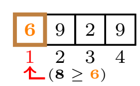
Restrictions

required(ITEM, [index, value])
ITEM.index ≥ 1
ITEM.index $\leq$ |TABLE|
|ITEM| $= 1$
|TABLE| > 0
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq$ |TABLE|
distinct(TABLE, index)

Purpose ITEM[1].value is greater than or equal to one of the entries (i.e., the value attribute) of the table TABLE.

Example

$$\left(\begin{array}{l}\langle \texttt{index}-1\ \texttt{value}-8\rangle, \\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6, \\ \texttt{index}-2 & \texttt{value}-9, \\ \texttt{index}-3 & \texttt{value}-2, \\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\rangle\end{array}\right)$$



The ELEMENT_GREATEREQ constraint holds since ITEM[1].value $= 8$ is greater than or equal to TABLE[ITEM[1].index].value $=$ TABLE[1].value $= 6$.

Typical

|TABLE| > 1
range(TABLE.value) > 1

Symmetries

- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEM.value or TABLE.value can be swapped; all occurrences of a value in ITEM.value or TABLE.value can be renamed to any unused value.

Usage Used for modelling variable subscripts in linear constraints [312].

Reformulation By introducing an extra variable VAL, the ELEMENT_GREATEREQ(⟨index − INDEX value − VALUE⟩, TABLE) constraint can be expressed in term of an ELEM(⟨index − INDEX value − VAL⟩, TABLE) constraint and of an inequality constraint VALUE $\geq$ VAL.

See also

common keyword: ELEMENT, ELEMENT_LESSEQ, ELEMENT_PRODUCT *(array constraint)*.

implied by: ELEM.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint arguments: binary constraint.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: linear programming, arc-consistency.

modelling: array constraint, table, variable subscript, variable indexing.

Arc input(s)	ITEM TABLE
Arc generator	$PRODUCT \mapsto$ collection(item, table)
Arc arity	2
Arc constraint(s)	• item.index = table.index • item.value $\geq$ table.value
Graph property(ies)	**NARC** = 1

Graph model Similar to the ELEMENT constraint except that the *equality* constraint of the second condition of the arc constraint is replaced by a *greater than or equal to* constraint.

Parts (A) and (B) of Figure 5.330 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

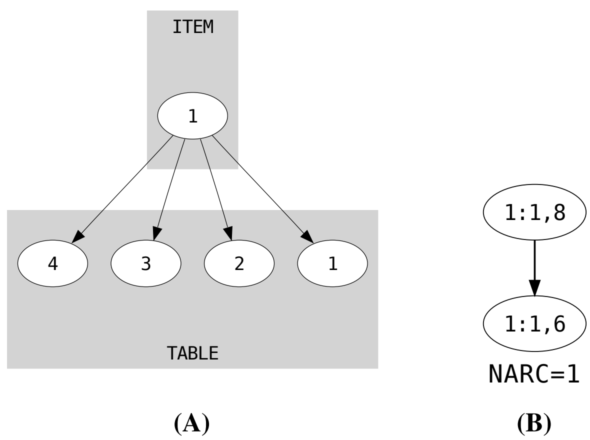


Figure 5.330: Initial and final graph of the ELEMENT_GREATEREQ constraint

Signature Since all the index attributes of TABLE are distinct and because of the first arc constraint the final graph cannot have more than one arc. Therefore we can rewrite **NARC** = 1 to **NARC** ≥ 1 and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 5.331 depicts the automaton associated with the ELEMENT_GREATEREQ constraint. Let `INDEX` and `VALUE` respectively be the `index` and the `value` attributes of the unique item of the `ITEM` collection. Let $\mathtt{INDEX}_i$ and $\mathtt{VALUE}_i$ respectively be the `index` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quadruple $(\mathtt{INDEX}, \mathtt{VALUE}, \mathtt{INDEX}_i, \mathtt{VALUE}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((\mathtt{INDEX} = \mathtt{INDEX}_i) \wedge (\mathtt{VALUE} \geq \mathtt{VALUE}_i)) \Leftrightarrow S_i$.

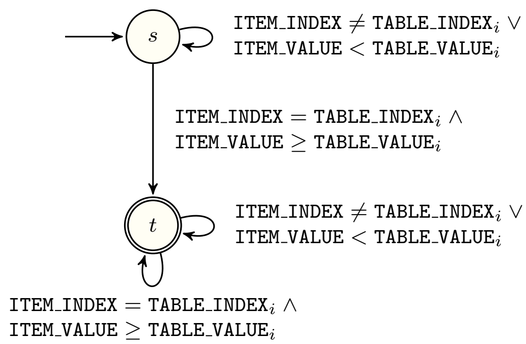


Figure 5.331: Automaton of the ELEMENT_GREATEREQ constraint

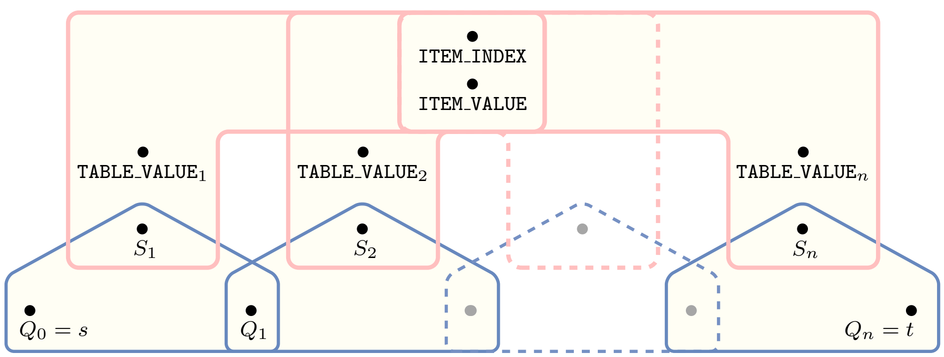


Figure 5.332: Hypergraph of the reformulation corresponding to the automaton of the ELEMENT_GREATEREQ constraint

5.143 ELEMENT_LESSEQ

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [312]

Constraint ELEMENT_LESSEQ(ITEM, TABLE)

Arguments

ITEM : collection(index−dvar, value−dvar)
TABLE : collection(index−int, value−int)

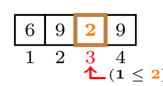
Restrictions

required(ITEM, [index, value])
ITEM.index ≥ 1
ITEM.index $\leq$ |TABLE|
|ITEM| $= 1$
|TABLE| > 0
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq$ |TABLE|
distinct(TABLE, index)

Purpose ITEM[1].value is less than or equal to one of the entries (i.e., the value attribute) of the table TABLE.

Example

$$\left(\langle \texttt{index}-3\ \texttt{value}-1 \rangle, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{value}-6, \\ \texttt{index}-2 & \texttt{value}-9, \\ \texttt{index}-3 & \texttt{value}-2, \\ \texttt{index}-4 & \texttt{value}-9 \end{array} \right\rangle \right)$$



The ELEMENT_LESSEQ constraint holds since ITEM[1].value $=$ 1 is less than or equal to TABLE[ITEM[1].index].value $=$ TABLE[3].value $= 2$.

Typical

|TABLE| > 1
range(TABLE.value) > 1

Symmetries

- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEM.value or TABLE.value can be swapped; all occurrences of a value in ITEM.value or TABLE.value can be renamed to any unused value.

Usage Used for modelling variable subscripts in linear constraints [312].

Reformulation By introducing an extra variable VAL, the ELEMENT_LESSEQ(⟨index − INDEX value − VALUE⟩, TABLE) constraint can be expressed in term of an ELEM(⟨index − INDEX value − VAL⟩, TABLE) constraint and of an inequality constraint VALUE $\leq$ VAL.

See also **common keyword:** ELEMENT, ELEMENT_GREATEREQ, ELEMENT_PRODUCT *(array constraint)*.

implied by: ELEM.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint arguments: binary constraint.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: linear programming, arc-consistency.

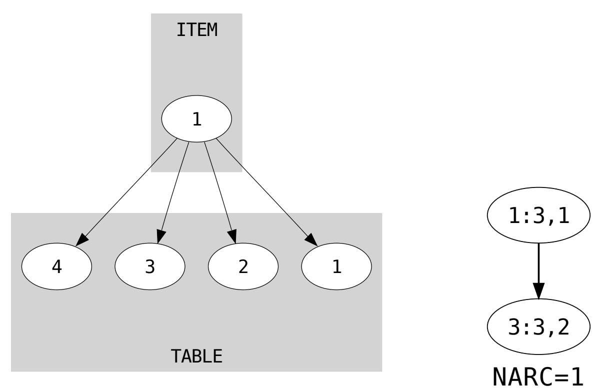
modelling: array constraint, table, variable subscript, variable indexing.

Cond. implications ELEMENT_LESSEQ$(\texttt{ITEM}, \texttt{TABLE})$
with $\texttt{minval}(\texttt{ITEM.value}) > 0$
and $\texttt{TABLE.value} > 0$
implies BIN_PACKING_CAPA$(\texttt{BINS} : \texttt{TABLE}, \texttt{ITEMS} : \texttt{ITEM})$.

Arc input(s)	ITEM TABLE
Arc generator	$PRODUCT \mapsto$ collection(item, table)
Arc arity	2
Arc constraint(s)	• item.index = table.index • item.value $\leq$ table.value
Graph property(ies)	**NARC** $= 1$

Graph model Similar to the ELEMENT constraint except that the *equality* constraint of the second condition of the arc constraint is replaced by a *less than or equal to* constraint.

Parts (A) and (B) of Figure 5.333 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.



(A) **(B)**

Figure 5.333: Initial and final graph of the ELEMENT_LESSEQ constraint

Signature Since all the index attributes of TABLE are distinct and because of the first arc constraint the final graph cannot have more than one arc. Therefore we can rewrite $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 5.334 depicts the automaton associated with the ELEMENT_LESSEQ constraint. Let INDEX and VALUE respectively be the index and the value attributes of the unique item of the ITEM collection. Let $\texttt{INDEX}_i$ and $\texttt{VALUE}_i$ respectively be the index and the value attributes of the i^{th} item of the TABLE collection. To each quadruple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{INDEX}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((\texttt{INDEX} = \texttt{INDEX}_i) \wedge (\texttt{VALUE} \leq \texttt{VALUE}_i)) \Leftrightarrow S_i$.

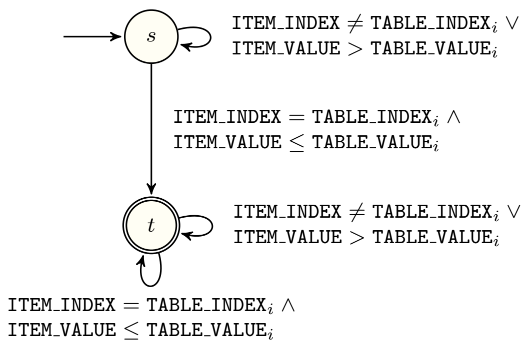


Figure 5.334: Automaton of the ELEMENT_LESSEQ constraint

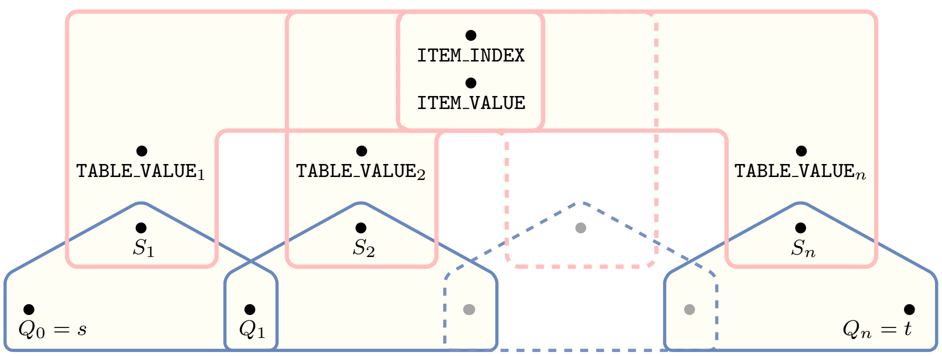


Figure 5.335: Hypergraph of the reformulation corresponding to the automaton of the ELEMENT_LESSEQ constraint

5.144 ELEMENT_MATRIX

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint ELEMENT_MATRIX(MAX_I, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE)

Synonyms ELEM_MATRIX, MATRIX.

Arguments

MAX_I	:	int
MAX_J	:	int
INDEX_I	:	dvar
INDEX_J	:	dvar
MATRIX	:	collection(i−int, j−int, v−int)
VALUE	:	dvar

Restrictions

MAX_I ≥ 1
MAX_J ≥ 1
INDEX_I ≥ 1
INDEX_I $\leq$ MAX_I
INDEX_J ≥ 1
INDEX_J $\leq$ MAX_J
required(MATRIX, [i, j, v])
increasing_seq(MATRIX, [i, j])
MATRIX.i ≥ 1
MATRIX.i $\leq$ MAX_I
MATRIX.j ≥ 1
MATRIX.j $\leq$ MAX_J
|MATRIX| = MAX_I $*$ MAX_J

Purpose

The MATRIX collection corresponds to the two-dimensional matrix MATRIX[1..MAX_I, 1..MAX_J]. VALUE is equal to the entry MATRIX[INDEX_I, INDEX_J] of the previous matrix.

Example

$$\left(4,3,1,3,\left\langle\begin{array}{lll} \mathtt{i}-1 & \mathtt{j}-1 & \mathtt{v}-4, \\ \mathtt{i}-1 & \mathtt{j}-2 & \mathtt{v}-1, \\ \mathtt{i}-1 & \mathtt{j}-3 & \mathtt{v}-7, \\ \mathtt{i}-2 & \mathtt{j}-1 & \mathtt{v}-1, \\ \mathtt{i}-2 & \mathtt{j}-2 & \mathtt{v}-0, \\ \mathtt{i}-2 & \mathtt{j}-3 & \mathtt{v}-8, \\ \mathtt{i}-3 & \mathtt{j}-1 & \mathtt{v}-3, \\ \mathtt{i}-3 & \mathtt{j}-2 & \mathtt{v}-2, \\ \mathtt{i}-3 & \mathtt{j}-3 & \mathtt{v}-1, \\ \mathtt{i}-4 & \mathtt{j}-1 & \mathtt{v}-0, \\ \mathtt{i}-4 & \mathtt{j}-2 & \mathtt{v}-0, \\ \mathtt{i}-4 & \mathtt{j}-3 & \mathtt{v}-6 \end{array}\right\rangle,7\right)$$

The ELEMENT_MATRIX constraint holds since its last argument $\texttt{VALUE} = 7$ is equal to the $\texttt{v}$ attribute of the k^{th} item of the $\texttt{MATRIX}$ collection such that $\texttt{MATRIX}[k].i = \texttt{INDEX\_I} = 1$ and $\texttt{MATRIX}[k].j = \texttt{INDEX\_J} = 3$.

Typical

$\texttt{MAX\_I} > 1$
$\texttt{MAX\_J} > 1$
$|\texttt{MATRIX}| > 3$
$\texttt{maxval}(\texttt{MATRIX.i}) > 1$
$\texttt{maxval}(\texttt{MATRIX.j}) > 1$
$\texttt{range}(\texttt{MATRIX.v}) > 1$

Symmetry

All occurrences of two distinct values in MATRIX.v or VALUE can be swapped; all occurrences of a value in MATRIX.v or VALUE can be renamed to any unused value.

Reformulation

The ELEMENT_MATRIX(MAX_I, MAX_J, INDEX_I, INDEX_J, MATRIX, VALUE) constraint can be expressed in term of MAX_I ELEMENT($\texttt{INDEX\_J}, \texttt{LINE}_i, \texttt{VAR}_i$) ($i \in [1, \texttt{MAX\_I}]$), where $\texttt{LINE}_i$ corresponds to the i-th line of the matrix MATRIX and of one ELEMENT($\texttt{INDEX\_I}, \langle \texttt{VAR}_1, \texttt{VAR}_2, \ldots, \texttt{VAR}_{\texttt{MAX\_I}} \rangle, \texttt{VALUE}$) constraint.

If we consider the **Example** slot we get the following ELEMENT constraints:

- ELEMENT$(3, \langle 4, 1, 7 \rangle, 7)$,
- ELEMENT$(3, \langle 1, 0, 8 \rangle, 8)$,
- ELEMENT$(3, \langle 3, 2, 1 \rangle, 1)$,
- ELEMENT$(3, \langle 0, 0, 6 \rangle, 6)$,
- ELEMENT$(1, \langle 7, 8, 1, 6 \rangle, 7)$.

Systems

NTH in **Choco**, ELEMENT in **Gecode**.

See also

common keyword: ELEM, ELEMENT *(array constraint)*.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: ternary constraint.

constraint network structure: centered cyclic(3) constraint network(1).

constraint type: data constraint.

filtering: arc-consistency.

modelling: array constraint, matrix.

Derived Collection

$$\text{col}\left(\begin{array}{l}\text{ITEM}-\text{collection}(\text{index\_i}-\text{dvar},\text{index\_j}-\text{dvar},\text{value}-\text{dvar}),\\ [\text{item}(\text{index\_i}-\text{INDEX\_I},\text{index\_j}-\text{INDEX\_J},\text{value}-\text{VALUE})]\end{array}\right)$$

Arc input(s) ITEM MATRIX

Arc generator $PRODUCT \mapsto$ collection(item, matrix)

Arc arity 2

Arc constraint(s)
- item.index_i = matrix.i
- item.index_j = matrix.j
- item.value = matrix.v

Graph property(ies) **NARC**= 1

Graph model Similar to the ELEMENT constraint except that the arc constraint is updated according to the fact that we have a two-dimensional matrix.

Parts (A) and (B) of Figure 5.336 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

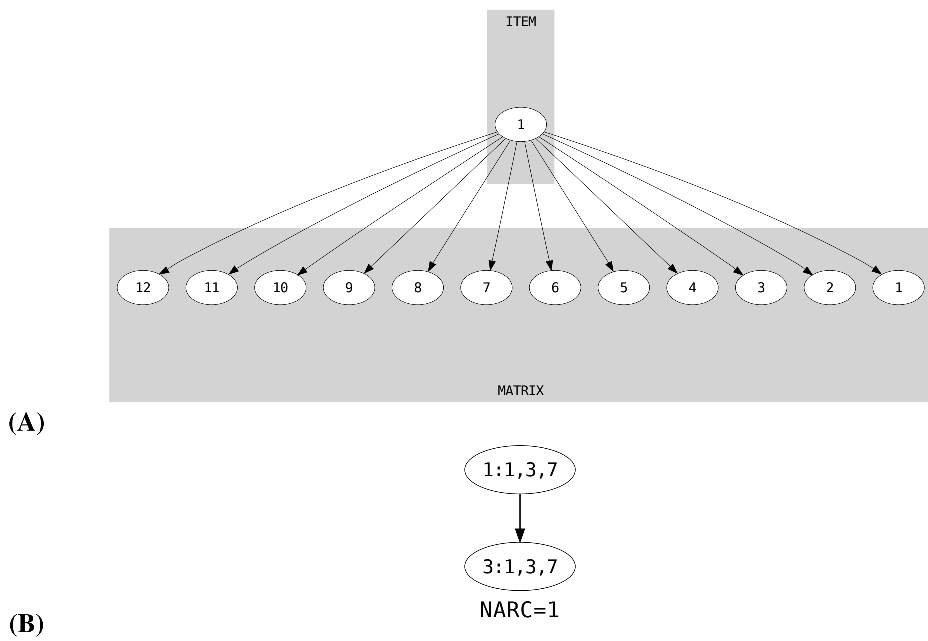


Figure 5.336: Initial and final graph of the ELEMENT_MATRIX constraint

Signature Because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite **NARC** = 1 to **NARC** ≥ 1 and simplify $\underline{\overline{\textbf{NARC}}}$ to $\overline{\textbf{NARC}}$.

Automaton Figure 5.337 depicts the automaton associated with the ELEMENT_MATRIX constraint. Let $\mathtt{I}_k$, $\mathtt{J}_k$ and $\mathtt{V}_k$ respectively be the `i`, the `j` and the `v` k^{th} attributes of the `MATRIX` collection. To each sextuple $(\mathtt{INDEX\_I}, \mathtt{INDEX\_J}, \mathtt{VALUE}, \mathtt{I}_k, \mathtt{J}_k, \mathtt{V}_k)$ corresponds a 0-1 signature variable S_k as well as the following signature constraint: $((\mathtt{INDEX\_I} = \mathtt{I}_k) \wedge (\mathtt{INDEX\_J} = \mathtt{J}_k) \wedge (\mathtt{VALUE} = \mathtt{V}_k)) \Leftrightarrow S_k$.

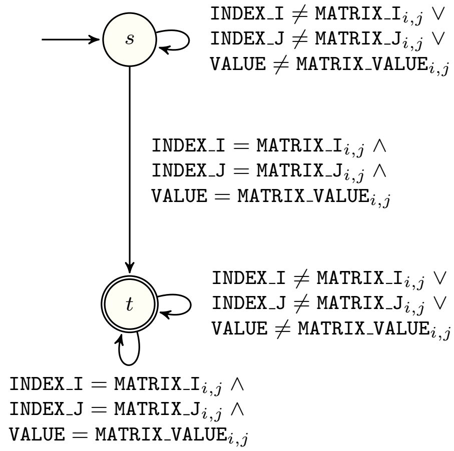


Figure 5.337: Automaton of the ELEMENT_MATRIX constraint

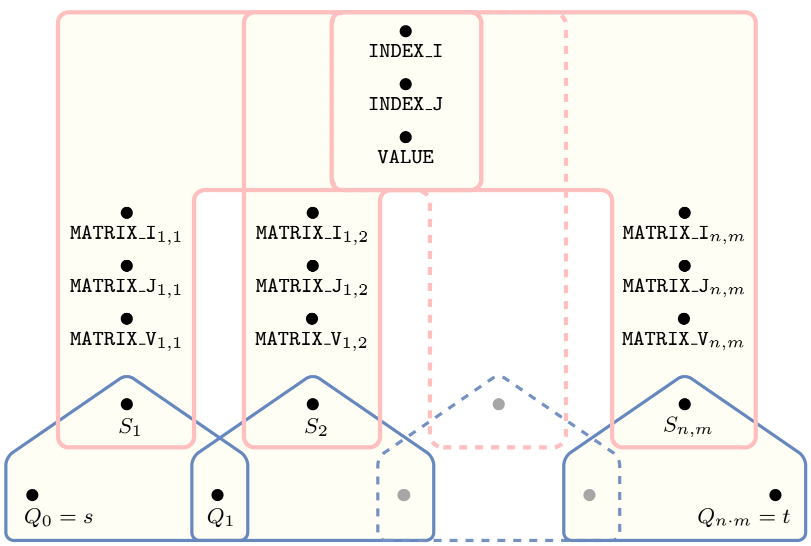


Figure 5.338: Hypergraph of the reformulation corresponding to the automaton of the ELEMENT_MATRIX constraint where n and m respectively stands for `MAX_I` and `MAX_J`

5.145 ELEMENT_PRODUCT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [311]

Constraint ELEMENT_PRODUCT(Y, TABLE, X, Z)

Synonym ELEMENT.

Arguments

Y	:	dvar
TABLE	:	collection(value−int)
X	:	dvar
Z	:	dvar

Restrictions

$\mathtt{Y} \geq 1$
$\mathtt{Y} \leq |\mathtt{TABLE}|$
$\mathtt{X} \geq 0$
$\mathtt{Z} \geq 0$
required(TABLE, value)
$\mathtt{TABLE.value} \geq 0$

Purpose Z is equal to the $\mathtt{Y}^{th}$ item of TABLE multiplied by X.

Example $(3, \langle 6, 9, 2, 9\rangle, 5, 10)$

The ELEMENT_PRODUCT constraint holds since its fourth argument $\mathtt{Z} = 10$ is equal to the 3^{th} ($\mathtt{Y} = 3$) item of the collection $\langle 6, 9, 2, 9\rangle$ multiplied by $\mathtt{X} = 5$.

Typical

$\mathtt{X} > 0$
$\mathtt{Z} > 0$
$|\mathtt{TABLE}| > 1$
$\mathtt{range}(\mathtt{TABLE.value}) > 1$
$\mathtt{TABLE.value} > 0$

Arg. properties

- Functional dependency: Z determined by Y, TABLE and X.
- Suffix-extensible wrt. TABLE.

Usage The ELEMENT_PRODUCT constraint was originally used in configuration problems [311]. In this context, Z denotes the cost of buying X units of type Y at cost TABLE[Y].value.

Reformulation By introducing an extra variable VAL, the ELEMENT_PRODUCT(Y, TABLE, X, Z) constraint can be expressed in term of an ELEMENT(Y, TABLE, VAL) constraint and of a product constraint $\mathtt{Z} = \mathtt{VAL} \cdot \mathtt{X}$.

See also **common keyword:** ELEM, ELEMENT, ELEMENT_GREATEREQ, ELEMENT_LESSEQ *(array constraint)*.

Keywords **application area:** configuration problem.

constraint arguments: pure functional dependency.

constraint type: data constraint.

modelling: array constraint, table, functional dependency, variable subscript.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{ITEM}-\texttt{collection}(\texttt{y}-\texttt{dvar},\texttt{x}-\texttt{dvar},\texttt{z}-\texttt{dvar}),\\ [\texttt{item}(\texttt{y}-\texttt{Y},\texttt{x}-\texttt{X},\texttt{z}-\texttt{Z})]\end{array}\right)$$

Arc input(s) ITEM TABLE

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{item},\texttt{table})$

Arc arity 2

Arc constraint(s)
- $\texttt{item.y} = \texttt{table.key}$
- $\texttt{item.z} = \texttt{item.x} * \texttt{table.value}$

Graph property(ies) **NARC**$= 1$

Graph model

We use the derived collection ITEM for putting together the Y, the X and Z parameters of the ELEMENT_PRODUCT constraint. Within the arc constraint we use the implicit attribute key that associates to each item of a collection its position within the collection.

Parts (A) and (B) of Figure 5.339 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

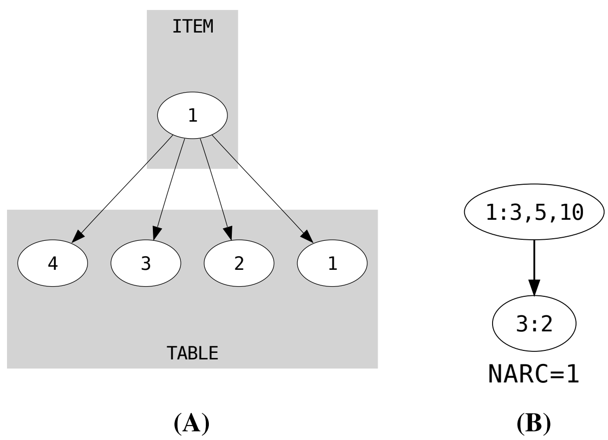


Figure 5.339: Initial and final graph of the ELEMENT_PRODUCT constraint

Signature

Because of the first condition of the arc constraint the final graph cannot have more than one arc. Therefore we can rewrite $\mathbf{NARC} = 1$ to $\mathbf{NARC} \geq 1$ and simplify $\overline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

5.146 ELEMENT_SPARSE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint ELEMENT_SPARSE(ITEM, TABLE, DEFAULT)

Usual name ELEMENT

Arguments

```
ITEM    : collection(index-dvar, value-dvar)
TABLE   : collection(index-int, value-int)
DEFAULT : int
```

Restrictions

required(ITEM, [index, value])
ITEM.index ≥ 1
$|$ITEM$| = 1$
$|$TABLE$| > 0$
required(TABLE, [index, value])
TABLE.index ≥ 1
distinct(TABLE, index)

Purpose

ITEM[1].value is equal to one of the entries of the table TABLE or to the default value DEFAULT if the entry ITEM[1].index does not exist in TABLE.

Example

$$\left(\begin{array}{l}\langle \texttt{index}-2\ \texttt{value}-5\rangle, \\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6, \\ \texttt{index}-2 & \texttt{value}-5, \\ \texttt{index}-4 & \texttt{value}-2, \\ \texttt{index}-8 & \texttt{value}-9\end{array}\right\rangle, 5\end{array}\right)$$

The ELEMENT_SPARSE constraint holds since its first argument ITEM corresponds to the second item of the TABLE collection.

Typical

$|$TABLE$| > 1$
range(TABLE.value) > 1

Symmetries

- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEM.value, TABLE.value or DEFAULT can be swapped; all occurrences of a value in ITEM.value, TABLE.value or DEFAULT can be renamed to any unused value.

Usage A sometimes more compact form of the ELEMENT constraint: we are not obliged to specify explicitly the table entries that correspond to the specified default value. This can sometimes reduce drastically memory utilisation.

Remark

The original constraint of **CHIP** had an additional parameter SIZE giving the maximum value of ITEM.index.

Reformulation

Let I and V respectively denote ITEM[1].index and ITEM[1].value. The ELEMENT_SPARSE(ITEM, TABLE, DEFAULT) constraint can be expressed in term of a reified constraint of the form:

$((\mathtt{I} = \mathtt{TABLE}[1].\mathtt{index} \wedge \mathtt{V} = \mathtt{TABLE}[1].\mathtt{value}) \vee$
$(\mathtt{I} = \mathtt{TABLE}[2].\mathtt{index} \wedge \mathtt{V} = \mathtt{TABLE}[2].\mathtt{value}) \vee$
$\dots$
$(\mathtt{I} = \mathtt{TABLE}[|\mathtt{TABLE}|].\mathtt{index} \wedge \mathtt{V} = \mathtt{TABLE}[\mathtt{TABLE}|].\mathtt{value})) \quad \vee$
$((\mathtt{I} \neq \mathtt{TABLE}[1].\mathtt{index}) \wedge$
$(\mathtt{I} \neq \mathtt{TABLE}[2].\mathtt{index}) \wedge$
$\dots$
$(\mathtt{I} \neq \mathtt{TABLE}[|\mathtt{TABLE}|].\mathtt{index}) \wedge$
$(\mathtt{V} = \mathtt{DEFAULT}))$.

See also

common keyword: ELEM, ELEMENT *(array constraint)*, ELEMENTS_SPARSE *(sparse table)*.

implies: ELEMENTS_SPARSE.

system of constraints: ELEMENTS_SPARSE.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: binary constraint.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: arc-consistency.

modelling: array constraint, table, sparse table, sparse functional dependency, variable indexing.

Derived Collections

$$\texttt{col}\left(\begin{array}{l}\texttt{DEF}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{value}-\texttt{int}),\\ [\texttt{item}(\texttt{index}-0,\texttt{value}-\texttt{DEFAULT})]\end{array}\right)$$

$$\texttt{col}\left(\begin{array}{l}\texttt{TABLE\_DEF}-\texttt{collection}(\texttt{index}-\texttt{dvar},\texttt{value}-\texttt{dvar}),\\ \left[\begin{array}{l}\texttt{item}(\texttt{index}-\texttt{TABLE.index},\texttt{value}-\texttt{TABLE.value}),\\ \texttt{item}(\texttt{index}-\texttt{DEF.index},\texttt{value}-\texttt{DEF.value})\end{array}\right]\end{array}\right)$$

Arc input(s) ITEM TABLE_DEF

Arc generator $PRODUCT \mapsto$ collection(item, table_def)

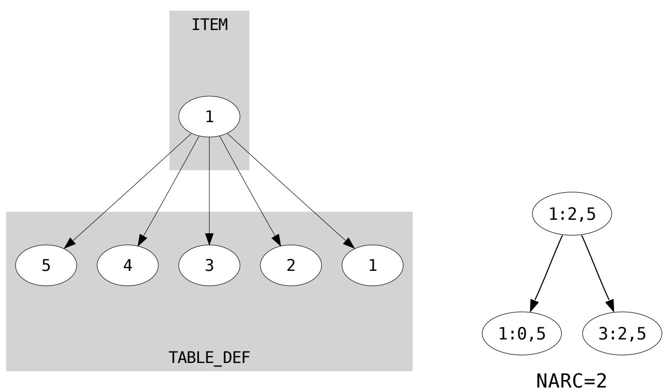
Arc arity 2

Arc constraint(s)
- item.value = table_def.value
- item.index = table_def.index $\vee$ table_def.index = 0

Graph property(ies) **NARC** ≥ 1

Graph model The final graph has between one and two arc constraints: it has two arcs when the default value DEFAULT occurs also in the table TABLE; otherwise it has only one arc.

Parts (A) and (B) of Figure 5.340 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property the arcs of the final graph are outline with thick lines.



(A) **(B)**

Figure 5.340: Initial and final graph of the ELEMENT_SPARSE constraint

Automaton Figure 5.341 depicts the automaton associated with the ELEMENT_SPARSE constraint. Let INDEX and VALUE respectively be the index and the value attributes of the unique item of the ITEM collection. Let $\mathtt{INDEX}_i$ and $\mathtt{VALUE}_i$ respectively be the index and the value attributes of the i^{th} item of the TABLE collection. To each quintuple $(\mathtt{INDEX}, \mathtt{VALUE}, \mathtt{DEFAULT}, \mathtt{INDEX}_i, \mathtt{VALUE}_i)$ corresponds a signature variable S_i as well as the following signature constraint:

$$\begin{cases} (\mathtt{INDEX} \neq \mathtt{INDEX}_i \ \wedge \mathtt{VALUE} \neq \mathtt{DEFAULT}) & \Leftrightarrow \quad S_i = 0 \ \wedge \\ (\mathtt{INDEX} = \mathtt{INDEX}_i \ \wedge \mathtt{VALUE} = \mathtt{VALUE}_i \ \ ) & \Leftrightarrow \quad S_i = 1 \ \wedge \\ (\mathtt{INDEX} \neq \mathtt{INDEX}_i \ \wedge \mathtt{VALUE} = \mathtt{DEFAULT}) & \Leftrightarrow \quad S_i = 2 \end{cases} .$$

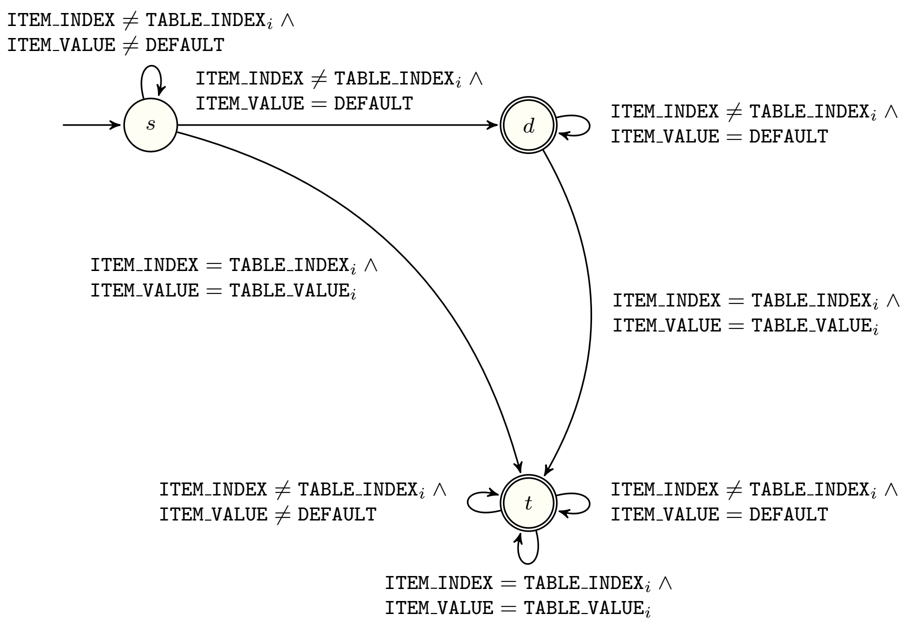


Figure 5.341: Automaton of the ELEMENT_SPARSE constraint

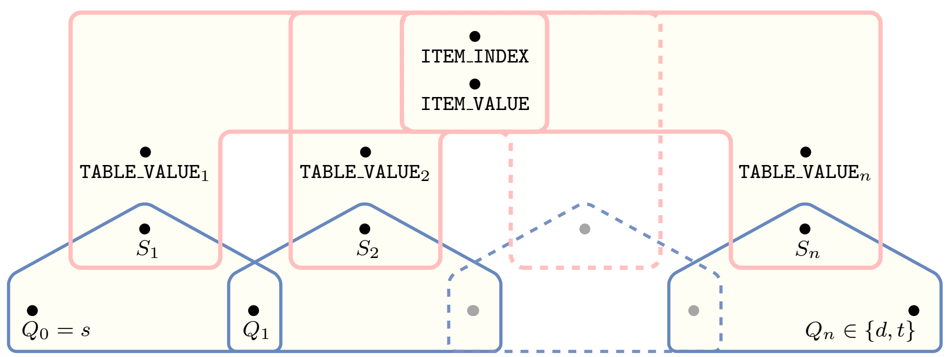


Figure 5.342: Hypergraph of the reformulation corresponding to the automaton of the ELEMENT_SPARSE constraint

5.147 ELEMENTN

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin P. Flener

Constraint ELEMENTN(INDEX, TABLE, ENTRIES)

Arguments

INDEX : dvar
TABLE : collection(value−int)
ENTRIES : collection(entry−dvar)

Restrictions

$\text{INDEX} \geq 1$
$\text{INDEX} \leq |\text{TABLE}| - |\text{ENTRIES}| + 1$
$|\text{TABLE}| > 0$
$|\text{ENTRIES}| > 0$
$|\text{TABLE}| \geq |\text{ENTRIES}|$
required(TABLE, value)
required(ENTRIES, entry)

Purpose $\forall i \in [1, |\text{ENTRIES}|] : \text{ENTRIES}[i].\text{entry} = \text{TABLE}[\text{INDEX} + i - 1].\text{value}$

Example $(3, \langle 6, 9, 2, 9\rangle, \langle 2, 9\rangle)$

The ELEMENTN constraint holds since its third argument ENTRIES $= \langle 2, 9\rangle$ is set to the subsequence starting at the third (i.e., INDEX $= 3$) item of the table TABLE $= \langle 6, 9, 2, 9\rangle$.

Typical

$|\text{TABLE}| > 1$
range(TABLE.value) > 1
$|\text{ENTRIES}| > 1$

Symmetry All occurrences of two distinct values in TABLE.value or ENTRIES.entry can be swapped; all occurrences of a value in TABLE.value or ENTRIES.entry can be renamed to any unused value.

Arg. properties Suffix-extensible wrt. TABLE.

Usage The ELEMENTN constraint is useful for extracting of subsequence of fixed length from a given sequence.

Reformulation Let $I_1 = \text{INDEX}, I_2 = \text{INDEX} + 1, \ldots, I_{|\text{ENTRIES}|} = \text{INDEX} + |\text{ENTRIES}| - 1$. The ELEMENTN(INDEX, TABLE, $\langle \text{entry}-E_1, \text{entry}-E_2, \ldots, \text{entry}-E_{|\text{ENTRIES}|}\rangle$) constraint can be expressed in term of a conjunction of $|\text{ENTRIES}|$ ELEMENT constraints of the form:

ELEMENT(I_1, TABLE, E_1),
ELEMENT(I_2, TABLE, E_2),
...
ELEMENT($\text{INDEX} + |\text{ENTRIES}| - 1$, TABLE, $E_{|\text{ENTRIES}|}$).

See also **common keyword:** ELEMENT *(data constraint)*.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: data constraint, sliding sequence constraint.

filtering: arc-consistency.

modelling: table.

Automaton Figure 5.343 depicts the automaton associated with the ELEMENTN constraint of the **Example** slot. Let `I` and $\mathtt{E}_k$ respectively denote the `INDEX` argument and the `entry` attribute of the k^{th} item of the `ENTRIES` collection. Figure 5.344 depicts the reformulation of the ELEMENTN constraint.

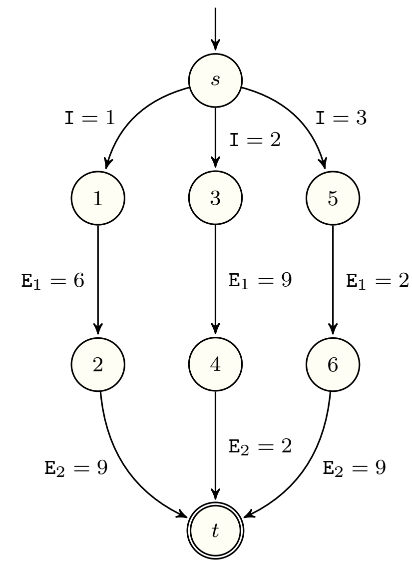


Figure 5.343: Automaton of the ELEMENTN constraint given in the example

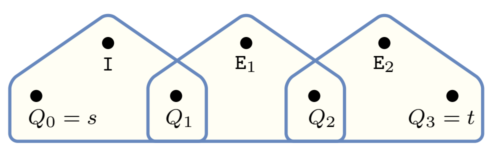


Figure 5.344: Hypergraph of the reformulation corresponding to the automaton of the ELEMENTN constraint

5.148 ELEMENTS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from ELEMENT.

Constraint $\text{ELEMENTS}(\texttt{ITEMS}, \texttt{TABLE})$

Arguments

$$\begin{array}{lll} \texttt{ITEMS} & : & \texttt{collection}(\texttt{index}-\texttt{dvar}, \texttt{value}-\texttt{dvar}) \\ \texttt{TABLE} & : & \texttt{collection}(\texttt{index}-\texttt{int}, \texttt{value}-\texttt{dvar}) \end{array}$$

Restrictions

$\texttt{required}(\texttt{ITEMS}, [\texttt{index}, \texttt{value}])$
$\texttt{ITEMS.index} \geq 1$
$\texttt{ITEMS.index} \leq |\texttt{TABLE}|$
$\texttt{required}(\texttt{TABLE}, [\texttt{index}, \texttt{value}])$
$\texttt{TABLE.index} \geq 1$
$\texttt{TABLE.index} \leq |\texttt{TABLE}|$
$\texttt{distinct}(\texttt{TABLE}, \texttt{index})$

Purpose All the items of ITEMS should be equal to one of the entries of the table TABLE.

Example

$$\left(\begin{array}{l} \langle \texttt{index}-4\ \texttt{value}-9, \texttt{index}-1\ \texttt{value}-6 \rangle, \\ \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{value}-6, \\ \texttt{index}-2 & \texttt{value}-9, \\ \texttt{index}-3 & \texttt{value}-2, \\ \texttt{index}-4 & \texttt{value}-9 \end{array} \right\rangle \end{array} \right)$$

The ELEMENTS constraint holds since each item of its first argument ITEMS corresponds to an item of the TABLE collection: the first item $\langle \texttt{index}-4\ \texttt{value}-9 \rangle$ of ITEMS corresponds to the fourth item of TABLE, while the second item $\langle \texttt{index}-1\ \texttt{value}-6 \rangle$ of ITEMS corresponds to the first item of TABLE.

Typical

$|\texttt{ITEMS}| > 1$
$\texttt{range}(\texttt{ITEMS.index}) > 1$
$|\texttt{TABLE}| > 1$
$\texttt{range}(\texttt{TABLE.value}) > 1$

Symmetries

- Items of ITEMS are permutable.
- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEMS.value or TABLE.value can be swapped; all occurrences of a value in ITEMS.value or TABLE.value can be renamed to any unused value.

Arg. properties Functional dependency: ITEMS.value determined by ITEMS.index and TABLE.

Usage Used for replacing several ELEMENT constraints sharing exactly the same table by a single constraint.

Reformulation The ELEMENTS($\langle$index $-$ I_1 value $-$ V_1, index $-$ I_2 value $-$ V_2, ..., index $-$ $I_{|\text{ITEMS}|}$ value $-$ $V_{|\text{ITEMS}|}\rangle$, TABLE) constraint can be expressed in term of a conjunction of |ITEMS| ELEM constraints of the form:

ELEM($\langle$index $-$ I_1 value $-$ $V_1\rangle$, TABLE),
ELEM($\langle$index $-$ I_2 value $-$ $V_2\rangle$, TABLE),
...
ELEM($\langle$index $-$ $I_{|\text{ITEMS}|}$ value $-$ $V_{|\text{ITEMS}|}\rangle$, TABLE).

See also **implied by:** ELEM, ELEMENTS_ALLDIFFERENT.

part of system of constraints: ELEM, ELEMENT.

Keywords **constraint arguments:** pure functional dependency.

constraint type: data constraint, system of constraints.

filtering: arc-consistency.

modelling: table, shared table, functional dependency.

Cond. implications ELEMENTS(ITEMS, TABLE)
with distinct(ITEMS, index)
and TABLE.value ≥ 0
implies BIN_PACKING_CAPA(TABLE, ITEMS).

Arc input(s)	ITEMS TABLE
Arc generator	$PRODUCT \mapsto$ collection(items, table)
Arc arity	2
Arc constraint(s)	• items.index = table.index • items.value = table.value
Graph property(ies)	**NARC** = |ITEMS|

Graph model Parts (A) and (B) of Figure 5.345 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

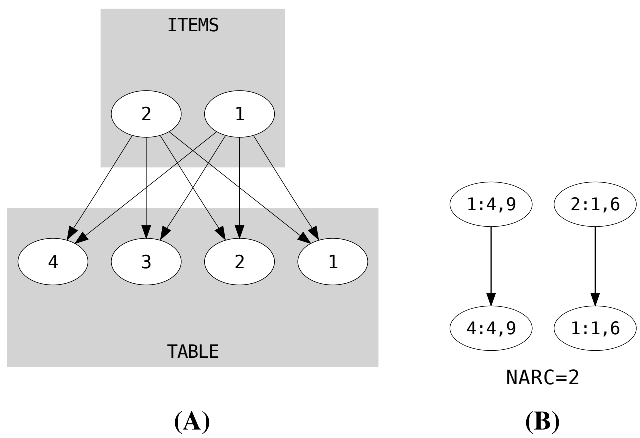


Figure 5.345: Initial and final graph of the ELEMENTS constraint

Signature Since all the index attributes of TABLE collection are distinct and because of the first condition items.index = table.index of the arc constraint, a source vertex of the final graph can have at most one successor. Therefore |ITEMS| is the maximum number of arcs of the final graph and we can rewrite **NARC** = |ITEMS| to **NARC** $\geq$ |ITEMS|. So we can simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.149 ELEMENTS_ALLDIFFERENT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from ELEMENTS and ALLDIFFERENT.

Constraint ELEMENTS_ALLDIFFERENT(ITEMS, TABLE)

Synonyms ELEMENTS_ALLDIFF, ELEMENTS_ALLDISTINCT, PERMUTATION.

Arguments

ITEMS : collection(index−dvar, value−dvar)
TABLE : collection(index−int, value−dvar)

Restrictions

required(ITEMS, [index, value])
ITEMS.index ≥ 1
ITEMS.index $\leq$ |TABLE|
|ITEMS| = |TABLE|
required(TABLE, [index, value])
TABLE.index ≥ 1
TABLE.index $\leq$ |TABLE|
distinct(TABLE, index)

Purpose All the items of the ITEMS collection should be equal to one of the entries of the table TABLE and all the variables ITEMS.index should take distinct values.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{ll}\texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-4 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2\end{array}\right\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\rangle\end{array}\right)$$

The ELEMENTS_ALLDIFFERENT constraint holds since, as depicted by Figure 5.346, there is a one to one correspondence between the items of the ITEMS collection and the items of the TABLE collection.

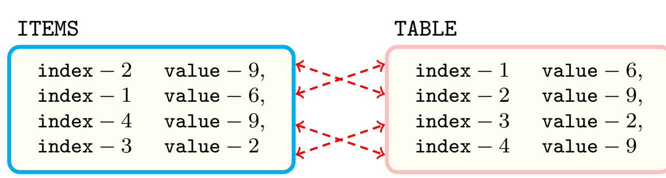


Figure 5.346: Illustration of the one to one correspondence between the items of ITEMS and the items of TABLE

Typical

$|\texttt{ITEMS}| > 1$
$\texttt{range}(\texttt{ITEMS.value}) > 1$
$|\texttt{TABLE}| > 1$
$\texttt{range}(\texttt{TABLE.value}) > 1$

Symmetries

- Arguments are permutable w.r.t. permutation $(\texttt{ITEMS}, \texttt{TABLE})$.
- Items of ITEMS are permutable.
- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEMS.value or TABLE.value can be swapped; all occurrences of a value in ITEMS.value or TABLE.value can be renamed to any unused value.

Arg. properties

Functional dependency: ITEMS.value determined by ITEMS.index and TABLE.

Usage

Used for replacing by a single ELEMENTS_ALLDIFFERENT constraint an ALLDIFFERENT and a set of ELEMENT constraints having the following structure:

- The union of the index variables of the ELEMENT constraints is equal to the set of variables of the ALLDIFFERENT constraint.
- All the ELEMENT constraints share exactly the same table.

For instance, the constraint given in the **Example** slot is equivalent to the conjunction of the following set of constraints:

$$\text{ALLDIFFERENT}(\langle \texttt{var}-2, \texttt{var}-1, \texttt{var}-4, \texttt{var}-3\rangle)$$

$$\text{ELEMENT}\left(\begin{array}{l}\langle\ \texttt{index}-2 \quad \texttt{value}-9\ \rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\rangle\end{array}\right)$$

$$\text{ELEMENT}\left(\begin{array}{l}\langle\ \texttt{index}-1 \quad \texttt{value}-6\ \rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\rangle\end{array}\right)$$

$$\text{ELEMENT}\left(\begin{array}{l}\langle\ \texttt{index}-3 \quad \texttt{value}-2\ \rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\rangle\end{array}\right)$$

$$\text{ELEMENT}\left(\begin{array}{l}\langle\ \texttt{index}-4 \quad \texttt{value}-9\ \rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-2,\\ \texttt{index}-4 & \texttt{value}-9\end{array}\right\rangle\end{array}\right)$$

As a practical example of utilisation of the ELEMENTS_ALLDIFFERENT constraint we show how to model the link between a permutation consisting of a single cycle and its expanded form. For instance, to the permutation $3, 6, 5, 2, 4, 1$ corresponds the sequence $3\ 5\ 4\ 2\ 6\ 1$. Let us note $S_1, S_2, S_3, S_4, S_5, S_6$ the permutation and $V_1 V_2 V_3 V_4 V_5 V_6$ its expanded form (see Figure 5.347).

The constraint:

$$\text{ELEMENTS\_ALLDIFFERENT}\left(\begin{array}{l} \left\langle\begin{array}{ll} \texttt{index}-V_1 & \texttt{value}-V_2, \\ \texttt{index}-V_2 & \texttt{value}-V_3, \\ \texttt{index}-V_3 & \texttt{value}-V_4, \\ \texttt{index}-V_4 & \texttt{value}-V_5, \\ \texttt{index}-V_5 & \texttt{value}-V_6, \\ \texttt{index}-V_6 & \texttt{value}-V_1 \end{array}\right\rangle, \\ \left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{value}-S_1, \\ \texttt{index}-2 & \texttt{value}-S_2, \\ \texttt{index}-3 & \texttt{value}-S_3, \\ \texttt{index}-4 & \texttt{value}-S_4, \\ \texttt{index}-5 & \texttt{value}-S_5, \\ \texttt{index}-6 & \texttt{value}-S_6 \end{array}\right\rangle \end{array}\right)$$

models the fact that $S_1, S_2, S_3, S_4, S_5, S_6$ corresponds to a permutation with a single cycle. It also expresses the link between the variables $S_1, S_2, S_3, S_4, S_5, S_6$ and $V_1, V_2, V_3, V_4, V_5, V_6$.

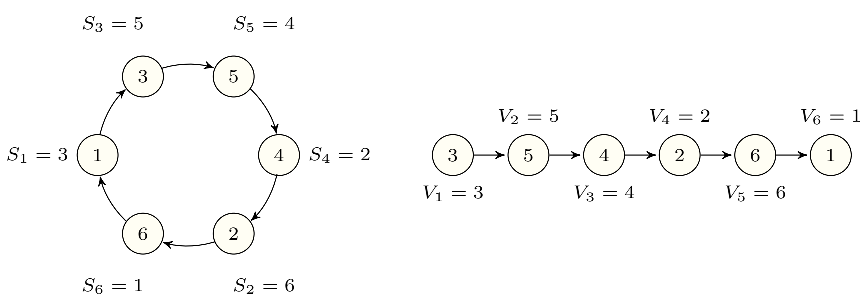


Figure 5.347: Two representations of a permutation containing a single cycle

Reformulation

The ELEMENTS_ALLDIFFERENT$(\langle \texttt{index} - I_1\ \texttt{value} - V_1, \texttt{index} - I_2\ \texttt{value} - V_2, \ldots, \texttt{index} - I_{|\texttt{ITEMS}|}\ \texttt{value} - V_{|\texttt{ITEMS}|}\rangle, \texttt{TABLE})$ constraint can be expressed in term of a conjunction of $|\texttt{ITEMS}|$ ELEM constraints and of one ALLDIFFERENT constraint of the form:

ELEM$(\langle \texttt{index} - I_1\ \texttt{value} - V_1\rangle, \texttt{TABLE})$,
ELEM$(\langle \texttt{index} - I_2\ \texttt{value} - V_2\rangle, \texttt{TABLE})$,
...
ELEM$(\langle \texttt{index} - I_{|\texttt{ITEMS}|}\ \texttt{value} - V_{|\texttt{ITEMS}|}\rangle, \texttt{TABLE})$,
ALLDIFFERENT$(\langle I_1, I_2, \ldots, I_{|\texttt{ITEMS}|}\rangle)$.

See also **implies:** ELEMENTS, INDEXED_SUM.

used in reformulation: ALLDIFFERENT, ELEM, ELEMENT.

Keywords **characteristic of a constraint:** disequality.

combinatorial object: permutation.

constraint type: data constraint.

modelling: array constraint, table, functional dependency.

Cond. implications ELEMENTS_ALLDIFFERENT(ITEMS, TABLE)
with TABLE.value ≥ 0
implies BIN_PACKING_CAPA(TABLE, ITEMS).

Arc input(s) ITEMS TABLE

Arc generator $PRODUCT \mapsto$ collection(items, table)

Arc arity 2

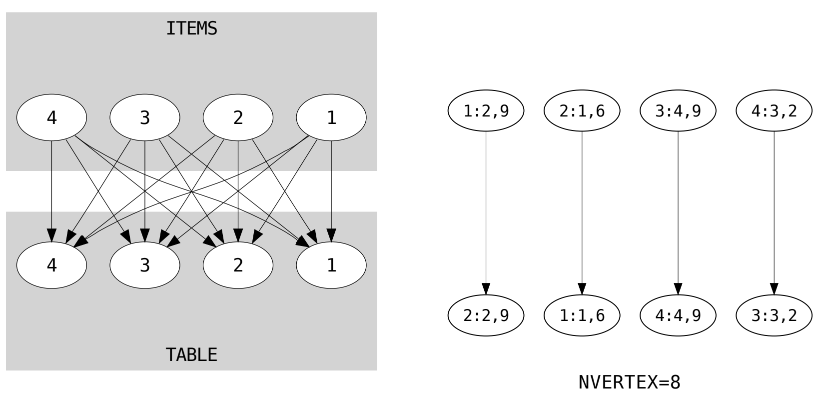
Arc constraint(s)
- items.index = table.index
- items.value = table.value

Graph property(ies) **NVERTEX** = |ITEMS| + |TABLE|

Graph model The fact that all variables ITEMS.index are pairwise different is derived from the conjunctions of the following facts:

- From the graph property **NVERTEX** = |ITEMS| + |TABLE| it follows that all vertices of the initial graph belong also to the final graph,
- A vertex v belongs to the final graph if there is at least one constraint involving v that holds,
- From the first condition items.index = table.index of the arc constraint, and from the restriction distinct(TABLE.index) it follows: for all vertices v generated from the collection ITEMS at most one constraint involving v holds.

Parts (A) and (B) of Figure 5.348 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NVERTEX** graph property, the vertices of the final graph are stressed in bold.



(A) **(B)**

Figure 5.348: Initial and final graph of the ELEMENTS_ALLDIFFERENT constraint

Signature Since the final graph cannot have more than |ITEMS| + |TABLE| vertices one can simplify $\underline{\overline{\mathbf{NVERTEX}}}$ to $\overline{\mathbf{NVERTEX}}$.

5.150 ELEMENTS_SPARSE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from ELEMENT_SPARSE.

Constraint ELEMENTS_SPARSE(ITEMS, TABLE, DEFAULT)

Arguments

ITEMS	:	collection(index−dvar, value−dvar)
TABLE	:	collection(index−int, value−int)
DEFAULT	:	int

Restrictions

required(ITEMS, [index, value])
ITEMS.index ≥ 1
required(TABLE, [index, value])
TABLE.index ≥ 1
distinct(TABLE, index)

Purpose All the items of ITEMS should be equal to one of the entries of the table TABLE or to the default value DEFAULT if the entry ITEMS.index does not occurs among the values of the index attribute of the TABLE collection.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{ll}\texttt{index}-8 & \texttt{value}-9,\\ \texttt{index}-3 & \texttt{value}-5,\\ \texttt{index}-2 & \texttt{value}-5\end{array}\right\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{value}-6,\\ \texttt{index}-2 & \texttt{value}-5,\\ \texttt{index}-4 & \texttt{value}-2,\\ \texttt{index}-8 & \texttt{value}-9\end{array}\right\rangle, 5\end{array}\right)$$

The ELEMENTS_SPARSE constraint holds since:

- The first and third items (items $\langle$index $-\,8$ value $-\,9\rangle$ and $\langle$index $-\,2$ value $-\,5\rangle$) of its ITEMS collection respectively correspond to the fourth and second item of its TABLE collection.
- The index attribute of the second item of its ITEMS collection (i.e., value 3) does not correspond to any index of the TABLE collection. Therefore the value attribute of the second item of the ITEMS collection is set the the default value 5 given by the last argument of the ELEMENTS_SPARSE constraint.

Typical

$|\texttt{ITEMS}| > 1$
range(ITEMS.value) > 1
$|\texttt{TABLE}| > 1$
range(TABLE.value) > 1

Symmetries

- Items of ITEMS are permutable.
- Items of TABLE are permutable.
- All occurrences of two distinct values in ITEMS.value, TABLE.value or DEFAULT can be swapped; all occurrences of a value in ITEMS.value, TABLE.value or DEFAULT can be renamed to any unused value.

Usage

Used for replacing several ELEMENT constraints sharing exactly the same sparse table by a single constraint.

Reformulation

Let $\mathtt{I}_k$ and $\mathtt{V}_k$ respectively denote $\mathtt{ITEMS}[k].\mathtt{index}$ and $\mathtt{ITEMS}[k].\mathtt{value}$ $(k \in [1, |\mathtt{ITEMS}|[])$. The ELEMENTS_SPARSE$(\mathtt{ITEMS}, \mathtt{TABLE}, \mathtt{DEFAULT})$ constraint can be expressed in term of $|\mathtt{ITEMS}|[$ reified constraints of the form:

$((\mathtt{I}_k = \mathtt{TABLE}[1].\mathtt{index} \wedge \mathtt{V}_k = \mathtt{TABLE}[1].\mathtt{value}) \vee$
$(\mathtt{I}_k = \mathtt{TABLE}[2].\mathtt{index} \wedge \mathtt{V}_k = \mathtt{TABLE}[2].\mathtt{value}) \vee$
$\dots$
$(\mathtt{I}_k = \mathtt{TABLE}[|\mathtt{TABLE}|].\mathtt{index} \wedge \mathtt{V}_k = \mathtt{TABLE}[\mathtt{TABLE}|].\mathtt{value})) \quad \vee$
$((\mathtt{I}_k \neq \mathtt{TABLE}[1].\mathtt{index}) \wedge$
$(\mathtt{I}_k \neq \mathtt{TABLE}[2].\mathtt{index}) \wedge$
$\dots$
$(\mathtt{I}_k \neq \mathtt{TABLE}[|\mathtt{TABLE}|].\mathtt{index}) \wedge$
$(\mathtt{V}_k = \mathtt{DEFAULT}))$.

See also

common keyword: ELEM, ELEMENT *(data constraint)*, ELEMENT_SPARSE *(sparse table)*.

implied by: ELEMENT_SPARSE.

part of system of constraints: ELEMENT_SPARSE.

Keywords

characteristic of a constraint: derived collection.

constraint type: data constraint, system of constraints.

filtering: arc-consistency.

modelling: table, shared table, sparse table, sparse functional dependency.

Derived Collections

$$\texttt{col}\left(\begin{array}{l}\texttt{DEF}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{value}-\texttt{int}),\\ [\texttt{item}(\texttt{index}-0,\texttt{value}-\texttt{DEFAULT})]\end{array}\right)$$

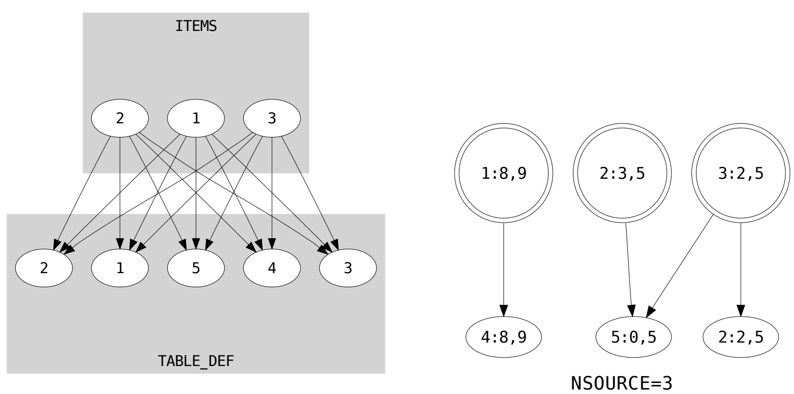
$$\texttt{col}\left(\begin{array}{l}\texttt{TABLE\_DEF}-\texttt{collection}(\texttt{index}-\texttt{dvar},\texttt{value}-\texttt{dvar}),\\ \left[\begin{array}{l}\texttt{item}(\texttt{index}-\texttt{TABLE.index},\texttt{value}-\texttt{TABLE.index}),\\ \texttt{item}(\texttt{index}-\texttt{DEF.index},\texttt{value}-\texttt{DEF.value})\end{array}\right]\end{array}\right)$$

Arc input(s)	ITEMS TABLE_DEF
Arc generator	$PRODUCT \mapsto \texttt{collection}(\texttt{items},\texttt{table\_def})$
Arc arity	2
Arc constraint(s)	• $\texttt{items.value} = \texttt{table\_def.value}$ • $\texttt{items.index} = \texttt{table\_def.index} \vee \texttt{table\_def.index} = 0$
Graph property(ies)	$\mathbf{NSOURCE} = \lvert\texttt{ITEMS}\rvert$

Graph model

An item of the ITEMS collection may have up to two successors (see, for example, the third item of the ITEMS collection of the **Example** slot). Therefore we use the graph property $\mathbf{NSOURCE} = \lvert\texttt{ITEMS}\rvert$ for enforcing the fact that each item of the ITEMS collection has at least one successor.

Parts (A) and (B) of Figure 5.349 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** graph property, the vertices of the final graph are drawn with a double circle.



(A) **(B)**

Figure 5.349: Initial and final graph of the ELEMENTS_SPARSE constraint

Signature

On the one hand note that ITEMS is equal to the number of sources of the initial graph. On the other hand note that, in the initial graph, all the vertices that are not sources correspond to sinks. Since isolated vertices are eliminated from the final graph the sinks of the initial graph cannot become sources of the final graph. Therefore the maximum number of sources of the final graph is equal to ITEMS. We can rewrite $\mathbf{NSOURCE} = \lvert\texttt{ITEMS}\rvert$ to $\mathbf{NSOURCE} \geq \lvert\texttt{ITEMS}\rvert$ and simplify $\underline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$.

5.151 EQ

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint EQ(VAR1, VAR2)

Synonym XEQY.

Arguments

VAR1 : dvar
VAR2 : dvar

Restriction

Purpose Enforce the fact that two variables are equal.

Example $(8, 8)$

The EQ constraint holds since 8 is equal to 8.

Symmetries

- Arguments are permutable w.r.t. permutation (VAR1, VAR2).
- All occurrences of a value in VAR1 or VAR2 can be renamed to any unused value.

Arg. properties

- Functional dependency: VAR2 determined by VAR1.
- Functional dependency: VAR1 determined by VAR2.

Systems EQ in **Choco**, REL in **Gecode**, XEQY in **JaCoP**, #= in **SICStus**.

See also

common keyword: GT, LT *(binary constraint,arithmetic constraint)*.

generalisation: ALL_EQUAL *(equality between more than two variables)*, EQ_CST *(*constant *added)*, EQ_SET *(*variable *replaced by* set variable*)*.

implies: ABS_VALUE, GEQ, LEQ, SAME_SIGN, ZERO_OR_NOT_ZERO.

negation: NEQ.

Keywords

constraint arguments: binary constraint, pure functional dependency.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

modelling: functional dependency.

5.152 EQ_CST

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint EQ_CST(VAR1, VAR2, CST2)

Arguments

VAR1	:	dvar
VAR2	:	dvar
CST2	:	int

Purpose Enforce the fact that the first variable is equal to the sum of the second variable and the constant.

Example $(8, 2, 6)$

The EQ_CST constraint holds since 8 is equal to $2 + 6$.

Typical $\texttt{CST2} \neq 0$

Symmetries
- Arguments are permutable w.r.t. permutation (VAR1) (VAR2, CST2).
- One and the same constant can be added to VAR1 and VAR2.
- One and the same constant can be added to VAR1 and CST2.

Arg. properties
- Functional dependency: VAR1 determined by VAR2 and CST2.
- Functional dependency: VAR2 determined by VAR1 and CST2.
- Functional dependency: CST2 determined by VAR1 and VAR2.

See also
implies: GEQ_CST, LEQ_CST.
negation: NEQ_CST.
specialisation: EQ *(*constant *set to* 0*)*.

Keywords
constraint arguments: binary constraint, pure functional dependency.
constraint type: predefined constraint, arithmetic constraint.
filtering: arc-consistency.
modelling: functional dependency.

5.153 EQ_SET

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Used for defining ALLDIFFERENT_BETWEEN_SETS.

Constraint EQ_SET(SET1, SET2)

Arguments

SET1	:	svar
SET2	:	svar

Purpose Constraint the set SET1 to be equal to the set SET2.

Example $(\{3, 5\}, \{3, 5\})$

Symmetries

- Arguments are permutable w.r.t. permutation (SET1, SET2).
- All occurrences of a value in SET1 or SET2 can be renamed to any unused value.

Systems EQ in **Choco**, REL in **Gecode**.

Used in ALLDIFFERENT_BETWEEN_SETS.

See also **specialisation:** EQ (set variable *replaced by* variable).

Keywords **characteristic of a constraint:** equality.
constraint arguments: binary constraint, constraint involving set variables.
constraint type: predefined constraint.

5.154 EQUAL_SBOXES

► ▷ ◁ ◄ DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [349]

Constraint EQUAL_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonym EQUAL.

Types

VARIABLES : collection(v−dvar)
INTEGERS : collection(v−int)
POSITIVES : collection(v−int)

Arguments

K : int
DIMS : sint
OBJECTS : collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES : collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

|VARIABLES| ≥ 1
|INTEGERS| ≥ 1
|POSITIVES| ≥ 1
required(VARIABLES, v)
|VARIABLES| = K
required(INTEGERS, v)
|INTEGERS| = K
required(POSITIVES, v)
|POSITIVES| = K
POSITIVES.v > 0
K > 0
DIMS ≥ 0
DIMS < K
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
OBJECTS.oid ≥ 1
OBJECTS.oid $\leq$ |OBJECTS|
OBJECTS.sid ≥ 1
OBJECTS.sid $\leq$ |SBOXES|
|SBOXES| ≥ 1
required(SBOXES, [sid, t, l])
SBOXES.sid ≥ 1
SBOXES.sid $\leq$ |SBOXES|
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i \neq j$, O_i and O_j coincide exactly with respect to a set of dimensions depicted by DIMS. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a K-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id sid, shift offset t, and sizes l. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier oid, shape id sid and origin x.
Two objects O_i and object O_j are *equal* with respect to a set of dimensions depicted by DIMS if and only if, for all shifted box s_i associated with O_i there exists a shifted box s_j such that, for all dimensions $d \in$ DIMS, (1) the origins of s_i and s_j coincide and, (2) the ends of s_i and s_j also coincide.

Example

$$\left(\begin{array}{l}2,\{0,1\},\\ \left\langle\begin{array}{lll}\texttt{oid}-1 & \texttt{sid}-2 & \texttt{x}-\langle 4,1\rangle,\\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,1\rangle,\\ \texttt{oid}-3 & \texttt{sid}-2 & \texttt{x}-\langle 4,1\rangle\end{array}\right\rangle,\\ \left\langle\begin{array}{lll}\texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle,\\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle,\\ \texttt{sid}-2 & \texttt{t}-\langle 1,0\rangle & \texttt{l}-\langle 1,3\rangle,\\ \texttt{sid}-2 & \texttt{t}-\langle 0,2\rangle & \texttt{l}-\langle 1,1\rangle,\\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,1\rangle,\\ \texttt{sid}-3 & \texttt{t}-\langle 0,1\rangle & \texttt{l}-\langle 1,1\rangle,\\ \texttt{sid}-3 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,1\rangle,\\ \texttt{sid}-4 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle\end{array}\right\rangle\end{array}\right)$$

Figure 5.350 shows the objects of the example. Since these objects coincide exactly the EQUAL_SBOXES constraint holds.

Typical

$|\texttt{OBJECTS}| > 1$

Symmetries

- Items of OBJECTS are permutable.
- Items of SBOXES are permutable.
- Items of OBJECTS.x, SBOXES.t and SBOXES.l are permutable (*same permutation used*).

Arg. properties

Suffix-contractible wrt. OBJECTS.

Remark

One of the eight relations of the *Region Connection Calculus* [349]. The constraint EQUAL_SBOXES is a restriction of the original relation since it requires to have exactly the same partition between the different objects.

See also

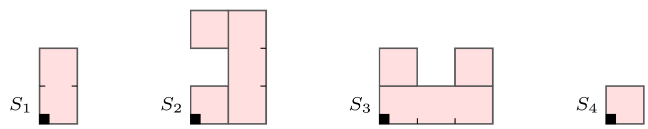
common keyword: CONTAINS_SBOXES, COVEREDBY_SBOXES, COVERS_SBOXES, DISJOINT_SBOXES, INSIDE_SBOXES, MEET_SBOXES *(rcc8)*, NON_OVERLAP_SBOXES *(geometrical constraint,logic)*, OVERLAP_SBOXES *(rcc8)*.

Keywords

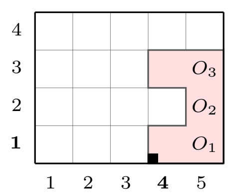
constraint type: logic.

geometry: geometrical constraint, rcc8.

miscellaneous: obscure.



(A) Possible shapes S_1, S_2, S_3 and S_4



OBJECTS

O_1:	oid $-$ 1	sid $-$ 2	x $-$ $\langle 4, 1\rangle$
O_2:	oid $-$ 2	sid $-$ 2	x $-$ $\langle 4, 1\rangle$
O_3:	oid $-$ 3	sid $-$ 2	x $-$ $\langle 4, 1\rangle$

(B) Three objects which exactly coincide

Figure 5.350: (B) The three mutually coinciding objects O_1, O_2, O_3 of the **Example** slot respectively assigned shape S_2; (A) shapes S_1, S_2, S_3 and S_4 are respectively made up from 1, 3, 3 and 1 disjoint shifted box.

Logic

- $\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D})$
- $\texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D}) + \texttt{S1.l}(\texttt{D})$
- $\texttt{equal\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=}$
  $\forall \texttt{D} \in \texttt{Dims}$
  $\bigwedge \left(\begin{array}{l} \texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) = \\ \texttt{origin}(\texttt{O2},\texttt{S2},\texttt{D}) \\ \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) = \\ \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) \end{array} , \right)$
- $\texttt{equal\_objects}(\texttt{Dims},\texttt{O1},\texttt{O2}) \stackrel{\text{def}}{=}$
  $\forall \texttt{S1} \in \texttt{sboxes}([\texttt{O1.sid}])$
  $\exists \texttt{S2} \in \texttt{sboxes}\left(\left[\texttt{O2.sid} \right] \right)$
  $\texttt{equal\_sboxes}\left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{S1}, \\ \texttt{O2}, \\ \texttt{S2} \end{array} \right)$
- $\texttt{all\_equal}(\texttt{Dims},\texttt{OIDS}) \stackrel{\text{def}}{=}$
  $\forall \texttt{O1} \in \texttt{objects}(\texttt{OIDS})$
  $\forall \texttt{O2} \in \texttt{objects}(\texttt{OIDS})$
  $\begin{array}{l} \texttt{O1.oid} = \\ \texttt{O2.oid} - 1 \end{array} \Rightarrow$
  $\texttt{equal\_objects}\left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{O2} \end{array} \right)$
- $\texttt{all\_equal}(\texttt{DIMENSIONS},\texttt{OIDS})$

5.155 EQUILIBRIUM

▶ ▷ ◁ ◀

DESCRIPTION LINKS

Origin Inspired by the Irish Collegiate Programming Competition 2012 (equilibrium index)

Constraint

$$\text{EQUILIBRIUM}\left(\begin{array}{l}\texttt{VARIABLES},\\ \texttt{INDEX1},\\ \texttt{INDEX2},\\ \texttt{EPSILON},\\ \texttt{COEF1},\\ \texttt{COEF2},\\ \texttt{TOLERANCE},\\ \texttt{CTR}\end{array}\right)$$

Synonym BALANCED.

Arguments

VARIABLES	:	collection(var−dvar)
INDEX1	:	dvar
INDEX2	:	dvar
EPSILON	:	int
COEF1	:	int
COEF2	:	int
TOLERANCE	:	int
CTR	:	atom

Restrictions

$|\mathtt{VARIABLES}| \geq 1$
$\mathtt{INDEX1} \geq 1$
$\mathtt{INDEX1} \leq |\mathtt{VARIABLES}|$
$\mathtt{INDEX2} \geq 1$
$\mathtt{INDEX2} \leq |\mathtt{VARIABLES}|$
$\mathtt{INDEX1} \leq \mathtt{INDEX2}$
$\mathtt{EPSILON} \geq 0$
$\mathtt{EPSILON} \leq 2$
$\mathtt{EPSILON} = \mathtt{INDEX2} - \mathtt{INDEX1}$
$\mathtt{COEF1} \neq 0$
$\mathtt{COEF2} \neq 0$
$\mathtt{TOLERANCE} \geq 0$

$$\mathtt{CTR} \in \left[\begin{array}{l} \text{AMONG\_DIFF\_0}, \\ \text{AND}, \\ \text{CHANGE}, \\ \text{DEEPEST\_VALLEY}, \\ \text{HIGHEST\_PEAK}, \\ \text{INCREASING\_NVALUE}, \\ \text{INFLEXION}, \\ \text{LONGEST\_CHANGE}, \\ \text{LONGEST\_DECREASING\_SEQUENCE}, \\ \text{LONGEST\_INCREASING\_SEQUENCE}, \\ \text{MAX\_DECREASING\_SLOPE}, \\ \text{MAX\_INCREASING\_SLOPE}, \\ \text{MIN\_DECREASING\_SLOPE}, \\ \text{MIN\_INCREASING\_SLOPE}, \\ \text{MIN\_WIDTH\_PEAK}, \\ \text{MIN\_WIDTH\_VALLEY}, \\ \text{PEAK}, \\ \text{SUM\_CTR}, \\ \text{VALLEY} \end{array}\right]$$

Given $\texttt{VARIABLES} = \langle \texttt{VAR}_1, \texttt{VAR}_2, \ldots, \texttt{VAR}_{|\texttt{VARIABLES}|}\rangle$, enforce the following conditions:

- $\texttt{INDEX1} \geq 1$
- $\texttt{INDEX2} \geq 1$
- $\texttt{EPSILON} \geq 0$
- $\texttt{INDEX1} \leq \texttt{INDEX2}$
- $\texttt{COEF1} \neq 0$
- $\texttt{INDEX1} \leq |\texttt{VARIABLES}|$
- $\texttt{INDEX2} \leq |\texttt{VARIABLES}|$
- $\texttt{EPSILON} \leq 2$
- $\texttt{INDEX2} - \texttt{INDEX1} = \texttt{EPSILON}$
- $\texttt{TOLERANCE} \geq 0$
- $\texttt{COEF2} \neq 0$

Purpose

$$\begin{cases} \text{if } \texttt{CTR} = \text{CHANGE}: \\ \quad \text{CHANGE}(C_1, \langle \texttt{VAR}_1, \ldots, \texttt{VAR}_{\texttt{INDEX1}}\rangle, \neq) \\ \quad \text{CHANGE}(C_2, \langle \texttt{VAR}_{\texttt{INDEX2}}, \ldots, \texttt{VAR}_{|\texttt{VARIABLES}|}\rangle, \neq) \\ \text{if } \texttt{CTR} = \text{LONGEST\_CHANGE}: \\ \quad \text{LONGEST\_CHANGE}(C_1, \langle \texttt{VAR}_1, \ldots, \texttt{VAR}_{\texttt{INDEX1}}\rangle, \neq) \\ \quad \text{LONGEST\_CHANGE}(C_2, \langle \texttt{VAR}_{\texttt{INDEX2}}, \ldots, \texttt{VAR}_{|\texttt{VARIABLES}|}\rangle, \neq) \\ \text{if } \texttt{CTR} = \text{SUM\_CTR}: \\ \quad \text{SUM\_CTR}(\langle \texttt{VAR}_1, \ldots, \texttt{VAR}_{\texttt{INDEX1}}\rangle, =, C_1) \\ \quad \text{SUM\_CTR}(\langle \texttt{VAR}_{\texttt{INDEX2}}, \ldots, \texttt{VAR}_{|\texttt{VARIABLES}|}\rangle, =, C_2) \\ \text{otherwise}: \\ \quad \texttt{CTR}(C_1, \langle \texttt{VAR}_1, \ldots, \texttt{VAR}_{\texttt{INDEX1}}\rangle) \\ \quad \texttt{CTR}(C_2, \langle \texttt{VAR}_{\texttt{INDEX2}}, \ldots, \texttt{VAR}_{|\texttt{VARIABLES}|}\rangle) \end{cases}$$

$$|\texttt{COEF1} \cdot C_1 - \texttt{COEF2} \cdot C_2| \leq \texttt{TOLERANCE}$$

Example

$(\langle 4,4,3,6,2\rangle, 2, 4, 2, 1, 1, 0, \text{SUM\_CTR})$
$(\langle -2,5,-2,6,-1,0,-3,5,-7,6,-1,7,0\rangle, 5, 5, 0, 1, 1, 0, \text{SUM\_CTR})$
$(\langle -2,5,-2,6,-1,0,-3,5,-7,6,-1,7,0\rangle, 11, 11, 0, 1, 1, 0, \text{SUM\_CTR})$
$(\langle 0,3,2,6,2,2,5,8,7,6,7,3\rangle, 5, 7, 2, 1, 1, 0, \text{PEAK})$
$(\langle 0,5,3,8,2,2,5,5,8,7,2,7,3\rangle, 7, 7, 0, 1, 1, 0, \text{CHANGE})$

The first example, EQUILIBRIUM$(\langle 4_1, 4_{\mathbf{2}}, 3_3, 6_{\mathbf{4}}, 2_5\rangle, \mathbf{2}, \mathbf{4}, \mathbf{2}, \mathbf{1}, \mathbf{1}, \mathbf{0},$ SUM_CTR$)$, holds since:

- INDEX1 $= \mathbf{2} \geq 1$,
- INDEX2 $= \mathbf{4} \geq 1$,
- EPSILON $= \mathbf{2} \geq 0$,
- INDEX1 $= \mathbf{2} \leq$ INDEX2 $= \mathbf{4}$,
- $C_1 = 4_1 + 4_{\mathbf{2}} = \mathbf{8}$,
- INDEX2 $-$ INDEX1 $=$ EPSILON $= \mathbf{2}$,
- INDEX1 $= \mathbf{2} \leq |$VARIABLES$| = 5$,
- INDEX2 $= \mathbf{4} \leq |$VARIABLES$| = 5$,
- EPSILON $= \mathbf{2} \leq 2$,
- TOLERANCE $= \mathbf{0} \geq 0$,
- $C_2 = 6_{\mathbf{4}} + 2_5 = \mathbf{8}$,
- $|\mathbf{1} \cdot \mathbf{8} - \mathbf{1} \cdot \mathbf{8}| \leq$ TOLERANCE $= \mathbf{0}$.

$|\mathbf{1} \cdot \mathbf{8} - \mathbf{1} \cdot \mathbf{8}| \leq$ TOLERANCE $= \mathbf{0}$

		4_1	$\boxed{4_{\mathbf{2}}}$	3_3	6_4	2_5	VARIABLES
		4	**8**	11	17	15	sum on prefixes
19	15	11	**8**	2			sum on suffixes
4_1	4_2	3_3	$\boxed{6_{\mathbf{4}}}$	2_5			VARIABLES

EPSILON $= \mathbf{2}$

Figure 5.351: Illustration of the first example of the **Example** slot

The second example, EQUILIBRIUM$(\langle -2_1, 5_2, -2_3, 6_4, -1_{\mathbf{5}}, 0_6, -3_7, 5_8, -7_9, 6_{10}, -1_{11}, 7_{12}, 0_{13}\rangle,$ $\mathbf{5}, \mathbf{5}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{0},$ SUM_CTR$)$, holds since:

- INDEX1 $= \mathbf{5} \geq 1$,
- INDEX2 $= \mathbf{5} \geq 1$,
- EPSILON $= \mathbf{0} \geq 0$,
- INDEX1 $= \mathbf{5} \leq$ INDEX2 $= \mathbf{5}$,
- $C_1 = -2_1 + 5_2 - 2_3 + 6_4 - 1_{\mathbf{5}} = \mathbf{6}$,
- INDEX2 $-$ INDEX1 $=$ EPSILON $= \mathbf{0}$,
- INDEX1 $= \mathbf{5} \leq |$VARIABLES$| = 13$,
- INDEX2 $= \mathbf{5} \leq |$VARIABLES$| = 13$,
- EPSILON $= \mathbf{0} \leq 2$,
- TOLERANCE $= \mathbf{0} \geq 0$,
- $C_2 = -1_{\mathbf{5}} + 0_6 - 3_7 + 5_8 - 7_9 + 6_{10} - 1_{11} + 7_{12} + 0_{13} = \mathbf{6}$,
- $|\mathbf{1} \cdot \mathbf{6} - \mathbf{1} \cdot \mathbf{6}| \leq$ TOLERANCE $= \mathbf{0}$.

$|\mathbf{1} \cdot \mathbf{6} - \mathbf{1} \cdot \mathbf{6}| \leq$ TOLERANCE $= \mathbf{0}$ $|\mathbf{1} \cdot \mathbf{6} - \mathbf{1} \cdot \mathbf{6}| \leq$ TOLERANCE $= \mathbf{0}$

-2_1	5_2	-2_3	6_4	$\boxed{-1_{\mathbf{5}}}$	0_6	-3_7	5_8	-7_9	6_{10}	$\boxed{-1_{\mathbf{11}}}$	7_{12}	0_{13}	VARIABLES
-2	3	1	7	**6**	6	3	8	1	7	**6**	13	13	sum on prefixes
13	15	10	12	**6**	7	7	10	5	12	**6**	7	0	sum on suffixes
-2_1	5_2	-2_3	6_4	$\boxed{-1_{\mathbf{5}}}$	0_6	-3_7	5_8	-7_9	6_{10}	$\boxed{-1_{\mathbf{11}}}$	7_{12}	0_{13}	VARIABLES

EPSILON $= \mathbf{0}$ EPSILON $= \mathbf{0}$

Figure 5.352: Illustration of the second and third examples of the **Example** slot

The third example, EQUILIBRIUM$(\langle -2_1, 5_2, -2_3, 6_4, -1_5, 0_6, -3_7, 5_8, -7_9, 6_{10}, -1_{\mathbf{11}}, 7_{12}, 0_{13}\rangle,$ $\mathbf{11}, \mathbf{11}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{0},$ SUM_CTR$)$, holds since:

- INDEX1 $= \mathbf{11} \geq 1$,
- INDEX2 $= \mathbf{11} \geq 1$,
- EPSILON $= \mathbf{0} \geq 0$,
- INDEX1 $= \mathbf{11} \leq$ INDEX2 $= \mathbf{11}$,
- $C_1 = -2_1 + 5_2 - 2_3 + 6_4 - 1_5 + 0_6 - 3_7 + 5_8 - 7_9 + 6_{10} - 1_{\mathbf{11}} = \mathbf{6}$,

- INDEX2 $-$ INDEX1 $=$ EPSILON $= \mathbf{0}$,
- INDEX1 $= \mathbf{11} \leq |$VARIABLES$| = 13$,
- INDEX2 $= \mathbf{11} \leq |$VARIABLES$| = 13$,
- EPSILON $= \mathbf{0} \leq 2$,
- TOLERANCE $= 0 \geq 0$,
- $C_2 = -1_{\mathbf{11}} + 7_{12} + 0_{13} = \mathbf{6}$,
- $|\mathbf{1} \cdot \mathbf{6} - \mathbf{1} \cdot \mathbf{6}| \leq$ TOLERANCE $= \mathbf{0}$.

The fourth example, EQUILIBRIUM$(\langle 0_1, 3_2, 2_3, 6_4, 2_{\mathbf{5}}, 2_6, 5_{\mathbf{7}}, 8_8, 7_9, 6_{10}, 7_{11}, 3_{12}\rangle$, $\mathbf{5}, \mathbf{7}, \mathbf{2}, \mathbf{1}, \mathbf{1}, \mathbf{0}$, PEAK), holds since:

- INDEX1 $= \mathbf{5} \geq 1$,
- INDEX2 $= \mathbf{7} \geq 1$,
- EPSILON $= \mathbf{2} \geq 0$,
- INDEX1 $= \mathbf{5} \leq$ INDEX2 $= \mathbf{7}$,
- the sequence $0_1\ 3_2\ 2_3\ 6_4\ 2_{\mathbf{5}}$ contains **2** peaks,
- INDEX2 $-$ INDEX1 $=$ EPSILON $= \mathbf{2}$,
- INDEX1 $= 5 \leq |$VARIABLES$| = 12$,
- INDEX2 $= 7 \leq |$VARIABLES$| = 12$,
- EPSILON $= \mathbf{2} \leq 2$,
- TOLERANCE $= 0 \geq 0$,
- The sequence $5_{\mathbf{7}}\ 8_8\ 7_9\ 6_{10}\ 7_{11}\ 3_{12}$ contains **2** peaks,
- $|\mathbf{1} \cdot \mathbf{2} - \mathbf{1} \cdot \mathbf{2}| \leq$ TOLERANCE $= \mathbf{0}$.

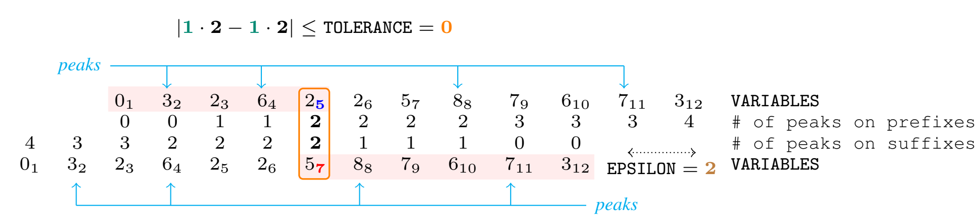


Figure 5.353: Illustration of the fourth example of the **Example** slot

The fifth example, EQUILIBRIUM$(\langle 0_1, 5_2, 3_3, 8_4, 2_5, 2_6, 5_{\mathbf{7}}, 5_8, 8_9, 7_{10}, 2_{11}, 7_{12}, 3_{13}\rangle$, $\mathbf{7}, \mathbf{7}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{0}$, CHANGE), holds since:

- INDEX1 $= \mathbf{7} \geq 1$,
- INDEX2 $= \mathbf{7} \geq 1$,
- EPSILON $= \mathbf{0} \geq 0$,
- INDEX1 $= \mathbf{7} \leq$ INDEX2 $= \mathbf{7}$,
- the sequence $0_1, 5_2, 3_3, 8_4, 2_5, 2_6, 5_{\mathbf{7}}$ contains **5** changes,
- INDEX2 $-$ INDEX1 $=$ EPSILON $= \mathbf{0}$,
- INDEX1 $= \mathbf{7} \leq |$VARIABLES$| = 12$,
- INDEX2 $= \mathbf{7} \leq |$VARIABLES$| = 12$,
- EPSILON $= \mathbf{0} \leq 2$,
- TOLERANCE $= 0 \geq 0$,
- The sequence $5_{\mathbf{7}}, 5_8, 8_9, 7_{10}, 2_{11}, 7_{12}, 3_{13}$ contains **5** changes,
- $|\mathbf{1} \cdot \mathbf{5} - \mathbf{1} \cdot \mathbf{5}| \leq$ TOLERANCE $= \mathbf{0}$.

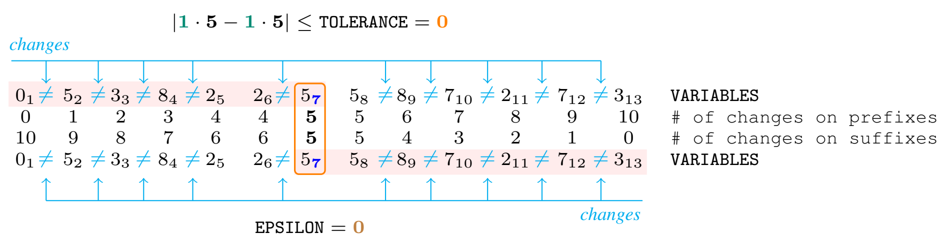


Figure 5.354: Illustration of the fifth example of the **Example** slot

Typical

$|\texttt{VARIABLES}| > 2$
$\texttt{INDEX1} > 1$
$\texttt{INDEX1} < |\texttt{VARIABLES}|$
$\texttt{INDEX2} > 1$
$\texttt{INDEX2} < |\texttt{VARIABLES}|$
$\texttt{COEF1} = 1$
$\texttt{COEF2} = 1$
$\texttt{EPSILON} = 1$
$\texttt{TOLERANCE} = 0$

See also **root concept:** BALANCE.

Keywords **characteristic of a constraint:** automaton with counters.

constraint type: predefined constraint.

5.156 EQUIVALENT

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint EQUIVALENT(VAR, VARIABLES)

Synonym EQ.

Arguments

VAR	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

$\texttt{VAR} \geq 0$
$\texttt{VAR} \leq 1$
$|\texttt{VARIABLES}| = 2$
required(VARIABLES, var)
$\texttt{VARIABLES.var} \geq 0$
$\texttt{VARIABLES.var} \leq 1$

Purpose Let VARIABLES be a collection of 0-1 variables $\texttt{VAR}_1, \texttt{VAR}_2$. Enforce $\texttt{VAR} = (\texttt{VAR}_1 \Leftrightarrow \texttt{VAR}_2)$.

Example

$(1, \langle 0, 0\rangle)$
$(0, \langle 0, 1\rangle)$
$(0, \langle 1, 0\rangle)$
$(1, \langle 1, 1\rangle)$

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of 0 in VAR and in VARIABLES.var can be set to 1.

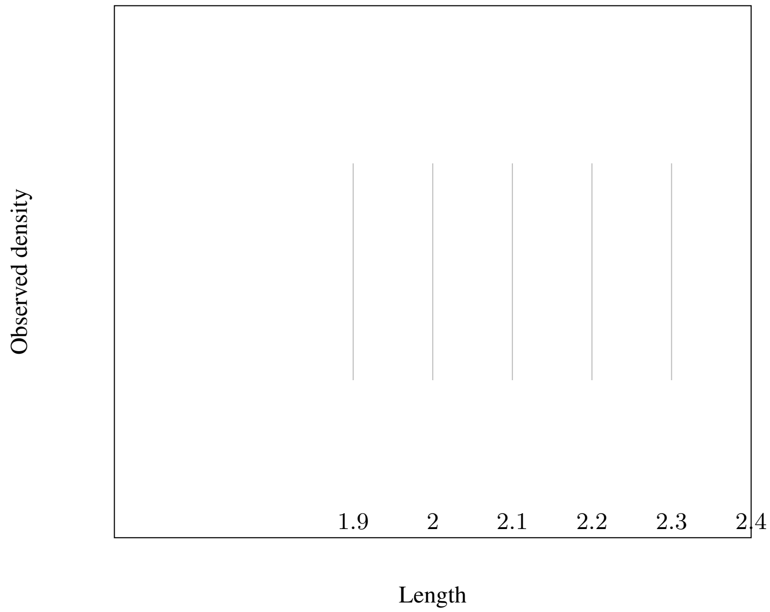
Arg. properties Functional dependency: VAR determined by VARIABLES.

Counting

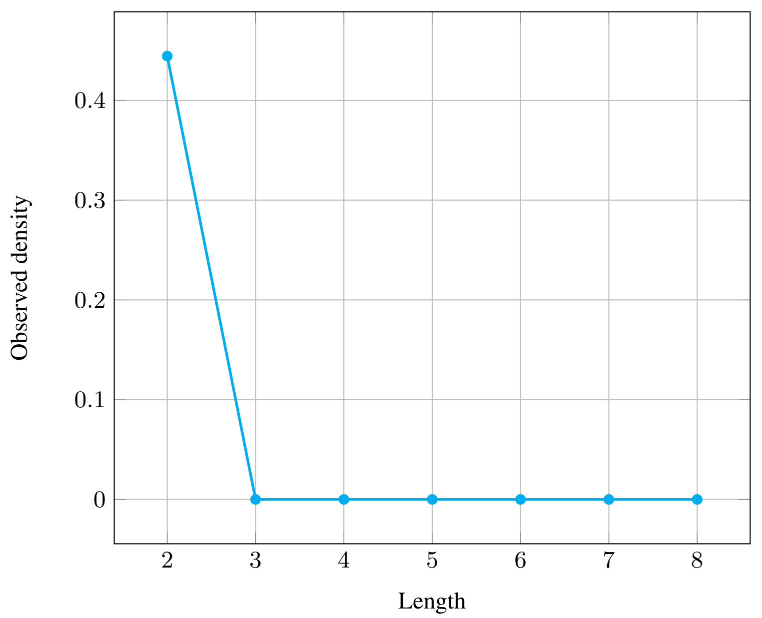
Length (n)	2	3	4	5	6	7	8
Solutions	4	0	0	0	0	0	0

Number of solutions for EQUIVALENT: domains $0..n$

Solution density for EQUIVALENT



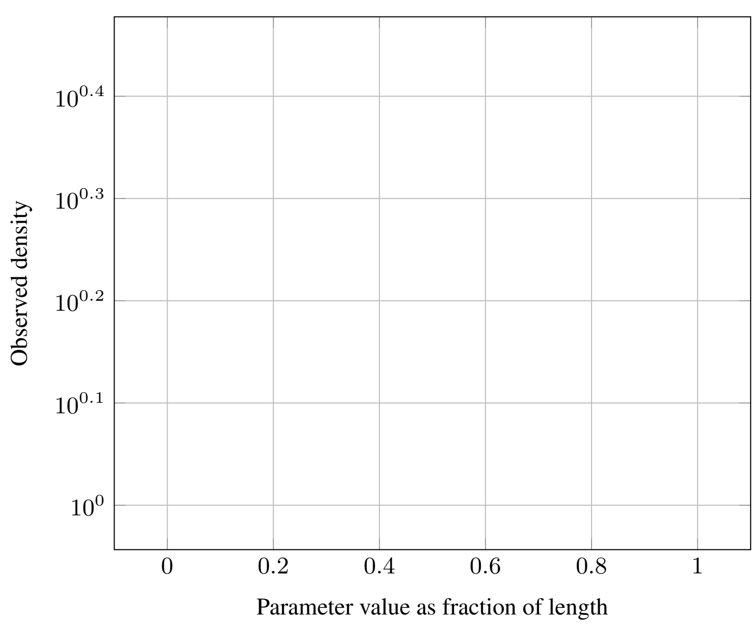
Solution density for EQUIVALENT

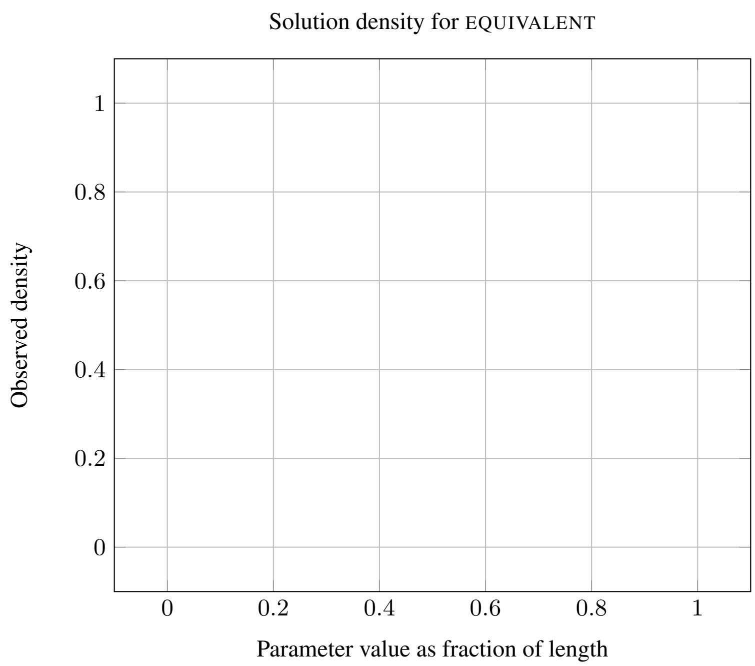


Length (n)		2
Total		4
Parameter value	0	2
	1	2

Solution count for EQUIVALENT: domains $0..n$

Solution density for EQUIVALENT





Systems IFONLYIF in **Choco**, REL in **Gecode**, EQBOOL in **JaCoP**, #<=> in **SICStus**.

See also **common keyword:** AND, IMPLY, NAND, NOR, OR, XOR *(Boolean constraint)*.

implies: ATLEAST_NVALUE, SOFT_ALL_EQUAL_MIN_CTR, SOFT_ALLDIFFERENT_CTR.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: functional dependency.

Automaton Figure 5.355 depicts the automaton associated with the EQUIVALENT constraint. To the first argument VAR of the EQUIVALENT constraint corresponds the first signature variable. To each variable $\mathtt{VAR}_i$ of the second argument VARIABLES of the EQUIVALENT constraint corresponds the next signature variable. There is no signature constraint.

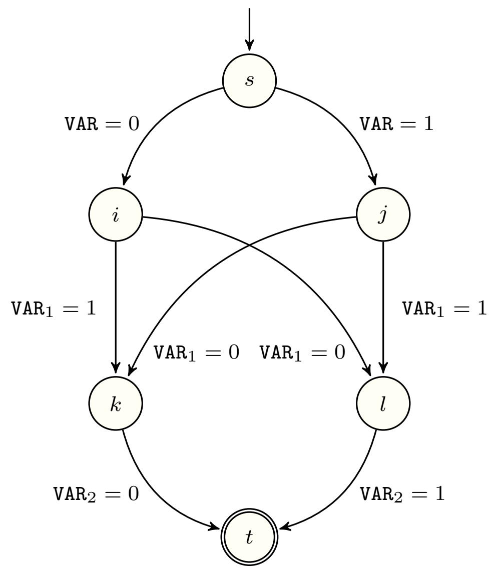


Figure 5.355: Automaton of the EQUIVALENT constraint

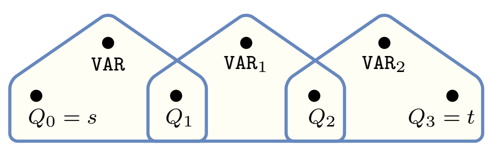


Figure 5.356: Hypergraph of the reformulation corresponding to the automaton of the EQUIVALENT constraint

5.157 EXACTLY

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin

Derived from ATLEAST and ATMOST.

Constraint

EXACTLY(N, VARIABLES, VALUE)

Synonym

COUNT.

Arguments

N	:	int
VARIABLES	:	collection(var−dvar)
VALUE	:	int

Restrictions

$\mathtt{N} \geq 0$
$\mathtt{N} \leq |\mathtt{VARIABLES}|$
required(VARIABLES, var)

Purpose

Exactly N variables of the VARIABLES collection are assigned value VALUE.

Example

$(2, \langle 4, 2, 4, 5\rangle, 4)$

The EXACTLY constraint holds since exactly $\mathtt{N} = 2$ variables of the $\mathtt{VARIABLES} = \langle 4, 2, 4, 5\rangle$ collection are assigned value $\mathtt{VALUE} = 4$.

Typical

$\mathtt{N} > 0$
$\mathtt{N} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that is different from VALUE can be replaced by any other value that is also different from VALUE.

Arg. properties

- Functional dependency: N determined by VARIABLES and VALUE.
- Aggregate: N(+), VARIABLES(union), VALUE(id).

Systems

OCCURENCE in **Choco**, COUNT in **Gecode**, EXACTLY in **Gecode**, COUNT in **JaCoP**, EXACTLY in **MiniZinc**, COUNT in **SICStus**.

See also

generalisation: AMONG *(*constant *replaced by* variable *and* value *replaced by* list *of* values*)*.

implies: ATLEAST *(*$=$ N *replaced by* $\geq$ N*)*, ATMOST *(*$=$ N *replaced by* $\leq$ N*)*.

Keywords

characteristic of a constraint: automaton, automaton with counters.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: value constraint, counting constraint.

filtering: glue matrix, arc-consistency.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$SELF \mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUE
Graph property(ies)	**NARC** = N

Graph model Since each arc constraint involves only one vertex (VALUE is fixed), we employ the $SELF$ arc generator in order to produce a graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.357 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold. The EXACTLY constraint holds since exactly two variables are assigned value 4.

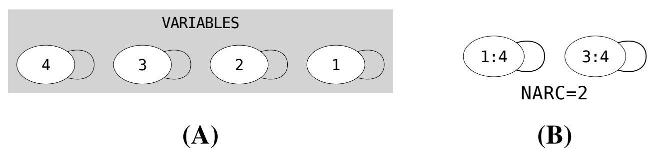


Figure 5.357: Initial and final graph of the EXACTLY constraint

Automaton

Figure 5.358 depicts the automaton associated with the EXACTLY constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i = \mathtt{VALUE} \Leftrightarrow S_i$.

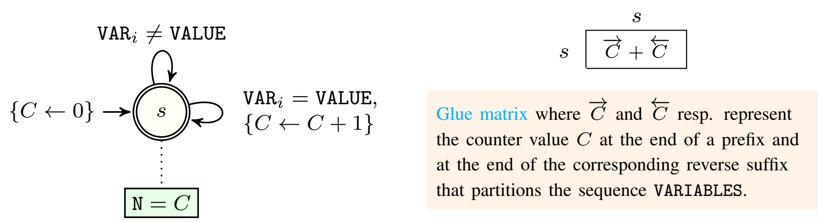


Figure 5.358: Automaton (with one counter) of the EXACTLY constraint and its glue matrix

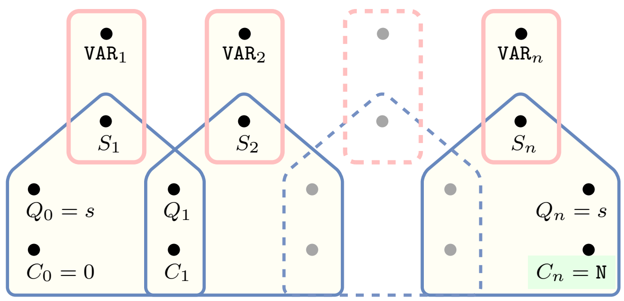


Figure 5.359: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the EXACTLY constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.158 FIRST_VALUE_DIFF_0

DESCRIPTION LINKS AUTOMATON

Origin Paparazzi puzzle

Constraint FIRST_VALUE_DIFF_0(VAR, VARIABLES)

Synonyms FIRST_VALUE_DIFF_FROM_0, FIRST_VALUE_DIFFERENT_FROM_0.

Arguments

VAR	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

VAR $\neq 0$
$|$VARIABLES$| \geq 1$
required(VARIABLES, var)

Purpose VAR is equal to the first non-zero variable of the collection VARIABLES.

Example

$(8, \langle 0, 0, 8, 0, 5\rangle)$
$(4, \langle 4, 0, 8, 0, 5\rangle)$

Typical

$|$VARIABLES$| > 1$
minval(VARIABLES.var) $< 0 \vee$ maxval(VARIABLES.var) > 1
$|$VARIABLES$|-$AMONG_DIFF_0(VARIABLES.var) ≥ 1
$\bigvee \left(\begin{array}{l} |\text{VARIABLES}| \leq 4, \\ |\text{VARIABLES}| - \text{AMONG\_DIFF\_0}(\text{VARIABLES.var}) > 1 \end{array} \right)$

Typical model

nval(VARIABLES.var) > 2
ATLEAST(2, VARIABLES, 0)

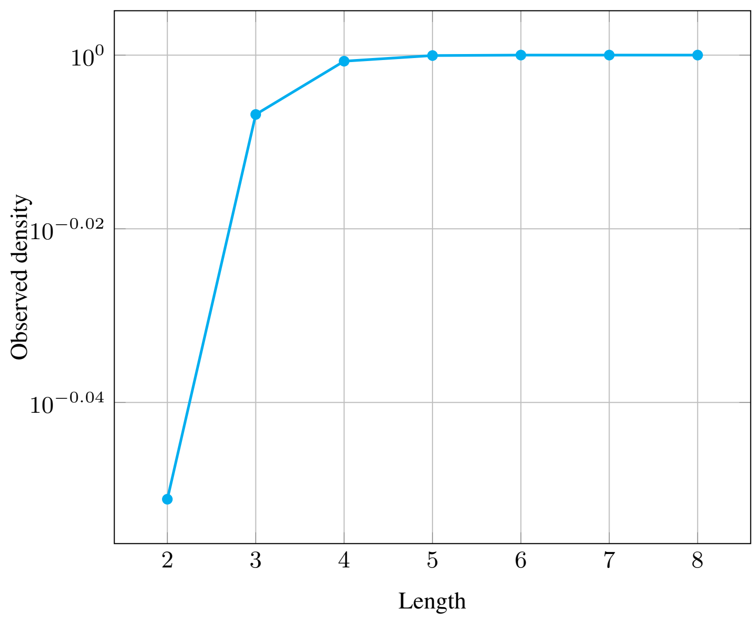
Arg. properties Functional dependency: VAR determined by VARIABLES.

Counting

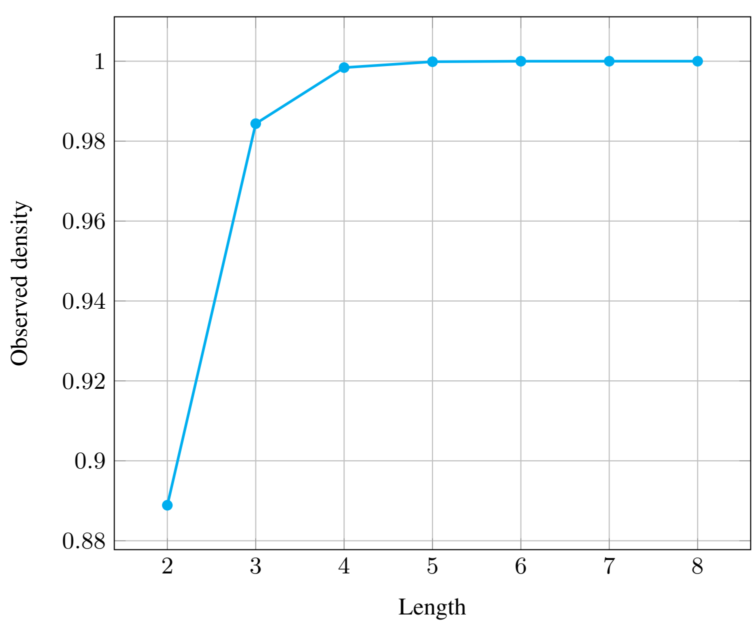
Length (n)	2	3	4	5	6	7	8
Solutions	8	63	624	7775	117648	2097151	43046720

Number of solutions for FIRST_VALUE_DIFF_0: domains $0..n$

Solution density for FIRST_VALUE_DIFF_0

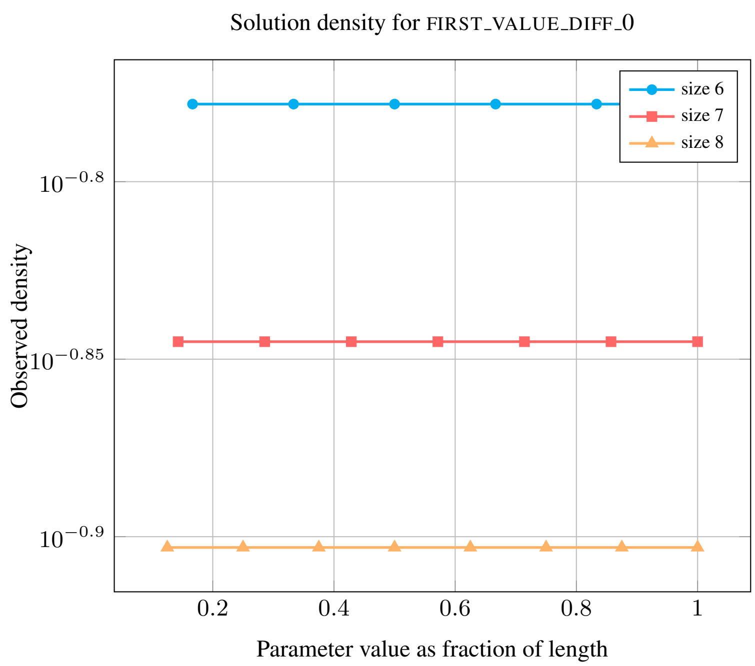


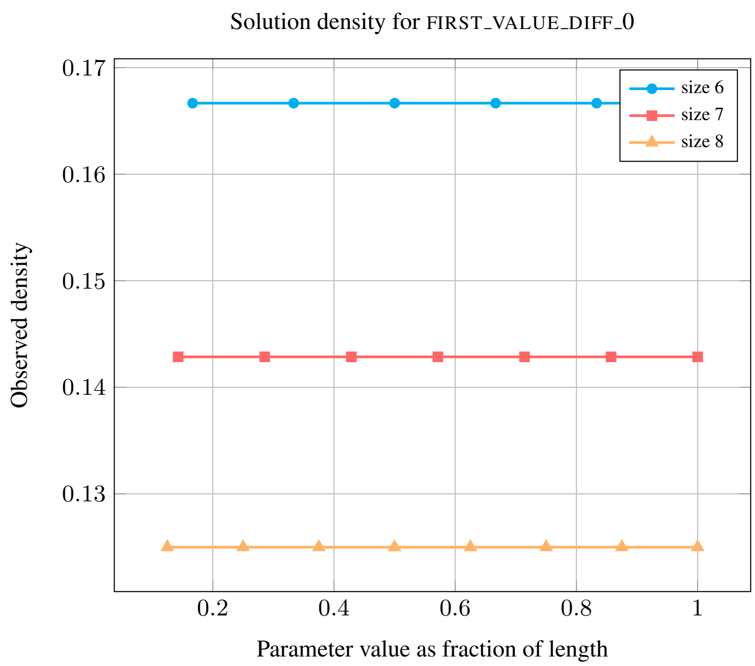
Solution density for FIRST_VALUE_DIFF_0



Length (n)		2	3	4	5	6	7	8
Total		8	63	624	7775	117648	2097151	43046720
Parameter value	1	4	21	156	1555	19608	299593	5380840
	2	4	21	156	1555	19608	299593	5380840
	3	-	21	156	1555	19608	299593	5380840
	4	-	-	156	1555	19608	299593	5380840
	5	-	-	-	1555	19608	299593	5380840
	6	-	-	-	-	19608	299593	5380840
	7	-	-	-	-	-	299593	5380840
	8	-	-	-	-	-	-	5380840

Solution count for FIRST_VALUE_DIFF_0: domains $0..n$





See also **implies:** BETWEEN_MIN_MAX.

Keywords **characteristic of a constraint:** joker value, automaton, automaton with counters.

modelling: functional dependency.

Automaton Figure 5.360 depicts an automaton that only accepts all the solutions to the FIRST_VALUE_DIFF_0 constraint. This automaton uses a counter in order to record the value of the first non-zero variable $\mathtt{VAR}_i$ already encountered. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i \neq 0 \Leftrightarrow S_i$.

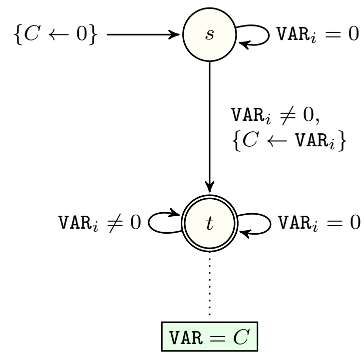


Figure 5.360: Automaton (with one counter) of the FIRST_VALUE_DIFF_0 constraint

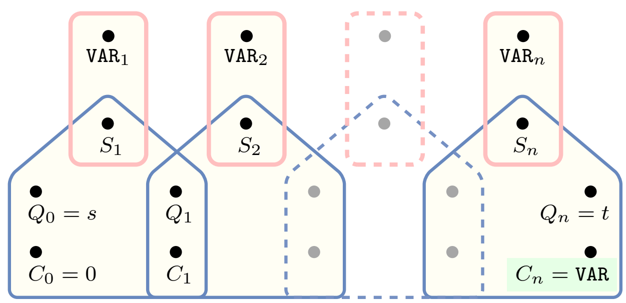


Figure 5.361: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the FIRST_VALUE_DIFF_0 constraint

5.159 FULL_GROUP

DESCRIPTION LINKS AUTOMATON

Origin Inspired by GROUP

Constraint

$$\text{FULL\_GROUP}\left(\begin{array}{l}\texttt{NGROUP},\\ \texttt{MIN\_SIZE},\\ \texttt{MAX\_SIZE},\\ \texttt{MIN\_DIST},\\ \texttt{MAX\_DIST},\\ \texttt{NVAL},\\ \texttt{VARIABLES},\\ \texttt{VALUES}\end{array}\right)$$

Synonym GROUP_WITHOUT_BORDER.

Arguments

NGROUP	:	dvar
MIN_SIZE	:	dvar
MAX_SIZE	:	dvar
MIN_DIST	:	dvar
MAX_DIST	:	dvar
NVAL	:	dvar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int)

Restrictions

$\texttt{NGROUP} \geq 0$
$\texttt{MIN\_SIZE} \geq 0$
$\texttt{MAX\_SIZE} \geq \texttt{MIN\_SIZE}$
$\texttt{MIN\_DIST} \geq 0$
$\texttt{MAX\_DIST} \geq \texttt{MIN\_DIST}$
$\texttt{MAX\_DIST} \leq |\texttt{VARIABLES}| - 2$
$\texttt{NVAL} \geq \texttt{MAX\_SIZE}$
$\texttt{NVAL} \geq \texttt{NGROUP}$
$\texttt{NVAL} \leq |\texttt{VARIABLES}| - 2$
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose

Let n be the number of variables of the collection VARIABLES. Let $X_i, X_{i+1}, \ldots, X_j$ $(1 \leq i \leq j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that all the following conditions simultaneously apply:

- All variables $X_i, \ldots, X_j$ take their values in the set of values VALUES,
- $i = 1$ or X_{i-1} does not take a value in VALUES,
- $j = n$ or X_{j+1} does not take a value in VALUES.

We call such a sequence of variables a *group*. A *full group* is a group that neither starts at position 1 nor ends at position n. Similarly an *anti-full group* is a maximum sequence of variables that are not assigned any value from VALUES that neither starts at position 1 nor ends at position n.
The constraint FULL_GROUP is true if all the following conditions hold:

- There are exactly NGROUP full groups of variables,
- MIN_SIZE is the number of variables of the smallest full group,
- MAX_SIZE is the number of variables of the largest full group,
- MIN_DIST is the number of variables of the smallest anti-full group,
- MAX_DIST is the number of variables of the largest anti-full group,
- NVAL is the number of variables that belong to a full group.

Example

$(2, 2, 3, 1, 1, 5, \langle 0, 1, 2, 6, 2, 7, 4, 8, 9\rangle, \langle 0, 2, 4, 6, 8\rangle)$

Given the fact that full groups are formed by even values in $\{0, 2, 4, 6, 8\}$ (i.e., values expressed by the VALUES collection), the FULL_GROUP constraint holds since:

- Its first argument, NGROUP, is set to value 2 since the sequence 0 1 2 6 2 7 4 8 9 contains two full groups of even values (i.e., group 2 6 2 and group 4 8). Note that the first 0 is not a full group since it is located at the first position of the sequence.
- Its second argument, MIN_SIZE, is set to value 2 since the smallest full group of even values involves only two elements (i.e., the full group 4 8).
- Its third argument, MAX_SIZE, is set to value 3 since the largest full group of even values involves three elements (i.e., the full group 2 6 2).
- Its fourth argument, MIN_DIST, is set to value 1 since the smallest anti-full groups involve a single element (i.e., the anti-full groups 1 and 7).
- Its fifth argument, MAX_DIST, is set to value 1 since the largest anti-full groups involve a single element (i.e., the anti-full groups 1 and 7).
- Its sixth argument, NVAL, is set to value 5 since the total number of even values part of a full group of the sequence 0 1 2 6 2 7 4 8 9 is equal to 5 (i.e., elements 2, 6, 2, 4 and 8).

Typical

NGROUP > 0
MIN_SIZE > 0
MAX_SIZE $>$ MIN_SIZE
MIN_DIST > 0
MAX_DIST $>$ MIN_DIST
MAX_DIST $<$ |VARIABLES|
NVAL $>$ MAX_SIZE
NVAL $>$ NGROUP
NVAL $<$ |VARIABLES|
|VARIABLES| > 1
range(VARIABLES.var) > 1
|VALUES| > 0
|VARIABLES| $>$ |VALUES|

Symmetries

- Items of VARIABLES can be reversed.
- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).

Arg. properties

- Functional dependency: NGROUP determined by VARIABLES and VALUES.
- Functional dependency: MIN_SIZE determined by VARIABLES and VALUES.
- Functional dependency: MAX_SIZE determined by VARIABLES and VALUES.
- Functional dependency: MIN_DIST determined by VARIABLES and VALUES.
- Functional dependency: MAX_DIST determined by VARIABLES and VALUES.
- Functional dependency: NVAL determined by VARIABLES and VALUES.

See also

common keyword: GROUP *(timetabling constraint,sequence)*.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

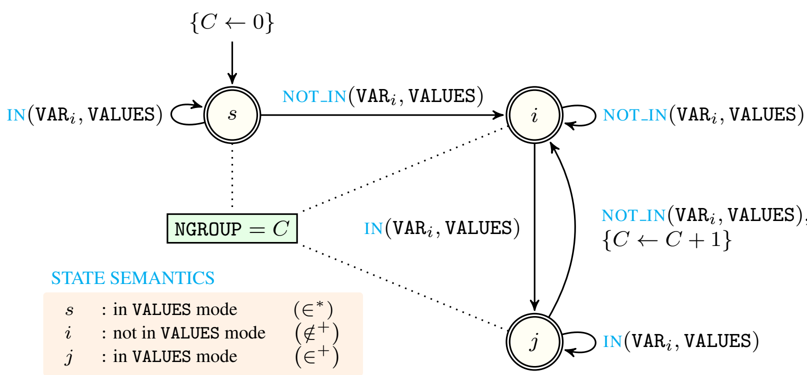
constraint network structure: alpha-acyclic constraint network(2), alpha-acyclic constraint network(3).

constraint type: timetabling constraint.

filtering: glue matrix.

modelling: functional dependency.

Automaton Figures 5.362, 5.364, 5.366, 5.368, 5.370 and 5.372 depict the different automata associated with the FULL_GROUP constraint. For the automata that respectively compute NGROUP, MIN_SIZE, MAX_SIZE, MIN_DIST, MAX_DIST and NVAL we have a 0-1 signature variable $\mathtt{S}_i$ for each variable $\mathtt{VAR}_i$ of the collection VARIABLES. The following signature constraint links $\mathtt{VAR}_i$ and $\mathtt{S}_i$: $\mathtt{VAR}_i \in \mathtt{VALUES} \Leftrightarrow \mathtt{S}_i$.



Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	$s\ (\in^*)$	$i\ (\notin^+)$	$j\ (\in^+)$
$s(\in^*)$	0	$\overleftarrow{C}$	$\overleftarrow{C}$
$i(\notin^+)$	$\overrightarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+1+\overleftarrow{C}$
$j(\in^+)$	$\overrightarrow{C}$	$\overrightarrow{C}+1+\overleftarrow{C}$	$\overrightarrow{C}+1+\overleftarrow{C}$

Figure 5.362: Automaton for the NGROUP argument of the FULL_GROUP constraint and its glue matrix

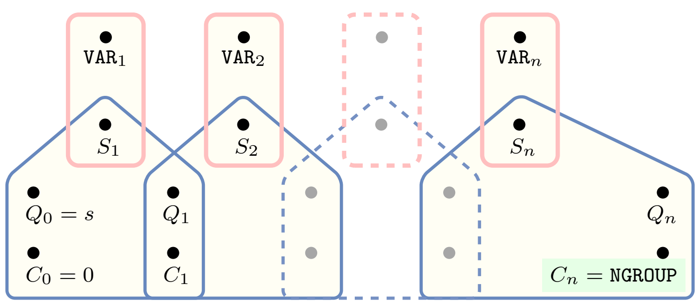


Figure 5.363: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the NGROUP argument of the FULL_GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

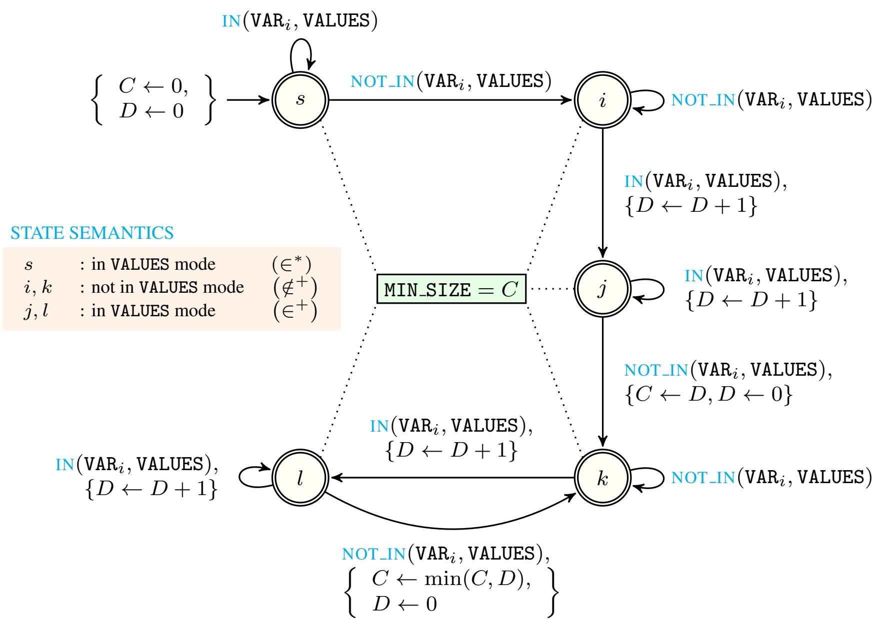


Figure 5.364: Automaton for the MIN_SIZE argument of the FULL_GROUP constraint

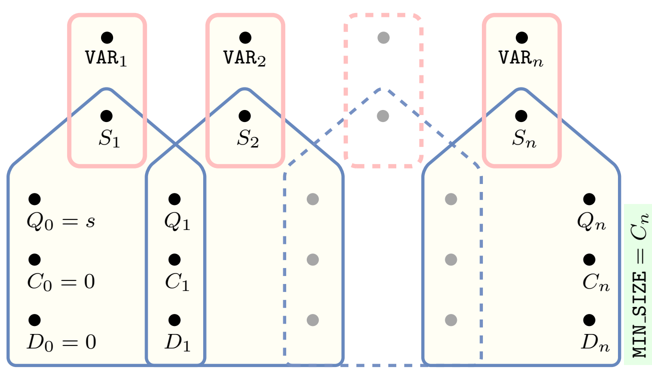
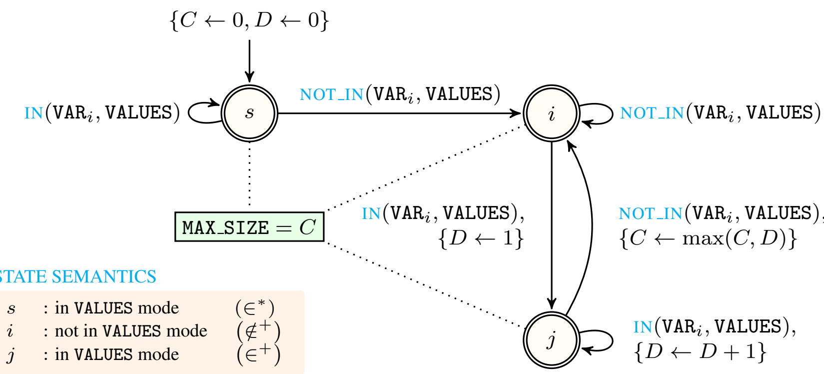


Figure 5.365: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MIN_SIZE argument of the FULL_GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counter value C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	$s\ (\in^*)$	$i\ (\notin^+)$	$j\ (\in^+)$
$s(\in^*)$	0	$\overleftarrow{C}$	$\overleftarrow{C}$
$i(\notin^+)$	$\overrightarrow{C}$	$\max(\overrightarrow{C}, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overleftarrow{D}, \overleftarrow{C})$
$j(\in^+)$	$\overrightarrow{C}$	$\max(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$

Figure 5.366: Automaton for the MAX_SIZE argument of the FULL_GROUP constraint and its glue matrix

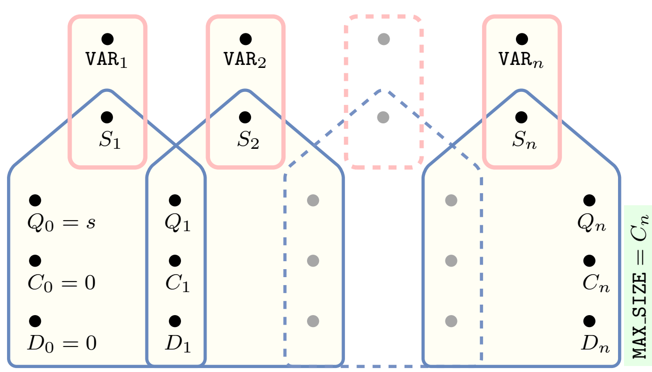


Figure 5.367: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MAX_SIZE argument of the FULL_GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

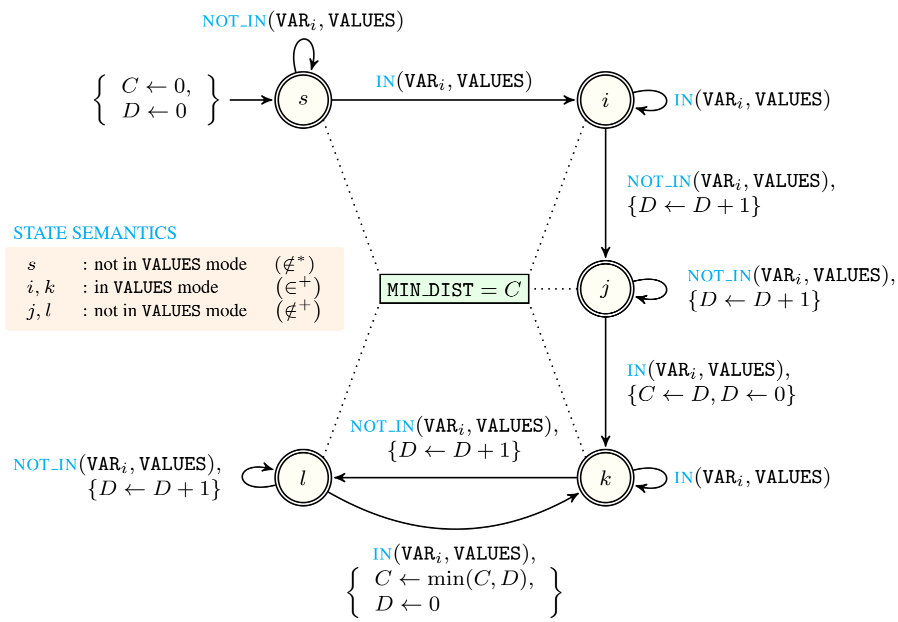


Figure 5.368: Automaton for the MIN_DIST argument of the FULL_GROUP constraint

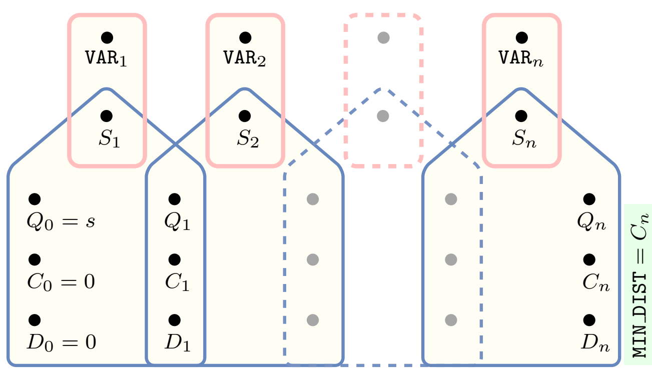
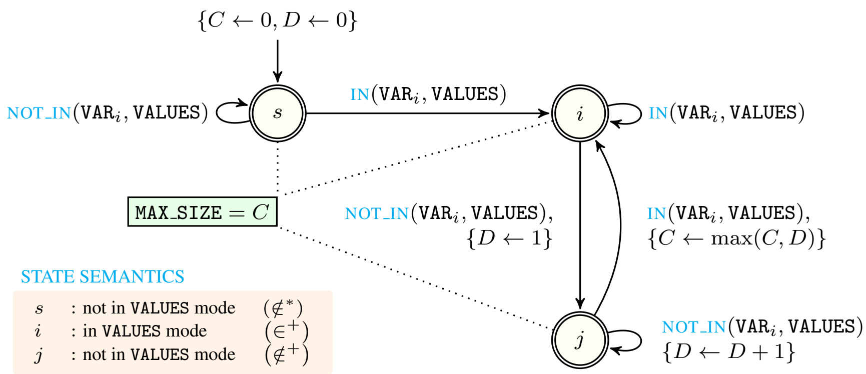


Figure 5.369: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MIN_DIST argument of the FULL_GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counter value C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	$s\,(\notin^*)$	$i\,(\in^+)$	$j\,(\notin^+)$
$s(\notin^*)$	0	$\overleftarrow{C}$	$\overleftarrow{C}$
$i(\in^+)$	$\overrightarrow{C}$	$\max(\overrightarrow{C}, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overleftarrow{D}, \overleftarrow{C})$
$j(\notin^+)$	$\overrightarrow{C}$	$\max(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$

Figure 5.370: Automaton for the MAX_DIST argument of the FULL_GROUP constraint and its glue matrix

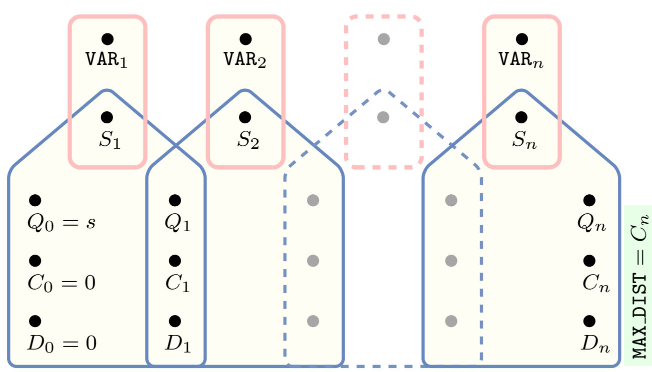
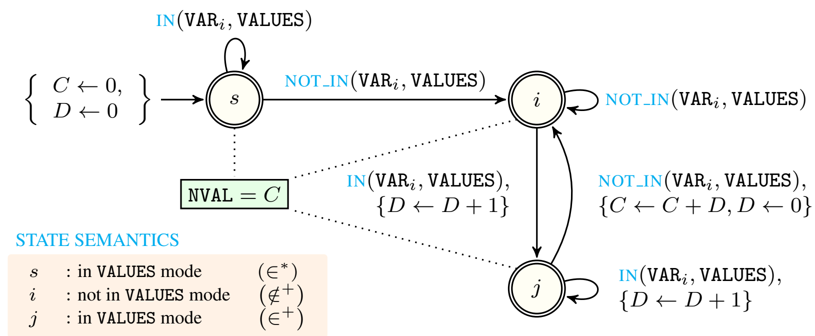


Figure 5.371: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MAX_DIST argument of the FULL_GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)



Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	$s\ (\in^*)$	$i\ (\notin^+)$	$j\ (\in^+)$
$s(\in^*)$	0	$\overleftarrow{C}$	$\overleftarrow{C}$
$i(\notin^+)$	$\overrightarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{D}+\overleftarrow{C}$
$j(\in^+)$	$\overrightarrow{C}$	$\overrightarrow{C}+\overrightarrow{D}+\overleftarrow{C}$	$\overrightarrow{C}+\overrightarrow{D}+\overleftarrow{D}+\overleftarrow{C}$

Figure 5.372: Automaton for the NVAL argument of the FULL_GROUP constraint and its glue matrix

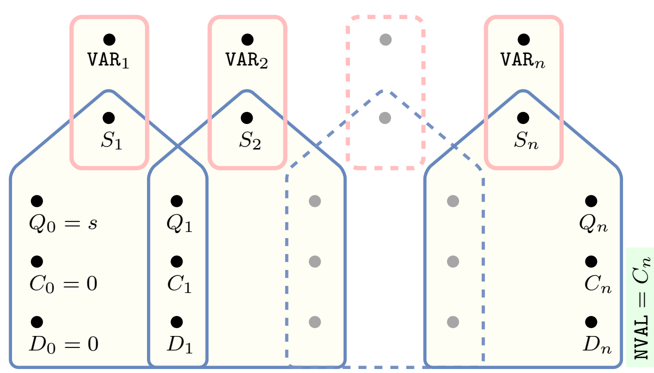


Figure 5.373: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the NVAL argument of the FULL_GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

5.160 GCD

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin [146]

Constraint GCD(X, Y, Z)

Arguments

X : dvar
Y : dvar
Z : dvar

Restrictions

$X > 0$
$Y > 0$
$Z > 0$

Purpose Enforce the fact that Z is the greatest common divisor of X and Y.

Example $(24, 60, 12)$

The GCD constraint holds since 12 is the greatest common divisor of 24 and 60.

Typical

$X > 1$
$Y > 1$

Symmetry Arguments are permutable w.r.t. permutation (X, Y) (Z).

Arg. properties Functional dependency: X determined by Y and Z.

Algorithm In [146] a filtering algorithm for the GCD constraint was automatically derived from the Euclidian algorithm by using constructive disjunction and abstract interpretation in order to approximate the behaviour of the while loop of the Euclidian algorithm.

See also **common keyword:** POWER *(abstract interpretation)*.

Keywords **constraint arguments:** ternary constraint, pure functional dependency.

constraint type: arithmetic constraint, predefined constraint.

filtering: abstract interpretation.

modelling: functional dependency.

5.161 GEOST

DESCRIPTION LINKS

Origin

Generalisation of DIFFN.

Constraint

GEOST(K, OBJECTS, SBOXES)

Types

VARIABLES : collection(v−dvar)
INTEGERS : collection(v−int)
POSITIVES : collection(v−int)

Arguments

K : int
OBJECTS : collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES : collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

$|\text{VARIABLES}| \geq 1$
$|\text{INTEGERS}| \geq 1$
$|\text{POSITIVES}| \geq 1$
required(VARIABLES, v)
$|\text{VARIABLES}| = \text{K}$
required(INTEGERS, v)
$|\text{INTEGERS}| = \text{K}$
required(POSITIVES, v)
$|\text{POSITIVES}| = \text{K}$
$\text{POSITIVES.v} > 0$
$\text{K} > 0$
required(OBJECTS, [oid, sid, x])
distinct(OBJECTS, oid)
$\text{OBJECTS.oid} \geq 1$
$\text{OBJECTS.oid} \leq |\text{OBJECTS}|$
$\text{OBJECTS.sid} \geq 1$
$\text{OBJECTS.sid} \leq |\text{SBOXES}|$
$|\text{SBOXES}| \geq 1$
required(SBOXES, [sid, t, l])
$\text{SBOXES.sid} \geq 1$
$\text{SBOXES.sid} \leq |\text{SBOXES}|$
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i < j$, O_i and O_j do not overlap with respect to a set of dimensions $\{1, 2, \ldots, \mathtt{K}\}$. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a K-dimensional space at a given offset (from the origin of the shape) with given sizes that are all strictly greater than 0. More precisely, a *shifted box* is an entity defined by its shape id sid, shift offset t, and sizes l. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier oid, shape id sid and origin x.
An object O_i *does not overlap* an object O_j with respect to the set of dimensions $\{1, 2, \ldots, \mathtt{K}\}$ if and only if for all shifted box s_i associated with O_i and for all shifted box s_j associated with O_j there exists a dimension $d \in \{1, 2, \ldots, \mathtt{K}\}$ such that the start of s_i in dimension d is greater than or equal to the end of s_j in dimension d, or the start of s_j in dimension d is greater than or equal to the end of s_i in dimension d.

Example

$$\left(\begin{array}{l} 2, \left\langle \begin{array}{lll} \mathtt{oid}-1 & \mathtt{sid}-1 & \mathtt{x}-\langle 1,2\rangle, \\ \mathtt{oid}-2 & \mathtt{sid}-5 & \mathtt{x}-\langle 2,1\rangle, \\ \mathtt{oid}-3 & \mathtt{sid}-8 & \mathtt{x}-\langle 4,1\rangle \end{array}\right\rangle, \\ \left\langle \begin{array}{lll} \mathtt{sid}-1 & \mathtt{t}-\langle 0,0\rangle & \mathtt{l}-\langle 2,1\rangle, \\ \mathtt{sid}-1 & \mathtt{t}-\langle 0,1\rangle & \mathtt{l}-\langle 1,2\rangle, \\ \mathtt{sid}-1 & \mathtt{t}-\langle 1,2\rangle & \mathtt{l}-\langle 3,1\rangle, \\ \mathtt{sid}-2 & \mathtt{t}-\langle 0,0\rangle & \mathtt{l}-\langle 3,1\rangle, \\ \mathtt{sid}-2 & \mathtt{t}-\langle 0,1\rangle & \mathtt{l}-\langle 1,3\rangle, \\ \mathtt{sid}-2 & \mathtt{t}-\langle 2,1\rangle & \mathtt{l}-\langle 1,1\rangle, \\ \mathtt{sid}-3 & \mathtt{t}-\langle 0,0\rangle & \mathtt{l}-\langle 2,1\rangle, \\ \mathtt{sid}-3 & \mathtt{t}-\langle 1,1\rangle & \mathtt{l}-\langle 1,2\rangle, \\ \mathtt{sid}-3 & \mathtt{t}-\langle -2,2\rangle & \mathtt{l}-\langle 3,1\rangle, \\ \mathtt{sid}-4 & \mathtt{t}-\langle 0,0\rangle & \mathtt{l}-\langle 3,1\rangle, \\ \mathtt{sid}-4 & \mathtt{t}-\langle 0,1\rangle & \mathtt{l}-\langle 1,1\rangle, \\ \mathtt{sid}-4 & \mathtt{t}-\langle 2,1\rangle & \mathtt{l}-\langle 1,3\rangle, \\ \mathtt{sid}-5 & \mathtt{t}-\langle 0,0\rangle & \mathtt{l}-\langle 2,1\rangle, \\ \mathtt{sid}-5 & \mathtt{t}-\langle 1,1\rangle & \mathtt{l}-\langle 1,1\rangle, \\ \mathtt{sid}-5 & \mathtt{t}-\langle 0,2\rangle & \mathtt{l}-\langle 2,1\rangle, \\ \mathtt{sid}-6 & \mathtt{t}-\langle 0,0\rangle & \mathtt{l}-\langle 3,1\rangle, \\ \mathtt{sid}-6 & \mathtt{t}-\langle 0,1\rangle & \mathtt{l}-\langle 1,1\rangle, \\ \mathtt{sid}-6 & \mathtt{t}-\langle 2,1\rangle & \mathtt{l}-\langle 1,1\rangle, \\ \mathtt{sid}-7 & \mathtt{t}-\langle 0,0\rangle & \mathtt{l}-\langle 3,2\rangle, \\ \mathtt{sid}-8 & \mathtt{t}-\langle 0,0\rangle & \mathtt{l}-\langle 2,3\rangle \end{array}\right\rangle \end{array}\right)$$

Parts (A), (B) and (C) of Figure 5.374 respectively represent the potential shapes associated with the three objects of the example. Part (D) shows the position of the three objects of the example, where the first, second and third objects were respectively assigned shapes 1, 5 and 8. The coordinates of the leftmost lowest corner of each object are stressed in bold. The GEOST constraint holds since the three objects do not overlap (i.e., see part (D) if Figure 5.374).

Typical

$|\mathtt{OBJECTS}| > 1$

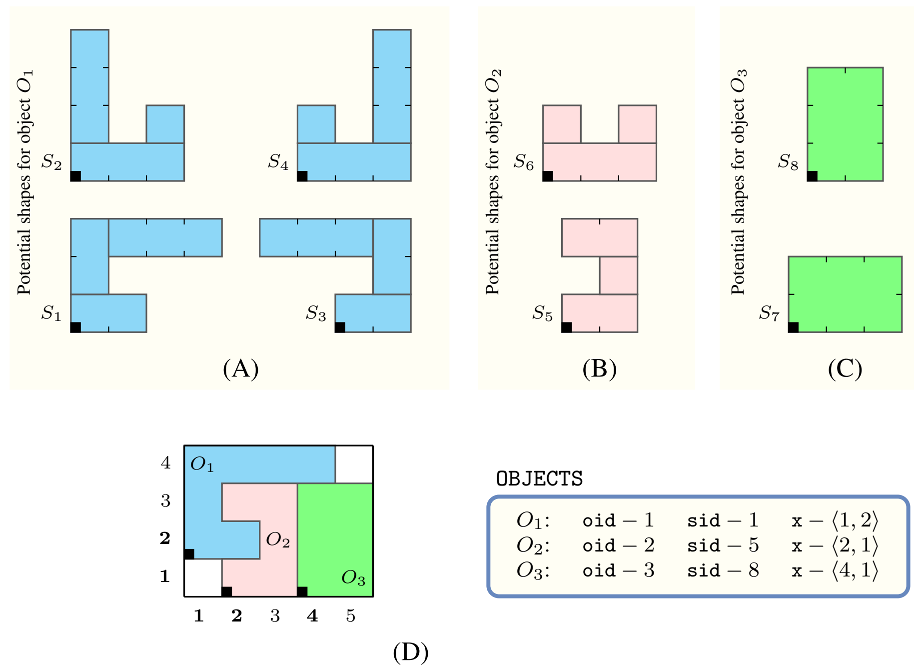


Figure 5.374: (D) The three non-overlapping objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_1, S_5, S_8; (A), (B), (C) shapes S_1, S_2, S_3, S_4, S_5, S_6, S_7 and S_8 are respectively made up from 3, 3, 3, 3, 3, 3, 1 and 1 disjoint shifted box.

Symmetries

- Items of `OBJECTS` are permutable.
- Items of `SBOXES` are permutable.
- Items of `OBJECTS.x`, `SBOXES.t` and `SBOXES.l` are permutable (*same permutation used*).
- `SBOXES.l.v` can be decreased to any value ≥ 1.

Usage

The GEOST constraint allows one to model directly a large number of placement problems.

Remark

In the two-dimensional case, when rectangles heights are all equal to one and when rectangles starts in the first dimension are all fixed, the GEOST constraint can be rewritten as a K_ALLDIFFERENT constraint corresponding to a system of ALLDIFFERENT constraints derived from the maximum cliques of the corresponding interval graph.

Algorithm

A sweep-based filtering algorithm for this constraint is described in [42]. Unlike previous sweep filtering algorithms which move a line for finding a feasible position for the origin of an object, this algorithm performs a recursive traversal of the multidimensional placement space. It explores all points of the domain of the origin of the object under focus, one by one, in increasing lexicographic order, until a point is found that is not infeasible for any non-overlapping constraints. To make the search efficient, instead of moving each time to the successor point, the search is arranged so that it skips points that are known to be infeasible for some non-overlapping constraint.

Within the context of breaking symmetries six different ways of integrating within GEOST a chain of lexicographical ordering constraints like LEX_CHAIN_LESS for enforcing a lexicographic ordering on the origin coordinates of identical objects, are described in [3].

Systems GEOST in **Choco**, GEOST in **JaCoP**, GEOST in **SICStus**.

See also **common keyword:** CALENDAR *(multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption)*, DIFFN *(geometrical constraint,non-overlapping)*, LEX_CHAIN_LESS, LEX_CHAIN_LESSEQ *(symmetry)*, NON_OVERLAP_SBOXES *(geometrical constraint,non-overlapping)*, VISIBLE *(geometrical constraint,sweep)*.

generalisation: GEOST_TIME *(*`temporal dimension` *added to* `geometrical dimensions`*)*.

specialisation: K_ALLDIFFERENT *(when rectangles heights are all equal to* 1 *and rectangles starts in the first dimension are all fixed)*, LEX_ALLDIFFERENT *(*`object` *replaced by* `vector`*)*.

Keywords **application area:** floor planning problem.

combinatorial object: pentomino.

constraint arguments: business rules.

constraint type: logic, decomposition, timetabling constraint, predefined constraint, relaxation.

filtering: sweep.

geometry: geometrical constraint, non-overlapping.

heuristics: heuristics for two-dimensional rectangle placement problems.

modelling: multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption, disjunction, assignment dimension, assigning and scheduling tasks that run in parallel, assignment to the same set of values, relaxation dimension.

modelling exercises: multi-site employee scheduling with calendar constraints, scheduling with machine choice, calendars and preemption, assigning and scheduling tasks that run in parallel, assignment to the same set of values, relaxation dimension.

problems: strip packing, two-dimensional orthogonal packing, pallet loading.

puzzles: squared squares, packing almost squares, Partridge, pentomino, Shikaku, smallest square for packing consecutive dominoes, smallest square for packing rectangles with distinct sizes, smallest rectangle area, Conway packing problem.

symmetry: symmetry.

5.162 GEOST_TIME

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Generalisation of DIFFN.

Constraint GEOST_TIME(K, DIMS, OBJECTS, SBOXES)

Types

$$\begin{array}{lll} \texttt{VARIABLES} & : & \texttt{collection}(\texttt{v}-\texttt{dvar}) \\ \texttt{INTEGERS} & : & \texttt{collection}(\texttt{v}-\texttt{int}) \\ \texttt{POSITIVES} & : & \texttt{collection}(\texttt{v}-\texttt{int}) \end{array}$$

Arguments

$$\begin{array}{lll} \texttt{K} & : & \texttt{int} \\ \texttt{DIMS} & : & \texttt{sint} \\ \texttt{OBJECTS} & : & \texttt{collection}\left(\begin{array}{l} \texttt{oid}-\texttt{int}, \\ \texttt{sid}-\texttt{dvar}, \\ \texttt{x}-\texttt{VARIABLES}, \\ \texttt{start}-\texttt{dvar}, \\ \texttt{duration}-\texttt{dvar}, \\ \texttt{end}-\texttt{dvar} \end{array}\right) \\ \texttt{SBOXES} & : & \texttt{collection}(\texttt{sid}-\texttt{int}, \texttt{t}-\texttt{INTEGERS}, \texttt{l}-\texttt{POSITIVES}) \end{array}$$

Restrictions

$|\texttt{VARIABLES}| \geq 1$
$|\texttt{INTEGERS}| \geq 1$
$|\texttt{POSITIVES}| \geq 1$
$\texttt{required}(\texttt{VARIABLES}, \texttt{v})$
$|\texttt{VARIABLES}| = \texttt{K}$
$\texttt{required}(\texttt{INTEGERS}, \texttt{v})$
$|\texttt{INTEGERS}| = \texttt{K}$
$\texttt{required}(\texttt{POSITIVES}, \texttt{v})$
$|\texttt{POSITIVES}| = \texttt{K}$
$\texttt{POSITIVES.v} > 0$
$\texttt{K} \geq 0$
$\texttt{DIMS} \geq 0$
$\texttt{DIMS} < \texttt{K}$
$\texttt{distinct}(\texttt{OBJECTS}, \texttt{oid})$
$\texttt{required}(\texttt{OBJECTS}, [\texttt{oid}, \texttt{sid}, \texttt{x}])$
$\texttt{require\_at\_least}(2, \texttt{OBJECTS}, [\texttt{start}, \texttt{duration}, \texttt{end}])$
$\texttt{OBJECTS.oid} \geq 1$
$\texttt{OBJECTS.oid} \leq |\texttt{OBJECTS}|$
$\texttt{OBJECTS.sid} \geq 1$
$\texttt{OBJECTS.sid} \leq |\texttt{SBOXES}|$
$\texttt{OBJECTS.duration} \geq 0$
$|\texttt{SBOXES}| \geq 1$
$\texttt{required}(\texttt{SBOXES}, [\texttt{sid}, \texttt{t}, \texttt{l}])$
$\texttt{SBOXES.sid} \geq 1$
$\texttt{SBOXES.sid} \leq |\texttt{SBOXES}|$
$\texttt{do\_not\_overlap}(\texttt{SBOXES})$

Purpose

Holds if (1) the difference between the end in time and the start in time of each object is equal to its duration in time, and if (2) for each pair of objects (O_i, O_j), $i < j$, O_i and O_j do not overlap with respect to a set of dimensions depicted by `DIMS` as well as to the time axis. Note that an object with duration zero can never overlap any other object. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a `K`-dimensional space at a given offset (from the origin of the shape) with given sizes that are all strictly greater than 0. More precisely, a *shifted box* is an entity defined by its shape id `sid`, shift offset `t`, and sizes `l`. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier `oid`, shape id `sid` and origin `x`.

An object O_i *does not overlap* an object O_j with respect to a set of dimensions depicted by `DIMS` as well as to the time axis if and only if:

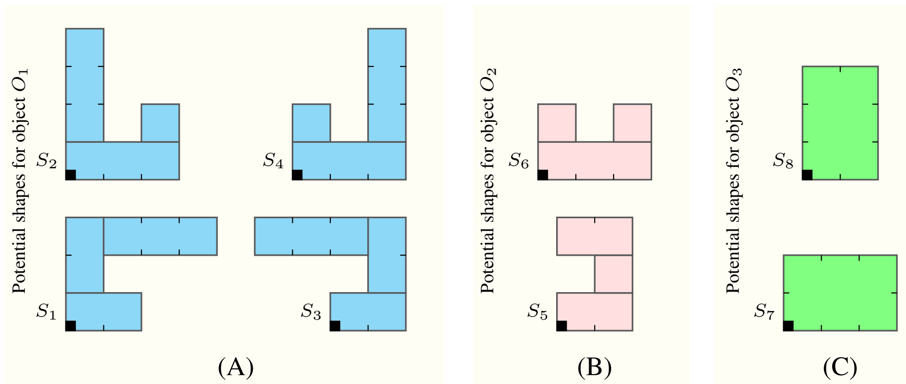
- The start in time of O_i is greater than or equal to the end in time of O_j.
- The start in time of O_j is greater than or equal to the end in time of O_i.
- For all shifted box s_i associated with O_i and for all shifted box s_j associated with O_j there exists a dimension $d \in$ `DIMS` such that the start of s_i in dimension d is greater than or equal to the end of s_j in dimension d, or the start of s_j in dimension d is greater than or equal to the end of s_i in dimension d.

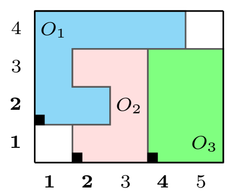
Example

$$\left(\begin{array}{l}
2, \{0,1\}, \\
\left\langle\begin{array}{llllll}
\texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{s}-0 & \texttt{d}-1 & \texttt{e}-1, \\
\texttt{oid}-2 & \texttt{sid}-5 & \texttt{x}-\langle 2,1\rangle & \texttt{s}-0 & \texttt{d}-1 & \texttt{e}-1, \\
\texttt{oid}-3 & \texttt{sid}-8 & \texttt{x}-\langle 4,1\rangle & \texttt{s}-0 & \texttt{d}-1 & \texttt{e}-1
\end{array}\right\rangle, \\
\left\langle\begin{array}{lll}
\texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,1\rangle, \\
\texttt{sid}-1 & \texttt{t}-\langle 0,1\rangle & \texttt{l}-\langle 1,2\rangle, \\
\texttt{sid}-1 & \texttt{t}-\langle 1,2\rangle & \texttt{l}-\langle 3,1\rangle, \\
\texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,1\rangle, \\
\texttt{sid}-2 & \texttt{t}-\langle 0,1\rangle & \texttt{l}-\langle 1,3\rangle, \\
\texttt{sid}-2 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,1\rangle, \\
\texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,1\rangle, \\
\texttt{sid}-3 & \texttt{t}-\langle 1,1\rangle & \texttt{l}-\langle 1,2\rangle, \\
\texttt{sid}-3 & \texttt{t}-\langle -2,2\rangle & \texttt{l}-\langle 3,1\rangle, \\
\texttt{sid}-4 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,1\rangle, \\
\texttt{sid}-4 & \texttt{t}-\langle 0,1\rangle & \texttt{l}-\langle 1,1\rangle, \\
\texttt{sid}-4 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,3\rangle, \\
\texttt{sid}-5 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,1\rangle, \\
\texttt{sid}-5 & \texttt{t}-\langle 1,1\rangle & \texttt{l}-\langle 1,1\rangle, \\
\texttt{sid}-5 & \texttt{t}-\langle 0,2\rangle & \texttt{l}-\langle 2,1\rangle, \\
\texttt{sid}-6 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,1\rangle, \\
\texttt{sid}-6 & \texttt{t}-\langle 0,1\rangle & \texttt{l}-\langle 1,1\rangle, \\
\texttt{sid}-6 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,1\rangle, \\
\texttt{sid}-7 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,2\rangle, \\
\texttt{sid}-8 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle
\end{array}\right\rangle
\end{array}\right)$$

(`s` for `start`, `d` for `duration`, `e` for `end`)

Parts (A), (B) and (C) of Figure 5.375 respectively represent the potential shapes associated with the three objects of the example. Part (D) shows the position of the three objects of the example, where the first, second and third objects were respectively assigned shapes 1, 5 and 8. The coordinates of the leftmost lowest corner of each object are stressed in bold. The GEOST_TIME constraint holds since the three objects do not overlap: even though the time intervals associated with each object overlap (i.e., they are in fact identical), their corresponding shapes do not overlap (i.e., see part (D) if Figure 5.375).





OBJECTS

O_1:	oid − 1	sid − 1	x − $\langle 1, 2\rangle$
O_2:	oid − 2	sid − 5	x − $\langle 2, 1\rangle$
O_3:	oid − 3	sid − 8	x − $\langle 4, 1\rangle$

(D)

Figure 5.375: (D) The three non-overlapping objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_1, S_5, S_8; (A), (B), (C) shapes S_1, S_2, S_3, S_4, S_5, S_6, S_7 and S_8 are respectively made up from 3, 3, 3, 3, 3, 3, 1 and 1 disjoint shifted box.

Typical

$|\texttt{OBJECTS}| > 1$

Symmetries

- Items of OBJECTS are permutable.
- Items of SBOXES are permutable.
- Items of OBJECTS.x, SBOXES.t and SBOXES.l are permutable (*same permutation used*).
- SBOXES.l.v can be decreased to any value ≥ 1.
- One and the same constant can be added to the start and end attributes of all items of OBJECTS.

Usage

The GEOST_TIME constraint allows one to model directly a large number of placement problems. Figure 5.376 sketches ten typical use of the GEOST_TIME constraint:

- The first case (A) corresponds to a non-overlapping constraint among three segments (or three tasks in disjunction).
- The second, third and fourth cases (B,C,D) correspond to a non-overlapping constraint between rectangles where (B) and (C) are special cases where the sizes of all rectangles in the second dimension are equal to 1; this can be interpreted as a *machine assignment problem* where each rectangle corresponds to a non-pre-emptive task that has to be placed in time and assigned to a specific machine so that no two tasks assigned to the same machine overlap in time. In Part (B) the duration of each task is fixed, while in Part (C) the duration depends on the machine to which the task is actually assigned. This dependence can be expressed by the ELEMENT constraint, which specifies the dependence between the shape variable and the assignment variable of each task.
- The fifth case (E) corresponds to a non-overlapping constraint between rectangles where each rectangle can have two orientations. This is achieved by associating with each rectangle two shapes of respective sizes $l \cdot h$ and $h \cdot l$. Since their orientations is not initially fixed, an ELEMENT_LESSEQ constraint can be used for enforcing the three rectangles to be included within the bounding box defined by the origin's coordinates $1, 1$ and sizes $8, 3$.
- The sixth case (F) corresponds to a non-overlapping constraint between more complex objects where each object is described by a given set of rectangles.
- The seventh case (G) describes a rectangle placement problem where one has to first assign each rectangle to a strip so that all rectangles that are assigned to the same strip do not overlap.
- The eighth case (H) corresponds to a non-overlapping constraint between parallelepipeds.
- The ninth case (I) can be interpreted as a non-overlapping constraint between parallelepipeds that are assigned to the same container. The first dimension corresponds to the identifier of the container, while the next three dimensions are associated with the position of a parallelepiped inside a container.
- Finally the tenth case (J) describes a rectangle placement problem over three consecutive time-slots: rectangles assigned to the same time-slot should not overlap in time. We initially start with the three rectangles 1, 2 and 3. Rectangle 3 is no more present at instant 2 (the arrow $\downarrow$ within rectangle 3 at time 1 indicates that rectangle 3 will disappear at the next time-point), while rectangle 4 appears at instant 2 (the arrow $\uparrow$ within rectangle 4 at time 2 denotes the fact that the rectangle 4 appears at instant 2). Finally rectangle 2 disappears at instant 3 and is replaced by rectangle 5.

Algorithm A sweep-based filtering algorithm for this constraint is described in [42]. Unlike previous sweep filtering algorithms which move a line for finding a feasible position for the origin of an object, this algorithm performs a recursive traversal of the multidimensional placement space. It explores all points of the domain of the origin of the object under focus, one by one, in increasing lexicographic order, until a point is found that is not infeasible for any non-overlapping constraints. To make the search efficient, instead of moving each time to the successor point, the search is arranged so that it skips points that are known to be infeasible for some non-overlapping constraint.

Systems GEOST in **Choco**, GEOST in **JaCoP**.

See also

common keyword: DIFFN, NON_OVERLAP_SBOXES *(geometrical constraint,non-overlapping)*, VISIBLE *(geometrical constraint,sweep)*.

specialisation: GEOST *(*`temporal dimension` *removed)*.

Keywords

constraint type: decomposition, timetabling constraint, predefined constraint.

filtering: sweep.

geometry: geometrical constraint, non-overlapping.

modelling: assignment dimension, assignment to the same set of values, assigning and scheduling tasks that run in parallel, disjunction.

modelling exercises: assignment to the same set of values, assigning and scheduling tasks that run in parallel.

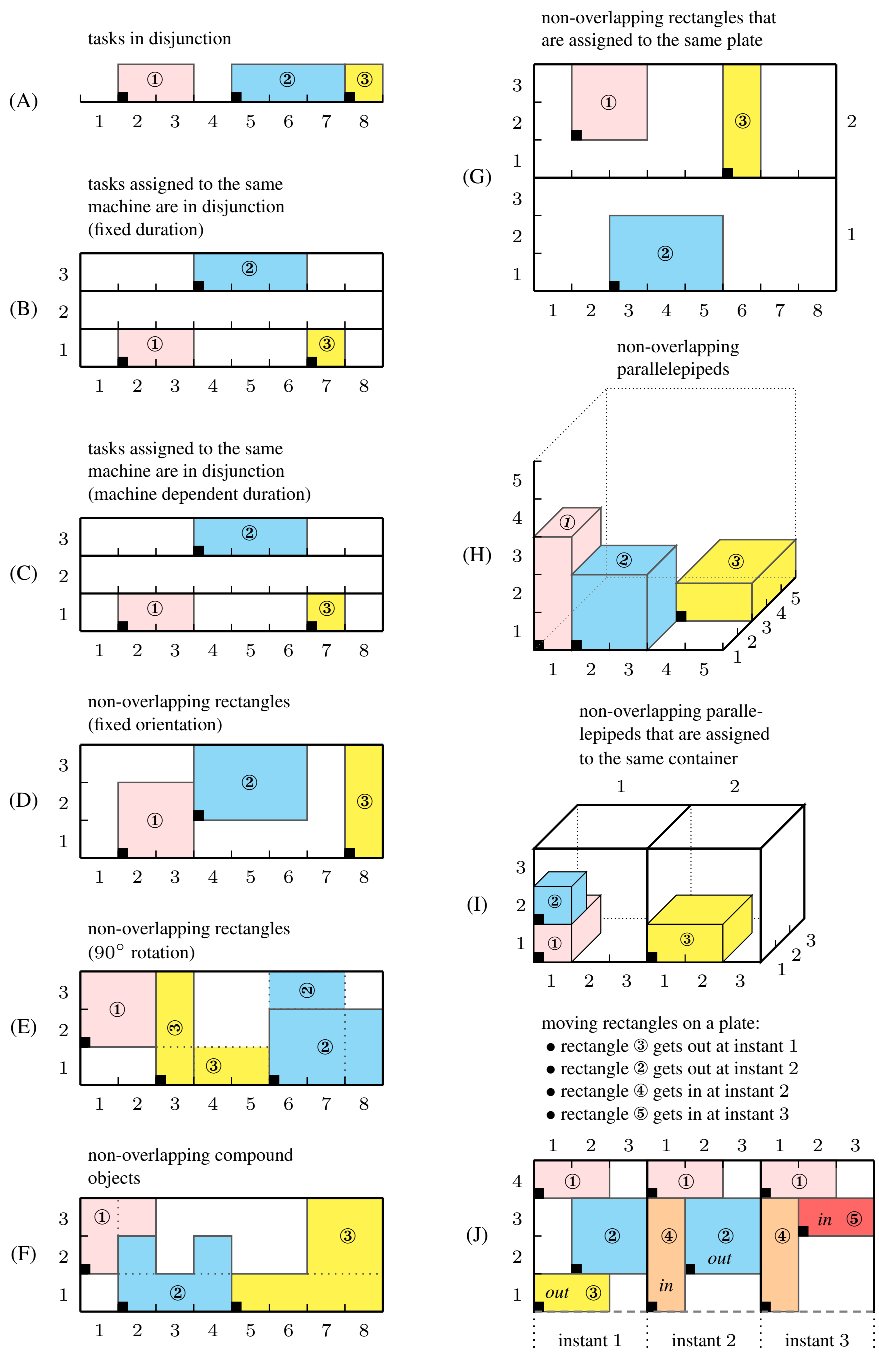


Figure 5.376: Ten typical examples of use of the GEOST_TIME constraint (ground instances)

5.163 GEQ

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint GEQ(VAR1, VAR2)

Synonyms REL, XGTEQY.

Arguments

VAR1 : dvar
VAR2 : dvar

Purpose Enforce the fact that the first variable is greater than or equal to the second variable.

Example $(8, 1)$

The GEQ constraint holds since 8 is greater than or equal to 1.

Symmetries

- VAR1 can be replaced by any value $\geq$ VAR2.
- VAR2 can be replaced by any value $\leq$ VAR1.

Systems GEQ in **Choco**, REL in **Gecode**, XGTEQY in **JaCoP**, #>= in **SICStus**.

See also

common keyword: NEQ *(binary constraint,arithmetic constraint)*.

generalisation: GEQ_CST *(constant added)*.

implied by: ABS_VALUE, EQ, GT.

implies (if swap arguments): LEQ.

negation: LT.

Keywords

constraint arguments: binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.164 GEQ_CST

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint GEQ_CST(VAR1, VAR2, CST2)

Arguments

VAR1	:	dvar
VAR2	:	dvar
CST2	:	int

Purpose Enforce the fact that the first variable is greater than or equal to the sum of the second variable and the constant.

Example $(8, 1, 7)$

The GEQ_CST constraint holds since 8 is greater than or equal to $1 + 7$.

Typical
- $\texttt{CST2} \neq 0$
- $\texttt{VAR1} > \texttt{VAR2} + \texttt{CST2}$

Symmetries
- Arguments are permutable w.r.t. permutation (VAR1) (VAR2, CST2).
- VAR1 can be replaced by any value $\geq$ VAR2 + CST2.
- VAR2 can be replaced by any value $\leq$ VAR1 − CST2.
- CST2 can be replaced by any value $\leq$ VAR1 − VAR2.

See also

common keyword: LEQ_CST *(binary constraint,arithmetic constraint)*.

implied by: EQ_CST.

specialisation: GEQ *(*constant *set to* 0*)*.

Keywords

constraint arguments: binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.165 GLOBAL_CARDINALITY

DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHARME [309]

Constraint GLOBAL_CARDINALITY(VARIABLES, VALUES)

Synonyms COUNT, DISTRIBUTE, DISTRIBUTION, GCC, CARD_VAR_GCC, EGCC, EXTENDED_GLOBAL_CARDINALITY.

Arguments

VARIABLES : collection(var−dvar)
VALUES : collection(val−int, noccurrence−dvar)

Restrictions

required(VARIABLES, var)
required(VALUES, [val, noccurrence])
distinct(VALUES, val)
VALUES.noccurrence ≥ 0
VALUES.noccurrence $\leq$ |VARIABLES|

Purpose Each value VALUES[i].val (with $i \in [1, |\text{VALUES}|]$) should be taken by exactly VALUES[i].noccurrence variables of the VARIABLES collection.

Example

$$\left(\begin{array}{l} \langle 3,3,8,6 \rangle, \\ \left\langle \begin{array}{ll} \texttt{val}-3 & \texttt{noccurrence}-2, \\ \texttt{val}-5 & \texttt{noccurrence}-0, \\ \texttt{val}-6 & \texttt{noccurrence}-1 \end{array} \right\rangle \end{array} \right)$$

The GLOBAL_CARDINALITY constraint holds since values 3, 5 and 6 respectively occur 2, 0 and 1 times within the collection $\langle 3,3,8,6 \rangle$ and since no restriction was specified for value 8.

All solutions Figure 5.377 gives all solutions to the following non ground instance of the GLOBAL_CARDINALITY constraint: $V_1 \in [3,4]$, $V_2 \in [2,3]$, $V_3 \in [1,2]$, $V_4 \in [2,4]$, $V_5 \in [2,3]$, $V_6 \in [1,2]$, $O_1 \in [1,1]$, $O_2 \in [2,3]$, $O_3 \in [0,1]$, $O_4 \in [2,3]$, GLOBAL_CARDINALITY($\langle V_1, V_2, V_3, V_4, V_5, V_6 \rangle, \langle 1\ O_1,\ 2\ O_2,\ 3\ O_3,\ 4\ O_4 \rangle$).

Typical

|VARIABLES| > 1
range(VARIABLES.var) > 1
|VALUES| > 1
|VARIABLES| $\geq$ |VALUES|
minval(VARIABLES.var) $= 0 \lor$ in_attr(VARIABLES, var, VALUES, val)

① $(\langle 4,2,1,4,2,2\rangle,\ \langle 1\ \mathbf{1},\ 2\ \mathbf{3},\ 3\ \mathbf{0},\ 4\ \mathbf{2}\rangle)$
② $(\langle 4,2,1,4,3,2\rangle,\ \langle 1\ \mathbf{1},\ 2\ \mathbf{2},\ 3\ \mathbf{1},\ 4\ \mathbf{2}\rangle)$
③ $(\langle 4,2,2,4,2,1\rangle,\ \langle 1\ \mathbf{1},\ 2\ \mathbf{3},\ 3\ \mathbf{0},\ 4\ \mathbf{2}\rangle)$
④ $(\langle 4,2,2,4,3,1\rangle,\ \langle 1\ \mathbf{1},\ 2\ \mathbf{2},\ 3\ \mathbf{1},\ 4\ \mathbf{2}\rangle)$
⑤ $(\langle 4,3,1,4,2,2\rangle,\ \langle 1\ \mathbf{1},\ 2\ \mathbf{2},\ 3\ \mathbf{1},\ 4\ \mathbf{2}\rangle)$
⑥ $(\langle 4,3,2,4,2,1\rangle,\ \langle 1\ \mathbf{1},\ 2\ \mathbf{2},\ 3\ \mathbf{1},\ 4\ \mathbf{2}\rangle)$

Figure 5.377: All solutions corresponding to the non ground example of the GLOBAL_CARDINALITY constraint of the **All solutions** slot

Symmetries

- Items of `VARIABLES` are permutable.
- Items of `VALUES` are permutable.
- An occurrence of a value of `VARIABLES.var` that does not belong to `VALUES.val` can be replaced by any other value that also does not belong to `VALUES.val`.
- All occurrences of two distinct values in `VARIABLES.var` or `VALUES.val` can be swapped; all occurrences of a value in `VARIABLES.var` or `VALUES.val` can be renamed to any unused value.

Arg. properties

- Functional dependency: `VALUES.noccurrence` determined by `VARIABLES` and `VALUES.val`.
- Contractible wrt. `VALUES`.

Usage

We show how to use the GLOBAL_CARDINALITY constraint in order to model the magic series problem [426, page 155] with a single GLOBAL_CARDINALITY constraint. A non-empty finite series $S = (s_0, s_1, \ldots, s_n)$ is *magic* if and only if there are s_i occurrences of i in S for each integer i ranging from 0 to n. This leads to the following model:

$$\texttt{global\_cardinality}\left(\begin{array}{l}\left\langle\ \texttt{var}-s_0, \texttt{var}-s_1, \ldots, \texttt{var}-s_n\ \right\rangle,\\ \left\langle\begin{array}{ll}\texttt{val}-0 & \texttt{noccurrence}-s_0,\\ \texttt{val}-1 & \texttt{noccurrence}-s_1,\\ & \vdots\\ \texttt{val}-n & \texttt{noccurrence}-s_n\end{array}\right\rangle\end{array}\right)$$

Remark

This is a generalised form of the original GLOBAL_CARDINALITY constraint: in the original GLOBAL_CARDINALITY constraint [353], one specifies for each value its minimum and maximum number of occurrences (i.e., see GLOBAL_CARDINALITY_LOW_UP). Here we give for each value v a domain variable that indicates how many time value v is actually used. By setting the minimum and maximum values of this variable to the appropriate constants we can express the same thing as in the original GLOBAL_CARDINALITY constraint. However, as shown in the *magic series* problem, we can also use this variable in other constraints. By reduction from 3-SAT, Claude-Guy Quimper shows in [342] that it is NP-hard to achieve arc-consistency for the count variables.

A last difference from the original GLOBAL_CARDINALITY constraint is that there is no constraint on the values that are not explicitly mentioned in the VALUES collection. In the original GLOBAL_CARDINALITY these values could not be assigned to the variables of the VARIABLES collection. However allowing values that are not mentioned in VALUES to be assigned to variables of VARIABLES can potentially avoid mentioning a huge number of unconstrained values in the VALUES collection, and as a side effect, prevent possibly[7] generating a dense graph (i.e., see DFS-bottleneck) for the corresponding underlying flow model).

Within [92] the GLOBAL_CARDINALITY constraint is called DISTRIBUTION. Within [361] the GLOBAL_CARDINALITY constraint is called CARD_VAR_GCC. Within [76] the GLOBAL_CARDINALITY constraint is called EGCC or RGCC. This later case corresponds to the fact that some variables are duplicated within the VARIABLES collection.

The GLOBAL_CARDINALITY constraint can be seen as a system (i.e., a conjunction) of AMONG constraints.

When all count variables (i.e., the variables $\texttt{VALUES}[i].\texttt{noccurrence}$ with $i \in [1, |\texttt{VALUES}|]$) *do not occur* in any other constraints of the problem, it may be operationally more efficient to replace the GLOBAL_CARDINALITY constraint by a GLOBAL_CARDINALITY_LOW_UP constraint where each count variable $\texttt{VALUES}[i].\texttt{noccurrence}$ is replaced by the corresponding interval $[\underline{\texttt{VALUES}[i].\texttt{noccurrence}}, \overline{\texttt{VALUES}[i].\texttt{noccurrence}}]$. This stands for two reasons:

- First, by using a GLOBAL_CARDINALITY_LOW_UP constraint rather than a GLOBAL_CARDINALITY constraint, we avoid the filtering algorithm related to the count variables.
- Second, unlike the GLOBAL_CARDINALITY constraint where we need to fix all its variables to get entailment, the GLOBAL_CARDINALITY_LOW_UP constraint can be entailed before all its variables get fixed. As a result, this potentially avoid unnecessary calls to its filtering algorithm.

When all values that can be assigned to the variables of the VARIABLES collection occur in the val attribute of the VALUES collection, two implicit *necessary* conditions[8] inferred by double counting with the GLOBAL_CARDINALITY constraint are depicted by the following expressions:

$$|\texttt{VARIABLES}| = \sum_{i=1}^{|\texttt{VALUES}|} \texttt{VALUES}[i].\texttt{noccurrence}$$

$$\sum_{i=1}^{|\texttt{VARIABLES}|} \texttt{VARIABLES}[i].\texttt{var} = \sum_{i=1}^{|\texttt{VALUES}|} \texttt{VALUES}[i].\texttt{val} \cdot \texttt{VALUES}[i].\texttt{noccurrence}$$

Within [328, pages 50–51] the previous condition where terms involving identical variables are grouped together (i.e., rule 5 of MALICE [327]) is mentioned as a crucial deduction rule for the autoref problem.

[7] Of course one could also, while generating a flow model, detect all unconstrained values in order to generate a single vertex in the flow model for the set of unconstrained values.

[8] Note that such necessary conditions can be derived by assigning an integer weight to each value [396], e.g. 1 for the first condition, the value itself for the second condition.

W.-J. van Hoeve *et al.* present two soft versions of the GLOBAL_CARDINALITY constraint in [435].

In **MiniZinc** (`http://www.minizinc.org/`) there is also a DISTRIBUTE constraint where the `val` attribute is not necessarily initially fixed and where a same value may occur more than once. There is also a GLOBAL_CARDINALITY_CLOSED constraint where all variables must be assigned a value from the `val` attribute.

Algorithm

A flow algorithm that handles the original GLOBAL_CARDINALITY constraint is described in [353]. The two approaches that were used to design bound-consistency algorithms for ALLDIFFERENT were generalised for the GLOBAL_CARDINALITY constraint. The algorithm in [345] identifies Hall intervals and the one in [244] exploits convexity to achieve a fast implementation of the flow-based arc-consistency algorithm. The later algorithm can also compute bound-consistency for the count variables [245, 242]. An improved algorithm for achieving arc-consistency is described in [344].

Systems

GLOBALCARDINALITY in **Choco**, COUNT in **Gecode**, GCC in **JaCoP**, GLOBAL_CARDINALITY in **MiniZinc**, GLOBAL_CARDINALITY in **SICStus**.

See also

common keyword: COUNT, MAX_NVALUE, MIN_NVALUE *(value constraint,counting constraint)*, NVALUE *(counting constraint)*, OPEN_GLOBAL_CARDINALITY_LOW_UP *(assignment,counting constraint)*.

cost variant: GLOBAL_CARDINALITY_WITH_COSTS *(*`cost` *associated with each* `variable`*,*`value` *pair)*.

implied by: GLOBAL_CARDINALITY_WITH_COSTS *(forget about cost)*, SAME_AND_GLOBAL_CARDINALITY *(conjoin* SAME *and* GLOBAL_CARDINALITY*)*.

part of system of constraints: AMONG.

related: ROOTS, SLIDING_CARD_SKIP0 *(counting constraint of a set of values on maximal sequences)*.

shift of concept: GLOBAL_CARDINALITY_NO_LOOP *(assignment of a* `variable` *to its position is ignored)*, ORDERED_GLOBAL_CARDINALITY *(restrictions are done on nested sets of values, all starting from first value)*, SYMMETRIC_CARDINALITY, SYMMETRIC_GCC.

soft variant: OPEN_GLOBAL_CARDINALITY *(a* `set variable` *defines the set of variables that are actually considered)*.

specialisation: ALLDIFFERENT *(each value should occur at most once)*, CARDINALITY_ATLEAST, CARDINALITY_ATMOST *(individual* `count variable` *for each value replaced by single* `count variable`*)*, CARDINALITY_ATMOST_PARTITION *(individual* `count variable` *for each value replaced by single* `count variable` *and* `variable` $\in$ `partition` *replaced by* `variable`*)*, GLOBAL_CARDINALITY_LOW_UP *(*`variable` *replaced by* `fixed interval`*)*.

system of constraints: COLORED_MATRIX *(one* GLOBAL_CARDINALITY *constraint for each* `row` *and each* `column` *of a* `matrix` *of* `variables`*)*.

uses in its reformulation: TREE_RANGE, TREE_RESOURCE.

Keywords

application area: assignment.

characteristic of a constraint: core, automaton, automaton with array of counters.

complexity: 3-SAT.

constraint arguments: pure functional dependency.

constraint type: value constraint, counting constraint, system of constraints.

filtering: Hall interval, bound-consistency, flow, duplicated variables, DFS-bottleneck.

modelling: functional dependency.

modelling exercises: magic series.

puzzles: magic series, autoref.

Cond. implications

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with minval(VARIABLES.var) $= 0$
  implies AND(VAR, VARIABLES)
  when VAR $= 0$.

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with maxval(VARIABLES.var) $= 1$
  implies OR(VAR, VARIABLES)
  when VAR $= 1$.

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with minval(VARIABLES.var) > 0
  implies MIN_SIZE_FULL_ZERO_STRETCH(MINSIZE, VARIABLES)
  when MINSIZE $=$ |VARIABLES|.

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with maxval(VARIABLES.var) < 0
  implies MIN_SIZE_FULL_ZERO_STRETCH(MINSIZE, VARIABLES)
  when MINSIZE $=$ |VARIABLES|.

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with range(VALUES.val) $=$nval(VALUES.val)
  and minval(VALUES.val) $\leq$minval(VARIABLES.var)
  and maxval(VALUES.val) $\geq$maxval(VARIABLES.var)
  implies AMONG_DIFF_0(NVAR, VARIABLES).

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with range(VALUES.val) $=$nval(VALUES.val)
  and minval(VALUES.val) $\leq$minval(VARIABLES.var)
  and maxval(VALUES.val) $\geq$maxval(VARIABLES.var)
  implies ATMOST_NVALUE(NVAL, VARIABLES).

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with range(VALUES.noccurrence) $= 1$
  and range(VALUES.val) $=$nval(VALUES.val)
  and minval(VALUES.val) $=$minval(VARIABLES.var)
  and maxval(VALUES.val) $=$maxval(VARIABLES.var)
  implies BALANCE(BALANCE, VARIABLES)
  when BALANCE $= 0$.

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with range(VALUES.val) $=$nval(VALUES.val)
  and minval(VALUES.val) $\leq$minval(VARIABLES.var)
  and maxval(VALUES.val) $\geq$maxval(VARIABLES.var)
  implies MAX_N(MAX, RANK, VARIABLES).

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with range(VALUES.val) = nval(VALUES.val)
  and minval(VALUES.val) $\leq$ minval(VARIABLES.var)
  and maxval(VALUES.val) $\geq$ maxval(VARIABLES.var)
  implies MAX_NVALUE(MAX, VARIABLES).

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with range(VALUES.val) = nval(VALUES.val)
  and minval(VALUES.val) $\leq$ minval(VARIABLES.var)
  and maxval(VALUES.val) $\geq$ maxval(VARIABLES.var)
  implies MIN_N(MIN, RANK, VARIABLES).

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with range(VALUES.val) = nval(VALUES.val)
  and minval(VALUES.val) $\leq$ minval(VARIABLES.var)
  and maxval(VALUES.val) $\geq$ maxval(VARIABLES.var)
  implies MIN_NVALUE(MIN, VARIABLES).

- GLOBAL_CARDINALITY(VARIABLES, VALUES)
  with range(VALUES.val) = nval(VALUES.val)
  and minval(VALUES.val) $\leq$ minval(VARIABLES.var)
  and maxval(VALUES.val) $\geq$ maxval(VARIABLES.var)
  implies RANGE_CTR(VARIABLES, CTR, R).

For all items of VALUES:

Arc input(s) VARIABLES

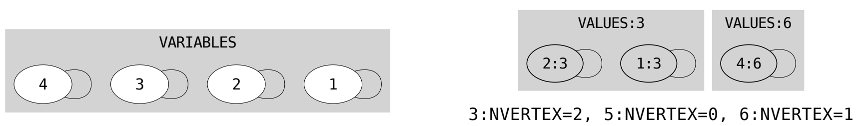
Arc generator *SELF* ↦ collection(variables)

Arc arity 1

Arc constraint(s) variables.var = VALUES.val

Graph property(ies) **NVERTEX** = VALUES.noccurrence

Graph model Since we want to express one unary constraint for each value we use the "For all items of VALUES" iterator. Part (A) of Figure 5.378 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.378 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold.



(A) **(B)**

Figure 5.378: Initial and final graph of the GLOBAL_CARDINALITY constraint

Automaton Figure 5.379 depicts the automaton associated with the GLOBAL_CARDINALITY constraint. To each item of the collection VARIABLES corresponds a signature variable S_i that is equal to 0. To each item of the collection VALUES corresponds a signature variable $S_{i+|\texttt{VARIABLES}|}$ that is equal to 1.

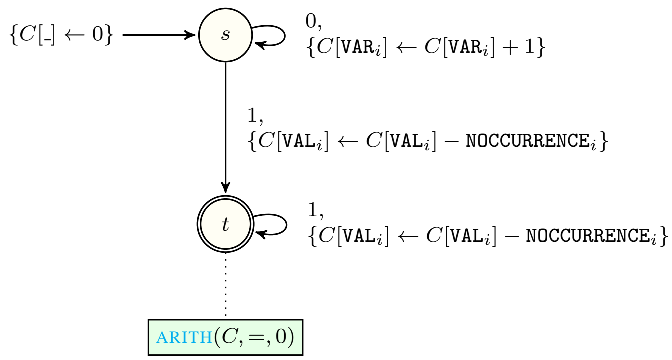


Figure 5.379: Automaton of the GLOBAL_CARDINALITY constraint

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint
GLOBAL_CARDINALITY$(\langle 2,4,2,2,1\rangle, \langle 0\ 0, 1\ 1, 2\ 3, 3\ 0, 4\ 1\rangle)$ hold?

B. Does the constraint
GLOBAL_CARDINALITY$(\langle 0,0,1,1\rangle, \langle 0\ 2, 1\ 2, 2\ 1, 3\ 0, 4\ 0\rangle)$ hold?

C. Does the constraint
GLOBAL_CARDINALITY$(\langle 2,3,4,5\rangle, \langle 0\ 0, 1\ 0, 2\ 1, 3\ 1, 4\ 1\rangle)$ hold?

[a]Hint: go back to the definition of GLOBAL_CARDINALITY.

EXERCISE 2 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\left\{\begin{array}{lll} V_1 \in [1,2], & V_2 \in [1,2], & V_3 \in [1,2], \\ V_4 \in [2,3], & V_5 \in [3,3], & \\ O_1 \in [1,2], & O_2 \in [2,3], & O_3 \in [0,1], \\ \multicolumn{3}{l}{\text{GLOBAL\_CARDINALITY}\left(\begin{array}{l} \langle V_1,V_2,V_3,V_4,V_5\rangle, \\ \left\langle\begin{array}{ll} \texttt{val}-1 & \texttt{occurrence}-O_1, \\ \texttt{val}-2 & \texttt{occurrence}-O_2, \\ \texttt{val}-3 & \texttt{occurrence}-O_3 \end{array}\right\rangle \end{array}\right)} \end{array}\right..$$

[a]Hint: focus on the variables of the first argument (since the counting variables of the second argument are functionally determined by the first argument), and enumerate solutions in lexicographic order.

EXERCISE 3 (identifying infeasible values)[a]

Identify all variable-value pairs (V_i, val) (respectively (O_i, val)) (with $i \in [1,5]$), such that the following constraint has no solution when variable V_i (respectively O_i) is assigned value val:

$$\left\{\begin{array}{lll}
V_1 \in [2,3], & V_2 \in [1,5], & V_3 \in [3,4], \\
V_4 \in [1,3], & V_5 \in [1,4], & \\
O_1 \in [1,4], & O_2 \in [0,1], & O_3 \in [0,1], \\
O_4 \in [1,5], & O_5 \in [1,4], & \\
\multicolumn{3}{l}{\text{GLOBAL_CARDINALITY}\left(\begin{array}{l}
\langle V_1, V_2, V_3, V_4, V_5 \rangle, \\
\left\langle \begin{array}{ll}
\texttt{val}-1 & \texttt{occurrence}-O_1, \\
\texttt{val}-2 & \texttt{occurrence}-O_2, \\
\texttt{val}-3 & \texttt{occurrence}-O_3, \\
\texttt{val}-4 & \texttt{occurrence}-O_4, \\
\texttt{val}-5 & \texttt{occurrence}-O_5
\end{array} \right\rangle
\end{array}\right).}
\end{array}\right.$$

[a]Hint: first restrict the occurrence variables $O_1, O_2, \ldots, O_5$, second restrict the decision variables $V_1, V_2, \ldots, V_5$, third check that all remaining values occur in at least one solution.

EXERCISE 4 (modelling a nurse assignment problem)[a]

Given a 24 hour period, you must schedule a pool of six nurses `Bea`, `Lea`, `Leo`, `Lio`, `Lili` and `Tom` to *at least two* and *at most three morning shifts*, to *at least two* and *at most three afternoon shifts*, to *at least one night shift*, while the other nurses are off-duty. In addition, due to past work, we have the following extra requirements:

- Since on the previous 24 hour period `Bea`, `Lea` and `Leo` were working in the afternoon shift they cannot be assigned to the night shift.
- `Leo`, `Lio` and `Lili` have to work since they already took all their days off.
- `Bea` and `Tom` have to work together since `Bea` supervises `Tom`.

Provide a model of this problem that uses the GLOBAL_CARDINALITY constraint.

A. Provide a solution that satisfies all the constraints, i.e., for each nurse give his/her assignment (`morning`, `afternoon`, `night`, `off-duty`).

B. Identify the decision variables and the values of the problem, i.e., how do we model the fact that nurse $x \in \{$`Bea`, `Lea`, `Leo`, `Lio`, `Lili`, `Tom`$\}$ is assigned shift $y \in \{$`morning`, `afternoon`, `night`, `off-duty`$\}$?

C. Using a bipartite graph, draw the relations between the variables and the values identified in the previous question and display the solution you came up with in the first question.

D. Provide a model of the problem that uses a single GLOBAL_CARDINALITY constraint.

- Explain how the minimum/maximum capacity constraints (i.e., at least/at most) are modelled.
- Explain how each extra requirement is modelled in your solution.

[a] Hint: focus on *what is a variable* and *what is a value* in your model, and how to model the capacity constraints with GLOBAL_CARDINALITY.

SOLUTION TO EXERCISE 1

A. *Yes, since within* $\langle 2,4,2,2,1\rangle$, *values* 0, 1, 2, 3 *and* 4 *are respectively used zero, one, three, zero, and one times.*

B. *No, since within* $\langle 0,0,1,1\rangle$, *value* 2 *is not used one time.*

C. *Yes, since within* $\langle 2,3,4,5\rangle$, *value* 0, 1, 2, 3 *and* 4 *are respectively used zero, zero, one, one, and one times. The presence of a* 5 *in the solution does not matter since value* 5 *is not mentioned in the values of the second argument of the* GLOBAL_CARDINALITY *constraint.*

SOLUTION TO EXERCISE 2

the six solutions

$\langle V_1, V_2, V_3, V_4, V_5\rangle$ $\langle 1\ O_1, 2\ O_2, 3\ O_3\rangle$

① $(\langle 1,1,2,2,3\rangle,\ \langle 1\ \mathbf{2}, 2\ \mathbf{2}, 3\ \mathbf{1}\rangle)$

② $(\langle 1,2,1,2,3\rangle,\ \langle 1\ \mathbf{2}, 2\ \mathbf{2}, 3\ \mathbf{1}\rangle)$

③ $(\langle 1,2,2,2,3\rangle,\ \langle 1\ \mathbf{1}, 2\ \mathbf{3}, 3\ \mathbf{1}\rangle)$

④ $(\langle 2,1,1,2,3\rangle,\ \langle 1\ \mathbf{2}, 2\ \mathbf{2}, 3\ \mathbf{1}\rangle)$

⑤ $(\langle 2,1,2,2,3\rangle,\ \langle 1\ \mathbf{1}, 2\ \mathbf{3}, 3\ \mathbf{1}\rangle)$

⑥ $(\langle 2,2,1,2,3\rangle,\ \langle 1\ \mathbf{1}, 2\ \mathbf{3}, 3\ \mathbf{1}\rangle)$

SOLUTION TO EXERCISE 3

As suggested by the hint we go through the following steps:

A. [*RESTRICTING THE OCCURRENCE VARIABLES* $O_1, O_2, \ldots, O_5$]

(a) [*PRUNING WRT THE MAXIMUM NUMBER OF OCCURRENCES OF EACH VALUE*]
Since values 1, 2, 3, 4 *and* 5 *can respectively be assigned to at most* 3, 4, 5, 3 *and* 1 *decision variables (e.g., value* 1 *can only be assigned to* V_2, V_4 *and* V_5*) we have* $O_1 \leq \min(3,4)$, $O_2 \leq \min(4,1)$, $O_3 \leq \min(5,1)$, $O_4 \leq \min(3,5)$, *and* $O_5 \leq \min(1,4)$.

(b) [*PRUNING WRT* $\sum_{i=1}^{5} O_i = 5$ *AND THE DOMAIN OF* V_1]
Since we have five decision variables the sum of the occurrence variables is equal to five (i.e., $O_1 + O_2 + O_3 + O_4 + O_5 = 5$*). Since values* 2 *or* 3 *have to be assigned to the decision variable* V_1 *we have* $O_2 + O_3 \geq 1$*. It follows that* $O_1 + O_4 + O_5 \leq 4$*. Since* $O_1 \in [1,3]$, $O_4 \in [1,3]$ *and* $O_5 = 1$ *we get* $O_1 + O_4 \leq 3$ *and consequently* $O_1 \leq 2$ *and* $O_4 \leq 2$.

B. [*RESTRICTING THE DECISION VARIABLES* $V_1, V_2, \ldots, V_5$]
At the end of step **A** *we obtain* $O_1 \in [1,2]$, $O_2 \in [0,1]$, $O_3 \in [0,1]$, $O_4 \in [1,2]$, *and* $O_5 \in [1,1]$*. Since* $O_5 = 1$ *and since* V_2 *is the only decision variable that can be assigned value* 5 *we have* $V_2 = 5$. *Consequently* $V_1 \in [2,3]$, $V_2 \in [5,5]$, $V_3 \in [3,4]$, $V_4 \in [1,3]$, *and* $V_5 \in [1,4]$.

C. [*CHECKING FOR A SUPPORT*]
To show that no value can be removed from the domain of the decision and occurrence variables we show that every value that is still in the domain of a variable is part of a solution.

(a) *A solution with* $O_1 = 2$ *is* $V_1 = 2$, $V_2 = 5$, $V_3 = 4$, $V_4 = 1$, $V_5 = 1$ *and* $O_1 = 2$, $O_2 = 1$, $O_3 = 0$, $O_4 = 1$, $O_5 = 1$.

(b) *A solution with* $O_4 = 2$ *is* $V_1 = 2$, $V_2 = 5$, $V_3 = 4$, $V_4 = 1$, $V_5 = 4$ *and* $O_1 = 1$, $O_2 = 1$, $O_3 = 0$, $O_4 = 2$, $O_5 = 1$.

(c) *We now assume that* $O_1 = O_2 = O_3 = O_4 = O_5 = 1$*, i.e., all decision variables must be distinct. Without loos of generality we ignore* V_2*, which is fixed to* 5*. We provide a set of solutions where* V_1, V_3, V_4 *and* V_5 *can respectively be assigned to all the values of their domains:*

i. $V_1 = 2$, $V_3 = 3$, $V_4 = 1$, $V_5 = 4$,
ii. $V_1 = 2$, $V_3 = 4$, $V_4 = 3$, $V_5 = 1$,
iii. $V_1 = 2$, $V_3 = 4$, $V_4 = 1$, $V_5 = 3$,
iv. $V_1 = 3$, $V_3 = 4$, $V_4 = 1$, $V_5 = 2$,
v. $V_1 = 3$, $V_3 = 4$, $V_4 = 2$, $V_5 = 1$.

SOLUTION TO EXERCISE 4

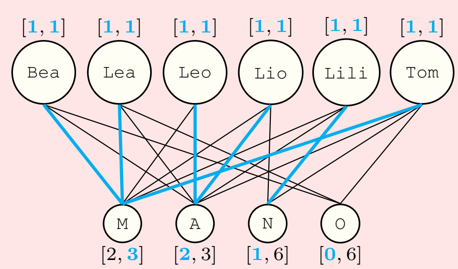
A. *A feasible solution is the following assignment*

$$\left(\begin{array}{lll} \texttt{Bea}:\texttt{morning} & \texttt{Lea}:\texttt{morning} & \texttt{Leo}:\texttt{afternoon} \\ \texttt{Lio}:\texttt{afternoon} & \texttt{Lili}:\texttt{night} & \texttt{Tom}:\texttt{morning} \end{array}\right) \textit{ since:}$$

- *the number of morning shifts is between* 2 *and* 3*,*
- *the number of afternoon shifts is between* 2 *and* 3*,*
- *the number of night shifts is at least* 1*,*
- `Bea`*,* `Lea` *and* `Leo` *are not assigned to a night shift,*
- `Leo`*,* `Lio` *and* `Lili` *work,*
- `Bea` *and* `Tom` *are both assigned the same shift.*

B. *To each nurse corresponds a variable whose initial domain is set to the types of shifts that nurse can actually perform (i.e., each shift type is encoded by a unique integer value).*

C. *The next figure provides a graphical representation of the assignment problem. To each nurse and to each shift type corresponds a vertex. There is an edge between a given nurse and a given shift type if and only if that nurse can perform that shift type. The solution given to question* **A** *is displayed with thick blue lines. The interval on top or below each vertex indicates the minimum and maximum number of edges that can reach the corresponding vertex in any solution; values in blue correspond to the number of edges of the displayed solution.*



D. *We get the following model*

$$\left\{\begin{array}{llll}
\mathtt{M}=1, & \mathtt{A}=2, & \mathtt{N}=3, & \mathtt{O}=4,\\
\mathtt{Bea}\in[\mathtt{M},\mathtt{O}], & \mathtt{Lea}\in[\mathtt{M},\mathtt{O}], & \mathtt{Leo}\in[\mathtt{M},\mathtt{O}], & \\
\mathtt{Lio}\in[\mathtt{M},\mathtt{O}], & \mathtt{Lili}\in[\mathtt{M},\mathtt{O}], & \mathtt{Tom}\in[\mathtt{M},\mathtt{O}], & \\
O_{\mathtt{M}}\in[2,3], & O_{\mathtt{A}}\in[2,3], & O_{\mathtt{N}}\in[1,6], & O_{\mathtt{O}}\in[0,6],\\
\mathtt{Bea}\neq\mathtt{N}, & \mathtt{Lea}\neq\mathtt{N}, & \mathtt{Leo}\neq\mathtt{N}, & \\
\mathtt{Leo}\neq\mathtt{O}, & \mathtt{Lio}\neq\mathtt{O}, & \mathtt{Lili}\neq\mathtt{O}, & \\
\mathtt{Bea}=\mathtt{Tom}, & & & \\
\multicolumn{4}{l}{\text{GLOBAL_CARDINALITY}\left(\begin{array}{l}\langle\mathtt{Bea},\mathtt{Lea},\mathtt{Leo},\mathtt{Lio},\mathtt{Lili},\mathtt{Tom}\rangle,\\ \left\langle\begin{array}{ll}\mathbf{val}-\mathtt{M} & \mathbf{occurrence}-O_{\mathtt{M}},\\ \mathbf{val}-\mathtt{A} & \mathbf{occurrence}-O_{\mathtt{A}},\\ \mathbf{val}-\mathtt{N} & \mathbf{occurrence}-O_{\mathtt{N}},\\ \mathbf{val}-\mathtt{O} & \mathbf{occurrence}-O_{\mathtt{O}}\end{array}\right\rangle\end{array}\right),}
\end{array}\right.$$

where:

- *line 1 declares the integer value of each shift type,*
- *lines 2, 3 and 4 declare the nurse and occurrence variables,*
- *line 5 enforces* `Bea`*,* `Lea` *and* `Leo` *not to work on a night shift,*
- *line 6 imposes* `Leo`*,* `Lio` *and* `Lili` *to work,*
- *line 7 constrains* `Bea` *and* `Tom` *to work on the same shift,*
- *line 8 restricts each shift type to occur within a given range.*

5.166 GLOBAL_CARDINALITY_LOW_UP

DESCRIPTION LINKS GRAPH

Origin

Used for defining SLIDING_DISTRIBUTION.

Constraint

GLOBAL_CARDINALITY_LOW_UP(VARIABLES, VALUES)

Synonyms

GCC_LOW_UP, GCC.

Arguments

VARIABLES : collection(var−dvar)
VALUES : collection(val−int, omin−int, omax−int)

Restrictions

required(VARIABLES, var)
$|\mathtt{VALUES}| > 0$
required(VALUES, [val, omin, omax])
distinct(VALUES, val)
$\mathtt{VALUES.omin} \geq 0$
$\mathtt{VALUES.omax} \leq |\mathtt{VARIABLES}|$
$\mathtt{VALUES.omin} \leq \mathtt{VALUES.omax}$

Purpose

Each value VALUES$[i]$.val $(1 \leq i \leq |\mathtt{VALUES}|)$ should be taken by at least VALUES$[i]$.omin and at most VALUES$[i]$.omax variables of the VARIABLES collection.

Example

$$\left(\begin{array}{l}\langle 3,3,8,6\rangle, \\ \left\langle\begin{array}{lll}\mathtt{val}-3 & \mathtt{omin}-2 & \mathtt{omax}-3, \\ \mathtt{val}-5 & \mathtt{omin}-0 & \mathtt{omax}-1, \\ \mathtt{val}-6 & \mathtt{omin}-1 & \mathtt{omax}-2\end{array}\right\rangle\end{array}\right)$$

The GLOBAL_CARDINALITY_LOW_UP constraint holds since values 3, 5 and 6 are respectively used 2 $(2 \leq 2 \leq 3)$, 0 $(0 \leq 0 \leq 1)$ and 1 $(1 \leq 1 \leq 2)$ times within the collection $\langle 3,3,8,6\rangle$ and since no constraint was specified for value 8.

Typical

$|\mathtt{VARIABLES}| > 1$
range(VARIABLES.var) > 1
$|\mathtt{VALUES}| > 1$
$\mathtt{VALUES.omin} \leq |\mathtt{VARIABLES}|$
$\mathtt{VALUES.omax} > 0$
$\mathtt{VALUES.omax} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > |\mathtt{VALUES}|$
in_attr(VARIABLES, var, VALUES, val)

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.
- Items of VALUES are permutable.
- VALUES.omin can be decreased to any value ≥ 0.
- VALUES.omax can be increased to any value $\leq$ |VARIABLES|.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Arg. properties

Contractible wrt. VALUES.

Remark

Within the context of linear programming [226, page 376] provides relaxations of the GLOBAL_CARDINALITY_LOW_UP constraint.

In **MiniZinc** (http://www.minizinc.org/) there is also a GLOBAL_CARDINALITY_LOW_UP_CLOSED constraint where all variables must be assigned a value from the val attribute.

Algorithm

A filtering algorithm achieving arc-consistency for the GLOBAL_CARDINALITY_LOW_UP constraint is given in [353]. This algorithm is based on a flow model of the GLOBAL_CARDINALITY_LOW_UP constraint where there is a one-to-one correspondence between feasible flows in the flow model and solutions to the GLOBAL_CARDINALITY_LOW_UP constraint. The leftmost part of Figure 3.30 illustrates this flow model.

The GLOBAL_CARDINALITY_LOW_UP constraint is entailed if and only if for each value v equal to VALUES$[i]$.val (with $1 \leq i \leq$ |VALUES|) the following two conditions hold:

1. The number of variables of the VARIABLES collection assigned value v is greater than or equal to VALUES$[i]$.omin.
2. The number of variables of the VARIABLES collection that can potentially be assigned value v is less than or equal to VALUES$[i]$.omax.

Reformulation

A reformulation of the GLOBAL_CARDINALITY_LOW_UP, involving linear constraints, preserving bound-consistency was introduced in [78]. For each potential interval $[l, u]$ of consecutive values this model uses |VARIABLES| 0-1 variables $B_{1,l,u}, B_{2,l,u}, \ldots, B_{|\texttt{VARIABLES}|,l,u}$ for modelling the fact that each variable of the collection VARIABLES is assigned a value within interval $[l, u]$ (i.e., $\forall i \in [1, |\texttt{VARIABLES}|]$: $B_{i,l,u} \Leftrightarrow l \leq \texttt{VARIABLES}[i].\texttt{var} \wedge \texttt{VARIABLES}[i].\texttt{var} \leq u$), as well as one domain variable $C_{l,u}$ for counting how many values of $[l, u]$ are assigned to variables of VARIABLES (i.e. $C_{l,u} = B_{1,l,u} + B_{2,l,u} + \cdots + B_{|\texttt{VARIABLES}|,l,u}$). The lower and upper bounds of variable $C_{l,u}$ are respectively initially set with respect to the minimum and maximum number of possible occurrences of the values of interval $[l, u]$. Finally, assuming that s is the smallest value that can be assigned to the variables of VARIABLES, the constraint $C_{s,u} = C_{s,k} + C_{k+1,u}$ is stated for each $k \in [s, u-1]$.

Systems

GLOBALCARDINALITY in **Choco**, GLOBAL_CARDINALITY_LOW_UP in **MiniZinc**.

Used in

SLIDING_DISTRIBUTION.

See also

common keyword: OPEN_GLOBAL_CARDINALITY *(assignment,counting constraint)*.

generalisation: CUMULATIVE *(*`variables` *replaced by* `tasks`*)*, GLOBAL_CARDINALITY *(*`fixed interval` *replaced by* `variable`*)*.

implied by: INCREASING_GLOBAL_CARDINALITY *(a* GLOBAL_CARDINALITY_LOW_UP *constraint where the* `variables` *are increasing)*, SAME_AND_GLOBAL_CARDINALITY_LOW_UP.

related: ORDERED_GLOBAL_CARDINALITY *(restrictions are done on nested sets of values, all starting from first value)*.

shift of concept: GLOBAL_CARDINALITY_LOW_UP_NO_LOOP *(assignment of a* `variable` *to its position is ignored)*.

soft variant: OPEN_GLOBAL_CARDINALITY_LOW_UP *(a* `set variable` *defines the set of variables that are actually considered)*.

specialisation: ALLDIFFERENT *(each value should occur at most once)*.

system of constraints: SLIDING_DISTRIBUTION *(one* GLOBAL_CARDINALITY_LOW_UP *constraint for each sliding sequence of* `SEQ` *consecutive* `variables`*)*.

Keywords

application area: assignment.

constraint type: value constraint, counting constraint.

filtering: flow, arc-consistency, bound-consistency, DFS-bottleneck, entailment.

Cond. implications

GLOBAL_CARDINALITY_LOW_UP(VARIABLES, VALUES)
with INCREASING(VARIABLES)
implies INCREASING_GLOBAL_CARDINALITY(VARIABLES, VALUES).

For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUES.val
Graph property(ies)	• **NVERTEX** ≥ VALUES.omin • **NVERTEX** ≤ VALUES.omax

Graph model Since we want to express one unary constraint for each value we use the "For all items of VALUES" iterator. Part (A) of Figure 5.380 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.380 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold.

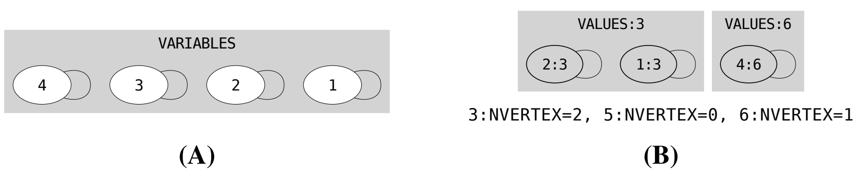


Figure 5.380: Initial and final graph of the GLOBAL_CARDINALITY_LOW_UP constraint

5.167 GLOBAL_CARDINALITY_LOW_UP_NO_LOOP

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

Derived from GLOBAL_CARDINALITY_LOW_UP and TREE.

Constraint

$$\text{GLOBAL\_CARDINALITY\_LOW\_UP\_NO\_LOOP}\left(\begin{array}{l}\texttt{MINLOOP},\\ \texttt{MAXLOOP},\\ \texttt{VARIABLES},\\ \texttt{VALUES}\end{array}\right)$$

Synonym

GCC_LOW_UP_NO_LOOP.

Arguments

MINLOOP	:	int
MAXLOOP	:	int
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int, omin−int, omax−int)

Restrictions

$\texttt{MINLOOP} \geq 0$
$\texttt{MINLOOP} \leq \texttt{MAXLOOP}$
$\texttt{MAXLOOP} \leq |\texttt{VARIABLES}|$
$\text{required}(\texttt{VARIABLES}, \texttt{var})$
$|\texttt{VALUES}| > 0$
$\text{required}(\texttt{VALUES}, [\texttt{val}, \texttt{omin}, \texttt{omax}])$
$\text{distinct}(\texttt{VALUES}, \texttt{val})$
$\texttt{VALUES.omin} \geq 0$
$\texttt{VALUES.omax} \leq |\texttt{VARIABLES}|$
$\texttt{VALUES.omin} \leq \texttt{VALUES.omax}$

Purpose

$\texttt{VALUES}[i].\texttt{omin}$ $(1 \leq i \leq |\texttt{VALUES}|)$ is less than or equal to the number of variables $\texttt{VARIABLES}[j].\texttt{var}$ $(j \neq i, 1 \leq j \leq |\texttt{VARIABLES}|)$ that are assigned value $\texttt{VALUES}[i].\texttt{val}$.
$\texttt{VALUES}[i].\texttt{omax}$ $(1 \leq i \leq |\texttt{VALUES}|)$ is greater than or equal to the number of variables $\texttt{VARIABLES}[j].\texttt{var}$ $(j \neq i, 1 \leq j \leq |\texttt{VARIABLES}|)$ that are assigned value $\texttt{VALUES}[i].\texttt{val}$.
The number of assignments of the form $\texttt{VARIABLES}[i].\texttt{var} = i$ $(i \in [1, |\texttt{VARIABLES}|])$ is greater than or equal to MINLOOP and less than or equal to MAXLOOP.

Example

$$\left(\begin{array}{l}1, 1, \langle 1, 1, 8, 6\rangle,\\ \left\langle\begin{array}{lll}\texttt{val}-1 & \texttt{omin}-1 & \texttt{omax}-1,\\ \texttt{val}-5 & \texttt{omin}-0 & \texttt{omax}-0,\\ \texttt{val}-6 & \texttt{omin}-1 & \texttt{omax}-2\end{array}\right\rangle\end{array}\right)$$

The GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraint holds since:

- Values 1, 5 and 6 are respectively assigned to the set of variables $\{\texttt{VARIABLES}[2].\texttt{var}\}$ (i.e., $\texttt{omin} = 1 \leq 1 \leq \texttt{omax} = 1$), $\{\}$ (i.e., $\texttt{omin} = 0 \leq$

$0 \le$ omax $= 0$) and $\{$VARIABLES$[4]$.var$\}$ (i.e., omin $= 1 \le 1 \le$ omax $= 2$). Note that, due to the definition of the constraint, the fact that VARIABLES$[1]$.var is assigned to 1 is not counted.

- In addition the number of assignments of the form VARIABLES$[i]$.var $= i$ ($i \in [1, 4]$) is greater than or equal to MINLOOP $= 1$ and less than or equal to MAXLOOP $= 1$.

Typical

|VARIABLES| > 1
range(VARIABLES.var) > 1
|VALUES| > 1
VALUES.omin $\le$ |VARIABLES|
VALUES.omax > 0
VALUES.omax $<$ |VARIABLES|
|VARIABLES| $>$ |VALUES|

Symmetries

- Items of VALUES are permutable.
- VALUES.omin can be decreased to any value ≥ 0.
- VALUES.omax can be increased to any value $\le$ |VARIABLES|.

Usage

Within the context of the TREE constraint the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraint allows one to model a minimum and maximum degree constraint on each vertex of our trees.

Algorithm

The flow algorithm that handles the original GLOBAL_CARDINALITY constraint [353] can be adapted to the context of the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraint. This is done by creating an extra *value* node representing the loops corresponding to the roots of the trees. The rightmost part of Figure 3.30 illustrates the corresponding flow model for the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraint where there is a one-to-one correspondence between feasible flows in the flow model and solutions to the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraint.

See also

generalisation: GLOBAL_CARDINALITY_NO_LOOP (fixed interval *replaced by* variable).

implied by: SAME_AND_GLOBAL_CARDINALITY_LOW_UP.

related: TREE (*graph partitioning by a set of trees with degree restrictions*).

root concept: GLOBAL_CARDINALITY_LOW_UP (*assignment of a* variable *to its position is ignored*).

Keywords

constraint type: value constraint.

filtering: flow.

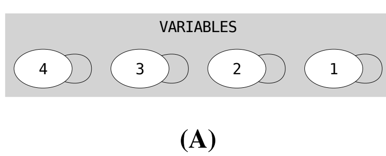
For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	• variables.var $=$ VALUES.val • variables.key $\neq$ VALUES.val
Graph property(ies)	• **NVERTEX** $\geq$ VALUES.omin • **NVERTEX** $\leq$ VALUES.omax

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var $=$ variables.key
Graph property(ies)	• **NARC** $\geq$ MINLOOP • **NARC** $\leq$ MAXLOOP

Graph model

Since, within the context of the first graph constraint, we want to express one unary constraint for each value we use the “For all items of VALUES” iterator. Part (A) of Figure 5.381 shows the initial graphs associated with each value 1, 5 and 6 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.381 shows the two corresponding final graphs respectively associated with values 1 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold.



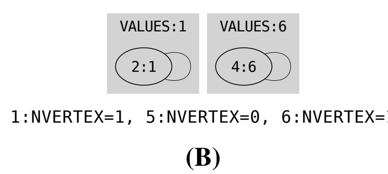


Figure 5.381: Initial and final graph of the GLOBAL_CARDINALITY_LOW_UP_NO_LOOP constraint

5.168 GLOBAL_CARDINALITY_NO_LOOP

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from GLOBAL_CARDINALITY and TREE.

Constraint GLOBAL_CARDINALITY_NO_LOOP($\texttt{NLOOP}, \texttt{VARIABLES}, \texttt{VALUES}$)

Synonym GCC_NO_LOOP.

Arguments

NLOOP	:	dvar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int, noccurrence−dvar)

Restrictions

$\texttt{NLOOP} \geq 0$
$\texttt{NLOOP} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)
$|\texttt{VALUES}| > 0$
required(VALUES, [val, noccurrence])
distinct(VALUES, val)
$\texttt{VALUES.noccurrence} \geq 0$
$\texttt{VALUES.noccurrence} \leq |\texttt{VARIABLES}|$

Purpose

$\texttt{VALUES}[i].\texttt{noccurrence}$ $(1 \leq i \leq |\texttt{VALUES}|)$ is equal to the number of variables $\texttt{VARIABLES}[j].\texttt{var}$ $(j \neq i, 1 \leq j \leq |\texttt{VARIABLES}|)$ that are assigned value $\texttt{VALUES}[i].\texttt{val}$.
The number of assignments of the form $\texttt{VARIABLES}[i].\texttt{var} = i$ $(i \in [1, |\texttt{VARIABLES}|])$ is equal to NLOOP.

Example

$$\left(\begin{array}{l} 1, \langle 1, 1, 8, 6\rangle, \\ \left\langle\begin{array}{ll} \texttt{val}-1 & \texttt{noccurrence}-1, \\ \texttt{val}-5 & \texttt{noccurrence}-0, \\ \texttt{val}-6 & \texttt{noccurrence}-1 \end{array}\right\rangle \end{array}\right)$$

The GLOBAL_CARDINALITY_NO_LOOP constraint holds since:

- Values 1, 5 and 6 are respectively assigned to the set of variables $\{\texttt{VARIABLES}[2].\texttt{var}\}$ (i.e., 1 occurrence of value 1), $\{\}$ (i.e., no occurrence of value 5) and $\{\texttt{VARIABLES}[4].\texttt{var}\}$ (i.e., 1 occurrence of value 6). Note that, due to the definition of the constraint, the fact that $\texttt{VARIABLES}[1].\texttt{var}$ is assigned to 1 is not counted.
- In addition the number of assignments of the form $\texttt{VARIABLES}[i].\texttt{var} = i$ $(i \in [1, 4])$ is equal to $\texttt{NLOOP} = 1$.

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VALUES}| > 1$
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$

Symmetry Items of VALUES are permutable.

Arg. properties

- Functional dependency: NLOOP determined by VARIABLES.
- Functional dependency: VALUES.noccurrence determined by VARIABLES and VALUES.val.

Usage Within the context of the TREE constraint the GLOBAL_CARDINALITY_NO_LOOP constraint allows one to model a minimum and maximum degree constraint on each vertex of our trees.

Algorithm The flow algorithm that handles the original GLOBAL_CARDINALITY constraint [353] can be adapted to the context of the GLOBAL_CARDINALITY_NO_LOOP constraint. This is done by creating an extra *value* node representing the loops corresponding to the roots of the trees.

See also

related: TREE *(graph partitioning by a set of trees with degree restrictions)*.

root concept: GLOBAL_CARDINALITY *(assignment of a* variable *to its position is ignored)*.

specialisation: GLOBAL_CARDINALITY_LOW_UP_NO_LOOP *(*variable *replaced by* fixed interval*)*.

Keywords

constraint arguments: pure functional dependency.

constraint type: value constraint.

filtering: flow.

modelling: functional dependency.

For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	• variables.var = VALUES.val • variables.key ≠ VALUES.val
Graph property(ies)	**NVERTEX** = VALUES.noccurrence

Arc input(s)	VARIABLES
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = variables.key
Graph property(ies)	**NARC** = NLOOP

Graph model Since, within the context of the first graph constraint, we want to express one unary constraint for each value we use the "For all items of VALUES" iterator. Part (A) of Figure 5.382 shows the initial graphs associated with each value 1, 5 and 6 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.382 shows the two corresponding final graphs respectively associated with values 1 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold.

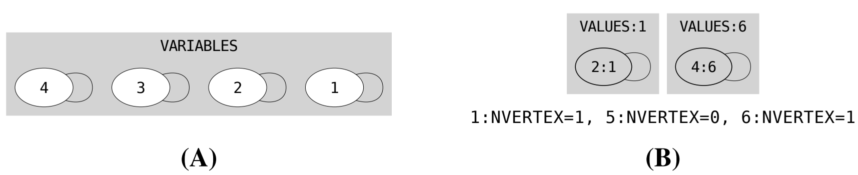


Figure 5.382: Initial and final graph of the GLOBAL_CARDINALITY_NO_LOOP constraint

5.169 GLOBAL_CARDINALITY_WITH_COSTS

DESCRIPTION LINKS GRAPH

Origin [355]

Constraint GLOBAL_CARDINALITY_WITH_COSTS(VARIABLES, VALUES, MATRIX, COST)

Synonyms GCCC, COST_GCC.

Arguments

VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int, noccurrence−dvar)
MATRIX	:	collection(i−int, j−int, c−int)
COST	:	dvar

Restrictions

required(VARIABLES, var)
$|\mathtt{VALUES}| > 0$
required(VALUES, [val, noccurrence])
distinct(VALUES, val)
$\mathtt{VALUES.noccurrence} \geq 0$
$\mathtt{VALUES.noccurrence} \leq |\mathtt{VARIABLES}|$
required(MATRIX, [i, j, c])
increasing_seq(MATRIX, [i, j])
$\mathtt{MATRIX.i} \geq 1$
$\mathtt{MATRIX.i} \leq |\mathtt{VARIABLES}|$
$\mathtt{MATRIX.j} \geq 1$
$\mathtt{MATRIX.j} \leq |\mathtt{VALUES}|$
$|\mathtt{MATRIX}| = |\mathtt{VARIABLES}| * |\mathtt{VALUES}|$

Purpose

Each value VALUES[i].val should be taken by exactly VALUES[i].noccurrence variables of the VARIABLES collection. In addition the COST of an assignment is equal to the sum of the elementary costs associated with the fact that we assign variable i of the VARIABLES collection to the j^{th} value of the VALUES collection. These elementary costs are given by the MATRIX collection.

Example

$$\left(\begin{array}{l}\langle 3,3,3,6\rangle, \\ \left\langle\begin{array}{ll}\texttt{val}-3 & \texttt{noccurrence}-3, \\ \texttt{val}-5 & \texttt{noccurrence}-0, \\ \texttt{val}-6 & \texttt{noccurrence}-1\end{array}\right\rangle, \\ \left\langle\begin{array}{lll}\texttt{i}-1 & \texttt{j}-1 & \texttt{c}-4, \\ \texttt{i}-1 & \texttt{j}-2 & \texttt{c}-1, \\ \texttt{i}-1 & \texttt{j}-3 & \texttt{c}-7, \\ \texttt{i}-2 & \texttt{j}-1 & \texttt{c}-1, \\ \texttt{i}-2 & \texttt{j}-2 & \texttt{c}-0, \\ \texttt{i}-2 & \texttt{j}-3 & \texttt{c}-8, \\ \texttt{i}-3 & \texttt{j}-1 & \texttt{c}-3, \\ \texttt{i}-3 & \texttt{j}-2 & \texttt{c}-2, \\ \texttt{i}-3 & \texttt{j}-3 & \texttt{c}-1, \\ \texttt{i}-4 & \texttt{j}-1 & \texttt{c}-0, \\ \texttt{i}-4 & \texttt{j}-2 & \texttt{c}-0, \\ \texttt{i}-4 & \texttt{j}-3 & \texttt{c}-6\end{array}\right\rangle, 14\end{array}\right)$$

The GLOBAL_CARDINALITY_WITH_COSTS constraint holds since:

- Values 3, 5 and 6 respectively occur 3, 0 and 1 times within the collection $\langle 3,3,3,6\rangle$.
- The COST argument corresponds to the sum of the costs respectively associated with the first, second, third and fourth items of $\langle 3,3,3,6\rangle$, namely 4, 1, 3 and 6.

All solutions

Figure 5.383 gives all solutions to the following non ground instance of the GLOBAL_CARDINALITY_WITH_COSTS constraint:
$\texttt{V}_1 \in [3,4]$, $\texttt{V}_2 \in [2,3]$, $\texttt{V}_3 \in [1,2]$, $\texttt{V}_4 \in [2,4]$, $\texttt{V}_5 \in [2,3]$, $\texttt{V}_6 \in [1,2]$,
$\texttt{O}_1 \in [1,1]$, $\texttt{O}_2 \in [2,3]$, $\texttt{O}_3 \in [0,1]$, $\texttt{O}_4 \in [2,3]$,
$\texttt{C} \in [0,16]$,

$$\begin{array}{l}\text{GLOBAL\_CARDINALITY\_WITH\_COSTS}(\langle \texttt{V}_1,\texttt{V}_2,\texttt{V}_3,\texttt{V}_4,\texttt{V}_5,\texttt{V}_6\rangle, \\ \quad\langle 1\ \texttt{O}_1,\ 2\ \texttt{O}_2,\ 3\ \texttt{O}_3,\ 4\ \texttt{O}_4\rangle, \\ \quad\langle 1\,1\,5,\ 1\,2\,0,\ 1\,3\,1,\ 1\,4\,1, \\ \quad\ 2\,1\,2,\ 2\,2\,7,\ 2\,3\,0,\ 2\,4\,2, \\ \quad\ 3\,1\,3,\ 3\,2\,3,\ 3\,3\,6,\ 3\,4\,6, \\ \quad\ 4\,1\,4,\ 4\,2\,3,\ 4\,3\,0,\ 4\,4\,0, \\ \quad\ 5\,1\,2,\ 5\,2\,0,\ 5\,3\,6,\ 5\,4\,3, \\ \quad\ 6\,1\,5,\ 6\,2\,4,\ 6\,3\,5,\ 6\,4\,4\rangle, \texttt{C}).\end{array}$$

Typical

$|\texttt{VARIABLES}| > 1$
$\text{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VALUES}| > 1$
$\text{range}(\texttt{VALUES.noccurrence}) > 1$
$\text{range}(\texttt{MATRIX.c}) > 1$
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$

Arg. properties

- Functional dependency: VALUES.noccurrence determined by VARIABLES.
- Functional dependency: COST determined by VARIABLES, VALUES and MATRIX.

Usage

A *classical utilisation* of the GLOBAL_CARDINALITY_WITH_COSTS constraint corresponds to the following assignment problem. We have a set of persons $\mathcal{P}$ as well as a set of jobs

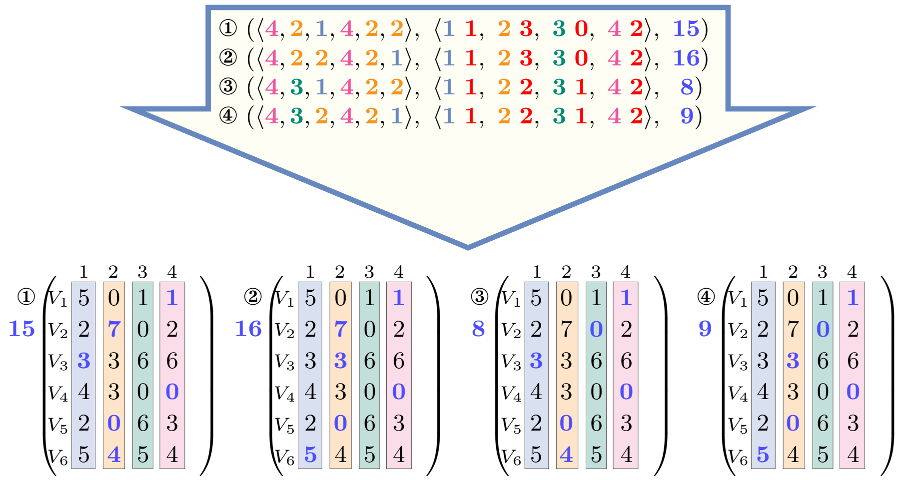


Figure 5.383: All solutions corresponding to the non ground example of the GLOBAL_CARDINALITY_WITH_COSTS constraint of the **All solutions** slot

$\mathcal{J}$ to perform. Each job requires a number of persons restricted to a specified interval. In addition each person p has to be assigned to one specific job taken from a subset $\mathcal{J}_p$ of $\mathcal{J}$. There is a cost C_{pj} associated with the fact that person p is assigned to job j. The previous problem is modelled with a single GLOBAL_CARDINALITY_WITH_COSTS constraint where the persons and the jobs respectively correspond to the items of the VARIABLES and VALUES collection.

The GLOBAL_CARDINALITY_WITH_COSTS constraint can also be used for modelling a conjunction ALLDIFFERENT$(\mathtt{X_1}, \mathtt{X_2}, \ldots, \mathtt{X_n})$ and $\alpha_1 \cdot \mathtt{X_1} + \alpha_2 \cdot \mathtt{X_2} + \cdots + \alpha_n \cdot \mathtt{X_n} = \mathtt{COST}$. For this purpose we set the domain of the noccurrence variables to $\{0, 1\}$ and the cost attribute c of a variable $\mathtt{X_i}$ and one of its potential value j to $\alpha_i \cdot \mathtt{j}$. In practice this can be used for the *magic squares* and the *magic hexagon* problems where all the α_i are set to 1.

Algorithm A filtering algorithm achieving arc-consistency independently on each side (i.e., the *greater than or equal to* side and the *less than or equal to* side) of the GLOBAL_CARDINALITY_WITH_COSTS constraint is described in [355, 357]. This algorithm assumes for each value a fixed minimum and maximum number of occurrences. If we rather have occurrence variables, the **Reformulation slot** explains how to also obtain some propagation from the cost variable back to the occurrence variables.

Reformulation Let n and m respectively denote the number of items of the VARIABLES and of the VALUES collections. Let $v_1, v_2, \ldots, v_m$ denote the values VALUES$[1]$.val, VALUES$[2]$.val, $\ldots$, VALUES$[m]$.val. In addition let $LINE_i$ (with $i \in [1, n]$) denote the values $\langle$MATRIX$[m \cdot (i-1) + 1]$.c, MATRIX$[m \cdot (i-1) + 2]$.c, $\ldots$, MATRIX$[m \cdot i]$.c$\rangle$, i.e., line i of the matrix MATRIX.

By introducing $2 \cdot n$ auxiliary variables $U_1, U_2, \ldots, U_n$ and $C_1, C_2, \ldots, C_n$, the GLOBAL_CARDINALITY_WITH_COSTS(VARIABLES, VALUES, MATRIX, COST) constraint can be expressed in term of the conjunction of one GLOBAL_CARDINALITY(VARIABLES, VALUES) constraint, $2 \cdot n$ ELEMENT constraints and one arithmetic constraint SUM_CTR.

For each variable V_i (with $i \in [1, |\texttt{VARIABLES}|]$) of the VARIABLES collection a first ELEMENT$(U_i, \langle v_1, v_2, \dots, v_m\rangle, V_i)$ constraint provides the correspondence between the variable V_i and the index of the value U_i to which it is assigned. A second ELEMENT$(U_i, LINE_i, C_i)$ links the previous index U_i to the cost C_i variable associated with variable V_i. Finally the total cost COST is equal to the sum $C_1 + C_2 + \cdots + C_n$.

In the context of the **Example** slot we get the following conjunction of constraints:

GLOBAL_CARDINALITY$(\langle 3, 3, 3, 6\rangle,$
$\langle \texttt{val} - 3\ \texttt{noccurrence} - 3,$
$\texttt{val} - 5\ \texttt{noccurrence} - 0,$
$\texttt{val} - 6\ \texttt{noccurrence} - 1\rangle),$
ELEMENT$(1, \langle 3, 5, 6\rangle, 3)$,
ELEMENT$(1, \langle 3, 5, 6\rangle, 3)$,
ELEMENT$(1, \langle 3, 5, 6\rangle, 3)$,
ELEMENT$(3, \langle 3, 5, 6\rangle, 6)$,
ELEMENT$(1, \langle 4, 1, 7\rangle, 4)$,
ELEMENT$(1, \langle 1, 0, 8\rangle, 1)$,
ELEMENT$(1, \langle 3, 2, 1\rangle, 3)$,
ELEMENT$(3, \langle 0, 0, 6\rangle, 6)$,
$14 = 4 + 1 + 3 + 6$.

We now show how to add implied constraints that can also propagate from the cost variable back to the occurrence variables. Let $O_1, O_2, \dots, O_m$ respectively denote the variables $\texttt{VALUES}[1].\texttt{noccurrence}, \texttt{VALUES}[2].\texttt{noccurrence}, \dots, \texttt{VALUES}[m].\texttt{noccurrence}$.
The idea is to get for each value v_i (with $i \in [1, m]$) an idea of its minimum and maximum contribution in the total cost COST that is linked to the number of times it is assigned to a variables of VARIABLES. E.g., if value v_i (with $i \in [1, m]$) is used twice, then the corresponding minimum (respectively maximum) contribution in the total cost COST will be at least equal to the sum of the two smallest (respectively largest) costs attached to row i. Let D_i (with $i \in [1, m]$) denotes the contribution that stems from the variables of VARIABLES that are assigned value v_i. For each value v_i (with $i \in [1, m]$) we create one ELEMENT constraint for linking $O_i + 1$ to the corresponding minimum contribution LOW_i. The table of that ELEMENT constraint has $n + 1$ entries, where entry j (with $j \in [0, n]$) corresponds to the sum of the j^{th} smallest entries of row i of the cost matrix MATRIX. Similarly we create for each value v_i (with $i \in [1, m]$) one ELEMENT constraint for linking $O_i + 1$ to the corresponding maximum contribution UP_i. The table of that ELEMENT constraint also has $n + 1$ entries, where entry j (with $j \in [0, n]$) corresponds to the sum of the j^{th} largest entries of row i of the cost matrix MATRIX.

In the context of the cost matrix of the **Example** slot we get the following conjunction of implied constraints:

$\texttt{COST} = D_1 + D_2 + D_3$,
$n = O_1 + O_2 + O_3$,
$P_1 = O_1 + 1$,
$P_2 = O_2 + 1$,
$P_3 = O_3 + 1$,
ELEMENT$(P_1, \langle 0, 0, 1, 4, 8\rangle, LOW_1)$,
ELEMENT$(P_2, \langle 0, 0, 0, 1, 3\rangle, LOW_2)$,
ELEMENT$(P_3, \langle 0, 1, 7, 14, 22\rangle, LOW_3)$,
ELEMENT$(P_1, \langle 0, 4, 7, 8, 8\rangle, UP_1)$,

ELEMENT$(P_2, \langle 0, 2, 3, 3, 3\rangle, UP_2)$,
ELEMENT$(P_3, \langle 0, 8, 15, 21, 22\rangle, UP_3)$,
$LOW_1 \leq D_1, D_1 \leq UP_1$,
$LOW_2 \leq D_2, D_2 \leq UP_2$,
$LOW_3 \leq D_3, D_3 \leq UP_3$.

Systems GLOBAL_CARDINALITY in **SICStus**.

See also **attached to cost variant:** GLOBAL_CARDINALITY *(*`cost` *associated with each* `variable`*,*`value` *pair removed)*.

common keyword: MINIMUM_WEIGHT_ALLDIFFERENT *(cost filtering constraint,weighted assignment)*, SUM_OF_WEIGHTS_OF_DISTINCT_VALUES, WEIGHTED_PARTIAL_ALLDIFF *(weighted assignment)*.

implies: GLOBAL_CARDINALITY.

Keywords **application area:** assignment.

constraint arguments: pure functional dependency.

filtering: cost filtering constraint.

heuristics: regret based heuristics, regret based heuristics in matrix problems.

modelling: cost matrix, scalar product, functional dependency.

problems: weighted assignment.

puzzles: magic square, magic hexagon.

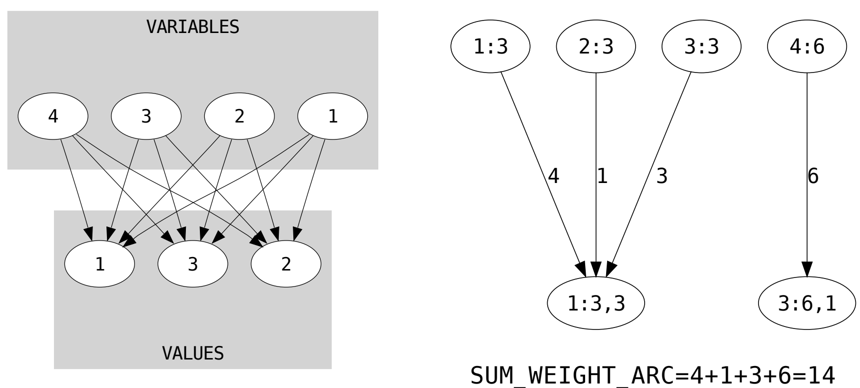
For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUES.val
Graph property(ies)	**NVERTEX** = VALUES.noccurrence

Arc input(s)	VARIABLES VALUES
Arc generator	*PRODUCT* $\mapsto$ collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	$\mathbf{SUM\_WEIGHT\_ARC}\left(\texttt{MATRIX}\left[\sum\left(\begin{array}{l}(\texttt{variables.key}-1)*|\texttt{VALUES}|,\\ \texttt{values.key}\end{array}\right)\right].\texttt{c}\right) = \texttt{COST}$

Graph model The first graph constraint forces each value of the VALUES collection to be taken by a specific number of variables of the VARIABLES collection. It is identical to the graph constraint used in the GLOBAL_CARDINALITY constraint. The second graph constraint expresses that the COST variable is equal to the sum of the elementary costs associated with each variable-value assignment. All these elementary costs are recorded in the MATRIX collection. More precisely, the cost c_{ij} is recorded in the attribute c of the $((i-1)\cdot|\texttt{VALUES})|+j)^{th}$ entry of the MATRIX collection. This is ensured by the increasing restriction that enforces the fact that the items of the MATRIX collection are sorted in lexicographically increasing order according to attributes i and j.

Parts (A) and (B) of Figure 5.384 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot.



(A) **(B)**

Figure 5.384: Initial and final graph of the GLOBAL_CARDINALITY_WITH_COSTS constraint

5.170 GLOBAL_CONTIGUITY

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [282]

Constraint GLOBAL_CONTIGUITY(VARIABLES)

Synonym CONTIGUITY.

Argument VARIABLES : collection(var−dvar)

Restrictions
required(VARIABLES, var)
$\mathtt{VARIABLES.var} \geq 0$
$\mathtt{VARIABLES.var} \leq 1$

Purpose Enforce all variables of the VARIABLES collection to be assigned value 0 or 1. In addition, all variables assigned to value 1 appear contiguously.

Example $(\langle 0, 1, 1, 0\rangle)$

The GLOBAL_CONTIGUITY constraint holds since the sequence 0 1 1 0 contains no more than one group of contiguous 1.

All solutions Figure 5.385 gives all solutions to the following non ground instance of the GLOBAL_CONTIGUITY constraint: $\mathtt{V}_1 \in [0, 1]$, $\mathtt{V}_2 \in [0, 1]$, $\mathtt{V}_3 = 1$, $\mathtt{V}_4 \in [0, 1]$, GLOBAL_CONTIGUITY$(\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4\rangle)$.

① $(\langle 0, 0, 1, 0\rangle)$
② $(\langle 0, 0, 1, 1\rangle)$
③ $(\langle 0, 1, 1, 0\rangle)$
④ $(\langle 0, 1, 1, 1\rangle)$
⑤ $(\langle 1, 1, 1, 0\rangle)$
⑥ $(\langle 1, 1, 1, 1\rangle)$

Figure 5.385: All solutions corresponding to the non ground example of the GLOBAL_CONTIGUITY constraint of the **All solutions** slot

Typical $|\mathtt{VARIABLES}| > 2$

Typical model
range(VARIABLES.var) > 1
ATLEAST(2, VARIABLES, 1)

Symmetry Items of `VARIABLES` can be reversed.

Arg. properties Contractible wrt. `VARIABLES`.

Usage The article [282] introducing this constraint refers to hardware configuration problems.

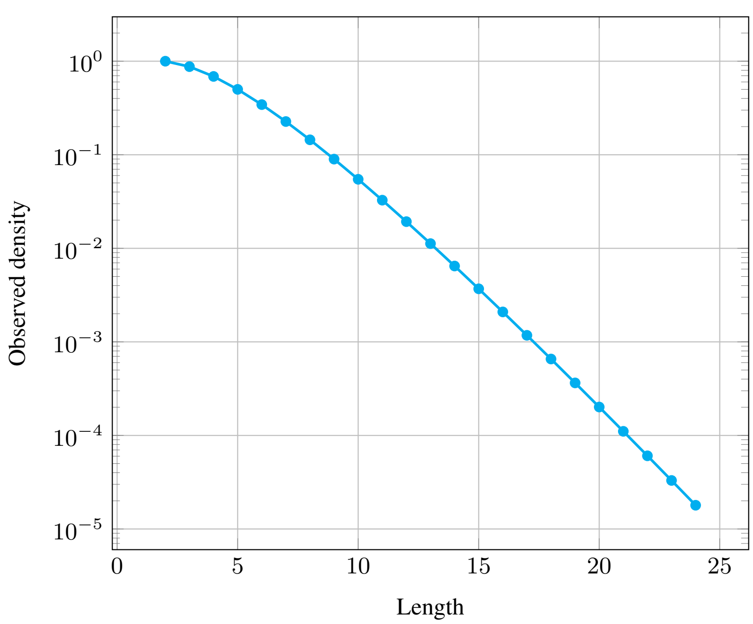
Algorithm A filtering algorithm for this constraint is described in [282].

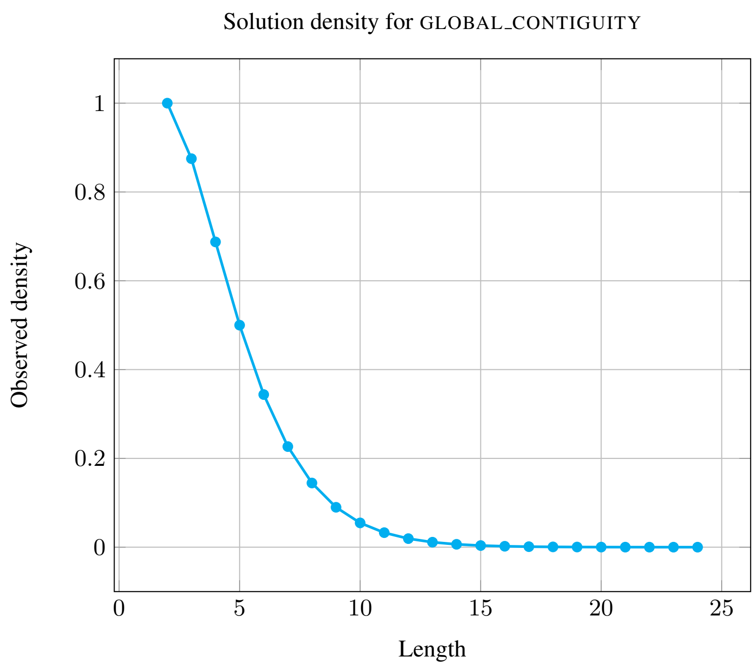
Counting

Length (n)	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Solutions	4	7	11	16	22	29	37	46	56	67	79	92	106	121	137	154	172	191	211	232	254	277	301

Number of solutions for GLOBAL_CONTIGUITY: domains 0..1

Solution density for GLOBAL_CONTIGUITY





See also

common keyword: GROUP, INFLEXION *(sequence)*.

implies: ALL_EQUAL_EXCEPT_0, CONSECUTIVE_VALUES, MULTI_GLOBAL_CONTIGUITY, NO_VALLEY.

related: ROOTS.

Keywords

characteristic of a constraint: convex, automaton, automaton without counters, automaton with same input symbol, reified automaton constraint.

combinatorial object: sequence.

constraint network structure: Berge-acyclic constraint network.

filtering: arc-consistency.

final graph structure: connected component.

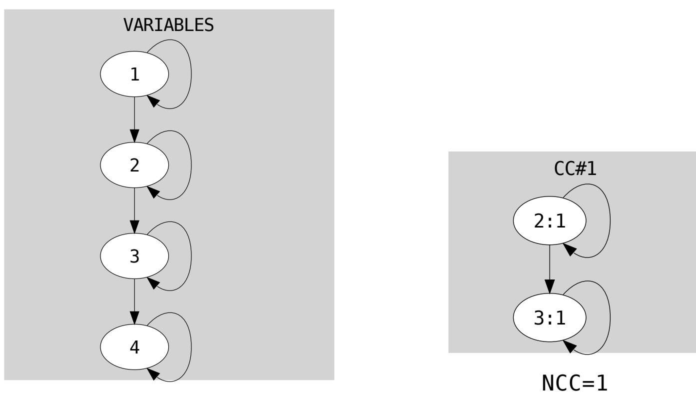
Cond. implications

GLOBAL_CONTIGUITY(VARIABLES)
with $|\texttt{VARIABLES}| > 2$
implies SOME_EQUAL(VARIABLES).

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2) $LOOP \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• variables1.var = variables2.var • variables1.var = 1
Graph property(ies)	**NCC** ≤ 1

Graph model Each connected component of the final graph corresponds to one set of contiguous variables that all take value 1.

Parts (A) and (B) of Figure 5.386 respectively show the initial and final graph associated with the **Example** slot. The GLOBAL_CONTIGUITY constraint holds since the final graph does not contain more than one connected component. This connected component corresponds to 2 contiguous variables that are both assigned to 1.



(A) **(B)**

Figure 5.386: Initial and final graph of the GLOBAL_CONTIGUITY constraint

Automaton Figure 5.387 depicts the automaton associated with the GLOBAL_CONTIGUITY constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a signature variable that is equal to $\mathtt{VAR}_i$. There is no signature constraint.

STATE SEMANTICS

s	: in only 0 mode	(0^*)
m	: in stretch of 1 mode	(1^+)
z	: in 0 mode (after stretch of 1)	(0^+)

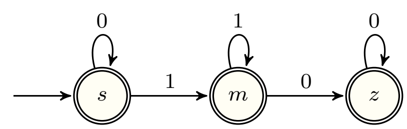


Figure 5.387: Automaton of the GLOBAL_CONTIGUITY constraint

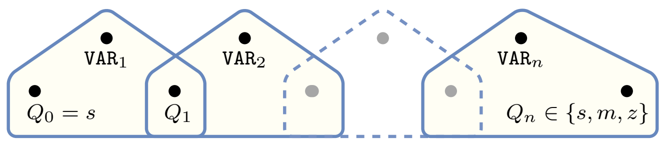


Figure 5.388: Hypergraph of the reformulation corresponding to the automaton of the GLOBAL_CONTIGUITY constraint

5.171 GOLOMB

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Inspired by [207].

Constraint GOLOMB(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions
required(VARIABLES, var)
VARIABLES.var ≥ 0
STRICTLY_INCREASING(VARIABLES)

Purpose Given a strictly increasing sequence $X_1, X_2, \ldots, X_n$, enforce all differences $X_i - X_j$ between two variables X_i and X_j $(i > j)$ to be distinct.

Example $(\langle 0, 1, 4, 6\rangle)$

Figure 5.389 gives a graphical interpretation of the solution given in the example in term of a graph: each vertex corresponds to a value of $\langle 0, 1, 4, 6\rangle$, while each arc depicts a difference between two values. The GOLOMB constraint holds since one can note that these differences 1, 4, 6, 3, 5, 2 are all-distinct.

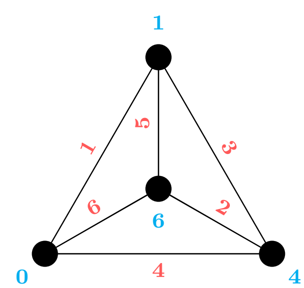


Figure 5.389: Graphical representation of the solution 0, 1, 4, 6 (differences are displayed in light red and are pairwise distinct).

Typical $|\texttt{VARIABLES}| > 2$

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties Contractible wrt. VARIABLES.

Usage This constraint refers to the Golomb ruler problem. We quote the definition from [391]: "A Golomb ruler is a set of integers (marks) $a_1 < \cdots < a_k$ such that all the differences $a_i - a_j$ $(i > j)$ are distinct".

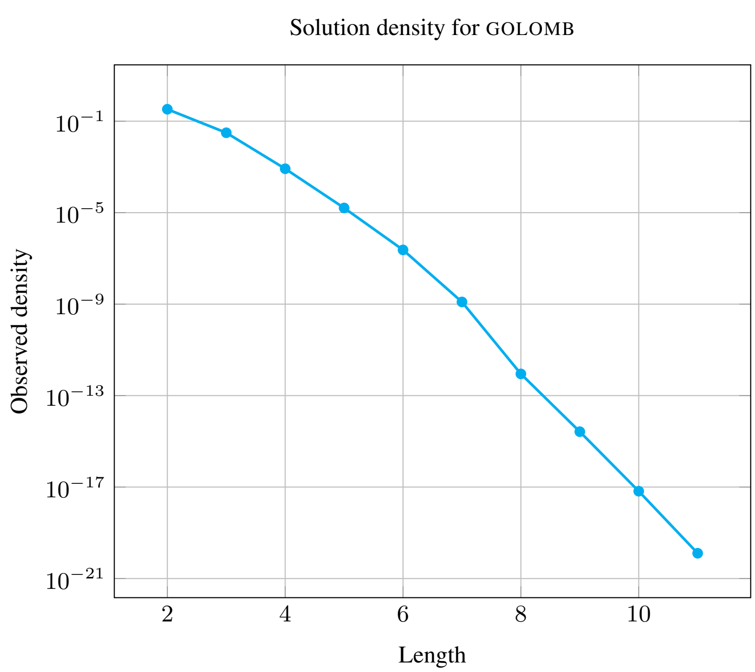
Remark Different constraints models for the Golomb ruler problem were presented in [404].

Algorithm At a first glance, one could think that, because it looks so similar to the ALLDIFFERENT constraint, we could have a perfect polynomial filtering algorithm. However this is not true since one retrieves the *same* variable in different vertices of the graph. This leads to the fact that one has incompatible arcs in the bipartite graph (the two classes of vertices correspond to the pair of variables and to the fact that the difference between two pairs of variables takes a specific value). However one can still reuse a similar filtering algorithm as for the ALLDIFFERENT constraint, but this will not lead to perfect pruning.

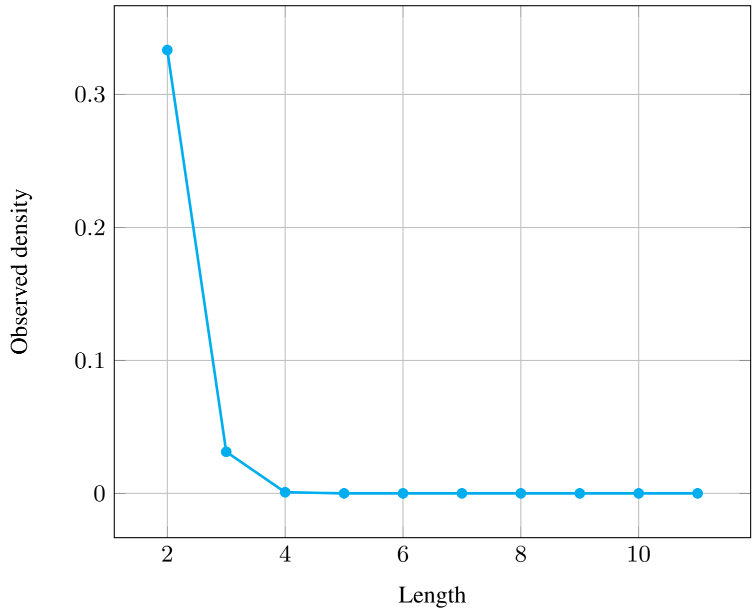
Counting

Length (n)	2	3	4	5	6	7	8	9	10	11
Solutions	3	2	2	4	8	10	2	2	2	4

Number of solutions for GOLOMB: domains $0..k$



Solution density for GOLOMB



See also **common keyword:** ALLDIFFERENT *(all different)*.

implies: STRICTLY_INCREASING.

Keywords **characteristic of a constraint:** disequality, difference, all different, derived collection.

puzzles: Golomb ruler.

Cond. implications

- GOLOMB(VARIABLES)
  implies INCREASING_NVALUE(NVAL, VARIABLES)
  when NVAL =nval(VARIABLES.var).
- GOLOMB(VARIABLES)
  implies SOFT_ALLDIFFERENT_CTR(C, VARIABLES).

Derived Collection

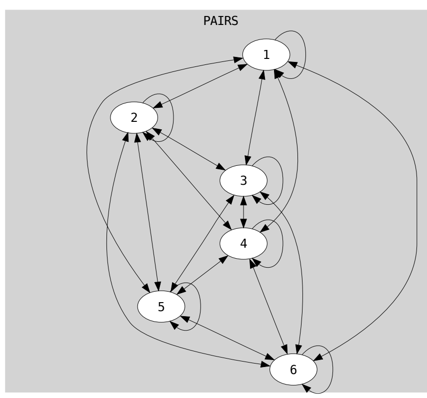
$$\texttt{col}\left(\begin{array}{l}\texttt{PAIRS}-\texttt{collection}(\texttt{x}-\texttt{dvar},\texttt{y}-\texttt{dvar}),\\ [>-\texttt{item}(\texttt{x}-\texttt{VARIABLES.var},\texttt{y}-\texttt{VARIABLES.var})]\end{array}\right)$$

Arc input(s)	PAIRS
Arc generator	$CLIQUE \mapsto$ collection(pairs1, pairs2)
Arc arity	2
Arc constraint(s)	pairs1.y − pairs1.x = pairs2.y − pairs2.x
Graph property(ies)	**MAX_NSCC** ≤ 1

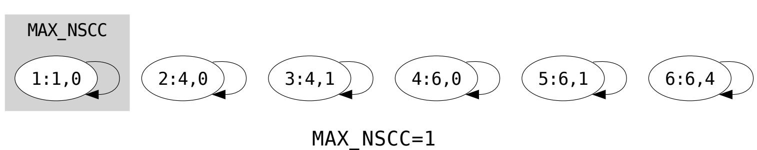
Graph model

When applied on the collection of items ⟨VAR1, VAR2, VAR3, VAR4⟩, the generator of derived collection generates the following collection of items: ⟨VAR2 VAR1, VAR3 VAR1, VAR3 VAR2, VAR4 VAR1, VAR4 VAR2, VAR4 VAR3⟩. Note that we use a binary arc constraint between two vertices and that this binary constraint involves four variables.

Parts (A) and (B) of Figure 5.390 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph. The constraint holds since all the strongly connected components have at most one vertex: the differences $1, 2, 3, 4, 5, 6$ that one can construct from the values $0, 1, 4, 6$ assigned to the variables of the VARIABLES collection are all-distinct.



(A)



(B)

Figure 5.390: Initial and final graph of the GOLOMB constraint

5.172 GRAPH_CROSSING

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint GRAPH_CROSSING(NCROSS, NODES)

Synonyms CROSSING, NCROSS.

Arguments

NCROSS : dvar
NODES : collection(succ−dvar, x−int, y−int)

Restrictions

NCROSS ≥ 0
required(NODES, [succ, x, y])
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

Purpose

NCROSS is the number of proper intersections between line segments, where each line segment is an arc of the directed graph defined by the arc linking a node and its unique successor.

Example

$$\left(2,\left\langle\begin{array}{lll} \texttt{succ}-1 & \texttt{x}-4 & \texttt{y}-7, \\ \texttt{succ}-1 & \texttt{x}-2 & \texttt{y}-5, \\ \texttt{succ}-1 & \texttt{x}-7 & \texttt{y}-6, \\ \texttt{succ}-2 & \texttt{x}-1 & \texttt{y}-2, \\ \texttt{succ}-3 & \texttt{x}-2 & \texttt{y}-2, \\ \texttt{succ}-2 & \texttt{x}-5 & \texttt{y}-3, \\ \texttt{succ}-3 & \texttt{x}-8 & \texttt{y}-2, \\ \texttt{succ}-9 & \texttt{x}-6 & \texttt{y}-2, \\ \texttt{succ}-10 & \texttt{x}-10 & \texttt{y}-6, \\ \texttt{succ}-8 & \texttt{x}-10 & \texttt{y}-1 \end{array}\right\rangle\right)$$

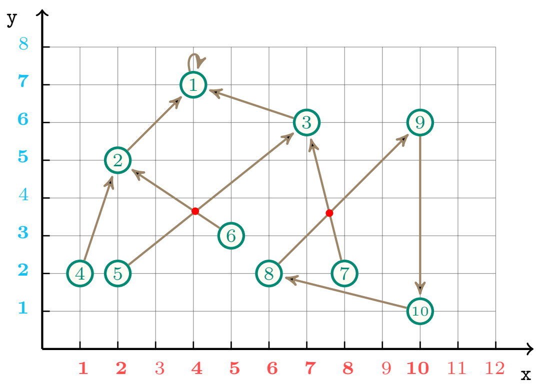
Figure 5.391 shows the line segments associated with the NODES collection. One can note the following line segments intersection:

- Arcs $8 \rightarrow 9$ and $7 \rightarrow 3$ cross,
- Arcs $5 \rightarrow 3$ and $6 \rightarrow 2$ cross also.

Consequently, the GRAPH_CROSSING constraint holds since its first argument NCROSS is set to 2.

Typical

|NODES| > 1
range(NODES.succ) > 1
range(NODES.x) > 1
range(NODES.y) > 1



NODES

1:	succ − 1	x − 4	y − 7
2:	succ − 1	x − 2	y − 5
3:	succ − 1	x − 7	y − 6
4:	succ − 2	x − 1	y − 2
5:	succ − 3	x − 2	y − 2
6:	succ − 2	x − 5	y − 3
7:	succ − 3	x − 8	y − 2
8:	succ − 9	x − 6	y − 2
9:	succ − 10	x − 10	y − 6
10:	succ − 8	x − 10	y − 1

Figure 5.391: Illustration of the **Example** slot: a graph covering with 2 line segments intersections in red (NCROSS = 2)

Symmetries

- Attributes of NODES are permutable w.r.t. permutation (succ) (x, y) (*permutation applied to all items*).
- One and the same constant can be added to the x attribute of all items of NODES.
- One and the same constant can be added to the y attribute of all items of NODES.

Arg. properties

Functional dependency: NCROSS determined by NODES.

Usage

This is a general crossing constraint that can be used in conjunction with one graph covering constraint such as CYCLE, TREE or MAP. In many practical problems ones want not only to cover a graph with specific patterns but also to avoid too much crossing between the arcs of the final graph.

Remark

We did not give a specific crossing constraint for each graph covering constraint. We feel that it is better to start first with a more general constraint before going in the specificity of the pattern that is used for covering the graph.

See also

common keyword: CROSSING (*line segments intersection*), CYCLE, MAP, TREE (*graph constraint, graph partitioning constraint*), TWO_LAYER_EDGE_CROSSING (*line segments intersection*).

Keywords

constraint arguments: pure functional dependency.

constraint type: graph constraint, graph partitioning constraint.

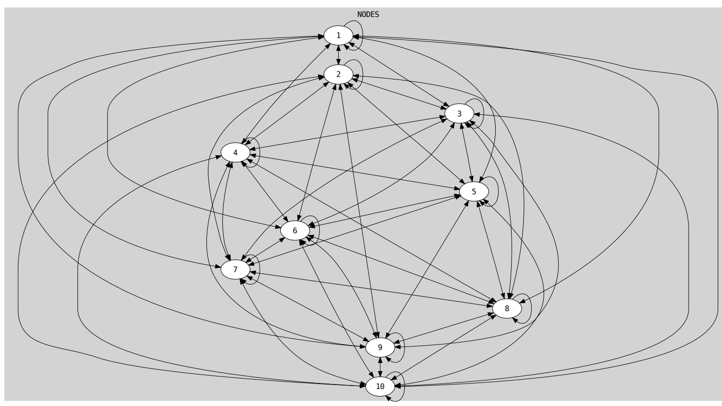
geometry: geometrical constraint, line segments intersection.

modelling: functional dependency.

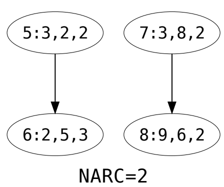
Arc input(s)	NODES
Arc generator	$CLIQUE(<) \mapsto$ collection(n1, n2)
Arc arity	2
Arc constraint(s)	• $\texttt{max}(\texttt{n1.x}, \texttt{NODES}[\texttt{n1.succ}].\texttt{x}) \geq \texttt{min}(\texttt{n2.x}, \texttt{NODES}[\texttt{n2.succ}].\texttt{x})$ • $\texttt{max}(\texttt{n2.x}, \texttt{NODES}[\texttt{n2.succ}].\texttt{x}) \geq \texttt{min}(\texttt{n1.x}, \texttt{NODES}[\texttt{n1.succ}].\texttt{x})$ • $\texttt{max}(\texttt{n1.y}, \texttt{NODES}[\texttt{n1.succ}].\texttt{y}) \geq \texttt{min}(\texttt{n2.y}, \texttt{NODES}[\texttt{n2.succ}].\texttt{y})$ • $\texttt{max}(\texttt{n2.y}, \texttt{NODES}[\texttt{n2.succ}].\texttt{y}) \geq \texttt{min}(\texttt{n1.y}, \texttt{NODES}[\texttt{n1.succ}].\texttt{y})$ • $\begin{array}{l} (\texttt{n2.x} - \texttt{NODES}[\texttt{n1.succ}].\texttt{x}) * (\texttt{NODES}[\texttt{n1.succ}].\texttt{y} - \texttt{n1.y}) - \\ (\texttt{NODES}[\texttt{n1.succ}].\texttt{x} - \texttt{n1.x}) * (\texttt{n2.y} - \texttt{NODES}[\texttt{n1.succ}].\texttt{y}) \end{array} \neq 0$ • $\begin{array}{l} (\texttt{NODES}[\texttt{n2.succ}].\texttt{x} - \texttt{NODES}[\texttt{n1.succ}].\texttt{x}) * (\texttt{n2.y} - \texttt{n1.y}) - \\ (\texttt{n2.x} - \texttt{n1.x}) * (\texttt{NODES}[\texttt{n2.succ}].\texttt{y} - \texttt{NODES}[\texttt{n1.succ}].\texttt{y}) \end{array} \neq 0$ • $\texttt{sign}\left(\begin{array}{l} \prod\left(\begin{array}{l}\texttt{n2.x} - \texttt{NODES}[\texttt{n1.succ}].\texttt{x},\\ \texttt{NODES}[\texttt{n1.succ}].\texttt{y} - \texttt{n1.y}\end{array}\right) - \\ \prod\left(\begin{array}{l}\texttt{NODES}[\texttt{n1.succ}].\texttt{x} - \texttt{n1.x},\\ \texttt{n2.y} - \texttt{NODES}[\texttt{n1.succ}].\texttt{y}\end{array}\right)\end{array}\right) \neq \texttt{sign}\left(\begin{array}{l} \prod\left(\begin{array}{l}\texttt{NODES}[\texttt{n2.succ}].\texttt{x} - \texttt{NODES}[\texttt{n1.succ}].\texttt{x},\\ \texttt{n2.y} - \texttt{n1.y}\end{array}\right) - \\ \prod\left(\begin{array}{l}\texttt{n2.x} - \texttt{n1.x},\\ \texttt{NODES}[\texttt{n2.succ}].\texttt{y} - \texttt{NODES}[\texttt{n1.succ}].\texttt{y}\end{array}\right)\end{array}\right)$
Graph property(ies)	**NARC** = NCROSS

Graph model Each node is described by its coordinates x and y, and by its successor succ in the final covering. Note that the co-ordinates are initially fixed. We use the arc generator $CLIQUE(<)$ in order to avoid counting twice the same line segment crossing.

Parts (A) and (B) of Figure 5.392 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Each arc of the final graph corresponds to a proper intersection between two line segments.



(A)



(B)

Figure 5.392: Initial and final graph of the GRAPH_CROSSING constraint

5.173 GRAPH_ISOMORPHISM

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin [288]

Constraint GRAPH_ISOMORPHISM(NODES_PATTERN, NODES_TARGET, FUNCTION)

Arguments

NODES_PATTERN	:	collection(index−int, succ−sint)
NODES_TARGET	:	collection(index−int, succ−sint)
FUNCTION	:	collection(image−dvar)

Restrictions

required(NODES_PATTERN, [index, succ])
NODES_PATTERN.index ≥ 1
NODES_PATTERN.index ≤ |NODES_PATTERN|
distinct(NODES_PATTERN, index)
NODES_PATTERN.succ ≥ 1
NODES_PATTERN.succ ≤ |NODES_PATTERN|
required(NODES_TARGET, [index, succ])
NODES_TARGET.index ≥ 1
NODES_TARGET.index ≤ |NODES_TARGET|
distinct(NODES_TARGET, index)
NODES_TARGET.succ ≥ 1
NODES_TARGET.succ ≤ |NODES_TARGET|
|NODES_TARGET| = |NODES_PATTERN|
required(FUNCTION, [image])
FUNCTION.image ≥ 1
FUNCTION.image ≤ |NODES_TARGET|
distinct(FUNCTION, image)
|FUNCTION| = |NODES_PATTERN|

Purpose

Given two directed graphs PATTERN and TARGET enforce a one to one correspondence, defined by the function FUNCTION, between the vertices of the graph PATTERN and the vertices of the graph TARGET so that:

1. if there is an arc from u to v in the graph PATTERN, then there is also an arc from the image of u to the image of v in the graph TARGET,
2. if there is no arc from u to v in the graph PATTERN, then there is also no arc from the image of u to the image of v in the graph TARGET.

Both, the PATTERN and TARGET are fixed, and the vertices of both graphs are respectively defined by the two collections of vertices NODES_PATTERN and NODES_TARGET.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\{2,4\},\\ \texttt{index}-2 & \texttt{succ}-\{1,3,4\},\\ \texttt{index}-3 & \texttt{succ}-\emptyset,\\ \texttt{index}-4 & \texttt{succ}-\emptyset\end{array}\right\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\emptyset,\\ \texttt{index}-2 & \texttt{succ}-\{1,3,4\},\\ \texttt{index}-3 & \texttt{succ}-\emptyset,\\ \texttt{index}-4 & \texttt{succ}-\{1,2\}\end{array}\right\rangle,\\ \langle 4,2,3,1\rangle\end{array}\right)$$

Figure 5.393 gives the pattern (see Part (A)) and target graph (see Part (B)) of the **Example** slot as well as the one to one correspondence (see Part (C)) between the pattern graph and the target graph. The GRAPH_ISOMORPHISM constraint since the pattern and target graphs have the same number of vertices and arcs and since:

- To the arc from vertex 1 to vertex 4 in the pattern graph corresponds the arc from vertex 4 to 1 in the target graph.
- To the arc from vertex 1 to vertex 2 in the pattern graph corresponds the arc from vertex 4 to 2 in the target graph.
- To the arc from vertex 2 to vertex 1 in the pattern graph corresponds the arc from vertex 2 to 4 in the target graph.
- To the arc from vertex 2 to vertex 4 in the pattern graph corresponds the arc from vertex 2 to 1 in the target graph.
- To the arc from vertex 2 to vertex 3 in the pattern graph corresponds the arc from vertex 2 to 3 in the target graph.

Typical

$|\texttt{NODES\_PATTERN}| > 1$

Symmetries

- Items of NODES_PATTERN are permutable.
- Items of NODES_TARGET are permutable.

Algorithm

A constraint approach is described in [406].

See also

related: SUBGRAPH_ISOMORPHISM.

Keywords

constraint arguments: constraint involving set variables.

constraint type: predefined constraint, graph constraint.

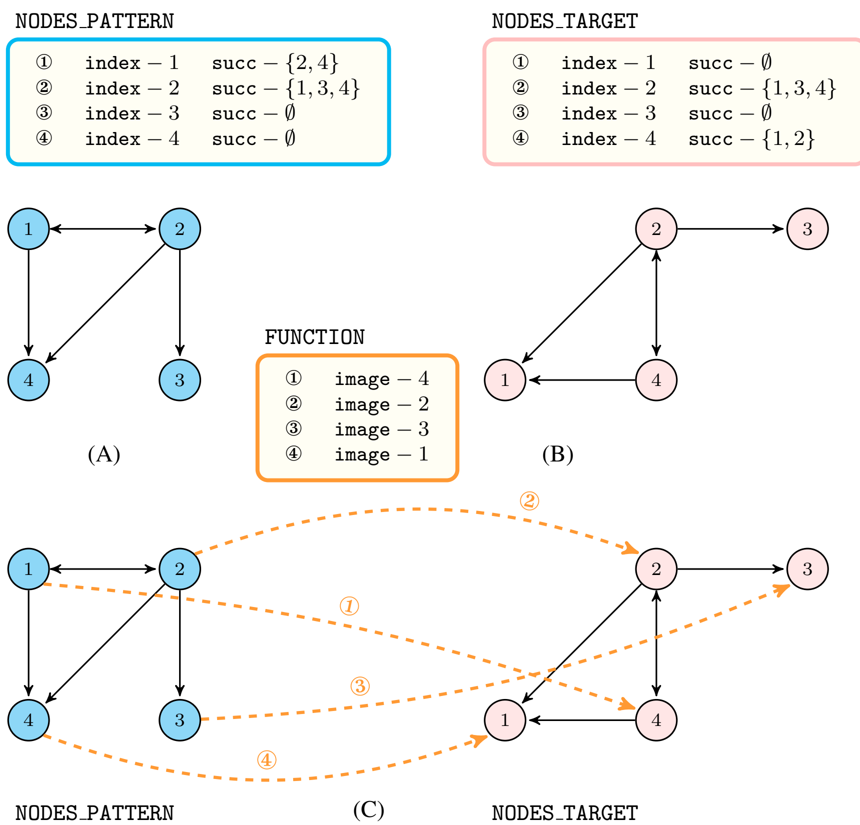


Figure 5.393: Illustration of the **Example** slot: (A) The pattern graph, (B) the target graph and (C) the correspondence, denoted by thick dashed arcs, between the vertices of the pattern graph and the vertices of the target graph

5.174 GROUP

DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint

$$\text{GROUP}\left(\begin{array}{l}\texttt{NGROUP},\\ \texttt{MIN\_SIZE},\\ \texttt{MAX\_SIZE},\\ \texttt{MIN\_DIST},\\ \texttt{MAX\_DIST},\\ \texttt{NVAL},\\ \texttt{VARIABLES},\\ \texttt{VALUES}\end{array}\right)$$

Arguments

NGROUP	:	dvar
MIN_SIZE	:	dvar
MAX_SIZE	:	dvar
MIN_DIST	:	dvar
MAX_DIST	:	dvar
NVAL	:	dvar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int)

Restrictions

$\texttt{NGROUP} \geq 0$
$\texttt{MIN\_SIZE} \geq 0$
$\texttt{MAX\_SIZE} \geq \texttt{MIN\_SIZE}$
$\texttt{MIN\_DIST} \geq 0$
$\texttt{MAX\_DIST} \geq \texttt{MIN\_DIST}$
$\texttt{MAX\_DIST} \leq |\texttt{VARIABLES}|$
$\texttt{NVAL} \geq \texttt{MAX\_SIZE}$
$\texttt{NVAL} \geq \texttt{NGROUP}$
$\texttt{NVAL} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose

Let n be the number of variables of the collection VARIABLES. Let $X_i, X_{i+1}, \ldots, X_j$ ($1 \leq i \leq j \leq n$) be consecutive variables of the collection of variables VARIABLES such that all the following conditions simultaneously apply:

- All variables $X_i, \ldots, X_j$ take their values in the set of values VALUES,
- $i = 1$ or X_{i-1} does not take a value in VALUES,
- $j = n$ or X_{j+1} does not take a value in VALUES.

We call such a sequence of variables a *group*. Similarly an *anti-group* is a maximum sequence of variables that are not assigned any value from VALUES. The constraint GROUP is true if all the following conditions hold:

- There are exactly NGROUP groups of variables,
- MIN_SIZE is the number of variables of the smallest group,
- MAX_SIZE is the number of variables of the largest group,
- MIN_DIST is the number of variables of the smallest anti-group,
- MAX_DIST is the number of variables of the largest anti-group,
- NVAL is the number of variables that take their values in the set of values VALUES.

Example

$(2, 1, 2, 2, 4, 3, \langle 2, 8, 1, 7, 4, 5, 1, 1, 1\rangle, \langle 0, 2, 4, 6, 8\rangle)$

Given the fact that groups are formed by even values in $\{0, 2, 4, 6, 8\}$ (i.e., values expressed by the VALUES collection), the GROUP constraint holds since:

- Its first argument, NGROUP, is set to value 2 since the sequence $2\ 8\ 1\ 7\ 4\ 5\ 1\ 1\ 1$ contains two groups of even values (i.e., group $2\ 8$ and group 4).
- Its second argument, MIN_SIZE, is set to value 1 since the smallest group of even values involves only a single value (i.e., value 4).
- Its third argument, MAX_SIZE, is set to value 2 since the largest group of even values involves two values (i.e., group $2\ 8$).
- Its fourth argument, MIN_DIST, is set to value 2 since the smallest anti-group involves two values (i.e., anti-group $1\ 7$).
- Its fifth argument, MAX_DIST, is set to value 4 since the largest anti-group involves four values (i.e., anti-group $5\ 1\ 1\ 1$).
- Its sixth argument, NVAL, is set to value 3 since the total number of even values of the sequence $2\ 8\ 1\ 7\ 4\ 5\ 1\ 1\ 1$ is equal to 3 (i.e., values 2, 8 and 4).

All solutions

Figure 5.394 gives all solutions to the following non ground instance of the GROUP constraint: NGROUP $\in [2, 3]$, MIN_SIZE $\in [3, 4]$, MAX_SIZE $\in [3, 5]$, MIN_DIST $\in [1, 2]$, MAX_DIST $\in [1, 2]$, NVAL $\in [5, 6]$, $\mathtt{V}_1 \in [0, 1]$, $\mathtt{V}_2 \in [0, 1]$, $\mathtt{V}_3 \in [0, 1]$, $\mathtt{V}_4 \in [0, 1]$, $\mathtt{V}_5 \in [0, 1]$, $\mathtt{V}_6 \in [0, 1]$, $\mathtt{V}_7 \in [0, 1]$, $\mathtt{V}_8 \in [0, 1]$, $\mathtt{V}_9 \in [0, 1]$, GROUP(NGROUP, MIN_SIZE, MAX_SIZE, MIN_DIST, MAX_DIST, NVAL, $\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4, \mathtt{V}_5, \mathtt{V}_6, \mathtt{V}_7, \mathtt{V}_8, \mathtt{V}_9\rangle$, $\langle 1\rangle$),

① $(\mathbf{2},\mathbf{3},\mathbf{3},\mathbf{1},\mathbf{2},\mathbf{6},\langle\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1}\rangle,\langle\mathbf{1}\rangle)$
② $(\mathbf{2},\mathbf{3},\mathbf{3},\mathbf{1},\mathbf{2},\mathbf{6},\langle\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1}\rangle,\langle\mathbf{1}\rangle)$
③ $(\mathbf{2},\mathbf{3},\mathbf{3},\mathbf{1},\mathbf{1},\mathbf{6},\langle\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0}\rangle,\langle\mathbf{1}\rangle)$
④ $(\mathbf{2},\mathbf{3},\mathbf{3},\mathbf{1},\mathbf{2},\mathbf{6},\langle\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0}\rangle,\langle\mathbf{1}\rangle)$
⑤ $(\mathbf{2},\mathbf{3},\mathbf{3},\mathbf{1},\mathbf{2},\mathbf{6},\langle\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{0}\rangle,\langle\mathbf{1}\rangle)$

Figure 5.394: All solutions corresponding to the non ground example of the GROUP constraint of the **All solutions** slot

Typical

`NGROUP` > 0
`MIN_SIZE` > 0
`MAX_SIZE` $>$ `MIN_SIZE`
`MIN_DIST` > 0
`MAX_DIST` $>$ `MIN_DIST`
`MAX_DIST` $<$ |`VARIABLES`|
`NVAL` $>$ `MAX_SIZE`
`NVAL` $>$ `NGROUP`
`NVAL` $<$ |`VARIABLES`|
|`VARIABLES`| > 1
range(`VARIABLES.var`) > 1
|`VALUES`| > 0
|`VARIABLES`| $>$ |`VALUES`|

Symmetries

- Items of `VARIABLES` can be reversed.
- Items of `VALUES` are permutable.
- An occurrence of a value of `VARIABLES.var` that belongs to `VALUES.val` (resp. does not belong to `VALUES.val`) can be replaced by any other value in `VALUES.val` (resp. not in `VALUES.val`).

Arg. properties

- Functional dependency: `NGROUP` determined by `VARIABLES` and `VALUES`.
- Functional dependency: `MIN_SIZE` determined by `VARIABLES` and `VALUES`.
- Functional dependency: `MAX_SIZE` determined by `VARIABLES` and `VALUES`.
- Functional dependency: `MIN_DIST` determined by `VARIABLES` and `VALUES`.
- Functional dependency: `MAX_DIST` determined by `VARIABLES` and `VALUES`.
- Functional dependency: `NVAL` determined by `VARIABLES` and `VALUES`.

Usage

A typical use of the GROUP constraint in the context of timetabling is as follow: The value of the i^{th} variable of the `VARIABLES` collection corresponds to the type of shift (i.e., night, morning, afternoon, rest) performed by a specific person on day i. A complete period of work is represented by the variables of the `VARIABLES` collection. In this context the GROUP constraint expresses for a person:

- The number of periods of consecutive night-shift during a complete period of work.
- The total number of night-shift during a complete period of work.
- The maximum number of allowed consecutive night-shift.
- The minimum number of days, which do not correspond to a night-shift, between two consecutive sequences of night-shift.

Remark

For this constraint we use the possibility to express directly more than one constraint on the parameters of the final graph we want to obtain. For more propagation, it is crucial to keep this in a single constraint, since strong relations relate the different parameters of a graph. This constraint is very similar to the GROUP constraint introduced in **CHIP**, except that here, the `MIN_DIST` and `MAX_DIST` constraints apply also for the two borders: we cannot start or end with a group of k consecutive variables that take their values outside `VALUES` and such that k is less than `MIN_DIST` or k is greater than `MAX_DIST`.

See also

common keyword: CHANGE_CONTINUITY, FULL_GROUP *(timetabling constraint,sequence)*, GLOBAL_CONTIGUITY *(sequence)*, GROUP_SKIP_ISOLATED_ITEM *(timetabling constraint,sequence)*, MULTI_GLOBAL_CONTIGUITY *(sequence)*, PATTERN, STRETCH_CIRCUIT *(timetabling constraint)*, STRETCH_PATH *(timetabling constraint,sequence)*.

shift of concept: CONSECUTIVE_GROUPS_OF_ONES.

used in graph description: IN, NOT_IN.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: alpha-acyclic constraint network(2), alpha-acyclic constraint network(3).

constraint type: timetabling constraint.

filtering: glue matrix.

final graph structure: connected component, vpartition, consecutive loops are connected.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2) $LOOP \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• IN(variables1.var, VALUES) • IN(variables2.var, VALUES)
Graph property(ies)	• **NCC**= NGROUP • **MIN_NCC**= MIN_SIZE • **MAX_NCC**= MAX_SIZE • **NVERTEX**= NVAL

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2) $LOOP \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• NOT_IN(variables1.var, VALUES) • NOT_IN(variables2.var, VALUES)
Graph property(ies)	• **MIN_NCC**= MIN_DIST • **MAX_NCC**= MAX_DIST

Graph model We use two graph constraints for modelling the GROUP constraint: a first one for specifying the constraints on NGROUP, MIN_SIZE, MAX_SIZE and NVAL, and a second one for stating the constraints on MIN_DIST and MAX_DIST. In order to generate the initial graph related to the first graph constraint we use:

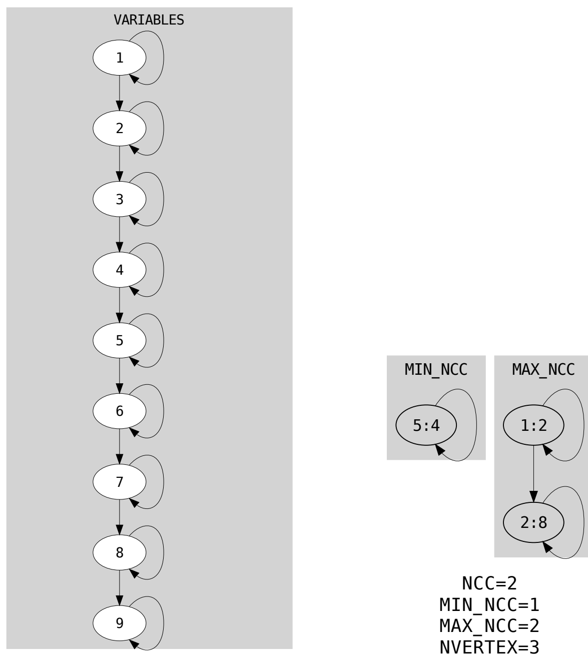
- The arc generators $PATH$ and $LOOP$,
- The binary constraint variables1.var $\in$ VALUES $\wedge$ variables2.var $\in$ VALUES.

On the first graph constraint of the **Example** slot this produces an initial graph depicted in part (A) of Figure 5.395. We use $PATH$ $LOOP$ and the binary constraint variables1.var $\in$ VALUES $\wedge$ variables2.var $\in$ VALUES in order to catch the two following situations:

- A binary constraint has to be used in order to get the notion of group: *Consecutive* variables that take their values in VALUES.
- If we only use $PATH$ then we would lose the groups that are composed from a single variable since the predecessor and the successor arc would be destroyed; this is why we use also the $LOOP$ arc generator.

Part (B) of Figure 5.395 shows the final graph associated with the first graph constraint of the **Example** slot. Since we use the **NVERTEX** graph property, the vertices of the final graph are stressed in bold. In addition, since we use the **MIN_NCC** and the **MAX_NCC** graph properties, we also show the smallest and largest connected components of the final graph.

The GROUP constraint of the **Example** slot holds since:



(A) **(B)**

Figure 5.395: Initial and final graph of the GROUP constraint

- The final graph of the first graph constraint has two connected components. Therefore the number of groups NGROUP is equal to two.
- The number of vertices of the smallest connected component of the final graph of the first graph constraint is equal to 1. Therefore MIN_SIZE is equal to 1.
- The number of vertices of the largest connected component of the final graph of the first graph constraint is equal to 2. Therefore MAX_SIZE is equal to 2.
- The number of vertices of the smallest connected component of the final graph of the second graph constraint is equal to 2. Therefore MIN_DIST is equal to 2.
- The number of vertices of the largest connected component of the final graph of the second graph constraint is equal to 4. Therefore MAX_DIST is equal to 4.
- The number of vertices of the final graph of the first graph constraint is equal to three. Therefore NVAL is equal to 3.

Automaton Figures 5.396, 5.398, 5.401, 5.403, 5.405 and 5.407 depict the different automata associated with the GROUP constraint. For the automata that respectively compute NGROUP, MIN_SIZE, MAX_SIZE, MIN_DIST, MAX_DIST and NVAL we have a 0-1 signature variable $\mathtt{S}_i$ for each variable $\mathtt{VAR}_i$ of the collection VARIABLES. The following signature constraint links $\mathtt{VAR}_i$ and $\mathtt{S}_i$: $\mathtt{VAR}_i \in \mathtt{VALUES} \Leftrightarrow \mathtt{S}_i$.

STATE SEMANTICS

s	: not in VALUES mode	$(\notin^*)$
i	: in VALUES mode	$(\in^+)$

	$s\ (\notin^*)$	$i\ (\in^+)$
$s\ (\notin^*)$	$\overrightarrow{C} + \overleftarrow{C}$	$\overrightarrow{C} + \overleftarrow{C}$
$i\ (\in^+)$	$\overrightarrow{C} + \overleftarrow{C}$	$\overrightarrow{C} - 1 + \overleftarrow{C}$

Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

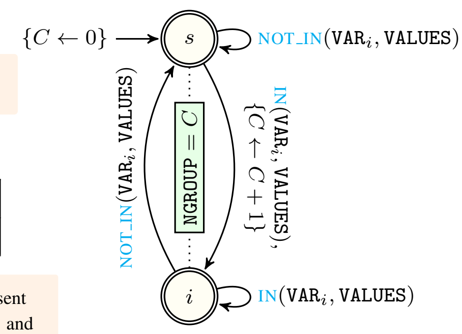


Figure 5.396: Automaton for the NGROUP argument of the GROUP constraint and its glue matrix, where counter C is the number of groups encountered so far

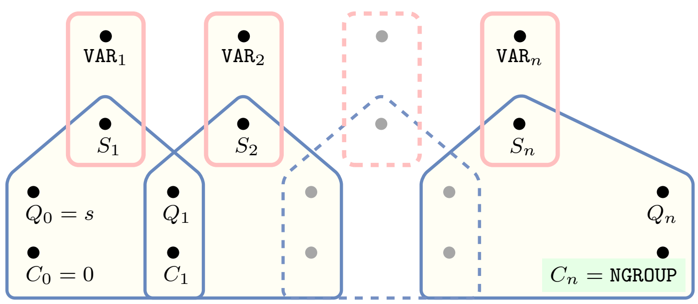
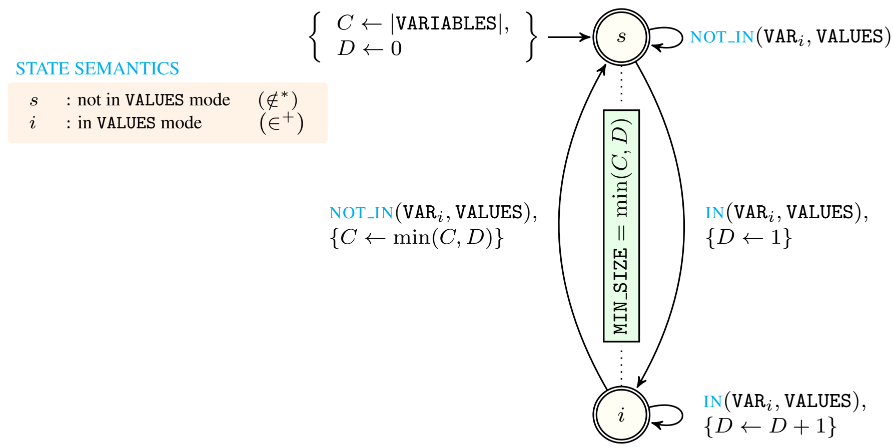


Figure 5.397: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the NGROUP argument of the GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence `VARIABLES`.

	s ($\notin^*$)	i ($\in^+$)
s ($\notin^*$)	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overleftarrow{D}, \overleftarrow{C})$
i ($\in^+$)	$\min(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$

Figure 5.398: Automaton for the `MIN_SIZE` argument of the GROUP constraint and its glue matrix; counters C and D respectively correspond to the size of the smallest group encountered so far, and to the size of the current group.

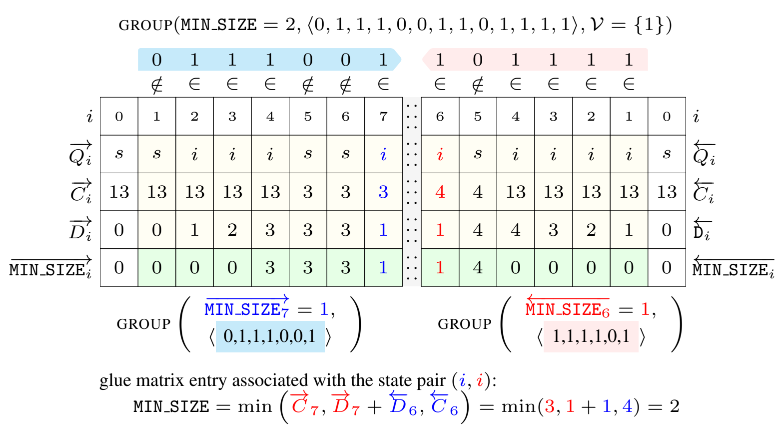


Figure 5.399: Illustrating the use of the state pair (i, i) of the glue matrix for linking `MIN_SIZE` with the counters variables obtained after reading the prefix $0, 1, 1, 1, 0, 0, 1$ and corresponding suffix $1, 0, 1, 1, 1, 1$ of the sequence $0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1$; note that the suffix $1, 0, 1, 1, 1, 1$ (in pink) is proceed in reverse order; the left (resp. right) table shows the initialisation (for $i = 0$) and the evolution (for $i > 0$) of the state of the automaton and its counters C and D upon reading the prefix $0, 1, 1, 1, 0, 0, 1$ (resp. the reverse suffix $1, 1, 1, 1, 0, 1$).

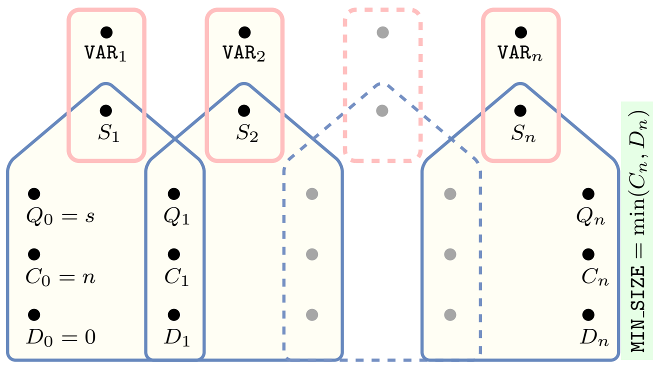


Figure 5.400: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MIN_SIZE argument of the GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

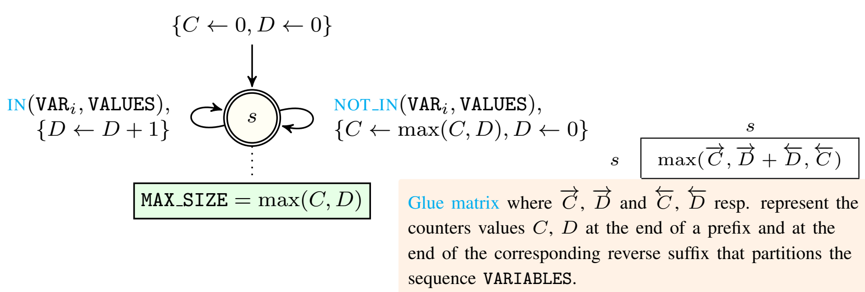


Figure 5.401: Automaton for the MAX_SIZE argument of the GROUP constraint and its glue matrix; counters C and D respectively correspond to the size of the largest group encountered so far, and to the size of the current group.

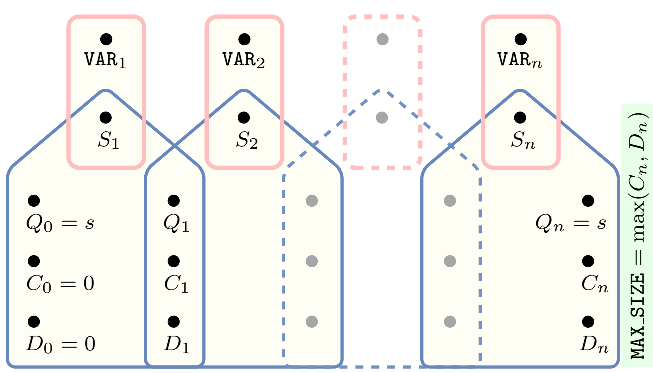
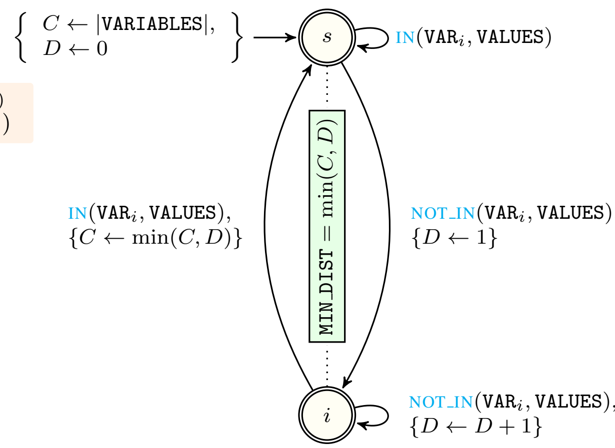


Figure 5.402: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MAX_SIZE argument of the GROUP constraint

STATE SEMANTICS

s	: in VALUES mode	$(\in^*)$
i	: not in VALUES mode	$(\notin^+)$



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s $(\in^*)$	i $(\notin^+)$
s $(\in^*)$	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overleftarrow{D}, \overleftarrow{C})$
i $(\notin^+)$	$\min(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$

Figure 5.403: Automaton for the MIN_DIST argument of the GROUP constraint and its glue matrix; counters C and D respectively correspond to the size of the smallest anti-group encountered so far, and to the size of the current anti-group.

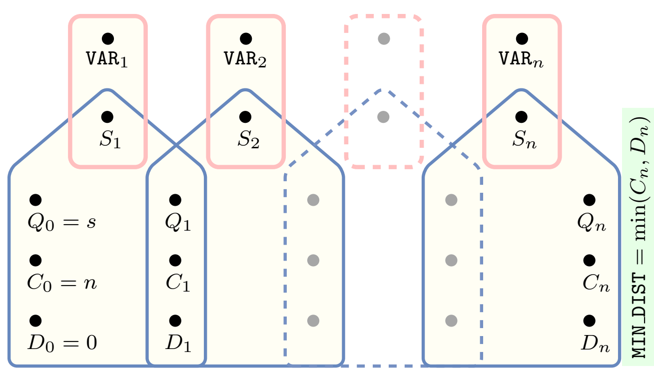


Figure 5.404: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MIN_DIST argument of the GROUP constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

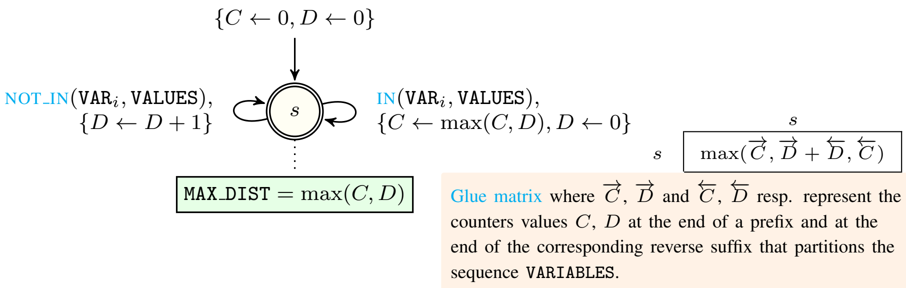


Figure 5.405: Automaton for the MAX_DIST argument of the GROUP constraint and its glue matrix; counters C and D respectively correspond to the size of the largest anti-group encountered so far, and to the size of the current anti-group.

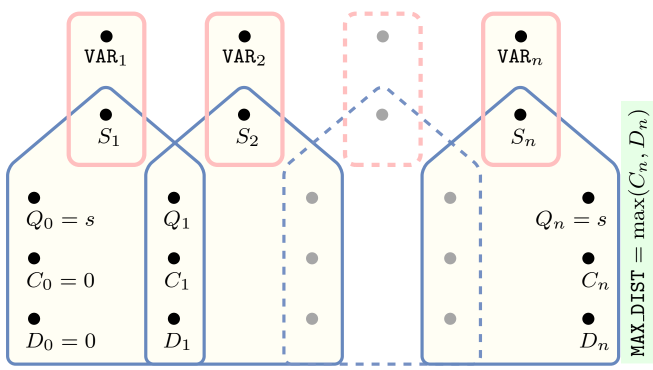


Figure 5.406: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MAX_DIST argument of the GROUP constraint

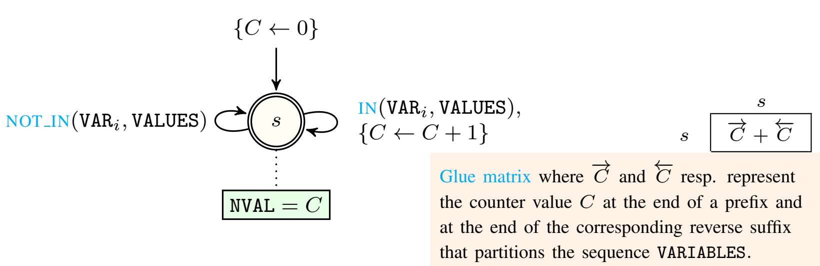


Figure 5.407: Automaton for the NVAL argument of the GROUP constraint and its glue matrix

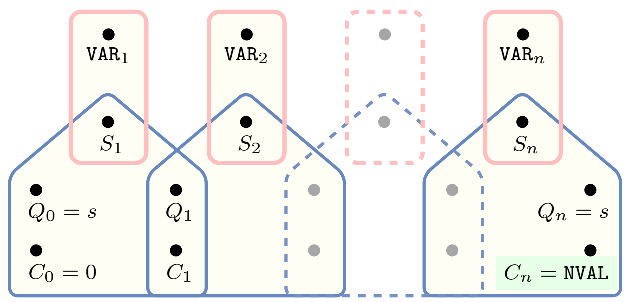


Figure 5.408: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the NVAL argument of the GROUP constraint

5.175 GROUP_SKIP_ISOLATED_ITEM

Origin

Derived from GROUP.

Constraint

GROUP_SKIP_ISOLATED_ITEM(NGROUP, MIN_SIZE, MAX_SIZE, NVAL, VARIABLES, VALUES)

Arguments

NGROUP	:	dvar
MIN_SIZE	:	dvar
MAX_SIZE	:	dvar
NVAL	:	dvar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int)

Restrictions

NGROUP ≥ 0
$3 *$ NGROUP $\leq$ |VARIABLES| $+ 1$
MIN_SIZE ≥ 0
MIN_SIZE $\neq 1$
MAX_SIZE $\geq$ MIN_SIZE
NVAL $\geq$ MAX_SIZE
NVAL $\geq$ NGROUP
NVAL $\leq$ |VARIABLES|
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose

Let n be the number of variables of the collection VARIABLES. Let $X_i, X_{i+1}, \ldots, X_j$ $(1 \leq i < j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ take their values in the set of values VALUES,
- $i = 1$ or X_{i-1} does not take a value in VALUES,
- $j = n$ or X_{j+1} does not take a value in VALUES.

We call such a set of variables a *group*. The constraint GROUP_SKIP_ISOLATED_ITEM is true if all the following conditions hold:

- There are exactly NGROUP groups of variables,
- The number of variables of the smallest group is MIN_SIZE,
- The number of variables of the largest group is MAX_SIZE,
- The number of variables that take their values in the set of values VALUES is equal to NVAL.

Example

$$\boxed{(1, 2, 2, 3, \langle 2, 8, 1, 7, 4, 5, 1, 1, 1\rangle, \langle 0, 2, 4, 6, 8\rangle)}$$

Given the fact that groups are formed by even values in $\{0, 2, 4, 6, 8\}$ (i.e., values expressed by the VALUES collection), and the fact that isolated even values are ignored, the GROUP_SKIP_ISOLATED_ITEM constraint holds since:

- Its first argument, NGROUP, is set to value 1 since the sequence 2 8 1 7 4 5 1 1 1 contains only one group of even values involving more than one even value (i.e., group 2 8).
- Its second and third arguments, MIN_SIZE and MAX_SIZE, are both set to 2 since the only group of even values with more than one even value involves two values (i.e., group 2 8).
- The fourth argument, NVAL, is fixed to 2 since it corresponds to the total number of even values belonging to groups involving more than one even value (i.e., value 4 is discarded since it is an isolated even value of the sequence 2 8 1 7 4 5 1 1 1).

Typical

NGROUP > 0
MIN_SIZE > 0
NVAL $>$ MAX_SIZE
NVAL $>$ NGROUP
NVAL $<$ |VARIABLES|
|VARIABLES| > 1
range(VARIABLES.var) > 1
|VALUES| > 0
|VARIABLES| $>$ |VALUES|

Symmetries

- Items of VARIABLES can be reversed.
- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).

Arg. properties

- Functional dependency: NGROUP determined by VARIABLES and VALUES.
- Functional dependency: MIN_SIZE determined by VARIABLES and VALUES.
- Functional dependency: MAX_SIZE determined by VARIABLES and VALUES.
- Functional dependency: NVAL determined by VARIABLES and VALUES.

Usage

This constraint is useful in order to specify rules about how rest days should be allocated to a person during a period of n consecutive days. In this case VALUES are the codes for the rest days (perhaps a single value) and VARIABLES corresponds to the amount of work done during n consecutive days. We can then express a rule like: in a month one should have at least 4 periods of at least 2 rest days (isolated rest days are not counted as rest periods).

Remark

The following invariant imposes a limit on the maximum number of groups wrt the minimum size of a group and the total number of variables: NGROUP$\cdot(\max($MIN_SIZE$, 2)+1) \leq$ |VARIABLES| $+ 1$.

See also

common keyword: CHANGE_CONTINUITY, GROUP, STRETCH_PATH *(timetabling constraint,sequence)*.

used in graph description: IN.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint.

constraint network structure: alpha-acyclic constraint network(2), alpha-acyclic constraint network(3).

constraint type: timetabling constraint.

filtering: glue matrix.

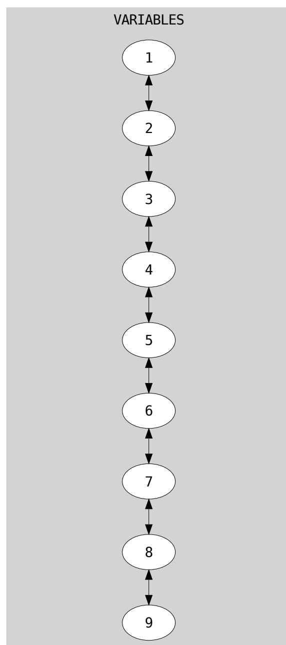
final graph structure: strongly connected component.

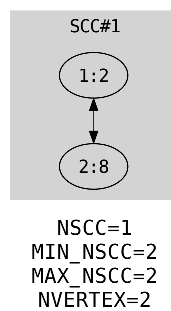
modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CHAIN \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• IN(variables1.var, VALUES) • IN(variables2.var, VALUES)
Graph property(ies)	• **NSCC**= NGROUP • **MIN_NSCC**= MIN_SIZE • **MAX_NSCC**= MAX_SIZE • **NVERTEX**= NVAL

Graph model We use the $CHAIN$ arc generator in order to produce the initial graph. In the context of the **Example** slot, this creates the graph depicted in part (A) of Figure 5.409. We use $CHAIN$ together with the arc constraint variables1.var $\in$ VALUES$\wedge$variables2.var $\in$ VALUES in order to skip the isolated variables that take a value in VALUES that we do not want to count as a group. This is why, on the example, value 4 is not counted as a group. Part (B) of Figure 5.409 shows the final graph associated with the **Example** slot. The GROUP_SKIP_ISOLATED_ITEM constraint of the **Example** slot holds since:

- The final graph contains one strongly connected component. Therefore the number of groups is equal to one.
- The unique strongly connected component of the final graph contains two vertices. Therefore MIN_SIZE and MAX_SIZE are both equal to 2.
- The number of vertices of the final graph is equal to two. Therefore NVAL is equal to 2.

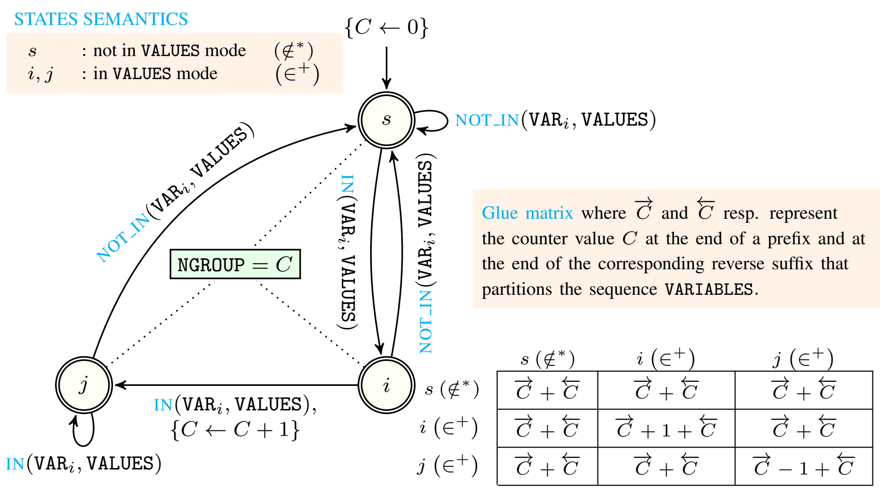




(A) **(B)**

Figure 5.409: Initial and final graph of the GROUP_SKIP_ISOLATED_ITEM constraint

Automaton Figures 5.410, 5.412, 5.414 and 5.416 depict the different automata associated with the GROUP_SKIP_ISOLATED_ITEM constraint. For the automata that respectively compute NGROUP, MIN_SIZE, MAX_SIZE and NVAL we have a 0-1 signature variable $\mathtt{S}_i$ for each variable $\mathtt{VAR}_i$ of the collection VARIABLES. The following signature constraint links $\mathtt{VAR}_i$ and $\mathtt{S}_i$: $\mathtt{VAR}_i \in \mathtt{VALUES} \Leftrightarrow \mathtt{S}_i$.



	$s\ (\notin^*)$	$i\ (\in^+)$	$j\ (\in^+)$
$s\ (\notin^*)$	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}$
$i\ (\in^+)$	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+1+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}$
$j\ (\in^+)$	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}+\overleftarrow{C}$	$\overrightarrow{C}-1+\overleftarrow{C}$

Figure 5.410: Automaton for the NGROUP argument of the GROUP_SKIP_ISOLATED_ITEM constraint and its glue matrix, where counter C is the number of groups encountered so far

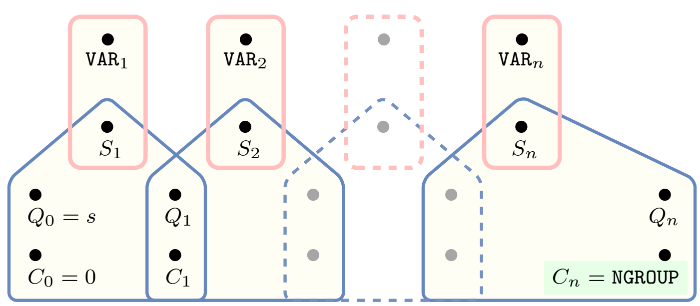
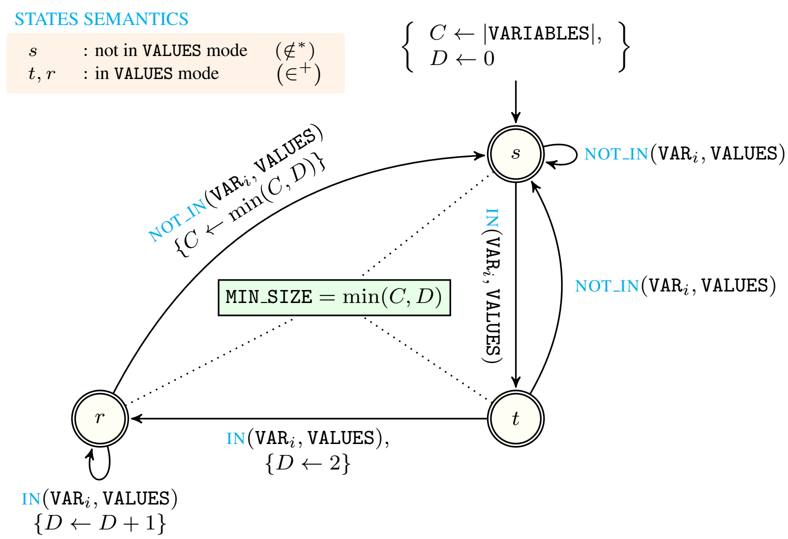


Figure 5.411: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the NGROUP argument of the GROUP_SKIP_ISOLATED_ITEM constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s ($\notin^*$)	t ($\in^+$)	r ($\in^+$)
s ($\notin^*$)	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overleftarrow{D}, \overleftarrow{C})$
t ($\in^+$)	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$	2	$\min(\overrightarrow{C}, \overleftarrow{D} + 1, \overleftarrow{C})$
r ($\in^+$)	$\min(\overrightarrow{C}, \overrightarrow{D}, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overrightarrow{D} + 1, \overleftarrow{C})$	$\min(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D}, \overleftarrow{C})$

Figure 5.412: Automaton for the MIN_SIZE argument of the GROUP_SKIP_ISOLATED_ITEM constraint and its glue matrix; counters C and D respectively correspond to the size of the smallest group encountered so far, and to the size of the current group.

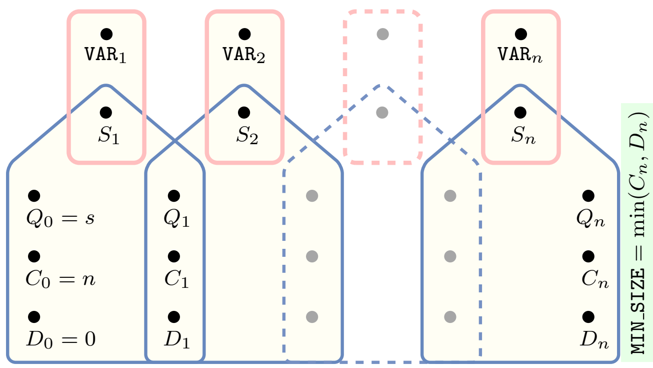
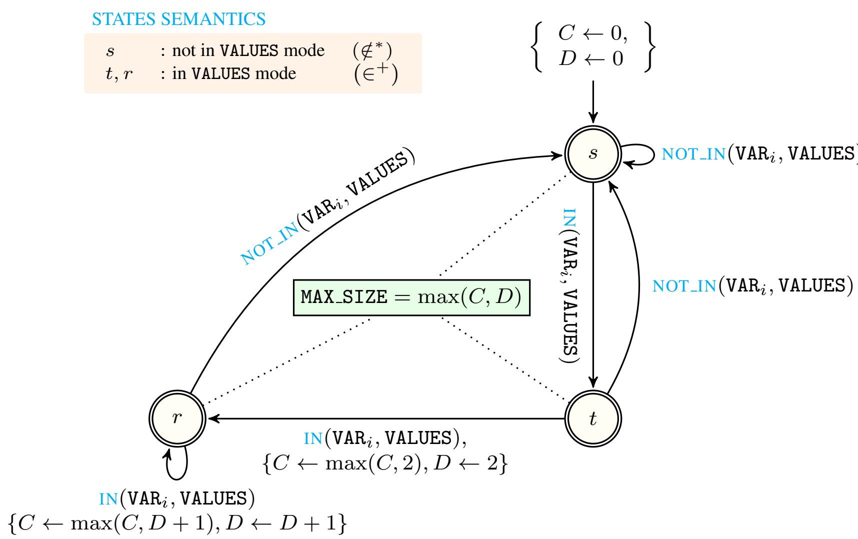


Figure 5.413: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MIN_SIZE argument of the GROUP_SKIP_ISOLATED_ITEM constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s ($\notin^*$)	t ($\in^+$)	r ($\in^+$)
s ($\notin^*$)	$\max(\overrightarrow{C}, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overleftarrow{C})$
t ($\in^+$)	$\max(\overrightarrow{C}, \overleftarrow{C})$	$\max(\overrightarrow{C}, 2, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overleftarrow{D}+1, \overleftarrow{C})$
r ($\in^+$)	$\max(\overrightarrow{C}, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overrightarrow{D}+1, \overleftarrow{C})$	$\max(\overrightarrow{C}, \overrightarrow{D}+\overleftarrow{D}, \overleftarrow{C})$

Figure 5.414: Automaton for the MAX_SIZE argument of the GROUP_SKIP_ISOLATED_ITEM constraint and its glue matrix; counters C and D respectively correspond to the size of the largest group encountered so far, and to the size of the current group.

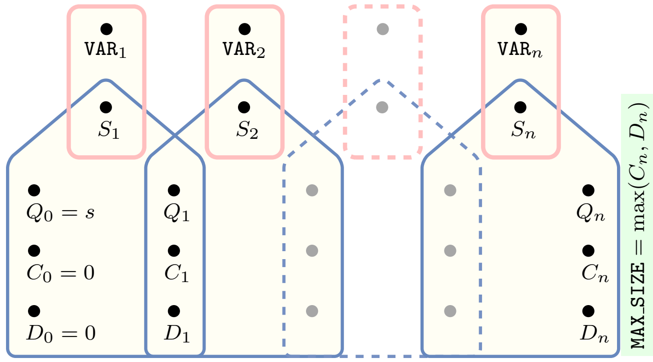


Figure 5.415: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MAX_SIZE argument of the GROUP_SKIP_ISOLATED_ITEM constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

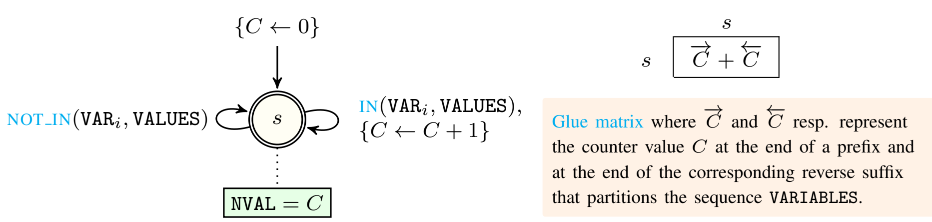


Figure 5.416: Automaton for the NVAL argument of the GROUP_SKIP_ISOLATED_ITEM constraint and its glue matrix

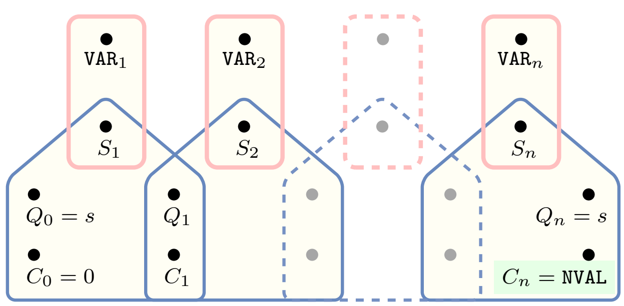


Figure 5.417: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the NVAL argument of the GROUP_SKIP_ISOLATED_ITEM constraint

5.176 GT

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint GT(VAR1, VAR2)

Synonyms REL, XGTY.

Arguments

VAR1 : dvar
VAR2 : dvar

Purpose Enforce the fact that the first variable is strictly greater than the second variable.

Example $(8, 1)$

The GT constraint holds since 8 is strictly greater than 1.

Symmetries

- VAR1 can be replaced by any value $>$ VAR2.
- VAR2 can be replaced by any value $<$ VAR1.

Systems GT in **Choco**, REL in **Gecode**, XGTY in **JaCoP**, #> in **SICStus**.

See also **common keyword:** EQ *(binary constraint,arithmetic constraint)*.

implies: GEQ, NEQ.

implies (if swap arguments): LT.

negation: LEQ.

Keywords **constraint arguments:** binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.177 HIGHEST_PEAK

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from PEAK.

Constraint HIGHEST_PEAK(HEIGHT, VARIABLES)

Arguments

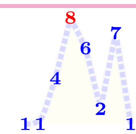
HEIGHT : dvar
VARIABLES : collection(var−dvar)

Restriction required(VARIABLES, var)

Purpose A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \dots, V_m$ is a *peak* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k > V_{k+1}$. HEIGHT is the maximum value of the peak variables. If no such variable exists HEIGHT is equal to MININT.

Example

$(8, \langle 1, 1, 4, 8, 6, 2, 7, 1 \rangle)$
$(1, \langle 0, 1, 1, 0, 0, 1, 0, 1 \rangle)$



The first HIGHEST_PEAK constraint holds since 8 is the maximum peak of the sequence 1 1 4 8 6 2 7 1.

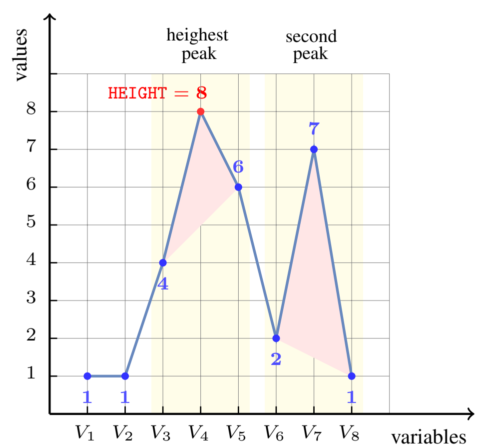


Figure 5.418: Illustration of the first constraint of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values 1, 1, 4, 8, 6, 2, 7, 1 and its corresponding highest peak 8

Typical

$|$VARIABLES$| > 2$
range(VARIABLES.var) > 2
PEAK(VARIABLES.var) > 0

Typical model `nval`(`VARIABLES.var`) > 2

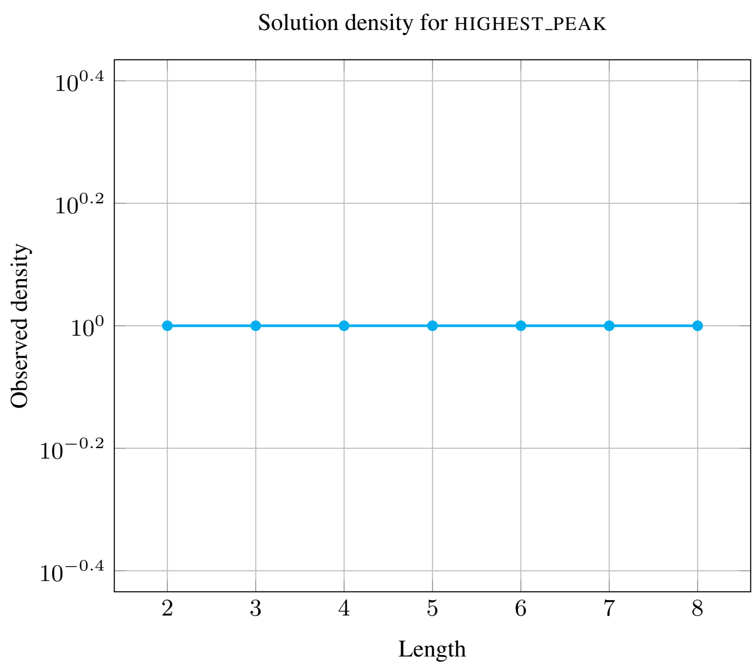
Symmetry Items of `VARIABLES` can be reversed.

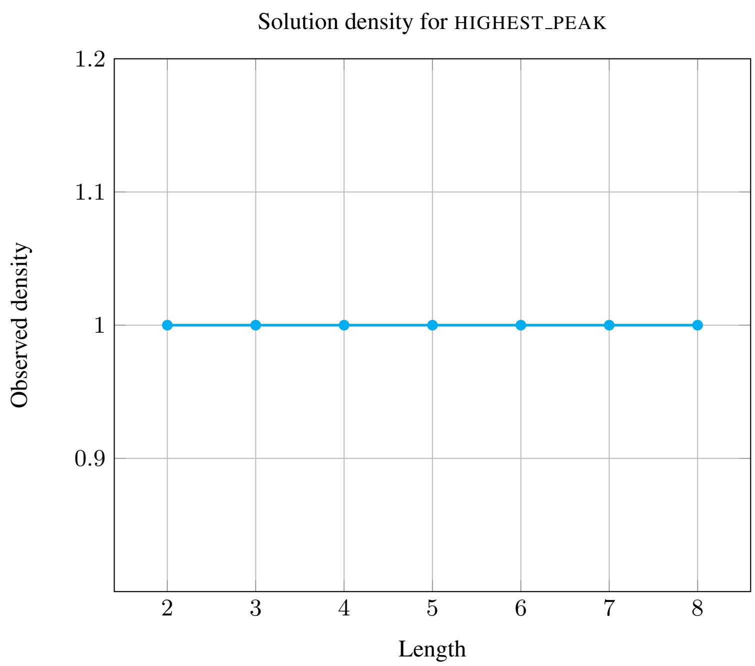
Arg. properties Functional dependency: `HEIGHT` determined by `VARIABLES`.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for HIGHEST_PEAK: domains $0..n$

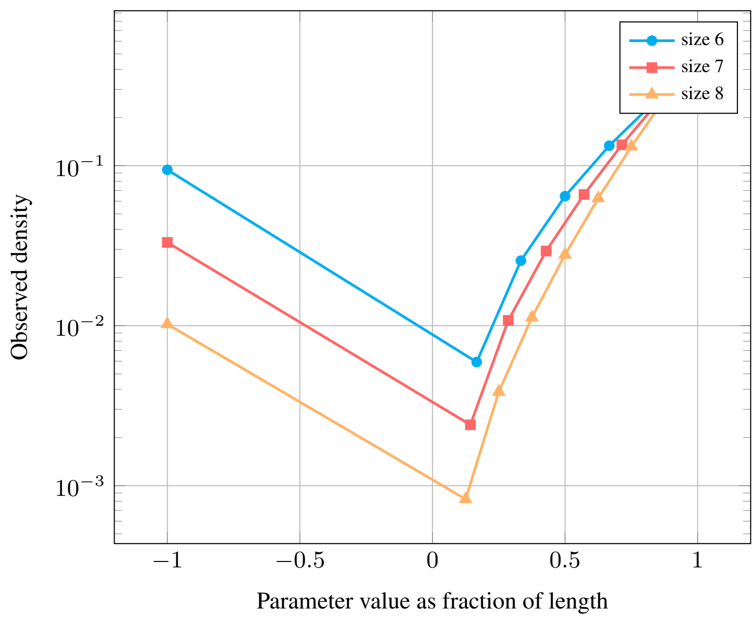




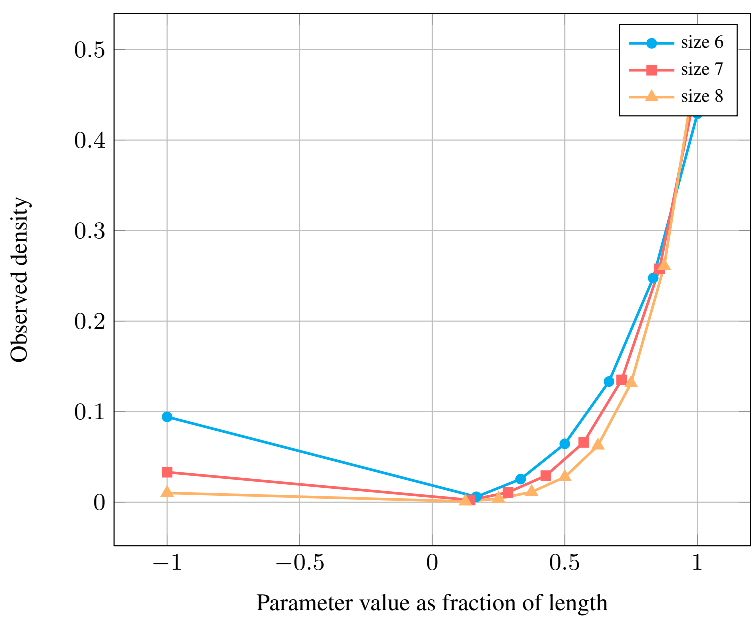
Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	-1000000	9	50	295	1792	11088	69498	439791
	1	-	1	11	92	697	5036	35443
	2	-	4	44	380	3000	22632	166208
	3	-	9	99	900	7587	61389	484020
	4	-	-	176	1712	15680	138544	1195056
	5	-	-	-	2900	29125	283250	2693425
	6	-	-	-	-	50472	540576	5665896
	7	-	-	-	-	-	976227	11233250
	8	-	-	-	-	-	-	21133632

Solution count for HIGHEST_PEAK: domains $0..n$

Solution density for HIGHEST_PEAK



Solution density for HIGHEST_PEAK



See also **common keyword:** DEEPEST_VALLEY, PEAK *(sequence)*.

implies: BETWEEN_MIN_MAX.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

filtering: glue matrix.

modelling: functional dependency.

Automaton Figure 5.419 depicts the automaton associated with the HIGHEST_PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i:

$\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0 \wedge \mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1 \wedge \mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2.$

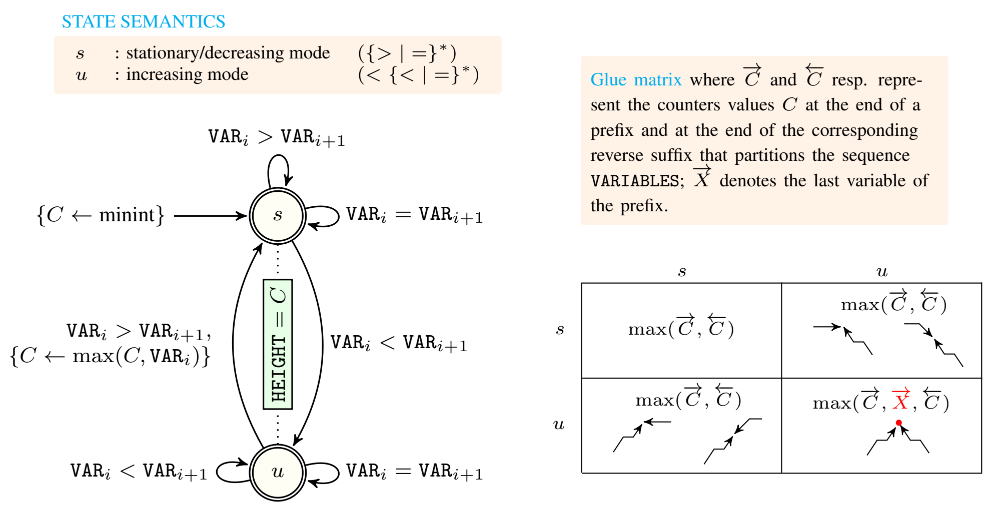


Figure 5.419: Automaton of the HIGHEST_PEAK constraint and its glue matrix (state s means that we are in *decreasing* or *stationary* mode, state u means that we are in *increasing* mode, a new peak is detected each time we switch from increasing to decreasing mode and the counter C is updated accordingly); minint is the smallest integer that can be represented on a machine

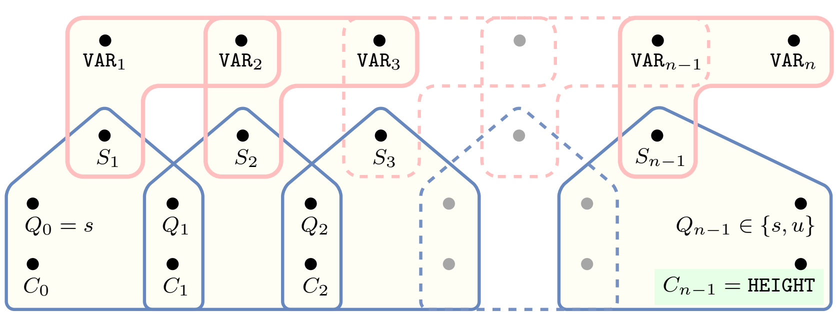


Figure 5.420: Hypergraph of the reformulation corresponding to the automaton of the HIGHEST_PEAK constraint (C_0 is set to minint the largest integer that can be represented on a machine)

5.178 IMPLY

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint IMPLY(VAR, VARIABLES)

Synonyms REL, IFTHEN.

Arguments

VAR : dvar
VARIABLES : collection(var−dvar)

Restrictions

VAR ≥ 0
VAR ≤ 1
$|$VARIABLES$| = 2$
required(VARIABLES, var)
VARIABLES.var ≥ 0
VARIABLES.var ≤ 1

Purpose Let VARIABLES be a collection of 0-1 variables $\text{VAR}_1, \text{VAR}_2$. Enforce $\text{VAR} = (\text{VAR}_1 \Rightarrow \text{VAR}_2)$.

Example

$(1, \langle 0, 0\rangle)$
$(1, \langle 0, 1\rangle)$
$(0, \langle 1, 0\rangle)$
$(1, \langle 1, 1\rangle)$

Symmetry All occurrences of 0 in VAR and in VARIABLES.var can be set to 1.

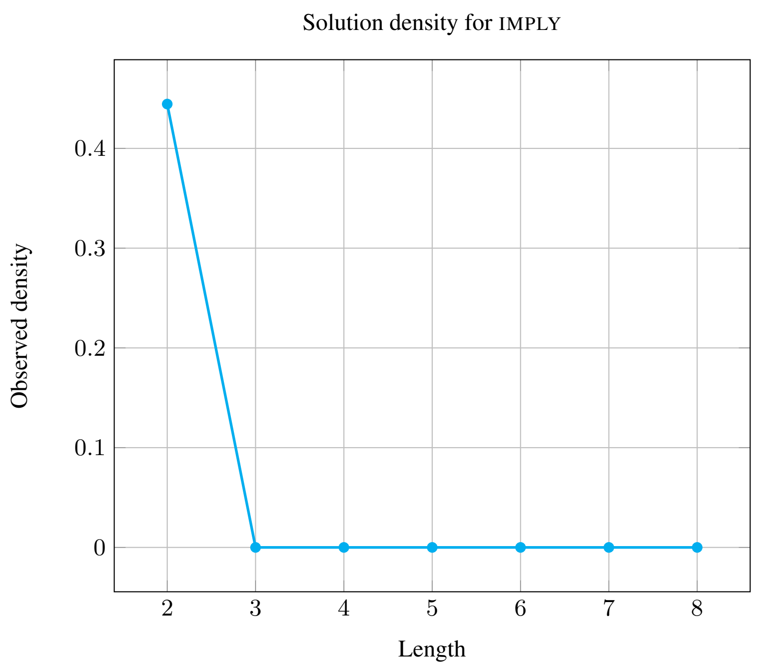
Arg. properties Functional dependency: VAR determined by VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	4	0	0	0	0	0	0

Number of solutions for IMPLY: domains $0..n$

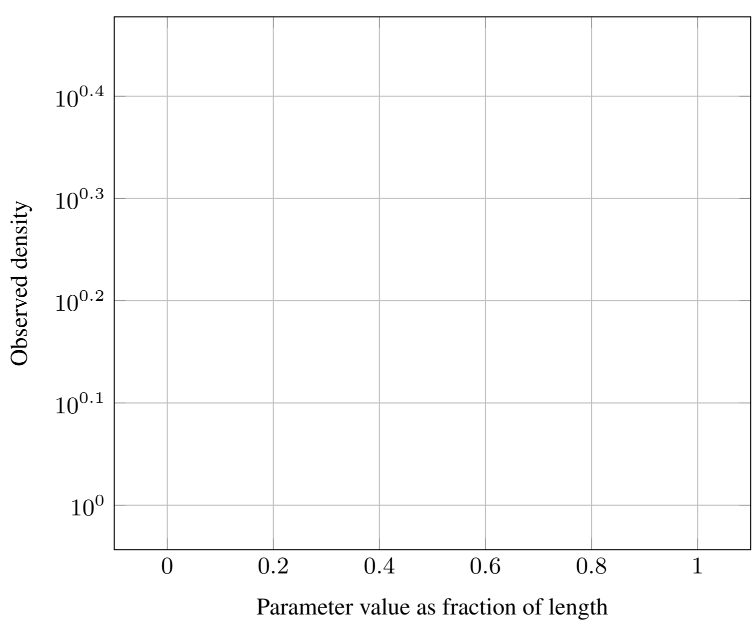

Solution density for IMPLY
Observed density
1.9
2
2.1
2.2
2.3
2.4
Length


Solution density for IMPLY
Observed density
0.4
0.3
0.2
0.1
0
2
3
4
5
6
7
8
Length

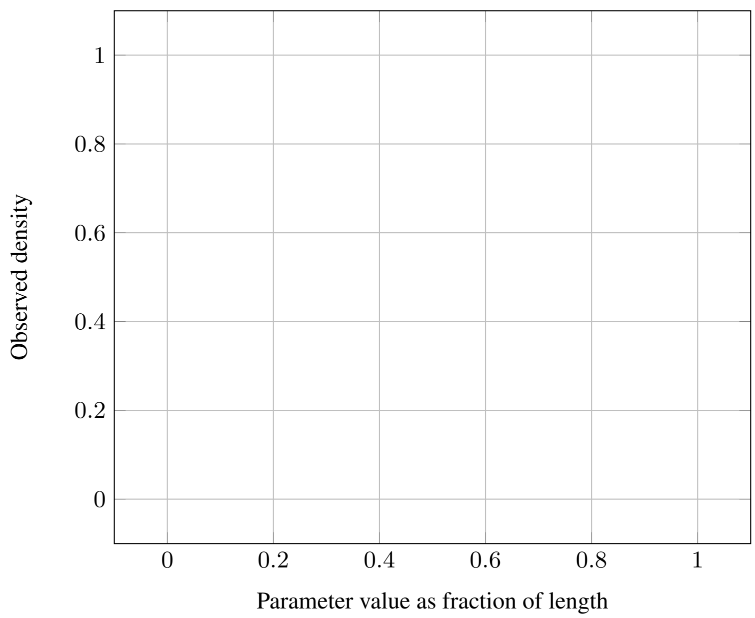
Length (n)		2
Total		4
Parameter value	0	1
	1	3

Solution count for IMPLY: domains $0..n$

Solution density for IMPLY



Solution density for IMPLY



Systems REIFIEDLEFTIMP in **Choco**, REL in **Gecode**, IFTHENBOOL in **JaCoP**, #=> in **SICStus**.

See also **common keyword:** AND, EQUIVALENT, NAND, NOR, OR, XOR *(Boolean constraint)*.

implies: ATLEAST_NVALUE, SOFT_ALLDIFFERENT_CTR.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: functional dependency.

Automaton Figure 5.421 depicts the automaton associated with the IMPLY constraint. To the first argument VAR of the IMPLY constraint corresponds the first signature variable. To each variable $\mathtt{VAR}_i$ of the second argument VARIABLES of the IMPLY constraint corresponds the next signature variable. There is no signature constraint.

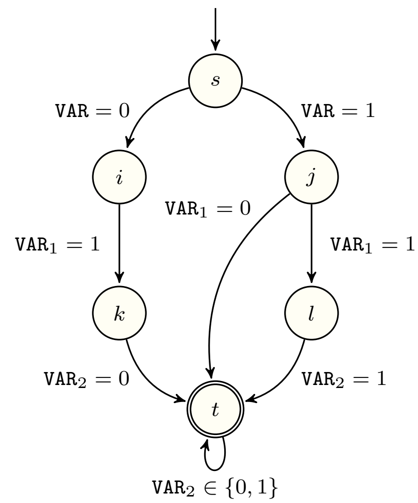


Figure 5.421: Automaton of the IMPLY constraint

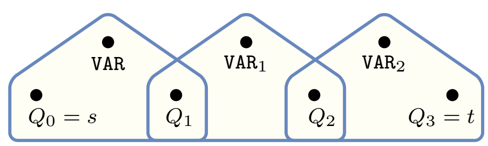


Figure 5.422: Hypergraph of the reformulation corresponding to the automaton of the IMPLY constraint

5.179 IN

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Domain definition.

Constraint IN(VAR, VALUES)

Synonyms DOM, IN_SET, MEMBER.

Arguments

VAR : dvar
VALUES : collection(val−int)

Restrictions

$|\mathtt{VALUES}| > 0$
required(VALUES, val)
distinct(VALUES, val)

Purpose Enforce the domain variable VAR to take a value within the values described by the VALUES collection.

Example $(3, \langle 1, 3\rangle)$

The IN constraint holds since its first argument VAR $= 3$ occurs within the collection of values VALUES $= \langle 1, 3\rangle$.

Typical $|\mathtt{VALUES}| > 1$

Symmetries

- Items of VALUES are permutable.
- VAR can be set to any value of VALUES.val.
- One and the same constant can be added to VAR as well as to the val attribute of all items of VALUES.

Arg. properties Extensible wrt. VALUES.

Remark Entailment occurs immediately after posting this constraint.

The IN constraint is called DOM in **Gecode** (http://www.gecode.org/), and MEMBER in **MiniZinc** (http://www.minizinc.org/). In **MiniZinc** the val attribute is not necessarily fixed, i.e. it can be a domain variable.

Systems MEMBER in **Choco**, REL in **Gecode**, DOM in **Gecode**, IN in **JaCoP**, MEMBER in **MiniZinc**, IN in **SICStus**, IN_SET in **SICStus**.

Used in AMONG, CARDINALITY_ATMOST_PARTITION, GROUP, GROUP_SKIP_ISOLATED_ITEM, IN_SAME_PARTITION, OPEN_AMONG.

See also

common keyword: DOMAIN *(domain definition)*, IN_INTERVAL, IN_SAME_PARTITION, IN_SET *(value constraint)*.

implied by: MAXIMUM, MINIMUM.

implies: BETWEEN_MIN_MAX.

negation: NOT_IN.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: unary constraint.

constraint network structure: centered cyclic(1) constraint network(1).

constraint type: value constraint.

filtering: arc-consistency.

modelling: included, domain definition.

Derived Collection
col(VARIABLES−collection(var−dvar), [item(var − VAR)])

Arc input(s) VARIABLES VALUES

Arc generator $PRODUCT \mapsto$ collection(variables, values)

Arc arity 2

Arc constraint(s) variables.var = values.val

Graph property(ies) **NARC** = 1

Graph model

Parts (A) and (B) of Figure 5.423 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

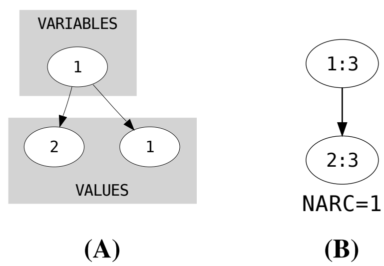


Figure 5.423: Initial and final graph of the IN constraint

Signature

Since all the val attributes of the VALUES collection are distinct and because of the arc constraint variables.var = values.val the final graph contains at most one arc. Therefore we can rewrite **NARC** = 1 to **NARC** $\geq$ 1 and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 5.424 depicts the automaton associated with the IN constraint. Let $\mathtt{VAL}_i$ be the `val` attribute of the i^{th} item of the `VALUES` collection. To each pair $(\mathtt{VAR}, \mathtt{VAL}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $\mathtt{VAR} = \mathtt{VAL}_i \Leftrightarrow S_i$.

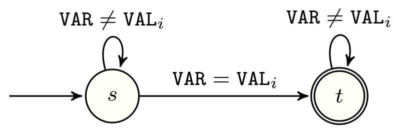


Figure 5.424: Automaton of the IN constraint

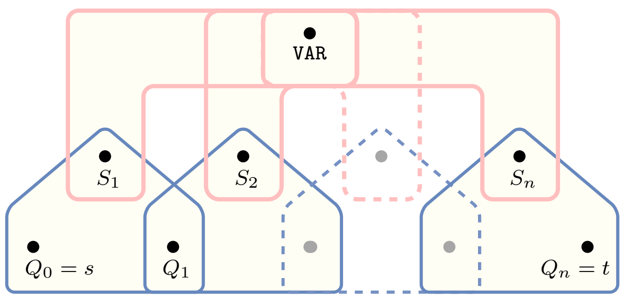


Figure 5.425: Hypergraph of the reformulation corresponding to the automaton of the IN constraint

5.180 IN_INTERVAL

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Domain definition.

Constraint IN_INTERVAL(VAR, LOW, UP)

Synonyms DOM, IN.

Arguments

VAR : dvar
LOW : int
UP : int

Restriction $\text{LOW} \leq \text{UP}$

Purpose Enforce the domain variable VAR to take a value within the interval [LOW, UP].

Example $(3, 2, 5)$

The IN_INTERVAL constraint holds since its first argument $\text{VAR} = 3$ is greater than or equal to its second argument $\text{LOW} = 2$ and less than or equal to its third argument $\text{UP} = 5$.

Typical

$\text{LOW} < \text{UP}$
$\text{VAR} > \text{LOW}$
$\text{VAR} < \text{UP}$

Symmetries

- LOW can be decreased.
- UP can be increased.
- An occurrence of a value of VAR can be replaced by any other value in [LOW, UP].
- One and the same constant can be added to VAR, LOW and UP.

Remark Entailment occurs immediately after posting this constraint.

The IN_INTERVAL constraint is referenced under the name DOM in **Gecode**.

Systems MEMBER in **Choco**, DOM in **Gecode**, IN in **JaCoP**, IN in **SICStus**.

See also **common keyword:** DOMAIN, IN *(domain definition)*.

generalisation: IN_INTERVAL_REIFIED *(reified version)*, IN_INTERVALS *(single interval replaced by a set of intervals)*, IN_SET *(interval replaced by set variable)*.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: unary constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: value constraint.

filtering: arc-consistency.

modelling: interval, domain definition.

Derived Collections

$\texttt{col}(\texttt{VARIABLE}-\texttt{collection}(\texttt{var}-\texttt{dvar}), [\texttt{item}(\texttt{var}-\texttt{VAR})])$

$\texttt{col}\left(\begin{array}{l}\texttt{INTERVAL}-\texttt{collection}(\texttt{low}-\texttt{int}, \texttt{up}-\texttt{int}),\\ [\texttt{item}(\texttt{low}-\texttt{LOW}, \texttt{up}-\texttt{UP})]\end{array}\right)$

Arc input(s) VARIABLE INTERVAL

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{variable}, \texttt{interval})$

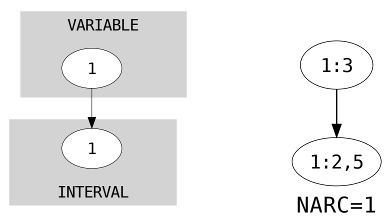
Arc arity 2

Arc constraint(s)

- $\texttt{variable.var} \geq \texttt{interval.low}$
- $\texttt{variable.var} \leq \texttt{interval.up}$

Graph property(ies) $\mathbf{NARC} = 1$

Graph model Parts (A) and (B) of Figure 5.426 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.



(A) (B)

Figure 5.426: Initial and final graph of the IN_INTERVAL constraint

Automaton Figure 5.427 depicts the automaton associated with the IN_INTERVAL constraint. We have a single 0-1 signature variable S as well as the following signature constraint: $\texttt{VAR} \geq \texttt{LOW} \wedge \texttt{VAR} \leq \texttt{UP} \Leftrightarrow S$.

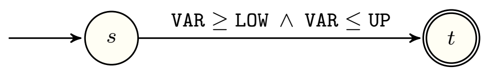


Figure 5.427: Automaton of the IN_INTERVAL constraint

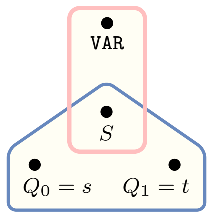


Figure 5.428: Hypergraph of the reformulation corresponding to the automaton of the IN_INTERVAL constraint

5.181 IN_INTERVAL_REIFIED

Origin Reified version of IN_INTERVAL.

Constraint IN_INTERVAL_REIFIED(VAR, LOW, UP, B)

Synonyms DOM_REIFIED, IN_REIFIED.

Arguments

VAR : dvar
LOW : int
UP : int
B : dvar

Restrictions

LOW $\leq$ UP
B ≥ 0
B ≤ 1

Purpose Enforce the following equivalence, VAR $\in$ [LOW, UP] $\Leftrightarrow$ B.

Example $(3, 2, 5, 1)$

The IN_INTERVAL_REIFIED constraint holds since:

- Its first argument VAR $= 3$ is greater than or equal to its second argument LOW $= 2$ and less than or equal to its third argument UP $= 5$ (i.e., $3 \in [2, 5]$).
- The corresponding Boolean variable B is set to 1 since condition $3 \in [2, 5]$ holds.

Typical

VAR $\neq$ LOW
VAR $\neq$ UP
LOW $<$ UP

Symmetries

- An occurrence of a value of VAR that belongs to [LOW, UP] (resp. does not belong to [LOW, UP]) can be replaced by any other value in [LOW, UP]) (resp. not in [LOW, UP]).
- One and the same constant can be added to VAR, LOW and UP.

Reformulation The IN_INTERVAL_REIFIED constraint can be reformulated in terms of linear constraints. For convenience, we rename VAR to x, LOW to l, UP to u, and B to y. The constraint is decomposed into the following conjunction of constraints:

$$x \geq l \quad \Leftrightarrow y_1,$$
$$x \leq u \quad \Leftrightarrow y_2,$$
$$y_1 \wedge y_2 \Leftrightarrow y\ .$$

We show how to encode these constraints with linear inequalities. The first constraint, i.e., $x \geq l \Leftrightarrow y_1$ is encoded by posting one of the following three constraints:

$$\begin{cases} \text{a)} & \text{if } \underline{x} \geq l: & y_1 = 1, \\ \text{b)} & \text{if } \overline{x} < l: & y_1 = 0, \\ \text{c)} & \text{otherwise}: & x \geq (l - \underline{x}) \cdot y_1 + \underline{x} \ \wedge \ x \leq (\overline{x} - l + 1) \cdot y_1 + l - 1. \end{cases}$$

On the one hand, cases a) and b) correspond to situations where one can fix y_1, no matter what value will be assigned to x. On the other hand, in case c), y_1 can take both values 0 or 1 depending on the value assigned to x. As shown by Figure 5.429, all possible solutions for the pair of variables (x, y_1) satisfy the following two linear inequalities $x \geq (l - \underline{x}) \cdot y_1 + \underline{x}$ and $x \leq (\overline{x} - l + 1) \cdot y_1 + l - 1$. The first inequality discards all points that are above the line that goes through the two extreme solution points $(\underline{x}, 0)$ and $(l, 1)$, while the second one removes all points that are below the line that goes through the two extreme solution points $(l - 1, 0)$ and $(\overline{x}, 1)$.

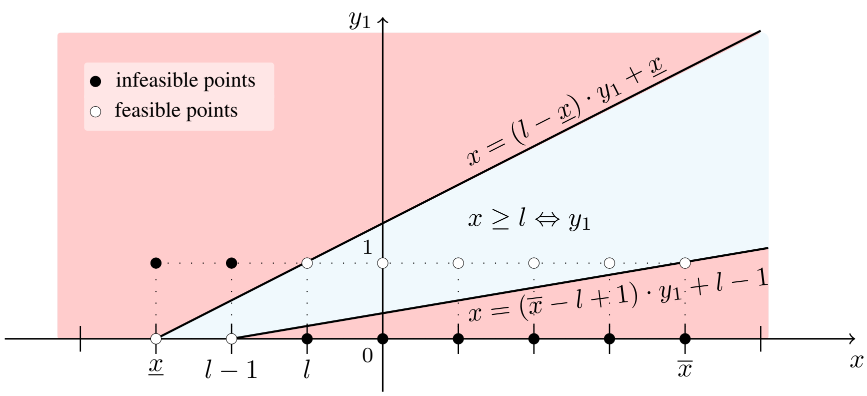


Figure 5.429: Illustration of the reformulation of the reified constraint $x \geq l \Leftrightarrow y_1$ with two linear inequalities

The second constraint, i.e., $x \leq u \Leftrightarrow y_2$ is encoded by posting one of the following three constraints:

$$\begin{cases} \text{d)} & \text{if } \overline{x} \leq u: & y_2 = 1, \\ \text{e)} & \text{if } \underline{x} > u: & y_2 = 0, \\ \text{f)} & \text{otherwise}: & x \leq (u - \overline{x}) \cdot y_2 + \overline{x} \ \wedge \ x \geq (\underline{x} - u - 1) \cdot y_2 + u + 1. \end{cases}$$

On the one hand, cases d) and e) correspond to situations where one can fix y_2, no matter what value will be assigned to x. On the other hand, in case f), y_2 can take both value 0 or 1 depending on the value assigned to x. As shown by Figure 5.430, all possible solutions for the pair of variables (x, y_2) satisfy the following two linear inequalities $x \leq (u - \overline{x}) \cdot y_2 + \overline{x}$ and $x \geq (\underline{x} - u - 1) \cdot y_2 + u + 1$. The first inequality discards all points that are above the line that goes through the two extreme solution points $(\overline{x}, 0)$ and $(u, 1)$, while the second one removes all points that are below the line that goes through the two extreme solution points $(u + 1, 0)$ and $(\underline{x}, 1)$.

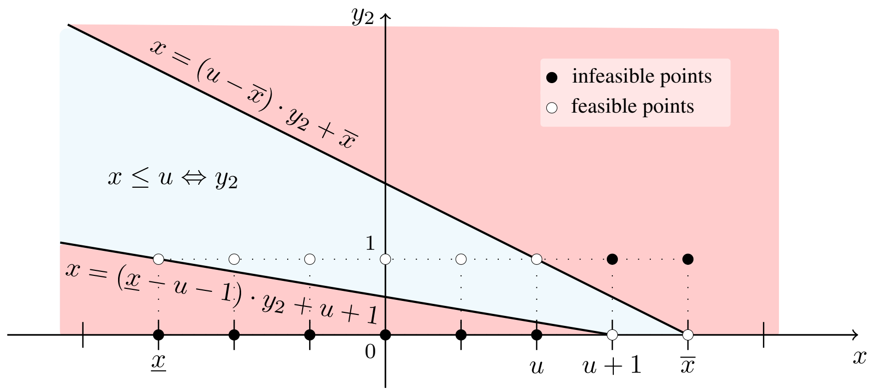


Figure 5.430: Illustration of the reformulation of the reified constraint $x \leq u \Leftrightarrow y_2$ with two linear inequalities

The third constraint, i.e., $y_1 \wedge y_2 \Leftrightarrow y$ is encoded as:

$$\begin{cases} \text{g)} & y \geq y_1 + y_2 - 1, \\ \text{h)} & y \leq y_1, \\ \text{i)} & y \leq y_2. \end{cases}$$

Case g) handles the implication $y_1 \wedge y_2 \Rightarrow y$, while cases h) and i) take care of the other side $y \Rightarrow y_1 \wedge y_2$.

See also

specialisation: IN_INTERVAL.

uses in its reformulation: ALLDIFFERENT *(bound consistency preserving reformulation)*.

Keywords

characteristic of a constraint: reified constraint.

constraint arguments: binary constraint.

constraint type: predefined constraint, value constraint.

filtering: arc-consistency.

5.182 IN_INTERVALS

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Domain definition.

Constraint IN_INTERVALS(VAR, INTERVALS)

Synonym IN.

Arguments

VAR	:	dvar
INTERVALS	:	collection(low−int, up−int)

Restrictions

required(INTERVALS, [low, up])
INTERVALS.low $\leq$ INTERVALS.up
$|$INTERVALS$| > 0$

Purpose Enforce the domain variable VAR to take a value within one of the intervals specified by the collection of intervals INTERVALS.

Example $(5, \langle$low $-$ 1 up $-$ 1, low $-$ 3 up $-$ 5, low $-$ 8 up $-$ 8$\rangle)$

The IN_INTERVALS constraint holds since its first argument VAR $=$ 5 belongs to the second intervals of the collection of intervals INTERVALS.

Typical $|$INTERVALS$| > 1$

Symmetries

- Items of INTERVALS are permutable.
- INTERVALS.low can be decreased.
- INTERVALS.up can be increased.
- One and the same constant can be added to VAR as well as to the low and up attributes of all items of INTERVALS.

Arg. properties Extensible wrt. INTERVALS.

Remark Entailment occurs immediately after posting this constraint.

Systems DOM in **Gecode**, IN in **JaCoP**, IN in **SICStus**.

See also **specialisation:** IN_INTERVAL *(set of intervals replaced by single interval)*.

Keywords **constraint arguments:** unary constraint.
constraint type: value constraint, predefined constraint.
filtering: arc-consistency.
modelling: interval, domain definition.

5.183 IN_RELATION

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Constraint explicitly defined by tuples of values.

Constraint IN_RELATION(VARIABLES, TUPLES_OF_VALS)

Synonyms CASE, EXTENSION, EXTENSIONAL, EXTENSIONAL_SUPPORT, EXTENSIONAL_SUPPORTVA, EXTENSIONAL_SUPPORTMDD, EXTENSIONAL_SUPPORTSTR, FEASTUPLEAC, TABLE.

Types

TUPLE_OF_VARS : collection(var−dvar)
TUPLE_OF_VALS : collection(val−int)

Arguments

VARIABLES : TUPLE_OF_VARS
TUPLES_OF_VALS : collection(tuple − TUPLE_OF_VALS)

Restrictions

required(TUPLE_OF_VARS, var)
|TUPLE_OF_VARS| ≥ 1
|TUPLE_OF_VALS| ≥ 1
|TUPLE_OF_VALS| $=$ |VARIABLES|
required(TUPLE_OF_VALS, val)
required(TUPLES_OF_VALS, tuple)

Purpose Enforce the tuple of variables VARIABLES to take its value out of a set of tuples of values TUPLES_OF_VALS. The *value* of a tuple of variables $\langle V_1, V_2, \ldots, V_n\rangle$ is a tuple of values $\langle U_1, U_2, \ldots, U_n\rangle$ if and only if $V_1 = U_1 \wedge V_2 = U_2 \wedge \cdots \wedge V_n = U_n$.

Example

$$\left(\begin{array}{l}\langle 5,3,3\rangle, \\ \langle \texttt{tuple} - \langle 5,2,3\rangle, \texttt{tuple} - \langle 5,2,6\rangle, \texttt{tuple} - \langle 5,3,3\rangle\rangle\end{array}\right)$$

The IN_RELATION constraint holds since its first argument $\langle 5,3,3\rangle$ corresponds to the third item of the collection of tuples TUPLES_OF_VALS.

Typical |TUPLE_OF_VARS| > 1

Symmetries

- Items of TUPLES_OF_VALS are permutable.
- Items of VARIABLES and TUPLES_OF_VALS.tuple are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values in VARIABLES or TUPLES_OF_VALS.tuple can be swapped; all occurrences of a tuple of values in VARIABLES or TUPLES_OF_VALS.tuple can be renamed to any unused tuple of values.

Arg. properties Extensible wrt. TUPLES_OF_VALS.

Usage Quite often some constraints cannot be easily expressed, neither by a formula, nor by a regular pattern. In this case one has to define the constraint by specifying in extension the combinations of allowed values.

Remark The IN_RELATION constraint is called EXTENSIONAL_SUPPORT in **JaCoP** (http://www.jacop.eu/). Within **SICStus Prolog** the constraint can be applied to more than a single tuple of variables and is called TABLE. Within [92] this constraint is called EXTENSION.

The IN_RELATION constraint is called TABLE in **MiniZinc** (http://www.minizinc.org/).

Systems FEASPAIRAC in **Choco**, INFEASPAIRAC in **Choco**, RELATIONPAIRAC in **Choco**, FEASTUPLEAC in **Choco**, INFEASTUPLEAC in **Choco**, RELATIONTUPLEAC in **Choco**, EXTENSIONAL in **Gecode**, EXTENSIONALSUPPORTVA in **JaCoP**, EXTENSIONALSUPPORTMDD in **JaCoP**, EXTENSIONALSUPPORTSTR in **JaCoP**, TABLE in **MiniZinc**, CASE in **SICStus**, RELATION in **SICStus**, TABLE in **SICStus**.

Used in COND_LEX_COST, COND_LEX_GREATER, COND_LEX_GREATEREQ, COND_LEX_LESS, COND_LEX_LESSEQ.

See also **common keyword:** ELEMENT *(data constraint)*.

cost variant: COND_LEX_COST *(COST parameter added)*.

used in graph description: VEC_EQ_TUPLE.

Keywords **characteristic of a constraint:** tuple, derived collection.

combinatorial object: relation.

constraint type: data constraint, extension.

filtering: arc-consistency.

Derived Collection

$\texttt{col}\left(\begin{array}{l}\texttt{TUPLES\_OF\_VARS}-\texttt{collection}(\texttt{vec}-\texttt{TUPLE\_OF\_VARS}),\\ [\texttt{item}(\texttt{vec}-\texttt{VARIABLES})]\end{array}\right)$

Arc input(s)	TUPLES_OF_VARS TUPLES_OF_VALS
Arc generator	$PRODUCT \mapsto$ collection(tuples_of_vars, tuples_of_vals)
Arc arity	2
Arc constraint(s)	VEC_EQ_TUPLE(tuples_of_vars.vec, tuples_of_vals.tuple)
Graph property(ies)	**NARC** ≥ 1

Graph model Parts (A) and (B) of Figure 5.431 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

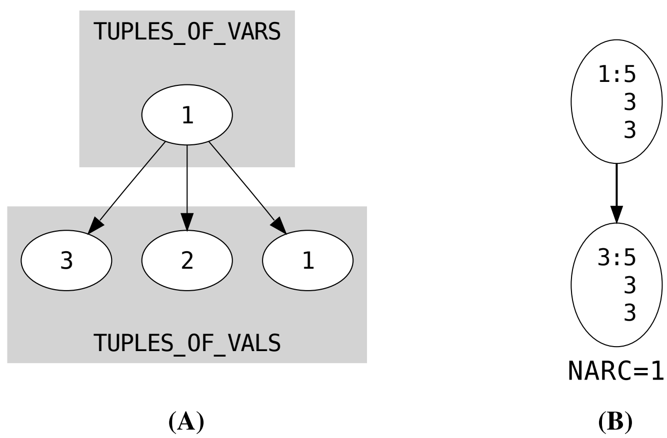


Figure 5.431: Initial and final graph of the IN_RELATION constraint

5.184 IN_SAME_PARTITION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Used for defining several entries of this catalog.

Constraint IN_SAME_PARTITION(VAR1, VAR2, PARTITIONS)

Type

VALUES : collection(val−int)

Arguments

VAR1 : dvar
VAR2 : dvar
PARTITIONS : collection(p − VALUES)

Restrictions

|VALUES| ≥ 1
required(VALUES, val)
distinct(VALUES, val)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2

Purpose Enforce VAR1 and VAR2 to be respectively assigned to values v_1 and v_2 that both belong to a same partition of the collection PARTITIONS.

Example

$(6, 2, \langle p - \langle 1, 3\rangle, p - \langle 4\rangle, p - \langle 2, 6\rangle\rangle)$

The IN_SAME_PARTITION constraint holds since its first and second arguments VAR1 = 6 and VAR2 = 2 both belong to the third partition $\langle 2, 6\rangle$ of its third argument PARTITIONS.

Typical VAR1 ≠ VAR2

Symmetries

- Arguments are permutable w.r.t. permutation (VAR1, VAR2) (PARTITIONS).
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.

Arg. properties Extensible wrt. PARTITIONS.

Used in ALLDIFFERENT_PARTITION, BALANCE_PARTITION, CHANGE_PARTITION, COMMON_PARTITION, NCLASS, SAME_PARTITION, SOFT_SAME_PARTITION_VAR, SOFT_USED_BY_PARTITION_VAR, USED_BY_PARTITION.

See also **common keyword:** ALLDIFFERENT_PARTITION *(partition)*, IN *(value constraint)*.

used in graph description: IN.

Keywords

characteristic of a constraint: partition, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: binary constraint.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: value constraint.

filtering: arc-consistency.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{VAR1}),\texttt{item}(\texttt{var}-\texttt{VAR2})]\end{array}\right)$$

Arc input(s) VARIABLES PARTITIONS

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{variables},\texttt{partitions})$

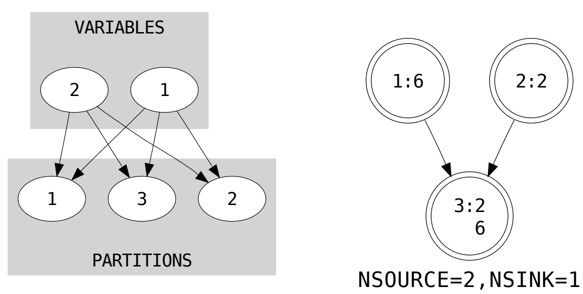
Arc arity 2

Arc constraint(s) IN(variables.var, partitions.p)

Graph property(ies)
- **NSOURCE**$= 2$
- **NSINK**$= 1$

Graph model VAR1 and VAR2 are put together in the derived collection VARIABLES. Since both VAR1 and VAR2 should take their values in one of the partition depicted by the PARTITIONS collection, the final graph should have two sources corresponding respectively to VAR1 and VAR2. Since two, possibly distinct, values should be assigned to VAR1 and VAR2 and since these values belong to the same partition p the final graph should only have one sink. This sink corresponds in fact to partition p.

Parts (A) and (B) of Figure 5.432 respectively show the initial and final graph associated with the **Example** slot. Since we both use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are shown with a double circle.



(A) **(B)**

Figure 5.432: Initial and final graph of the IN_SAME_PARTITION constraint

Signature Note that the sinks of the initial graph cannot become sources of the final graph since isolated vertices are eliminated from the final graph. Since the final graph contains two sources it also includes one arc between a source and a sink. Therefore the minimum number of sinks of the final graph is equal to one. So we can rewrite **NSINK** $= 1$ to **NSINK** ≥ 1 and simplify $\underline{\overline{\textbf{NSINK}}}$ to $\overline{\textbf{NSINK}}$.

Automaton Figure 5.433 depicts the automaton associated with the IN_SAME_PARTITION constraint. Let $\texttt{VALUES}_i$ be the p attribute of the i^{th} item of the PARTITIONS collection. To each triple $(\texttt{VAR1}, \texttt{VAR2}, \texttt{VALUES}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((\texttt{VAR1} \in \texttt{VALUES}_i) \wedge (\texttt{VAR2} \in \texttt{VALUES}_i)) \Leftrightarrow S_i$.

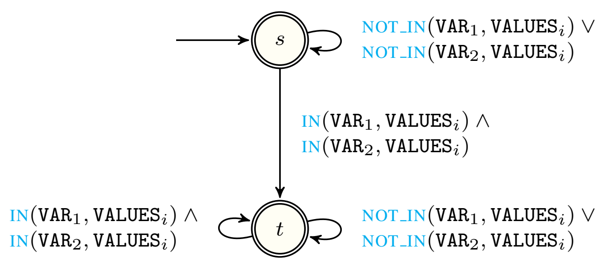


Figure 5.433: Automaton of the IN_SAME_PARTITION constraint

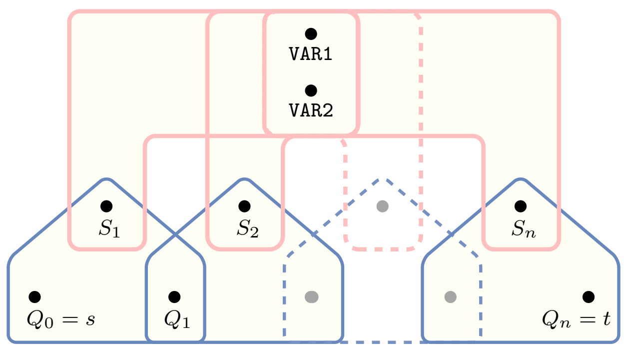


Figure 5.434: Hypergraph of the reformulation corresponding to the automaton of the IN_SAME_PARTITION constraint

5.185 IN_SET

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Used for defining constraints with set variables.

Constraint IN_SET(VAL, SET)

Synonyms DOM, MEMBER.

Arguments

VAL	:	dvar
SET	:	svar

Purpose Constraint variable VAL to belong to set SET.

Example $(3, \{1, 3\})$

Remark When SET is fixed the IN_SET constraint is referenced under the name DOM in **Gecode**.

Systems MEMBER in **Choco**, REL in **Gecode**, DOM in **Gecode**.

Used in BIPARTITE, CLIQUE, CONNECTED, CUTSET, DAG, DISCREPANCY, DISJ, INVERSE_SET, K_CUT, LINK_SET_TO_BOOLEANS, OPEN_ALLDIFFERENT, OPEN_AMONG, OPEN_ATLEAST, OPEN_ATMOST, OPEN_GLOBAL_CARDINALITY, OPEN_GLOBAL_CARDINALITY_LOW_UP, PATH_FROM_TO, PROPER_FOREST, ROOTS, STRONGLY_CONNECTED, SUM, SUM_SET, SYMMETRIC, SYMMETRIC_CARDINALITY, SYMMETRIC_GCC, TOUR.

See also **common keyword:** IN *(value constraint)*.

specialisation: IN_INTERVAL *(set variable replaced by fixed interval)*.

Keywords **constraint arguments:** constraint involving set variables.

constraint type: predefined constraint, value constraint.

modelling: included.

5.186 INCOMPARABLE

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Inspired by incomparable rectangles.

Constraint INCOMPARABLE(VECTOR1, VECTOR2)

Synonym INCOMPARABLES.

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

Restrictions

required(VECTOR1, var)
required(VECTOR2, var)
$|\texttt{VECTOR1}| \geq 1$
$|\texttt{VECTOR2}| \geq 1$
$|\texttt{VECTOR1}| = |\texttt{VECTOR2}|$

Purpose Enforce that when the components of VECTOR1 and VECTOR2 are ordered, and respectively denoted by SVECTOR1 and SVECTOR2, we neither have $\texttt{SVECTOR1}[i].\texttt{var} \leq \texttt{SVECTOR2}[i].\texttt{var}$ (for all $i \in [1, |\texttt{SVECTOR1}|]$) nor have $\texttt{SVECTOR2}[i].\texttt{var} \leq \texttt{SVECTOR1}[i].\texttt{var}$ (for all $i \in [1, |\texttt{SVECTOR1}|]$).

Example $(\langle 16, 2\rangle, \langle 4, 11\rangle)$

The INCOMPARABLE constraint holds since $16 > 4$ and $2 < 11$.

Typical $|\texttt{VECTOR1}| > 1$

Symmetries

- Items of VECTOR1 are permutable.
- Items of VECTOR2 are permutable.
- Arguments are permutable w.r.t. permutation (VECTOR1, VECTOR2).

Used in ALL_INCOMPARABLE.

See also **implies:** LEX_DIFFERENT.

system of constraints: ALL_INCOMPARABLE.

Keywords **characteristic of a constraint:** vector.

constraint arguments: constraint between two collections of variables.

constraint type: predefined constraint.

Cond. implications

- INCOMPARABLE(VECTOR1, VECTOR2)
  with $|\texttt{VECTOR1}| = 2$
  implies DISJOINT(VARIABLES1 : VECTOR1, VARIABLES2 : VECTOR2).

- INCOMPARABLE(VECTOR1, VECTOR2)
  with $|\texttt{VECTOR1}| = 2$
  implies INT_VALUE_PRECEDE_CHAIN(VALUES : VECTOR1, VARIABLES : VECTOR2).

5.187 INCREASING

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin KOALOG

Constraint INCREASING(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restriction required(VARIABLES, var)

Purpose The variables of the collection VARIABLES are increasing.

Example $(\langle 1, 1, 4, 8\rangle)$

The INCREASING constraint holds since $1 \leq 1 \leq 4 \leq 8$.

Typical $|\text{VARIABLES}| > 2$
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

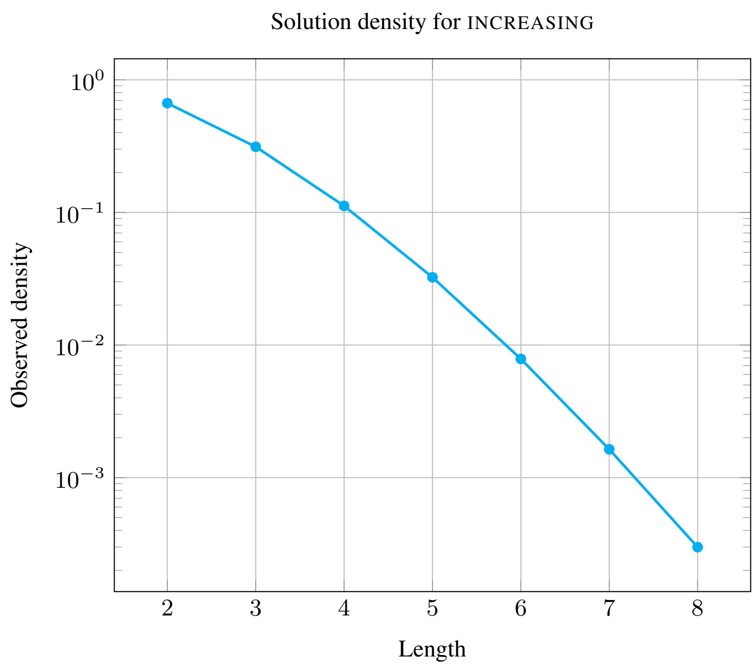
Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

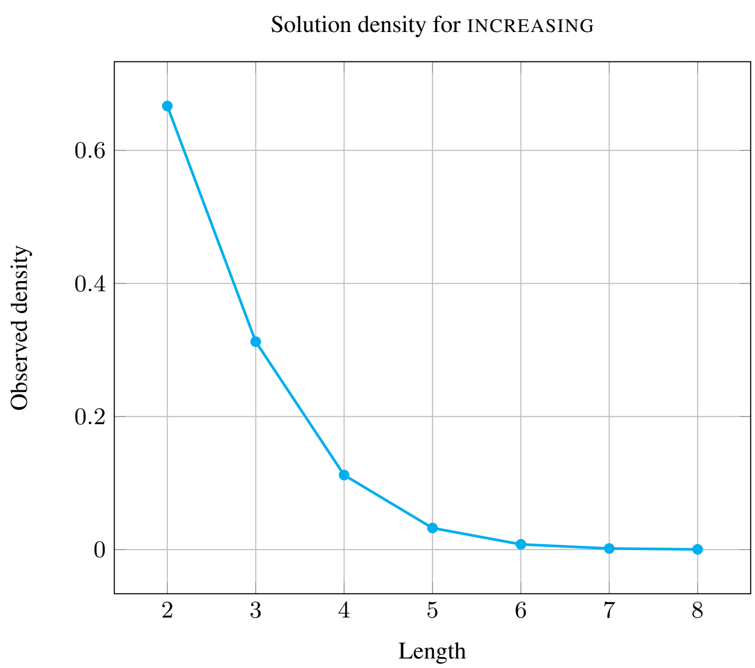
Arg. properties Contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	6	20	70	252	924	3432	12870

Number of solutions for INCREASING: domains $0..n$





Systems INCREASINGNVALUE in **Choco**, REL in **Gecode**, INCREASING in **MiniZinc**.

Used in GLOBAL_CARDINALITY_LOW_UP, INCREASING_GLOBAL_CARDINALITY,

INCREASING_NVALUE, INCREASING_SUM, NVALUE, SUM_CTR.

See also

common keyword: PRECEDENCE, STRICTLY_DECREASING *(order constraint)*.

comparison swapped: DECREASING.

implied by: ALL_EQUAL, INCREASING_GLOBAL_CARDINALITY, INCREASING_NVALUE *(remove* `NVAL` *parameter from* INCREASING_NVALUE*)*, INCREASING_SUM *(remove* `SUM` *parameter from* INCREASING_SUM*)*, STRICTLY_INCREASING.

implies: MULTI_GLOBAL_CONTIGUITY, NO_PEAK, NO_VALLEY.

uses in its reformulation: SORT_PERMUTATION.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: sliding cyclic(1) constraint network(1).

constraint type: decomposition, order constraint.

filtering: arc-consistency.

Arc input(s)	`VARIABLES`
Arc generator	$PATH \mapsto$ `collection(variables1, variables2)`
Arc arity	2
Arc constraint(s)	`variables1.var` $\leq$ `variables2.var`
Graph property(ies)	**NARC** $= \lvert$`VARIABLES`$\rvert - 1$

Graph model Parts (A) and (B) of Figure 5.435 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

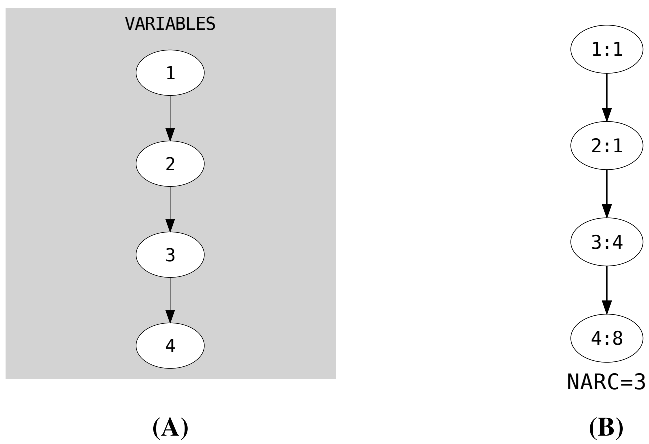


Figure 5.435: Initial and final graph of the INCREASING constraint

Automaton Figure 5.436 depicts the automaton associated with the INCREASING constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $\mathtt{VAR}_i \leq \mathtt{VAR}_{i+1} \Leftrightarrow S_i$.

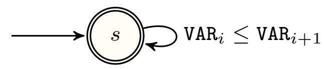


Figure 5.436: Automaton of the INCREASING constraint

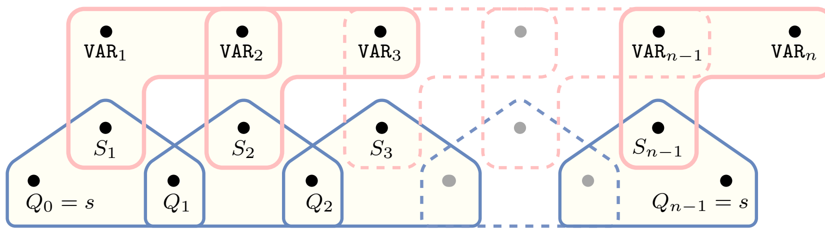


Figure 5.437: Hypergraph of the reformulation corresponding to the automaton of the INCREASING constraint

5.188 INCREASING_GLOBAL_CARDINALITY

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Conjoin GLOBAL_CARDINALITY_LOW_UP and INCREASING.

Constraint INCREASING_GLOBAL_CARDINALITY(VARIABLES, VALUES)

Synonyms INCREASING_GLOBAL_CARDINALITY_LOW_UP, INCREASING_GCC, INCREASING_GCC_LOW_UP.

Arguments

VARIABLES : collection(var−dvar)
VALUES : collection(val−int, omin−int, omax−int)

Restrictions

required(VARIABLES, var)
INCREASING(VARIABLES)
$|\mathtt{VALUES}| > 0$
required(VALUES, [val, omin, omax])
distinct(VALUES, val)
$\mathtt{VALUES.omin} \geq 0$
$\mathtt{VALUES.omax} \leq |\mathtt{VARIABLES}|$
$\mathtt{VALUES.omin} \leq \mathtt{VALUES.omax}$

Purpose The variables of the collection VARIABLES are increasing. In addition, each value $\mathtt{VALUES}[i].\mathtt{val}$ $(1 \leq i \leq |\mathtt{VALUES}|)$ should be taken by at least $\mathtt{VALUES}[i].\mathtt{omin}$ and at most $\mathtt{VALUES}[i].\mathtt{omax}$ variables of the VARIABLES collection.

Example

$$\left(\begin{array}{l}\langle 3,3,6,8\rangle, \\ \left\langle\begin{array}{lll}\mathtt{val}-3 & \mathtt{omin}-2 & \mathtt{omax}-3, \\ \mathtt{val}-5 & \mathtt{omin}-0 & \mathtt{omax}-1, \\ \mathtt{val}-6 & \mathtt{omin}-1 & \mathtt{omax}-2\end{array}\right\rangle\end{array}\right)$$

The INCREASING_GLOBAL_CARDINALITY constraint holds since:

- The values of the collection $\langle 3,3,6,8\rangle$ are sorted in increasing order.
- Values 3, 5 and 6 are respectively used 2 $(2 \leq 2 \leq 3)$, 0 $(0 \leq 0 \leq 1)$ and 1 $(1 \leq 1 \leq 2)$ times within the collection $\langle 3,3,6,8\rangle$ and since no constraint was specified for value 8.

Typical

$|\mathtt{VARIABLES}| > 1$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 1$
$|\mathtt{VALUES}| > 1$
$\mathtt{VALUES.omin} \leq |\mathtt{VARIABLES}|$
$\mathtt{VALUES.omax} > 0$
$\mathtt{VALUES.omax} \leq |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > |\mathtt{VALUES}|$

Typical model $\texttt{nval}(\texttt{VARIABLES.var}) > 2$

Symmetry Items of VALUES are permutable.

Usage This constraint can be used in order to break symmetry in the context of the following pattern. We have a matrix $\mathcal{M}$ of variables with the same constraint on each row and a GLOBAL_CARDINALITY_LOW_UP constraint on each column. Beside lexicographically ordering the rows of $\mathcal{M}$ with a LEX_CHAIN_LESSEQ constraint, one can also state a INCREASING_GLOBAL_CARDINALITY on the first column of $\mathcal{M}$ in order to improve propagation on the corresponding variables.

Reformulation The INCREASING_GLOBAL_CARDINALITY constraint can be expressed in term of a conjunction of a GLOBAL_CARDINALITY_LOW_UP and an INCREASING constraints. Even if we achieve arc-consistency on these two constraints this hinders propagation as shown by the following small example.

We have two variables X and Y $(X \leq Y)$, which both take their values in the set $\{2, 3\}$. In addition, assume that the minimum number of occurrences of values 0, 1 and 2 are respectively equal to 0, 1 and 1. Similarly assume that, the maximum number of occurrences of values 0, 1 and 2 are respectively equal to 1, 1 and 2. The reformulation does not reduce the domain of variables X, Y in any way, while the automaton described in the **Automaton** slot fixes X to 2 and Y to 3.

See also **implies:** GLOBAL_CARDINALITY_LOW_UP, INCREASING.

related: ORDERED_GLOBAL_CARDINALITY.

Keywords **application area:** assignment.

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: value constraint, order constraint.

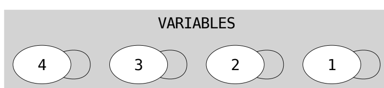
filtering: arc-consistency.

symmetry: symmetry, matrix symmetry.

For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUES.val
Graph property(ies)	• **NVERTEX** $\geq$ VALUES.omin • **NVERTEX** $\leq$ VALUES.omax

Graph model Since we want to express one unary constraint for each value we use the "For all items of VALUES" iterator. Part (A) of Figure 5.438 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.438 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold.



(A)

(B)

Figure 5.438: Initial and final graph of the INCREASING_GLOBAL_CARDINALITY constraint

Automaton A first systematic approach for creating an automaton that only recognises the solutions to the INCREASING_GLOBAL_CARDINALITY constraint could be to:

- First, create an automaton that recognises the solutions to the INCREASING constraint.
- Second, create an automaton that recognises the solutions to the GLOBAL_CARDINALITY_LOW_UP constraint.
- Third, make the product of the two previous automata and minimise the resulting automaton.

However this approach is not going to scale well in practice since the automaton associated with the GLOBAL_CARDINALITY_LOW_UP constraint may have a too big size. Therefore we propose an approach where we directly construct in a single step the automaton that only recognises the solutions to the INCREASING_GLOBAL_CARDINALITY constraint. Note that we do not have any formal proof that the resulting automaton is always minimum.

Without loss of generality, we assume that:

- All items of the VALUES collection are sorted in increasing value on the attribute val.
- All the potential values of the variables of the VARIABLES collection are included within the set of values of the collection VALUES (i.e., the val attribute).[9]
- All values of the VALUES collection for which the attribute omax is set to 0 cannot be assigned to the variables of the VARIABLES collection.[10]

Before defining the states of the automaton, we first need to introduce the following notion. A value $\texttt{VALUES}[v].\texttt{val}$ is *constrained by its maximum number of occurrences* if and only if $\texttt{VALUES}[v].\texttt{omax} \leq 1 \vee \texttt{VALUES}[v].\texttt{omax} < |\texttt{VARIABLES}| - \sum_{u=1,u\neq v}^{|\texttt{VALUES}|} \texttt{VALUES}[u].\texttt{omin}$.[11] Let $\mathcal{V}$ denote the set of constrained values (i.e., their indexes within the collection VALUES) by their respective maximum number of occurrences.

After determining the set $\mathcal{V}$, the omax attribute of each potential value is normalised in the following way:

- For an unconstrained value $\texttt{VALUES}[v].\texttt{val}$ we reset $\texttt{VALUES}[v].\texttt{omax}$ to $\max(1, \texttt{VALUES}[v].\texttt{omin})$.
- For a constrained value $\texttt{VALUES}[v].\texttt{val}$ we reset $\texttt{VALUES}[v].\texttt{omax}$ to 1 if its current value is smaller than 1.

We are now in position to introduce the states of the automaton.

The $1 + \sum_{v=1,v\in\mathcal{V}}^{|\texttt{VALUES}|} \texttt{VALUES}[v].\texttt{omax} + \sum_{v=1,v\notin\mathcal{V}}^{|\texttt{VALUES}|} \texttt{VALUES}[v].\texttt{omin}$ states of the automaton that only accepts solutions to the INCREASING_GLOBAL_CARDINALITY constraint are defined in the following way:

- For the v^{th} item of the collection VALUES we have:
  - If $v \in \mathcal{V}$, $\texttt{VALUES}[v].\texttt{omax}$ states labelled by s_{vo} $(1 \leq o \leq \texttt{VALUES}[v].\texttt{omax})$.

[9] If this is not the case, we can include these values within the VALUES collection and set their minimum and maximum number of occurrences to 0 and $|\texttt{VARIABLES}| - \sum_{v=1}^{|\texttt{VALUES}|} \texttt{VALUES}[v].\texttt{omin}$.

[10] We initially remove such values from all variables of the VARIABLES collection.

[11] When $\texttt{VALUES}[v].\texttt{omax} \leq 1$ we cannot reduce the number of states related to value $\texttt{VALUES}[v].\texttt{val}$ and we therefore consider that we are in the constrained case.

  – If $v \notin \mathcal{V}$, `VALUES`$[v]$.`omin` states labelled by s_{vo} ($1 \leq o \leq$ `VALUES`$[v]$.`omin`).

- We have an initial state labelled by s_{00}.

Terminal states correspond to those states s_{vo} such that, both (1) o is greater than or equal to `VALUES`$[v]$.`omin`, and (2) there is no value item `VALUES`$[w]$ ($w > v$) such that `VALUES`$[w]$.`omin` > 0. Transitions are defined in the following way:

- There is an arc, labelled by `VALUES`$[v]$.`val`, from the initial state s_{00} to every state s_{v1} where `VALUES`$[v]$ is an item for which all values `VALUES`$[u]$.`val` strictly less than `VALUES`$[v]$.`val` verify the condition `VALUES`$[u]$.`omin` $= 0$.
- For each value `VALUES`$[v]$.`val` constrained by its maximum number of occurrences (i.e., $v \in \mathcal{V}$), there is an arc, labelled by `VALUES`$[v]$.`val`, from the state s_{vk} to the state s_{vk+1} for all k in $[1,$ `VALUES`$[v]$.`omax` $- 1]$.
- For each value `VALUES`$[v]$.`val` unconstrained by its maximum number of occurrences (i.e., $v \notin \mathcal{V}$), there is an arc, labelled by `VALUES`$[v]$.`val`, from the state s_{vk} to the state s_{vk+1} for all k in $[1,$ `VALUES`$[v]$.`omin` $- 1]$. There is also a loop, labelled by `VALUES`$[v]$.`val`, from state s_{vk} to the state s_{vk} for $k =$ `VALUES`$[v]$.`omin`.
- For each value `VALUES`$[v]$.`val` constrained by its maximum number of occurrences (i.e., $v \in \mathcal{V}$), there is an arc, labelled by `VALUES`$[w]$.`val`, from state s_{vk} to state s_{w1} ($v < w$) for all k in [`VALUES`$[v]$.`omin`, `VALUES`$[v]$.`omax`] and for all w such that $\forall u \in [v+1, w-1]$: `VALUES`$[u]$.`omin` $= 0$.
- For each value `VALUES`$[v]$.`val` unconstrained by its maximum number of occurrences (i.e., $v \notin \mathcal{V}$), there is an arc, labelled by `VALUES`$[w]$.`val`, from state s_{vk} to state s_{w1} ($v < w$) for $k =$ `VALUES`$[v]$.`omin` and for all w such that $\forall u \in [v+1, w-1]$: `VALUES`$[u]$.`omin` $= 0$.

Figure 5.439 depicts the automaton associated with the INCREASING_GLOBAL_CARDINALITY constraint of the **Example** slot. For this purpose we assume without loss of generality that we have four decision variables that all take their potential values within interval $[3, 8]$. Consequently, values 4, 7 and 8 are first added to the items of the `VALUES` collection. Both values 3 and 6 are unconstrained by their respective maximum number of occurrences. Therefore their `omax` attributes are respectively reduced to 2 and 1. All other values, namely values 4, 5, 7 and 8, are constrained values. The INCREASING_GLOBAL_CARDINALITY constraint holds since the corresponding sequence of visited states, s_{00} s_{11} s_{12} s_{41} s_{61}, ends up in an accepting state (i.e., accepting states are denoted graphically by a double circle in the figure). Note that non initial states are first indexed by the position of an item within the `VALUES` collection, and not by the value itself (e.g., within s_{12} the 1 designates value 3). For instance state s_{11} depicts the fact that the automaton has already recognised a single occurrence of value 3, while s_{12} corresponds to the fact that the automaton has already seen at least two occurrences of value 3.[12]

[12] The at least comes from the loop on state s_{12}.

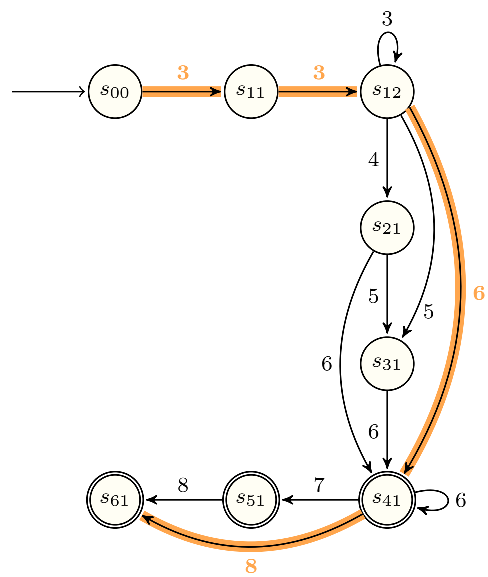


Figure 5.439: Automaton of the INCREASING_GLOBAL_CARDINALITY constraint of the **Example** slot: the path corresponding to the solution $\langle \mathbf{3}, \mathbf{3}, \mathbf{6}, \mathbf{8} \rangle$ is depicted by thick orange arcs

5.189 INCREASING_NVALUE

DESCRIPTION LINKS GRAPH AUTOMATON

Origin Conjoin NVALUE and INCREASING.

Constraint INCREASING_NVALUE(NVAL, VARIABLES)

Arguments

NVAL	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

NVAL $\geq$ min(1, |VARIABLES|)
NVAL $\leq$ |VARIABLES|
required(VARIABLES, var)
INCREASING(VARIABLES)

Purpose The variables of the collection VARIABLES are increasing. In addition, NVAL is the number of distinct values taken by the variables of the collection VARIABLES.

Example

$(2, \langle 6, 6, 8, 8, 8\rangle)$
$(1, \langle 6, 6, 6, 6, 6\rangle)$
$(5, \langle 0, 2, 3, 6, 7\rangle)$

The first INCREASING_NVALUE constraint (see Figure 5.440 for a graphical representation) holds since:

- The values of the collection $\langle 6, 6, 8, 8, 8\rangle$ are sorted in increasing order.
- NVAL $= 2$ is set to the number of distinct values occurring within the collection $\langle 6, 6, 8, 8, 8\rangle$.

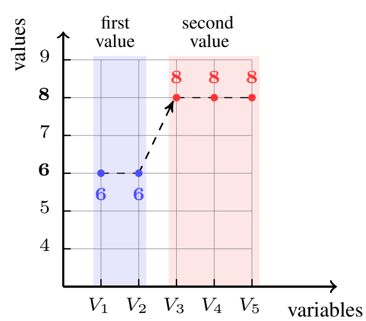


Figure 5.440: Illustration of the first example of the **Example** slot: five variables V_1, V_2, V_3, V_4, V_5 respectively fixed to values 6, 6, 8, 8 and 8, and the corresponding number of distinct values NVAL $= 2$

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

Typical model

$\texttt{nval}(\texttt{VARIABLES.var}) > 2$

Symmetry

One and the same constant can be added to the `var` attribute of all items of `VARIABLES`.

Arg. properties

Functional dependency: `NVAL` determined by `VARIABLES`.

Algorithm

A complete filtering algorithm in a linear time complexity over the sum of the domain sizes is described in [51].

Reformulation

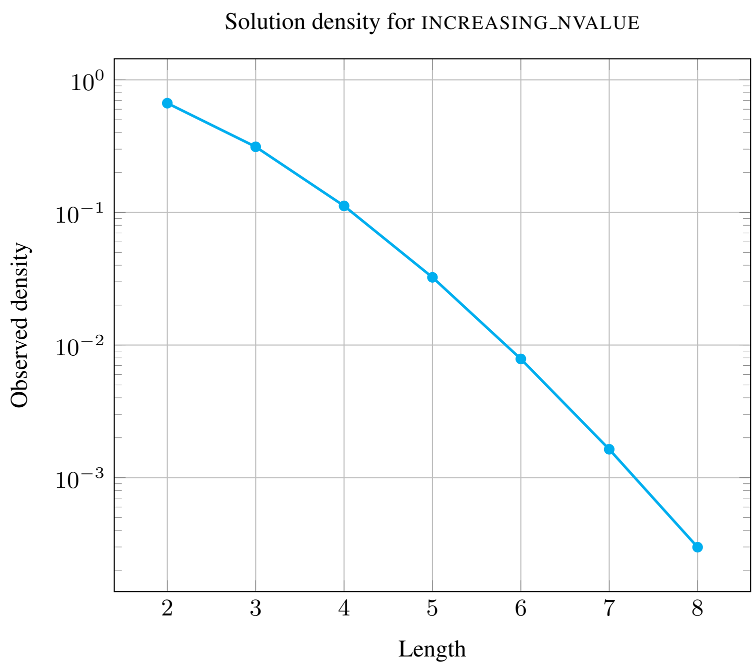
The INCREASING_NVALUE constraint can be expressed in term of a conjunction of a NVALUE and an INCREASING constraints (i.e., a chain of non strict inequality constraints on adjacent variables of the collection `VARIABLES`). But as shown by the following example, $V_1 \in [1,2]$, $V_2 \in [1,2]$, $V_1 \leq V_2$, NVALUE$(2, \langle V_1, V_2\rangle)$, this hinders propagation (i.e., the unique solution $V_1 = 1$, $V_2 = 2$ is not directly obtained after stating all the previous constraints).

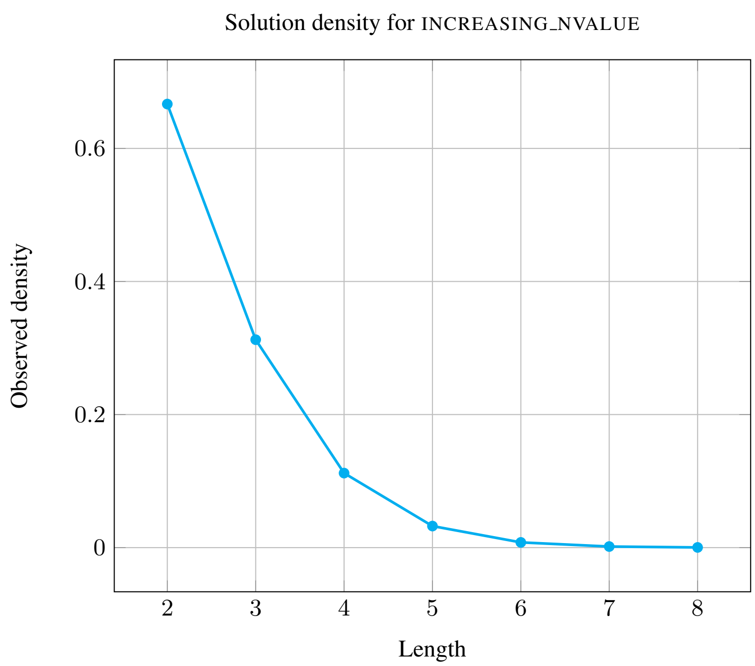
A better reformulation achieving arc-consistency uses the SEQ_BIN constraint [321] that we now introduce. Given `N` a domain variable, `X` a sequence of domain variables, and `C` and `B` two binary constraints, SEQ_BIN$(\texttt{N}, \texttt{X}, \texttt{C}, \texttt{B})$ holds if (1) `N` is equal to the number of `C`-stretches in the sequence `X`, and (2) `B` holds on any pair of consecutive variables in `X`. A `C`-stretch is a generalisation of the notion of stretch introduced by G. Pesant [316], where the equality constraint is made explicit by replacing it by a binary constraint `C`, i.e., a `C`-stretch is a maximal length subsequence of `X` for which the binary constraint `C` is satisfied on consecutive variables. INCREASING_NVALUE$(\texttt{NVAL}, \texttt{VARIABLES})$ can be reformulated as SEQ_BIN$(\texttt{NVAL}, \texttt{VARIABLES}, =, \leq)$.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	6	20	70	252	924	3432	12870

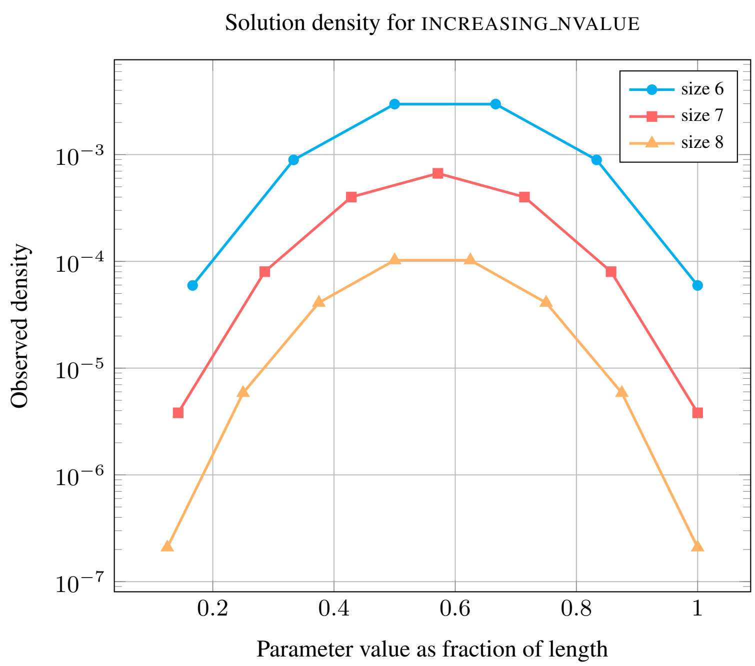
Number of solutions for INCREASING_NVALUE: domains $0..n$

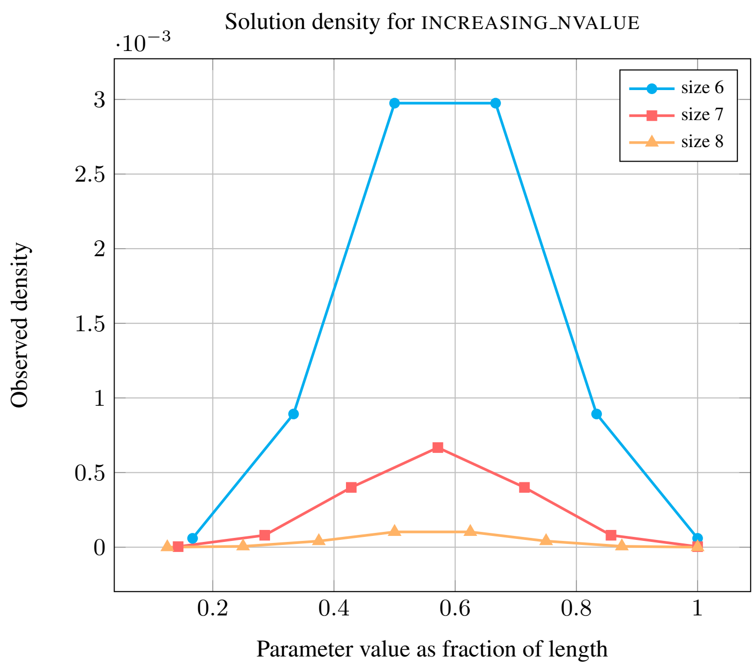

Solution density for INCREASING_NVALUE
10^{0}
10^{-1}
10^{-2}
10^{-3}
Observed density
2
3
4
5
6
7
8
Length


Solution density for INCREASING_NVALUE
0.6
0.4
0.2
0
Observed density
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		6	20	70	252	924	3432	12870
Parameter value	1	3	4	5	6	7	8	9
	2	3	12	30	60	105	168	252
	3	-	4	30	120	350	840	1764
	4	-	-	5	60	350	1400	4410
	5	-	-	-	6	105	840	4410
	6	-	-	-	-	7	168	1764
	7	-	-	-	-	-	8	252
	8	-	-	-	-	-	-	9

Solution count for INCREASING_NVALUE: domains $0..n$





Systems INCREASINGNVALUE in **Choco**.

See also **implies:** INCREASING *(remove* `NVAL` *parameter from* INCREASING_NVALUE*)*, NVALUE, NVISIBLE_FROM_START.

related: INCREASING_NVALUE_CHAIN.

shift of concept: ORDERED_NVECTOR *(*`variable` *replaced by* `vector` *and* $\leq$ *replaced by* LEX_LESSEQ*)*.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: counting constraint, value partitioning constraint, order constraint.

filtering: arc-consistency.

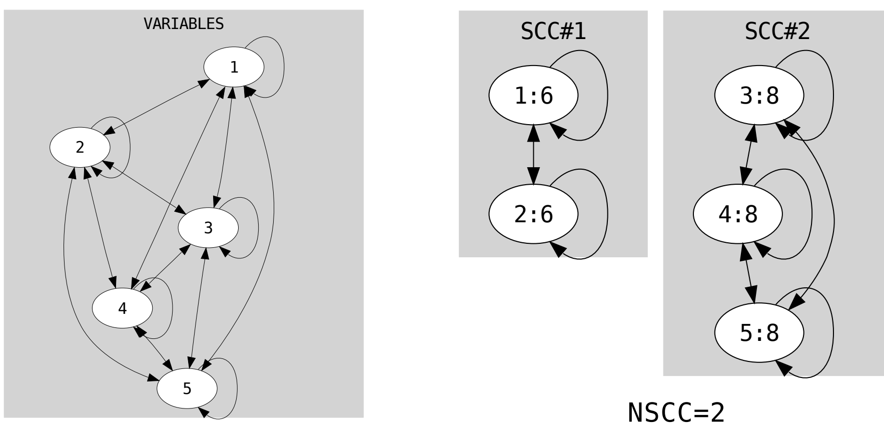
final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, number of distinct values, functional dependency.

symmetry: symmetry.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var $=$ variables2.var
Graph property(ies)	**NSCC**$=$ NVAL
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.441 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The 2 following values 6 and 8 are used by the variables of the VARIABLES collection.



(A) (B)

Figure 5.441: Initial and final graph of the INCREASING_NVALUE constraint

Automaton

A first systematic approach for creating an automaton that only recognises the solutions to the INCREASING_NVALUE constraint could be to:

- First, create an automaton that recognises the solutions to the INCREASING constraint.
- Second, create an automaton that recognises the solutions to the NVALUE constraint.
- Third, make the product of the two previous automata and minimise the resulting automaton.

However this approach is not going to scale well in practice since the automaton associated with the NVALUE constraint has a too big size. Therefore we propose an approach where we directly construct in a single step the automaton that only recognises the solutions to the INCREASING_NVALUE constraint. Note that we do not have any formal proof that the resulting automaton is always minimum.

Without loss of generality, assume that the collection of variables `VARIABLES` contains at least one variable (i.e., $|\texttt{VARIABLES}| \geq 1$). Let l, m, n, min and max respectively denote the minimum and maximum possible value of variable `NVAL`, the number of variables of the collection `VARIABLES`, the smallest value that can be assigned to the variables of `VARIABLES`, and the largest value that can be assigned to the variables of `VARIABLES`. Let $s = max - min + 1$ denote the total number of potential values. Clearly, the maximum number of distinct values that can be assigned to the variables of the collection `VARIABLES` cannot exceed the quantity $d = \min(m, n, s)$. The $\frac{s\cdot(s+1)}{2} - \frac{(s-d)\cdot(s-d+1)}{2} + 1$ states of the automaton that only accepts solutions to the INCREASING_NVALUE constraint can be defined in the following way:

- We have an initial state labelled by s_{00}.
- We have $\frac{s\cdot(s+1)}{2} - \frac{(s-d)\cdot(s-d+1)}{2}$ states labelled by s_{ij} $(1 \leq i \leq d, i \leq j \leq s)$. The first index i of a state s_{ij} corresponds to the number of distinct values already encountered, while the second index j denotes the the current value (i.e., more precisely the index of the current value, where the minimum value has index 1).

Terminal states depend on the possible values of variable `NVAL` and correspond to those states s_{ij} such that i is a possible value for variable `NVAL`. Note that we assume no further restriction on the domain of `NVAL` (otherwise the set of accepting states needs to be reduced in order to reflect the current set of possible values of `NVAL`). Three classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by $min + j - 1$, from the initial state s_{00} to the state s_{1j} $(1 \leq j \leq s)$.
2. There is a loop, labelled by $min + j - 1$ for every state s_{ij} $(1 \leq i \leq d, i \leq j \leq s)$.
3. $\forall i \in [1, d-1], \forall j \in [i, s], \forall k \in [j+1, s]$ there is a transition labelled by $min+k-1$ from s_{ij} to s_{i+1k}.

We respectively have s transitions of class 1, $\frac{s\cdot(s+1)}{2} - \frac{(s-d)\cdot(s-d+1)}{2}$ transitions of class 2, and $\frac{(s-1)\cdot s\cdot(s+1)}{6} - \frac{(s-d)\cdot(s-d+1)\cdot(s-d+2)}{6}$ transitions of class 3.

Note that all states s_{ij} such that $i + s - j < l$ can be discarded since they do not allow to reach the minimum number of distinct values required l.

Part (A) of Figure 5.442 depicts the automaton associated with the INCREASING_NVALUE constraint of the first example of the **Example** slot. For this purpose, we assume that variable NVAL is fixed to value 2 and that variables of the collection VARIABLES take their values within interval $[6, 8]$. Part (B) of Figure 5.442 represents the simplified automaton where all states that do not allow to reach an accepting state were removed. The corresponding INCREASING_GLOBAL_CARDINALITY constraint holds since the corresponding sequence of visited states, s_{00} s_{11} s_{11} s_{23} s_{23} s_{23}, ends up in an accepting state (i.e., accepting states are denoted graphically by a double circle).

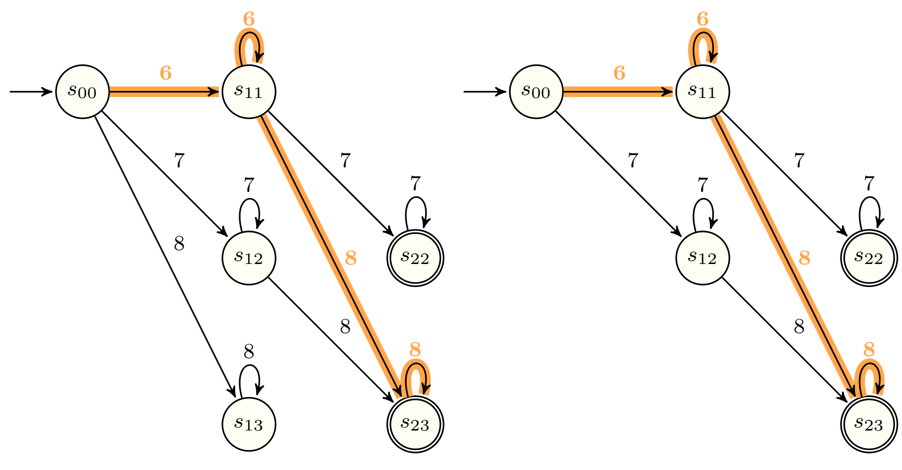


Figure 5.442: Automaton – Part (A) – and simplified automaton – Part (B) – of the INCREASING_NVALUE$(2, \langle 6, 6, 8, 8, 8\rangle)$ constraint of the first example of the **Example** slot: the path corresponding to the second argument $\langle 6, 6, 8, 8, 8\rangle$ is depicted by thick orange arcs, where the self-loop on state s_{23} is applied twice

Figure 5.443 depicts a second deterministic automaton with one counter associated with the INCREASING_NVALUE constraint, where the argument NVAL is unified to the final value of the counter.

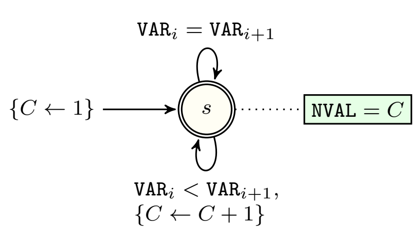


Figure 5.443: Automaton (with one counter) of the INCREASING_VALUE constraint for a non-empty collection of variables

5.190 INCREASING_NVALUE_CHAIN

DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from INCREASING_NVALUE.

Constraint INCREASING_NVALUE_CHAIN(NVAL, VARIABLES)

Arguments

NVAL	:	dvar
VARIABLES	:	collection(b−dvar, var−dvar)

Restrictions

NVAL $\geq$ min$(1, |$VARIABLES$|)$
NVAL $\leq |$VARIABLES$|$
required(VARIABLES, [b, var])
VARIABLES.b ≥ 0
VARIABLES.b ≤ 1

Purpose

For each consecutive pair of items VARIABLES$[i]$, VARIABLES$[i+1]$ $(1 \leq i < |$VARIABLES$|)$ of the VARIABLES collection at least one of the following conditions hold:

1. VARIABLES$[i+1]$.b $= 0$,
2. VARIABLES$[i]$.var $\leq$ VARIABLES$[i+1]$.var.

In addition, NVAL is equal to number of pairs of variables VARIABLES$[i]$, VARIABLES$[i+1]$ $(1 \leq i < |$VARIABLES$|)$ plus one, which verify at least one of the following conditions:

1. VARIABLES$[i+1]$.b $= 0$,
2. VARIABLES$[i]$.var $<$ VARIABLES$[i+1]$.var.

Note that VARIABLES$[1]$.b is not referenced at all in the previous definition (i.e., its value does not influence at all the values assigned to the other variables).

Example

$$\left(6, \left\langle \begin{array}{ll} \texttt{b}-0 & \texttt{var}-2, \\ \texttt{b}-1 & \texttt{var}-4, \\ \texttt{b}-1 & \texttt{var}-4, \\ \texttt{b}-1 & \texttt{var}-4, \\ \texttt{b}-0 & \texttt{var}-4, \\ \texttt{b}-1 & \texttt{var}-8, \\ \texttt{b}-0 & \texttt{var}-1, \\ \texttt{b}-0 & \texttt{var}-7, \\ \texttt{b}-1 & \texttt{var}-7 \end{array} \right\rangle \right)$$

The INCREASING_NVALUE_CHAIN constraint holds since:

1. The condition VARIABLES$[i+1]$.b $= 0 \vee$ VARIABLES$[i]$.var $\leq$ VARIABLES$[i+1]$.var holds for every pair of adjacent items of the VARIABLES collection:

- For the pair $(\texttt{VARIABLES}[1].\texttt{var}, \texttt{VARIABLES}[2].\texttt{var})$ we have $\texttt{VARIABLES}[1].\texttt{var} \leq \texttt{VARIABLES}[2].\texttt{var}$ $(2 \leq 4)$.
- For the pair $(\texttt{VARIABLES}[2].\texttt{var}, \texttt{VARIABLES}[3].\texttt{var})$ we have $\texttt{VARIABLES}[2].\texttt{var} \leq \texttt{VARIABLES}[3].\texttt{var}$ $(4 \leq 4)$.
- For the pair $(\texttt{VARIABLES}[3].\texttt{var}, \texttt{VARIABLES}[4].\texttt{var})$ we have $\texttt{VARIABLES}[3].\texttt{var} \leq \texttt{VARIABLES}[4].\texttt{var}$ $(4 \leq 4)$.
- For the pair $(\texttt{VARIABLES}[4].\texttt{var}, \texttt{VARIABLES}[5].\texttt{var})$ we have $\texttt{VARIABLES}[5].\texttt{b} = 0$.
- For the pair $(\texttt{VARIABLES}[5].\texttt{var}, \texttt{VARIABLES}[6].\texttt{var})$ we have $\texttt{VARIABLES}[5].\texttt{var} \leq \texttt{VARIABLES}[6].\texttt{var}$ $(4 \leq 8)$.
- For the pair $(\texttt{VARIABLES}[6].\texttt{var}, \texttt{VARIABLES}[7].\texttt{var})$ we have $\texttt{VARIABLES}[7].\texttt{b} = 0$.
- For the pair $(\texttt{VARIABLES}[7].\texttt{var}, \texttt{VARIABLES}[8].\texttt{var})$ we have $\texttt{VARIABLES}[8].\texttt{b} = 0$.
- For the pair $(\texttt{VARIABLES}[8].\texttt{var}, \texttt{VARIABLES}[9].\texttt{var})$ we have $\texttt{VARIABLES}[8].\texttt{var} \leq \texttt{VARIABLES}[9].\texttt{var}$ $(7 \leq 7)$.

2. NVAL is equal to number of pairs of variables $\texttt{VARIABLES}[i], \texttt{VARIABLES}[i+1]$ $(1 \leq i < |\texttt{VARIABLES}|)$ plus one which verify at least $\texttt{VARIABLES}[i+1].\texttt{b} = 0 \vee \texttt{VARIABLES}[i].\texttt{var} < \texttt{VARIABLES}[i+1].\texttt{var}$. Beside the *plus one*, the following five pairs contribute for 1 in NVAL:

   - For the pair $(\texttt{VARIABLES}[1].\texttt{var}, \texttt{VARIABLES}[2].\texttt{var})$ we have $\texttt{VARIABLES}[1].\texttt{var} \leq \texttt{VARIABLES}[2].\texttt{var}$ $(2 < 4)$.
   - For the pair $(\texttt{VARIABLES}[4].\texttt{var}, \texttt{VARIABLES}[5].\texttt{var})$ we have $\texttt{VARIABLES}[5].\texttt{b} = 0$.
   - For the pair $(\texttt{VARIABLES}[5].\texttt{var}, \texttt{VARIABLES}[6].\texttt{var})$ we have $\texttt{VARIABLES}[5].\texttt{var} \leq \texttt{VARIABLES}[6].\texttt{var}$ $(4 < 8)$.
   - For the pair $(\texttt{VARIABLES}[6].\texttt{var}, \texttt{VARIABLES}[7].\texttt{var})$ we have $\texttt{VARIABLES}[7].\texttt{b} = 0$.
   - For the pair $(\texttt{VARIABLES}[7].\texttt{var}, \texttt{VARIABLES}[8].\texttt{var})$ we have $\texttt{VARIABLES}[8].\texttt{b} = 0$.

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.b}) > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

See also

related: INCREASING_NVALUE, NVALUE, ORDERED_NVECTOR.

Keywords

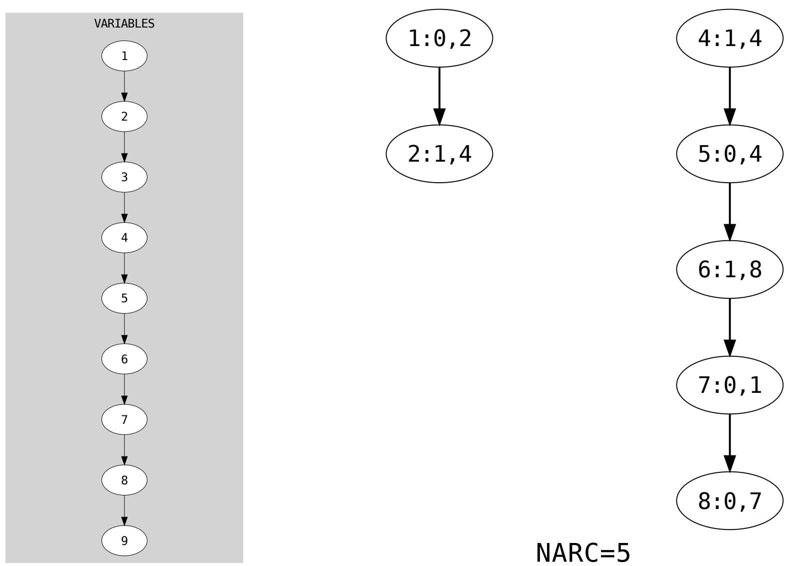
constraint type: counting constraint, order constraint.

modelling: number of distinct values.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables2.b $= 0 \lor$ variables1.var $\leq$ variables2.var
Graph property(ies)	**NARC** $= \vert$VARIABLES$\vert - 1$

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables2.b $= 0 \lor$ variables1.var $<$ variables2.var
Graph property(ies)	**NARC** $=$ NVAL $- 1$

Graph model Parts (A) and (B) of Figure 5.444 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the **NARC** graph property the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.444: Initial and final graph of the INCREASING_NVALUE_CHAIN constraint

Automaton Without loss of generality, assume that the collection VARIABLES contains at least one variable (i.e., $|\texttt{VARIABLES}| \geq 1$). Let l, m, n, min and max respectively denote the minimum and maximum possible value of variable NVAL, the number of items of the collection VARIABLES, the smallest value that can be assigned to $\texttt{VARIABLES}[i].\texttt{var}$ $(1 \leq i \leq n)$, and the largest value that can be assigned to $\texttt{VARIABLES}[i].\texttt{var}$ $(1 \leq i \leq n)$. Let $s = max - min + 1$ denote the total number of potential values. Clearly, the maximum value of NVAL cannot exceed the quantity $d = \min(m, n)$. The states of the automaton that only accepts solutions to the INCREASING_NVALUE_CHAIN constraint can be defined in the following way:

- We have an initial state labelled by s_{00}.
- We have $d \cdot s$ states labelled by s_{ij} $(1 \leq i \leq d, 1 \leq j \leq s)$.

Terminal states depend on the possible values of variable NVAL and correspond to those states s_{ij} such that i is a possible value for variable NVAL. Note that we assume no further restriction on the domain of NVAL (otherwise the set of accepting states needs to be reduced in order to reflect the current set of possible values of NVAL).

Transitions of the automaton are labelled by a pair of values (α, β) and correspond to a condition of the form $\texttt{VARIABLES}[i].\texttt{b} = \alpha \wedge \texttt{VARIABLES}[i].\texttt{var} = \beta$, $(1 \leq i \leq n)$. Characters $*$ and $+$ respectively represent all values in $\{0, 1\}$ and all values in $\{min, min + 1, \ldots, max\}$. Four classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by the pair $(*, min + j - 1)$, from the initial state s_{00} to the state s_{1j} $(1 \leq j \leq s)$. We use the $*$ character since $\texttt{VARIABLES}[1].\texttt{b}$ is not use at all in the definition of the INCREASING_NVALUE_CHAIN constraint.
2. There is a loop, labelled by the pair $(1, min + j - 1)$ for every state s_{ij} $(1 \leq i \leq d, 1 \leq j \leq s)$.
3. $\forall i \in [1, d-1], \forall j \in [1, s], \forall k \in [j+1, s]$ there is a transition labelled by the pair $(1, min + k - 1)$ from s_{ij} to s_{i+1k}.
4. $\forall i \in [1, d-1], \forall j \in [1, s]$ there is a transition labelled by the pair $(0, +)$ from s_{ij} to $s_{i+1\,1}$.

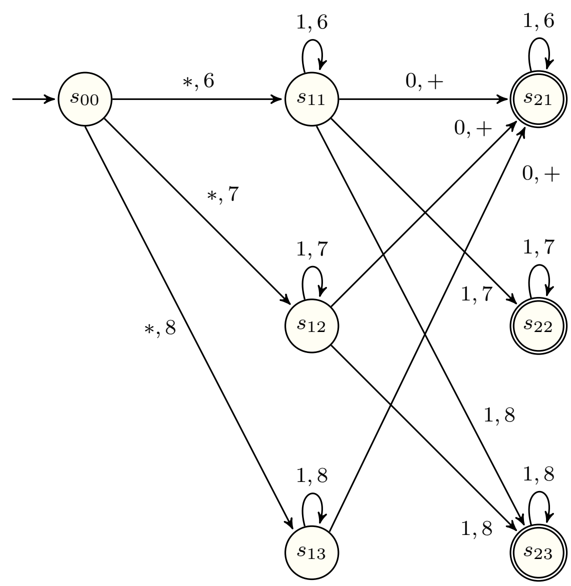


Figure 5.445: Automaton of the INCREASING_NVALUE_CHAIN constraint under the hypothesis that all variables are assigned a value in $\{6, 7, 8\}$ and that NVAL is equal to 2. The character $*$ on a transition corresponds to a 0 or to a 1 and the $+$ corresponds to a 6, 7 or 8.

5.191 INCREASING_PEAK

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from PEAK and INCREASING.

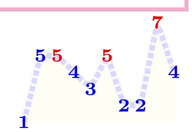
Constraint INCREASING_PEAK(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions |VARIABLES| > 0
required(VARIABLES, var)

Purpose A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \dots, V_m$ is a *peak* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \dots = V_k$ and $V_k > V_{k+1}$.
When considering all the peaks of the sequence VARIABLES from left to right enforce all peaks to be increasing, i.e. the altitude of each peak is greater than or equal to the altitude of its preceding peak when it exists.

Example $(\langle 1, 5, 5, 3, 5, 2, 2, 7, 4\rangle)$



The INCREASING_PEAK constraint holds since the sequence 1 5 **5** 3 **5** 2 2 **7** 4 contains three peaks, in bold, that are increasing.

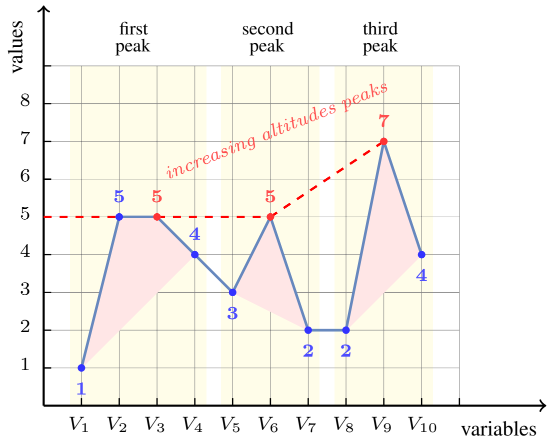


Figure 5.446: Illustration of the **Example** slot: a sequence of ten variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10} respectively fixed to values 1, 5, 5, 4, 3, 5, 2, 2, 7, 4 and its corresponding three peaks, in red, respectively located at altitudes 5, 5 and 7

Typical

$|\texttt{VARIABLES}| \geq 7$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$\texttt{PEAK}(\texttt{VARIABLES.var}) \geq 3$

Typical model

$\texttt{nval}(\texttt{VARIABLES.var}) > 2$

Symmetry

One and the same constant can be added to the `var` attribute of all items of `VARIABLES`.

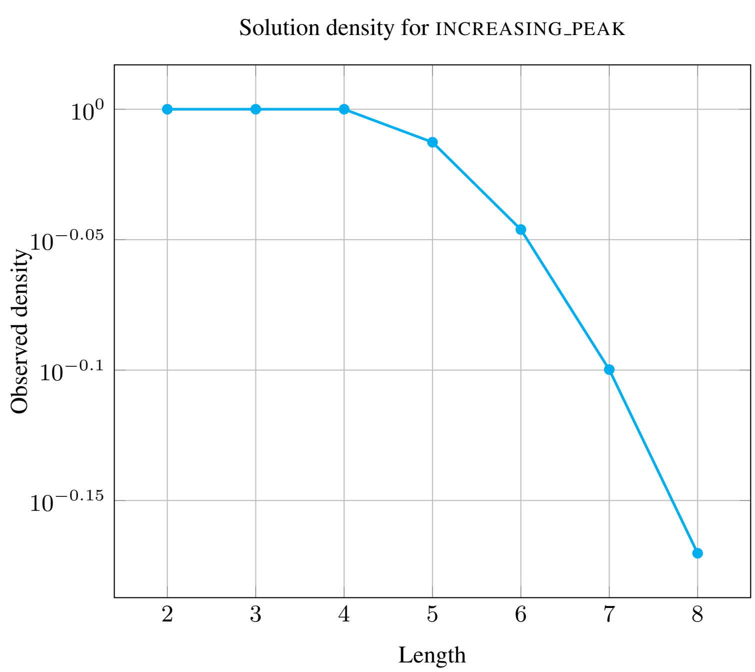
Arg. properties

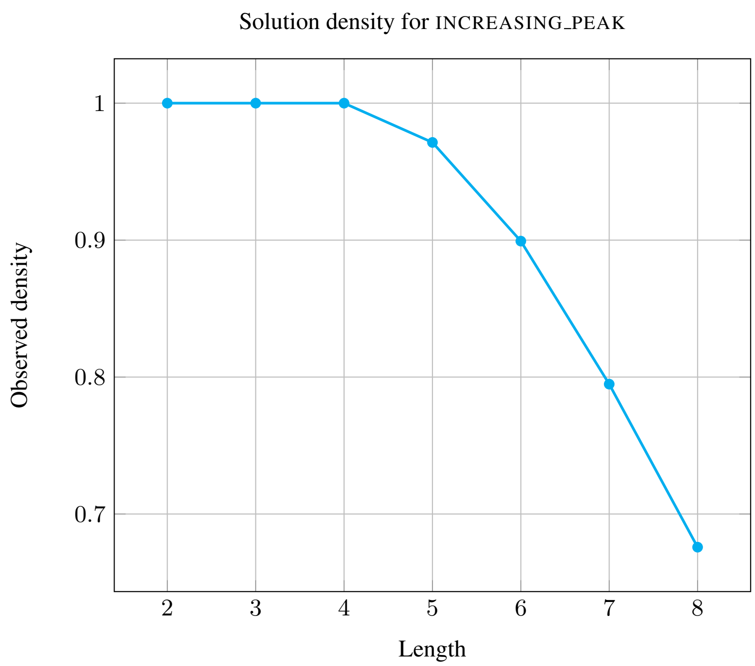
- Prefix-contractible wrt. `VARIABLES`.
- Suffix-contractible wrt. `VARIABLES`.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7553	105798	1666878	29090469

Number of solutions for INCREASING_PEAK: domains $0..n$





See also **implied by:** ALL_EQUAL_PEAK.

related: DECREASING_PEAK, PEAK.

Keywords **characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

Cond. implications INCREASING_PEAK(VARIABLES)
with PEAK(VARIABLES.var) > 0
implies NOT_ALL_EQUAL(VARIABLES).

Automaton Figure 5.447 depicts the automaton associated with the INCREASING_PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: initial stationary or decreasing mode	$(\{= \mid >\}^*)$
u	: increasing (before first potential peak) mode	$(< \{< \mid =\}^*)$
v	: decreasing (after a peak) mode	$(> \{> \mid =\}^*)$
w	: increasing (after a peak) mode	$(< \{< \mid =\}^*)$

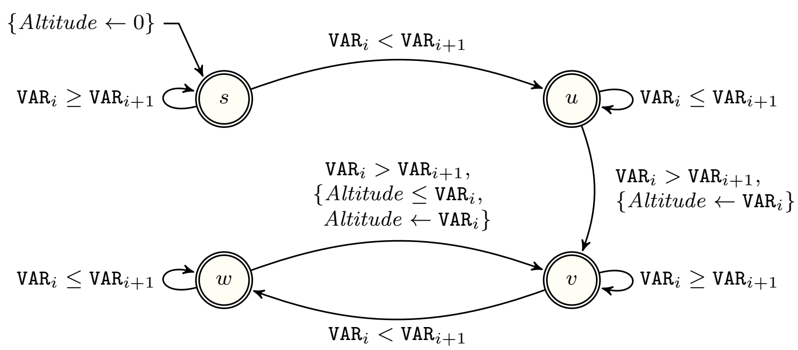


Figure 5.447: Automaton for the INCREASING_PEAK constraint (note the conditional transition from state w to state v testing that the counter $Altitude$ is less than or equal to $\mathtt{VAR}_i$ for enforcing that all peaks from left to right are in increasing altitude)

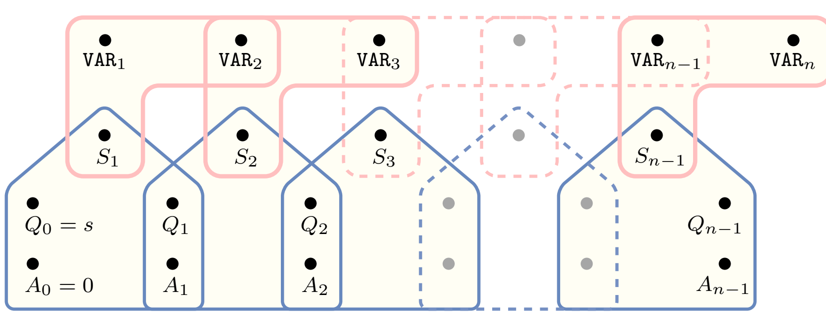


Figure 5.448: Hypergraph of the reformulation corresponding to the automaton of the INCREASING_PEAK constraint where A_i stands for the value of the counter $Altitude$ (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.192 INCREASING_SUM

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin

Conjoin INCREASING and SUM_CTR.

Constraint

INCREASING_SUM(VARIABLES, S)

Synonyms

INCREASING_SUM_CTR, INCREASING_SUM_EQ.

Arguments

VARIABLES	:	collection(var−dvar)
S	:	dvar

Restrictions

required(VARIABLES, var)
INCREASING(VARIABLES)

Purpose

The variables of the collection VARIABLES are increasing. In addition, S is the sum of the variables of the collection VARIABLES.

Example

$(\langle 3, 3, 6, 8\rangle, 20)$

The INCREASING_SUM constraint holds since:

- The values of the collection $\langle 3, 3, 6, 8\rangle$ are sorted in increasing order.
- $\mathtt{S} = 20$ is set to the sum $\langle 3 + 3 + 6 + 8\rangle$.

Typical

$|\mathtt{VARIABLES}| > 1$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 1$

Typical model

$\mathtt{nval}(\mathtt{VARIABLES.var}) > 2$

Arg. properties

Functional dependency: S determined by VARIABLES.

Usage

The INCREASING_SUM constraint can be used for breaking some symmetries in bin packing problems. Given a set of n bins with the same maximum capacity, and a set of items each of them with a specific height, the problem is to pack all items in the bins. To break symmetry we order bins by increasing use. This is done by introducing a variable x_i $(0 \leq i < n)$ for each bin i giving its use, i.e., the sum of items heights assigned to bin i, and by posting the following INCREASING_SUM$(\langle x_0, x_1, \ldots, x_{n-1}\rangle, s)$ where s denotes the sum of the heights of all the items to pack.

Algorithm

A linear time filtering algorithm achieving bound-consistency for the INCREASING_SUM constraint is described in [324]. This algorithm was motivated by the fact that achieving bound-consistency on the inequality constraints and on the sum constraint independently hinders propagation, as illustrated by the following small example, where the maximum value of x_1 is not reduced to 2: $x_1 \in [1, 3]$, $x_2 \in [2, 5]$, $s \in [5, 6]$, $x_1 < x_2$, $x_1 + x_2 = s$.

Given an INCREASING_SUM$(\langle x_0, x_1, \ldots, x_{n-1}\rangle, s)$ constraint, the bound-consistency algorithm consists of three phases:

1. A normalisation phase adjusts the minimum and maximum value of variables $x_0, x_1, \ldots, x_{n-1}$ with respect to the chain of inequalities $x_0 \leq x_1 \leq \cdots \leq x_{n-1}$. A forward phase adjusts the minimum value of $x_1, x_2, \ldots, x_{n-1}$ (i.e., $\underline{x_{i+1}} \geq \underline{x_i}$), while a backward phase adjusts the maximum value of $x_{n-2}, x_{n-1}, \ldots, x_0$ (i.e., $\overline{x_{i-1}} \leq \overline{x_i}$).

2. A phase restricts the minimum and maximum value of the sum variable s with respect to the chain of inequalities $x_0 \leq x_1 \leq \cdots \leq x_{n-1}$ (i.e., $\underline{s} \geq \sum_{0 \leq i < n} \underline{x_i}$ and $\overline{s} \leq \sum_{0 \leq i < n} \overline{x_i}$).

3. A final phase reduces the minimum and maximum value of variables $x_0, x_1, \ldots, x_{n-1}$ both from the bounds of s and from the chain of inequalities. Without loss of generality we now focus on the pruning of the maximum value of variables $x_0, x_1, \ldots, x_{n-1}$. For this purpose we first need to introduce the notion of *last intersecting index of a variable* x_i, denoted by $last_i$. This corresponds to the greatest index in $[i+1, n-1]$ such that $\overline{x_i} > \underline{x_{last_i}}$, or i if no such integer exists. Then the increase of the minimum value of s when x_i is equal to $\overline{x_i}$ is equal to $\sum_{k \in [i, last_i]} (\overline{x_i} - \underline{x_k})$. When this increase exceeds the available margin, i.e. $\overline{s} - \sum_{0 \leq i < n} \underline{x_i}$, we update the maximum value of x_i.

We illustrate a part of the final phase on the following example INCREASING_SUM$(\langle x_0, x_1, x_2, x_3, x_4, x_5 \rangle, s)$, where $x_0 \in [2,6]$, $x_1 \in [4,7]$, $x_2 \in [4,7]$, $x_3 \in [5,7]$, $x_4 \in [6,9]$, $x_5 \in [7,9]$ and $s \in [28,29]$. Observe that the domains are consistent with the first two phases of the algorithm, since,

1. the minimum (and maximum) values of variables $x_0, x_1, x_2, x_3, x_4, x_5$ are increasing,

2. the sum of the minimum of the variables $x_0, x_1, x_2, x_3, x_4, x_5$, i.e., 28 is less than or equal to the maximum value of s,

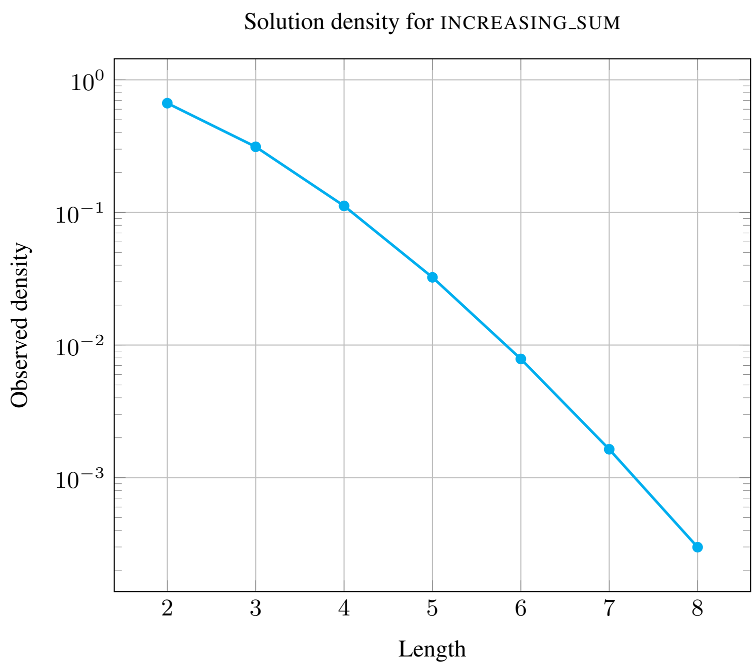
3. the sum of the maximum of the variables $x_0, x_1, x_2, x_3, x_4, x_5$, i.e., 45 is greater than or equal to the minimum value of s.

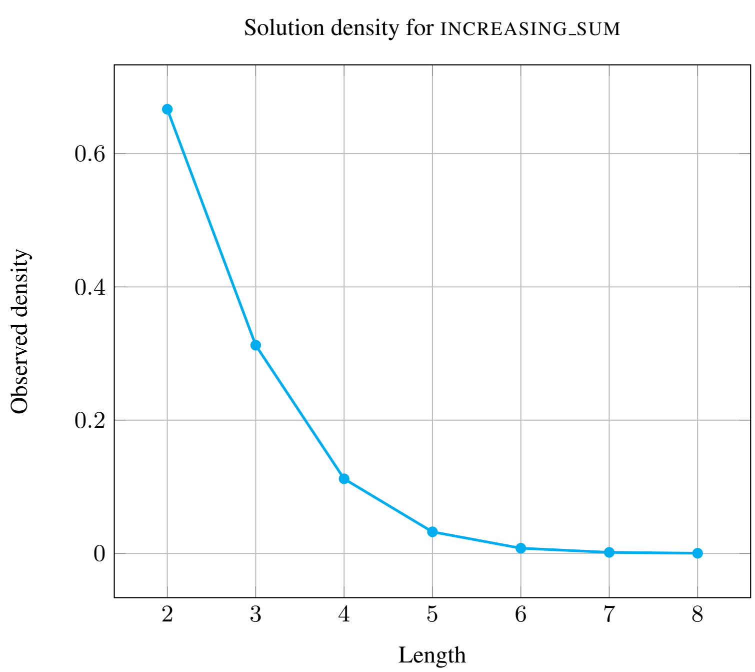
Now, assume we want to know the increase of the minimum value of s when x_0 is set to its maximum value 6. First we compute the last intersecting index of variable x_0. Since x_4 is the last variable for which the minimum value is less than or equal to maximum value of x_0 we have $last_0 = 4$. The increase is equal to $\sum_{k \in [0,4]} (\overline{x_0} - \underline{x_k}) = (6-2)+(6-4)+(6-4)+(6-5)+(6-6) = 9$. Since it exceeds the margin $29-(2+4+4+5+6+7) = 1$ we have to reduce the maximum value of x_0. How to do this incrementally is described in [324].

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	6	20	70	252	924	3432	12870

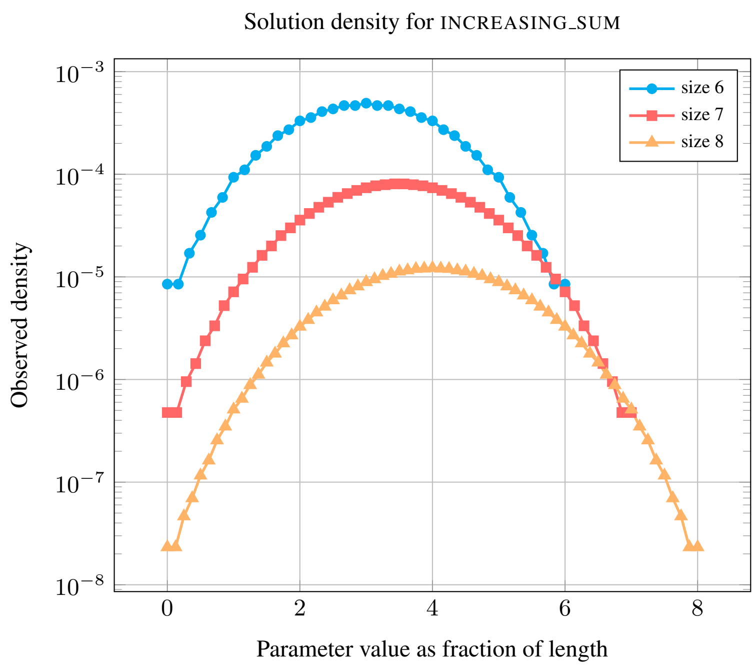
Number of solutions for INCREASING_SUM: domains $0..n$

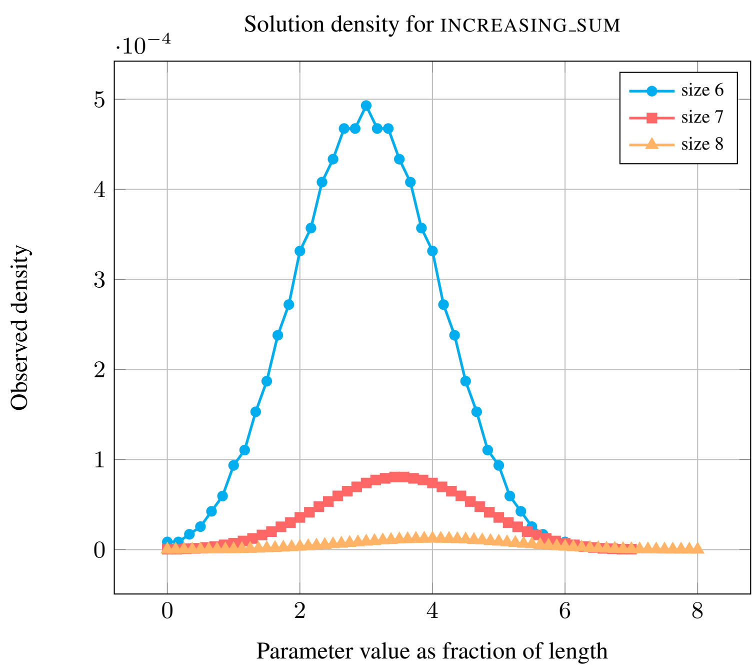

Solution density for INCREASING_SUM
Observed density
10^{0}
10^{-1}
10^{-2}
10^{-3}
2
3
4
5
6
7
8
Length


Solution density for INCREASING_SUM
Observed density
0.6
0.4
0.2
0
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		6	20	70	252	924	3432	12870
	0	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2
	3	1	3	3	3	3	3	3
	4	1	3	5	5	5	5	5
	5	-	3	5	7	7	7	7
	6	-	3	7	9	11	11	11
	7	-	2	7	11	13	15	15
	8	-	1	8	14	18	20	22
	9	-	1	7	16	22	26	28
	10	-	-	7	18	28	34	38
	11	-	-	5	19	32	42	48
	12	-	-	5	20	39	53	63
	13	-	-	3	20	42	63	77
	14	-	-	2	19	48	75	97
	15	-	-	1	18	51	87	116
	16	-	-	1	16	55	100	141
	17	-	-	-	14	55	112	164
	18	-	-	-	11	58	125	194
	19	-	-	-	9	55	136	221
	20	-	-	-	7	55	146	255
	21	-	-	-	5	51	155	284
	22	-	-	-	3	48	162	319
	23	-	-	-	2	42	166	348
	24	-	-	-	1	39	169	383
	25	-	-	-	1	32	169	409
	26	-	-	-	-	28	166	440
	27	-	-	-	-	22	162	461
	28	-	-	-	-	18	155	486
	29	-	-	-	-	13	146	499
	30	-	-	-	-	11	136	515
Parameter value	31	-	-	-	-	7	125	519
	32	-	-	-	-	5	112	526
	33	-	-	-	-	3	100	519
	34	-	-	-	-	2	87	515
	35	-	-	-	-	1	75	499
	36	-	-	-	-	1	63	486
	37	-	-	-	-	-	53	461
	38	-	-	-	-	-	42	440
	39	-	-	-	-	-	34	409
	40	-	-	-	-	-	26	383
	41	-	-	-	-	-	20	348
	42	-	-	-	-	-	15	319
	43	-	-	-	-	-	11	284
	44	-	-	-	-	-	7	255
	45	-	-	-	-	-	5	221
	46	-	-	-	-	-	3	194
	47	-	-	-	-	-	2	164
	48	-	-	-	-	-	1	141
	49	-	-	-	-	-	1	116
	50	-	-	-	-	-	-	97
	51	-	-	-	-	-	-	77
	52	-	-	-	-	-	-	63
	53	-	-	-	-	-	-	48
	54	-	-	-	-	-	-	38
	55	-	-	-	-	-	-	28
	56	-	-	-	-	-	-	22
	57	-	-	-	-	-	-	15
	58	-	-	-	-	-	-	11
	59	-	-	-	-	-	-	7
	60	-	-	-	-	-	-	5
	61	-	-	-	-	-	-	3
	62	-	-	-	-	-	-	2
	63	-	-	-	-	-	-	1
	64	-	-	-	-	-	-	1

Solution count for INCREASING_SUM: domains $0..n$





See also **common keyword:** SUM_CTR *(sum)*.

implies: INCREASING.

Keywords

characteristic of a constraint: sum.

constraint type: predefined constraint, order constraint, arithmetic constraint.

filtering: bound-consistency.

modelling: functional dependency.

symmetry: symmetry.

Cond. implications

- INCREASING_SUM(VARIABLES, S)
  with minval(VARIABLES.var) > 0
  implies ATMOST_NVALUE(S, VARIABLES).

- INCREASING_SUM(VARIABLES, S)
  with minval(VARIABLES.var) > 0
  implies SUM_OF_INCREMENTS(VARIABLES, LIMIT).

5.193 INCREASING_VALLEY

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from VALLEY and INCREASING.

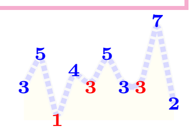
Constraint INCREASING_VALLEY(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions |VARIABLES| > 0
required(VARIABLES, var)

Purpose A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \ldots, V_m$ is a *valley* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k < V_{k+1}$.
When considering all the valleys of the sequence VARIABLES from left to right enforce all valleys to be increasing, i.e. the altitude of each valley is greater than or equal to the altitude of its preceding valley when it exists.

Example $(\langle 3, 5, 1, 4, 3, 5, 3, 3, 7, 2\rangle)$



The INCREASING_VALLEY constraint holds since the sequence 3 5 **1** 4 **3** 5 3 **3** 7 2 contains three valleys, in bold, that are increasing.

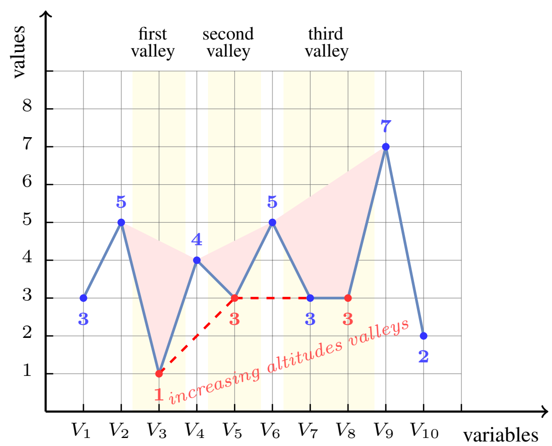


Figure 5.449: Illustration of the **Example** slot: a sequence of ten variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10} respectively fixed to values 3, 5, 1, 4, 3, 5, 3, 3, 7, 2 and its corresponding three valleys, in red, respectively located at altitudes 1, 3 and 3

Typical

$|$VARIABLES$| \geq 7$
range(VARIABLES.var) > 1
VALLEY(VARIABLES.var) ≥ 3

Typical model

nval(VARIABLES.var) > 2

Symmetry

One and the same constant can be added to the var attribute of all items of VARIABLES.

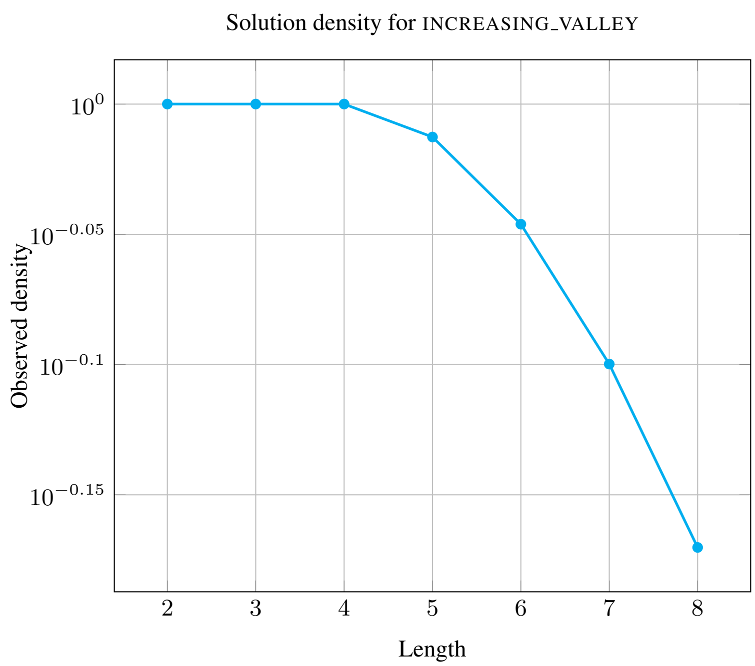
Arg. properties

- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

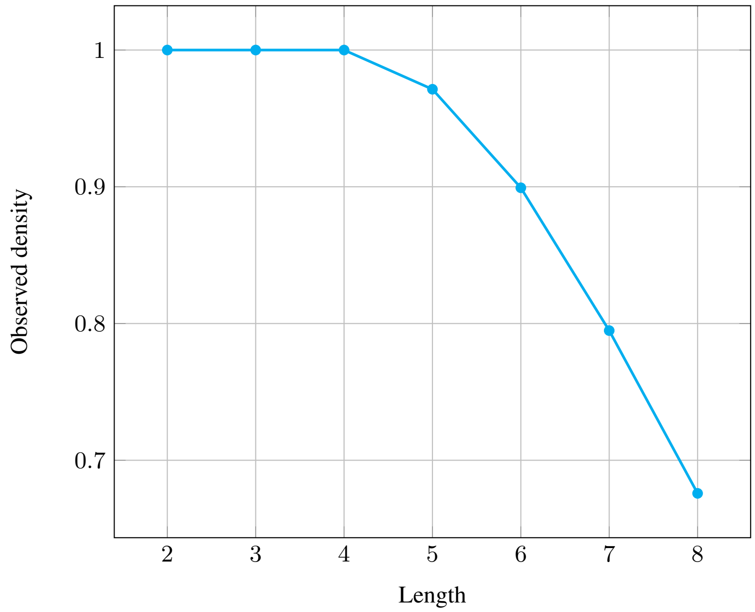
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7553	105798	1666878	29090469

Number of solutions for INCREASING_VALLEY: domains $0..n$



Solution density for INCREASING_VALLEY



See also **implied by:** ALL_EQUAL_VALLEY.

related: DECREASING_VALLEY, VALLEY.

Keywords **characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(2).

Cond. implications INCREASING_VALLEY(VARIABLES)
with VALLEY(VARIABLES.var) > 0
implies NOT_ALL_EQUAL(VARIABLES).

Automaton Figure 5.450 depicts the automaton associated with the INCREASING_VALLEY constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: initial stationary or increasing mode	$(\{= \mid <\}^*)$
u	: decreasing (before first potential valley) mode	$(> \{> \mid =\}^*)$
v	: increasing (after a valley) mode	$(< \{< \mid =\}^*)$
w	: decreasing (after a valley) mode	$(> \{> \mid =\}^*)$

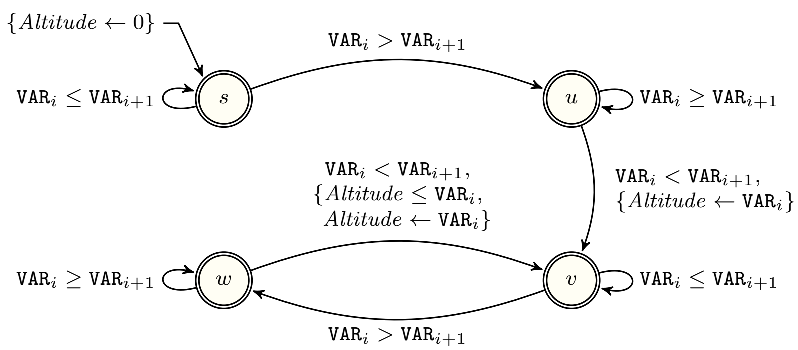


Figure 5.450: Automaton for the INCREASING_VALLEY constraint (note the conditional transition from state w to state v testing that the counter $Altitude$ is less than or equal to $\mathtt{VAR}_i$ for enforcing that all valleys from left to right are in increasing altitude)

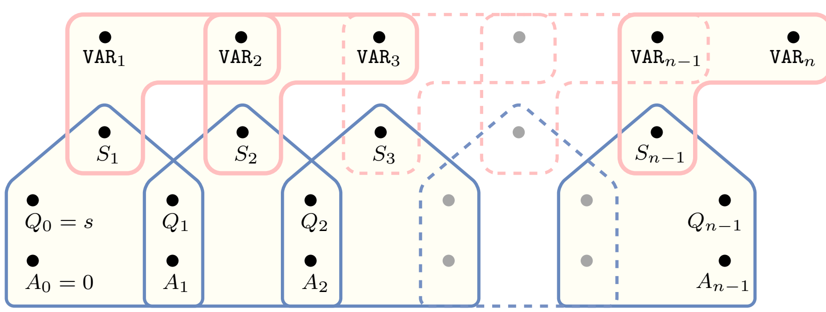


Figure 5.451: Hypergraph of the reformulation corresponding to the automaton of the INCREASING_VALLEY constraint where A_i stands for the value of the counter $Altitude$ (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.194 INDEXED_SUM

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint INDEXED_SUM(ITEMS, TABLE)

Arguments

ITEMS : collection(index−dvar, weight−dvar)
TABLE : collection(index−int, summation−dvar)

Restrictions

$|\mathtt{ITEMS}| > 0$
$|\mathtt{TABLE}| > 0$
required(ITEMS, [index, weight])
ITEMS.index ≥ 1
ITEMS.index $\leq |\mathtt{TABLE}|$
required(TABLE, [index, summation])
TABLE.index ≥ 1
TABLE.index $\leq |\mathtt{TABLE}|$
increasing_seq(TABLE, index)

Purpose

Given several items of the collection ITEMS (each of them having a specific fixed index as well as a weight that may be negative or positive), and a table TABLE (each entry of TABLE corresponding to a summation variable), assign each item to an entry of TABLE so that the sum of the weights of the items assigned to that entry is equal to the corresponding summation variable.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{ll}\mathtt{index}-3 & \mathtt{weight}-{-4},\\ \mathtt{index}-1 & \mathtt{weight}-6,\\ \mathtt{index}-3 & \mathtt{weight}-1\end{array}\right\rangle,\\ \left\langle\begin{array}{ll}\mathtt{index}-1 & \mathtt{summation}-6,\\ \mathtt{index}-2 & \mathtt{summation}-0,\\ \mathtt{index}-3 & \mathtt{summation}-{-3}\end{array}\right\rangle\end{array}\right)$$

The INDEXED_SUM constraint holds since the summation variables associated with each entry of TABLE are equal to the sum of the weights of the items assigned to the corresponding entry:

- TABLE[1].summation $=$ ITEMS[2].weight $= 6$ (since TABLE[1].index $=$ ITEMS[2].index $= 1$),
- TABLE[2].summation $= 0$ (since TABLE[2].index $= 2$ does not occur as a value of the index attribute of an item of ITEMS),
- TABLE[3].summation $=$ ITEMS[1].weight $+$ ITEMS[3].weight $= -4 + 1 = -3$ (since TABLE[3].index $=$ ITEMS[1].index $=$ ITEMS[3].index $= 3$).

Typical

$|\texttt{ITEMS}| > 1$
$\texttt{range}(\texttt{ITEMS.index}) > 1$
$|\texttt{TABLE}| > 1$
$\texttt{range}(\texttt{TABLE.summation}) > 1$

Symmetries

- Items of ITEMS are permutable.
- Items of TABLE are permutable.

Reformulation

The INDEXED_SUM(ITEMS, TABLE) constraint can be expressed in term of a set of reified constraints and of $|\texttt{TABLE}|$ arithmetic constraints (i.e., SCALAR_PRODUCT constraints).

1. For each item $\texttt{ITEMS}[i]$ ($i \in [1, |\texttt{ITEMS}|]$) and for each table entry j ($j \in [1, |\texttt{TABLE}|]$) of TABLE we create a 0-1 variable B_{ij} that will be set to 1 if and only if $\texttt{ITEMS}[i].\texttt{index}$ is fixed to j (i.e., $B_{ij} \Leftrightarrow \texttt{ITEMS}[i].\texttt{index} = j$).
2. For each entry j of the table TABLE, we impose the sum $\texttt{ITEMS}[1].\texttt{weight} \cdot B_{1j} + \texttt{ITEMS}[2].\texttt{weight} \cdot B_{2j} + \cdots + \texttt{ITEMS}[|\texttt{ITEMS}|].\texttt{weight} \cdot B_{|\texttt{ITEMS}|j}$ to be equal to $\texttt{TABLE}[j].\texttt{summation}$.

See also

implied by: ELEMENTS_ALLDIFFERENT.

specialisation: BIN_PACKING *(negative contribution not allowed, effective use variable for each bin replaced by an overall fixed capacity)*, BIN_PACKING_CAPA *(negative contribution not allowed, effective use variable for each bin replaced by a fixed capacity for each bin)*.

used in graph description: SUM_CTR.

Keywords

application area: assignment.

modelling: variable indexing, variable subscript.

For all items of TABLE:

Arc input(s)	ITEMS TABLE
Arc generator	$PRODUCT \mapsto$ collection(items, table)
Arc arity	2
Arc constraint(s)	items.index = table.index
Sets	SUCC $\mapsto$ [source, variables − col (VARIABLES−collection(var−dvar), [item(var − ITEMS.weight)])]
Constraint(s) on sets	SUM_CTR(variables, =, TABLE.summation)

Graph model We enforce the SUM_CTR constraint on the weight of the items that are assigned to the same entry. Within the context of the **Example** slot, part (A) of Figure 5.452 shows the initial graphs associated with entries 1, 2 and 3 (i.e., one initial graph for each item of the TABLE collection). Part (B) of Figure 5.452 shows the corresponding final graphs associated with entries 1 and 3. Each source vertex of the final graph can be interpreted as an item assigned to a specific entry of TABLE.

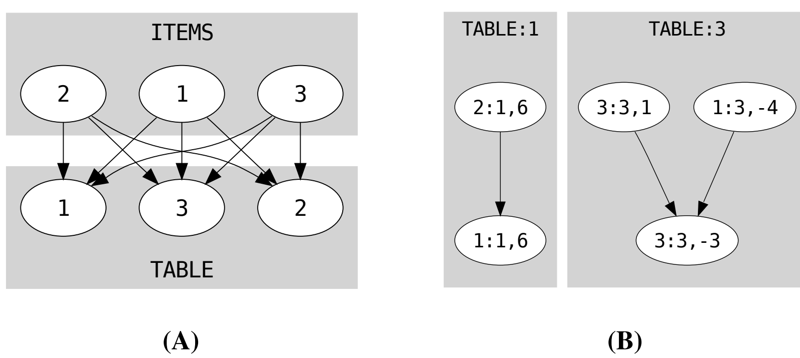


Figure 5.452: Initial and final graph of the INDEXED_SUM constraint

5.195 INFLEXION

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin N. Beldiceanu

Constraint INFLEXION(N, VARIABLES)

Arguments

N : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\text{N} \geq 0$
$\text{N} \leq \max(0, |\text{VARIABLES}| - 2)$
required(VARIABLES, var)

Purpose

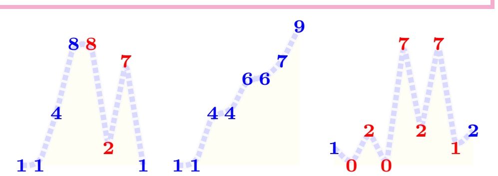
N is equal to the number of times that the following conjunctions of constraints hold:

- X_i CTR $X_{i+1} \wedge X_i \neq X_{i+1}$,
- $X_{i+1} = X_{i+2} \wedge \cdots \wedge X_{j-2} = X_{j-1}$,
- $X_{j-1} \neq X_j \wedge X_{j-1}$ ¬CTR X_j.

where X_k is the k^{th} item of the VARIABLES collection and $1 \leq i, i+2 \leq j, j \leq n$ and CTR is $<$ or $>$.

Example

(3, ⟨1, 1, 4, 8, 8, 2, 7, 1⟩)
(0, ⟨1, 1, 4, 4, 6, 6, 7, 9⟩)
(7, ⟨1, 0, 2, 0, 7, 2, 7, 1, 2⟩)



The first INFLEXION constraint holds since the sequence 1 1 4 8 8 2 7 1 contains three inflexions peaks that respectively correspond to values 8, 2 and 7.

All solutions

Figure 5.454 gives all solutions to the following non ground instance of the INFLEXION constraint: $\text{N} \in \{0, 2\}$, $\text{V}_1 = 2$, $\text{V}_2 \in [2, 3]$, $\text{V}_3 \in [1, 2]$, $\text{V}_4 \in [1, 2]$, $\text{V}_5 = 3$, INFLEXION(N, ⟨$\text{V}_1, \text{V}_2, \text{V}_3, \text{V}_4, \text{V}_5$⟩).

Typical

$\text{N} > 0$
$|\text{VARIABLES}| > 2$
range(VARIABLES.var) > 1

Typical model

nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Functional dependency: N determined by VARIABLES.

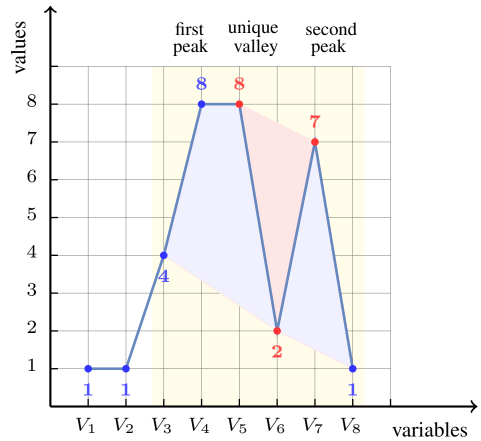


Figure 5.453: Illustration of the first example of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values $1, 1, 4, 8, 8, 2, 7, 1$ and its three inflexions in red, two peaks and one valley ($\mathtt{N} = 3$)

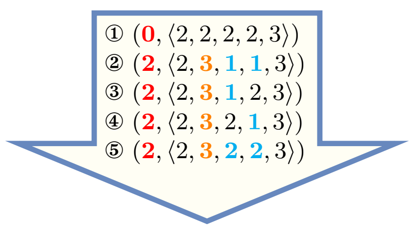


Figure 5.454: All solutions corresponding to the non ground example of the INFLEXION constraint of the **All solutions** slot where each inflexion (i.e. peak or valley) is coloured in orange or cyan

Usage

Useful for constraining the number of *inflexions* of a sequence of domain variables.

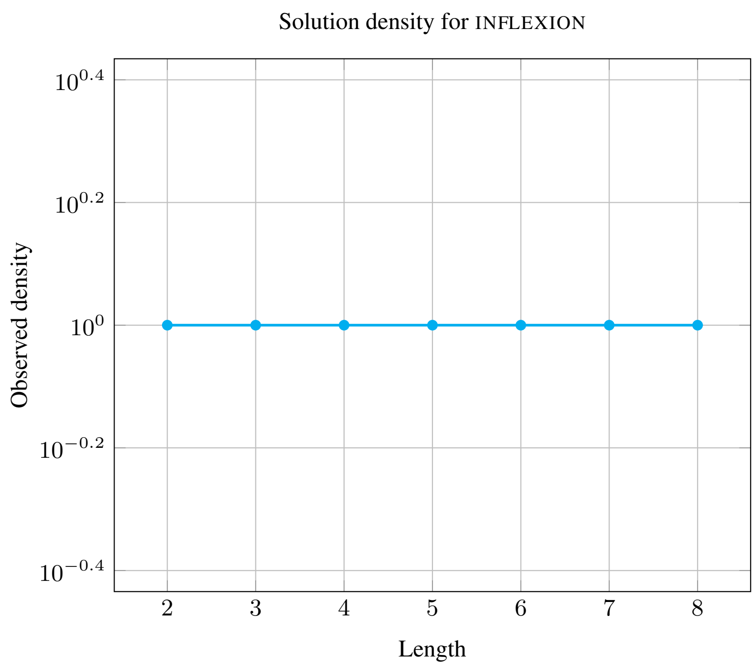
Remark

Since the arity of the arc constraint is not fixed, the INFLEXION constraint cannot be currently described with the graph-based representation. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.

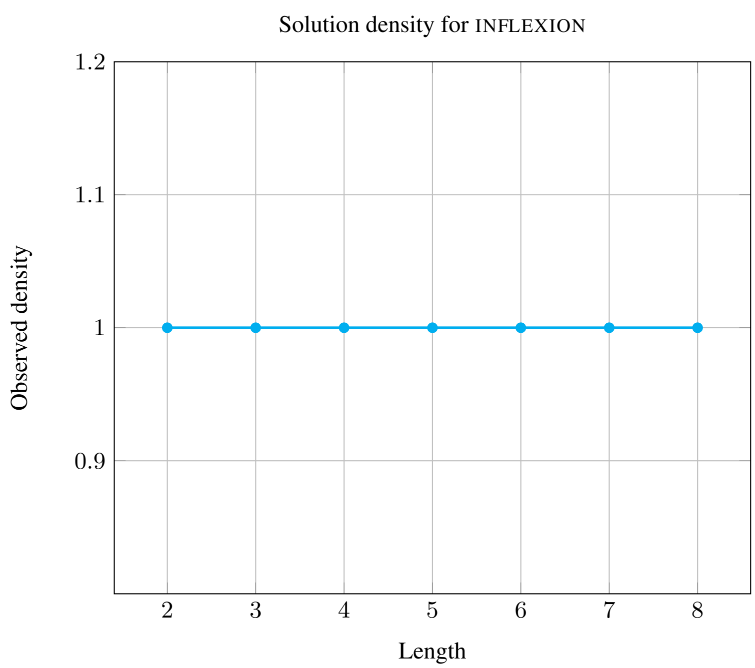
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for INFLEXION: domains $0..n$



Solution density for INFLEXION
Observed density
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
2
3
4
5
6
7
8
Length

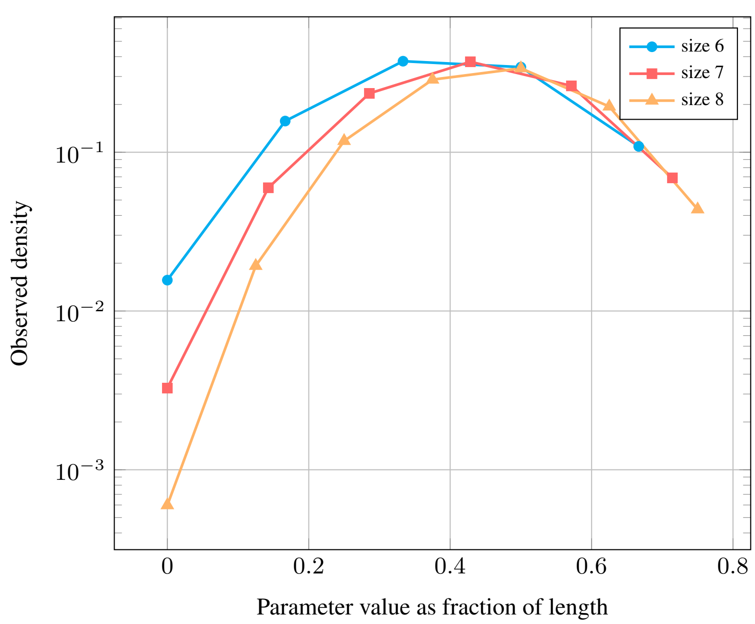


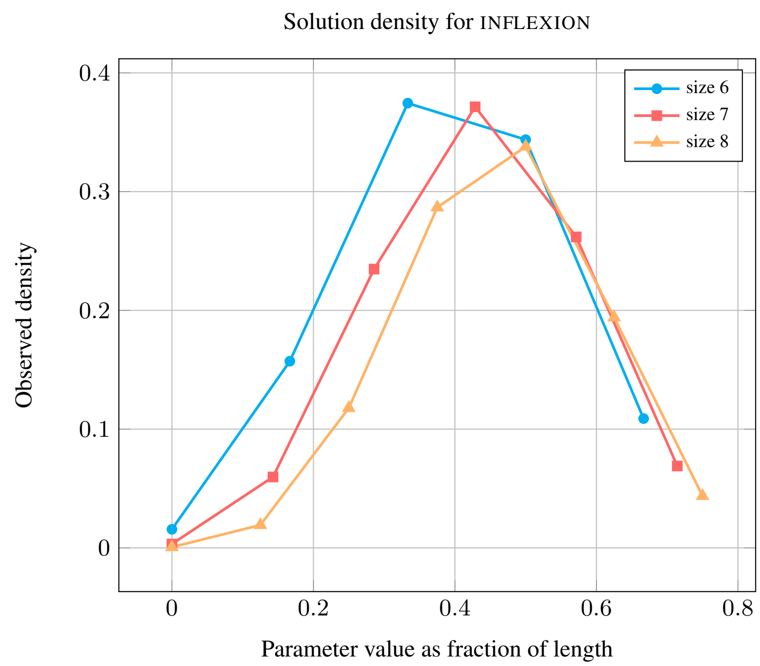
Solution density for INFLEXION
Observed density
1.2
1.1
1
0.9
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	9	36	135	498	1841	6856	25731
	1	-	28	320	2588	18494	125284	828120
	2	-	-	170	3348	44058	492320	5069970
	3	-	-	-	1342	40446	778936	12341184
	4	-	-	-	-	12810	549152	14547186
	5	-	-	-	-	-	144604	8354520
	6	-	-	-	-	-	-	1880010

Solution count for INFLEXION: domains $0..n$

Solution density for INFLEXION





See also **common keyword:** GLOBAL_CONTIGUITY, MIN_DIST_BETWEEN_INFLEXION, PEAK, VALLEY *(sequence)*.

Keywords **characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

filtering: glue matrix.

modelling: functional dependency.

Cond. implications

- INFLEXION(N, VARIABLES)
  with $\texttt{N} > 0$
  implies ATLEAST_NVALUE(NVAL, VARIABLES)
  when $\texttt{NVAL} = 2$.

- INFLEXION(N, VARIABLES)
  with VALLEY(VARIABLES.var) $= 0$
  implies PEAK(N, VARIABLES).

- INFLEXION(N, VARIABLES)
  with PEAK(VARIABLES.var) $= 0$
  implies VALLEY(N, VARIABLES).

Automaton Figure 5.455 depicts the automaton associated with the INFLEXION constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

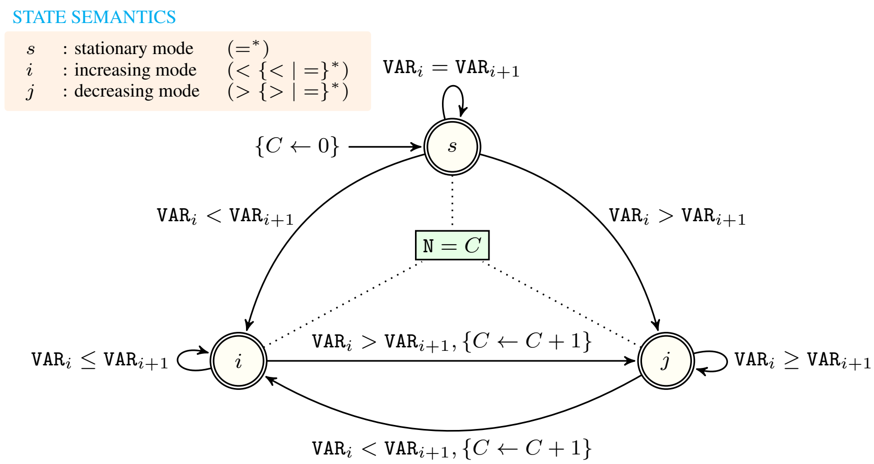


Figure 5.455: Automaton of the INFLEXION constraint (state s means that we are in *stationary* mode, state i means that we are in *increasing* mode, state j means that we are in *decreasing* mode, a new inflexion is detected each time we switch from increasing to decreasing mode – or conversely from decreasing to increasing mode – and the counter C is incremented accordingly)

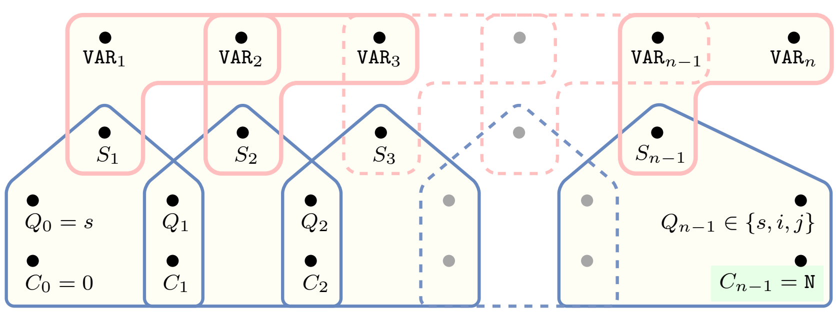


Figure 5.456: Hypergraph of the reformulation corresponding to the automaton of the INFLEXION constraint

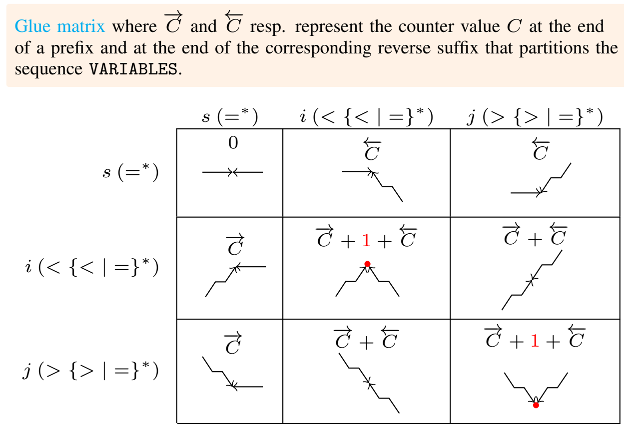


Figure 5.457: Glue matrix associated with the automaton of the INFLEXION constraint

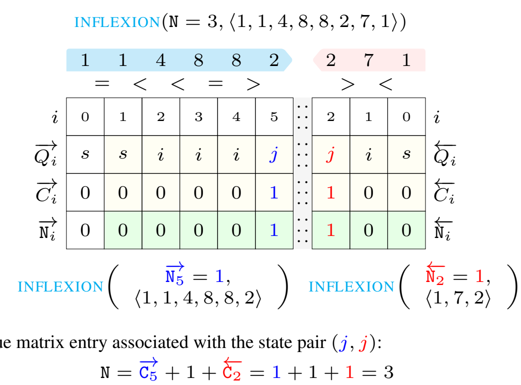


Figure 5.458: Illustrating the use of the state pair (j, j) of the glue matrix for linking N with the counters variables obtained after reading the prefix $1, 1, 4, 8, 8, 2$ and corresponding suffix $2, 7, 1$ of the sequence $1, 1, 4, 8, 8, 2, 7, 1$; note that the suffix $2, 7, 1$ (in pink) is proceed in reverse order; the left (resp. right) table shows the initialisation (for $i = 0$) and the evolution (for $i > 0$) of the state of the automaton and its counter C upon reading the prefix $1, 1, 4, 8, 8, 2$ (resp. the reverse suffix $1, 7, 2$).

5.196 INSIDE_SBOXES

DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [349]

Constraint INSIDE_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonym INSIDE.

Types

VARIABLES	:	collection(v−dvar)
INTEGERS	:	collection(v−int)
POSITIVES	:	collection(v−int)

Arguments

K	:	int
DIMS	:	sint
OBJECTS	:	collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES	:	collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

$|\mathtt{VARIABLES}| \geq 1$
$|\mathtt{INTEGERS}| \geq 1$
$|\mathtt{POSITIVES}| \geq 1$
required(VARIABLES, v)
$|\mathtt{VARIABLES}| = \mathtt{K}$
required(INTEGERS, v)
$|\mathtt{INTEGERS}| = \mathtt{K}$
required(POSITIVES, v)
$|\mathtt{POSITIVES}| = \mathtt{K}$
$\mathtt{POSITIVES.v} > 0$
$\mathtt{K} > 0$
$\mathtt{DIMS} \geq 0$
$\mathtt{DIMS} < \mathtt{K}$
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
$\mathtt{OBJECTS.oid} \geq 1$
$\mathtt{OBJECTS.oid} \leq |\mathtt{OBJECTS}|$
$\mathtt{OBJECTS.sid} \geq 1$
$\mathtt{OBJECTS.sid} \leq |\mathtt{SBOXES}|$
$|\mathtt{SBOXES}| \geq 1$
required(SBOXES, [sid, t, l])
$\mathtt{SBOXES.sid} \geq 1$
$\mathtt{SBOXES.sid} \leq |\mathtt{SBOXES}|$
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i < j$, O_i is inside O_j with respect to a set of dimensions depicted by `DIMS`. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a `K`-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id `sid`, shift offset `t`, and sizes `l`. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier `oid`, shape id `sid` and origin `x`.

An object O_i *is inside* an object O_j with respect to a set of dimensions depicted by `DIMS` if and only if, for all shifted boxes s_i associated with O_i, there exists a shifted box s_j of O_j such that s_j is inside s_i. A shifted box s_j *is inside* a shifted box s_i if and only if, for all dimensions $d \in$ `DIMS`, (1) the start of s_j in dimension d is strictly less than the start of s_i in dimension d, and (2) the end of s_i in dimension d is strictly less than the end of s_j in dimension d.

Example

$$\left(\begin{array}{l} 2, \{0,1\}, \\ \left\langle\begin{array}{lll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 3,3\rangle, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 2,2\rangle, \\ \texttt{oid}-3 & \texttt{sid}-3 & \texttt{x}-\langle 1,1\rangle \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,3\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 5,5\rangle \end{array}\right\rangle \end{array}\right)$$

Figure 5.459 shows the objects of the example. Since O_1 is inside O_2 and O_3, and since O_2 is also inside O_3, the INSIDE_SBOXES constraint holds.

Typical

$|\texttt{OBJECTS}| > 1$

Symmetries

- Items of `SBOXES` are permutable.
- Items of `OBJECTS.x`, `SBOXES.t` and `SBOXES.l` are permutable (*same permutation used*).

Arg. properties

Suffix-contractible wrt. `OBJECTS`.

Remark

One of the eight relations of the *Region Connection Calculus* [349]. The constraint INSIDE_SBOXES is a restriction of the original relation since it requires that each box of an object is contained by one box of the other object.

See also

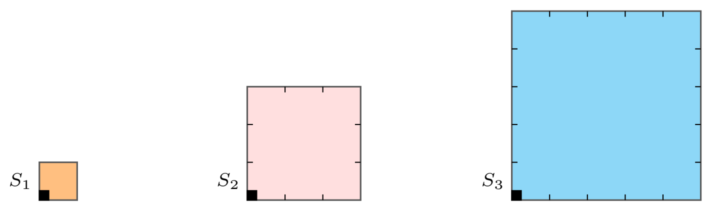
common keyword: CONTAINS_SBOXES, COVEREDBY_SBOXES, COVERS_SBOXES, DISJOINT_SBOXES, EQUAL_SBOXES, MEET_SBOXES *(rcc8)*, NON_OVERLAP_SBOXES *(geometrical constraint,logic)*, OVERLAP_SBOXES *(rcc8)*.

Keywords

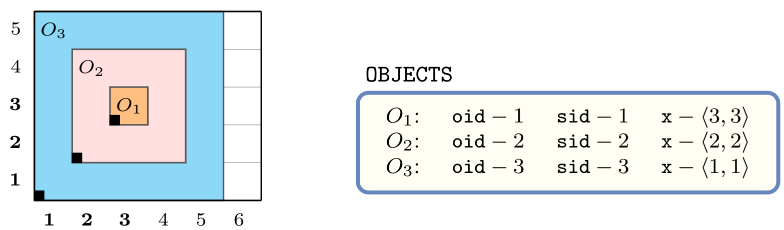
constraint type: logic.

geometry: geometrical constraint, rcc8.

miscellaneous: obscure.



(A) Shape of the first object (B) Shape of the second object (C) Shape of the third object



(D) Three objects O_1, O_2, O_3, where O_1 is inside O_2 and O_3, and where O_2 is inside O_3

Figure 5.459: (D) the three nested objects O_3, O_2, O_1 of the **Example** slot respectively assigned shapes S_3, S_2, S_1; (A), (B), (C) shapes S_1, S_2 and S_3 are made up from a single shifted box.

Logic

- $\mathtt{origin}(\mathtt{O1},\mathtt{S1},\mathtt{D}) \stackrel{\text{def}}{=} \mathtt{O1.x}(\mathtt{D}) + \mathtt{S1.t}(\mathtt{D})$
- $\mathtt{end}(\mathtt{O1},\mathtt{S1},\mathtt{D}) \stackrel{\text{def}}{=} \mathtt{O1.x}(\mathtt{D}) + \mathtt{S1.t}(\mathtt{D}) + \mathtt{S1.l}(\mathtt{D})$
- $\mathtt{inside\_sboxes}(\mathtt{Dims},\mathtt{O1},\mathtt{S1},\mathtt{O2},\mathtt{S2}) \stackrel{\text{def}}{=}$
  $$\forall \mathtt{D} \in \mathtt{Dims}$$
  $$\bigwedge \left(\begin{array}{l} \mathtt{origin}(\mathtt{O2},\mathtt{S2},\mathtt{D}) < \\ \mathtt{origin}(\mathtt{O1},\mathtt{S1},\mathtt{D}) \\ \mathtt{end}(\mathtt{O1},\mathtt{S1},\mathtt{D}) < \\ \mathtt{end}(\mathtt{O2},\mathtt{S2},\mathtt{D}) \end{array} , \right)$$
- $\mathtt{inside\_objects}(\mathtt{Dims},\mathtt{O1},\mathtt{O2}) \stackrel{\text{def}}{=}$
  $$\forall \mathtt{S1} \in \mathtt{sboxes}([\mathtt{O1.sid}])$$
  $$\exists \mathtt{S2} \in \mathtt{sboxes}\left(\left[\ \mathtt{O2.sid} \ \right] \right)$$
  $$\mathtt{inside\_sboxes} \left(\begin{array}{l} \mathtt{Dims}, \\ \mathtt{O1}, \\ \mathtt{S1}, \\ \mathtt{O2}, \\ \mathtt{S2} \end{array} \right)$$
- $\mathtt{all\_inside}(\mathtt{Dims},\mathtt{OIDS}) \stackrel{\text{def}}{=}$
  $$\forall \mathtt{O1} \in \mathtt{objects}(\mathtt{OIDS})$$
  $$\forall \mathtt{O2} \in \mathtt{objects}(\mathtt{OIDS})$$
  $$\begin{array}{l} \mathtt{O1.oid} < \\ \mathtt{O2.oid} \end{array} \Rightarrow$$
  $$\mathtt{inside\_objects} \left(\begin{array}{l} \mathtt{Dims}, \\ \mathtt{O1}, \\ \mathtt{O2} \end{array} \right)$$
- $\mathtt{all\_inside}(\mathtt{DIMENSIONS},\mathtt{OIDS})$

5.197 INT_VALUE_PRECEDE

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin [269]

Constraint INT_VALUE_PRECEDE(S, T, VARIABLES)

Synonyms PRECEDE, PRECEDENCE, VALUE_PRECEDE.

Arguments

S	:	int
T	:	int
VARIABLES	:	collection(var−dvar)

Restrictions

$\mathtt{S} \neq \mathtt{T}$
required(VARIABLES, var)

Purpose If value T occurs in the collection of variables VARIABLES then its first occurrence should be preceded by an occurrence of value S.

Example $(0, 1, \langle 4, 0, 6, 1, 0\rangle)$

The INT_VALUE_PRECEDE constraint holds since the first occurrence of value 0 precedes the first occurrence of value 1.

Typical

$\mathtt{S} < \mathtt{T}$
$|\mathtt{VARIABLES}| > 1$
ATLEAST(1, VARIABLES, S)
ATLEAST(1, VARIABLES, T)

Symmetries

- An occurrence of a value of VARIABLES.var that is different from S and T can be replaced by any other value that is also different from S and T.
- All occurrences of values S and T can be swapped in S, T and VARIABLES.var.

Arg. properties

- Suffix-contractible wrt. VARIABLES.
- Aggregate: S(id), T(id), VARIABLES(union).

Algorithm A filtering algorithm for maintaining value precedence is presented in [269]. Its complexity is linear to the number of variables of the collection VARIABLES.

Systems PRECEDE in **Gecode**, VALUE_PRECEDE in **MiniZinc**.

See also **generalisation:** INT_VALUE_PRECEDE_CHAIN *(*sequence *of* 2 values *replaced by* sequence *of at least* 2 values*)*, SET_VALUE_PRECEDE *(*sequence *of* domain variables *replaced by* sequence *of* set variables*)*.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: arc-consistency.

symmetry: symmetry, indistinguishable values, value precedence.

Automaton Figure 5.460 depicts the automaton associated with the INT_VALUE_PRECEDE constraint. Let $\mathtt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each triple $(\mathtt{S}, \mathtt{T}, \mathtt{VAR}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\mathtt{VAR}_i = \mathtt{S} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i = \mathtt{T} \Leftrightarrow S_i = 2) \wedge (\mathtt{VAR}_i \neq \mathtt{S} \wedge \mathtt{VAR}_i \neq \mathtt{T} \Leftrightarrow S_i = 3)$.

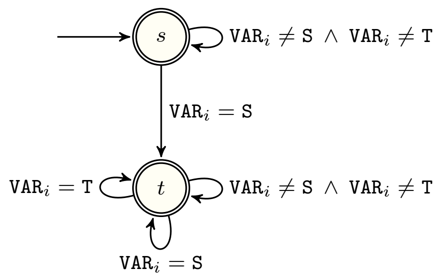


Figure 5.460: Automaton of the INT_VALUE_PRECEDE constraint (state s means that value S was not yet encountered, while state t means that value S was already encountered)

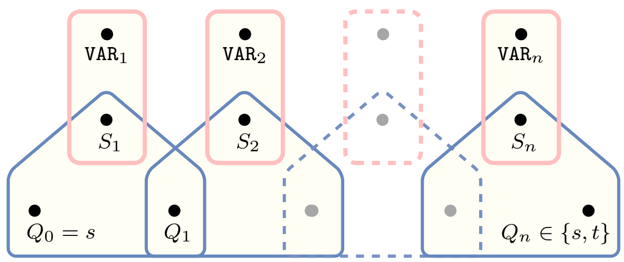


Figure 5.461: Hypergraph of the reformulation corresponding to the automaton of the INT_VALUE_PRECEDE constraint

5.198 INT_VALUE_PRECEDE_CHAIN

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin [269]

Constraint INT_VALUE_PRECEDE_CHAIN(VALUES, VARIABLES)

Synonyms PRECEDE, PRECEDENCE, VALUE_PRECEDE_CHAIN.

Arguments

VALUES	:	collection(var−int)
VARIABLES	:	collection(var−dvar)

Restrictions

required(VALUES, var)
distinct(VALUES, var)
required(VARIABLES, var)

Purpose Assuming n denotes the number of items of the VALUES collection, the following condition holds for every $i \in [1, n-1]$: When it is defined, the first occurrence of the $(i+1)^{th}$ value of the VALUES collection should be preceded by the first occurrence of the i^{th} value of the VALUES collection.

Example $(\langle 4,0,1\rangle, \langle 4,0,6,1,0\rangle)$

The INT_VALUE_PRECEDE_CHAIN constraint holds since within the sequence 4, 0, 6, 1, 0:

- The first occurrence of value 4 occurs before the first occurrence of value 0.
- The first occurrence of value 0 occurs before the first occurrence of value 1.

Typical

$|\mathtt{VALUES}| > 1$
STRICTLY_INCREASING(VALUES)
$|\mathtt{VARIABLES}| > |\mathtt{VALUES}|$
range(VARIABLES.var) > 1
USED_BY(VARIABLES, VALUES)

Symmetry An occurrence of a value of VARIABLES.var that does not occur in VALUES.var can be replaced by any other value that also does not occur in VALUES.var.

Arg. properties

- Contractible wrt. VALUES.
- Suffix-contractible wrt. VARIABLES.
- Aggregate: VALUES(id), VARIABLES(union).

Usage The INT_VALUE_PRECEDE_CHAIN constraint is useful for breaking symmetries in graph colouring problems. We set a INT_VALUE_PRECEDE_CHAIN constraint on all variables $V_1, V_2, \ldots, V_n$ associated with the vertices of the graph to colour, where we state that the first occurrence of colour i should be located before the first occurrence of colour $i+1$ within the sequence $V_1, V_2, \ldots, V_n$.

Figure 5.462 illustrates the problem of *colouring earth and mars* from Thom Sulanke. Part (A) of Figure 5.462 provides a solution where the first occurrence of each value of i, ($i \in \{1, 2, \ldots, 8\}$) is located before the first occurrence of value $i+1$. This is obtained by using the following constraints:

$$\left\{\begin{array}{l}
\mathtt{A} \neq \mathtt{B}, \mathtt{A} \neq \mathtt{E}, \mathtt{A} \neq \mathtt{F}, \mathtt{A} \neq \mathtt{G}, \mathtt{A} \neq \mathtt{H}, \mathtt{A} \neq \mathtt{I}, \mathtt{A} \neq \mathtt{J}, \mathtt{A} \neq \mathtt{K},\\
\mathtt{B} \neq \mathtt{A}, \mathtt{B} \neq \mathtt{C}, \mathtt{B} \neq \mathtt{F}, \mathtt{B} \neq \mathtt{G}, \mathtt{B} \neq \mathtt{H}, \mathtt{B} \neq \mathtt{I}, \mathtt{B} \neq \mathtt{J}, \mathtt{B} \neq \mathtt{K},\\
\mathtt{C} \neq \mathtt{B}, \mathtt{C} \neq \mathtt{D}, \mathtt{C} \neq \mathtt{F}, \mathtt{C} \neq \mathtt{G}, \mathtt{C} \neq \mathtt{H}, \mathtt{C} \neq \mathtt{I}, \mathtt{C} \neq \mathtt{J}, \mathtt{C} \neq \mathtt{K},\\
\mathtt{D} \neq \mathtt{C}, \mathtt{D} \neq \mathtt{E}, \mathtt{D} \neq \mathtt{F}, \mathtt{D} \neq \mathtt{G}, \mathtt{D} \neq \mathtt{H}, \mathtt{D} \neq \mathtt{I}, \mathtt{D} \neq \mathtt{J}, \mathtt{D} \neq \mathtt{K},\\
\mathtt{E} \neq \mathtt{A}, \mathtt{E} \neq \mathtt{D}, \mathtt{E} \neq \mathtt{F}, \mathtt{E} \neq \mathtt{G}, \mathtt{E} \neq \mathtt{H}, \mathtt{E} \neq \mathtt{I}, \mathtt{E} \neq \mathtt{J}, \mathtt{E} \neq \mathtt{K},\\
\mathtt{F} \neq \mathtt{A}, \mathtt{F} \neq \mathtt{B}, \mathtt{F} \neq \mathtt{C}, \mathtt{F} \neq \mathtt{D}, \mathtt{F} \neq \mathtt{E}, \mathtt{F} \neq \mathtt{G}, \mathtt{F} \neq \mathtt{H}, \mathtt{F} \neq \mathtt{I}, \mathtt{F} \neq \mathtt{J}, \mathtt{F} \neq \mathtt{K},\\
\mathtt{G} \neq \mathtt{A}, \mathtt{G} \neq \mathtt{B}, \mathtt{G} \neq \mathtt{C}, \mathtt{G} \neq \mathtt{D}, \mathtt{G} \neq \mathtt{E}, \mathtt{G} \neq \mathtt{F}, \mathtt{G} \neq \mathtt{H}, \mathtt{G} \neq \mathtt{I}, \mathtt{G} \neq \mathtt{J}, \mathtt{G} \neq \mathtt{K},\\
\mathtt{H} \neq \mathtt{A}, \mathtt{H} \neq \mathtt{B}, \mathtt{H} \neq \mathtt{C}, \mathtt{H} \neq \mathtt{D}, \mathtt{H} \neq \mathtt{E}, \mathtt{H} \neq \mathtt{F}, \mathtt{H} \neq \mathtt{G}, \mathtt{H} \neq \mathtt{I}, \mathtt{H} \neq \mathtt{J}, \mathtt{H} \neq \mathtt{K},\\
\mathtt{I} \neq \mathtt{A}, \mathtt{I} \neq \mathtt{B}, \mathtt{I} \neq \mathtt{C}, \mathtt{I} \neq \mathtt{D}, \mathtt{I} \neq \mathtt{E}, \mathtt{I} \neq \mathtt{F}, \mathtt{I} \neq \mathtt{G}, \mathtt{I} \neq \mathtt{H}, \mathtt{I} \neq \mathtt{J}, \mathtt{I} \neq \mathtt{K},\\
\mathtt{J} \neq \mathtt{A}, \mathtt{J} \neq \mathtt{B}, \mathtt{J} \neq \mathtt{C}, \mathtt{J} \neq \mathtt{D}, \mathtt{J} \neq \mathtt{E}, \mathtt{J} \neq \mathtt{F}, \mathtt{J} \neq \mathtt{G}, \mathtt{J} \neq \mathtt{H}, \mathtt{J} \neq \mathtt{I}, \mathtt{J} \neq \mathtt{K},\\
\mathtt{K} \neq \mathtt{A}, \mathtt{K} \neq \mathtt{B}, \mathtt{K} \neq \mathtt{C}, \mathtt{K} \neq \mathtt{D}, \mathtt{K} \neq \mathtt{E}, \mathtt{K} \neq \mathtt{F}, \mathtt{K} \neq \mathtt{G}, \mathtt{K} \neq \mathtt{H}, \mathtt{K} \neq \mathtt{I}, \mathtt{K} \neq \mathtt{J},\\
\text{INT_VALUE_PRECEDE_CHAIN}(\langle 1,2,3,4,5,6,7,8,9\rangle, \langle \mathtt{A},\mathtt{B},\mathtt{C},\mathtt{D},\mathtt{E},\mathtt{F},\mathtt{G},\mathtt{H},\mathtt{I},\mathtt{J},\mathtt{K}\rangle).
\end{array}\right.$$

Part (B) provides a symmetric solution where the value precedence constraints between the pairs of values $(1, 2)$, $(2, 3)$, $(4, 5)$, $(7, 8)$ and $(8, 9)$ are all violated (each violation is depicted by a dashed arc).

For one other example of use of the INT_VALUE_PRECEDE_CHAIN constraint in the context of bin packing problems see the exercise called switching time and resource and breaking symmetry of the CUMULATIVE constraint.

Remark When we have more than one class of interchangeable values (i.e., a partition of interchangeable values) we can use one INT_VALUE_PRECEDE_CHAIN constraint for breaking value symmetry in each class of interchangeable values. However it was shown in [450] that enforcing arc-consistency for such a conjunction of INT_VALUE_PRECEDE_CHAIN constraints is NP-hard.

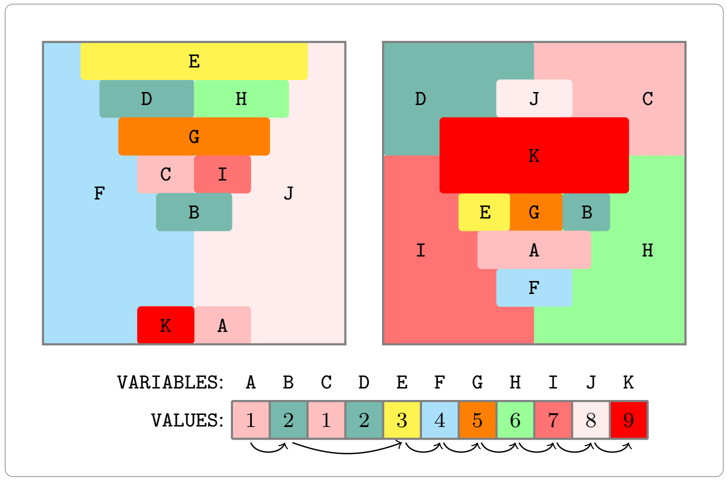
Algorithm The 2004 reformulation [30] associated with the automaton of the **Automaton** slot achieves arc-consistency since the corresponding constraint network is a Berge-acyclic constraint network. Later on, another formulation into a sequence of ternary sliding constraints was proposed by [449]. It also achieves arc-consistency for the same reason.

Systems PRECEDE in **Gecode**, VALUE_PRECEDE_CHAIN in **MiniZinc**.

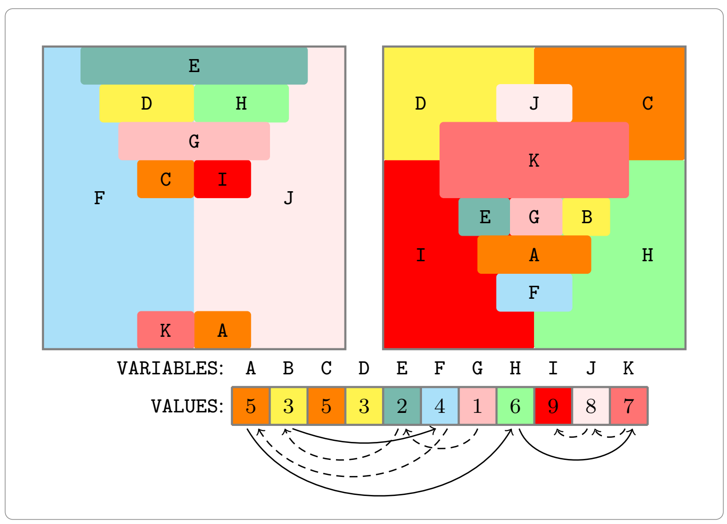
See also **specialisation:** INT_VALUE_PRECEDE *(*`sequence` *of at least* 2 `values` *replaced by* `sequence` *of* 2 `values`*)*.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.



(A) Chain of satisfied value precedences between first occurrence of consecutive values



(B) Broken value precedences between first occurrence of consecutive values (in dashed)

Figure 5.462: Using the INT_VALUE_PRECEDE_CHAIN constraint for breaking symmetries in graph colouring problems; there is an arc between the first occurrence of value v $(1 \leq v \leq 8)$ in the sequence of variables A, B, C, D, E, F, G, H, I, J, K, and the first occurrence of value $v+1$ (a plain arc if the corresponding value precedence constraint holds, a dashed arc otherwise)

constraint type: order constraint.

filtering: arc-consistency.

problems: graph colouring.

symmetry: symmetry, indistinguishable values, value precedence.

Automaton Figure 5.463 depicts the automaton associated with the INT_VALUE_PRECEDE_CHAIN constraint. Let n and m respectively denote the number of variables of the VARIABLES collection and the number of values of the VALUES collection. Let $\mathtt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. Let $\mathtt{val}_v$ $(1 \leq v \leq m)$ denote the v^{th} value of the VALUES collection.

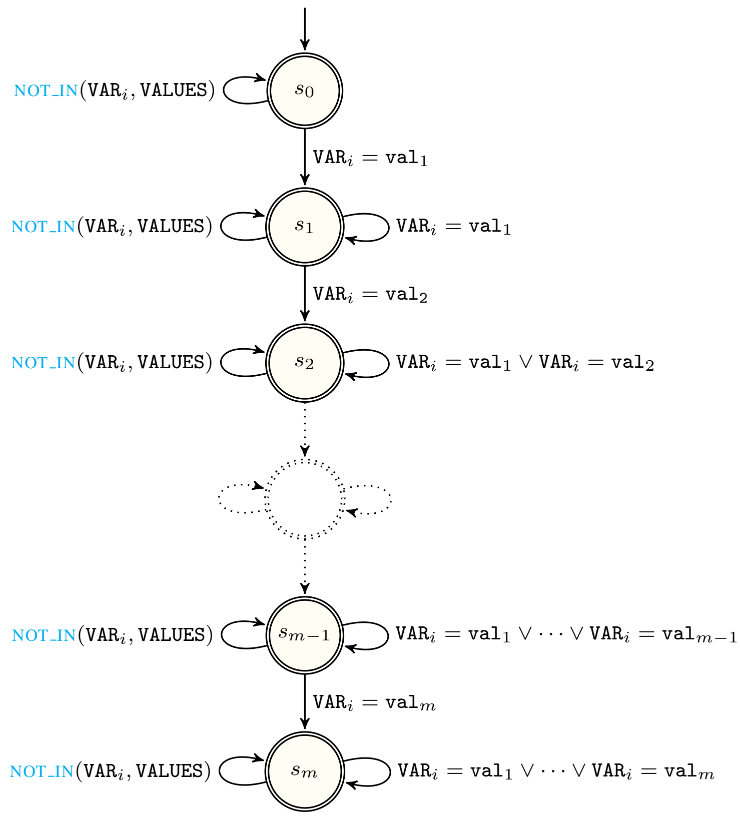


Figure 5.463: Automaton of the INT_VALUE_PRECEDE_CHAIN constraint (state s_i means that (1) each value $\mathtt{val}_1, \mathtt{val}_2, \ldots, \mathtt{val}_i$ was already encountered at least once, and that (2) value $\mathtt{val}_{i+1}$ was not yet found)

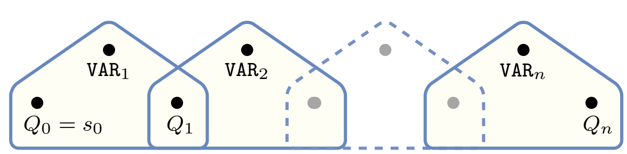


Figure 5.464: Hypergraph of the reformulation corresponding to the automaton of the INT_VALUE_PRECEDE_CHAIN constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

We now show how to construct such an automaton systematically. For this purpose let us first introduce some notations:

- Without loss of generality we assume that we have at least two values (i.e., $m \geq 2$).
- Let $\mathcal{C}$ be the set of values that can be potentially assigned to a variable of the `VARIABLES` collection, but which do not belong to the values of the `VALUES` collection (i.e., $\mathcal{C} = (dom(\mathtt{VAR}_1) \cup dom(\mathtt{VAR}_2) \cup \cdots \cup dom(\mathtt{VAR}_n) - \{\mathtt{val}_1, \mathtt{val}_2, \ldots, \mathtt{val}_m\} = \{w_1, w_2, \ldots, w_{|\mathcal{C}|}\}$.

The states and transitions of the automaton are respectively defined in the following way:

- We have $m+1$ states labelled $s_0, s_1, \ldots, s_m$ from which s_0 is the initial state. All states are accepting states.
- We have the following three sets of transitions:
  1. For all $v \in [0, m-1]$, a transition from s_v to s_{v+1} labelled by value $\mathtt{val}_{v+1}$. Each transition of this type will be triggered on the first occurrence of value $\mathtt{val}_{v+1}$ within the variables of the `VARIABLES` collection.
  2. For all $v \in [1, m]$ and for all $w \in [1, v]$, a self loop on s_v labelled by value $\mathtt{val}_w$. Such transitions encode the fact that we stay in the same state as long as we have a value that was already encountered.
  3. If the set $\mathcal{C}$ is not empty, then for all $v \in [0, m]$ a self loop on s_v labelled by the fact that we take a value not in `VALUES` (i.e., a value in $\mathcal{C}$). This models the fact that, encountering a value that does not belong to the set of values of the `VALUES` collection, leaves us in the same state.

5.199 INTERVAL_AND_COUNT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [135]

Constraint INTERVAL_AND_COUNT(ATMOST, COLOURS, TASKS, SIZE_INTERVAL)

Arguments

ATMOST	:	int
COLOURS	:	collection(val−int)
TASKS	:	collection(origin−dvar, colour−dvar)
SIZE_INTERVAL	:	int

Restrictions

ATMOST ≥ 0
required(COLOURS, val)
distinct(COLOURS, val)
required(TASKS, [origin, colour])
TASKS.origin ≥ 0
SIZE_INTERVAL > 0

Purpose

First consider the set of tasks of the TASKS collection, where each task has a specific colour that may not be initially fixed. Then consider the intervals of the form $[k \cdot \texttt{SIZE\_INTERVAL}, k \cdot \texttt{SIZE\_INTERVAL} + \texttt{SIZE\_INTERVAL} - 1]$, where k is an integer. The INTERVAL_AND_COUNT constraint forces that, for each interval I_k previously defined, the total number of tasks, which both are assigned to I_k and take their colours in COLOURS, does not exceed the limit ATMOST.

Example

$$\left(\begin{array}{l} 2, \langle 4 \rangle, \\ \left\langle \begin{array}{ll} \texttt{origin}-1 & \texttt{colour}-4, \\ \texttt{origin}-0 & \texttt{colour}-9, \\ \texttt{origin}-10 & \texttt{colour}-4, \\ \texttt{origin}-4 & \texttt{colour}-4 \end{array} \right\rangle, 5 \end{array}\right)$$

Figure 5.465 shows the solution associated with the example. The constraint INTERVAL_AND_COUNT holds since, for each interval, the number of tasks taking colour 4 does not exceed the limit 2.

Typical

ATMOST > 0
ATMOST $<$ |TASKS|
|COLOURS| > 0
|TASKS| > 1
range(TASKS.origin) > 1
range(TASKS.colour) > 1
SIZE_INTERVAL > 1

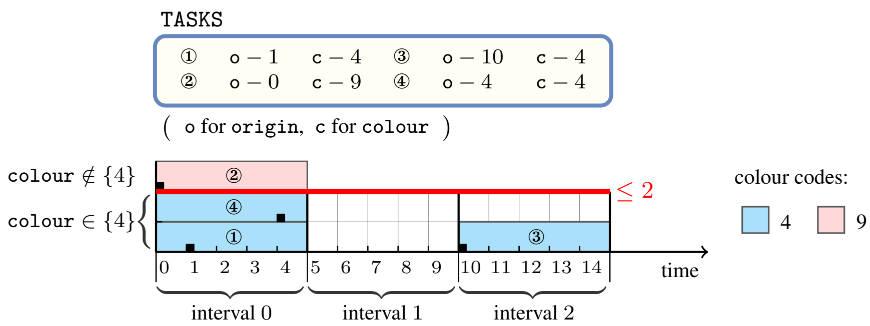


Figure 5.465: The INTERVAL_AND_COUNT solution to the **Example** slot with the use of each interval

Symmetries

- ATMOST can be increased.
- Items of COLOURS are permutable.
- Items of TASKS are permutable.
- One and the same constant can be added to the origin attribute of all items of TASKS.
- An occurrence of a value of TASKS.origin that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.
- An occurrence of a value of TASKS.colour that belongs to COLOURS.val (resp. does not belong to COLOURS.val) can be replaced by any other value in COLOURS.val (resp. not in COLOURS.val).

Arg. properties

- Contractible wrt. COLOURS.
- Contractible wrt. TASKS.

Usage This constraint was originally proposed for dealing with timetabling problems. In this context the different intervals are interpreted as morning and afternoon periods of different consecutive days. Each colour corresponds to a type of course (i.e., French, mathematics). There is a restriction on the maximum number of courses of a given type each morning as well as each afternoon.

Remark If we want to only consider intervals that correspond to the morning or to the afternoon we could extend the INTERVAL_AND_COUNT constraint in the following way:

- We introduce two extra parameters REST and QUOTIENT that correspond to non-negative integers such that REST is strictly less than QUOTIENT,
- We add the following condition to the arc constraint: $(\texttt{tasks1.origin}/\texttt{SIZE\_INTERVAL}) \equiv \texttt{REST}(\bmod \texttt{QUOTIENT})$

Now, if we want to express a constraint on the morning intervals, we set REST to 0 and QUOTIENT to 2.

Reformulation Let K denote the index of the last possible interval where the tasks can be assigned: $K = \lfloor \frac{\max_{i\in[1,|\texttt{TASKS}|]}(\texttt{TASKS}[i].\texttt{origin})+\texttt{SIZE\_INTERVAL}-1}{\texttt{SIZE\_INTERVAL}} \rfloor$. The

INTERVAL_AND_COUNT(ATMOST, COLOURS, TASKS, SIZE_INTERVAL) constraint can be expressed in term of a set of reified constraints and of K arithmetic constraints (i.e., SUM_CTR constraints).

1. For each task TASKS$[i]$ ($i \in [1, |\texttt{TASKS}|]$) of the TASKS collection we create a 0-1 variable B_i that will be set to 1 if and only if task TASKS$[i]$ takes a colour within the set of colours COLOURS:
$B_i \Leftrightarrow \texttt{TASKS}[i].\texttt{colour} = \texttt{COLOURS}[1].\texttt{val} \vee$
$\texttt{TASKS}[i].\texttt{colour} = \texttt{COLOURS}[2].\texttt{val} \vee$
$\dots\dots\dots\dots\dots\dots$
$\texttt{TASKS}[i].\texttt{colour} = \texttt{COLOURS}[|\texttt{COLOURS}|].\texttt{val}$.
2. For each task TASKS$[i]$ ($i \in [1, |\texttt{TASKS}|]$) and for each interval $[k \cdot \texttt{SIZE\_INTERVAL}, k \cdot \texttt{SIZE\_INTERVAL} + \texttt{SIZE\_INTERVAL} - 1]$ ($k \in [0, K]$) we create a 0-1 variable B_{ik} that will be set to 1 if and only if, both task TASKS$[i]$ takes a colour within the set of colours COLOURS, and the origin of task TASKS$[i]$ is assigned within interval $[k \cdot \texttt{SIZE\_INTERVAL}, k \cdot \texttt{SIZE\_INTERVAL} + \texttt{SIZE\_INTERVAL} - 1]$:
$B_{ik} \Leftrightarrow B_i \wedge$
$\texttt{TASKS}[i].\texttt{origin} \geq k \cdot \texttt{SIZE\_INTERVAL} \wedge$
$\texttt{TASKS}[i].\texttt{origin} \leq k \cdot \texttt{SIZE\_INTERVAL} + \texttt{SIZE\_INTERVAL} - 1$
3. Finally, for each interval $[k \cdot \texttt{SIZE\_INTERVAL}, k \cdot \texttt{SIZE\_INTERVAL} + \texttt{SIZE\_INTERVAL} - 1]$ ($k \in [0, K]$), we impose the sum $B_{1k} + B_{2k} + \cdots + B_{|\texttt{TASKS}|k}$ to not exceed the maximum allowed capacity ATMOST.

See also

assignment dimension removed: AMONG_LOW_UP *(assignment dimension corresponding to intervals is removed).*

related: INTERVAL_AND_SUM *(*AMONG_LOW_UP *constraint replaced by* SUM_CTR*).*

used in graph description: AMONG_LOW_UP.

Keywords

application area: assignment.

characteristic of a constraint: coloured, automaton, automaton with array of counters.

constraint type: timetabling constraint, resource constraint, temporal constraint.

modelling: assignment dimension, interval.

Arc input(s)	TASKS TASKS
Arc generator	$PRODUCT \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	tasks1.origin/SIZE_INTERVAL = tasks2.origin/SIZE_INTERVAL
Sets	SUCC $\mapsto$ [source, variables − col(VARIABLES−collection(var−dvar), [item(var − TASKS.colour)])]
Constraint(s) on sets	AMONG_LOW_UP(0, ATMOST, variables, COLOURS)

Graph model We use a bipartite graph where each class of vertices corresponds to the different tasks of the TASKS collection. There is an arc between two tasks if their origins belong to the same interval. Finally we enforce an AMONG_LOW_UP constraint on each set $\mathcal{S}$ of successors of the different vertices of the final graph. This put a restriction on the maximum number of tasks of $\mathcal{S}$ for which the colour attribute takes its value in COLOURS.

Parts (A) and (B) of Figure 5.466 respectively show the initial and final graph associated with the **Example** slot. Each connected component of the final graph corresponds to items that are all assigned to the same interval.

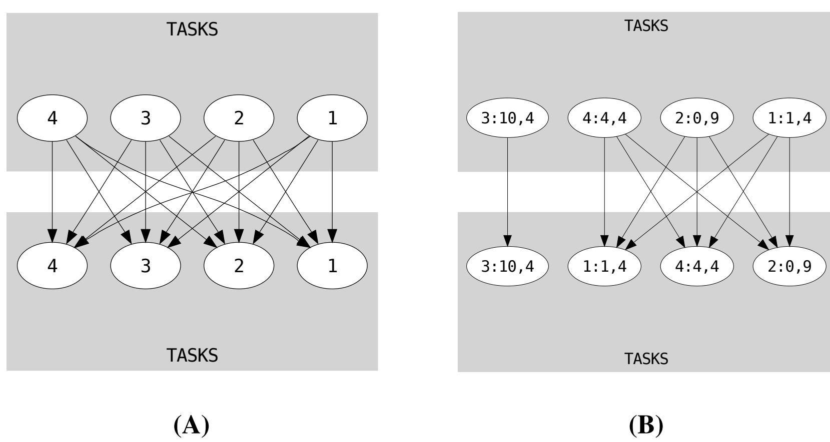


Figure 5.466: Initial and final graph of the INTERVAL_AND_COUNT constraint

Automaton Figure 5.467 depicts the automaton associated with the INTERVAL_AND_COUNT constraint. Let $\texttt{COLOUR}_i$ be the `colour` attribute of the i^{th} item of the `TASKS` collection. To each pair $(\texttt{COLOURS}, \texttt{COLOUR}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $\texttt{COLOUR}_i \in \texttt{COLOURS} \Leftrightarrow S_i$.

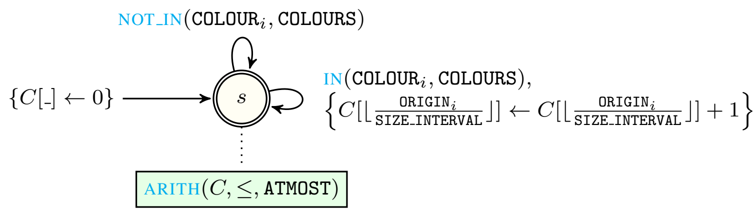


Figure 5.467: Automaton of the INTERVAL_AND_COUNT constraint

5.200 INTERVAL_AND_SUM

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from CUMULATIVE.

Constraint INTERVAL_AND_SUM(SIZE_INTERVAL, TASKS, LIMIT)

Arguments

SIZE_INTERVAL	:	int
TASKS	:	collection(origin−dvar, height−dvar)
LIMIT	:	int

Restrictions

SIZE_INTERVAL > 0
required(TASKS, [origin, height])
TASKS.origin ≥ 0
TASKS.height ≥ 0
LIMIT ≥ 0

Purpose

A maximum resource capacity constraint: We have to fix the origins of a collection of tasks in such a way that, for all the tasks that are allocated to the same interval, the sum of the heights does not exceed a given capacity. All the intervals we consider have the following form: $[k \cdot \mathtt{SIZE\_INTERVAL}, k \cdot \mathtt{SIZE\_INTERVAL} + \mathtt{SIZE\_INTERVAL} - 1]$, where k is an integer.

Example

$$\left(5, \left\langle \begin{array}{ll} \mathtt{origin}-1 & \mathtt{height}-2, \\ \mathtt{origin}-10 & \mathtt{height}-2, \\ \mathtt{origin}-10 & \mathtt{height}-3, \\ \mathtt{origin}-4 & \mathtt{height}-1 \end{array} \right\rangle, 5\right)$$

Figure 5.468 shows the solution associated with the example. The constraint INTERVAL_AND_SUM holds since the sum of the heights of the tasks that are located in the same interval does not exceed the limit 5. Each task t is depicted by a rectangle r associated with the interval to which the task t is assigned. The rectangle r is labelled with the position of t within the items of the TASKS collection. The origin of task t is represented by a small black square located within its corresponding rectangle r. Finally, the height of a rectangle r is equal to the height of the task t to which it corresponds.

Typical

SIZE_INTERVAL > 1
$|\mathtt{TASKS}| > 1$
range(TASKS.origin) > 1
range(TASKS.height) > 1
LIMIT $<$ sum(TASKS.height)

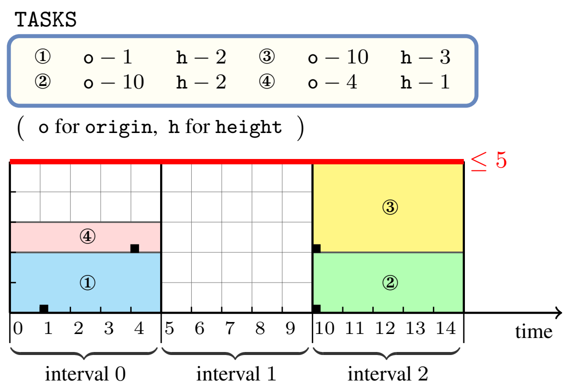


Figure 5.468: The INTERVAL_AND_SUM solution to the **Example** slot with the use of each interval

Symmetries

- Items of TASKS are permutable.
- One and the same constant can be added to the origin attribute of all items of TASKS.
- An occurrence of a value of TASKS.origin that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.
- TASKS.height can be decreased to any value ≥ 0.
- LIMIT can be increased.

Arg. properties

Contractible wrt. TASKS.

Usage

This constraint can be use for timetabling problems. In this context the different intervals are interpreted as morning and afternoon periods of different consecutive days. We have a capacity constraint for all tasks that are assigned to the same morning or afternoon of a given day.

Reformulation

Let K denote the index of the last possible interval where the tasks can be assigned: $K = \lfloor \frac{\max_{i\in[1,|\mathtt{TASKS}|]}(\overline{\mathtt{TASKS}[i].\mathtt{origin}})+\mathtt{SIZE\_INTERVAL}-1}{\mathtt{SIZE\_INTERVAL}} \rfloor$. The INTERVAL_AND_SUM(SIZE_INTERVAL, TASKS, LIMIT) constraint can be expressed in term of a set of reified constraints and of K arithmetic constraints (i.e., SCALAR_PRODUCT constraints).

1. For each task TASKS$[i]$ ($i \in [1, |\mathtt{TASKS}|]$) and for each interval $[k \cdot \mathtt{SIZE\_INTERVAL}, k \cdot \mathtt{SIZE\_INTERVAL} + \mathtt{SIZE\_INTERVAL} - 1]$ ($k \in [0, K]$) we create a 0-1 variable B_{ik} that will be set to 1 if and only if the origin of task TASKS$[i]$ is assigned within interval $[k \cdot \mathtt{SIZE\_INTERVAL}, k \cdot \mathtt{SIZE\_INTERVAL} + \mathtt{SIZE\_INTERVAL} - 1]$:
$B_{ik} \Leftrightarrow \mathtt{TASKS}[i].\mathtt{origin} \geq k \cdot \mathtt{SIZE\_INTERVAL} \wedge$
$\mathtt{TASKS}[i].\mathtt{origin} \leq k \cdot \mathtt{SIZE\_INTERVAL} + \mathtt{SIZE\_INTERVAL} - 1$
2. Finally, for each interval $[k \cdot \mathtt{SIZE\_INTERVAL}, k \cdot \mathtt{SIZE\_INTERVAL} + \mathtt{SIZE\_INTERVAL} - 1]$ ($k \in [0, K]$), we impose the sum $\mathtt{TASKS}[1].\mathtt{height} \cdot$

B_{1k} + TASKS[2].height $\cdot B_{2k} + \cdots +$ TASKS[|TASKS|].height $\cdot B_{|\mathtt{TASKS}|k}$ to not exceed the maximum allowed capacity LIMIT.

See also

assignment dimension removed: SUM_CTR *(assignment dimension corresponding to intervals is removed).*

related: INTERVAL_AND_COUNT *(*SUM_CTR *constraint replaced by* AMONG_LOW_UP*).*

used in graph description: SUM_CTR.

Keywords

application area: assignment.

characteristic of a constraint: automaton, automaton with array of counters.

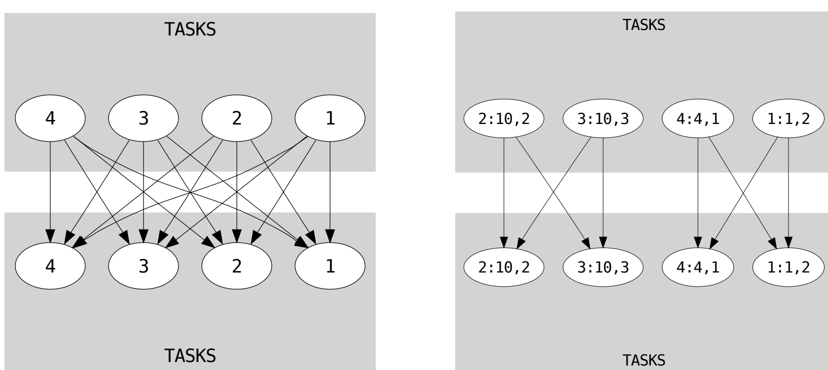
constraint type: timetabling constraint, resource constraint, temporal constraint.

modelling: assignment dimension, interval.

Arc input(s)	TASKS TASKS
Arc generator	$PRODUCT \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	tasks1.origin/SIZE_INTERVAL = tasks2.origin/SIZE_INTERVAL
Sets	SUCC $\mapsto$ [source, variables − col(VARIABLES−collection(var−dvar), [item(var − TASKS.height)])]
Constraint(s) on sets	SUM_CTR(variables, $\leq$, LIMIT)

Graph model We use a bipartite graph where each class of vertices corresponds to the different tasks of the TASKS collection. There is an arc between two tasks if their origins belong to the same interval. Finally we enforce a SUM_CTR constraint on each set $\mathcal{S}$ of successors of the different vertices of the final graph. This put a restriction on the maximum value of the sum of the height attributes of the tasks of $\mathcal{S}$.

Parts (A) and (B) of Figure 5.469 respectively show the initial and final graph associated with the **Example** slot. Each connected component of the final graph corresponds to items that are all assigned to the same interval.



(A) **(B)**

Figure 5.469: Initial and final graph of the INTERVAL_AND_SUM constraint

Automaton Figure 5.470 depicts the automaton associated with the INTERVAL_AND_SUM constraint. To each item of the collection TASKS corresponds a signature variable S_i that is equal to 1.

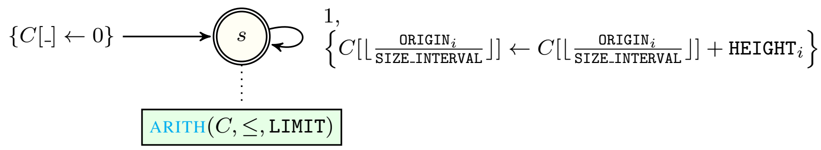


Figure 5.470: Automaton of the INTERVAL_AND_SUM constraint

5.201 INVERSE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint INVERSE(NODES)

Synonyms ASSIGNMENT, CHANNEL, INVERSE_CHANNELING.

Argument NODES : collection(index−int, succ−dvar, pred−dvar)

Restrictions

required(NODES, [index, succ, pred])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |NODES|
NODES.pred ≥ 1
NODES.pred ≤ |NODES|

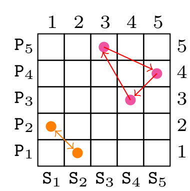
Purpose

Enforce each vertex of a digraph to have exactly one predecessor and one successor. In addition the following two statements are equivalent:

1. The successor of the i^{th} node is the j^{th} node.
2. The predecessor of the j^{th} node is the i^{th} node.

Example

$$\left(\left\langle\begin{array}{lll}\texttt{index}-1 & \texttt{succ}-2 & \texttt{pred}-2,\\ \texttt{index}-2 & \texttt{succ}-1 & \texttt{pred}-1,\\ \texttt{index}-3 & \texttt{succ}-5 & \texttt{pred}-4,\\ \texttt{index}-4 & \texttt{succ}-3 & \texttt{pred}-5,\\ \texttt{index}-5 & \texttt{succ}-4 & \texttt{pred}-3\end{array}\right\rangle\right)$$



The INVERSE constraint holds since:

- NODES[1].succ = 2 ⇔ NODES[2].pred = 1,
- NODES[2].succ = 1 ⇔ NODES[1].pred = 2,
- NODES[3].succ = 5 ⇔ NODES[5].pred = 3,
- NODES[4].succ = 3 ⇔ NODES[3].pred = 4,
- NODES[5].succ = 4 ⇔ NODES[4].pred = 5.

All solutions

Figure 5.471 gives all solutions to the following non ground instance of the INVERSE constraint: $S_1 \in [2,4]$, $P_1 \in [1,4]$, $S_2 \in [1,2]$, $P_2 \in [1,4]$, $S_3 \in [1,4]$, $P_3 \in [1,4]$, $S_4 \in [2,4]$, $P_4 \in [1,3]$, INVERSE(⟨1 S_1 P_1, 2 S_2 P_2, 3 S_3 P_3, 4 S_4 P_4⟩).

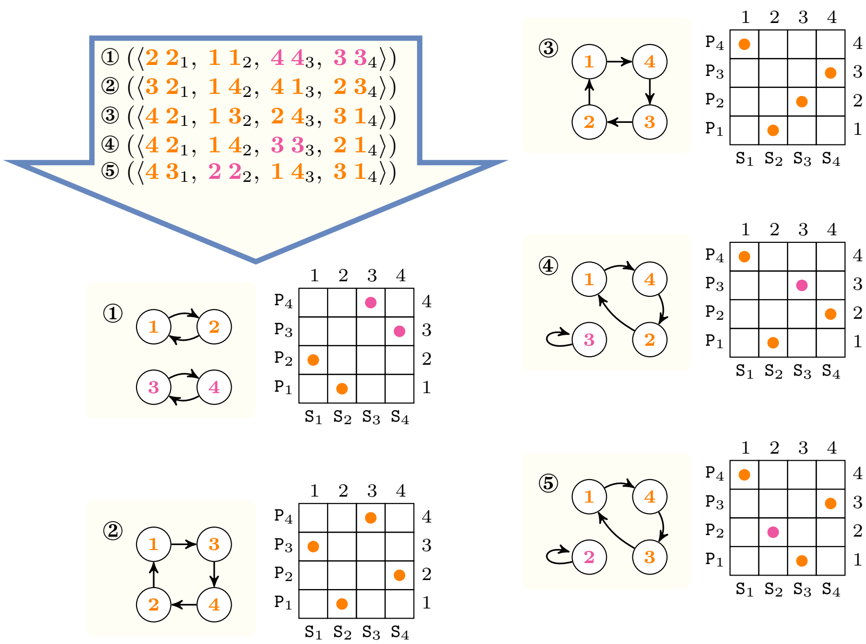


Figure 5.471: All solutions corresponding to the non ground example of the INVERSE constraint of the **All solutions** slot; in the left-hand side the `index` attributes are displayed as indices of the `pred` attribute, while in the right-hand side and in the lower part they are directly displayed within each node of the corresponding graph.

Typical $|\texttt{NODES}| > 1$

Symmetries

- Items of `NODES` are permutable.
- Attributes of `NODES` are permutable w.r.t. permutation (`index`) (`succ`, `pred`) (*permutation applied to all items*).

Arg. properties

- Functional dependency: `NODES.succ` determined by `NODES.index` and `NODES.pred`.
- Functional dependency: `NODES.pred` determined by `NODES.index` and `NODES.succ`.

Usage This constraint is used in order to make the link between the successor and the predecessor variables. This is sometimes required by specific heuristics that use both predecessor and successor variables. In some problems, the successor and predecessor variables are respectively interpreted as *column* an *row* variables (i.e., we have a bijection between the successor variables and their values). This is the case, for example, in the n-queens problem (i.e., place n queens on an n by n chessboard in such a way that no two queens are on the same row, the same column or the same diagonal) when we use the following model: to each column of the chessboard we associate a variable that gives the row where the corresponding queen is located. Symmetrically, to each row of the chessboard we create a variable that indicates the column where the associated queen is placed. Having these two sets of variables, we can now write a heuristic that selects the column or the row for which we have the fewest number of alternatives for placing a queen.

Remark In the original INVERSE constraint of **CHIP** the `index` attribute was not explicitly present. It was implicitly defined as the position of a variable in a list, the first position being 1. This is also the case for **SICStus Prolog**, **JaCoP** and **Gecode** where the variables are respectively indexed from 1, 0 and 0. Within **SICStus Prolog** and **JaCoP** (`http://www.jacop.eu/`), the INVERSE constraint is called ASSIGNMENT. Within **Gecode**, it is called CHANNEL (`http://www.gecode.org/`).

Algorithm An arc-consistency filtering algorithm for the INVERSE constraint is described in [138, 139]. The algorithm is based on the following ideas:

- We first normalize the domains of the variables by removing value i from the j^{th} predecessor variable if value j does not belong to the i^{th} successor variable, and by removing value j from the i^{th} successor variable if value i does not belong to the j^{th} predecessor variable.
- Second, one can map solutions to the INVERSE constraint to perfect matchings in a so-called variable bipartite graph derived from the domain of the variables of the constraint in the following way: to each successor variable corresponds a vertex; similarly to each predecessor variable corresponds a vertex; there is and edge between the i^{th} successor variable and the j^{th} predecessor variable if and only if value i belongs to the domain of the j^{th} predecessor variable and value j belongs to the domain of the i^{th} successor variable.
- Third, Dulmage-Mendelsohn decomposition [157] is used to characterise all edges that do not belong to any perfect matching, and therefore prune the corresponding variables.

Systems INVERSECHANNELING in **Choco**, CHANNEL in **Gecode**, INVERSE in **MiniZinc**, ASSIGNMENT in **SICStus**.

See also **common keyword:** CYCLE, SYMMETRIC_ALLDIFFERENT *(permutation)*.

generalisation: INVERSE_OFFSET *(do not assume anymore that the smallest value of the* `pred` *or* `succ` *attributes is equal to* 1*)*, INVERSE_SET *(*`domain variable` *replaced by* `set variable`*)*, INVERSE_WITHIN_RANGE *(partial mapping between two collections of distinct size)*.

implies (items to collection): LEX_ALLDIFFERENT.

related: INVERSE_EXCEPT_LOOP.

Keywords **characteristic of a constraint:** automaton, automaton with array of counters.

combinatorial object: permutation.

constraint arguments: pure functional dependency.

constraint type: graph constraint.

filtering: bipartite matching, arc-consistency.

heuristics: heuristics.

modelling: channelling constraint, permutation channel, dual model, functional dependency.

modelling exercises: n-Amazons, zebra puzzle.

puzzles: n-Amazons, n-queens, zebra puzzle.

Arc input(s)	NODES
Arc generator	*CLIQUE*↦collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	• nodes1.succ = nodes2.index • nodes2.pred = nodes1.index
Graph property(ies)	**NARC**= |NODES|

Graph model In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the INVERSE constraint considers objects that have three attributes:

- One fixed attribute index that is the identifier of the vertex,
- One variable attribute succ that is the successor of the vertex,
- One variable attribute pred that is the predecessor of the vertex.

Parts (A) and (B) of Figure 5.472 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

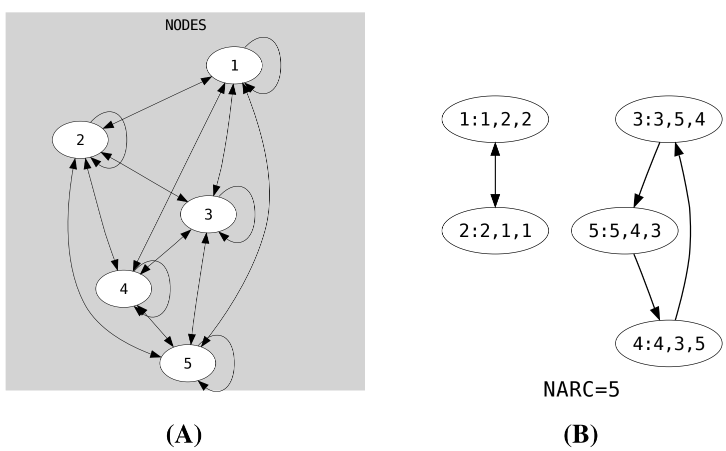


Figure 5.472: Initial and final graph of the INVERSE constraint

Signature Since all the index attributes of the NODES collection are distinct and because of the first condition nodes1.succ = nodes2.index of the arc constraint all the vertices of the final graph have at most one predecessor.

Since all the index attributes of the NODES collection are distinct and because of the second condition nodes2.pred = nodes1.index of the arc constraint all the vertices of the final graph have at most one successor.

From the two previous remarks it follows that the final graph is made up from disjoint paths and disjoint circuits. Therefore the maximum number of arcs of the final graph is equal to its maximum number of vertices NODES. So we can rewrite the graph property $\mathbf{NARC} = |\mathtt{NODES}|$ to $\mathbf{NARC} \geq |\mathtt{NODES}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 5.473 depicts the automaton associated with the INVERSE constraint. To each item of the collection NODES corresponds a signature variable S_i that is equal to 1.

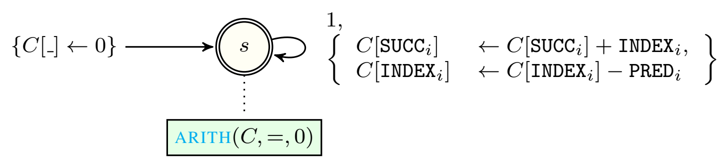


Figure 5.473: Automaton of the INVERSE constraint

5.202 INVERSE_EXCEPT_LOOP

DESCRIPTION LINKS

Origin Derived from INVERSE

Constraint INVERSE_EXCEPT_LOOP(NODES)

Argument NODES : collection(index−int, succ−dvar, pred−dvar)

Restrictions
required(NODES, [index, succ, pred])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|
NODES.pred ≥ 1
NODES.pred $\leq$ |NODES|

Purpose

Enforce the following conditions:

1. $\texttt{NODES}[i].\texttt{succ} = j \wedge i \neq j \Leftrightarrow \texttt{NODES}[j].\texttt{pred} = i \wedge i \neq j$,
2. $\texttt{NODES}[i].\texttt{succ} = i \Leftrightarrow \forall j \in [1, n] : \texttt{NODES}[j].\texttt{pred} \neq i$,
3. $\texttt{NODES}[i].\texttt{pred} = i \Leftrightarrow \forall j \in [1, n] : \texttt{NODES}[j].\texttt{succ} \neq i$.

Example

$$\left(\left\langle\begin{array}{lll} \texttt{index}-1 & \texttt{succ}-3 & \texttt{pred}-1, \\ \texttt{index}-2 & \texttt{succ}-4 & \texttt{pred}-2, \\ \texttt{index}-3 & \texttt{succ}-3 & \texttt{pred}-1, \\ \texttt{index}-4 & \texttt{succ}-5 & \texttt{pred}-2, \\ \texttt{index}-5 & \texttt{succ}-5 & \texttt{pred}-4 \end{array}\right\rangle\right)$$

Figure 5.474 illustrates the constraint of the **Example** slot. The INVERSE_EXCEPT_LOOP constraint holds since:

1.
   - $\texttt{NODES}[1].\texttt{succ} = 3 \wedge 1 \neq 3 \Leftrightarrow \texttt{NODES}[3].\texttt{pred} = 1 \wedge 3 \neq 1$,
   - $\texttt{NODES}[2].\texttt{succ} = 4 \wedge 2 \neq 4 \Leftrightarrow \texttt{NODES}[4].\texttt{pred} = 2 \wedge 4 \neq 2$,
   - $\texttt{NODES}[4].\texttt{succ} = 5 \wedge 4 \neq 5 \Leftrightarrow \texttt{NODES}[5].\texttt{pred} = 4 \wedge 5 \neq 4$.
2.
   - $\texttt{NODES}[3].\texttt{succ} = 3 \Leftrightarrow \forall j \in [1, 5] : \texttt{NODES}[j].\texttt{pred} \neq 3$,
   - $\texttt{NODES}[5].\texttt{succ} = 5 \Leftrightarrow \forall j \in [1, 5] : \texttt{NODES}[j].\texttt{pred} \neq 5$.
3.
   - $\texttt{NODES}[1].\texttt{pred} = 1 \Leftrightarrow \forall j \in [1, 5] : \texttt{NODES}[j].\texttt{succ} \neq 1$,
   - $\texttt{NODES}[2].\texttt{pred} = 2 \Leftrightarrow \forall j \in [1, 5] : \texttt{NODES}[j].\texttt{succ} \neq 2$.

Typical $|\texttt{NODES}| > 1$

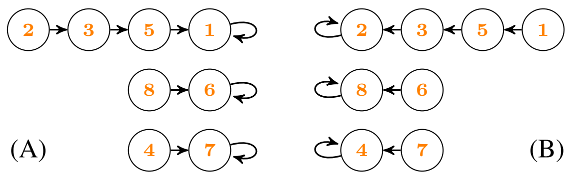


Figure 5.474: (A) Successor and (B) predecessor views corresponding to the constraint of the **Example** slot

Arg. properties

- Functional dependency: NODES.succ determined by NODES.index and NODES.pred.
- Functional dependency: NODES.pred determined by NODES.index and NODES.succ.

Usage

The INVERSE_EXCEPT_LOOP constraint can be used in the reformulation of the PATH(NPATH, $\langle 1\ s_1, \ldots, n\ s_n \rangle$) constraint. This reformulation is based on the use of two TREE constraints as well as on the use of one INVERSE_EXCEPT_LOOP channeling constraint for connecting the two TREE constraints:

- TREE(NPATH, $\langle 1\ s_1, \ldots, n\ s_n \rangle$),
- INVERSE_EXCEPT_LOOP($\langle 1\ s_1\ p_1, \ldots, n\ s_n\ p_n \rangle$),
- TREE(NPATH, $\langle 1\ p_1, \ldots, n\ p_n \rangle$).

In the context of one single path this reformulation was first mentioned in the PhD thesis of J.-G. Fages [166].

See also

related: INVERSE.

Keywords

constraint arguments: pure functional dependency.

constraint type: graph constraint.

modelling: channelling constraint, dual model, functional dependency.

5.203 INVERSE_OFFSET

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Gecode

Constraint INVERSE_OFFSET(SOFFSET, POFFSET, NODES)

Synonym CHANNEL.

Arguments

SOFFSET : int
POFFSET : int
NODES : collection(index−int, succ−dvar, pred−dvar)

Restrictions

required(NODES, [index, succ, pred])
$\text{NODES.index} \geq 1$
$\text{NODES.index} \leq |\text{NODES}|$
distinct(NODES, index)
$\text{NODES.succ} \geq 1 + \text{SOFFSET}$
$\text{NODES.succ} \leq |\text{NODES}| + \text{SOFFSET}$
$\text{NODES.pred} \geq 1 + \text{POFFSET}$
$\text{NODES.pred} \leq |\text{NODES}| + \text{POFFSET}$

Purpose

Enforce each vertex of a digraph to have exactly one predecessor and one successor. In addition the following two statements are equivalent:

1. The successor of the i^{th} node minus **SOFFSET** is equal to j.
2. The predecessor of the j^{th} node minus **POFFSET** is equal to i.

I.e., $\text{NODES}[i].\text{succ} - \textbf{SOFFSET} = j \Leftrightarrow \text{NODES}[j].\text{pred} - \textbf{POFFSET} = i$.

Example

$$\left(\begin{array}{l} -1, 0, \left\langle \begin{array}{lll} \text{index}-1 & \text{succ}-4 & \text{pred}-3, \\ \text{index}-2 & \text{succ}-2 & \text{pred}-5, \\ \text{index}-3 & \text{succ}-0 & \text{pred}-2, \\ \text{index}-4 & \text{succ}-6 & \text{pred}-8, \\ \text{index}-5 & \text{succ}-1 & \text{pred}-1, \\ \text{index}-6 & \text{succ}-7 & \text{pred}-7, \\ \text{index}-7 & \text{succ}-5 & \text{pred}-4, \\ \text{index}-8 & \text{succ}-3 & \text{pred}-6 \end{array} \right\rangle \end{array}\right)$$

The INVERSE_OFFSET constraint holds since:

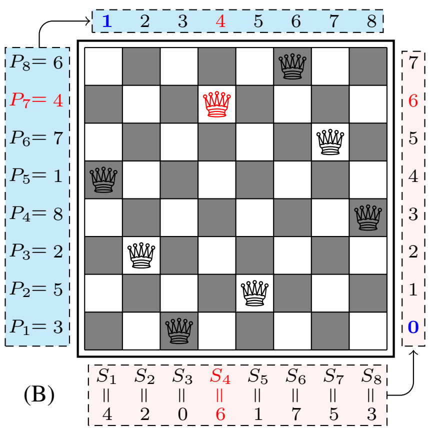
- $\text{NODES}[1].\text{succ} - (-1) = 5 \Leftrightarrow \text{NODES}[5].\text{pred} - 0 = 1$,
- $\text{NODES}[2].\text{succ} - (-1) = 3 \Leftrightarrow \text{NODES}[3].\text{pred} - 0 = 2$,
- $\text{NODES}[3].\text{succ} - (-1) = 1 \Leftrightarrow \text{NODES}[1].\text{pred} - 0 = 3$,
- $\text{NODES}[4].\text{succ} - (-1) = 7 \Leftrightarrow \text{NODES}[7].\text{pred} - 0 = 4$,
- $\text{NODES}[5].\text{succ} - (-1) = 2 \Leftrightarrow \text{NODES}[2].\text{pred} - 0 = 5$.

- $\texttt{NODES}[6].\texttt{succ} - (-1) = 8 \Leftrightarrow \texttt{NODES}[8].\texttt{pred} - 0 = 6$.
- $\texttt{NODES}[7].\texttt{succ} - (-1) = 6 \Leftrightarrow \texttt{NODES}[6].\texttt{pred} - 0 = 7$.
- $\texttt{NODES}[8].\texttt{succ} - (-1) = 4 \Leftrightarrow \texttt{NODES}[4].\texttt{pred} - 0 = 8$.

Figure 5.475 shows the board that can be associated with this example.

(A) NODES

①	index − 1	succ − 4	pred − 3
②	index − 2	succ − 2	pred − 5
③	index − 3	succ − 0	pred − 2
④	index − 4	succ − 6	pred − 8
⑤	index − 5	succ − 1	pred − 1
⑥	index − 6	succ − 7	pred − 7
⑦	index − 7	succ − 5	pred − 4
⑧	index − 8	succ − 3	pred − 6



(B)

SOFFSET = 1 POFFSET = 0

$S_1 - 1 = 5 \Leftrightarrow P_5 - 0 = 1$

$S_2 - 1 = 3 \Leftrightarrow P_3 - 0 = 2$

$S_3 - 1 = 1 \Leftrightarrow P_1 - 0 = 3$

$S_4 - 1 = 7 \Leftrightarrow P_7 - 0 = 4$

$S_5 - 1 = 2 \Leftrightarrow P_2 - 0 = 5$

$S_6 - 1 = 8 \Leftrightarrow P_8 - 0 = 6$

$S_7 - 1 = 6 \Leftrightarrow P_6 - 0 = 7$

$S_8 - 1 = 4 \Leftrightarrow P_4 - 0 = 8$

(C)

Figure 5.475: **Example** slot where we highlight the fourth item in red showing the relation between S_4 and P_7, where S_i and P_i (with $1 \leq i \leq 8$) respectively stands for the successor and predecessor attributes of the i^{th} item of the NODES collection (A) Collection of nodes passed to the INVERSE_OFFSET constraint, (B) Corresponding board, (C) Conditions linking the successor and the predecessor attributes via the offsets SOFFSET = 1 and POFFSET = 0.

Typical

SOFFSET ≥ -1
SOFFSET ≤ 1
POFFSET ≥ -1
POFFSET ≤ 1
$|\texttt{NODES}| > 1$

Symmetry

Items of NODES are permutable.

Arg. properties

- Functional dependency: NODES.succ determined by SOFFSET, POFFSET, NODES.index and NODES.pred.
- Functional dependency: NODES.pred determined by SOFFSET, POFFSET, NODES.index and NODES.succ.

Remark

The INVERSE_OFFSET constraint is called CHANNEL in **Gecode** (http://www.gecode.org/). Having two offsets was motivated by the fact that it is possible to declare arrays at any position in the **MiniZinc** modelling language.

Systems

INVERSECHANNELING in **Choco**, CHANNEL in **Gecode**.

See also

specialisation: INVERSE *(assume that* SOFFSET *and* POFFSET *are both equal to* 0*)*.

Keywords

constraint arguments: pure functional dependency.

constraint type: graph constraint.

filtering: arc-consistency.

heuristics: heuristics.

modelling: channelling constraint, dual model, functional dependency.

Arc input(s) NODES

Arc generator $CLIQUE \mapsto$ `collection`(nodes1, nodes2)

Arc arity 2

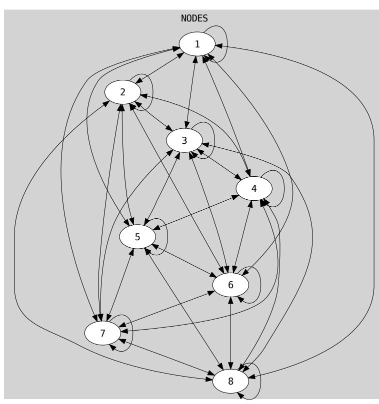
Arc constraint(s)
- nodes1.succ − SOFFSET = nodes2.index
- nodes2.pred − POFFSET = nodes1.index

Graph property(ies) **NARC**= |NODES|

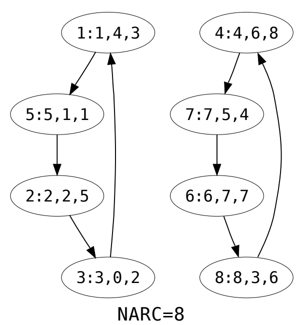
Graph model In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices. This is why the INVERSE_OFFSET constraint considers objects that have three attributes:

- One fixed attribute `index` that is the identifier of the vertex,
- One variable attribute `succ` that is the successor of the vertex,
- One variable attribute `pred` that is the predecessor of the vertex.

Parts (A) and (B) of Figure 5.476 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A)



(B)

Figure 5.476: Initial and final graph of the INVERSE_OFFSET constraint

5.204 INVERSE_SET

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from INVERSE.

Constraint INVERSE_SET(X, Y)

Arguments

$$\begin{array}{lll} \texttt{X} & : & \texttt{collection}(\texttt{index}-\texttt{int}, \texttt{set}-\texttt{svar}) \\ \texttt{Y} & : & \texttt{collection}(\texttt{index}-\texttt{int}, \texttt{set}-\texttt{svar}) \end{array}$$

Restrictions

$\texttt{required}(\texttt{X}, [\texttt{index}, \texttt{set}])$
$\texttt{required}(\texttt{Y}, [\texttt{index}, \texttt{set}])$
$\texttt{increasing\_seq}(\texttt{X}, \texttt{index})$
$\texttt{increasing\_seq}(\texttt{Y}, \texttt{index})$
$\texttt{X.index} \geq 1$
$\texttt{X.index} \leq |\texttt{X}|$
$\texttt{Y.index} \geq 1$
$\texttt{Y.index} \leq |\texttt{Y}|$
$\texttt{X.set} \geq 1$
$\texttt{X.set} \leq |\texttt{Y}|$
$\texttt{Y.set} \geq 1$
$\texttt{Y.set} \leq |\texttt{X}|$

Purpose

The following two statements are equivalent:

1. Value j belongs to the set variable of the i^{th} item of the X collection.
2. Value i belongs to the set variable of the j^{th} item of the Y collection.

I.e., $j \in \texttt{X}[i] \Leftrightarrow i \in \texttt{Y}[j]$.

Example

$$\left(\begin{array}{l} \left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{set}-\{2,4\}, \\ \texttt{index}-2 & \texttt{set}-\{4\}, \\ \texttt{index}-3 & \texttt{set}-\{1\}, \\ \texttt{index}-4 & \texttt{set}-\{4\} \end{array}\right\rangle, \\ \left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{set}-\{3\}, \\ \texttt{index}-2 & \texttt{set}-\{1\}, \\ \texttt{index}-3 & \texttt{set}-\emptyset, \\ \texttt{index}-4 & \texttt{set}-\{1,2,4\}, \\ \texttt{index}-5 & \texttt{set}-\emptyset \end{array}\right\rangle \end{array}\right)$$

The INVERSE_SET constraint holds since:

$$\left\{\begin{array}{l} 2 \in \texttt{X}[1].\texttt{set} \Leftrightarrow 1 \in \texttt{Y}[2].\texttt{set}, \quad 4 \in \texttt{X}[1].\texttt{set} \Leftrightarrow 1 \in \texttt{Y}[4].\texttt{set}, \\ 4 \in \texttt{X}[2].\texttt{set} \Leftrightarrow 2 \in \texttt{Y}[4].\texttt{set}, \\ 1 \in \texttt{X}[3].\texttt{set} \Leftrightarrow 3 \in \texttt{Y}[1].\texttt{set}, \\ 4 \in \texttt{X}[4].\texttt{set} \Leftrightarrow 4 \in \texttt{Y}[4].\texttt{set}. \end{array}\right.$$

Typical

$|\mathtt{X}| > 1$
$|\mathtt{Y}| > 1$

Symmetries

- Arguments are permutable w.r.t. permutation $(\mathtt{X}, \mathtt{Y})$.
- Items of X are permutable.
- Items of Y are permutable.

Usage

The INVERSE_SET constraint can be used, for example, to model problems where one has to place items on a rectangular board in such a way that a column or a row can have more than one item. We have one set variable for each row of the board; Its values are the column indexes corresponding to the positions where an item is placed. Similarly we have also one set variable for each column of the board; Its values are the row indexes corresponding to the positions where an item is placed. The INVERSE_SET constraint maintains the link between the rows and the columns variables. Figure 5.477 shows the board that can be associated with the example of the **Example** slot.

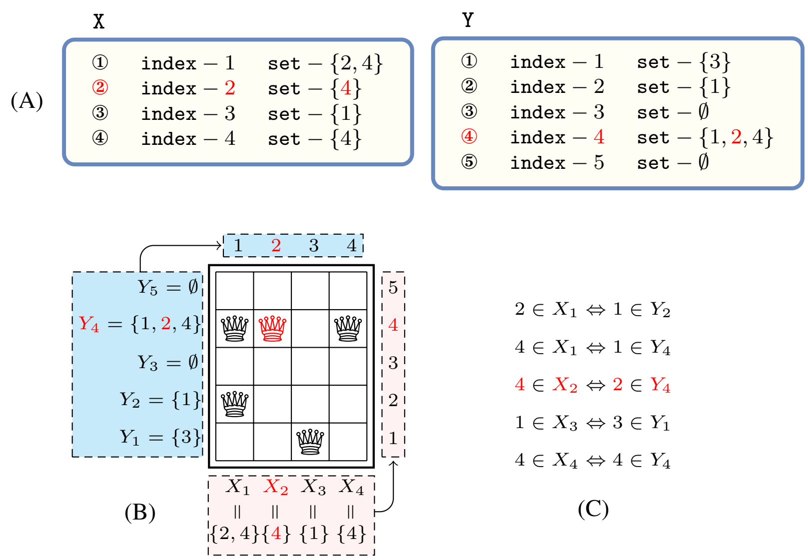


Figure 5.477: Illustration of the **Example** slot where we highlight in red the second item of the X collection and the fourth item of the Y collection showing the relation between X_2 and Y_4, where X_i (with $1 \leq i \leq 4$) and Y_j (with $1 \leq j \leq 5$) respectively stands for the set attribute of the i^{th} item of the X collection and of the j^{th} item of the Y collection (A) Collections X and Y passed to the INVERSE_SET constraint, (B) Corresponding board, (C) Conditions linking the items of X and the items of Y.

Systems

INVERSESET in **Choco**, INVERSE_SET in **MiniZinc**.

See also

common keyword: INVERSE_WITHIN_RANGE *(channelling constraint)*.

specialisation: INVERSE *(*set variable *replaced by* domain variable*)*.

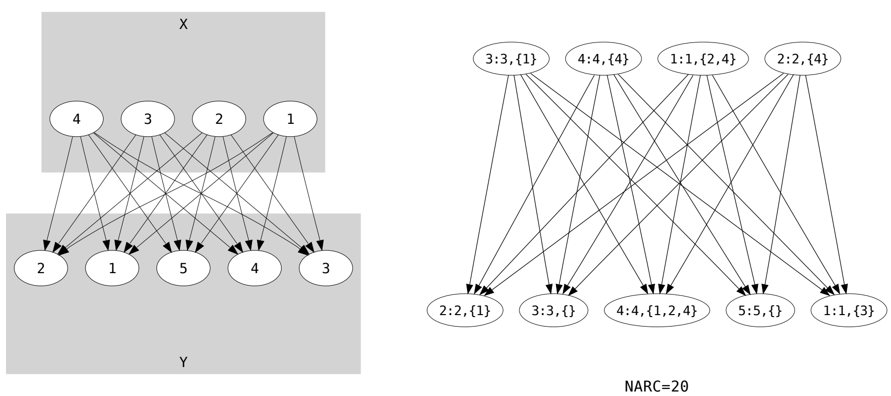
used in graph description: IN_SET.

Keywords **constraint arguments:** constraint involving set variables.

modelling: channelling constraint, set channel, dual model.

Arc input(s)	X Y
Arc generator	*PRODUCT* $\mapsto$ collection(x, y)
Arc arity	2
Arc constraint(s)	IN_SET(y.index, x.set) $\Leftrightarrow$ IN_SET(x.index, y.set)
Graph property(ies)	**NARC** $= \lvert X\rvert * \lvert Y\rvert$

Graph model Parts (A) and (B) of Figure 5.478 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.478: Initial and final graph of the INVERSE_SET constraint

5.205 INVERSE_WITHIN_RANGE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from INVERSE.

Constraint INVERSE_WITHIN_RANGE(X, Y)

Synonyms INVERSE_IN_RANGE, INVERSE_RANGE.

Arguments

X : collection(var−dvar)
Y : collection(var−dvar)

Restrictions

required(X, var)
required(Y, var)

Purpose

If the i^{th} variable of the collection X is assigned to j and if j is greater than or equal to 1 and less than or equal to the number of items of the collection Y then the j^{th} variable of the collection Y is assigned to i.
Conversely, if the j^{th} variable of the collection Y is assigned to i and if i is greater than or equal to 1 and less than or equal to the number of items of the collection X then the i^{th} variable of the collection X is assigned to j.

Example

$(\langle 9, 4, 2\rangle, \langle 9, 3, 9, 2\rangle)$

Since the second item of X is assigned to 4, the fourth item of Y is assigned to 2. Similarly, since the third item of X is assigned to 2, the second item of Y is assigned to 3. Figure 5.479 illustrates the correspondence between X and Y.

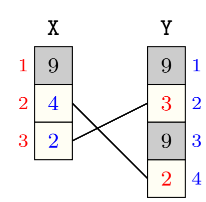


Figure 5.479: Correspondence between the items of $X = \langle 9, 4, 2\rangle$ and the items of $Y = \langle 9, 3, 9, 2\rangle$: on the X side values between 1 and $|Y| = 4$ are shown in blue, on the Y side values between 1 and $|X| = 3$ are shown in red.

Typical

$|X| > 1$
range(X.var) > 1
$|Y| > 1$
range(Y.var) > 1

Symmetry Arguments are permutable w.r.t. permutation (X, Y).

Usage Consider an integer value m and a sequence of n variables S from which you have to select a subsequence S' such that:

- All variables of S' have to be assigned to distinct values from $[1, m]$,
- All variables not in S' have to be assigned a value, not necessarily distinct, outside $[1, m]$.

As for the INVERSE constraint we may want to create explicitly a *value variable* for each value in [1,m] in order to state some specific constraints on the *value variables* or to use a heuristic involving the original variables of S as well as the *value variables*. The purpose of the INVERSE_WITHIN_RANGE constraint is to link the variables of S with the *value variables*.

See also **common keyword:** INVERSE_SET *(channelling constraint)*.

specialisation: INVERSE *(the 2 collections have not necessarly the same number of items)*.

Keywords **constraint arguments:** constraint between two collections of variables.

constraint type: graph constraint.

final graph structure: bipartite, no loop, symmetric.

heuristics: heuristics.

modelling: channelling constraint, dual model.

Arc input(s)	X Y
Arc generator	$SYMMETRIC\_PRODUCT \mapsto$ collection(s1, s2)
Arc arity	2
Arc constraint(s)	s1.var = s2.key
Graph class	• BIPARTITE • NO_LOOP • SYMMETRIC

5.206 ITH_POS_DIFFERENT_FROM_0

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin N. Beldiceanu

Constraint ITH_POS_DIFFERENT_FROM_0(ITH, POS, VARIABLES)

Arguments

ITH	:	int
POS	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

ITH ≥ 1
ITH $\leq$ |VARIABLES|
POS $\geq$ ITH
POS $\leq$ |VARIABLES|
required(VARIABLES, var)

Purpose POS is the position of the ITHth non-zero item of the sequence of variables VARIABLES.

Example $(2, 4, \langle 3, 0, 0, 8, 6\rangle)$

The ITH_POS_DIFFERENT_FROM_0 constraint holds since 4 corresponds to the position of the 2^{th} non-zero item of the sequence 3 0 0 8 6.

Typical

|VARIABLES| > 1
range(VARIABLES.var) > 1
ATLEAST(1, VARIABLES, 0)

Typical model ATLEAST(2, VARIABLES, 0)

Symmetry An occurrence of a value of VARIABLES.var that is different from 0 can be replaced by any other value that is also different from 0.

Arg. properties Suffix-extensible wrt. VARIABLES.

Keywords

characteristic of a constraint: joker value, automaton, automaton with counters.

constraint network structure: alpha-acyclic constraint network(3).

constraint type: data constraint.

modelling: table.

Automaton Figure 5.480 depicts the automaton associated with the ITH_POS_DIFFERENT_FROM_0 constraint. To each variable $\mathtt{VAR}_i$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$ and S_i: $\mathtt{VAR}_i = 0 \Leftrightarrow S_i$.

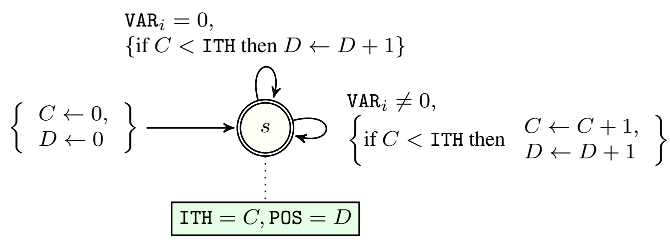


Figure 5.480: Automaton of the ITH_POS_DIFFERENT_FROM_0 constraint

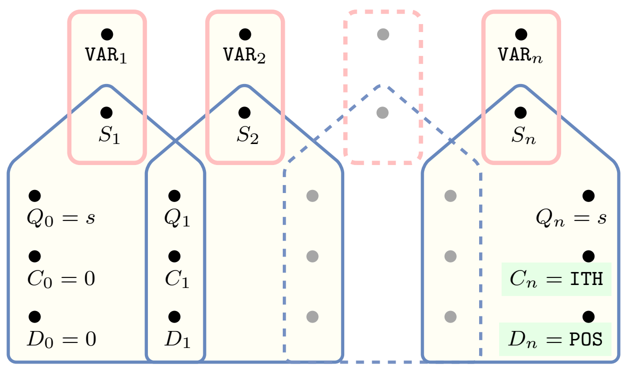


Figure 5.481: Hypergraph of the reformulation corresponding to the automaton of the ITH_POS_DIFFERENT_FROM_0 constraint

5.207 K_ALLDIFFERENT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [160]

Constraint K_ALLDIFFERENT(VARS)

Synonyms K_ALLDIFF, K_ALLDISTINCT, SOME_DIFFERENT.

Type X : collection(x−dvar)

Argument VARS : collection(vars − X)

Restrictions
$|\mathtt{X}| \geq 1$
required(X, x)
required(VARS, vars)
$|\mathtt{VARS}| \geq 1$

Purpose For each collection of variables depicted by an item of VARS, enforce their corresponding variables to take distinct values. Usually some variables occur in several collections.

Example $(\langle \mathtt{vars} - \langle 5, 6, 0, 9, 3\rangle, \mathtt{vars} - \langle 5, 6, 1, 2\rangle\rangle)$

The K_ALLDIFFERENT constraint holds since all the values 5, 6, 0, 9 and 3 are distinct and since all the values 5, 6, 1 and 2 are distinct as well.

Typical
$|\mathtt{X}| > 1$
$|\mathtt{VARS}| > 1$

Symmetries
- Items of VARS are permutable.
- Items of VARS.vars are permutable.
- All occurrences of two distinct values of VARS.vars.x can be swapped; all occurrences of a value of VARS.vars.x can be renamed to any unused value.

Arg. properties Contractible wrt. VARS.

Usage Systems of ALLDIFFERENT constraints sharing variables occurs frequently in practice. We give 4 typical problems that can be modelled by a combination of ALLDIFFERENT constraints as well as one problem where a system of ALLDIFFERENT constraints provides a necessary condition.

- The *graph colouring* problem is to colour with a restricted number of colours the vertices of a given undirected graph in such a way that adjacent vertices are coloured with distinct colours. The problem can be modelled by a system of ALLDIFFERENT constraints. All the next problems can been seen as graph colouring problems where the graphs have some specific structure.

- A *Latin square of order* n is an $n \times n$ array in which n distinct numbers in $[1, n]$ are arranged so that each number occurs once in each row and column. The problem is to complete a partially filled Latin square. Part (A) of Figure 5.482 gives a partially filled Latin square, while part (B) provides a possible completion.

1			
			3
3			
			1

(A)

1	2	3	4
2	1	4	**3**
3	4	1	2
4	3	2	**1**

(B)

Figure 5.482: (A) A partially filled Latin square and (B) a possible completion

- A *Sudoku* is a Latin square of order 9×9 such that the numbers in each major 3×3 block are distinct. As for the Latin square problem, the problem is to complete a partially filled board. Part (A) of Figure 5.483 gives a partially filled Sudoku board, while part (B) provides a possible completion. A constraint programming approach for solving Sudoku puzzles is depicted in [395]. It shows how to generate redundant constraints as well as shaving [287] in order to find a solution without guessing.

	2	6				8	1	
3			7		8			6
4				5				7
	5		1		7		9	
		3	9		5	1		
	4		3		2		5	
1				3				2
5			2		4			9
	3	8				4	6	

(A)

7	2	6	4	9	3	8	1	5
3	1	5	7	2	8	9	4	6
4	8	9	6	5	1	2	3	7
8	5	2	1	4	7	6	9	3
6	7	3	9	8	5	1	2	4
9	4	1	3	6	2	7	5	8
1	9	4	8	3	6	5	7	2
5	6	7	2	1	4	3	8	9
2	3	8	5	7	9	4	6	1

(B)

Figure 5.483: (A) A partially filled Sudoku square and (B) its unique completion

- A *task assignment* problem consists to assign a given set of non-preemptive tasks, which are fixed in time (i.e., the origin, duration and end of each task are fixed), to a set of resources so that, tasks that are assigned to the same resource do not overlap in time. Each task can be assigned to a predefined set of resources. Problems like *aircraft stand allocation* [149], [394] or *air traffic flow management* [21] correspond to an example of a real-life task assignment problem. *Assignment of service professionals* [14] is yet another industrial example where professionals have to be assigned positions in such a way that positions assigned to a given professional do not overlap in time.

  Part (A) of Figure 5.484 gives an example of task assignment problem. For each task we indicate the set of resources where it can potentially be assigned (i.e., the domain of its assignment variable). For instance, task t_1 can be assigned to resources 1 or 2. Part (B) of Figure 5.484 gives the corresponding interval graph: We have one vertex for each task and an edge between two tasks that overlap in time. We have a system of ALLDIFFERENT constraints corresponding to the maximum cliques of the interval graph (i.e., $\{t_1, t_5, t_8\}$, $\{t_2, t_5, t_8\}$, $\{t_2, t_6\}$, $\{t_3, t_6, t_9\}$, $\{t_3, t_7, t_9\}$, $\{t_4, t_7, t_9\}$). Finally, part (C) of Figure 5.484 provides a possible solution to the task assignment problem where tasks t_1, t_2, t_9 are assigned to resource 1, tasks t_3, t_4, t_8 are assigned to resource 2, and tasks t_5, t_6, t_7 are assigned to resource 3.

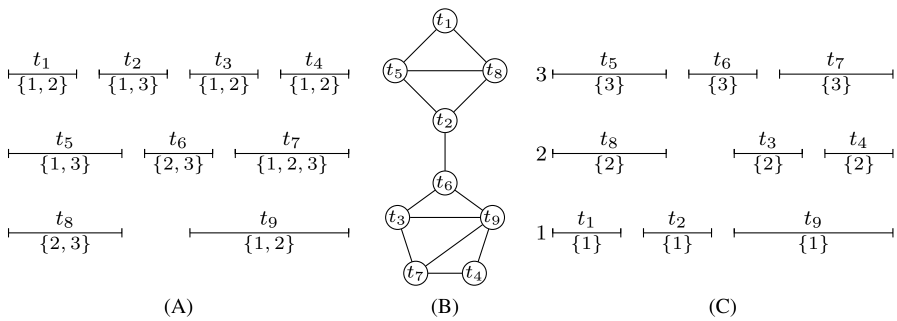


Figure 5.484: (A) Tasks $t_1, t_2, \ldots, t_9$ with their potential assignments 1, 2 or 3 (B) Interval graph where to each task of corresponds a vertex, and to each pair of overlapping tasks corresponds an edge (C) A valid assignment where tasks assigned to a same machine do not overlap

- The *tree partitioning with precedences* problem is to compute a vertex-partitioning of a given digraph $\mathcal{G}$ in disjoint trees (i.e., a forest), so that a given set of precedences holds. The problem can be modelled with a TREE_PRECEDENCE(`NTREE`, `VERTICES`) constraint, where `NTREE` is a domain variable specifying the numbers of trees in the forest and `VERTICES` is a collection of the digraph's n vertices. Each item $v \in$ `VERTICES` has the following attributes, which complete the description of the digraph:

  - `index` is an integer in $[1, n]$ that can be interpreted as the *label* of v.
  - `father` is a domain variable whose domain consists of elements (vertex label) of $[1, n]$. It can be interpreted as the *unique successor* of v.
  - `preds` is a possibly empty set of integers, its elements (vertex label) being in $[1, n]$. It can be interpreted as the *mandatory ancestors* of v.

  We model the TREE_PRECEDENCE constraint by the digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in which the vertices represent the elements of `VERTICES` and the arcs represent the successors relations between them. Formally, $\mathcal{G}$ is defined as follows:

  - To the i^{th} vertex ($1 \leq i \leq n$), `VERTICES`$[i]$, of the `VERTICES` collection corresponds a vertex of $\mathcal{V}$ denoted by v_i.
  - For every pair of vertices (`VERTICES`$[i]$,`VERTICES`$[j]$), where i and j are not necessarily distinct, there is an arc from v_i to v_j in $\mathcal{E}$.

  The TREE_PRECEDENCE constraint specifies that its associated digraph $\mathcal{G}$ should be a forest that fulfils the precedence constraints. Formally a ground instance of a TREE_PRECEDENCE(NTREE, VERTICES) constraint is satisfied if and only if the following conditions hold:

1. $\forall i \in [1, n]$: VERTICES$[i]$.index $= i$,
2. Its associated digraph $\mathcal{G}$ consists of NTREE connected components,
3. Each connected component of $\mathcal{G}$ does not contain any circuit involving more than one vertex,
4. For every vertex VERTICES$[i]$ such that $j \in$ VERTICES$[i]$.preds there must be an elementary path in $\mathcal{G}$ from VERTICES$[j]$ to VERTICES$[i]$.

We can build the following system of ALLDIFFERENT constraints that corresponds to a necessary condition for the TREE_PRECEDENCE constraint: To each vertex v of $\mathcal{G}$, which both has no predecessors and cannot be the root of a tree, we generate an ALLDIFFERENT constraint involving the father variables of those descendants of v in $\mathcal{G}$ that cannot be the root of a tree.

For the set of precedences depicted by part (A) of Figure 5.485[13], where we assume that VERTICES$[12]$ is the only vertex that can be a root and where F_i denotes the father variable associated with VERTICES$[i]$, we get the following system of ALLDIFFERENT constraints:

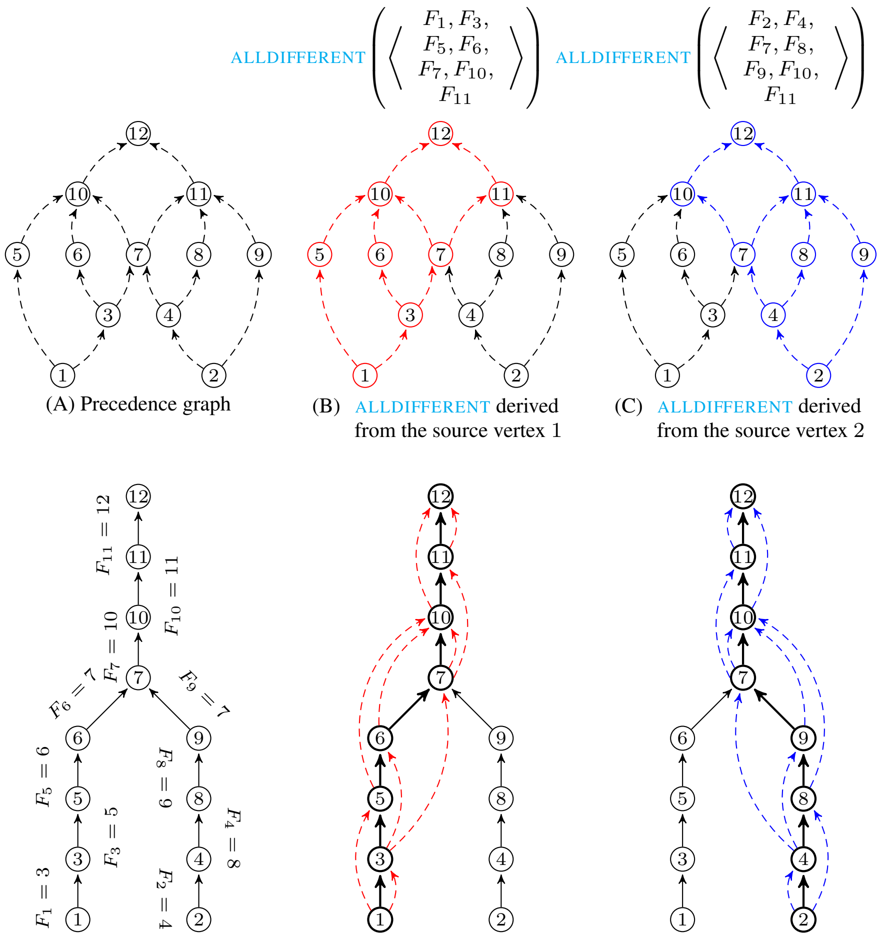
- ALLDIFFERENT$(\langle F_1, F_3, F_5, F_6, F_7, F_{10}, F_{11}\rangle)$,
- ALLDIFFERENT$(\langle F_2, F_4, F_7, F_8, F_9, F_{10}, F_{11}\rangle)$.

The variables of these two ALLDIFFERENT constraints respectively correspond to the descendants of the two source vertices (i.e., F_1 and F_2) of the precedence graph depicted by parts (B) and (C) of Figure 5.485. On part (B) and (C) of Figure 5.485 the descendants of F_1 and F_2 are respectively depicted in red and blue. Their intersections, $\{F_7, F_{10}, F_{11}, F_{12}\}$, from which we remove F_{12} belong to the two ALLDIFFERENT constraints. In fact, F_{12} is not mentioned in the two ALLDIFFERENT constraints since its corresponding vertex is the root of a tree. Part (D) of Figure 5.485 gives a possible tree satisfying all the precedences constraints expressed by part (A). It corresponds to the following ground solution:

TREE_PRECEDENCE($\langle$ index $-$ 1 father $-$ 3 preds $-\{\}$,
index $-$ 2 father $-$ 4 preds $-\{\}$,
index $-$ 3 father $-$ 5 preds $-\{1\}$,
index $-$ 4 father $-$ 8 preds $-\{2\}$,
index $-$ 5 father $-$ 6 preds $-\{1\}$,
index $-$ 6 father $-$ 7 preds $-\{3\}$,
index $-$ 7 father $-$ 10 preds $-\{3,4\}$,
index $-$ 8 father $-$ 9 preds $-\{4\}$,
index $-$ 9 father $-$ 7 preds $-\{2\}$,
index $-$ 10 father $-$ 11 preds $-\{5,6,7\}$,
index $-$ 11 father $-$ 12 preds $-\{7,8,9\}$,
index $-$ 12 father $-$ 12 preds $-\{10,11\}$ $\rangle$)

Parts (E) and (F) of Figure 5.485 illustrate how the precedence constraints are satisfied by the solution depicted by part (D): each precedence, represented by a dashed arc, links two vertices that belong to a same path of the tree that is directed toward the root of the tree.

[13] The number in a vertex gives the value of the index attribute of the corresponding item.



(A) Precedence graph

(B) ALLDIFFERENT derived from the source vertex 1

(C) ALLDIFFERENT derived from the source vertex 2

(D) A tree satisfying the precedence graph

(E) Checking the red precedences on the path from the source 1 to the sink 12 of the tree depicted in (D)

(F) Checking the blue precedences on the path from the source 2 to the sink 12 of the tree depicted in (D)

Figure 5.485: (A) A set of precedences and (D) a corresponding feasible tree where F_i stands for the father of the i^{th} vertex; (B) the ALLDIFFERENT constraint associated with the source vertex 1 and (E) the satisfied precedences in red along the paths of the tree of (D); (C) the ALLDIFFERENT constraint associated with the source vertex 2 and (F) the satisfied precedences in blue along the paths of the tree of (D);

Remark It was shown in [161] that, finding out whether a system of two ALLDIFFERENT constraints sharing some variables has a solution or not is NP-hard. This was achieved by reduction from set packing.

A slight variation in the way of describing the arguments of the K_ALLDIFFERENT constraint appears in [368] under the name of SOME_DIFFERENT: the set of disequalities is described by a set of pairs of variables, where each pair corresponds to a disequality constraint between two given variables.

Within the context of linear programming, a relaxation of the K_ALLDIFFERENT constraint is provided in [9]. The special case where $k = 2$ is discussed in [10].

Algorithm Even if there is no filtering algorithm for the K_ALLDIFFERENT constraint, one can enforce redundant constraints for the following patterns:

- Within the context of graph colouring, one can state an NVALUE constraint for every cycle of odd length of the graph to colour enforcing that the corresponding variables have to be assigned to at least three distinct values.
- Within the context of Latin squares, one can state a COLORED_MATRIX constraint enforcing that each value is used exactly once in each row and column.
- Within the context of two ALLDIFFERENT constraints ALLDIFFERENT($\langle U_1, \ldots, U_n, V_1, \ldots, V_m \rangle$) and ALLDIFFERENT($\langle U_1, \ldots, U_n, W_1, \ldots, W_m \rangle$) where the domain of all variables $U_1, \ldots, U_n$, $V_1, \ldots, V_m$, $W_1, \ldots, W_m$ is included in the interval $[1, n + m]$, one can state a SAME_AND_GLOBAL_CARDINALITY constraint stating that the variables $V_1, \ldots, V_m$ should correspond to a permutation of the variables $W_1, \ldots, W_m$ and that the variables $V_1, \ldots, V_m$ should be assigned to distinct values.
- In the general case of two ALLDIFFERENT constraints ALLDIFFERENT($\langle U_1, \ldots, U_n, V_1, \ldots, V_m \rangle$) and ALLDIFFERENT($\langle U_1, \ldots, U_n, W_1, \ldots, W_o \rangle$), one can state an NVALUE constraint involving the variables $V_1, \ldots, V_m$ and $W_1, \ldots, W_o$ enforcing that these variables should not use more than $s - n$ distinct values, where s denotes the cardinality of the union of the domains of the variables $U_1, \ldots, U_n$, $V_1, \ldots, V_m$, $W_1, \ldots, W_o$.

Several propagation rules for the K_ALLDIFFERENT constraint are also described in [264].

Reformulation Given two ALLDIFFERENT constraints that share some variables, a reformulation preserving bound-consistency was introduced in [80]. This reformulation is based on an extension of Hall's theorem that is presented in the same paper.

See also

common keyword: COLORED_MATRIX *(system of constraints)*.

generalisation: DIFFN, GEOST *(tasks for which the start attribute is not fixed)*.

part of system of constraints: ALLDIFFERENT.

related: NVALUE *(implied by two overlapping* ALLDIFFERENT*)*, SAME_AND_GLOBAL_CARDINALITY *(implied by two overlapping* ALLDIFFERENT *and restriction on values)*.

Keywords

application area: air traffic management, assignment.

characteristic of a constraint: all different, disequality.

combinatorial object: permutation, Latin square.

complexity: set packing.

constraint type: system of constraints, overlapping alldifferent, value constraint, decomposition.

filtering: bound-consistency, duplicated variables.

problems: graph colouring.

puzzles: Sudoku.

For all items of VARS:

Arc input(s)	VARS.vars
Arc generator	$CLIQUE \mapsto$ collection(x1, x2)
Arc arity	2
Arc constraint(s)	x1.x = x2.x
Graph property(ies)	**MAX_NSCC** ≤ 1

Graph model For each collection of variables depicted by an item of VARS we generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one.

5.208 K_CUT

DESCRIPTION LINKS GRAPH

Origin E. Althaus

Constraint K_CUT(K, NODES)

Arguments

K : int
NODES : collection(index−int, succ−svar)

Restrictions

$\mathtt{K} \geq 1$
$\mathtt{K} \leq |\mathtt{NODES}|$
required(NODES, [index, succ])
$\mathtt{NODES.index} \geq 1$
$\mathtt{NODES.index} \leq |\mathtt{NODES}|$
distinct(NODES, index)
$\mathtt{NODES.succ} \geq 1$
$\mathtt{NODES.succ} \leq |\mathtt{NODES}|$

Purpose Select some arcs of a digraph in order to have at least K connected components (an isolated vertex, i.e. a vertex without any ingoing or outgoing arc, is counted as one connected component).

Example

$$\left(3, \left\langle \begin{array}{ll} \mathtt{index}-1 & \mathtt{succ}-\emptyset, \\ \mathtt{index}-2 & \mathtt{succ}-\{3,5\}, \\ \mathtt{index}-3 & \mathtt{succ}-\{5\}, \\ \mathtt{index}-4 & \mathtt{succ}-\emptyset, \\ \mathtt{index}-5 & \mathtt{succ}-\{2,3\} \end{array} \right\rangle \right)$$

The K_CUT constraint holds since the graph corresponding to the NODES collection contains 3 connected components (i.e., two connected components respectively involving vertices 1 and 4 and a third connected component containing the remaining vertices 2, 3 and 5), and since the first argument K enforces to have at least 3 connected components.

Typical $|\mathtt{NODES}| > 1$

Symmetries

- K can be decreased to any value ≥ 1.
- Items of NODES are permutable.

See also **common keyword:** LINK_SET_TO_BOOLEANS *(constraint involving set variables)*.

used in graph description: IN_SET.

Keywords **constraint arguments:** constraint involving set variables.

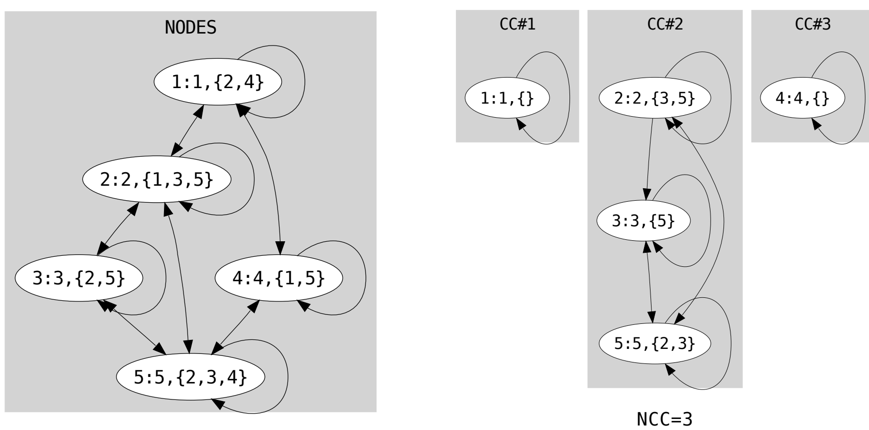
constraint type: graph constraint.

filtering: linear programming.

final graph structure: connected component.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \texttt{nodes1.index} = \texttt{nodes2.index}, \\ \texttt{IN\_SET}(\texttt{nodes2.index}, \texttt{nodes1.succ}) \end{array} \right)$
Graph property(ies)	**NCC** $\geq$ K

Graph model nodes1.index = nodes2.index holds if nodes1 and nodes2 correspond to the same vertex. It is used in order to enforce keeping all the vertices of the initial graph. This is because an isolated vertex counts always as one connected component. Within the context of the **Example** slot, part (A) of Figure 5.486 shows the initial graph from which we have chosen to start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.486 gives the final graph associated with the example of the **Example** slot. The K_CUT constraint holds since we have at least K = 3 connected components in the final graph.



(A) **(B)**

Figure 5.486: Initial and final graph of the K_CUT set constraint

5.209 K_DISJOINT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from DISJOINT

Constraint K_DISJOINT(SETS)

Type VARIABLES : collection(var−dvar)

Argument SETS : collection(set − VARIABLES)

Restrictions
required(VARIABLES, var)
$|\texttt{VARIABLES}| \geq 1$
required(SETS, set)
$|\texttt{SETS}| > 1$

Purpose Given |SETS| sets of domain variables, the K_DISJOINT constraint forces that no value is assigned to more than one set.

Example $(\langle \texttt{set} - \langle 1, 9, 1, 5\rangle, \texttt{set} - \langle 2, 7, 7, 0, 6, 8\rangle, \texttt{set} - \langle 4, 4, 3\rangle\rangle)$

The K_DISJOINT constraint holds since:

- The set of values $\{1, 5, 9\}$ and $\{0, 2, 6, 7, 8\}$ respectively assigned to the variables of the first and second collections have an empty intersection.
- The set of values $\{1, 5, 9\}$ and $\{3, 4\}$ respectively assigned to the variables of the first and third collections have an empty intersection.
- The set of values $\{0, 2, 6, 7, 8\}$ and $\{3, 4\}$ respectively assigned to the variables of the second and third collections have an empty intersection.

Typical $|\texttt{VARIABLES}| > 1$

Symmetries
- Items of SETS are permutable.
- Items of SETS.set are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any value of VARIABLES.var.
- All occurrences of two distinct values of SETS.set.var can be swapped; all occurrences of a value of SETS.set.var can be renamed to any unused value.

Arg. properties Contractible wrt. SETS.

See also **part of system of constraints:** DISJOINT.
used in graph description: DISJOINT.

Keywords **characteristic of a constraint:** disequality.

constraint type: system of constraints, decomposition, value constraint.

modelling: empty intersection.

Arc input(s)	SETS
Arc generator	$CLIQUE(<) \mapsto$ collection(set1, set2)
Arc arity	2
Arc constraint(s)	DISJOINT(set1.set, set2.set)
Graph property(ies)	**NARC** $= \lvert$SETS$\rvert * (\lvert$SETS$\rvert - 1)/2$

Graph model Parts (A) and (B) of Figure 5.487 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a DISJOINT constraint.

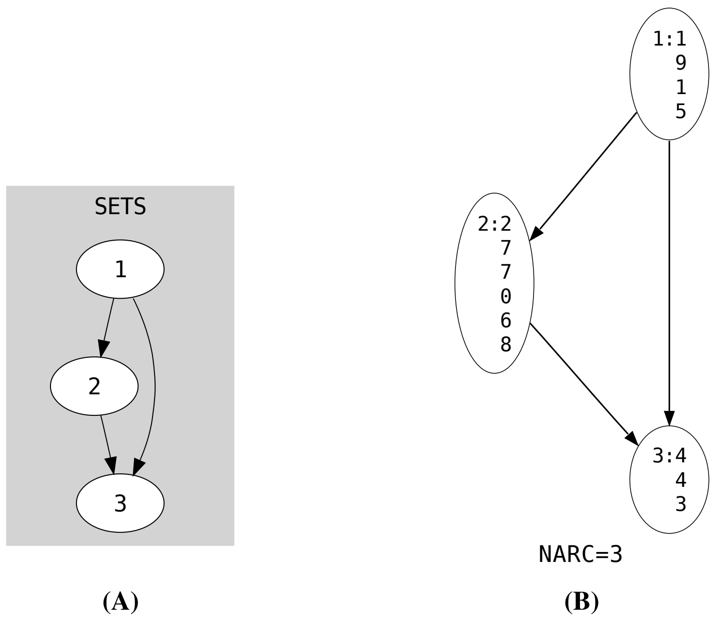


Figure 5.487: Initial and final graph of the K_DISJOINT constraint

5.210 K_SAME

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [160]

Constraint K_SAME(SETS)

Type VARIABLES : collection(var−dvar)

Argument SETS : collection(set − VARIABLES)

Restrictions
required(VARIABLES, var)
$|\text{VARIABLES}| \geq 1$
required(SETS, set)
$|\text{SETS}| > 1$
same_size(SETS, set)

Purpose Given |SETS| sets, each containing the same number of domain variables, the K_SAME constraint forces that the multisets of values assigned to each set are all identical.

Example

$$\left(\left\langle \begin{array}{l} \text{set} - \langle 1,9,1,5,2,1 \rangle, \\ \text{set} - \langle 9,1,1,1,2,5 \rangle, \\ \text{set} - \langle 5,2,1,1,9,1 \rangle \end{array} \right\rangle \right)$$

The K_SAME constraint holds since:

- The first and second collections of variables are assigned to the same multiset.
- The second and third collections of variables are also assigned to the same multiset.

Typical $|\text{VARIABLES}| > 1$

Symmetries
- Items of SETS are permutable.
- Items of SETS.set are permutable.
- All occurrences of two distinct values of SETS.set.var can be swapped; all occurrences of a value of SETS.set.var can be renamed to any unused value.

Arg. properties Contractible wrt. SETS.

Remark It was shown in [160] that, finding out whether the K_SAME constraint has a solution or not is NP-hard when we have more than one SAME constraint. This was achieved by reduction from 3-dimensional-matching in the context where we have 2 SAME constraints.

See also

common keyword: K_SAME_INTERVAL, K_SAME_MODULO, K_SAME_PARTITION *(system of constraints)*.

implies: K_USED_BY.

part of system of constraints: SAME.

used in graph description: SAME.

Keywords

characteristic of a constraint: sort based reformulation.

combinatorial object: permutation, multiset.

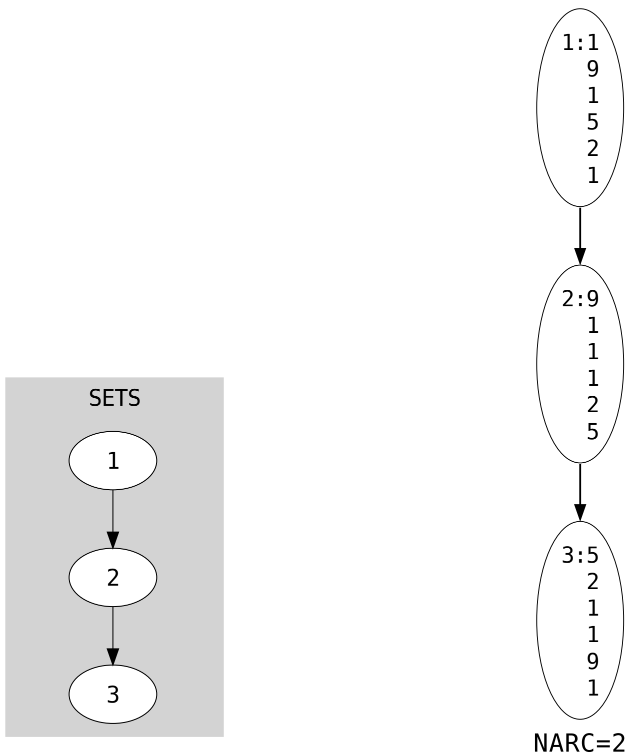
complexity: 3-dimensional-matching.

constraint type: system of constraints, decomposition.

modelling: equality between multisets.

Arc input(s)	SETS
Arc generator	$PATH \mapsto$ collection(set1, set2)
Arc arity	2
Arc constraint(s)	SAME(set1.set, set2.set)
Graph property(ies)	**NARC** $= \lvert$SETS$\rvert - 1$

Graph model Parts (A) and (B) of Figure 5.488 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a SAME constraint.



(A) (B)

Figure 5.488: Initial and final graph of the K_SAME constraint

5.211 K_SAME_INTERVAL

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from SAME_INTERVAL and from K_SAME.

Constraint K_SAME_INTERVAL(SETS, SIZE_INTERVAL)

Type VARIABLES : collection(var−dvar)

Arguments

SETS : collection(set − VARIABLES)
SIZE_INTERVAL : int

Restrictions

required(VARIABLES, var)
$|\texttt{VARIABLES}| \geq 1$
required(SETS, set)
$|\texttt{SETS}| > 1$
same_size(SETS, set)
$\texttt{SIZE\_INTERVAL} > 0$

Purpose Given a collection of |SETS| sets, each containing the same number of domain variables, the K_SAME_INTERVAL constraint forces a SAME_INTERVAL constraint between each pair of consecutive sets.

Example

$$\left(\left\langle\begin{array}{l}\texttt{set} - \langle 1,1,6,0,1,7\rangle,\\ \texttt{set} - \langle 8,8,0,0,1,2\rangle,\\ \texttt{set} - \langle 2,1,1,2,6,6\rangle\end{array}\right\rangle, 3\right)$$

In the example, the second argument SIZE_INTERVAL $= 3$ of the K_SAME_INTERVAL constraint defines the following family of intervals $[3\cdot k, 3\cdot k+2]$, where k is an integer. The K_SAME_INTERVAL constraint holds since:

- The first and second collections of variables are assigned 4 values in the interval $[0,2]$ as well as 2 values in the interval $[6,8]$.
- The second and third collections of variables are also assigned 4 values in the interval $[0,2]$ as well as 2 values in the interval $[6,8]$.

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{SIZE\_INTERVAL} > 1$

Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- An occurrence of a value of SETS.set.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties Contractible wrt. SETS.

See also

common keyword: K_SAME (*system of constraints*).

implies: K_USED_BY_INTERVAL.

part of system of constraints: SAME_INTERVAL.

used in graph description: SAME_INTERVAL.

Keywords

characteristic of a constraint: sort based reformulation.

combinatorial object: permutation.

constraint type: system of constraints, decomposition.

modelling: interval.

Arc input(s)	SETS
Arc generator	$PATH \mapsto$ collection(set1, set2)
Arc arity	2
Arc constraint(s)	SAME_INTERVAL(set1.set, set2.set, SIZE_INTERVAL)
Graph property(ies)	**NARC**$= \|$SETS$\| - 1$

Graph model Parts (A) and (B) of Figure 5.489 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a SAME_INTERVAL constraint.

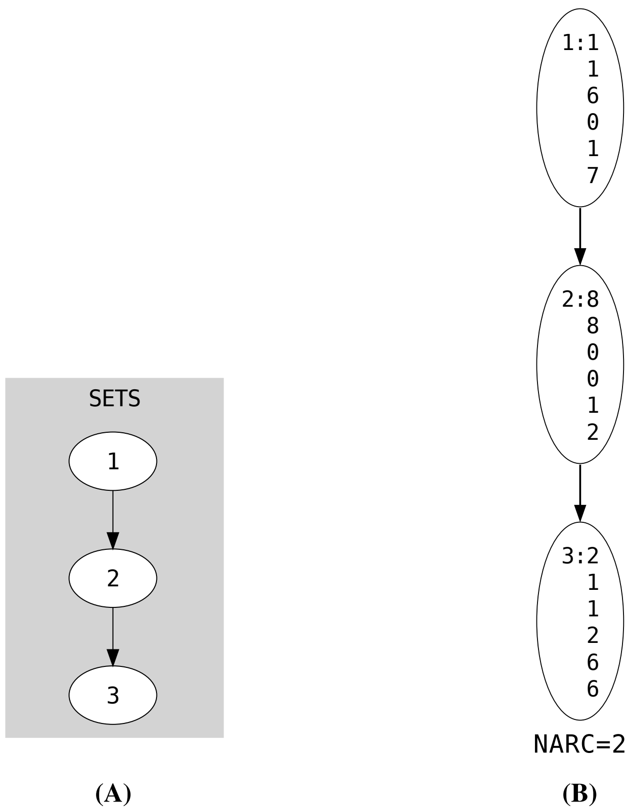


Figure 5.489: Initial and final graph of the K_SAME_INTERVAL constraint

5.212 K_SAME_MODULO

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from SAME_MODULO and from K_SAME.

Constraint K_SAME_MODULO(SETS, M)

Type VARIABLES : collection(var−dvar)

Arguments

SETS : collection(set − VARIABLES)
M : int

Restrictions

required(VARIABLES, var)
$|\text{VARIABLES}| \geq 1$
required(SETS, set)
$|\text{SETS}| > 1$
same_size(SETS, set)
$\text{M} > 0$

Purpose Given a collection of |SETS| sets, each containing the same number of domain variables, the K_SAME_MODULO constraint forces a SAME_MODULO constraint between each pair of consecutive sets.

Example

$$\left(\left\langle\begin{array}{l}\text{set} - \langle 1,9,1,5,2,1\rangle,\\ \text{set} - \langle 6,4,1,1,5,5\rangle,\\ \text{set} - \langle 1,3,4,2,8,7\rangle\end{array}\right\rangle, 3\right)$$

The K_SAME_MODULO constraint holds since:

- The first and second collections of variables are assigned 1 value in $\{0,3,\ldots,3\cdot k\}$, 3 values in $\{1,4,\ldots,1+3\cdot k\}$ and 2 values in $\{2,5,\ldots,2+3\cdot k\}$.
- The second and third collections of variables are also assigned 1 value in $\{0,3,\ldots,3\cdot k\}$, 3 values in $\{1,4,\ldots,1+3\cdot k\}$ and 2 values in $\{2,5,\ldots,2+3\cdot k\}$.

Typical

$|\text{VARIABLES}| > 1$
$\text{M} > 1$

Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- An occurrence of a value u of SETS.set.var can be replaced by any other value v such that v is congruent to u modulo M.

Arg. properties Contractible wrt. SETS.

See also

common keyword: K_SAME *(system of constraints)*.

implies: K_USED_BY_MODULO.

part of system of constraints: SAME_MODULO.

used in graph description: SAME_MODULO.

Keywords

characteristic of a constraint: sort based reformulation, modulo.

combinatorial object: permutation.

constraint type: system of constraints, decomposition.

Arc input(s)	SETS
Arc generator	$PATH \mapsto$ collection(set1, set2)
Arc arity	2
Arc constraint(s)	SAME_MODULO(set1.set, set2.set, M)
Graph property(ies)	**NARC** $= \vert$SETS$\vert - 1$

Graph model Parts (A) and (B) of Figure 5.490 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a SAME_MODULO constraint.

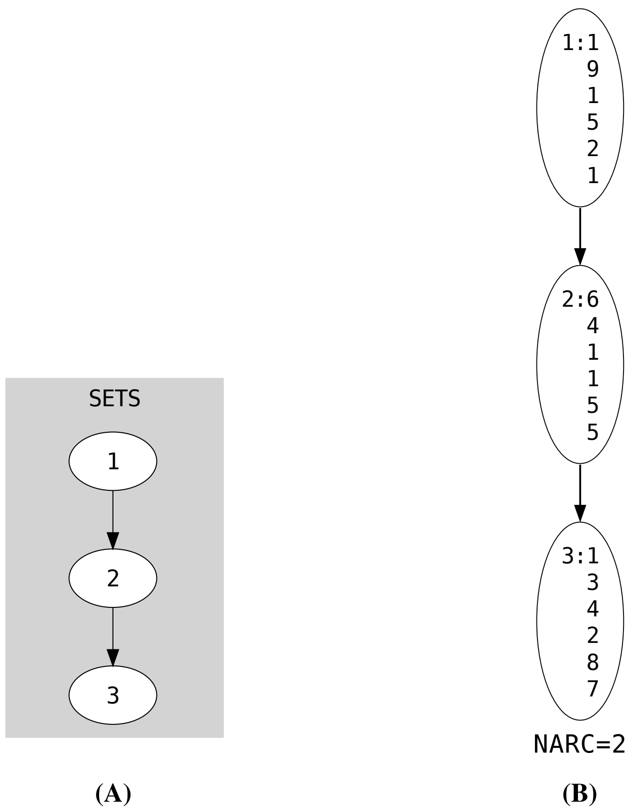


Figure 5.490: Initial and final graph of the K_SAME_MODULO constraint

5.213 K_SAME_PARTITION

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from SAME_PARTITION and from K_SAME.

Constraint K_SAME_PARTITION(SETS, PARTITIONS)

Types

VARIABLES : collection(var−dvar)
VALUES : collection(val−int)

Arguments

SETS : collection(set − VARIABLES)
PARTITIONS : collection(p − VALUES)

Restrictions

required(VARIABLES, var)
$|\texttt{VARIABLES}| \geq 1$
$|\texttt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)
required(SETS, set)
$|\texttt{SETS}| > 1$
same_size(SETS, set)
required(PARTITIONS, p)
$|\texttt{PARTITIONS}| \geq 2$

Purpose Given a collection of |SETS| sets, each containing the same number of domain variables, the K_SAME_PARTITION constraint forces a SAME_PARTITION constraint between each pair of consecutive sets.

Example

$$\left(\begin{array}{c}\left\langle\begin{array}{l}\texttt{set}-\langle 1,2,6,3,1,2\rangle,\\ \texttt{set}-\langle 6,6,2,3,1,3\rangle,\\ \texttt{set}-\langle 2,2,2,1,1,1\rangle\end{array}\right\rangle,\\ \langle \texttt{p}-\langle 1,3\rangle,\texttt{p}-\langle 4\rangle,\texttt{p}-\langle 2,6\rangle\rangle\end{array}\right)$$

The first argument SETS of the K_SAME_PARTITION constraint corresponds to 3 collections of variables, while the second argument PARTITIONS defines the 3 sets of values $\{1,3\}$, $\{4\}$ and $\{2,6\}$. The K_SAME_PARTITION constraint holds since:

- The first and second collections of variables are assigned 3 values in the $\{1,3\}$ as well as 3 values in $\{2,6\}$.
- The second and third collections of variables are also assigned 3 values in the $\{1,3\}$ as well as 3 values in $\{2,6\}$.

Typical $|\texttt{VARIABLES}| > 1$

Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of SETS.set.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Arg. properties

Contractible wrt. SETS.

See also

common keyword: K_SAME *(system of constraints)*.

implies: K_USED_BY_PARTITION.

part of system of constraints: SAME_PARTITION.

used in graph description: SAME_PARTITION.

Keywords

characteristic of a constraint: sort based reformulation, partition.

combinatorial object: permutation.

constraint type: system of constraints, decomposition.

Arc input(s)	SETS
Arc generator	$PATH \mapsto$ collection(set1, set2)
Arc arity	2
Arc constraint(s)	SAME_PARTITION(set1.set, set2.set, PARTITIONS)
Graph property(ies)	**NARC** $= \|$SETS$\| - 1$

Graph model Parts (A) and (B) of Figure 5.491 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a SAME_PARTITION constraint.

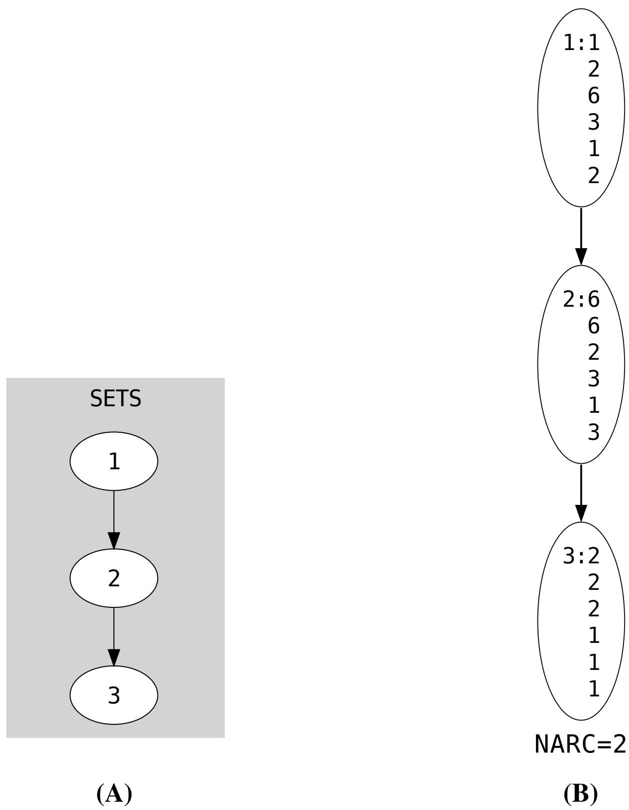


Figure 5.491: Initial and final graph of the K_SAME_PARTITION constraint

5.214 K_USED_BY

DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY

Constraint K_USED_BY(SETS)

Type VARIABLES : collection(var−dvar)

Argument SETS : collection(set − VARIABLES)

Restrictions
required(VARIABLES, var)
$|\text{VARIABLES}| \geq 1$
required(SETS, set)
$|\text{SETS}| > 1$
non_increasing_size(SETS, set)

Purpose Given |SETS| sets of domain variables, the K_USED_BY constraint forces a USED_BY constraint between each pair of consecutive sets.

Example

$$\left(\left\langle\begin{array}{l}\text{set} - \langle 1,9,1,5,2,1\rangle,\\ \text{set} - \langle 9,1,1,1,2,5\rangle,\\ \text{set} - \langle 1,1,2,5\rangle\end{array}\right\rangle\right)$$

The K_USED_BY constraint holds since:

- The multiset of values $\{\{1,1,1,2,5,9\}\}$ associated with the second collection of variables is included into the multiset $\{\{1,1,1,2,5,9\}\}$ associated with the first collection of variables.
- The multiset of values $\{\{1,1,2,5\}\}$ associated with the third collection of variables is included into the multiset $\{\{1,1,1,2,5,9\}\}$ associated with the second collection of variables.

Typical $|\text{VARIABLES}| > 1$

Symmetries
- Items of SETS are permutable.
- Items of SETS.set are permutable.
- All occurrences of two distinct values of SETS.set.var can be swapped; all occurrences of a value of SETS.set.var can be renamed to any unused value.

Arg. properties Contractible wrt. SETS.

Remark Similarly to the K_SAME constraint [160], finding out whether the K_USED_BY constraint has a solution or not is NP-hard when we have more than one USED_BY constraint.

See also **common keyword:** K_USED_BY_INTERVAL, K_USED_BY_MODULO, K_USED_BY_PARTITION *(system of constraints)*.

implied by: K_SAME.

part of system of constraints: USED_BY.

used in graph description: USED_BY.

Keywords **characteristic of a constraint:** sort based reformulation.

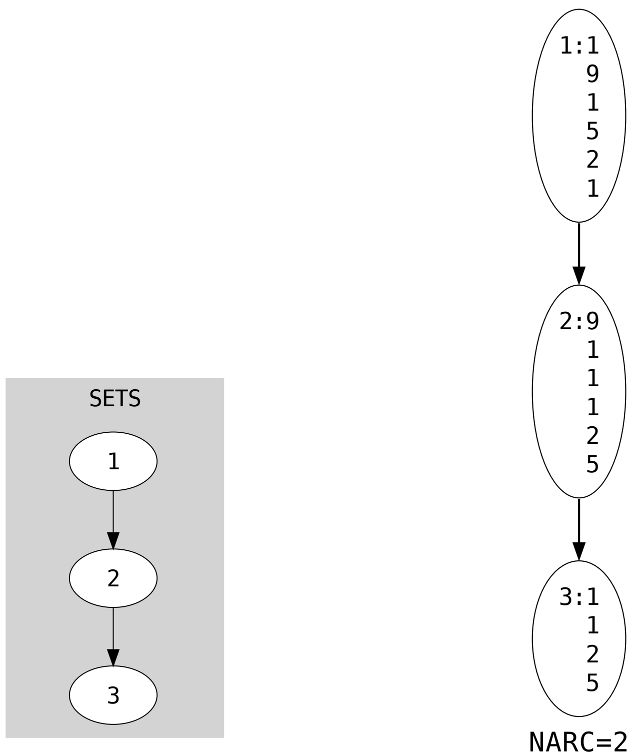
combinatorial object: multiset.

constraint type: system of constraints, decomposition.

modelling: inclusion.

Arc input(s)	SETS
Arc generator	$PATH \mapsto$ collection(set1, set2)
Arc arity	2
Arc constraint(s)	USED_BY(set1.set, set2.set)
Graph property(ies)	**NARC** $= \|$SETS$\| - 1$

Graph model Parts (A) and (B) of Figure 5.492 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a USED_BY constraint.



(A) **(B)**

Figure 5.492: Initial and final graph of the K_USED_BY constraint

5.215 K_USED_BY_INTERVAL

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY_INTERVAL and from K_USED_BY.

Constraint K_USED_BY_INTERVAL(SETS, SIZE_INTERVAL)

Type VARIABLES : collection(var−dvar)

Arguments

SETS : collection(set − VARIABLES)
SIZE_INTERVAL : int

Restrictions

required(VARIABLES, var)
$|\texttt{VARIABLES}| \geq 1$
required(SETS, set)
$|\texttt{SETS}| > 1$
non_increasing_size(SETS, set)
$\texttt{SIZE\_INTERVAL} > 0$

Purpose Given |SETS| sets of domain variables, the K_USED_BY_INTERVAL constraint forces a USED_BY_INTERVAL constraint between each pair of consecutive sets.

Example

$(\langle \texttt{set} - \langle 1, 1, 1, 8, 6, 2\rangle, \texttt{set} - \langle 1, 0, 7, 7\rangle, \texttt{set} - \langle 1, 2\rangle\rangle, 3)$

In the example, the second argument $\texttt{SIZE\_INTERVAL} = 3$ defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Consequently, the K_USED_BY_INTERVAL constraint holds since:

- The first collection of variables is assigned 4 values in the interval $[0, 2]$ as well as 2 values in the interval $[6, 8]$, while the second collection of variables is assigned no more values in the previous two intervals.
- The second collection of variables is assigned 2 values in the interval $[0, 2]$ as well as 2 values in the interval $[6, 8]$, while the third collection of variables is assigned no more values in the previous two intervals.

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{SIZE\_INTERVAL} > 0$

Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- An occurrence of a value of SETS.set.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties Contractible wrt. SETS.

See also

common keyword: K_USED_BY *(system of constraints)*.

implied by: K_SAME_INTERVAL.

part of system of constraints: USED_BY_INTERVAL.

used in graph description: USED_BY_INTERVAL.

Keywords

characteristic of a constraint: sort based reformulation.

constraint type: system of constraints, decomposition.

modelling: inclusion, interval.

Arc input(s)	SETS
Arc generator	$PATH$ ↦ collection(set1, set2)
Arc arity	2
Arc constraint(s)	USED_BY_INTERVAL(set1.set, set2.set, SIZE_INTERVAL)
Graph property(ies)	**NARC** = |SETS| − 1

Graph model Parts (A) and (B) of Figure 5.493 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a USED_BY_INTERVAL constraint.

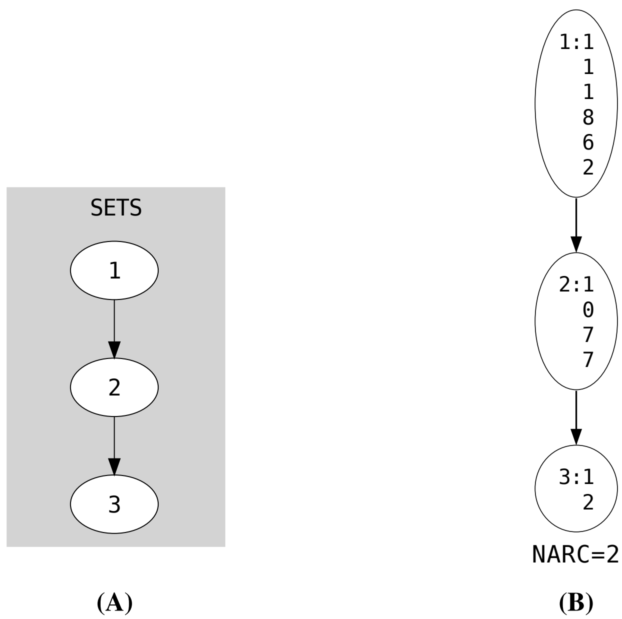


Figure 5.493: Initial and final graph of the K_USED_BY_INTERVAL constraint

5.216 K_USED_BY_MODULO

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY_MODULO and from K_USED_BY.

Constraint K_USED_BY_MODULO(SETS, M)

Type VARIABLES : collection(var−dvar)

Arguments

SETS : collection(set − VARIABLES)
M : int

Restrictions

required(VARIABLES, var)
$|$VARIABLES$| \geq 1$
required(SETS, set)
$|$SETS$| > 1$
non_increasing_size(SETS, set)
$\mathtt{M} > 0$

Purpose Given |SETS| sets of domain variables, the K_USED_BY_MODULO constraint forces a USED_BY_MODULO constraint between each pair of consecutive sets.

Example $(\langle \mathtt{set} - \langle 1, 9, 4, 5, 2, 1\rangle, \mathtt{set} - \langle 7, 1, 2, 5\rangle, \mathtt{set} - \langle 1, 1\rangle\rangle, 3)$

The K_USED_BY_MODULO constraint holds since:

- The first collection of variables is assigned 1 value in $\{0, 3, \ldots, 3 \cdot k\}$, 3 values in $\{1, 4, \ldots, 1+3\cdot k\}$ and 2 values in $\{2, 5, \ldots, 2+3\cdot k\}$, while the second collection of variables is assigned no more values in the previous three sets of values.
- The second collection of variables is assigned 2 values in $\{0, 3, \ldots, 3 \cdot k\}$ and 2 values in $\{2, 5, \ldots, 2 + 3 \cdot k\}$, while the third collection of variables is assigned no more values in the previous three sets of values.

Typical

$|$VARIABLES$| > 1$
$\mathtt{M} > 1$

Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- An occurrence of a value u of SETS.set.var can be replaced by any other value v such that v is congruent to u modulo M.

Arg. properties Contractible wrt. SETS.

See also

common keyword: K_USED_BY *(system of constraints)*.

implied by: K_SAME_MODULO.

part of system of constraints: USED_BY_MODULO.

used in graph description: USED_BY_MODULO.

Keywords

characteristic of a constraint: modulo, sort based reformulation.

constraint type: system of constraints, decomposition.

modelling: inclusion.

Arc input(s)	SETS
Arc generator	$PATH \mapsto$ collection(set1, set2)
Arc arity	2
Arc constraint(s)	USED_BY_MODULO(set1.set, set2.set, M)
Graph property(ies)	**NARC** $= \vert$SETS$\vert - 1$

Graph model Parts (A) and (B) of Figure 5.494 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a USED_BY_MODULO constraint.

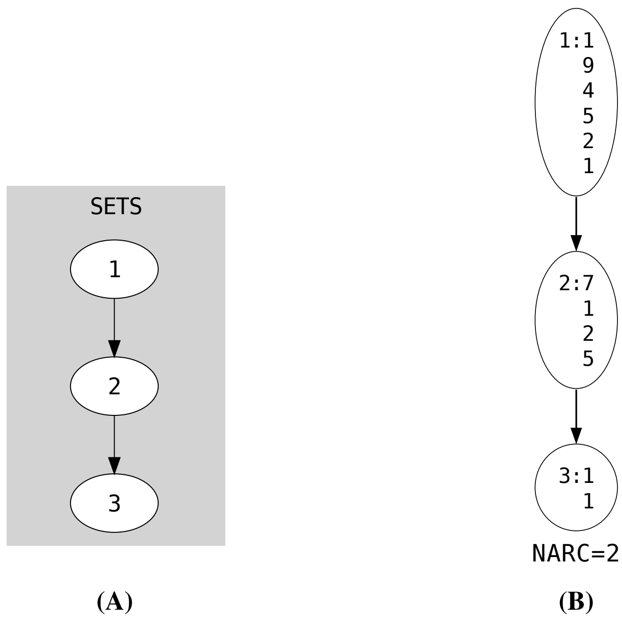


Figure 5.494: Initial and final graph of the K_USED_BY_MODULO constraint

5.217 K_USED_BY_PARTITION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY_PARTITION and from K_USED_BY.

Constraint K_USED_BY_PARTITION(SETS, PARTITIONS)

Types

VARIABLES : collection(var−dvar)
VALUES : collection(val−int)

Arguments

SETS : collection(set − VARIABLES)
PARTITIONS : collection(p − VALUES)

Restrictions

required(VARIABLES, var)
$|\text{VARIABLES}| \geq 1$
$|\text{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)
required(SETS, set)
$|\text{SETS}| > 1$
non_increasing_size(SETS, set)
required(PARTITIONS, p)
$|\text{PARTITIONS}| \geq 2$

Purpose Given |SETS| sets of domain variables, the K_USED_BY_PARTITION constraint forces a USED_BY_PARTITION constraint between each pair of consecutive sets.

Example

$$\left(\begin{array}{l}\langle \texttt{set} - \langle 1,9,1,6,2,3\rangle, \texttt{set} - \langle 1,3,6,6\rangle, \texttt{set} - \langle 2,2\rangle\rangle, \\ \langle \texttt{p} - \langle 1,3\rangle, \texttt{p} - \langle 4\rangle, \texttt{p} - \langle 2,6\rangle\rangle\end{array}\right)$$

The K_USED_BY_PARTITION constraint holds since:

- The first collection of variables is assigned 3 values in $\{1,3\}$, 0 value in $\{4\}$ and 2 values in $\{2,6\}$, while the second collection of variables is assigned no more values in the previous three sets of values.
- The second collection of variables is assigned 2 values in $\{1,3\}$, 0 value in $\{4\}$ and 2 values in $\{2,6\}$, while the third collection of variables is assigned no more values in the previous three sets of values.

Typical $|\text{VARIABLES}| > 1$

Symmetries

- Items of SETS are permutable.
- Items of SETS.set are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of SETS.set.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Arg. properties

Contractible wrt. SETS.

See also

common keyword: K_USED_BY *(system of constraints)*.

implied by: K_SAME_PARTITION.

part of system of constraints: USED_BY_PARTITION.

used in graph description: USED_BY_PARTITION.

Keywords

characteristic of a constraint: partition, sort based reformulation.

constraint type: system of constraints, decomposition.

Arc input(s)	SETS
Arc generator	*PATH* ↦ collection(set1, set2)
Arc arity	2
Arc constraint(s)	USED_BY_PARTITION(set1.set, set2.set, PARTITIONS)
Graph property(ies)	**NARC** $= \lvert$SETS$\rvert - 1$

Graph model Parts (A) and (B) of Figure 5.495 respectively show the initial and final graph associated with the **Example** slot. To each vertex corresponds a collection of variables, while to each arc corresponds a USED_BY_PARTITION constraint.

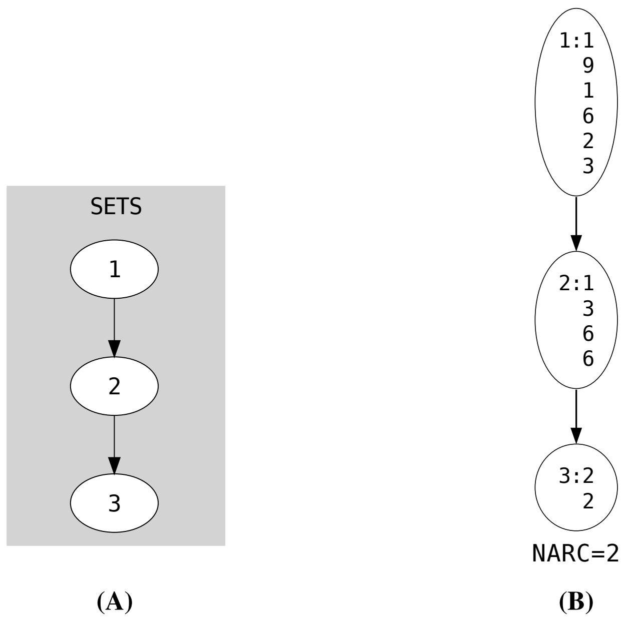


Figure 5.495: Initial and final graph of the K_USED_BY_PARTITION constraint

5.218 LENGTH_FIRST_SEQUENCE

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Inspired by STRETCH_PATH

Constraint LENGTH_FIRST_SEQUENCE(LEN, VARIABLES)

Synonym LENGTH_FIRST_STRETCH.

Arguments

LEN	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

LEN ≥ 0
LEN $\leq$ |VARIABLES|
required(VARIABLES, var)

Purpose LEN is the length of the maximum sequence of variables that take the same value that contains the first variable of the collection VARIABLES (or 0 if the collection is empty).

Example

$(3, \langle 4, 4, 4, 5, 5, 4\rangle)$
$(6, \langle 4, 4, 4, 4, 4, 4\rangle)$
$(5, \langle 4, 4, 4, 4, 4, 1\rangle)$

4 4 4 5 5 4
4 4 4 4 4 4
4 4 4 4 4 1

The first LENGTH_FIRST_SEQUENCE constraint holds since the sequence associated with the first value of the collection VARIABLES $= \langle 4, 4, 4, 5, 5, 4\rangle$ spans over three consecutive variables.

Typical

LEN $<$ |VARIABLES|
|VARIABLES| > 1

Typical model nval(VARIABLES.var) > 2

Symmetry All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Functional dependency: LEN determined by VARIABLES.

Reformulation Without loss of generality let assume that the collection VARIABLES $= \langle V_1, V_2, \ldots, V_n\rangle$ has more than one variable. By introducing $2 \cdot n - 1$ 0-1 variables, the LENGTH_FIRST_SEQUENCE(LEN, VARIABLES) constraint can be expressed in term of $2 \cdot n - 1$ reified constraints and one arithmetic constraint (i.e., a SUM_CTR constraint). We first introduce $n-1$ variables that are respectively set to 1 if and only if two given consecutive variables of the collection VARIABLES are equal:

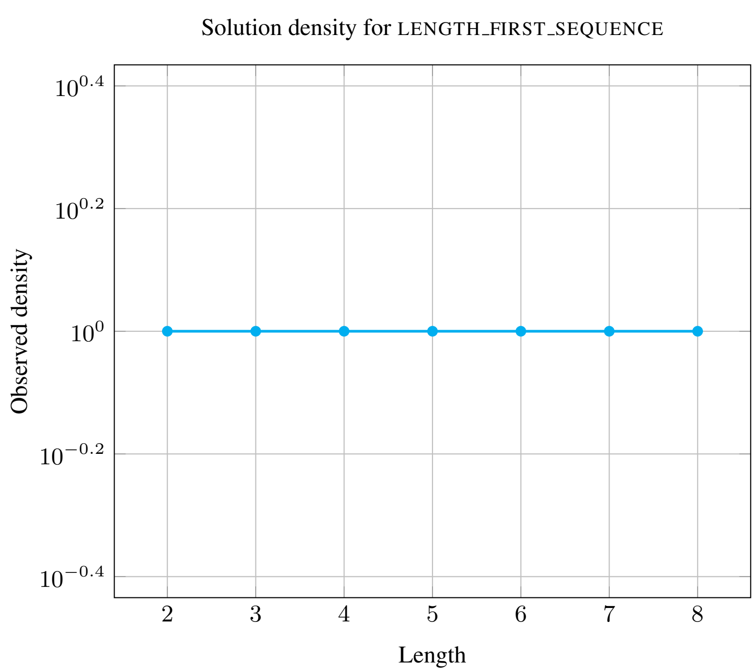
$B_{1,2} \quad \Leftrightarrow V_1 = V_2,$
$B_{2,3} \quad \Leftrightarrow V_2 = V_3,$

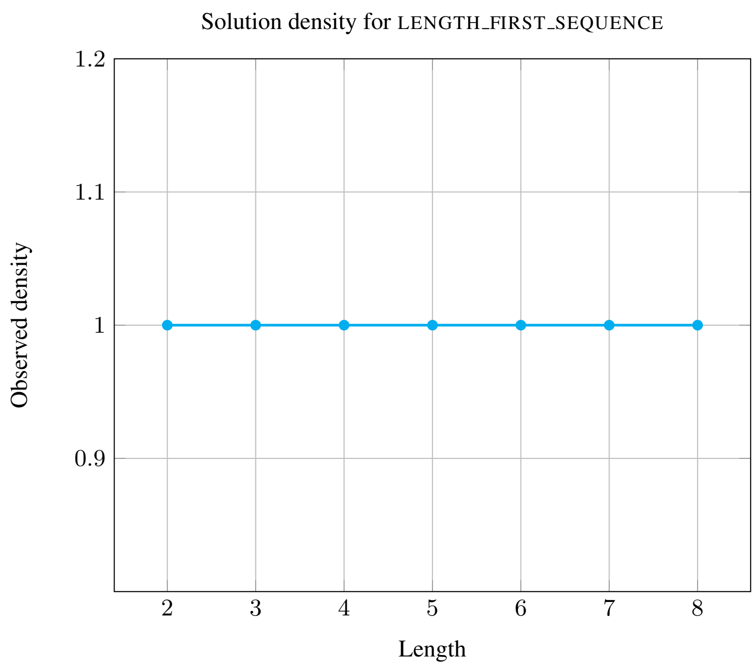
....
$B_{n-1,n} \Leftrightarrow V_{n-1} = V_n$.
We then introduce n variables $A_1, A_2, \ldots, A_n$ that are respectively associated to the different sliding sequences starting on the first variable of the sequence $V_1\ V_2\ \ldots\ V_n$. Variable A_i is set to 1 if and only if $V_1 = V_2 = \cdots = V_i$:
$A_1 = 1$,
$A_2 \Leftrightarrow B_{1,2} \quad \wedge A_1$,
$A_3 \Leftrightarrow B_{2,3} \quad \wedge A_2$,
....
$A_n \Leftrightarrow B_{n-1,n} \wedge A_{n-1}$.
Finally we state the following arithmetic constraint:
`LEN` $= A_1 + A_2 + \cdots + A_n$.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

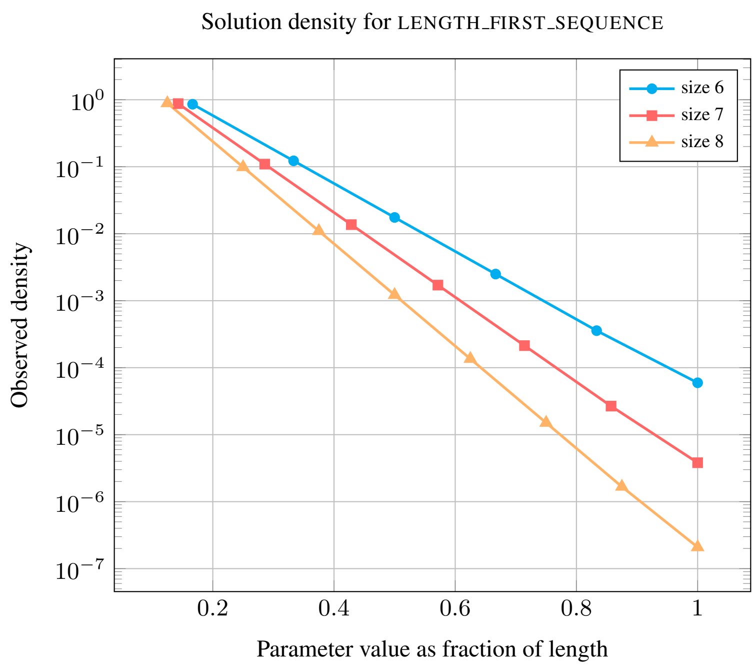
Number of solutions for LENGTH_FIRST_SEQUENCE: domains $0..n$

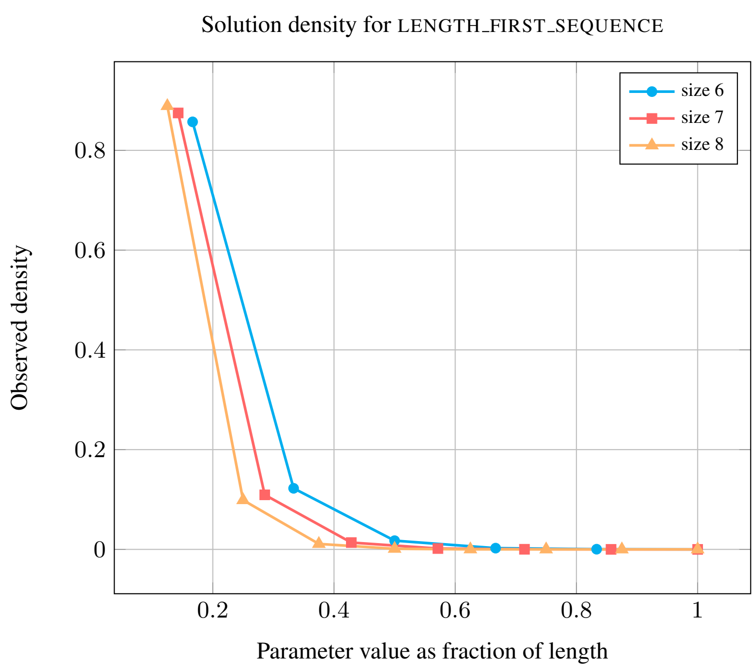




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	6	48	500	6480	100842	1835008	38263752
	2	3	12	100	1080	14406	229376	4251528
	3	-	4	20	180	2058	28672	472392
	4	-	-	5	30	294	3584	52488
	5	-	-	-	6	42	448	5832
	6	-	-	-	-	7	56	648
	7	-	-	-	-	-	8	72
	8	-	-	-	-	-	-	9

Solution count for LENGTH_FIRST_SEQUENCE: domains $0..n$





See also **common keyword:** LENGTH_LAST_SEQUENCE (*counting constraint*, *sequence*).

Keywords **characteristic of a constraint:** automaton, automaton with counters.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

constraint type: value constraint, counting constraint.

filtering: glue matrix.

modelling: functional dependency.

Automaton Figure 5.496 depicts the automaton associated with the LENGTH_FIRST_SEQUENCE constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i$.

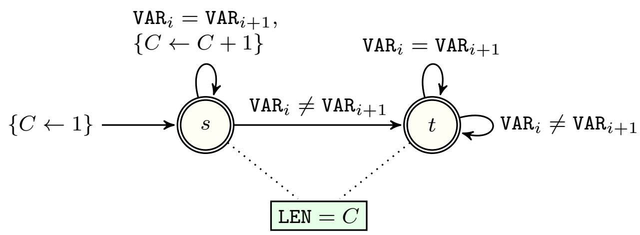


Figure 5.496: Automaton of the LENGTH_FIRST_SEQUENCE constraint when $|\mathtt{VARIABLES}| \geq 2$

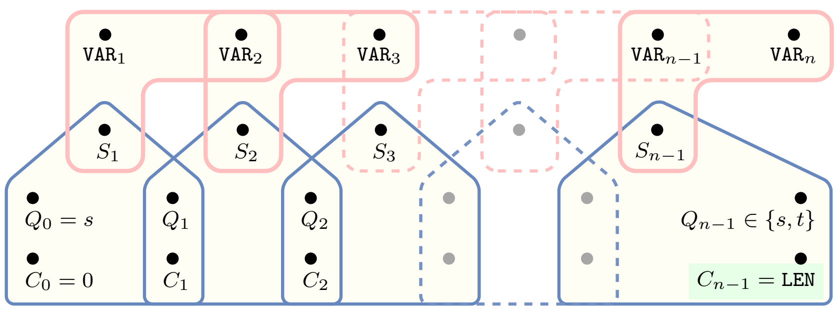
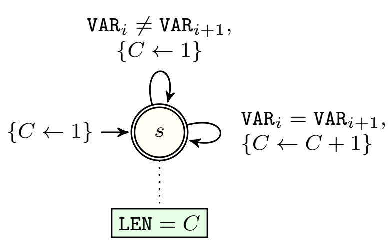


Figure 5.497: Hypergraph of the reformulation corresponding to the automaton of the LENGTH_FIRST_SEQUENCE constraint



	s
s	$\overrightarrow{C} + \overleftarrow{C} - 1$
t	$\overrightarrow{C}$

Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counters values C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

Figure 5.498: Automaton of the reverse of the LENGTH_FIRST_SEQUENCE constraint (i.e., the LENGTH_LAST_SEQUENCE constraint) when $|\mathtt{VARIABLES}| \geq 2$ and corresponding glue matrix between LENGTH_FIRST_SEQUENCE and its reverse LENGTH_LAST_SEQUENCE

5.219 LENGTH_LAST_SEQUENCE

DESCRIPTION LINKS AUTOMATON

Origin Inspired by STRETCH_PATH

Constraint LENGTH_LAST_SEQUENCE(LEN, VARIABLES)

Synonym LENGTH_LAST_STRETCH.

Arguments

LEN : dvar
VARIABLES : collection(var−dvar)

Restrictions

LEN ≥ 0
LEN $\leq |$VARIABLES$|$
required(VARIABLES, var)

Purpose LEN is the length of the maximum sequence of variables that take the same value that contains the last variable of the collection VARIABLES (or 0 if the collection is empty).

Example

$(1, \langle 4, 4, 4, 5, 5, 4\rangle)$
$(6, \langle 4, 4, 4, 4, 4, 4\rangle)$
$(5, \langle 2, 4, 4, 4, 4, 4\rangle)$

4 4 4 5 5 4
4 4 4 4 4 4
2 4 4 4 4 4

The first LENGTH_LAST_SEQUENCE constraint holds since the sequence associated with the last value of the collection VARIABLES $= \langle 4, 4, 4, 5, 5, 4\rangle$ spans over a single variable.

Typical

LEN $<|$VARIABLES$|$
$|$VARIABLES$| > 1$

Typical model nval(VARIABLES.var) > 2

Symmetry All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Functional dependency: LEN determined by VARIABLES.

Reformulation Without loss of generality let assume that the collection VARIABLES $= \langle V_1, V_2, \ldots, V_n\rangle$ has more than one variable. By introducing $2 \cdot n - 1$ 0-1 variables, the LENGTH_LAST_SEQUENCE(LEN, VARIABLES) constraint can be expressed in term of $2 \cdot n - 1$ reified constraints and one arithmetic constraint (i.e., a SUM_CTR constraint). We first introduce $n-1$ variables that are respectively set to 1 if and only if two given consecutive variables of the collection VARIABLES are equal:

$B_{n-1,n} \Leftrightarrow V_{n-1} = V_n,$
$B_{n-2,n-1} \Leftrightarrow V_{n-2} = V_{n-1},$

....
$B_{1,2} \quad \Leftrightarrow V_1 = V_2$.

We then introduce n variables $A_n, A_{n-1}, \ldots, A_1$ that are respectively associated to the different sliding sequences ending on the last variable of the sequence $V_1\ V_2\ \ldots\ V_n$. Variable A_i is set to 1 if and only if $V_n = V_{n-1} = \cdots = V_i$:

$A_n = 1$,
$A_{n-1} \Leftrightarrow B_{n-1,n} \quad \wedge A_n$,
$A_{n-2} \Leftrightarrow B_{n-2,n-1} \wedge A_{n-1}$,
....
$A_1 \quad \Leftrightarrow B_{1,2} \quad \wedge A_2$.

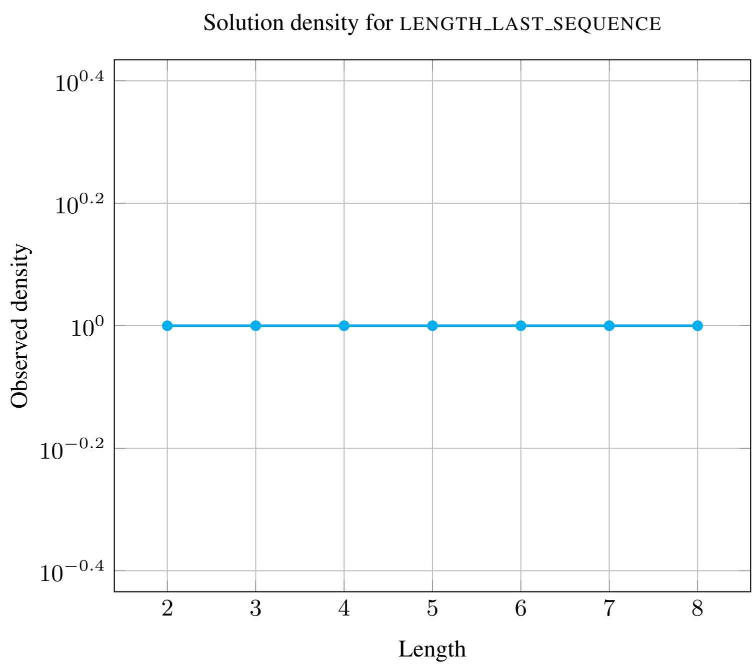
Finally we state the following arithmetic constraint:

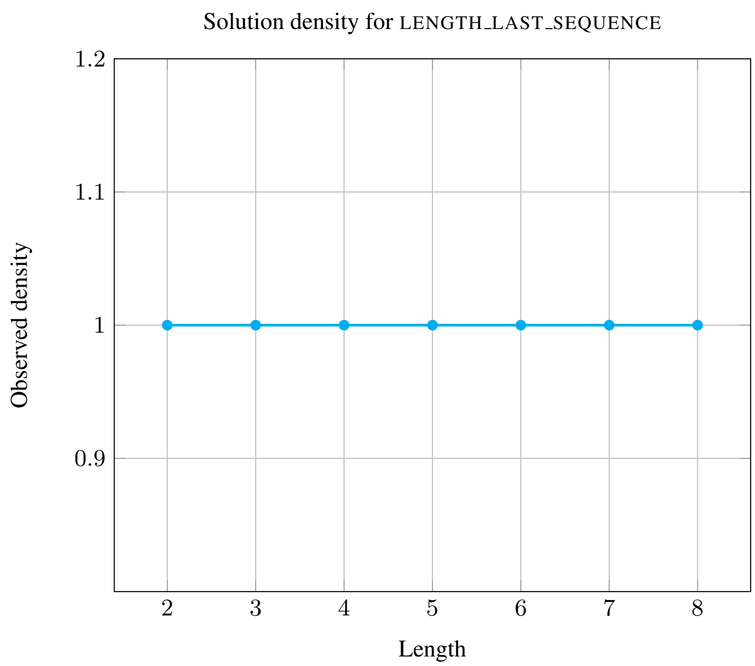
$\texttt{LEN} = A_n + A_{n-1} + \cdots + A_1$.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

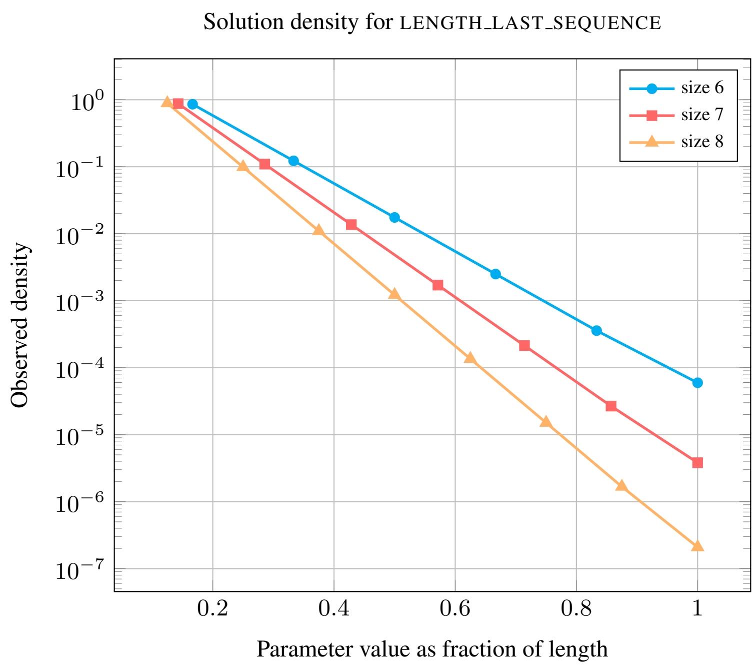
Number of solutions for LENGTH_LAST_SEQUENCE: domains $0..n$

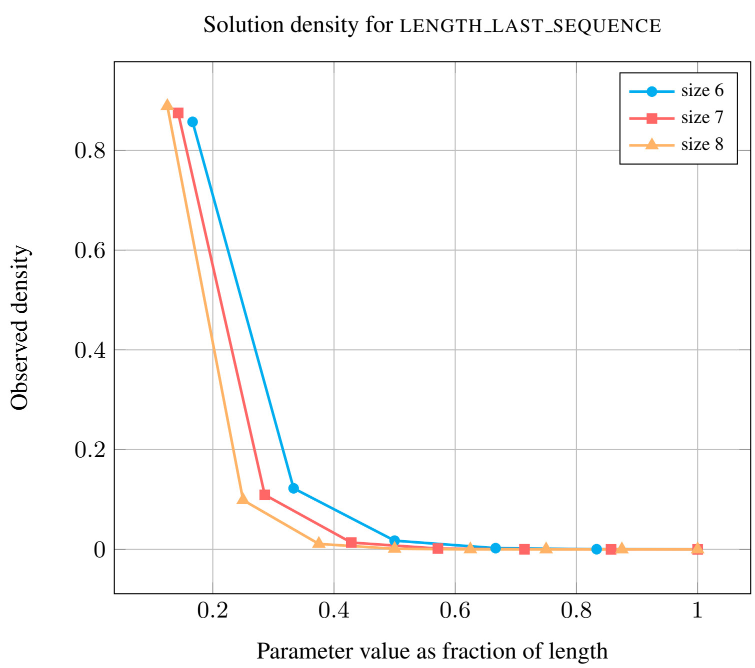




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	6	48	500	6480	100842	1835008	38263752
	2	3	12	100	1080	14406	229376	4251528
	3	-	4	20	180	2058	28672	472392
	4	-	-	5	30	294	3584	52488
	5	-	-	-	6	42	448	5832
	6	-	-	-	-	7	56	648
	7	-	-	-	-	-	8	72
	8	-	-	-	-	-	-	9

Solution count for LENGTH_LAST_SEQUENCE: domains $0..n$





See also **common keyword:** LENGTH_FIRST_SEQUENCE *(counting constraint,sequence)*.

Keywords **characteristic of a constraint:** automaton, automaton with counters.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

constraint type: value constraint, counting constraint.

filtering: glue matrix.

modelling: functional dependency.

Automaton Figure 5.499 depicts the automaton associated with the LENGTH_LAST_SEQUENCE constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i$.

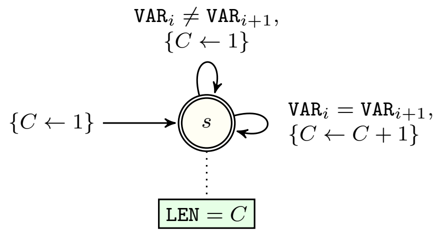


Figure 5.499: Automaton of the LENGTH_LAST_SEQUENCE constraint when $|\mathtt{VARIABLES}| \geq 2$

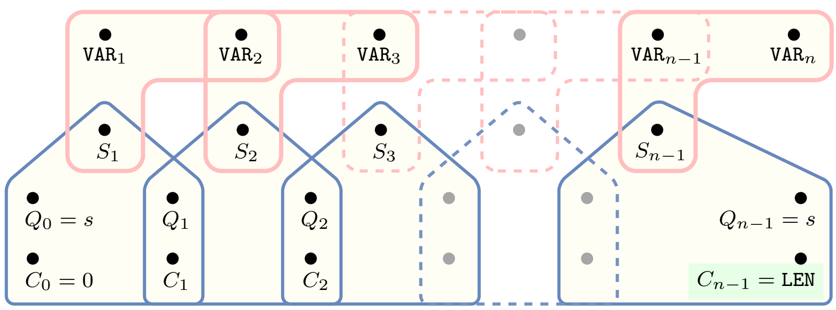
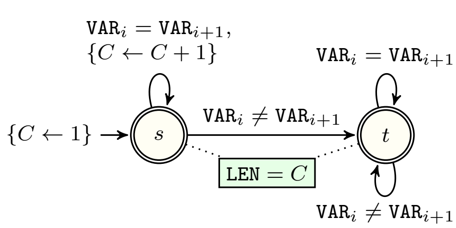


Figure 5.500: Hypergraph of the reformulation corresponding to the automaton of the LENGTH_LAST_SEQUENCE constraint



	s	t
s	$\overrightarrow{C} + \overleftarrow{C} - 1$	$\overrightarrow{C}$

Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counters values C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

Figure 5.501: Automaton of the reverse of the LENGTH_LAST_SEQUENCE constraint (i.e., the LENGTH_FIRST_SEQUENCE constraint) when $|\mathtt{VARIABLES}| \geq 2$ and corresponding glue matrix between LENGTH_LAST_SEQUENCE and its reverse LENGTH_FIRST_SEQUENCE

5.220 LEQ

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint LEQ(VAR1, VAR2)

Synonyms REL, XLTEQY.

Arguments

VAR1	:	dvar
VAR2	:	dvar

Purpose Enforce the fact that the first variable is less than or equal to the second variable.

Example $(1, 8)$

The LEQ constraint holds since 1 is greater than or equal to 8.

Symmetries

- VAR1 can be replaced by any value $\leq$ VAR2.
- VAR2 can be replaced by any value $\geq$ VAR1.

Systems LEQ in **Choco**, REL in **Gecode**, XLTEQY in **JaCoP**, #=< in **SICStus**.

See also

common keyword: NEQ *(binary constraint,arithmetic constraint)*.

generalisation: LEQ_CST *(*`constant` *added)*.

implied by: EQ, LT.

implies (if swap arguments): GEQ.

negation: GT.

Keywords

constraint arguments: binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.221 LEQ_CST

► ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic.

Constraint LEQ_CST(VAR1, VAR2, CST2)

Arguments

VAR1	:	dvar
VAR2	:	dvar
CST2	:	int

Purpose Enforce the fact that the first variable is less than or equal to the sum of the second variable and the constant.

Example $(5, 2, 4)$

The LEQ_CST constraint holds since 5 is less than or equal to $2 + 4$.

Typical

$\texttt{CST2} \neq 0$
$\texttt{VAR1} < \texttt{VAR2} + \texttt{CST2}$

Symmetries

- Arguments are permutable w.r.t. permutation (VAR1) (VAR2, CST2).
- VAR1 can be replaced by any value $\leq$ VAR2 + CST2.
- VAR2 can be replaced by any value $\geq$ VAR1 − CST2.
- CST2 can be replaced by any value $\geq$ VAR1 − VAR2.

See also

common keyword: GEQ_CST *(binary constraint,arithmetic constraint)*.

implied by: DISTANCE, EQ_CST.

specialisation: LEQ *(constant set to 0)*.

Keywords

constraint arguments: binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

modelling exercises: metro.

5.222 LEX2

► ▷ ◁ ◄ **DESCRIPTION** **LINKS**

Origin [179]

Constraint LEX2(MATRIX)

Synonyms DOUBLE_LEX, ROW_AND_COLUMN_LEX.

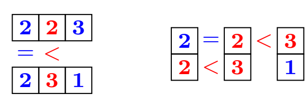
Type VECTOR : collection(var−dvar)

Argument MATRIX : collection(vec − VECTOR)

Restrictions
|VECTOR| ≥ 1
required(VECTOR, var)
required(MATRIX, vec)
same_size(MATRIX, vec)

Purpose Given a matrix of domain variables, enforces that both adjacent rows, and adjacent columns are lexicographically ordered (adjacent rows and adjacent columns can be equal).

Example $(\langle \texttt{vec} - \langle 2,2,3\rangle, \texttt{vec} - \langle 2,3,1\rangle\rangle)$



The LEX2 constraint holds since:

- The first row $\langle 2,2,3\rangle$ is lexicographically less than or equal to the second row $\langle 2,3,1\rangle$.
- The first column $\langle 2,2\rangle$ is lexicographically less than or equal to the second column $\langle 2,3\rangle$.
- The second column $\langle 2,3\rangle$ is lexicographically less than or equal to the third column $\langle 3,1\rangle$.

Typical
|VECTOR| > 1
|MATRIX| > 1

Symmetry One and the same constant can be added to the var attribute of all items of MATRIX.vec.

Usage A *symmetry-breaking* constraint.

Remark The idea of this *symmetry-breaking* constraint can already be found in the following articles of A. Lubiw [278, 279].

In block designs you sometimes want repeated blocks, so using the non-strict order would be required in this case.

Reformulation The LEX2 constraint can be expressed as a conjunction of two LEX_CHAIN_LESSEQ constraints: A first LEX_CHAIN_LESSEQ constraint on the MATRIX argument and a second LEX_CHAIN_LESSEQ constraint on the transpose of the MATRIX argument.

Systems LEX2 in **MiniZinc**.

See also **common keyword:** ALLPERM, LEX_LESSEQ *(matrix symmetry,lexicographic order)*.

implied by: STRICT_LEX2.

implies: LEX_CHAIN_LESSEQ.

part of system of constraints: LEX_CHAIN_LESSEQ.

Keywords **constraint type:** predefined constraint, system of constraints, order constraint.

modelling: matrix, matrix model.

symmetry: symmetry, matrix symmetry, lexicographic order.

5.223 LEX_ALLDIFFERENT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin J. Pearson

Constraint LEX_ALLDIFFERENT(VECTORS)

Synonyms LEX_ALLDIFF, LEX_ALLDISTINCT, ALLDIFF_ON_TUPLES, ALLDIFFERENT_ON_TUPLES, ALLDISTINCT_ON_TUPLES.

Type VECTOR : collection(var−dvar)

Argument VECTORS : collection(vec − VECTOR)

Restrictions
|VECTOR| ≥ 1
required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose All the vectors of the collection VECTORS are distinct. Two vectors $(u_1, u_2, \ldots, u_n)$ and $(v_1, v_2, \ldots, v_n)$ are distinct if and only if there exists $i \in [1, n]$ such that $u_i \neq v_i$.

Example $(\langle \texttt{vec} - \langle 5,2,3\rangle, \texttt{vec} - \langle 5,2,6\rangle, \texttt{vec} - \langle 5,3,3\rangle\rangle)$

The LEX_ALLDIFFERENT constraint holds since:

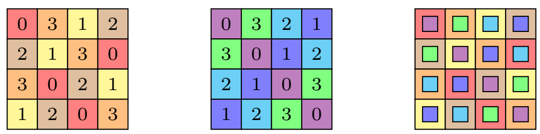
- The first vector $\langle 5,2,3\rangle$ and the second vector $\langle 5,2,6\rangle$ of the VECTORS collection differ in their third components (i.e., $3 \neq 6$).
- The first vector $\langle 5,2,3\rangle$ and the third vector $\langle 5,3,3\rangle$ of the VECTORS collection differ in their second components (i.e., $2 \neq 3$).
- The second vector $\langle 5,2,6\rangle$ and the third vector $\langle 5,3,3\rangle$ of the VECTORS collection differ in their second and third components (i.e., $2 \neq 3$ and $6 \neq 3$).

Typical
|VECTOR| > 1
|VECTORS| > 1

Symmetries
- Items of VECTORS are permutable.
- Items of VECTORS.vec are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values of VECTORS.vec can be swapped; all occurrences of a tuple of values of VECTORS.vec can be renamed to any unused tuple of values.

Arg. properties
- Contractible wrt. VECTORS.
- Extensible wrt. VECTORS.vec (*add items at same position*).

Usage When the vectors have two components, the LEX_ALLDIFFERENT constraint allows to directly enforce difference constraints between pairs of variables. Such difference constraints occur, for example, in block design problems (e.g., Steiner triples, Kirkman schoolgirls, orthogonal Latin squares problems). However, in all these problems a same variable may occur in more than one pair of variables. Consequently, arc-consistency is not achieved any more by the filtering algorithm described in [346]. Figure 5.502 illustrates the use of the LEX_ALLDIFFERENT constraint in the context of the orthogonal Latin squares problem, i.e. two Latin squares such that the pairs of corresponding cells are distinct.



(A) First Latin square (B) Second Latin square (C) Corresponding orthogonal Latin square

$$\left(\left\langle\begin{array}{llll}
\texttt{vec} - \langle 0,0\rangle, & \texttt{vec} - \langle 3,3\rangle, & \texttt{vec} - \langle 1,2\rangle, & \texttt{vec} - \langle 2,1\rangle, \\
\texttt{vec} - \langle 2,3\rangle, & \texttt{vec} - \langle 1,0\rangle, & \texttt{vec} - \langle 3,1\rangle, & \texttt{vec} - \langle 0,2\rangle, \\
\texttt{vec} - \langle 3,2\rangle, & \texttt{vec} - \langle 0,1\rangle, & \texttt{vec} - \langle 2,0\rangle, & \texttt{vec} - \langle 1,3\rangle, \\
\texttt{vec} - \langle 1,1\rangle, & \texttt{vec} - \langle 2,2\rangle, & \texttt{vec} - \langle 0,3\rangle, & \texttt{vec} - \langle 3,0\rangle
\end{array}\right\rangle\right)$$

(D) Argument of the LEX_ALLDIFFERENT constraint enforcing that the pairs of corresponding cells are all distinct

Figure 5.502: Illustrating the use of the LEX_ALLDIFFERENT constraint in the context of orthogonal Latin squares

Algorithm A filtering algorithm achieving arc-consistency for the LEX_ALLDIFFERENT constraint is proposed by C.-G. Quimper and T. Walsh in [346] and a longer version is available in [347] and in [348].

Reformulation The LEX_ALLDIFFERENT(VECTORS) constraint can be expressed as a clique of LEX_DIFFERENT constraints. By associating a n-dimensional box for which all sizes are equal to 1, one can also express the LEX_ALLDIFFERENT(VECTORS) constraint as a DIFFN or a GEOST constraint. Enforcing all the n-dimensional boxes to not overlap is equivalent as enforcing all the vectors to be distinct. In the context of the multidimensional sweep algorithm of the GEOST constraint [42], it makes more sense to make a complete sweep over the domain of each variable in order not to only restrict the minimum and maximum value of each variable.

See also **generalisation:** DIFFN (`vector` *replaced by* `orthotope`), GEOST (`vector` *replaced by* `object`).

implied by: ALL_INCOMPARABLE, LEX_CHAIN_GREATER, LEX_CHAIN_LESS.

implies: LEX_ALLDIFFERENT_EXCEPT_0.

part of system of constraints: LEX_DIFFERENT.

specialisation: ALLDIFFERENT (`vector` *replaced by* `variable`).

used in graph description: LEX_DIFFERENT.

Keywords

characteristic of a constraint: vector.

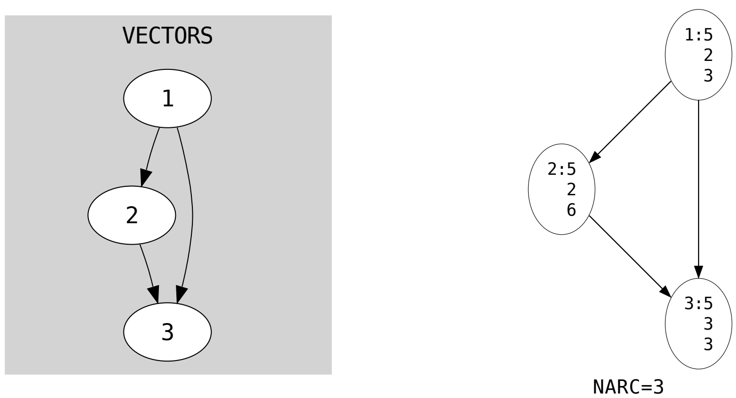
constraint type: system of constraints, decomposition.

filtering: bipartite matching, arc-consistency.

modelling: difference between pairs of variables.

Arc input(s)	VECTORS
Arc generator	$CLIQUE(<) \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_DIFFERENT(vectors1.vec, vectors2.vec)
Graph property(ies)	**NARC** $= |\texttt{VECTORS}| * (|\texttt{VECTORS}| - 1)/2$

Graph model Parts (A) and (B) of Figure 5.503 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.503: Initial and final graph of the LEX_ALLDIFFERENT constraint

Signature Since we use the $CLIQUE(<)$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to $|\texttt{VECTORS}| \cdot (|\texttt{VECTORS}| - 1)/2$. For this reason we can rewrite $\mathbf{NARC} = |\texttt{VECTORS}| \cdot (|\texttt{VECTORS}| - 1)/2$ to $\mathbf{NARC} \geq |\texttt{VECTORS}| \cdot (|\texttt{VECTORS}| - 1)/2$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.224 LEX_ALLDIFFERENT_EXCEPT_0

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin H. Simonis

Constraint LEX_ALLDIFFERENT_EXCEPT_0(VECTORS)

Synonyms LEX_ALLDIFF_EXCEPT_0, LEX_ALLDISTINCT_EXCEPT_0, ALLDIFF_ON_TUPLES_EXCEPT_0, ALLDIFFERENT_ON_TUPLES_EXCEPT_0, ALLDISTINCT_ON_TUPLES_EXCEPT_0.

Type VECTOR : collection(var−dvar)

Argument VECTORS : collection(vec − VECTOR)

Restrictions
$|\texttt{VECTOR}| \geq 1$
required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose All the non null vectors of the collection VECTORS are distinct. A vector is *null* if all its components are equal to zero. Two non null vectors $(u_1, u_2, \ldots, u_n)$ and $(v_1, v_2, \ldots, v_n)$ are *distinct* if and only if there exists $i \in [1, n]$ such that $u_i \neq v_i$.

Example

$$\left(\left\langle\begin{array}{l}\texttt{vec} - \langle 0,0,0\rangle,\\ \texttt{vec} - \langle 5,2,0\rangle,\\ \texttt{vec} - \langle 5,8,0\rangle,\\ \texttt{vec} - \langle 0,0,0\rangle\end{array}\right\rangle\right)$$

The LEX_ALLDIFFERENT_EXCEPT_0 constraint holds since its two non null vectors, i.e. the second and third vectors are distinct (the vectors $\langle 5, 2, 0\rangle$ and $\langle 5, 8, 0\rangle$ differ in their second components.

Typical
$|\texttt{VECTOR}| > 1$
$|\texttt{VECTORS}| > 1$

Arg. properties Contractible wrt. VECTORS.

See also **implied by:** LEX_ALLDIFFERENT.

Keywords **characteristic of a constraint:** vector, joker value.
constraint type: predefined constraint.
modelling: difference between pairs of variables.

5.225 LEX_BETWEEN

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin [104]

Constraint LEX_BETWEEN(LOWER_BOUND, VECTOR, UPPER_BOUND)

Synonym BETWEEN.

Arguments

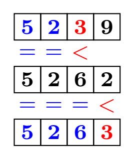
LOWER_BOUND	:	collection(var−int)
VECTOR	:	collection(var−dvar)
UPPER_BOUND	:	collection(var−int)

Restrictions

required(LOWER_BOUND, var)
required(VECTOR, var)
required(UPPER_BOUND, var)
$|$LOWER_BOUND$|$ = $|$VECTOR$|$
$|$UPPER_BOUND$|$ = $|$VECTOR$|$
LEX_LESSEQ(LOWER_BOUND, VECTOR)
LEX_LESSEQ(VECTOR, UPPER_BOUND)

Purpose The vector VECTOR is lexicographically greater than or equal to the fixed vector LOWER_BOUND and lexicographically smaller than or equal to the fixed vector UPPER_BOUND.

Example $(\langle 5, 2, 3, 9\rangle, \langle 5, 2, 6, 2\rangle, \langle 5, 2, 6, 3\rangle)$



The LEX_BETWEEN constraint holds since:

- The vector VECTOR = $\langle 5, 2, 6, 2\rangle$ is greater than or equal to the vector LOWER_BOUND = $\langle 5, 2, 3, 9\rangle$.
- The vector VECTOR = $\langle 5, 2, 6, 2\rangle$ is less than or equal to the vector UPPER_BOUND = $\langle 5, 2, 6, 3\rangle$.

Typical

$|$LOWER_BOUND$|$ > 1
LEX_LESSEQ(LOWER_BOUND, UPPER_BOUND)

Symmetries

- LOWER_BOUND.var can be decreased.
- UPPER_BOUND.var can be increased.

Arg. properties Suffix-contractible wrt. LOWER_BOUND, VECTOR and UPPER_BOUND (*remove items from same position*).

Usage This constraint does usually not occur explicitly in practice. However it shows up indirectly in the context of the LEX_CHAIN_LESS and the LEX_CHAIN_LESSEQ constraints: in order to have a complete filtering algorithm for the LEX_CHAIN_LESS and the LEX_CHAIN_LESSEQ constraints one has to come up with a complete filtering algorithm for the LEX_BETWEEN constraint. The reason is that the LEX_CHAIN_LESS as well as the LEX_CHAIN_LESSEQ constraints both compute feasible lower and upper bounds for each vector they mention. Therefore one ends up with a LEX_BETWEEN constraint for each vector of the LEX_CHAIN_LESS and LEX_CHAIN_LESSEQ constraints.

Algorithm [104].

Reformulation The LEX_BETWEEN(LOWER_BOUND, VECTORS, UPPER_BOUND) constraint can be expressed as the conjunction LEX_LESSEQ(LOWER_BOUND, VECTORS) $\wedge$ LEX_LESSEQ(VECTORS, UPPER_BOUND).

Systems LEXCHAINEQ in **Choco**, LEX_CHAIN in **SICStus**.

See also **common keyword:** LEX_CHAIN_GREATER, LEX_CHAIN_GREATEREQ, LEX_CHAIN_LESS, LEX_CHAIN_LESSEQ, LEX_GREATER, LEX_GREATEREQ, LEX_LESS *(lexicographic order)*.

part of system of constraints: LEX_LESSEQ.

Keywords **characteristic of a constraint:** vector, automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint, system of constraints.

filtering: arc-consistency.

symmetry: symmetry, lexicographic order.

Automaton Figure 5.504 depicts the automaton associated with the LEX_BETWEEN constraint. Let $\mathtt{L}_i$, $\mathtt{V}_i$ and $\mathtt{U}_i$ respectively be the var attributes of the i^{th} items of the LOWER_BOUND, the VECTOR and the UPPER_BOUND collections. To each triple $(\mathtt{L}_i, \mathtt{V}_i, \mathtt{U}_i)$ corresponds a signature variable S_i as well as the following signature constraint:

$(\mathtt{L}_i < \mathtt{V}_i) \wedge (\mathtt{V}_i < \mathtt{U}_i) \Leftrightarrow S_i = 0 \wedge$

$(\mathtt{L}_i < \mathtt{V}_i) \wedge (\mathtt{V}_i = \mathtt{U}_i) \Leftrightarrow S_i = 1 \wedge$

$(\mathtt{L}_i < \mathtt{V}_i) \wedge (\mathtt{V}_i > \mathtt{U}_i) \Leftrightarrow S_i = 2 \wedge$

$(\mathtt{L}_i = \mathtt{V}_i) \wedge (\mathtt{V}_i < \mathtt{U}_i) \Leftrightarrow S_i = 3 \wedge$

$(\mathtt{L}_i = \mathtt{V}_i) \wedge (\mathtt{V}_i = \mathtt{U}_i) \Leftrightarrow S_i = 4 \wedge$

$(\mathtt{L}_i = \mathtt{V}_i) \wedge (\mathtt{V}_i > \mathtt{U}_i) \Leftrightarrow S_i = 5 \wedge$

$(\mathtt{L}_i > \mathtt{V}_i) \wedge (\mathtt{V}_i < \mathtt{U}_i) \Leftrightarrow S_i = 6 \wedge$

$(\mathtt{L}_i > \mathtt{V}_i) \wedge (\mathtt{V}_i = \mathtt{U}_i) \Leftrightarrow S_i = 7 \wedge$

$(\mathtt{L}_i > \mathtt{V}_i) \wedge (\mathtt{V}_i > \mathtt{U}_i) \Leftrightarrow S_i = 8$.

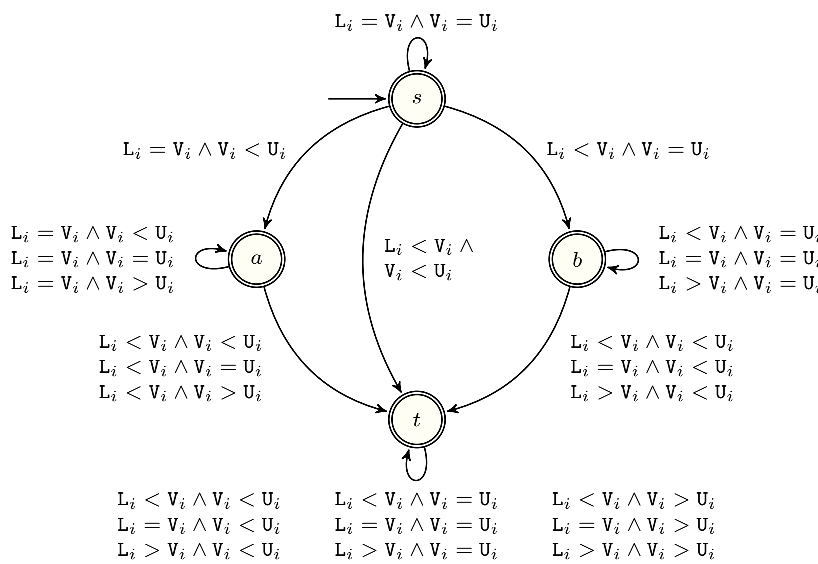


Figure 5.504: Automaton of the LEX_BETWEEN constraint

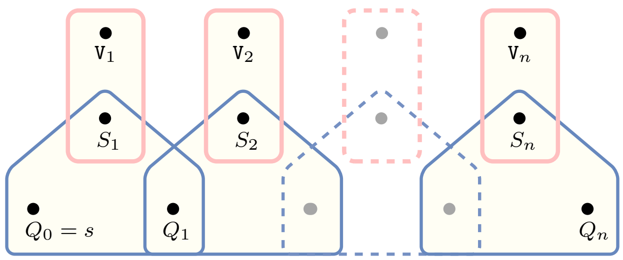


Figure 5.505: Hypergraph of the reformulation corresponding to the automaton of the LEX_BETWEEN constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

5.226 LEX_CHAIN_GREATER

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from LEX_CHAIN_LESS

Constraint LEX_CHAIN_GREATER(VECTORS)

Usual name LEX_CHAIN

Type VECTOR : collection(var−dvar)

Argument VECTORS : collection(vec − VECTOR)

Restrictions

$|\texttt{VECTOR}| \geq 1$
required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose

For each pair of consecutive vectors $\texttt{VECTOR}_i$ and $\texttt{VECTOR}_{i+1}$ of the VECTORS collection we have that $\texttt{VECTOR}_i$ is lexicographically strictly greater than $\texttt{VECTOR}_{i+1}$. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically strictly greater than* $\vec{Y}$ if and only if $X_0 > Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is lexicographically strictly greater than $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$(\langle \texttt{vec} - \langle 5, 2, 6, 3\rangle, \texttt{vec} - \langle 5, 2, 6, 2\rangle, \texttt{vec} - \langle 5, 2, 3, 9\rangle\rangle)$

5	2	6	3
=	=	=	>
5	2	6	2
=	=	>	
5	2	3	9

The LEX_CHAIN_GREATER constraint holds since:

- The first vector $\langle 5, 2, 6, 3\rangle$ of the VECTORS collection is lexicographically strictly greater than the second vector $\langle 5, 2, 6, 2\rangle$ of the VECTORS collection.
- The second vector $\langle 5, 2, 6, 2\rangle$ of the VECTORS collection is lexicographically strictly greater than the third vector $\langle 5, 2, 3, 9\rangle$ of the VECTORS collection.

Typical

$|\texttt{VECTOR}| > 1$
$|\texttt{VECTORS}| > 1$

Arg. properties

- Contractible wrt. VECTORS.
- Suffix-extensible wrt. VECTORS.vec (*add items at same position*).

Usage This constraint was motivated for breaking symmetry: more precisely when one wants to lexicographically order the consecutive columns of a matrix of decision variables. A further motivation is that using a set of lexicographic ordering constraints between two vectors does usually not allows the solver to come up with a complete pruning.

Algorithm A filtering algorithm achieving arc-consistency for a chain of lexicographical ordering constraints is presented in [104].

See also **common keyword:** LEX_BETWEEN, LEX_GREATEREQ, LEX_LESS, LEX_LESSEQ *(lexicographic order)*.

implies: LEX_ALLDIFFERENT, LEX_CHAIN_GREATEREQ.

part of system of constraints: LEX_GREATER.

used in graph description: LEX_GREATER.

Keywords **application area:** floor planning problem.

characteristic of a constraint: vector.

constraint type: decomposition, order constraint, system of constraints.

filtering: arc-consistency.

heuristics: heuristics and lexicographical ordering.

modelling: degree of diversity of a set of solutions.

modelling exercises: degree of diversity of a set of solutions.

symmetry: symmetry, matrix symmetry, lexicographic order.

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_GREATER(vectors1.vec, vectors2.vec)
Graph property(ies)	$\mathbf{NARC} = \lvert\texttt{VECTORS}\rvert - 1$

Graph model Parts (A) and (B) of Figure 5.506 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The LEX_CHAIN_GREATER constraint holds since all the arc constraints of the initial graph are satisfied.

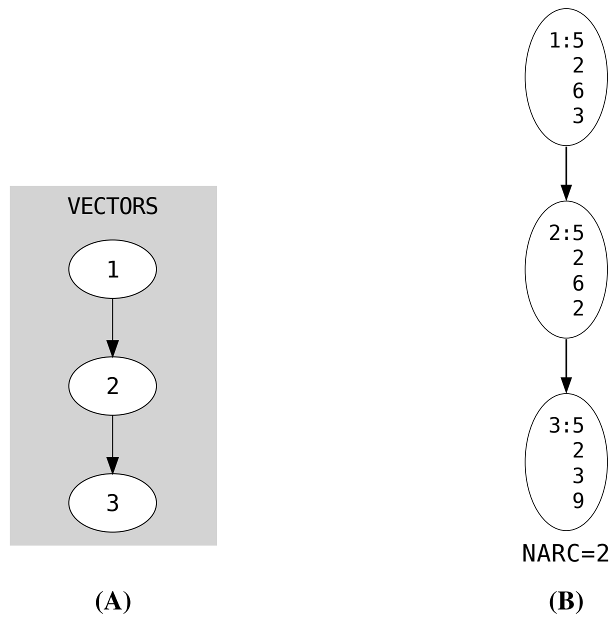


Figure 5.506: Initial and final graph of the LEX_CHAIN_GREATER constraint

Signature Since we use the $PATH$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to $\lvert\texttt{VECTORS}\rvert - 1$. For this reason we can rewrite $\mathbf{NARC} = \lvert\texttt{VECTORS}\rvert - 1$ to $\mathbf{NARC} \geq \lvert\texttt{VECTORS}\rvert - 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.227 LEX_CHAIN_GREATEREQ

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from LEX_CHAIN_LESSEQ

Constraint LEX_CHAIN_GREATEREQ(VECTORS)

Usual name LEX_CHAIN

Type VECTOR : collection(var−dvar)

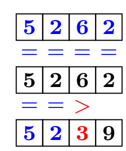
Argument VECTORS : collection(vec − VECTOR)

Restrictions
|VECTOR| ≥ 1
required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose For each pair of consecutive vectors $\texttt{VECTOR}_i$ and $\texttt{VECTOR}_{i+1}$ of the VECTORS collection we have that $\texttt{VECTOR}_i$ is lexicographically greater than or equal to $\texttt{VECTOR}_{i+1}$. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically greater than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 > Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is lexicographically greater than or equal to $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$$(\langle \texttt{vec} - \langle 5, 2, 6, 2\rangle, \texttt{vec} - \langle 5, 2, 6, 2\rangle, \texttt{vec} - \langle 5, 2, 3, 9\rangle\rangle)$$



The LEX_CHAIN_GREATEREQ constraint holds since:

- The first vector $\langle 5, 2, 6, 2\rangle$ of the VECTORS collection is lexicographically greater than or equal to the second vector $\langle 5, 2, 6, 2\rangle$ of the VECTORS collection.
- The second vector $\langle 5, 2, 6, 2\rangle$ of the VECTORS collection is lexicographically greater than or equal to the third vector $\langle 5, 2, 3, 9\rangle$ of the VECTORS collection.

Typical
|VECTOR| > 1
|VECTORS| > 1

Arg. properties
- Contractible wrt. VECTORS.
- Suffix-contractible wrt. VECTORS.vec (*remove items from same position*).

Usage This constraint was motivated for breaking symmetry: more precisely when one wants to lexicographically order the consecutive columns of a matrix of decision variables. A further motivation is that using a set of lexicographic ordering constraints between two vectors does usually not allows the solver to come up with a complete pruning.

Algorithm

A filtering algorithm achieving arc-consistency for a chain of lexicographical ordering constraints is presented in [104].

Six different ways of integrating a chain of lexicographical ordering constraints within non-overlapping constraints like DIFFN or GEOST and within their corresponding necessary conditions like the CUMULATIVE constraint are shown in [3].

See also

common keyword: LEX_BETWEEN, LEX_GREATER, LEX_LESS, LEX_LESSEQ *(lexicographic order)*.

implied by: LEX_CHAIN_GREATER *(non-strict order implied by strict order)*.

part of system of constraints: LEX_GREATEREQ.

used in graph description: LEX_GREATEREQ.

Keywords

characteristic of a constraint: vector.

constraint type: system of constraints, decomposition, order constraint.

filtering: arc-consistency.

heuristics: heuristics and lexicographical ordering.

symmetry: symmetry, matrix symmetry, lexicographic order.

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESSEQ(vectors1.vec, vectors2.vec)
Graph property(ies)	$\mathbf{NARC} = \lvert\text{VECTORS}\rvert - 1$

Graph model Parts (A) and (B) of Figure 5.507 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The LEX_CHAIN_GREATEREQ constraint holds since all the arc constraints of the initial graph are satisfied.

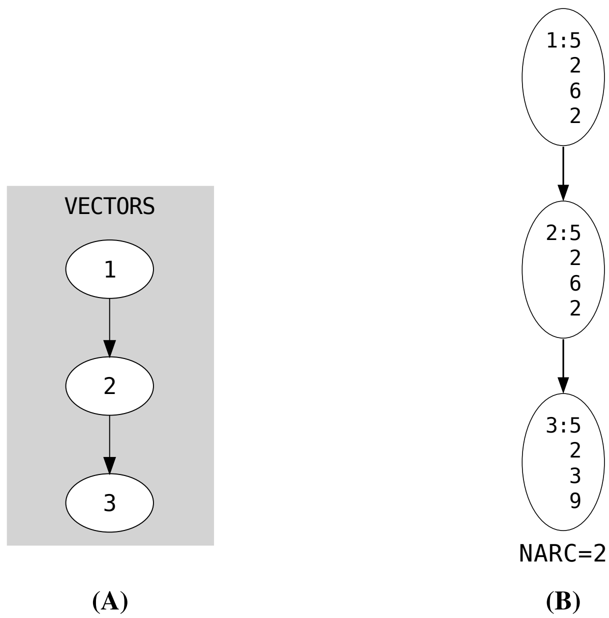


Figure 5.507: Initial and final graph of the LEX_CHAIN_GREATEREQ constraint

Signature Since we use the $PATH$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to $\lvert\text{VECTORS}\rvert - 1$. For this reason we can rewrite $\mathbf{NARC} = \lvert\text{VECTORS}\rvert - 1$ to $\mathbf{NARC} \geq \lvert\text{VECTORS}\rvert - 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.228 LEX_CHAIN_LESS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [104]

Constraint LEX_CHAIN_LESS(VECTORS)

Usual name LEX_CHAIN

Type VECTOR : collection(var−dvar)

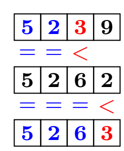
Argument VECTORS : collection(vec − VECTOR)

Restrictions
|VECTOR| ≥ 1
required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose
For each pair of consecutive vectors $\texttt{VECTOR}_i$ and $\texttt{VECTOR}_{i+1}$ of the VECTORS collection we have that $\texttt{VECTOR}_i$ is lexicographically strictly less than $\texttt{VECTOR}_{i+1}$. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically strictly less than* $\vec{Y}$ if and only if $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is lexicographically strictly less than $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$(\langle \texttt{vec} - \langle 5,2,3,9\rangle, \texttt{vec} - \langle 5,2,6,2\rangle, \texttt{vec} - \langle 5,2,6,3\rangle\rangle)$



The LEX_CHAIN_LESS constraint holds since:

- The first vector $\langle 5,2,3,9\rangle$ of the VECTORS collection is lexicographically strictly less than the second vector $\langle 5,2,6,2\rangle$ of the VECTORS collection.
- The second vector $\langle 5,2,6,2\rangle$ of the VECTORS collection is lexicographically strictly less than the third vector $\langle 5,2,6,3\rangle$ of the VECTORS collection.

Typical
|VECTOR| > 1
|VECTORS| > 1

Arg. properties
- Contractible wrt. VECTORS.
- Suffix-extensible wrt. VECTORS.vec (*add items at same position*).

Usage This constraint was motivated for breaking symmetry: more precisely when one wants to lexicographically order the consecutive columns of a matrix of decision variables. A further motivation is that using a set of lexicographic ordering constraints between two vectors does usually not allows the solver to come up with a complete pruning.

Algorithm

A filtering algorithm achieving arc-consistency for a chain of lexicographical ordering constraints is presented in [104].

Six different ways of integrating a chain of lexicographical ordering constraints within non-overlapping constraints like DIFFN or GEOST and within their corresponding necessary conditions like the CUMULATIVE constraint are shown in [3].

Systems

LEXCHAIN in **Choco**, LEX_CHAIN in **SICStus**.

See also

common keyword: GEOST *(symmetry, lexicographic ordering on the origins of* `tasks`*,* `rectangles`*, ...)*, LEX_BETWEEN, LEX_GREATER, LEX_GREATEREQ, LEX_LESSEQ *(lexicographic order)*.

implied by: STRICT_LEX2.

implies: LEX_ALLDIFFERENT, LEX_CHAIN_LESSEQ.

part of system of constraints: LEX_LESS.

related: CUMULATIVE, DIFFN *(lexicographic ordering on the origins of* `tasks`*,* `rectangles`*, ...)*.

system of constraints: STRICT_LEX2.

used in graph description: LEX_LESS.

Keywords

application area: floor planning problem.

characteristic of a constraint: vector.

constraint type: decomposition, order constraint, system of constraints.

filtering: arc-consistency.

heuristics: heuristics and lexicographical ordering.

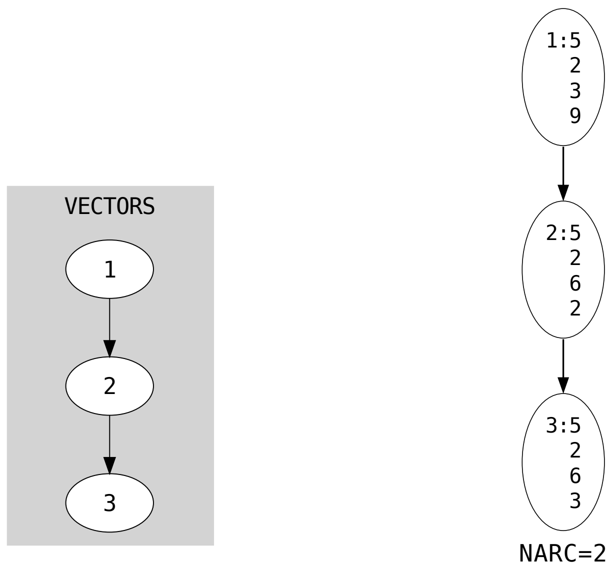
modelling: degree of diversity of a set of solutions.

modelling exercises: degree of diversity of a set of solutions.

symmetry: symmetry, matrix symmetry, lexicographic order.

Arc input(s)	VECTORS
Arc generator	*PATH* $\mapsto$collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESS(vectors1.vec, vectors2.vec)
Graph property(ies)	**NARC**$= \vert$VECTORS$\vert - 1$

Graph model Parts (A) and (B) of Figure 5.508 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The LEX_CHAIN_LESS constraint holds since all the arc constraints of the initial graph are satisfied.



(A) **(B)**

Figure 5.508: Initial and final graph of the LEX_CHAIN_LESS constraint

Signature Since we use the *PATH* arc generator on the VECTORS collection the number of arcs of the initial graph is equal to $\vert$VECTORS$\vert - 1$. For this reason we can rewrite $\mathbf{NARC} = \vert$VECTORS$\vert - 1$ to $\mathbf{NARC} \geq \vert$VECTORS$\vert - 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.229 LEX_CHAIN_LESSEQ

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [104]

Constraint LEX_CHAIN_LESSEQ(VECTORS)

Usual name LEX_CHAIN

Type

VECTOR : collection(var−dvar)

Argument

VECTORS : collection(vec − VECTOR)

Restrictions

|VECTOR| ≥ 1
required(VECTOR, var)
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose

For each pair of consecutive vectors VECTOR_i and VECTOR_{i+1} of the VECTORS collection we have that VECTOR_i is lexicographically less than or equal to VECTOR_{i+1}. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically less than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is lexicographically less than or equal to $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$(\langle \text{vec} - \langle 5, 2, 3, 9\rangle, \text{vec} - \langle 5, 2, 6, 2\rangle, \text{vec} - \langle 5, 2, 6, 2\rangle\rangle)$

5	2	3	9
=	=	<	
5	2	6	2
=	=	=	=
5	2	6	2

The LEX_CHAIN_LESSEQ constraint holds since:

- The first vector $\langle 5, 2, 3, 9\rangle$ of the VECTORS collection is lexicographically less than or equal to the second vector $\langle 5, 2, 6, 2\rangle$ of the VECTORS collection.
- The second vector $\langle 5, 2, 6, 2\rangle$ of the VECTORS collection is lexicographically less than or equal to the third vector $\langle 5, 2, 6, 2\rangle$ of the VECTORS collection.

Typical

|VECTOR| > 1
|VECTORS| > 1

Arg. properties

- Contractible wrt. VECTORS.
- Suffix-contractible wrt. VECTORS.vec (*remove items from same position*).

Usage This constraint was motivated for breaking symmetry: more precisely when one wants to lexicographically order the consecutive columns of a matrix of decision variables. A further motivation is that using a set of lexicographic ordering constraints between two vectors does usually not allows the solver to come up with a complete pruning.

Algorithm A filtering algorithm achieving arc-consistency for a chain of lexicographical ordering constraints is presented in [104].

Six different ways of integrating a chain of lexicographical ordering constraints within non-overlapping constraints like DIFFN or GEOST and within their corresponding necessary conditions like the CUMULATIVE constraint are shown in [3].

Systems LEXCHAINEQ in **Choco**, LEX_CHAIN in **SICStus**.

See also **common keyword:** ALLPERM *(lexicographic order)*, GEOST *(symmetry, lexicographic ordering on the origins of* `tasks`*,* `rectangles`*, ...)*, LEX_BETWEEN, LEX_GREATER, LEX_GREATEREQ, LEX_LESS *(lexicographic order)*.

implied by: LEX2 *(columns lex ordering imposed by constraint* LEX2 *removed)*, LEX_CHAIN_LESS *(non-strict order implied by strict order)*, ORDERED_ATLEAST_NVECTOR *(*`NVEC` *of constraint* ORDERED_ATLEAST_NVECTOR *removed)*, ORDERED_ATMOST_NVECTOR *(*`NVEC` *of constraint* ORDERED_ATMOST_NVECTOR *removed)*, ORDERED_NVECTOR *(*`NVEC` *of constraint* ORDERED_NVECTOR *removed)*.

part of system of constraints: LEX_LESSEQ.

related: CUMULATIVE, DIFFN *(lexicographic ordering on the origins of* `tasks`*,* `rectangles`*, ...)*.

system of constraints: LEX2.

used in graph description: LEX_LESSEQ.

Keywords **characteristic of a constraint:** vector.

constraint type: system of constraints, decomposition, order constraint.

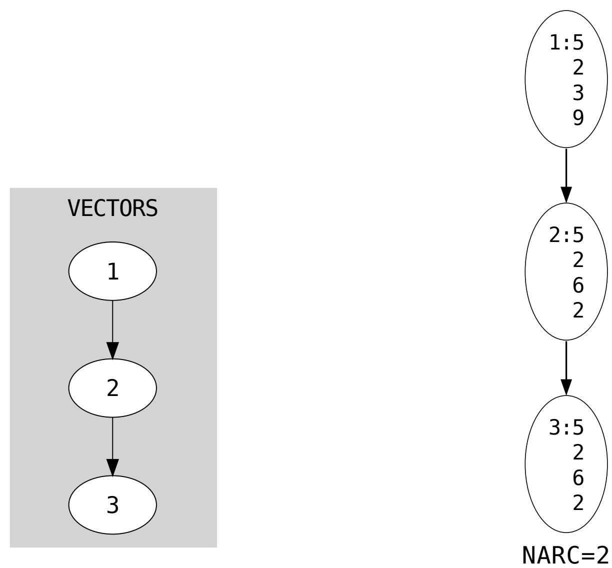
filtering: arc-consistency.

heuristics: heuristics and lexicographical ordering.

symmetry: symmetry, matrix symmetry, lexicographic order.

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESSEQ(vectors1.vec, vectors2.vec)
Graph property(ies)	$\mathbf{NARC}$ = |VECTORS| − 1

Graph model Parts (A) and (B) of Figure 5.509 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold. The LEX_CHAIN_LESSEQ constraint holds since all the arc constraints of the initial graph are satisfied.



(A) (B)

Figure 5.509: Initial and final graph of the LEX_CHAIN_LESSEQ constraint

Signature Since we use the $PATH$ arc generator on the VECTORS collection the number of arcs of the initial graph is equal to \|VECTORS\| − 1. For this reason we can rewrite $\mathbf{NARC}$ = \|VECTORS\| − 1 to $\mathbf{NARC} \geq$ \|VECTORS\| − 1 and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.230 LEX_DIFFERENT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Used for defining LEX_ALLDIFFERENT.

Constraint LEX_DIFFERENT(VECTOR1, VECTOR2)

Synonyms DIFFERENT, DIFF.

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

Restrictions

required(VECTOR1, var)
required(VECTOR2, var)
$|\mathtt{VECTOR1}| > 0$
$|\mathtt{VECTOR1}| = |\mathtt{VECTOR2}|$

Purpose Vectors VECTOR1 and VECTOR2 differ in at least one component.

Example $(\langle 5, 2, 7, 1\rangle, \langle 5, 3, 7, 1\rangle)$

5	2	7	1
$=$	$\neq$	$=$	$=$
5	3	7	1

The LEX_DIFFERENT constraint holds since VECTOR1 $= \langle 5, 2, 7, 1\rangle$ and VECTOR2 $= \langle 5, 3, 7, 1\rangle$ differ in their second components.

Typical

$|\mathtt{VECTOR1}| > 1$
range(VECTOR1.var) > 1
range(VECTOR2.var) > 1

Symmetries

- Arguments are permutable w.r.t. permutation (VECTOR1, VECTOR2).
- Items of VECTOR1 and VECTOR2 are permutable (*same permutation used*).

Arg. properties Extensible wrt. VECTOR1 and VECTOR2 (*add items at same position*).

Reformulation The LEX_DIFFERENT$(\langle \mathtt{var} - U_1, \mathtt{var} - U_2, \dots, \mathtt{var} - U_{|\mathtt{VECTOR1}|}\rangle, \langle \mathtt{var} - V_1, \mathtt{var} - V_2, \dots, \mathtt{var} - V_{|\mathtt{VECTOR2}|}\rangle)$ constraint can be expressed in term of the following disjunction of disequality constraints $U_1 \neq V_1 \vee U_2 \neq V_2 \vee \dots \vee U_{|\mathtt{VECTOR1}|} \neq V_{|\mathtt{VECTOR2}|}$.

Used in LEX_ALLDIFFERENT, SORT_PERMUTATION.

See also

common keyword: LEX_GREATEREQ, LEX_LESSEQ *(vector)*.

implied by: DISJOINT, INCOMPARABLE, LEX_GREATER, LEX_LESS.

negation: LEX_EQUAL.

system of constraints: LEX_ALLDIFFERENT.

Keywords

characteristic of a constraint: vector, disequality, automaton, automaton without counters, reified automaton constraint.

constraint arguments: constraint between two collections of variables.

constraint network structure: Berge-acyclic constraint network.

filtering: arc-consistency.

Arc input(s)	VECTOR1 VECTOR2
Arc generator	$PRODUCT(=) \mapsto$ collection(vector1, vector2)
Arc arity	2
Arc constraint(s)	vector1.var $\neq$ vector2.var
Graph property(ies)	**NARC** ≥ 1

Graph model Parts (A) and (B) of Figure 5.510 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold. It corresponds to a component where the two vectors differ.

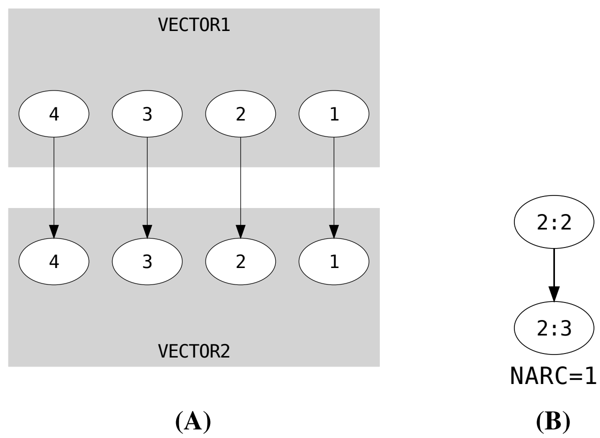


Figure 5.510: Initial and final graph of the LEX_DIFFERENT constraint

Automaton Figure 5.511 depicts the automaton associated with the LEX_DIFFERENT constraint. Let $\mathtt{VAR1}_i$ and $\mathtt{VAR2}_i$ respectively be the `var` attributes of the i^{th} items of the `VECTOR1` and the `VECTOR2` collections. To each pair $(\mathtt{VAR1}_i, \mathtt{VAR2}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $\mathtt{VAR1}_i = \mathtt{VAR2}_i \Leftrightarrow S_i$.

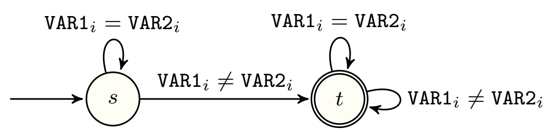


Figure 5.511: Automaton of the LEX_DIFFERENT constraint

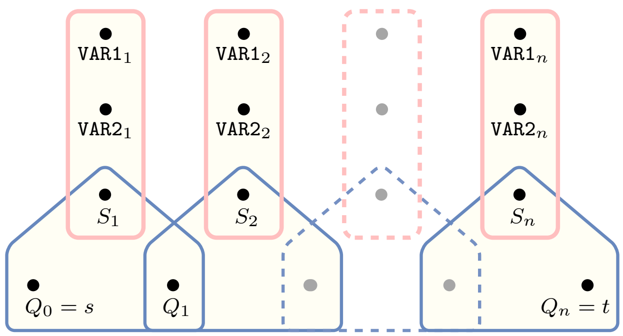


Figure 5.512: Hypergraph of the reformulation corresponding to the automaton of the LEX_DIFFERENT constraint

5.231 LEX_EQUAL

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Initially introduced for defining NVECTOR

Constraint LEX_EQUAL(VECTOR1, VECTOR2)

Synonyms EQUAL, EQ.

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

Restrictions

required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|

Purpose VECTOR1 is *equal to* VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 = Y_0 \wedge X_1 = Y_1 \wedge \cdots \wedge X_{n-1} = Y_{n-1}$.

Example $(\langle 1, 9, 1, 5\rangle, \langle 1, 9, 1, 5\rangle)$

1	9	1	5
=	=	=	=
1	9	1	5

The LEX_EQUAL constraint holds since (1) the first component of the first vector is equal to the first component of the second vector, (2) the second component of the first vector is equal to the second component of the second vector, (3) the third component of the first vector is equal to the third component of the second vector and (4) the fourth component of the first vector is equal to the fourth component of the second vector.

All solutions Figure 5.513 gives all solutions to the following non ground instance of the LEX_EQUAL constraint: $X_0 \in [1, 2]$, $X_1 \in [1, 2]$, $X_2 \in [1, 2]$, $Y_0 \in [0, 1]$, $Y_1 \in [0, 2]$, $Y_2 \in [2, 4]$, LEX_EQUAL0($\langle X_0, X_1, X_2\rangle, \langle Y_0, Y_1, Y_2\rangle$).

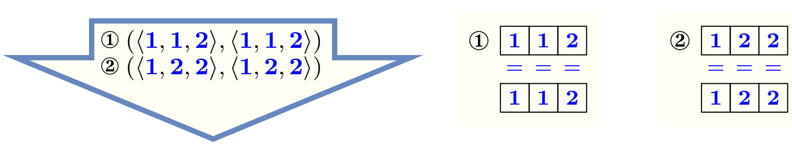


Figure 5.513: All solutions corresponding to the non ground example of the LEX_EQUAL constraint

Typical

|VECTOR1| > 1
range(VECTOR1.var) > 1
range(VECTOR2.var) > 1

Symmetries

- Arguments are permutable w.r.t. permutation (VECTOR1, VECTOR2).
- Items of VECTOR1 and VECTOR2 are permutable (*same permutation used*).

Arg. properties

Contractible wrt. VECTOR1 and VECTOR2 (*remove items from same position*).

Used in

ATLEAST_NVECTOR, ATMOST_NVECTOR, NVECTOR, NVECTORS.

See also

common keyword: NVECTOR *(vector)*.

implied by: VEC_EQ_TUPLE.

implies: LEX_GREATEREQ, LEX_LESSEQ, SAME.

negation: LEX_DIFFERENT.

specialisation: VEC_EQ_TUPLE *(*variable *replaced by* integer *in second argument)*.

Keywords

characteristic of a constraint: vector, automaton, automaton without counters, reified automaton constraint.

constraint arguments: constraint between two collections of variables.

constraint network structure: Berge-acyclic constraint network.

filtering: arc-consistency.

final graph structure: acyclic, bipartite, no loop.

Arc input(s)	VECTOR1 VECTOR2
Arc generator	$PRODUCT(=) \mapsto$ collection(vector1, vector2)
Arc arity	2
Arc constraint(s)	vector1.var $=$ vector2.var
Graph property(ies)	**NARC** $= \|$VECTOR1$\|$
Graph class	• ACYCLIC • BIPARTITE • NO_LOOP

Graph model Parts (A) and (B) of Figure 5.514 respectively show the initial and final graphs associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

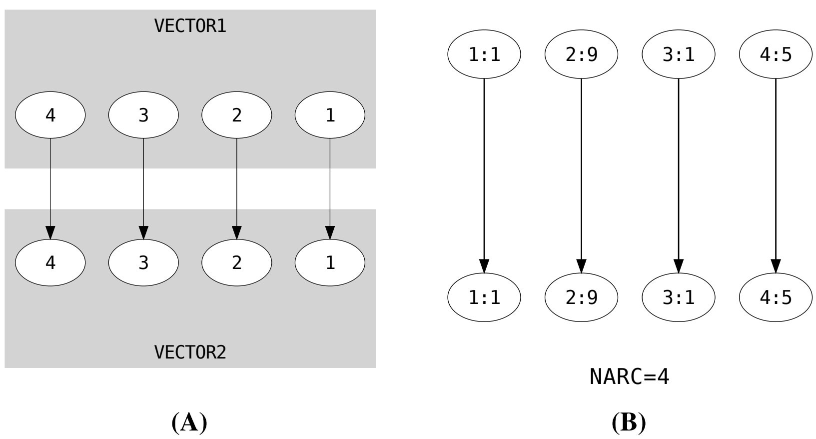


Figure 5.514: Initial and final graph of the LEX_EQUAL constraint

Automaton Figure 5.515 depicts the automaton associated with the LEX_EQUAL constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the var attributes of the i^{th} items of the VECTOR1 and the VECTOR2 collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\texttt{VAR1}_i \neq \texttt{VAR2}_i \Leftrightarrow S_i = 0) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow S_i = 1)$.

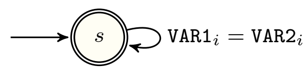


Figure 5.515: Automaton of the LEX_EQUAL constraint

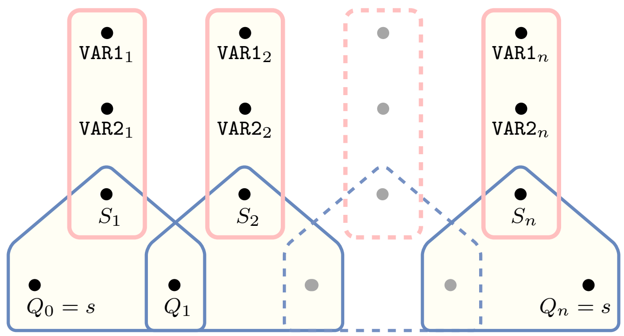


Figure 5.516: Hypergraph of the reformulation corresponding to the automaton of the LEX_EQUAL constraint

5.232 LEX_GREATER

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint LEX_GREATER(VECTOR1, VECTOR2)

Synonyms LEX, LEX_CHAIN, REL, GREATER, GT.

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

Restrictions

required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|

Purpose

VECTOR1 is *lexicographically strictly greater than* VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically strictly greater than* $\vec{Y}$ if and only if $X_0 > Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is *lexicographically strictly greater than* $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$(\langle 5, 2, 7, 1\rangle, \langle 5, 2, 6, 2\rangle)$

5	2	7	1
=	=	>	
5	2	6	2

The LEX_GREATER constraint holds since VECTOR1 $= \langle 5, 2, 7, 1\rangle$ is lexicographically strictly greater than VECTOR2 $= \langle 5, 2, 6, 2\rangle$.

All solutions

Figure 5.517 gives all solutions to the following non ground instance of the LEX_GREATER constraint: $X_0 \in [0, 1]$, $X_1 \in [0, 2]$, $X_2 \in [1, 1]$, $Y_0 \in [0, 1]$, $Y_1 \in [1, 2]$, $Y_2 \in [4, 4]$, LEX_GREATER($\langle X_0, X_1, X_2\rangle$, $\langle Y_0, Y_1, Y_2\rangle$).

Typical

$$|\text{VECTOR1}| > 1$$
$$\vee \left(\begin{array}{l} |\text{VECTOR1}| < 5, \\ \text{nval}([\text{VECTOR1.var}, \text{VECTOR2.var}]) < 2 * |\text{VECTOR1}| \end{array} \right)$$
$$\vee \left(\begin{array}{l} \text{maxval}([\text{VECTOR1.var}, \text{VECTOR2.var}]) \leq 1, \\ 2 * |\text{VECTOR1}| - \text{MAX\_NVALUE}([\text{VECTOR1.var}, \text{VECTOR2.var}]) > 2 \end{array} \right)$$

Symmetries

- VECTOR1.var can be increased.
- VECTOR2.var can be decreased.

Arg. properties

Suffix-extensible wrt. VECTOR1 and VECTOR2 (*add items at same position*).

Remark

A *multiset ordering* constraint and its corresponding filtering algorithm are described in [185].

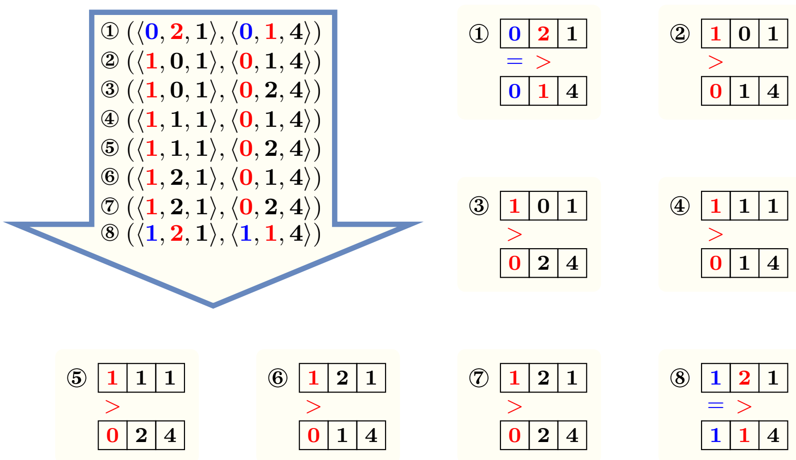


Figure 5.517: All solutions corresponding to the non ground example of the LEX_GREATER constraint

Algorithm The first filtering algorithm maintaining arc-consistency for this constraint was presented in [184]. A second filtering algorithm maintaining arc-consistency and detecting entailment in a more eager way, was given in [105]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [184] detecting entailment is given in the PhD thesis of Z. Kızıltan [250, page 95]. The previous thesis [250, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence. Finally, T. Frühwirth shows how to encode lexicographic ordering constraints within the context of CHR [186] in [187].

Reformulation The following reformulations in term of arithmetic and/or logical expressions exist for enforcing the *lexicographically strictly greater than* constraint. The first one converts $\vec{X}$ and $\vec{Y}$ into numbers and post an inequality constraint. It assumes all components of $\vec{X}$ and $\vec{Y}$ to be within $[0, a-1]$:

$$a^{n-1}Y_0 + a^{n-2}Y_1 + \cdots + a^0 Y_{n-1} < a^{n-1}X_0 + a^{n-2}X_1 + \cdots + a^0 X_{n-1}$$

Since the previous reformulation can only be used with small values of n and a, W. Harvey came up with the following alternative model that maintains arc-consistency:

$$(Y_0 < X_0 + (Y_1 < X_1 + (\cdots + (Y_{n-1} < X_{n-1} + 0)\ldots))) = 1$$

Finally, the *lexicographically strictly greater than* constraint can be expressed as a conjunction or a disjunction of constraints:

$$
\begin{array}{rl}
Y_0 \leq X_0 & \wedge \\
(Y_0 = X_0) \Rightarrow Y_1 \leq X_1 & \wedge \\
(Y_0 = X_0 \wedge Y_1 = X_1) \Rightarrow Y_2 \leq X_2 & \wedge \\
\vdots & \\
(Y_0 = X_0 \wedge Y_1 = X_1 \wedge \cdots \wedge Y_{n-2} = X_{n-2}) \Rightarrow Y_{n-1} < X_{n-1} &
\end{array}
$$

$$
\begin{array}{rl}
Y_0 < X_0 & \vee \\
Y_0 = X_0 \wedge Y_1 < X_1 & \vee \\
Y_0 = X_0 \wedge Y_1 = X_1 \wedge Y_2 < X_2 & \vee \\
\vdots & \\
Y_0 = X_0 \wedge Y_1 = X_1 \wedge \cdots \wedge Y_{n-2} = X_{n-2} \wedge Y_{n-1} < X_{n-1} &
\end{array}
$$

When used separately, the two previous logical decompositions do not maintain arc-consistency.

Systems LEX in **Choco**, REL in **Gecode**, LEX_GREATER in **MiniZinc**, LEX_CHAIN in **SICStus**.

Used in LEX_CHAIN_GREATER.

See also **common keyword:** COND_LEX_GREATER, LEX_BETWEEN, LEX_CHAIN_GREATEREQ, LEX_CHAIN_LESS, LEX_CHAIN_LESSEQ *(lexicographic order)*.

implies: LEX_DIFFERENT, LEX_GREATEREQ.

implies (if swap arguments): LEX_LESS.

negation: LEX_LESSEQ.

system of constraints: LEX_CHAIN_GREATER.

Keywords **characteristic of a constraint:** vector, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: constraint between two collections of variables.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: duplicated variables, arc-consistency.

heuristics: heuristics and lexicographical ordering.

symmetry: symmetry, matrix symmetry, lexicographic order, multiset ordering.

Derived Collections

$$\texttt{col}\left(\begin{array}{l}\texttt{DESTINATION}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{x}-\texttt{int},\texttt{y}-\texttt{int}),\\ [\texttt{item}(\texttt{index}-0,\texttt{x}-0,\texttt{y}-0)]\end{array}\right)$$

$$\texttt{col}\left(\begin{array}{l}\texttt{COMPONENTS}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{x}-\texttt{dvar},\texttt{y}-\texttt{dvar}),\\ \left[\texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{VECTOR1.key},\\ \texttt{x}-\texttt{VECTOR1.var},\\ \texttt{y}-\texttt{VECTOR2.var}\end{array}\right)\right]\end{array}\right)$$

Arc input(s) COMPONENTS DESTINATION

Arc generator $PRODUCT(PATH, VOID) \mapsto \texttt{collection}(\texttt{item1},\texttt{item2})$

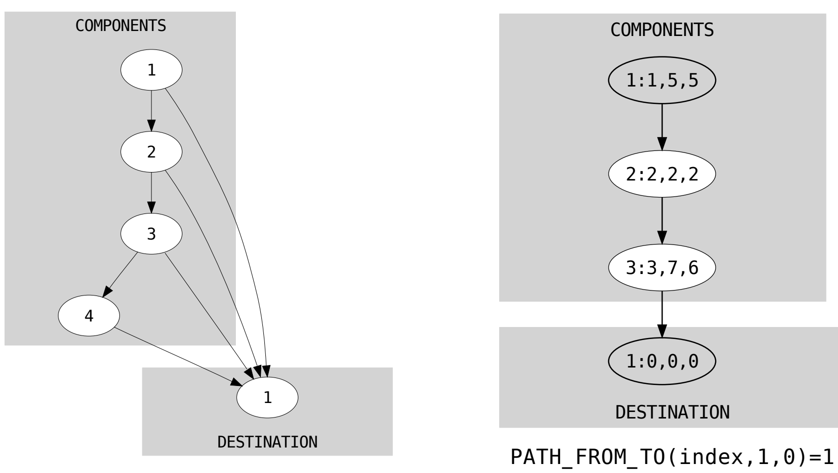
Arc arity 2

Arc constraint(s) $\bigvee\left(\begin{array}{l}\texttt{item2.index}>0\wedge\texttt{item1.x}=\texttt{item1.y},\\ \texttt{item2.index}=0\wedge\texttt{item1.x}>\texttt{item1.y}\end{array}\right)$

Graph property(ies) $\mathbf{PATH\_FROM\_TO}(\texttt{index},1,0)=1$

Graph model Parts (A) and (B) of Figure 5.518 respectively show the initial and final graph associated with the **Example** slot. Since we use the **PATH_FROM_TO** graph property we show the following information on the final graph:

- The vertices, which respectively correspond to the start and the end of the required path, are stressed in bold.
- The arcs on the required path are also stressed in bold.



(A) **(B)**

Figure 5.518: Initial and final graph of the LEX_GREATER constraint

The vertices of the initial graph are generated in the following way:

- We create a vertex c_i for each pair of components that both have the same index i.

- We create an additional dummy vertex called d.

The arcs of the initial graph are generated in the following way:

- We create an arc between c_i and d. We associate to this arc the arc constraint $\mathtt{item}_1.x > \mathtt{item}_2.y$.
- We create an arc between c_i and c_{i+1}. We associate to this arc the arc constraint $\mathtt{item}_1.x = \mathtt{item}_2.y$.

The LEX_GREATER constraint holds when there exist a path from c_1 to d. This path can be interpreted as a sequence of *equality* constraints on the prefix of both vectors, immediately followed by a *greater than* constraint.

Signature Since the maximum value returned by the graph property $\mathbf{PATH\_FROM\_TO}$ is equal to 1 we can rewrite $\mathbf{PATH\_FROM\_TO}(\mathtt{index}, 1, 0) = 1$ to $\mathbf{PATH\_FROM\_TO}(\mathtt{index}, 1, 0) \geq 1$. Therefore we simplify $\overline{\underline{\mathbf{PATH\_FROM\_TO}}}$ to $\overline{\mathbf{PATH\_FROM\_TO}}$.

Automaton Figure 5.519 depicts the automaton associated with the LEX_GREATER constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the `var` attributes of the i^{th} items of the `VECTOR1` and the `VECTOR2` collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\texttt{VAR1}_i < \texttt{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\texttt{VAR1}_i > \texttt{VAR2}_i \Leftrightarrow S_i = 3)$.

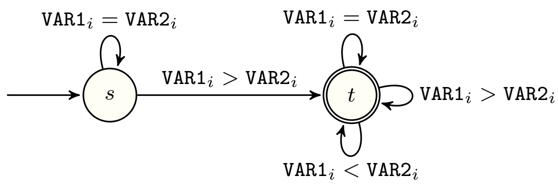


Figure 5.519: Automaton of the LEX_GREATER constraint

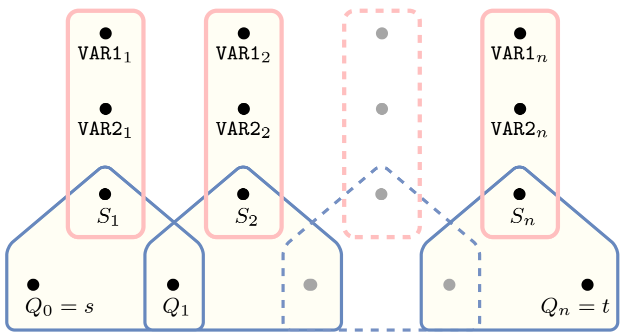


Figure 5.520: Hypergraph of the reformulation corresponding to the automaton of the LEX_GREATER constraint

5.233 LEX_GREATEREQ

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint LEX_GREATEREQ(VECTOR1, VECTOR2)

Synonyms LEXEQ, LEX_CHAIN, REL, GREATEREQ, GEQ, LEX_GEQ.

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

Restrictions

required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|

Purpose

VECTOR1 is *lexicographically greater than or equal to* VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \dots, X_{n-1}\rangle$ and $\langle Y_0, \dots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically greater than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 > Y_0$ or $X_0 = Y_0$ and $\langle X_1, \dots, X_{n-1}\rangle$ is *lexicographically greater than or equal to* $\langle Y_1, \dots, Y_{n-1}\rangle$.

Example

$(\langle 5, 2, 8, 9\rangle, \langle 5, 2, 6, 2\rangle)$
$(\langle 5, 2, 3, 9\rangle, \langle 5, 2, 3, 9\rangle)$

5	2	8	9
=	=	>	
5	2	6	2

5	2	3	9
=	=	=	=
5	2	3	9

The LEX_GREATEREQ constraints associated with the first and second examples hold since:

- Within the first example VECTOR1 $= \langle 5, 2, 8, 9\rangle$ is lexicographically greater than or equal to VECTOR2 $= \langle 5, 2, 6, 2\rangle$.
- Within the second example VECTOR1 $= \langle 5, 2, 3, 9\rangle$ is lexicographically greater than or equal to VECTOR2 $= \langle 5, 2, 3, 9\rangle$.

All solutions

Figure 5.521 gives all solutions to the following non ground instance of the LEX_GREATEREQ constraint: $X_0 \in [0, 1]$, $X_1 \in [0, 2]$, $X_2 \in [1, 1]$, $Y_0 \in [1, 2]$, $Y_1 \in [1, 2]$, $Y_2 \in [1, 2]$, LEX_GREATEREQ0$(\langle X_0, X_1, X_2\rangle, \langle Y_0, Y_1, Y_2\rangle)$.

Typical

$$|\text{VECTOR1}| > 1$$
$$\vee \left(\begin{array}{l} |\text{VECTOR1}| < 5, \\ \text{nval}([\text{VECTOR1.var}, \text{VECTOR2.var}]) < 2 * |\text{VECTOR1}| \end{array} \right)$$
$$\vee \left(\begin{array}{l} \text{maxval}([\text{VECTOR1.var}, \text{VECTOR2.var}]) \leq 1, \\ 2 * |\text{VECTOR1}| - \text{MAX\_NVALUE}([\text{VECTOR1.var}, \text{VECTOR2.var}]) > 2 \end{array} \right)$$

Symmetries

- VECTOR1.var can be increased.
- VECTOR2.var can be decreased.

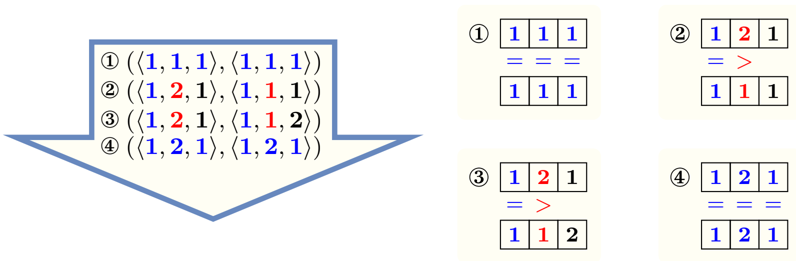


Figure 5.521: All solutions corresponding to the non ground example of the LEX_GREATEREQ constraint

Arg. properties Suffix-contractible wrt. `VECTOR1` and `VECTOR2` (*remove items from same position*).

Remark A *multiset ordering* constraint and its corresponding filtering algorithm are described in [185].

Algorithm The first filtering algorithm maintaining arc-consistency for this constraint was presented in [184]. A second filtering algorithm maintaining arc-consistency and detecting entailment in a more eager way, was given in [105]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [184] detecting entailment is given in the PhD thesis of Z. Kızıltan [250, page 95]. The previous thesis [250, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence. Finally, T. Frühwirth shows how to encode lexicographic ordering constraints within the context of CHR [186] in [187].

Reformulation The following reformulations in term of arithmetic and/or logical expressions exist for enforcing the *lexicographically greater than or equal to* constraint. The first one converts $\vec{X}$ and $\vec{Y}$ into numbers and post an inequality constraint. It assumes all components of $\vec{X}$ and $\vec{Y}$ to be within $[0, a-1]$:

$$a^{n-1}Y_0 + a^{n-2}Y_1 + \cdots + a^0 Y_{n-1} \leq a^{n-1}X_0 + a^{n-2}X_1 + \cdots + a^0 X_{n-1}$$

Since the previous reformulation can only be used with small values of n and a, W. Harvey came up with the following alternative model that maintains arc-consistency:

$$(Y_0 < X_0 + (Y_1 < X_1 + (\cdots + (Y_{n-1} < X_{n-1} + 1)\ldots))) = 1$$

Finally, the *lexicographically greater than or equal to* constraint can be expressed as a conjunction or a disjunction of constraints:

$$\begin{array}{rl} Y_0 \leq X_0 & \wedge \\ (Y_0 = X_0) \Rightarrow Y_1 \leq X_1 & \wedge \\ (Y_0 = X_0 \wedge Y_1 = X_1) \Rightarrow Y_2 \leq X_2 & \wedge \\ \vdots & \\ (Y_0 = X_0 \wedge Y_1 = X_1 \wedge \cdots \wedge Y_{n-2} = X_{n-2}) \Rightarrow Y_{n-1} \leq X_{n-1} & \end{array}$$

$$\begin{array}{rl} Y_0 < X_0 & \vee \\ Y_0 = X_0 \wedge Y_1 < X_1 & \vee \\ Y_0 = X_0 \wedge Y_1 = X_1 \wedge Y_2 < X_2 & \vee \\ \vdots & \\ Y_0 = X_0 \wedge Y_1 = X_1 \wedge \cdots \wedge Y_{n-2} = X_{n-2} \wedge Y_{n-1} \leq X_{n-1} & \end{array}$$

When used separately, the two previous logical decompositions do not maintain arc-consistency.

Systems

LEXEQ in **Choco**, REL in **Gecode**, LEX_GREATEREQ in **MiniZinc**, LEX_CHAIN in **SICStus**.

See also

common keyword: COND_LEX_GREATEREQ, LEX_BETWEEN, LEX_CHAIN_GREATER, LEX_CHAIN_LESS, LEX_CHAIN_LESSEQ *(lexicographic order)*, LEX_DIFFERENT *(vector)*.

implied by: LEX_EQUAL, LEX_GREATER, SORT.

implies (if swap arguments): LEX_LESSEQ.

negation: LEX_LESS.

system of constraints: LEX_CHAIN_GREATEREQ.

uses in its reformulation: LEX_CHAIN_GREATEREQ.

Keywords

characteristic of a constraint: vector, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: constraint between two collections of variables.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: duplicated variables, arc-consistency.

heuristics: heuristics and lexicographical ordering.

symmetry: symmetry, matrix symmetry, lexicographic order, multiset ordering.

Derived Collections

$$\texttt{col}\left(\begin{array}{l}\texttt{DESTINATION}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{x}-\texttt{int},\texttt{y}-\texttt{int}),\\ [\texttt{item}(\texttt{index}-0,\texttt{x}-0,\texttt{y}-0)]\end{array}\right)$$

$$\texttt{col}\left(\begin{array}{l}\texttt{COMPONENTS}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{x}-\texttt{dvar},\texttt{y}-\texttt{dvar}),\\ \left[\texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{VECTOR1.key},\\ \texttt{x}-\texttt{VECTOR1.var},\\ \texttt{y}-\texttt{VECTOR2.var}\end{array}\right)\right]\end{array}\right)$$

Arc input(s) COMPONENTS DESTINATION

Arc generator $PRODUCT(PATH, VOID)\mapsto\texttt{collection}(\texttt{item1},\texttt{item2})$

Arc arity 2

Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\texttt{item2.index}>0\wedge\texttt{item1.x}=\texttt{item1.y},\\ \bigwedge\left(\begin{array}{l}\texttt{item1.index}<|\texttt{VECTOR1}|,\\ \texttt{item2.index}=0,\\ \texttt{item1.x}>\texttt{item1.y}\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{item1.index}=|\texttt{VECTOR1}|,\\ \texttt{item2.index}=0,\\ \texttt{item1.x}\geq\texttt{item1.y}\end{array}\right)\end{array}\right)$$

Graph property(ies) $\mathbf{PATH\_FROM\_TO}(\texttt{index},1,0)=1$

Graph model

Parts (A) and (B) of Figure 5.522 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **PATH_FROM_TO** graph property we show on the final graph the following information:

- The vertices, which respectively correspond to the start and the end of the required path, are stressed in bold.
- The arcs on the required path are also stressed in bold.

The vertices of the initial graph are generated in the following way:

- We create a vertex c_i for each pair of components that both have the same index i.
- We create an additional dummy vertex called d.

The arcs of the initial graph are generated in the following way:

- We create an arc between c_i and d. When c_i was generated from the last components of both vectors We associate to this arc the arc constraint $\texttt{item}_1.x\geq\texttt{item}_2.y$; Otherwise we associate to this arc the arc constraint $\texttt{item}_1.x>\texttt{item}_2.y$;
- We create an arc between c_i and c_{i+1}. We associate to this arc the arc constraint $\texttt{item}_1.x=\texttt{item}_2.y$.

The LEX_GREATEREQ constraint holds when there exist a path from c_1 to d. This path can be interpreted as a maximum sequence of *equality* constraints on the prefix of both vectors, possibly followed by a *greater than* constraint.

Signature

Since the maximum value returned by the graph property **PATH_FROM_TO** is equal to 1 we can rewrite $\mathbf{PATH\_FROM\_TO}(\texttt{index},1,0)=1$ to $\underline{\mathbf{PATH\_FROM\_TO}}(\texttt{index},1,0)\geq 1$. Therefore we simplify $\underline{\overline{\mathbf{PATH\_FROM\_TO}}}$ to $\mathbf{PATH\_FROM\_TO}$.

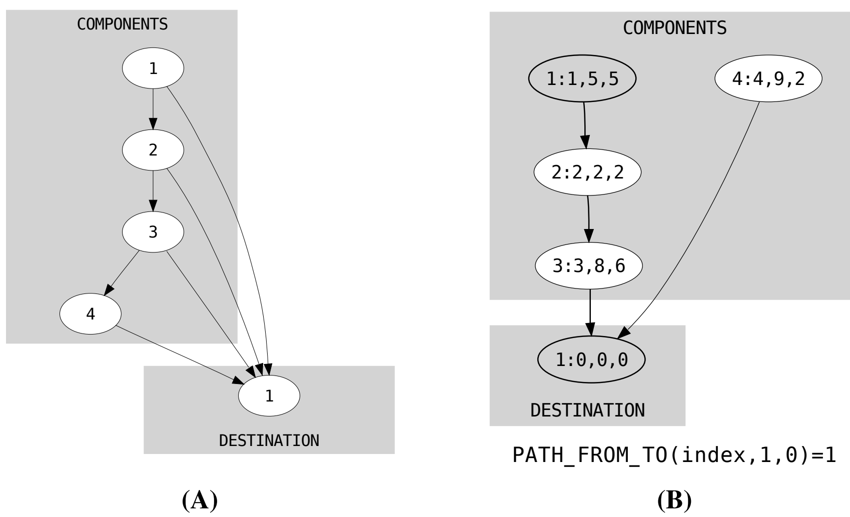


Figure 5.522: Initial and final graph of the LEX_GREATEREQ constraint

Automaton

Figure 5.523 depicts the automaton associated with the LEX_GREATEREQ constraint. Let VAR1_i and VAR2_i respectively be the var attributes of the i^{th} items of the VECTOR1 and the VECTOR2 collections. To each pair $(\text{VAR1}_i, \text{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\text{VAR1}_i < \text{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\text{VAR1}_i = \text{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\text{VAR1}_i > \text{VAR2}_i \Leftrightarrow S_i = 3)$.

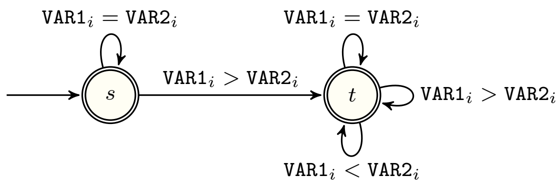


Figure 5.523: Automaton of the LEX_GREATEREQ constraint

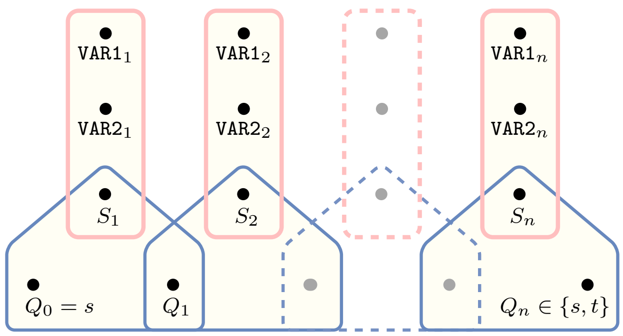


Figure 5.524: Hypergraph of the reformulation corresponding to the automaton of the LEX_GREATEREQ constraint

5.234 LEX_LESS

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint LEX_LESS(VECTOR1, VECTOR2)

Synonyms LEX, LEX_CHAIN, REL, LESS.

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

Restrictions

required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|

Purpose VECTOR1 is *lexicographically strictly less than* VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically strictly less than* $\vec{Y}$ if and only if $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is *lexicographically strictly less than* $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example $(\langle 5,2,3,9\rangle, \langle 5,2,6,2\rangle)$

5	2	3	9
=	=	<	
5	2	6	2

The LEX_LESS constraint holds since VECTOR1 $= \langle 5,2,3,9\rangle$ is lexicographically strictly less than VECTOR2 $= \langle 5,2,6,2\rangle$.

All solutions Figure 5.525 gives all solutions to the following non ground instance of the LEX_LESS constraint: $X_0 \in [0,1]$, $X_1 \in [1,2]$, $X_2 \in [4,4]$, $Y_0 \in [0,1]$, $Y_1 \in [0,2]$, $Y_2 \in [1,1]$, LEX_LESS($\langle X_0, X_1, X_2\rangle$, $\langle Y_0, Y_1, Y_2\rangle$).

Typical

$$|\texttt{VECTOR1}| > 1$$
$$\vee \left(\begin{array}{l} |\texttt{VECTOR1}| < 5, \\ \texttt{nval}([\texttt{VECTOR1.var}, \texttt{VECTOR2.var}]) < 2 * |\texttt{VECTOR1}| \end{array} \right)$$
$$\vee \left(\begin{array}{l} \texttt{maxval}([\texttt{VECTOR1.var}, \texttt{VECTOR2.var}]) \leq 1, \\ 2 * |\texttt{VECTOR1}| - \text{MAX\_NVALUE}([\texttt{VECTOR1.var}, \texttt{VECTOR2.var}]) > 2 \end{array} \right)$$

Symmetries

- VECTOR1.var can be decreased.
- VECTOR2.var can be increased.

Arg. properties Suffix-extensible wrt. VECTOR1 and VECTOR2 (*add items at same position*).

Remark A *multiset ordering* constraint and its corresponding filtering algorithm are described in [185].

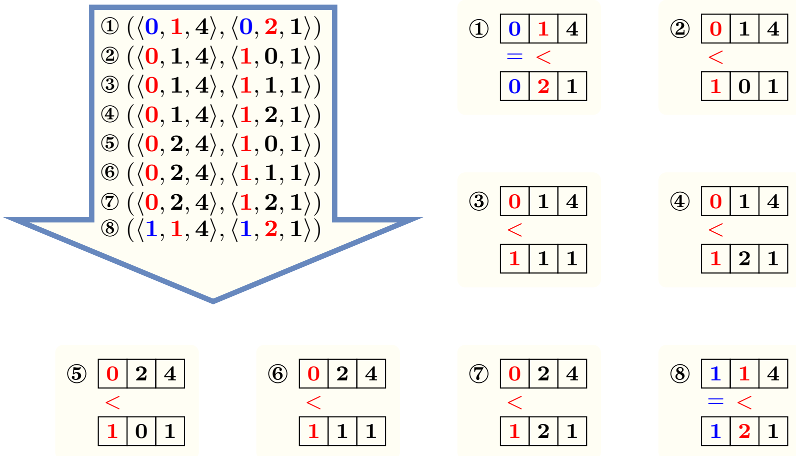


Figure 5.525: All solutions corresponding to the non ground example of the LEX_LESS constraint

Algorithm The first filtering algorithm maintaining arc-consistency for this constraint was presented in [184]. A second filtering algorithm maintaining arc-consistency and detecting entailment in a more eager way, was given in [105]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [184] detecting entailment is given in the PhD thesis of Z. Kızıltan [250, page 95]. The previous thesis [250, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence. Finally, T. Frühwirth shows how to encode lexicographic ordering constraints within the context of CHR [186] in [187].

Reformulation The following reformulations in term of arithmetic and/or logical expressions exist for enforcing the *lexicographically strictly less than* constraint. The first one converts $\vec{X}$ and $\vec{Y}$ into numbers and post an inequality constraint. It assumes all components of $\vec{X}$ and $\vec{Y}$ to be within $[0, a-1]$:

$$a^{n-1}X_0 + a^{n-2}X_1 + \cdots + a^0X_{n-1} < a^{n-1}Y_0 + a^{n-2}Y_1 + \cdots + a^0Y_{n-1}$$

Since the previous reformulation can only be used with small values of n and a, W. Harvey came up with the following alternative model that maintains arc-consistency:

$$(X_0 < Y_0 + (X_1 < Y_1 + (\cdots + (X_{n-1} < Y_{n-1} + 0)\ldots))) = 1$$

Finally, the *lexicographically strictly less than* constraint can be expressed as a conjunction or a disjunction of constraints:

$$\begin{array}{rl} X_0 \le Y_0 & \wedge \\ (X_0 = Y_0) \Rightarrow X_1 \le Y_1 & \wedge \\ (X_0 = Y_0 \wedge X_1 = Y_1) \Rightarrow X_2 \le Y_2 & \wedge \\ \vdots & \\ (X_0 = Y_0 \wedge X_1 = Y_1 \wedge \cdots \wedge X_{n-2} = Y_{n-2}) \Rightarrow X_{n-1} < Y_{n-1} & \end{array}$$

$$\begin{array}{rl} X_0 < Y_0 & \vee \\ X_0 = Y_0 \wedge X_1 < Y_1 & \vee \\ X_0 = Y_0 \wedge X_1 = Y_1 \wedge X_2 < Y_2 & \vee \\ \vdots & \\ X_0 = Y_0 \wedge X_1 = Y_1 \wedge \cdots \wedge X_{n-2} = Y_{n-2} \wedge X_{n-1} < Y_{n-1} & \end{array}$$

When used separately, the two previous logical decompositions do not maintain arc-consistency.

Systems LEX in **Choco**, REL in **Gecode**, LEX_LESS in **MiniZinc**, LEX_CHAIN in **SICStus**.

Used in LEX_CHAIN_LESS, ORDERED_ATLEAST_NVECTOR, ORDERED_ATMOST_NVECTOR, ORDERED_NVECTOR.

See also **common keyword:** COND_LEX_LESS, LEX_BETWEEN, LEX_CHAIN_GREATER, LEX_CHAIN_GREATEREQ, LEX_CHAIN_LESSEQ *(lexicographic order)*.

implies: LEX_DIFFERENT, LEX_LESSEQ.

implies (if swap arguments): LEX_GREATER.

negation: LEX_GREATEREQ.

system of constraints: LEX_CHAIN_LESS.

Keywords **characteristic of a constraint:** vector, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: constraint between two collections of variables.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: duplicated variables, arc-consistency.

heuristics: heuristics and lexicographical ordering.

symmetry: symmetry, matrix symmetry, lexicographic order, multiset ordering.

Derived Collections

$$\texttt{col}\left(\begin{array}{l}\texttt{DESTINATION}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{x}-\texttt{int},\texttt{y}-\texttt{int}),\\ [\texttt{item}(\texttt{index}-0,\texttt{x}-0,\texttt{y}-0)]\end{array}\right)$$

$$\texttt{col}\left(\begin{array}{l}\texttt{COMPONENTS}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{x}-\texttt{dvar},\texttt{y}-\texttt{dvar}),\\ \left[\texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{VECTOR1.key},\\ \texttt{x}-\texttt{VECTOR1.var},\\ \texttt{y}-\texttt{VECTOR2.var}\end{array}\right)\right]\end{array}\right)$$

Arc input(s) COMPONENTS DESTINATION

Arc generator $PRODUCT(PATH, VOID) \mapsto \texttt{collection}(\texttt{item1},\texttt{item2})$

Arc arity 2

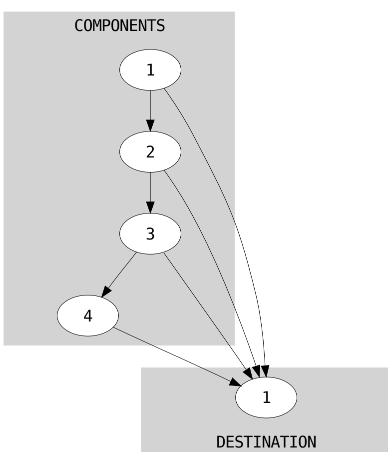
Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\texttt{item2.index} > 0 \wedge \texttt{item1.x} = \texttt{item1.y},\\ \texttt{item2.index} = 0 \wedge \texttt{item1.x} < \texttt{item1.y}\end{array}\right)$$

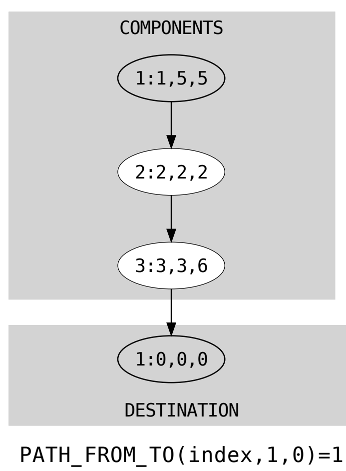
Graph property(ies) $\mathbf{PATH\_FROM\_TO}(\texttt{index},1,0) = 1$

Graph model Parts (A) and (B) of Figure 5.526 respectively show the initial and final graph associated with the **Example** slot. Since we use the **PATH_FROM_TO** graph property we show on the final graph the following information:

- The vertices, which respectively correspond to the start and the end of the required path, are stressed in bold.
- The arcs on the required path are also stressed in bold.



(A)



(B)

Figure 5.526: Initial and final graph of the LEX_LESS constraint

The vertices of the initial graph are generated in the following way:

- We create a vertex c_i for each pair of components that both have the same index i.

- We create an additional dummy vertex called d.

The arcs of the initial graph are generated in the following way:

- We create an arc between c_i and d. We associate to this arc the arc constraint $\texttt{item}_1.x < \texttt{item}_2.y$.
- We create an arc between c_i and c_{i+1}. We associate to this arc the arc constraint $\texttt{item}_1.x = \texttt{item}_2.y$.

The LEX_LESS constraint holds when there exist a path from c_1 to d. This path can be interpreted as a sequence of *equality* constraints on the prefix of both vectors, immediately followed by a *less than* constraint.

Signature

Since the maximum value returned by the graph property $\mathbf{PATH\_FROM\_TO}$ is equal to 1 we can rewrite $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) = 1$ to $\mathbf{PATH\_FROM\_TO}(\texttt{index}, 1, 0) \geq 1$. Therefore we simplify $\underline{\overline{\mathbf{PATH\_FROM\_TO}}}$ to $\overline{\mathbf{PATH\_FROM\_TO}}$.

Automaton Figure 5.527 depicts the automaton associated with the LEX_LESS constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the var attributes of the i^{th} items of the VECTOR1 and the VECTOR2 collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\texttt{VAR1}_i < \texttt{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\texttt{VAR1}_i > \texttt{VAR2}_i \Leftrightarrow S_i = 3)$.

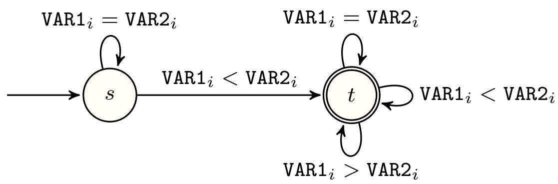


Figure 5.527: Automaton of the LEX_LESS constraint

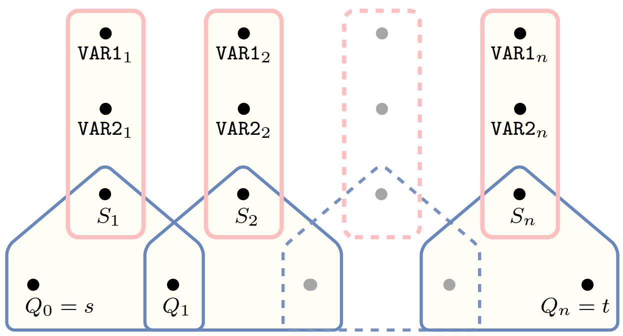


Figure 5.528: Hypergraph of the reformulation corresponding to the automaton of the LEX_LESS constraint

5.235 LEX_LESSEQ

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint LEX_LESSEQ(VECTOR1, VECTOR2)

Synonyms LEXEQ, LEX_CHAIN, REL, LESSEQ, LEQ, LEX_LEQ.

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

Restrictions

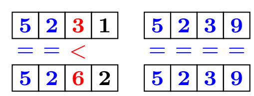
required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|

Purpose

VECTOR1 is *lexicographically less than or equal to* VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically less than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is *lexicographically less than or equal to* $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$(\langle 5, 2, 3, 1\rangle, \langle 5, 2, 6, 2\rangle)$
$(\langle 5, 2, 3, 9\rangle, \langle 5, 2, 3, 9\rangle)$



The LEX_LESSEQ constraints associated with the first and second examples hold since:

- Within the first example VECTOR1 $= \langle 5, 2, 3, 1\rangle$ is lexicographically less than or equal to VECTOR2 $= \langle 5, 2, 6, 2\rangle$.
- Within the second example VECTOR1 $= \langle 5, 2, 3, 9\rangle$ is lexicographically less than or equal to VECTOR2 $= \langle 5, 2, 3, 9\rangle$.

All solutions

Figure 5.529 gives all solutions to the following non ground instance of the LEX_LESSEQ constraint: $X_0 \in [1, 2]$, $X_1 \in [1, 2]$, $X_2 \in [1, 2]$, $Y_0 \in [0, 1]$, $Y_1 \in [0, 2]$, $Y_2 \in [1, 1]$, LEX_LESSEQ$(\langle X_0, X_1, X_2\rangle, \langle Y_0, Y_1, Y_2\rangle)$.

Typical

$$|\mathtt{VECTOR1}| > 1$$
$$\vee \left(\begin{array}{l}|\mathtt{VECTOR1}| < 5,\\ \mathrm{nval}([\mathtt{VECTOR1.var}, \mathtt{VECTOR2.var}]) < 2 * |\mathtt{VECTOR1}|\end{array}\right)$$
$$\vee \left(\begin{array}{l}\mathrm{maxval}([\mathtt{VECTOR1.var}, \mathtt{VECTOR2.var}]) \leq 1,\\ 2 * |\mathtt{VECTOR1}| - \mathrm{MAX\_NVALUE}([\mathtt{VECTOR1.var}, \mathtt{VECTOR2.var}]) > 2\end{array}\right)$$

Symmetries

- VECTOR1.var can be decreased.
- VECTOR2.var can be increased.

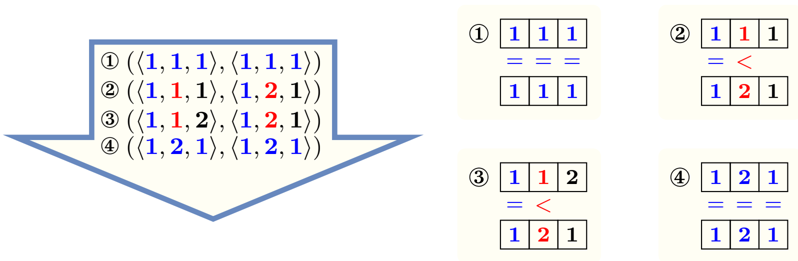


Figure 5.529: All solutions corresponding to the non ground example of the LEX_LESSEQ constraint

Arg. properties

Suffix-contractible wrt. VECTOR1 and VECTOR2 (*remove items from same position*).

Remark

A *multiset ordering* constraint and its corresponding filtering algorithm are described in [185].

Algorithm

The first filtering algorithm maintaining arc-consistency for this constraint was presented in [184]. A second filtering algorithm maintaining arc-consistency and detecting entailment in a more eager way, was given in [105]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [184] detecting entailment is given in the PhD thesis of Z. Kızıltan [250, page 95]. The previous thesis [250, pages 105–109] presents also a filtering algorithm handling the fact that a given variable has more than one occurrence. Finally, T. Frühwirth shows how to encode lexicographic ordering constraints within the context of CHR [186] in [187].

Reformulation

The following reformulations in term of arithmetic and/or logical expressions exist for enforcing the *lexicographically less than or equal to* constraint. The first one converts $\vec{X}$ and $\vec{Y}$ into numbers and post an inequality constraint. It assumes all components of $\vec{X}$ and $\vec{Y}$ to be within $[0, a-1]$:

$$a^{n-1}X_0 + a^{n-2}X_1 + \cdots + a^0 X_{n-1} \leq a^{n-1}Y_0 + a^{n-2}Y_1 + \cdots + a^0 Y_{n-1}$$

Since the previous reformulation can only be used with small values of n and a, W. Harvey came up with the following alternative model that maintains arc-consistency:

$$(X_0 < Y_0 + (X_1 < Y_1 + (\cdots + (X_{n-1} < Y_{n-1} + 1)\ldots))) = 1$$

Finally, the *lexicographically less than or equal to* constraint can be expressed as a conjunction or a disjunction of constraints:

$$\begin{array}{rl} X_0 \leq Y_0 & \wedge \\ (X_0 = Y_0) \Rightarrow X_1 \leq Y_1 & \wedge \\ (X_0 = Y_0 \wedge X_1 = Y_1) \Rightarrow X_2 \leq Y_2 & \wedge \\ \vdots & \\ (X_0 = Y_0 \wedge X_1 = Y_1 \wedge \cdots \wedge X_{n-2} = Y_{n-2}) \Rightarrow X_{n-1} \leq Y_{n-1} & \end{array}$$

$$\begin{array}{rl} X_0 < Y_0 & \vee \\ X_0 = Y_0 \wedge X_1 < Y_1 & \vee \\ X_0 = Y_0 \wedge X_1 = Y_1 \wedge X_2 < Y_2 & \vee \\ \vdots & \\ X_0 = Y_0 \wedge X_1 = Y_1 \wedge \cdots \wedge X_{n-2} = Y_{n-2} \wedge X_{n-1} \leq Y_{n-1} & \end{array}$$

When used separately, the two previous logical decompositions do not maintain arc-consistency.

Systems LEXEQ in **Choco**, REL in **Gecode**, LEX_LESSEQ in **MiniZinc**, LEX_CHAIN in **SICStus**.

Used in LEX_BETWEEN, LEX_CHAIN_GREATEREQ, LEX_CHAIN_LESSEQ, ORDERED_ATLEAST_NVECTOR, ORDERED_ATMOST_NVECTOR, ORDERED_NVECTOR.

See also **common keyword:** ALLPERM, COND_LEX_LESSEQ *(lexicographic order)*, LEX2 *(matrix symmetry,lexicographic order)*, LEX_CHAIN_GREATER, LEX_CHAIN_GREATEREQ, LEX_CHAIN_LESS *(lexicographic order)*, LEX_DIFFERENT *(vector)*, STRICT_LEX2 *(matrix symmetry,lexicographic order)*.

implied by: LEX_EQUAL, LEX_LESS, LEX_LESSEQ_ALLPERM.

implies (if swap arguments): LEX_GREATEREQ.

negation: LEX_GREATER.

system of constraints: LEX_BETWEEN, LEX_CHAIN_LESSEQ.

Keywords **characteristic of a constraint:** vector, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: constraint between two collections of variables.

constraint network structure: Berge-acyclic constraint network.

constraint type: order constraint.

filtering: duplicated variables, arc-consistency.

heuristics: heuristics and lexicographical ordering.

symmetry: symmetry, matrix symmetry, lexicographic order, multiset ordering.

Derived Collections

$$\texttt{col}\left(\begin{array}{l}\texttt{DESTINATION}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{x}-\texttt{int},\texttt{y}-\texttt{int}),\\ [\texttt{item}(\texttt{index}-0,\texttt{x}-0,\texttt{y}-0)]\end{array}\right)$$

$$\texttt{col}\left(\begin{array}{l}\texttt{COMPONENTS}-\texttt{collection}(\texttt{index}-\texttt{int},\texttt{x}-\texttt{dvar},\texttt{y}-\texttt{dvar}),\\ \left[\texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{VECTOR1.key},\\ \texttt{x}-\texttt{VECTOR1.var},\\ \texttt{y}-\texttt{VECTOR2.var}\end{array}\right)\right]\end{array}\right)$$

Arc input(s) COMPONENTS DESTINATION

Arc generator $PRODUCT(PATH,VOID)\mapsto\texttt{collection}(\texttt{item1},\texttt{item2})$

Arc arity 2

Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\texttt{item2.index}>0\wedge\texttt{item1.x}=\texttt{item1.y},\\ \bigwedge\left(\begin{array}{l}\texttt{item1.index}<|\texttt{VECTOR1}|,\\ \texttt{item2.index}=0,\\ \texttt{item1.x}<\texttt{item1.y}\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{item1.index}=|\texttt{VECTOR1}|,\\ \texttt{item2.index}=0,\\ \texttt{item1.x}\leq\texttt{item1.y}\end{array}\right)\end{array}\right)$$

Graph property(ies) $\mathbf{PATH\_FROM\_TO}(\texttt{index},1,0)=1$

Graph model

Parts (A) and (B) of Figure 5.530 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **PATH_FROM_TO** graph property we show on the final graph the following information:

- The vertices, which respectively correspond to the start and the end of the required path, are stressed in bold.
- The arcs on the required path are also stressed in bold.

The vertices of the initial graph are generated in the following way:

- We create a vertex c_i for each pair of components that both have the same index i.
- We create an additional dummy vertex called d.

The arcs of the initial graph are generated in the following way:

- We create an arc between c_i and d. When c_i was generated from the last components of both vectors We associate to this arc the arc constraint $\texttt{item}_1.x \leq \texttt{item}_2.y$; Otherwise we associate to this arc the arc constraint $\texttt{item}_1.x < \texttt{item}_2.y$;
- We create an arc between c_i and c_{i+1}. We associate to this arc the arc constraint $\texttt{item}_1.x = \texttt{item}_2.y$.

The LEX_LESSEQ constraint holds when there exist a path from c_1 to d. This path can be interpreted as a maximum sequence of *equality* constraints on the prefix of both vectors, possibly followed by a *less than* constraint.

Signature

Since the maximum value returned by the graph property **PATH_FROM_TO** is equal to 1 we can rewrite $\mathbf{PATH\_FROM\_TO}(\texttt{index},1,0)=1$ to $\mathbf{PATH\_FROM\_TO}(\texttt{index},1,0)\geq 1$. Therefore we simplify $\underline{\overline{\mathbf{PATH\_FROM\_TO}}}$ to $\overline{\mathbf{PATH\_FROM\_TO}}$.

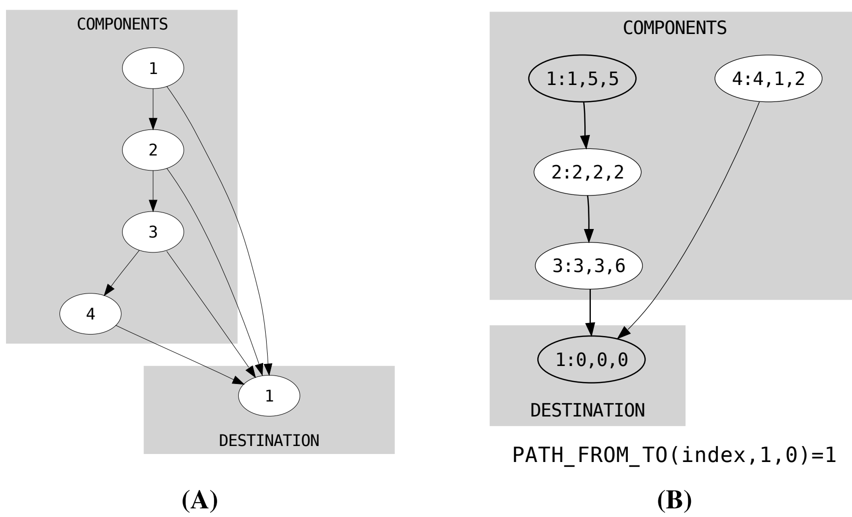


Figure 5.530: Initial and final graph of the LEX_LESSEQ constraint

Automaton Figure 5.531 depicts the automaton associated with the LEX_LESSEQ constraint. Let $\texttt{VAR1}_i$ and $\texttt{VAR2}_i$ respectively be the var attributes of the i^{th} items of the VECTOR1 and the VECTOR2 collections. To each pair $(\texttt{VAR1}_i, \texttt{VAR2}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\texttt{VAR1}_i < \texttt{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\texttt{VAR1}_i > \texttt{VAR2}_i \Leftrightarrow S_i = 3)$.

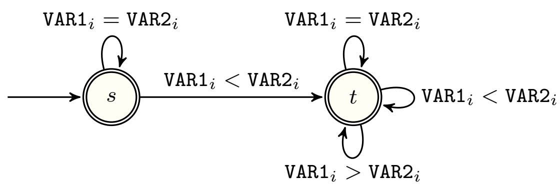


Figure 5.531: Automaton of the LEX_LESSEQ constraint

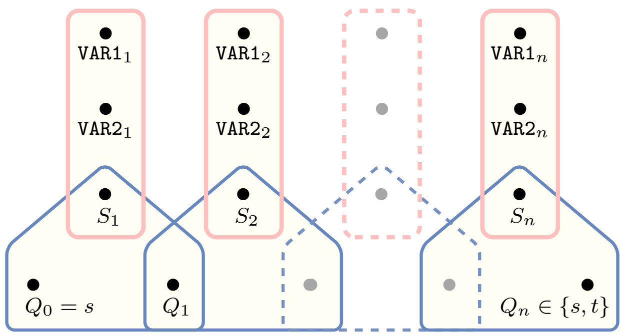


Figure 5.532: Hypergraph of the reformulation corresponding to the automaton of the LEX_LESSEQ constraint

5.236 LEX_LESSEQ_ALLPERM

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Inspired by [179]

Constraint LEX_LESSEQ_ALLPERM(VECTOR1, VECTOR2)

Synonym LEXIMIN.

Arguments

VECTOR1 : collection(var−dvar)
VECTOR2 : collection(var−dvar)

Restrictions

required(VECTOR1, var)
required(VECTOR2, var)
|VECTOR1| = |VECTOR2|

Purpose VECTOR1 is *lexicographically less than or equal to* all permutations of VECTOR2. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically less than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is *lexicographically less than or equal to* $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example $(\langle 1,2,3\rangle, \langle 3,1,2\rangle)$

1 2 3	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3
<	<	= <	= = =	<	<
3 1 2	3 2 1	1 3 2	1 2 3	2 1 3	2 3 1

The LEX_LESSEQ_ALLPERM constraint holds since vector $\langle 1,2,3\rangle$ is lexicographically less than or equal to all the permutations of vector $\langle 3,1,2\rangle$ (i.e., $\langle 1,2,3\rangle$, $\langle 1,3,2\rangle$, $\langle 2,1,3\rangle$, $\langle 2,3,1\rangle$, $\langle 3,1,2\rangle$, $\langle 3,2,1\rangle$).

Typical |VECTOR1| > 1

Symmetry All occurrences of two distinct values in VECTOR1.var or VECTOR2.var can be swapped; all occurrences of a value in VECTOR1.var or VECTOR2.var can be renamed to any unused value.

Arg. properties Suffix-contractible wrt. VECTOR1 and VECTOR2 (*remove items from same position*).

Remark The LEX_LESSEQ_ALLPERM(VECTOR1, VECTOR2) can be reformulated as the conjunction SORT(VECTOR2, VECTOR), LEX_LESSEQ(VECTOR1, VECTOR).

Systems LEXIMIN in **Choco**.

Used in ALLPERM.

See also **common keyword:** ALLPERM (*matrix symmetry*, *lexicographic order*).

implies: LEX_LESSEQ.

system of constraints: ALLPERM.

Keywords

characteristic of a constraint: vector.

constraint arguments: constraint between two collections of variables.

constraint type: predefined constraint, order constraint.

symmetry: symmetry, matrix symmetry, lexicographic order.

5.237 LINK_SET_TO_BOOLEANS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Inspired by DOMAIN_CONSTRAINT.

Constraint LINK_SET_TO_BOOLEANS(SVAR, BOOLEANS)

Arguments

SVAR : svar
BOOLEANS : collection(bool−dvar, val−int)

Restrictions

required(BOOLEANS, [bool, val])
BOOLEANS.bool ≥ 0
BOOLEANS.bool ≤ 1
distinct(BOOLEANS, val)

Purpose Make the link between a set variable SVAR and those 0-1 variables that are associated with each potential value belonging to SVAR: The 0-1 variables, which are associated with a value belonging to the set variable SVAR, are equal to 1, while the remaining 0-1 variables are all equal to 0.

Example

$$\left(\begin{array}{l}\{1,3,4\},\\ \left\langle\begin{array}{ll}\texttt{bool}-0 & \texttt{val}-0,\\ \texttt{bool}-1 & \texttt{val}-1,\\ \texttt{bool}-0 & \texttt{val}-2,\\ \texttt{bool}-1 & \texttt{val}-3,\\ \texttt{bool}-1 & \texttt{val}-4,\\ \texttt{bool}-0 & \texttt{val}-5\end{array}\right\rangle\end{array}\right)$$

In the example, the 0-1 variables associated with the values 1, 3 and 4 are all set to 1, while the other 0-1 variables are set to 0. Consequently, the LINK_SET_TO_BOOLEANS constraint holds since its first argument SVAR is set to $\{1,3,4\}$.

Typical

$|\texttt{BOOLEANS}| > 1$
range(BOOLEANS.bool) > 1

Symmetry Items of BOOLEANS are permutable.

Usage This constraint is used in order to make the link between a formulation using set variables and a formulation based on linear programming.

Systems CHANNEL in **Gecode**, LINK_SET_TO_BOOLEANS in **MiniZinc**.

See also **common keyword:** ALLDIFFERENT_BETWEEN_SETS, CLIQUE *(constraint involving set variables)*, DOMAIN_CONSTRAINT *(channelling constraint)*, K_CUT, PATH_FROM_TO, ROOTS, STRONGLY_CONNECTED, SYMMETRIC_CARDINALITY, SYMMETRIC_GCC, TOUR *(constraint involving set variables)*.

Keywords

characteristic of a constraint: derived collection.

constraint arguments: constraint involving set variables.

constraint type: decomposition, value constraint.

filtering: linear programming.

modelling: channelling constraint, set channel.

Derived Collection

$$\mathtt{col}\left(\begin{array}{l}\mathtt{SET}-\mathtt{collection}(\mathtt{one}-\mathtt{int},\mathtt{setvar}-\mathtt{svar}),\\ [\mathtt{item}(\mathtt{one}-1,\mathtt{setvar}-\mathtt{SVAR})]\end{array}\right)$$

Arc input(s)	SET BOOLEANS
Arc generator	$PRODUCT \mapsto$ collection(set, booleans)
Arc arity	2
Arc constraint(s)	booleans.bool = set.one $\Leftrightarrow$ IN_SET(booleans.val, set.setvar)
Graph property(ies)	**NARC** = |BOOLEANS|

Graph model

The LINK_SET_TO_BOOLEANS constraint is modelled with the following bipartite graph. The first set of vertices corresponds to a single vertex containing the set variable. The second class of vertices contains one vertex for each item of the collection BOOLEANS. The arc constraint between the set variable SVAR and one potential value v of the set variable expresses the following:

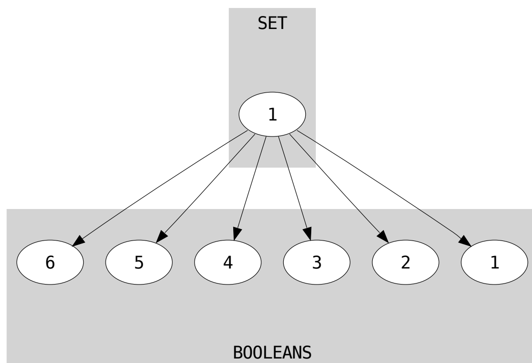
- If the 0-1 variable associated with v is equal to 1 then v should belong to SVAR.
- Otherwise if the 0-1 variable associated with v is equal to 0 then v should not belong to SVAR.

Since all arc constraints should hold the final graph contains exactly |BOOLEANS| arcs.

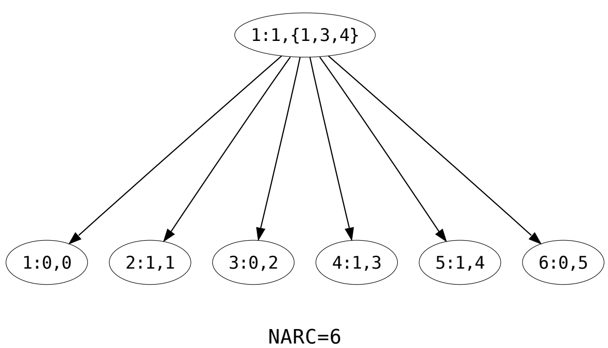
Parts (A) and (B) of Figure 5.533 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The LINK_SET_TO_BOOLEANS constraint holds since the final graph contains exactly 6 arcs (one for each 0-1 variable).

Signature

Since the initial graph contains |BOOLEANS| arcs the maximum number of arcs of the final graph is equal to |BOOLEANS|. Therefore we can rewrite the graph property **NARC** = |BOOLEANS| to **NARC** $\geq$ |BOOLEANS| and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.



(A)



(B)

Figure 5.533: Initial and final graph of the LINK_SET_TO_BOOLEANS constraint

5.238 LONGEST_CHANGE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin

Derived from CHANGE.

Constraint

LONGEST_CHANGE(SIZE, VARIABLES, CTR)

Arguments

SIZE : dvar
VARIABLES : collection(var−dvar)
CTR : atom

Restrictions

SIZE ≥ 0
SIZE $\leq$ |VARIABLES|
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

SIZE is the maximum number of consecutive variables of the collection VARIABLES for which constraint CTR holds in an uninterrupted way (0 if the constraint CTR does not hold at all). We count a change when X CTR Y holds; X and Y are two consecutive variables of the collection VARIABLES.

Example

$(4, \langle 8, 8, 3, 4, 1, 1, 5, 5, 2\rangle, \neq)$

The LONGEST_CHANGE constraint holds since its first argument SIZE $= 4$ is fixed to the length of the longest subsequence of consecutive values of the collection $\langle 8, 8, 3, 4, 1, 1, 5, 5, 2\rangle$ such that two consecutive values are distinct (i.e., subsequence 8 3 4 1).

Typical

|VARIABLES| > 2
range(VARIABLES.var) > 1
CTR $\in [\neq]$

Symmetry

One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Functional dependency: SIZE determined by VARIABLES and CTR.

See also

root concept: CHANGE.

Keywords

characteristic of a constraint: automaton, automaton with counters.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(3).

constraint type: timetabling constraint.

filtering: glue matrix.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var CTR variables2.var
Graph property(ies)	$\mathbf{MAX\_NCC}$= SIZE

Graph model In order to specify the LONGEST_CHANGE constraint, we use $\mathbf{MAX\_NCC}$, which is the number of vertices of the largest connected component. Since the initial graph corresponds to a path, this will be the length of the longest path in the final graph.

Parts (A) and (B) of Figure 5.534 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{MAX\_NCC}$ graph property we show the largest connected component of the final graph. It corresponds to the longest period of uninterrupted changes: sequence $8, 3, 4, 1$ that involves 4 consecutive variables.

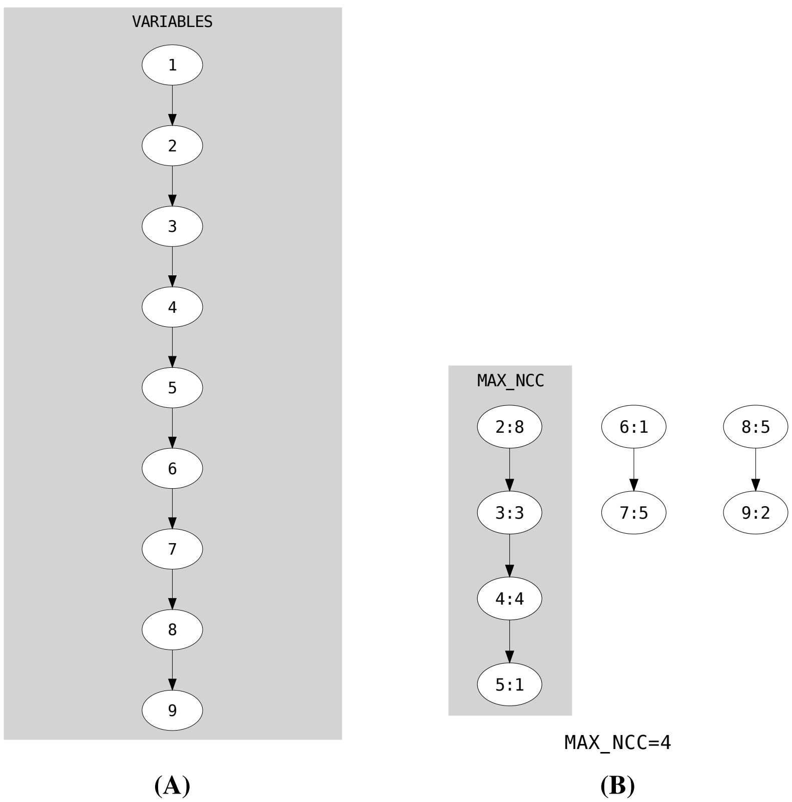
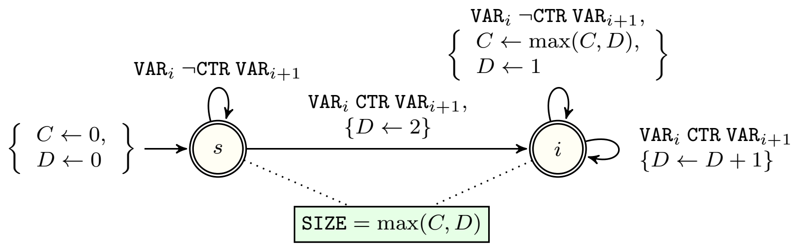


Figure 5.534: Initial and final graph of the LONGEST_CHANGE constraint

Automaton Figure 5.535 depicts the automaton associated with the LONGEST_CHANGE constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $\mathtt{VAR}_i$ CTR $\mathtt{VAR}_{i+1} \Leftrightarrow S_i$.



Glue matrix where $\overrightarrow{C}$, $\overrightarrow{D}$ and $\overleftarrow{C}$, $\overleftarrow{D}$ resp. represent the counters values C, D at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	i
s	0	$\max\left(\overleftarrow{D}, \overleftarrow{C}\right)$
i	$\max\left(\overrightarrow{C}, \overrightarrow{D}\right)$	$\max\left(\overrightarrow{C}, \overrightarrow{D} + \overleftarrow{D} - 1, \overleftarrow{C}\right)$

Figure 5.535: Automaton of the LONGEST_CHANGE constraint and its glue matrix

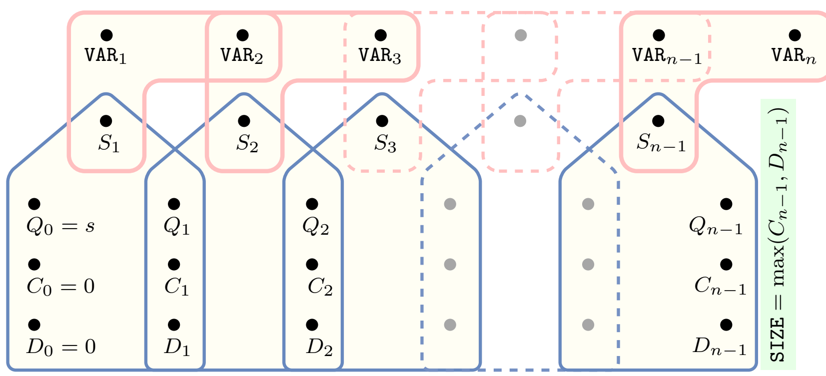


Figure 5.536: Hypergraph of the reformulation corresponding to the automaton of the LONGEST_CHANGE constraint

5.239 LONGEST_DECREASING_SEQUENCE

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin constraint on sequences

Constraint LONGEST_DECREASING_SEQUENCE(L, VARIABLES)

Synonym SIZE_LONGEST_DECREASING_SEQUENCE.

Arguments

L	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

L ≥ 0
L $<$range(VARIABLES.var)
required(VARIABLES, var)

Purpose

L is the largest difference between the first and the last value of the maximum decreasing sequences of the collection VARIABLES.

A sequence of consecutive variables $X_i, X_{i+1}, \ldots, X_j$ ($1 \leq i \leq j \leq |$VARIABLES$|$) of the collection of variables VARIABLES is a *maximum decreasing sequence* if all the following conditions simultaneously apply:

- $X_i \geq X_{i+1} \geq \cdots \geq X_j$,
- $i = 1$ or $X_{i-1} < X_i$,
- $i = |$VARIABLES$|$ or $X_j < X_{j+1}$.

Example

$(0, \langle 0, 1, 2, 5\rangle)$
$(0, \langle 8, 8\rangle)$
$(6, \langle 10, 8, 8, 6, 4, 9, 10, 8\rangle)$

Figure 5.537 gives a graphical representation of the third example of the **Example** slot with its two maximum decreasing sequences in red of respective size 6 and 2. The corresponding LONGEST_DECREASING_SEQUENCE constraint holds since its first argument L is fixed to the maximum size 6.

Typical

L > 0
$|$VARIABLES$| > 1$
nval(VARIABLES.var) > 2

Typical model nval(VARIABLES.var) > 2

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: L determined by VARIABLES.

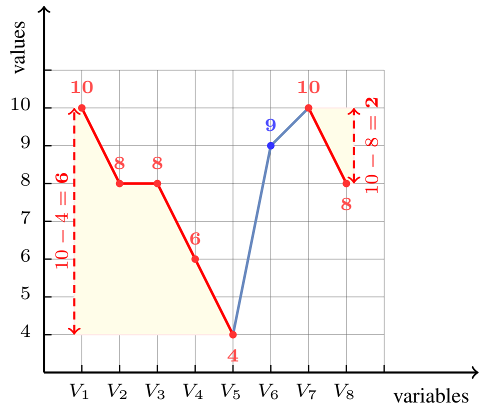
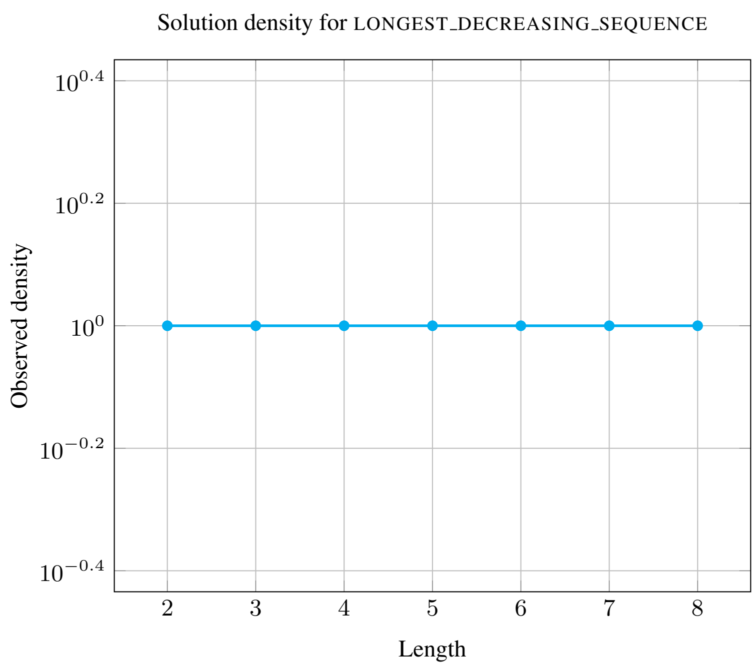


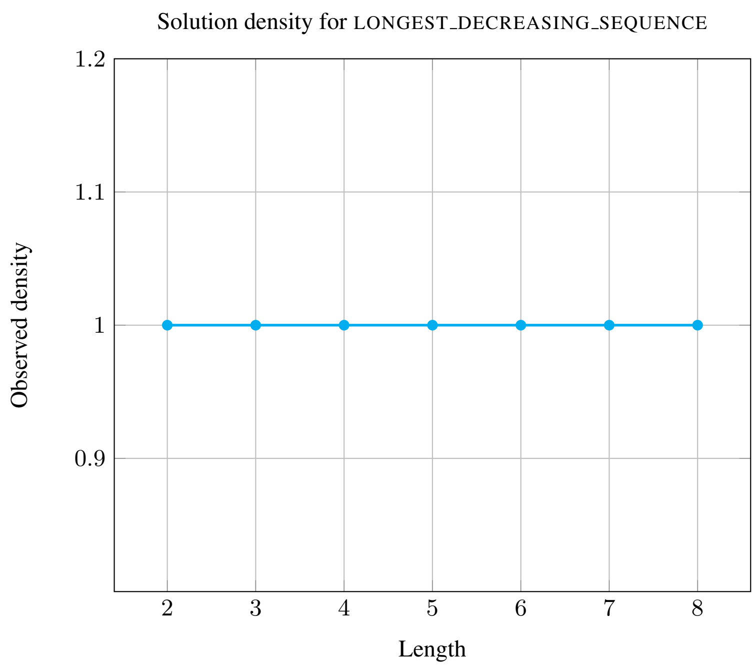
Figure 5.537: Illustration of the third example of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values 10, 8, 8, 6, 4, 9, 10, 8 and its two maximum decreasing sequences in red of respective size $10-4=6$ and $10-8=2$.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for LONGEST_DECREASING_SEQUENCE: domains $0..n$

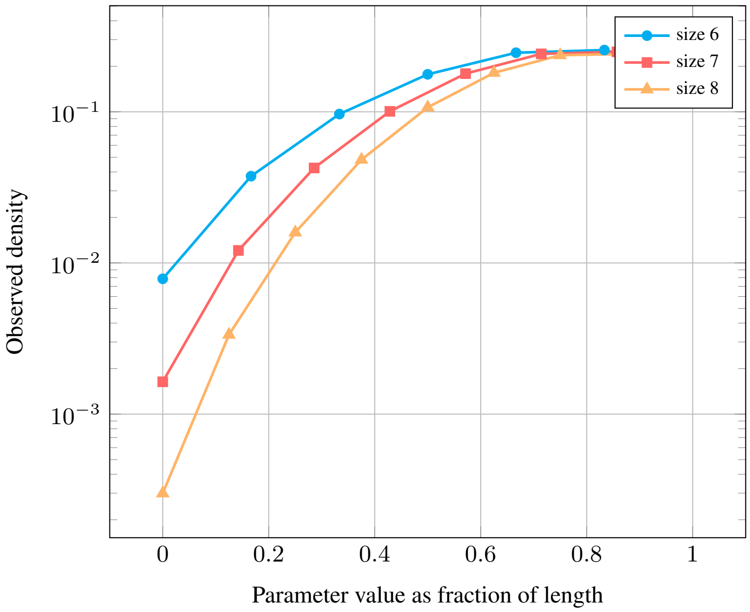




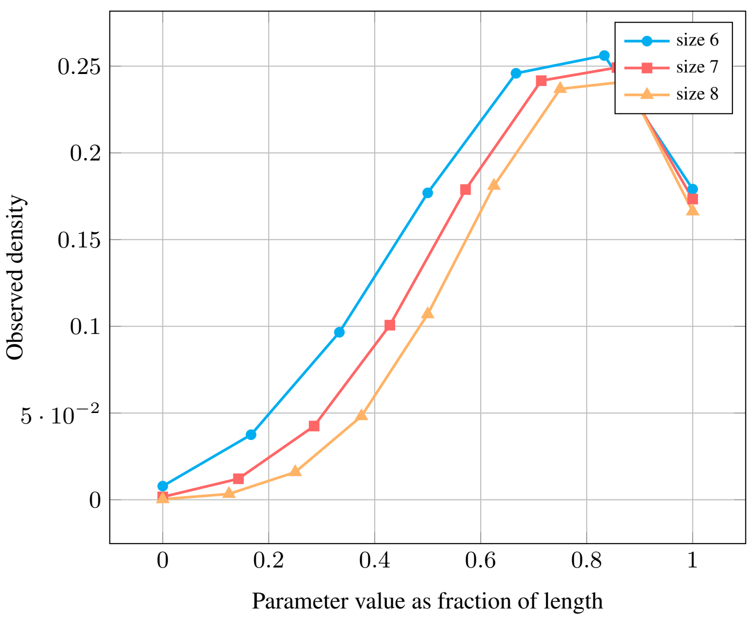
Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	6	20	70	252	924	3432	12870
	1	2	18	122	750	4412	25382	144314
	2	1	16	161	1398	11361	89132	685090
	3	-	10	162	1942	20816	211106	2074365
	4	-	-	110	2024	28930	375084	4603682
	5	-	-	-	1410	30134	506766	7792840
	6	-	-	-	-	21072	522648	10197174
	7	-	-	-	-	-	363602	10379696
	8	-	-	-	-	-	-	7156690

Solution count for LONGEST_DECREASING_SEQUENCE: domains $0..n$

Solution density for LONGEST_DECREASING_SEQUENCE



Solution density for LONGEST_DECREASING_SEQUENCE



See also **common keyword:** LONGEST_INCREASING_SEQUENCE, MIN_DIST_BETWEEN_INFLEXION *(sequence)*.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

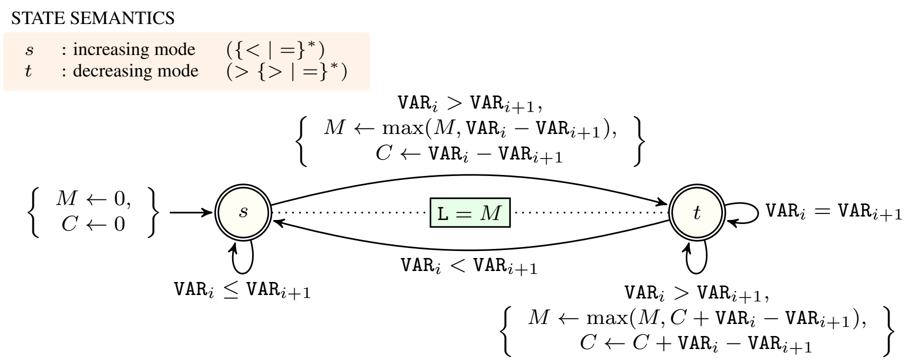
combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

filtering: glue matrix.

modelling: functional dependency.

Automaton Figure 5.538 depicts the automaton associated with the LONGEST_DECREASING_SEQUENCE constraint.



Glue matrix where $\overrightarrow{M}$, $\overrightarrow{C}$ and $\overleftarrow{M}$, $\overleftarrow{C}$ resp. represent the counters values M, C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	$s\ (\{> \mid =\}^*)$	$t\ (< \{< \mid =\}^*)$
$s\ (\{< \mid =\}^*)$	$\max(\overrightarrow{M}, \overleftarrow{M})$	$\max(\overrightarrow{M}, \overleftarrow{M})$
$t\ (> \{> \mid =\}^*)$	$\max(\overrightarrow{M}, \overleftarrow{M})$	$\max(\overrightarrow{M}, \overrightarrow{C} + \overleftarrow{C}, \overleftarrow{M})$

Figure 5.538: Automaton of the LONGEST_DECREASING_SEQUENCE constraint and its glue matrix (note that the reverse of the LONGEST_DECREASING_SEQUENCE constraint is the LONGEST_INCREASING_SEQUENCE constraint)

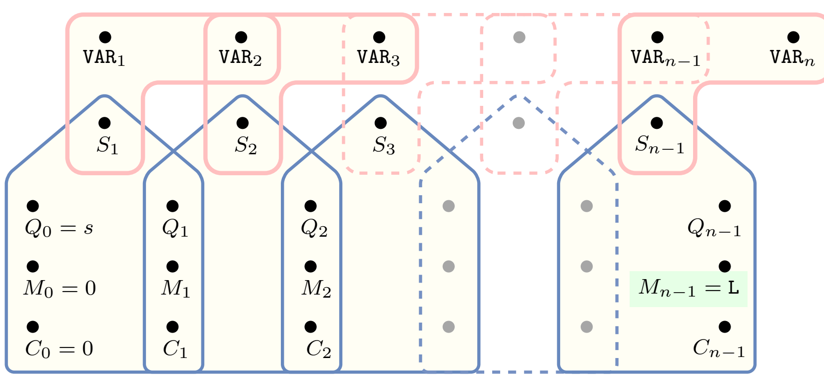


Figure 5.539: Hypergraph of the reformulation corresponding to the automaton of the LONGEST_DECREASING_SEQUENCE constraint

5.240 LONGEST_INCREASING_SEQUENCE

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin constraint on sequences

Constraint LONGEST_INCREASING_SEQUENCE(L, VARIABLES)

Synonym SIZE_LONGEST_INCREASING_SEQUENCE.

Arguments

L	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

L ≥ 0
L $<$range(VARIABLES.var)
required(VARIABLES, var)

Purpose

L is the largest difference between the first and the last value of the maximum increasing sequences of the collection VARIABLES.
A sequence of consecutive variables $X_i, X_{i+1}, \ldots, X_j$ ($1 \leq i \leq j \leq |\text{VARIABLES}|$) of the collection of variables VARIABLES is a *maximum increasing sequence* if all the following conditions simultaneously apply:

- $X_i \leq X_{i+1} \leq \cdots \leq X_j$,
- $i = 1$ or $X_{i-1} > X_i$,
- $i = |\text{VARIABLES}|$ or $X_j > X_{j+1}$.

Example

$(7, \langle 10, 8, 8, 6, 4, 9, 11, 8 \rangle)$
$(0, \langle 10, 8, 7, 5, 4, 3, 1, 0 \rangle)$

Figure 5.540 gives a graphical representation of the first example of the **Example** slot with its two maximum increasing sequences in red of respective size 0 and 7. The corresponding LONGEST_INCREASING_SEQUENCE constraint holds since its first argument L is fixed to the maximum size 7.

Typical

L > 0
$|\text{VARIABLES}| > 1$
nval(VARIABLES.var) > 2

Typical model nval(VARIABLES.var) > 2

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: L determined by VARIABLES.

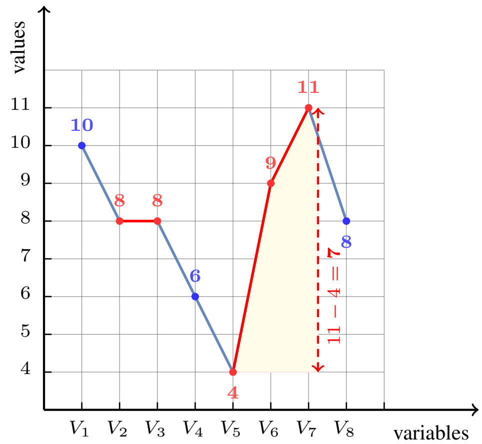
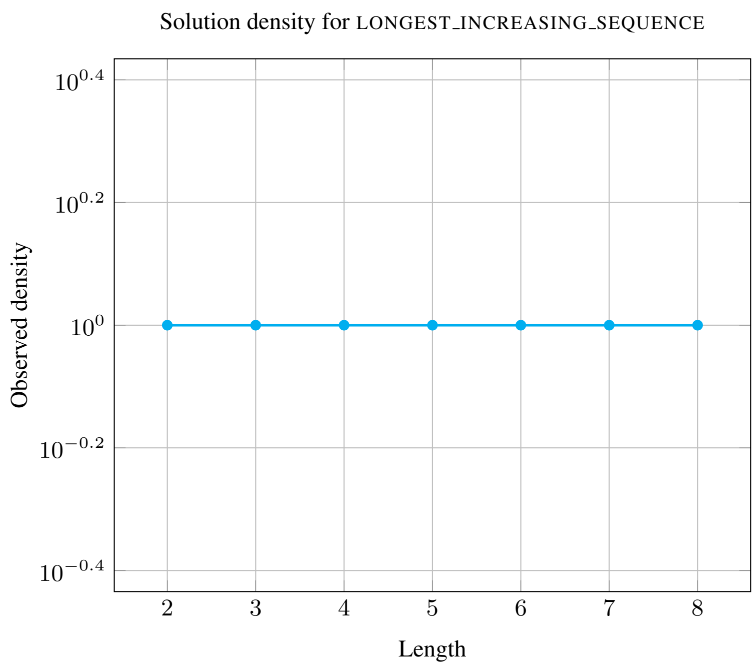


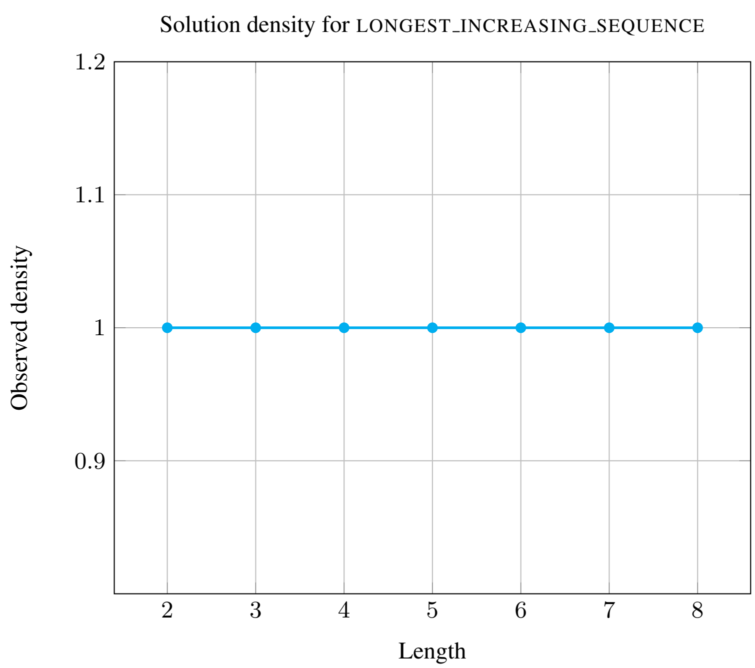
Figure 5.540: Illustration of the first example of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values 10, 8, 8, 6, 4, 9, 11, 8 and its two maximum increasing sequences in red of respective size $8 - 8 = 0$ and $11 - 4 = 7$.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

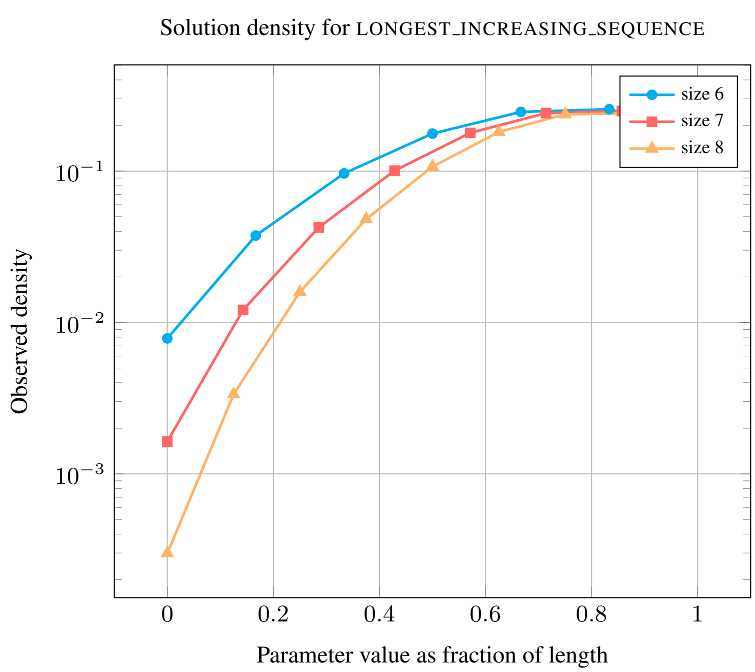
Number of solutions for LONGEST_INCREASING_SEQUENCE: domains $0..n$

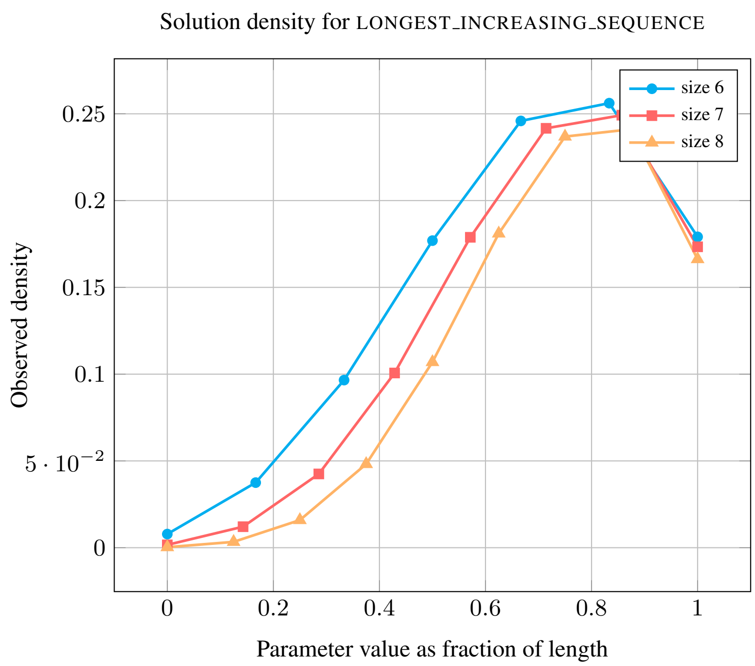

Solution density for LONGEST_INCREASING_SEQUENCE
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
Observed density
2 3 4 5 6 7 8
Length


Solution density for LONGEST_INCREASING_SEQUENCE
1.2
1.1
1
0.9
Observed density
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	6	20	70	252	924	3432	12870
	1	2	18	122	750	4412	25382	144314
	2	1	16	161	1398	11361	89132	685090
	3	-	10	162	1942	20816	211106	2074365
	4	-	-	110	2024	28930	375084	4603682
	5	-	-	-	1410	30134	506766	7792840
	6	-	-	-	-	21072	522648	10197174
	7	-	-	-	-	-	363602	10379696
	8	-	-	-	-	-	-	7156690

Solution count for LONGEST_INCREASING_SEQUENCE: domains $0..n$





See also **common keyword:** LONGEST_DECREASING_SEQUENCE, MIN_DIST_BETWEEN_INFLEXION *(sequence)*.

Keywords **characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

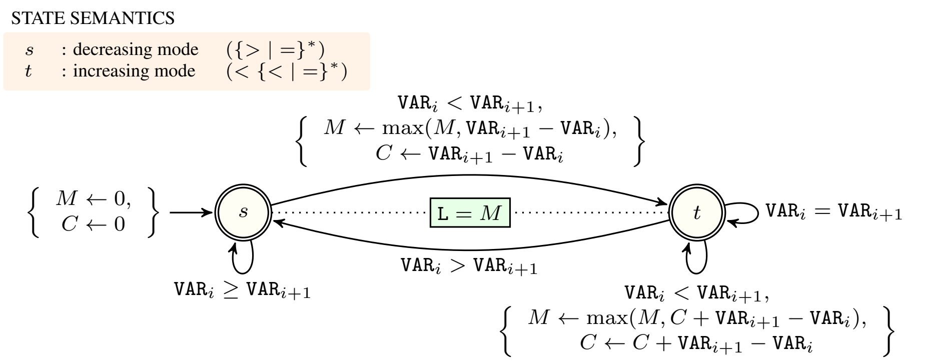
combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

filtering: glue matrix.

modelling: functional dependency.

Automaton Figure 5.541 depicts the automaton associated with the LONGEST_INCREASING_SEQUENCE constraint.



Glue matrix where $\overrightarrow{M}$, $\overrightarrow{C}$ and $\overleftarrow{M}$, $\overleftarrow{C}$ resp. represent the counters values M, C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s $(\{< \mid =\}^*)$	t $(> \{> \mid =\}^*)$
s $(\{> \mid =\}^*)$	$\max(\overrightarrow{M}, \overleftarrow{M})$	$\max(\overrightarrow{M}, \overleftarrow{M})$
t $(< \{< \mid =\}^*)$	$\max(\overrightarrow{M}, \overleftarrow{M})$	$\max(\overrightarrow{M}, \overrightarrow{C} + \overleftarrow{C}, \overleftarrow{M})$

Figure 5.541: Automaton of the LONGEST_INCREASING_SEQUENCE constraint and its glue matrix (note that the reverse of the LONGEST_INCREASING_SEQUENCE constraint is the LONGEST_DECREASING_SEQUENCE constraint)

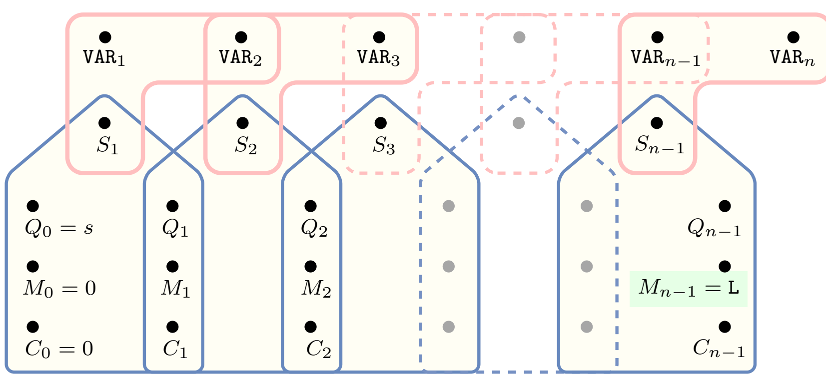


Figure 5.542: Hypergraph of the reformulation corresponding to the automaton of the LONGEST_INCREASING_SEQUENCE constraint

5.241 LT

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint LT(VAR1, VAR2)

Synonyms REL, XLTY.

Arguments

```
VAR1 : dvar
VAR2 : dvar
```

Purpose Enforce the fact that the first variable is strictly less than the second variable.

Example $(1, 8)$

The LT constraint holds since 1 is strictly less than 8.

Symmetries

- VAR1 can be replaced by any value $<$ VAR2.
- VAR2 can be replaced by any value $>$ VAR1.

Systems LT in **Choco**, REL in **Gecode**, XLTY in **JaCoP**, #< in **SICStus**.

See also **common keyword:** EQ *(binary constraint,arithmetic constraint)*.

implies: LEQ, NEQ.

implies (if swap arguments): GT.

negation: GEQ.

Keywords **constraint arguments:** binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.242 MAP

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Inspired by [387]

Constraint MAP(NBCYCLE, NBTREE, NODES)

Arguments

```
NBCYCLE : dvar
NBTREE  : dvar
NODES   : collection(index-int, succ-dvar)
```

Restrictions

NBCYCLE ≥ 0
NBTREE ≥ 0
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

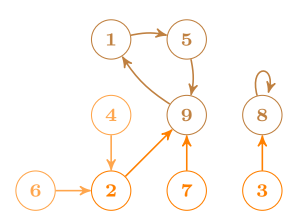
Purpose

Number of trees and number of cycles of a map. We take the description of a map from [387, page 459]:

> *"Every map decomposes into a set of connected components, also called connected maps. Each component consists of the set of all points that wind up on the same cycle, with each point on the cycle attached to a tree of all points that enter the cycle at that point."*

Example

$$\left(\begin{array}{l} 2,3,\left\langle\begin{array}{ll} \text{index}-1 & \text{succ}-5, \\ \text{index}-2 & \text{succ}-9, \\ \text{index}-3 & \text{succ}-8, \\ \text{index}-4 & \text{succ}-2, \\ \text{index}-5 & \text{succ}-9, \\ \text{index}-6 & \text{succ}-2, \\ \text{index}-7 & \text{succ}-9, \\ \text{index}-8 & \text{succ}-8, \\ \text{index}-9 & \text{succ}-1 \end{array}\right\rangle \end{array}\right)$$



The MAP constraint holds since, as shown by part (B) of Figure 5.543, the graph corresponding to the NODES collection is a map containing NBCYCLE $=2$ cycles (i.e., a first cycle involving vertices 1, 5 and 9 and a second cycle involving vertex 8) and 3 trees (i.e., two trees respectively involving vertices 7 and 4, 6, 2 and attached to the first cycle, and one tree mentioning vertex 3 linked to the second cycle.)

Typical

$\mathtt{NBCYCLE} > 0$
$\mathtt{NBTREE} > 0$
$\mathtt{NBCYCLE} < |\mathtt{NODES}|$
$\mathtt{NBCYCLE} < \mathtt{NBTREE}$
$|\mathtt{NODES}| > 2$

Symmetry

Items of NODES are permutable.

Arg. properties

- Functional dependency: NBCYCLE determined by NODES.
- Functional dependency: NBTREE determined by NODES.

See also

common keyword: CYCLE, GRAPH_CROSSING, TREE *(graph partitioning constraint)*.

Keywords

constraint arguments: pure functional dependency.

constraint type: graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck.

final graph structure: connected component.

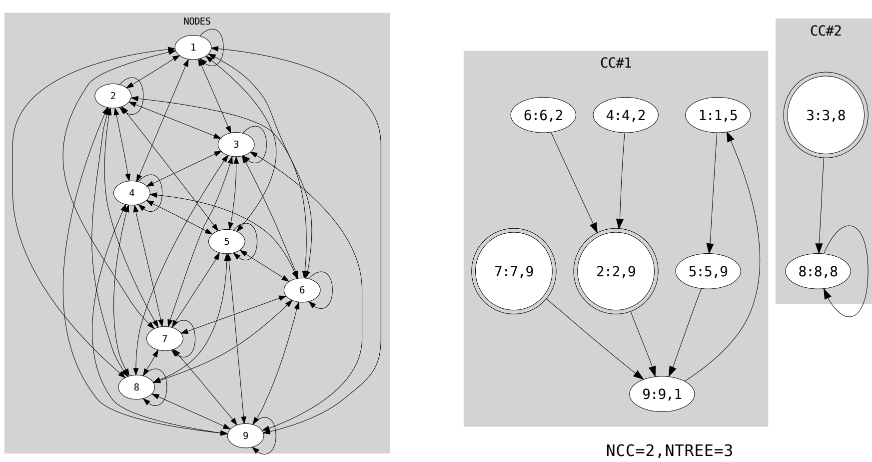
modelling: functional dependency.

Arc input(s)	NODES
Arc generator	*CLIQUE* $\mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **NCC**= NBCYCLE • **NTREE**= NBTREE

Graph model Note that, for the argument NBTREE of the MAP constraint, we consider a definition different from the one used for the argument NTREES of the TREE constraint:

- In the MAP constraint the number of trees NBTREE is equal to the number of vertices of the final graph, which both do not belong to any circuit and have a successor that is located on a circuit. Therefore we count three trees in the context of the **Example** slot.
- In the TREE constraint the number of trees NTREES is equal to the number of connected components of the final graph.

Parts (A) and (B) of Figure 5.543 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NCC** graph property, we display the two connected components of the final graph. Each of them corresponds to a connected map. The first connected map is made up from one circuit and two trees, while the second one consists of one circuit and one tree. Since we also use the **NTREE** graph property, we display with a double circle those vertices that do not belong to any circuit but for which at least one successor belongs to a circuit.



(A) **(B)**

Figure 5.543: Initial and final graph of the MAP constraint

5.243 MAX_DECREASING_SLOPE

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin
Motivated by time series.

Constraint
MAX_DECREASING_SLOPE(MAX, VARIABLES)

Arguments

MAX	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

MAX ≥ 0
MAX $<$range(VARIABLES.var)
required(VARIABLES, var)
$|$VARIABLES$| > 0$

Purpose

Given a sequence of variables VARIABLES $= V_1, V_2, \ldots, V_n$, sets MAX to 0 if $\nexists i \in [1, n-1] | V_i > V_{i+1}$, otherwise sets MAX to $\max_{i \in [1,n-1] | V_i > V_{i+1}} (V_i - V_{i+1})$.

Example

$(4, \langle 1, 1, 5, 8, 6, 2, 4, 1, 2 \rangle)$
$(0, \langle 1, 3, 5, 8 \rangle)$
$(8, \langle 3, 1, 9, 1 \rangle)$

The first MAX_DECREASING_SLOPE constraint holds since the sequence 1 1 5 8 6 2 4 1 2 contains two decreasing subsequences 8 6 2 and 4 1 and the maximum slope is equal to $\max(8-6, 6-2, 4-1) = 4$ as shown on Figure 5.544.

Typical

MAX > 0
MAX $<$range(VARIABLES.var) $- 1$
$|$VARIABLES$| > 2$
range(VARIABLES.var) > 2

Typical model
nval(VARIABLES.var) > 2

Symmetry
One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties
Functional dependency: MAX determined by VARIABLES.

Usage
Getting the maximum slope over the decreasing sequences of time series.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for MAX_DECREASING_SLOPE: domains $0..n$

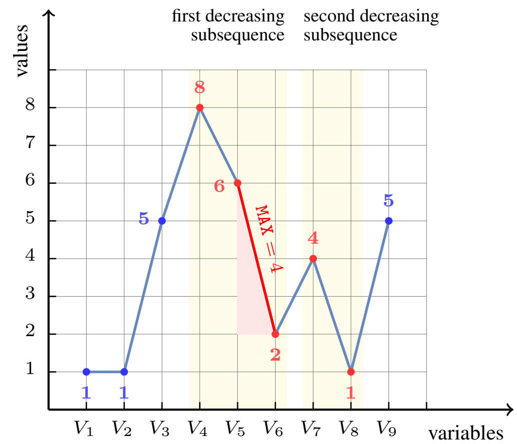
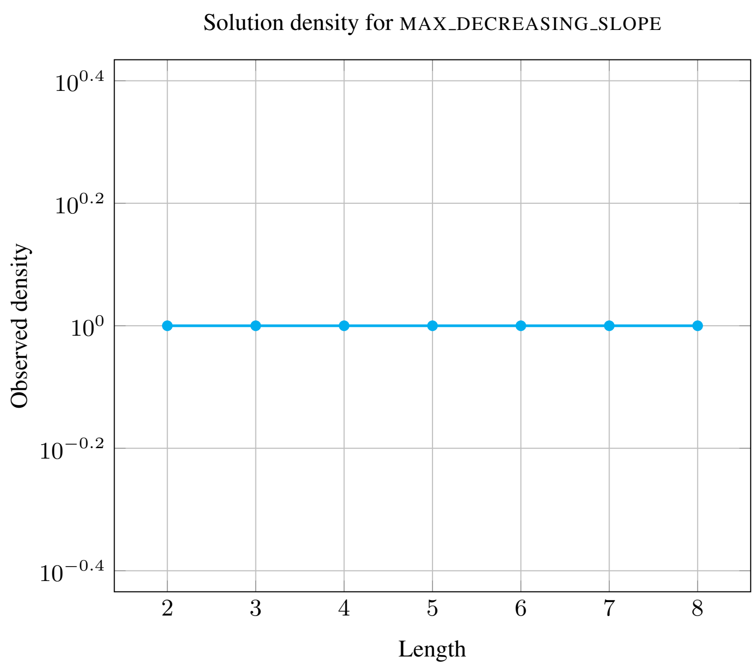
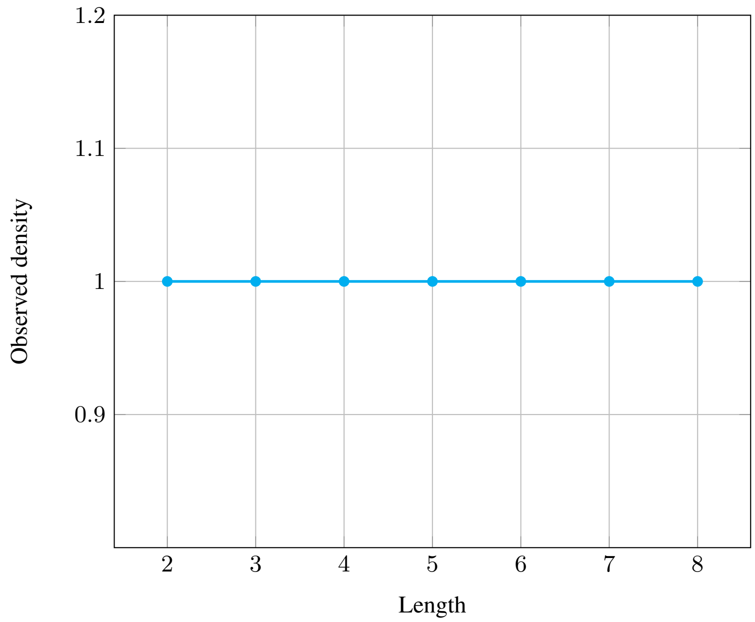


Figure 5.544: Illustration of the first example of the **Example** slot: a sequence of nine variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9 respectively fixed to values 1, 1, 5, 8, 6, 2, 4, 1, 5 and the corresponding maximum slope on the strictly decreasing subsequences 8 6 2 and 4 1 (MAX $= 4$)

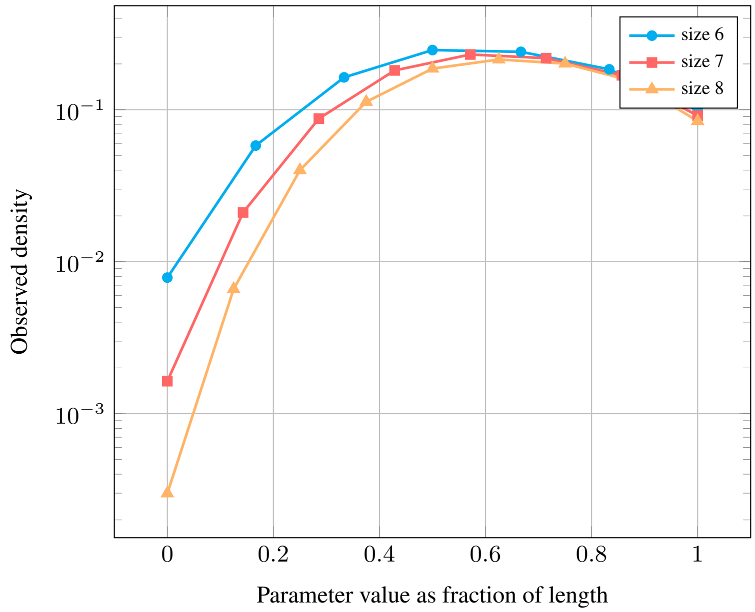




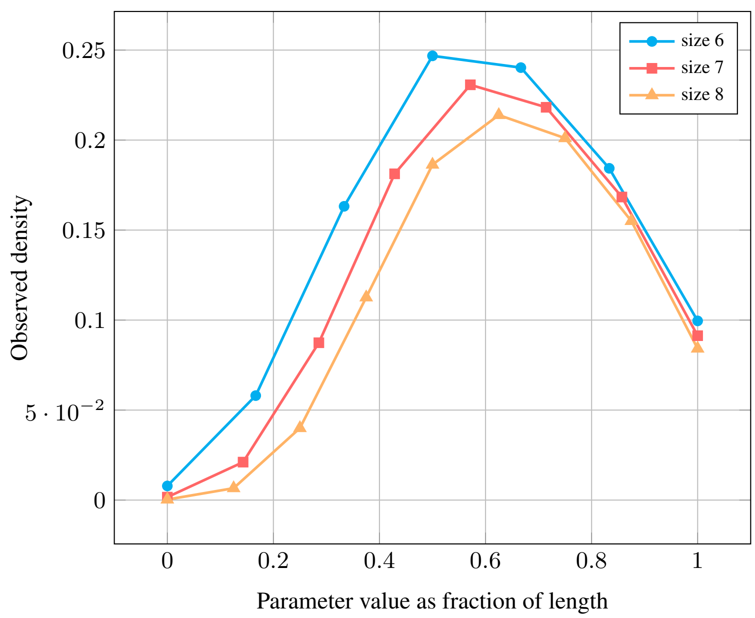
Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	6	20	70	252	924	3432	12870
	1	2	20	151	1036	6828	44220	284405
	2	1	16	188	1952	19200	183304	1721425
	3	-	8	142	2106	29035	380116	4847301
	4	-	-	74	1584	28266	483840	8021350
	5	-	-	-	846	21684	457632	9208124
	6	-	-	-	-	11712	353088	8654931
	7	-	-	-	-	-	191520	6673834
	8	-	-	-	-	-	-	3622481

Solution count for MAX_DECREASING_SLOPE: domains $0..n$

Solution density for MAX_DECREASING_SLOPE



Solution density for MAX_DECREASING_SLOPE



Keywords **characteristic of a constraint:** automaton, automaton with counters.
combinatorial object: sequence.
constraint arguments: reverse of a constraint, pure functional dependency.

filtering: glue matrix.

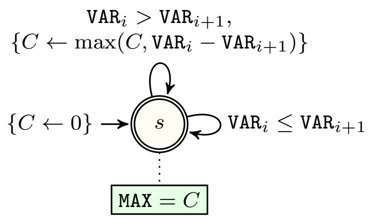
modelling: functional dependency.

Cond. implications

- MAX_DECREASING_SLOPE(MAX, VARIABLES)
  with range(VARIABLES.var) = MAX + 1
  implies LONGEST_DECREASING_SEQUENCE(L, VARIABLES)
  when range(VARIABLES.var) = L + 1.

- MAX_DECREASING_SLOPE(MAX, VARIABLES)
  with MAX = 1
  implies MIN_DECREASING_SLOPE(MIN, VARIABLES)
  when MIN = 1.

Automaton Figure 5.545 depicts the automaton associated with the MAX_DECREASING_SLOPE constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i \leq \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1)$.



	s
s	$\max(\overrightarrow{C}, \overleftarrow{C})$

Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

Figure 5.545: Automaton for the MAX_DECREASING_SLOPE constraint and its glue matrix (note that the reverse of MAX_DECREASING_SLOPE is MAX_INCREASING_SLOPE)

5.244 MAX_INCREASING_SLOPE

DESCRIPTION LINKS AUTOMATON

Origin Motivated by time series.

Constraint MAX_INCREASING_SLOPE(MAX, VARIABLES)

Arguments

MAX : dvar
VARIABLES : collection(var−dvar)

Restrictions

MAX ≥ 0
MAX $<$range(VARIABLES.var)
required(VARIABLES, var)
$|$VARIABLES$| > 0$

Purpose Given a sequence of variables VARIABLES $= V_1, V_2, \ldots, V_n$, sets MAX to 0 if $\nexists i \in [1, n-1] | V_i < V_{i+1}$, otherwise sets MAX to $\max_{i \in [1,n-1] | V_i < V_{i+1}} (V_{i+1} - V_i)$.

Example

$(4, \langle 1, 1, 5, 8, 6, 2, 2, 1, 2 \rangle)$
$(0, \langle 9, 8, 6, 4, 1, 0 \rangle)$
$(8, \langle 9, 6, 6, 4, 1, 9 \rangle)$

The first MAX_INCREASING_SLOPE constraint holds since the sequence 1 1 5 8 6 2 2 1 2 contains two increasing subsequences 1 5 8 and 1 2 and the maximum slope is equal to $\max(5-1, 8-5, 2-1) = 4$ as shown on Figure 5.546.

Typical

MAX > 0
MAX $<$range(VARIABLES.var) $- 1$
$|$VARIABLES$| > 2$
range(VARIABLES.var) > 2

Typical model nval(VARIABLES.var) > 2

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: MAX determined by VARIABLES.

Usage Getting the maximum slope over the increasing sequences of time series.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for MAX_INCREASING_SLOPE: domains $0..n$

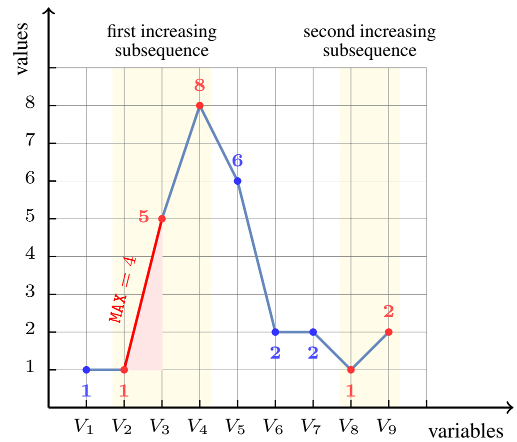
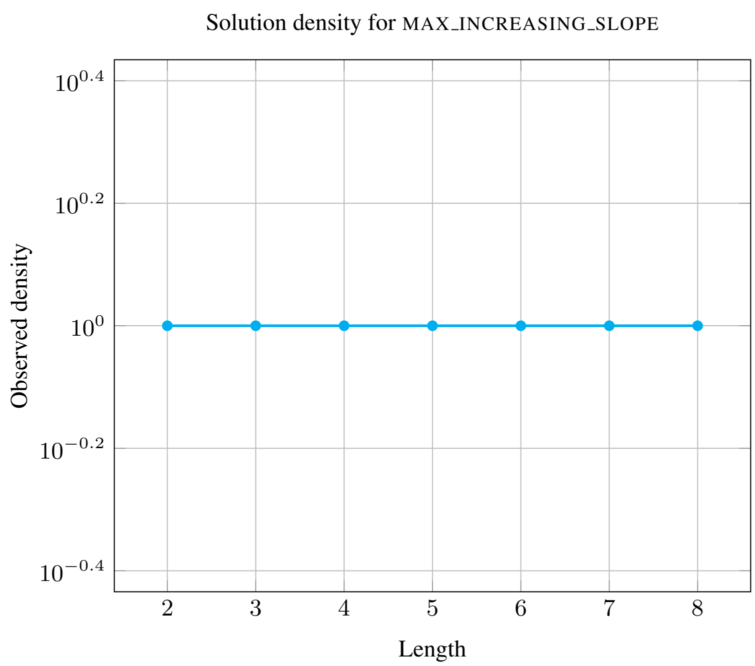
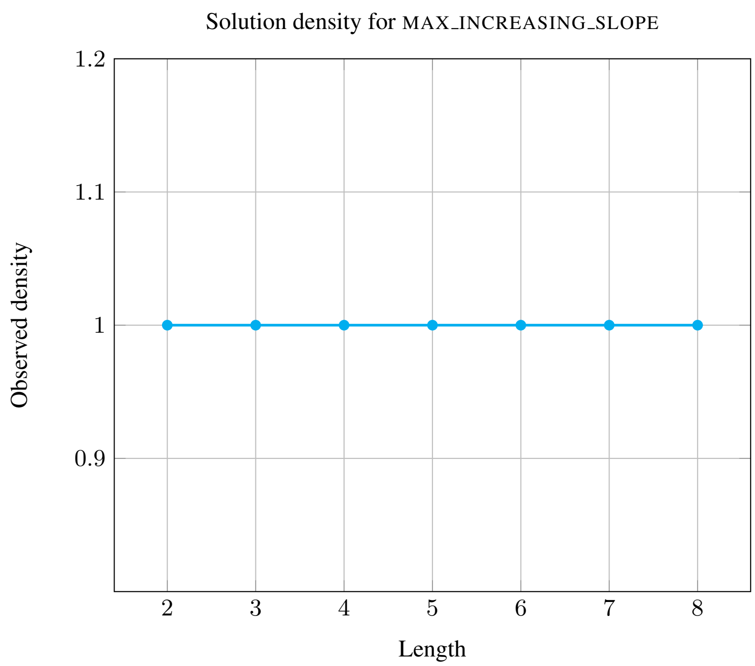


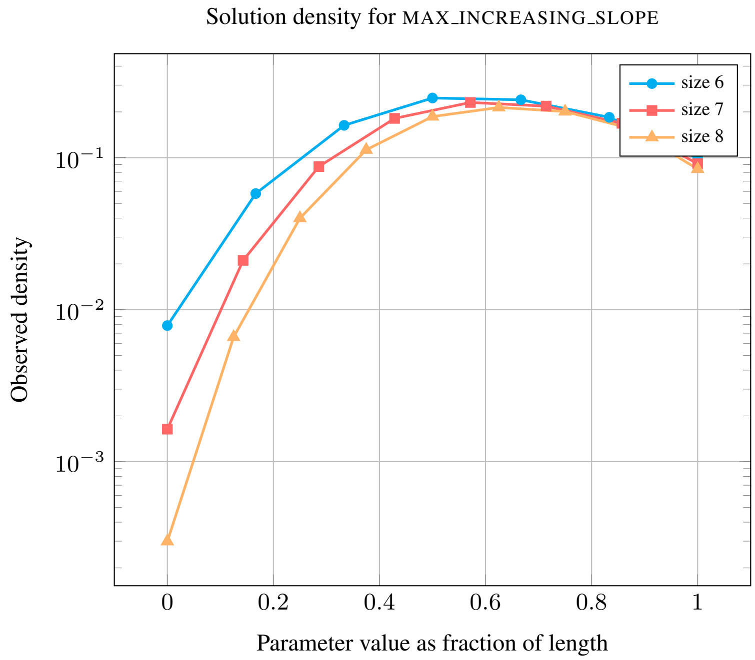
Figure 5.546: Illustration of the first example of the **Example** slot: a sequence of nine variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9 respectively fixed to values 1, 1, 5, 8, 6, 2, 2, 1, 2 and the corresponding maximum slope on the strictly increasing subsequences 1 5 8 and 1 2 ($\mathtt{MAX} = 4$)

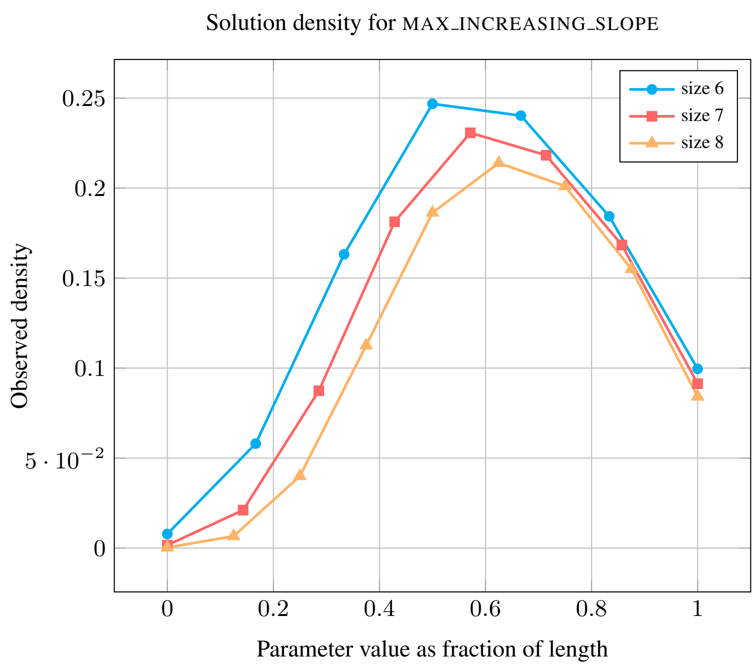




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	6	20	70	252	924	3432	12870
	1	2	20	151	1036	6828	44220	284405
	2	1	16	188	1952	19200	183304	1721425
	3	-	8	142	2106	29035	380116	4847301
	4	-	-	74	1584	28266	483840	8021350
	5	-	-	-	846	21684	457632	9208124
	6	-	-	-	-	11712	353088	8654931
	7	-	-	-	-	-	191520	6673834
	8	-	-	-	-	-	-	3622481

Solution count for MAX_INCREASING_SLOPE: domains $0..n$





Keywords **characteristic of a constraint:** automaton, automaton with counters.
combinatorial object: sequence.
constraint arguments: reverse of a constraint, pure functional dependency.

filtering: glue matrix.

modelling: functional dependency.

Cond. implications

- MAX_INCREASING_SLOPE(MAX, VARIABLES)
  with range(VARIABLES.var) = MAX + 1
  implies LONGEST_INCREASING_SEQUENCE(L, VARIABLES)
  when range(VARIABLES.var) = L + 1.

- MAX_INCREASING_SLOPE(MAX, VARIABLES)
  with MAX = 1
  implies MIN_INCREASING_SLOPE(MIN, VARIABLES)
  when MIN = 1.

Automaton Figure 5.547 depicts the automaton associated with the MAX_INCREASING_SLOPE constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and S_i: $(\texttt{VAR}_i \geq \texttt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow S_i = 1)$.

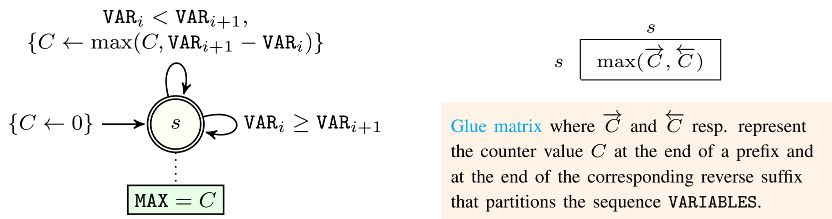


Figure 5.547: Automaton for the MAX_INCREASING_SLOPE constraint and its glue matrix (note that the reverse of MAX_INCREASING_SLOPE is MAX_DECREASING_SLOPE)

5.245 MAX_INDEX

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint MAX_INDEX(MAX_INDEX, VARIABLES)

Arguments

MAX_INDEX : dvar
VARIABLES : collection(index−int, var−dvar)

Restrictions

$|$VARIABLES$| > 0$
MAX_INDEX ≥ 0
MAX_INDEX $\leq |$VARIABLES$|$
required(VARIABLES, [index, var])
VARIABLES.index ≥ 1
VARIABLES.index $\leq |$VARIABLES$|$
distinct(VARIABLES, index)

Purpose MAX_INDEX is one of the indices of the collection of variables VARIABLES corresponding to its maximum value.

Example

$$\left(3, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{var}-3, \\ \texttt{index}-2 & \texttt{var}-2, \\ \texttt{index}-3 & \texttt{var}-7, \\ \texttt{index}-4 & \texttt{var}-2, \\ \texttt{index}-5 & \texttt{var}-7 \end{array} \right\rangle \right)$$

The attribute var $=$ 7 of the third and fifth items of the collection VARIABLES is the maximum value over values $3, 2, 7, 2, 7$. Consequently, the MAX_INDEX constraint holds since its first argument MAX_INDEX is set to $3 \in \{3, 5\}$.

Typical

$|$VARIABLES$| > 0$
range(VARIABLES.var) > 1

Symmetries

- Items of VARIABLES are permutable.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

See also **comparison swapped:** MIN_INDEX.

Keywords

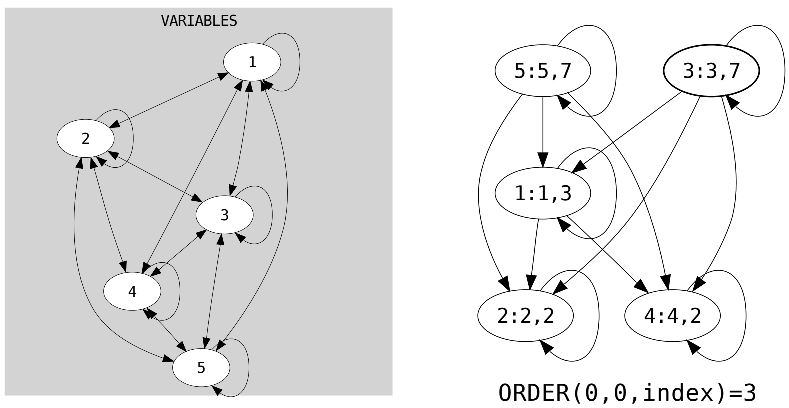
characteristic of a constraint: maximum.

constraint type: order constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	$\bigvee\left(\begin{array}{l}\texttt{variables1.key} = \texttt{variables2.key},\\ \texttt{variables1.var} > \texttt{variables2.var}\end{array}\right)$
Graph property(ies)	**ORDER**(0, 0, index) = MAX_INDEX

Graph model Parts (A) and (B) of Figure 5.548 respectively show the initial and final graph associated with the **Example** slot. Since we use the **ORDER** graph property, the vertex of rank 0 (without considering the loops) of the final graph is outlined with a thick circle.



(A) **(B)**

Figure 5.548: Initial and final graph of the MAX_INDEX constraint

5.246 MAX_N

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [29]

Constraint MAX_N(MAX, RANK, VARIABLES)

Arguments

MAX	:	dvar
RANK	:	int
VARIABLES	:	collection(var−dvar)

Restrictions

RANK ≥ 0
RANK $<$ |VARIABLES|
|VARIABLES| > 0
required(VARIABLES, var)

Purpose

MAX is the maximum value of rank RANK (i.e., the RANKth largest distinct value, identical values are merged) of the collection of domain variables VARIABLES. The maximum value has rank 0.

Example

$(6, 1, \langle 3, 1, 7, 1, 6\rangle)$

The MAX_N constraint holds since its first argument MAX $= 6$ is fixed to the second (i.e., RANK $+ 1$) largest distinct value of the collection $\langle 3, 1, 7, 1, 6\rangle$.

Typical

RANK > 0
RANK < 3
|VARIABLES| > 1
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES are permutable.
- One and the same constant can be added to MAX as well as to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: MAX determined by RANK and VARIABLES.

Algorithm [29].

Reformulation The constraint AMONG_VAR(1, ⟨MAX⟩, VARIABLES) enforces MAX to be assigned one of the values of VARIABLES. The constraint NVALUE(NVAL, VARIABLES) provides a hand on the number of distinct values assigned to the variables of VARIABLES. By associating to each variable V_i ($i \in [1, |\text{VARIABLES}|]$) of the VARIABLES collection a *rank* variable $R_i \in$

$[0, |\texttt{VARIABLES}| - 1]$ with the reified constraint $R_i = \texttt{RANK} \Leftrightarrow V_i = \texttt{MAX}$, the inequality $R_i < \texttt{NVAL}$, and by creating for each pair of variables V_i, V_j $(i, j < i \in [1, |\texttt{VARIABLES}|])$ the reified constraints

$V_i > V_j \Leftrightarrow R_i < R_j$,
$V_i = V_j \Leftrightarrow R_i = R_j$,
$V_i < V_j \Leftrightarrow R_i > R_j$,

one can reformulate the MAX_N constraint in term of $3 \cdot \frac{|\texttt{VARIABLES}| \cdot (|\texttt{VARIABLES}| - 1)}{2} + 1$ reified constraints.

See also

comparison swapped: MIN_N.

generalisation: MAXIMUM *(absolute maximum replaced by maximum or order* n*)*.

Keywords

characteristic of a constraint: rank, maximum.

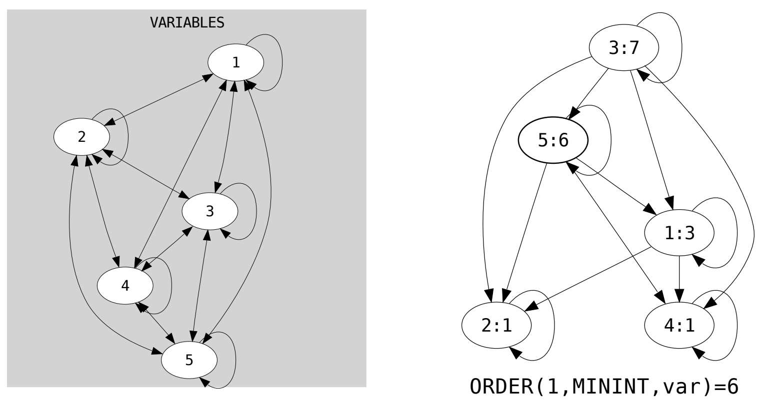
constraint arguments: pure functional dependency.

constraint type: order constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \texttt{variables1.key} = \texttt{variables2.key}, \\ \texttt{variables1.var} > \texttt{variables2.var} \end{array} \right)$
Graph property(ies)	**ORDER**(RANK, MININT, var) = MAX

Graph model Parts (A) and (B) of Figure 5.549 respectively show the initial and final graph associated with the **Example** slot. Since we use the **ORDER** graph property, the vertex of rank 1 (without considering the loops) of the final graph is outlined with a thick circle.



(A) **(B)**

Figure 5.549: Initial and final graph of the MAX_N constraint

5.247 MAX_NVALUE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from NVALUE.

Constraint MAX_NVALUE(MAX, VARIABLES)

Arguments

MAX : dvar
VARIABLES : collection(var−dvar)

Restrictions

MAX $\geq$ 1
MAX $\leq$ |VARIABLES|
required(VARIABLES, var)

Purpose MAX is the maximum number of times that the same value is taken by the variables of the collection VARIABLES.

Example

$(3, \langle 9, 1, 7, 1, 1, 6, 7, 7, 4, 9\rangle)$
$(1, \langle 9, 1, 7, 3, 2, 6\rangle)$
$(6, \langle 5, 5, 5, 5, 5, 5\rangle)$

In the first example, values $1, 4, 6, 7, 9$ are respectively used $3, 1, 1, 3, 2$ times. So the maximum number of time MAX that a same value occurs is 3. Consequently the corresponding MAX_NVALUE constraint holds.

Typical

MAX $>$ 1
MAX $<$ |VARIABLES|
|VARIABLES| $>$ 1
range(VARIABLES.var) $>$ 1

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Functional dependency: MAX determined by VARIABLES.

Usage This constraint may be used in order to replace a set of COUNT or AMONG constraints were one would have to generate explicitly one constraint for each potential value. Also useful for constraining the number of occurrences of the mostly used value without knowing this value in advance and without giving explicitly an upper limit on the number of occurrences of each value as it is done in the GLOBAL_CARDINALITY constraint.

Reformulation Assume that VARIABLES is not empty. Let α and β respectively denote the smallest and largest possible values that can be assigned to the variables of the VARIABLES collection. Let the variables $O_\alpha, O_{\alpha+1}, \ldots, O_\beta$ respectively correspond to the number of occurrences of values $\alpha, \alpha+1, \ldots, \beta$ within the variables of the VARIABLES collection. The

MAX_NVALUE constraint can be expressed as the conjunction of the following two constraints:

GLOBAL_CARDINALITY $(\texttt{VARIABLES},$
$\langle \texttt{val} - \alpha\ \texttt{noccurrence} - O_\alpha,$
$\texttt{val} - \alpha + 1\ \texttt{noccurrence} - O_{\alpha+1},$
$\ldots$
$\texttt{val} - \beta\ \texttt{noccurrence} - O_\beta\rangle),$

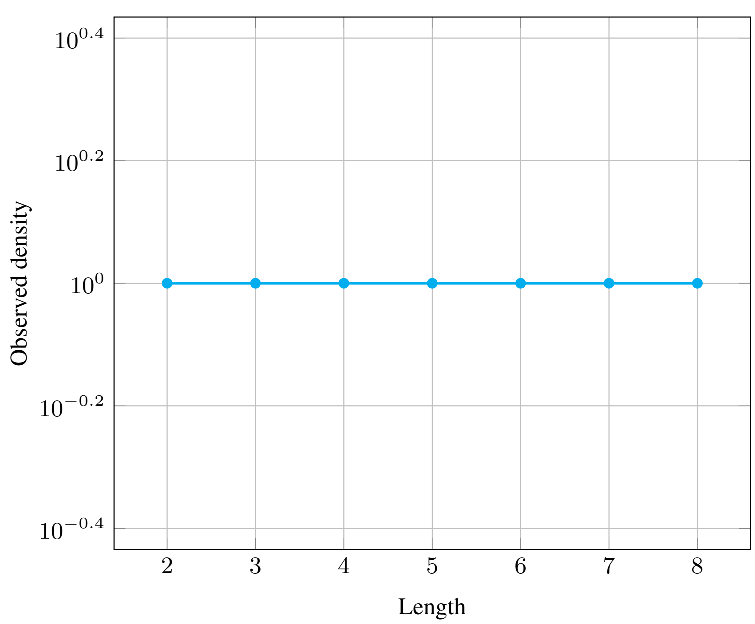
MAXIMUM$(\texttt{MAX}, \langle O_\alpha, O_{\alpha+1}, \ldots, O_\beta\rangle).$

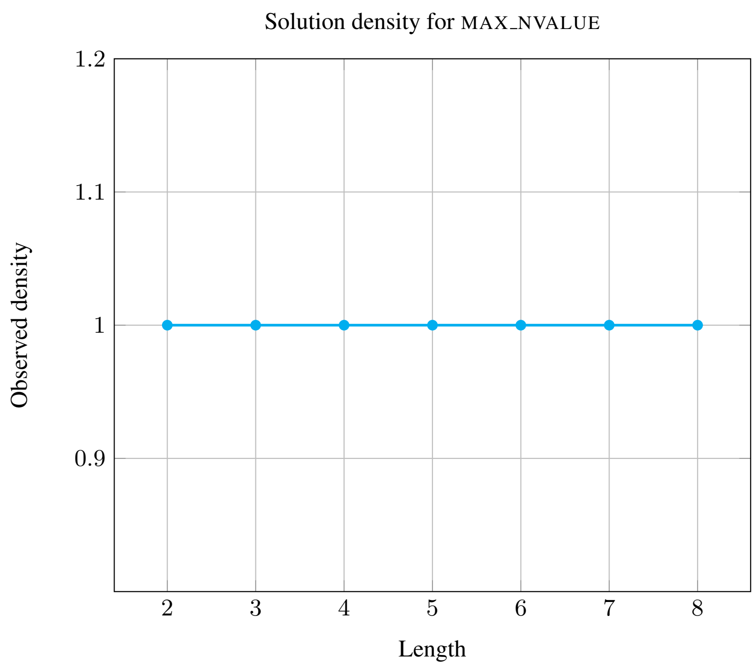
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for MAX_NVALUE: domains $0..n$

Solution density for MAX_NVALUE

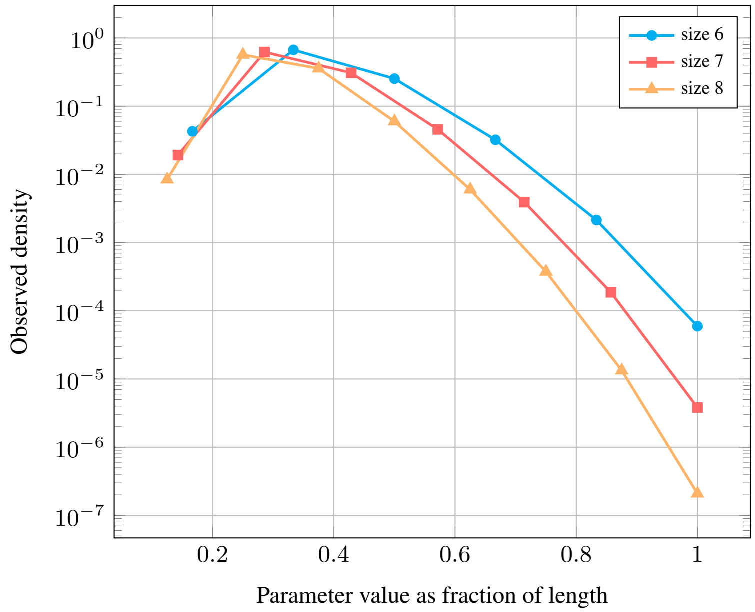




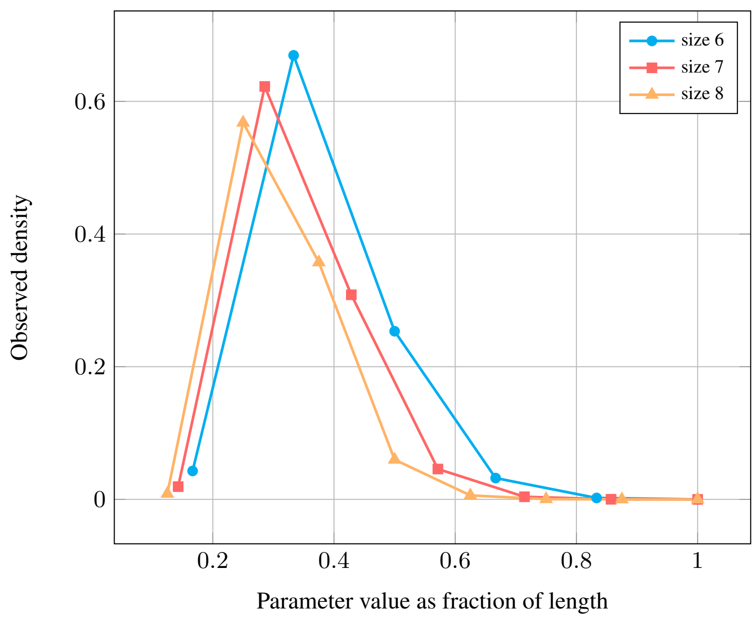
Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	6	24	120	720	5040	40320	362880
	2	3	36	420	5400	78750	1305360	24449040
	3	-	4	80	1500	29820	646800	15382080
	4	-	-	5	150	3780	96040	2577960
	5	-	-	-	6	252	8232	258048
	6	-	-	-	-	7	392	16128
	7	-	-	-	-	-	8	576
	8	-	-	-	-	-	-	9

Solution count for MAX_NVALUE: domains $0..n$

Solution density for MAX_NVALUE



Solution density for MAX_NVALUE



See also **common keyword:** AMONG *(counting constraint)*, COUNT, GLOBAL_CARDINALITY *(value constraint,counting constraint)*, MIN_NVALUE, NVALUE *(counting constraint)*.

Keywords

application area: assignment.

characteristic of a constraint: maximum, automaton, automaton with array of counters.

constraint arguments: pure functional dependency.

constraint type: value constraint, counting constraint.

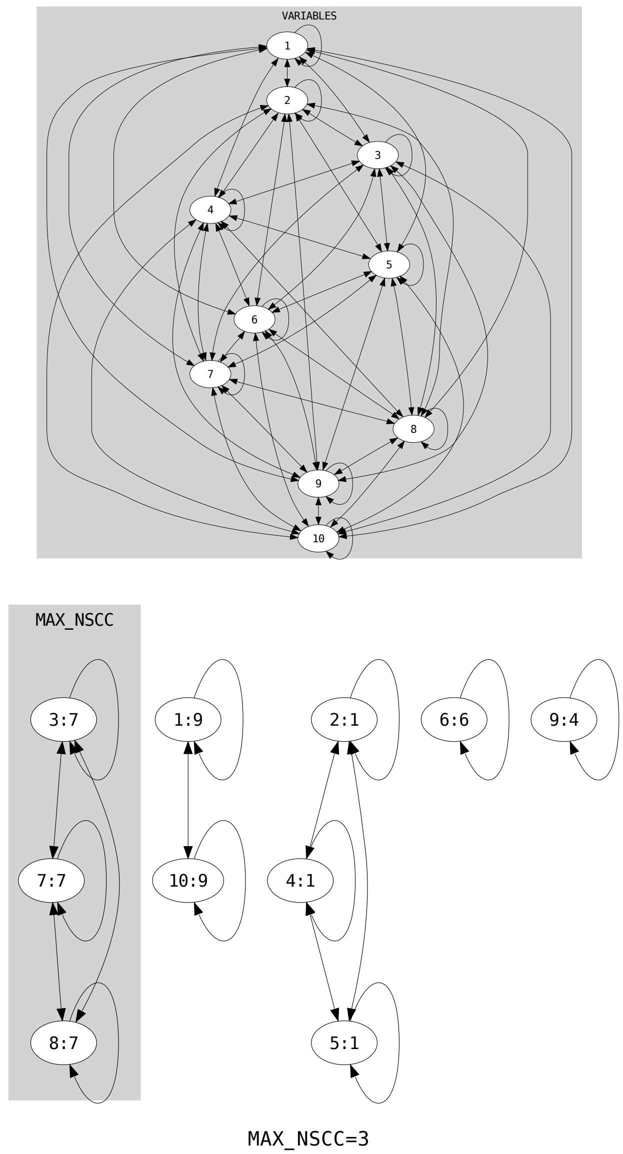
final graph structure: equivalence.

modelling: maximum number of occurrences, functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**MAX_NSCC** = MAX

Graph model Because of the arc constraint, each strongly connected component of the final graph corresponds to a distinct value that is assigned to a subset of variables of the VARIABLES collection. Therefore the number of vertices of the largest strongly connected component is equal to the mostly used value.

Parts (A) and (B) of Figure 5.550 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **MAX_NSCC** graph property, we show the largest strongly connected component of the final graph.



(A)

(B)

Figure 5.550: Initial and final graph of the MAX_NVALUE constraint

Automaton Figure 5.551 depicts the automaton associated with the MAX_NVALUE constraint. To each item of the collection VARIABLES corresponds a signature variable S_i that is equal to 0.

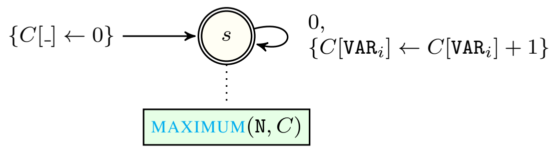


Figure 5.551: Automaton of the MAX_NVALUE constraint

5.248 MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Design.

Constraint MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES(MAX, K, VECTORS)

Type

VECTOR : collection(var−dvar)

Arguments

MAX : int
K : int
VECTORS : collection(vec − VECTOR)

Restrictions

required(VECTOR, var)
$|\texttt{VECTOR}| \geq 2$
ALLDIFFERENT(VECTOR)
$\texttt{MAX} \geq 1$
$\texttt{K} \geq 2$
$\texttt{K} < |\texttt{VECTOR}|$
required(VECTORS, vec)
$|\texttt{VECTORS}| \geq 1$
same_size(VECTORS, vec)

Purpose

MAX is equal to the maximum number of occurrences of identical vectors derived from the vectors VECTORS in the following way. To each vector $\langle v_1, v_2, \ldots, v_m\rangle$ of VECTORS (with $v_1, v_2, \ldots, v_m$ distinct) we generate all vectors $\langle u_1, u_2, \ldots, u_{\texttt{K}}\rangle$ such that $u_1 = v_p$, $u_2 = v_{p+1}, \ldots, u_{\texttt{K}} = v_{p+\texttt{K}-1}$ or $u_1 = v_{p+\texttt{K}-1}$, $u_2 = v_{p+\texttt{K}-2}, \ldots, u_{\texttt{K}} = v_p$ (with $1 \leq p \leq m - \texttt{K} + 1$).

Example

$(1, 2, \langle \texttt{vec} - \langle 4, 1, 3\rangle, \texttt{vec} - \langle 2, 7, 6\rangle, \texttt{vec} - \langle 5, 9, 8\rangle\rangle)$

Given the three vectors of the example we respectively generate:

- the pairs $\langle 4, 1\rangle$, $\langle 1, 4\rangle$, $\langle 1, 3\rangle$, $\langle 3, 1\rangle$ from the triple $\langle 4, 1, 3\rangle$,
- the pairs $\langle 2, 7\rangle$, $\langle 7, 2\rangle$, $\langle 7, 6\rangle$, $\langle 6, 7\rangle$ from the triple $\langle 2, 7, 6\rangle$,
- the pairs $\langle 5, 9\rangle$, $\langle 9, 5\rangle$, $\langle 9, 8\rangle$, $\langle 8, 9\rangle$ from the triple $\langle 5, 9, 8\rangle$.

Putting these pairs together, we get the set of pairs $\{\langle 1, 3\rangle, \langle 1, 4\rangle, \langle 2, 7\rangle, \langle 3, 1\rangle, \langle 4, 1\rangle, \langle 5, 9\rangle, \langle 6, 7\rangle, \langle 7, 2\rangle, \langle 7, 6\rangle, \langle 8, 9\rangle, \langle 9, 5\rangle, \langle 9, 8\rangle\}$. The MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES constraint holds since the components of each of the original three vectors are distinct, and since MAX is set to one and all the generated pairs are distinct.

Typical

$\texttt{MAX} = 1$
$\texttt{K} = 2$
$|\texttt{VECTORS}| > 2$

Arg. properties

- Functional dependency: MAX determined by K and VECTORS.
- Contractible wrt. VECTORS when $\texttt{MAX} = 1$.

Usage

This constraint occurs in balanced block design problems [374].

See also

common keyword: MAX_OCC_OF_SORTED_TUPLES_OF_VALUES, MAX_OCC_OF_TUPLES_OF_VALUES *(vector)*.

Keywords

characteristic of a constraint: vector.

constraint type: predefined constraint.

modelling: functional dependency.

5.249 MAX_OCC_OF_SORTED_TUPLES_OF_VALUES

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Design.

Constraint MAX_OCC_OF_SORTED_TUPLES_OF_VALUES(MAX, K, VECTORS)

Type VECTOR : collection(var−dvar)

Arguments

MAX : int
K : int
VECTORS : collection(vec − VECTOR)

Restrictions

required(VECTOR, var)
$|\text{VECTOR}| \geq 2$
ALLDIFFERENT(VECTOR)
$\text{MAX} \geq 1$
$\text{K} \geq 2$
$\text{K} < |\text{VECTOR}|$
required(VECTORS, vec)
$|\text{VECTORS}| \geq 1$
same_size(VECTORS, vec)

Purpose MAX is equal to the maximum number of occurrences of identical vectors derived from the vectors VECTORS in the following way. To each vector $\langle v_1, v_2, \ldots, v_m\rangle$ of VECTORS (with $v_1, v_2, \ldots, v_m$ distinct) let $\langle s_1, s_2, \ldots, s_m\rangle$ be the corresponding sorted vector by increasing component. We generate all vectors $\langle u_1, u_2, \ldots, u_K\rangle$ such that $u_1 = s_{i_1}$, $u_2 = s_{i_2}, \ldots, u_K = s_{i_K}$ (with $1 \leq i_1 < i_2 < \cdots < i_K \leq m$).

Example

$$\left(1, 2, \left\langle \begin{array}{l} \text{vec} - \langle 4,2,1\rangle, \\ \text{vec} - \langle 2,3,5\rangle, \\ \text{vec} - \langle 3,6,4\rangle, \\ \text{vec} - \langle 5,4,7\rangle, \\ \text{vec} - \langle 6,5,1\rangle, \\ \text{vec} - \langle 7,6,2\rangle, \\ \text{vec} - \langle 3,1,7\rangle \end{array} \right\rangle \right)$$

Given the seven vectors of the example we respectively generate:

- the pairs $\langle 1,2\rangle$, $\langle 1,4\rangle$ and $\langle 2,4\rangle$ from the triple $\langle 4,2,1\rangle$,
- the pairs $\langle 2,3\rangle$, $\langle 2,5\rangle$ and $\langle 3,5\rangle$ from the triple $\langle 2,3,5\rangle$,
- the pairs $\langle 3,4\rangle$, $\langle 3,6\rangle$ and $\langle 4,6\rangle$ from the triple $\langle 3,6,4\rangle$,
- the pairs $\langle 4,5\rangle$, $\langle 4,7\rangle$ and $\langle 5,7\rangle$ from the triple $\langle 5,4,7\rangle$,
- the pairs $\langle 1,5\rangle$, $\langle 1,6\rangle$ and $\langle 5,6\rangle$ from the triple $\langle 6,5,1\rangle$,

- the pairs $\langle 2,6\rangle$, $\langle 2,7\rangle$ and $\langle 6,7\rangle$ from the triple $\langle 7,6,2\rangle$,
- the pairs $\langle 1,3\rangle$, $\langle 1,7\rangle$ and $\langle 3,7\rangle$ from the triple $\langle 3,1,7\rangle$.

Putting these pairs together, we get the set of pairs $\{\langle 1,2\rangle$, $\langle 1,3\rangle$, $\langle 1,4\rangle$, $\langle 1,5\rangle$, $\langle 1,6\rangle$, $\langle 1,7\rangle$, $\langle 2,3\rangle$, $\langle 2,4\rangle$, $\langle 2,5\rangle$, $\langle 2,6\rangle$, $\langle 2,7\rangle$, $\langle 3,4\rangle$, $\langle 3,5\rangle$, $\langle 3,6\rangle$, $\langle 3,7\rangle$, $\langle 4,5\rangle$, $\langle 4,6\rangle$, $\langle 4,7\rangle$, $\langle 5,6\rangle$, $\langle 5,7\rangle$, $\langle 6,7\rangle\}$. The MAX_OCC_OF_SORTED_TUPLES_OF_VALUES constraint holds since each vector has pairwise distinct components, and since MAX is set to one and all the generated pairs are distinct.

Typical

$\texttt{MAX} = 1$
$\texttt{K} + 1 = |\texttt{VECTOR}|$
$|\texttt{VECTORS}| > 2$

Arg. properties

- Functional dependency: MAX determined by K and VECTORS.
- Contractible wrt. VECTORS when $\texttt{MAX} = 1$.

Usage

This constraint occurs in balanced block design problems where all vectors are not necessarily sorted.

See also

common keyword: MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES, MAX_OCC_OF_TUPLES_OF_VALUES *(vector)*.

implied by: MAX_OCC_OF_TUPLES_OF_VALUES.

Keywords

characteristic of a constraint: vector.

constraint type: predefined constraint.

modelling: functional dependency.

5.250 MAX_OCC_OF_TUPLES_OF_VALUES

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Design.

Constraint MAX_OCC_OF_TUPLES_OF_VALUES(MAX, K, VECTORS)

Type VECTOR : collection(var−dvar)

Arguments

MAX : int
K : int
VECTORS : collection(vec − VECTOR)

Restrictions

required(VECTOR, var)
$|\mathtt{VECTOR}| \geq 2$
STRICTLY_INCREASING(VECTOR)
$\mathtt{MAX} \geq 1$
$\mathtt{K} \geq 2$
$\mathtt{K} < |\mathtt{VECTOR}|$
required(VECTORS, vec)
$|\mathtt{VECTORS}| \geq 1$
same_size(VECTORS, vec)

Purpose

MAX is equal to the maximum number of occurrences of identical vectors derived from the vectors VECTORS in the following way. To each vector $\langle v_1, v_2, \ldots, v_m\rangle$ (with $v_1 < v_2 \wedge \cdots \wedge v_{m-1} < v_m$) of VECTORS we generate all vectors $\langle u_1, u_2, \ldots, u_{\mathtt{K}}\rangle$ such that $u_1 = v_{i_1}, u_2 = v_{i_2}, \ldots, u_{\mathtt{K}} = v_{i_{\mathtt{K}}}$ (with $1 \leq i_1 < i_2 < \cdots < i_{\mathtt{K}} \leq m$).

Example

$$\left(1, 2, \left\langle \begin{array}{l} \mathtt{vec} - \langle 1,2,4\rangle, \\ \mathtt{vec} - \langle 2,3,5\rangle, \\ \mathtt{vec} - \langle 3,4,6\rangle, \\ \mathtt{vec} - \langle 4,5,7\rangle, \\ \mathtt{vec} - \langle 1,5,6\rangle, \\ \mathtt{vec} - \langle 2,6,7\rangle, \\ \mathtt{vec} - \langle 1,3,7\rangle \end{array} \right\rangle\right)$$

Given the seven vectors of the example we respectively generate:

- the pairs $\langle 1,2\rangle$, $\langle 1,4\rangle$ and $\langle 2,4\rangle$ from the triple $\langle 1,2,4\rangle$,
- the pairs $\langle 2,3\rangle$, $\langle 2,5\rangle$ and $\langle 3,5\rangle$ from the triple $\langle 2,3,5\rangle$,
- the pairs $\langle 3,4\rangle$, $\langle 3,6\rangle$ and $\langle 4,6\rangle$ from the triple $\langle 3,4,6\rangle$,
- the pairs $\langle 4,5\rangle$, $\langle 4,7\rangle$ and $\langle 5,7\rangle$ from the triple $\langle 4,5,7\rangle$,
- the pairs $\langle 1,5\rangle$, $\langle 1,6\rangle$ and $\langle 5,6\rangle$ from the triple $\langle 1,5,6\rangle$,
- the pairs $\langle 2,6\rangle$, $\langle 2,7\rangle$ and $\langle 6,7\rangle$ from the triple $\langle 2,6,7\rangle$,

- the pairs $\langle 1,3\rangle$, $\langle 1,7\rangle$ and $\langle 3,7\rangle$ from the triple $\langle 1,3,7\rangle$.

Putting these pairs together, we get the set of pairs $\{\langle 1,2\rangle$, $\langle 1,3\rangle$, $\langle 1,4\rangle$, $\langle 1,5\rangle$, $\langle 1,6\rangle$, $\langle 1,7\rangle$, $\langle 2,3\rangle$, $\langle 2,4\rangle$, $\langle 2,5\rangle$, $\langle 2,6\rangle$, $\langle 2,7\rangle$, $\langle 3,4\rangle$, $\langle 3,5\rangle$, $\langle 3,6\rangle$, $\langle 3,7\rangle$, $\langle 4,5\rangle$, $\langle 4,6\rangle$, $\langle 4,7\rangle$, $\langle 5,6\rangle$, $\langle 5,7\rangle$, $\langle 6,7\rangle\}$. The MAX_OCC_OF_TUPLES_OF_VALUES constraint holds since the components of the original seven vectors are strictly increasing, and since MAX is set to one and all the generated pairs are distinct.

Typical

$\mathtt{MAX} \leq 2$
$|\mathtt{VECTOR}| < \mathtt{K} + 5$
$\mathtt{K} = 2 \vee \mathtt{K} + 1 = |\mathtt{VECTOR}|$
$|\mathtt{VECTORS}| > 2$

Arg. properties

- Functional dependency: MAX determined by K and VECTORS.
- Contractible wrt. VECTORS when $\mathtt{MAX} = 1$.

Usage

This constraint occurs in balanced block design problems [218, 273] such as Steiner or Kirkman triples.

See also

common keyword: MAX_OCC_OF_CONSECUTIVE_TUPLES_OF_VALUES, MAX_OCC_OF_SORTED_TUPLES_OF_VALUES *(vector)*.

implies: MAX_OCC_OF_SORTED_TUPLES_OF_VALUES.

Keywords

characteristic of a constraint: vector.

constraint type: predefined constraint.

modelling: functional dependency.

5.251 MAX_SIZE_SET_OF_CONSECUTIVE_VAR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint MAX_SIZE_SET_OF_CONSECUTIVE_VAR(MAX, VARIABLES)

Arguments

MAX	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

MAX ≥ 1
MAX $\leq$ |VARIABLES|
required(VARIABLES, var)

Purpose MAX is the size of the largest set of variables of the collection VARIABLES that all take their values in a set of consecutive values.

Example

(6, ⟨3, 1, 3, 7, 4, 1, 2, 8, 7, 6⟩)
(2, ⟨2, 6, 7, 3, 0, 9⟩)

In the first example, the two sets $\{3, 1, 3, 4, 1, 2\}$ and $\{7, 8, 7, 6\}$ take respectively their values in the two following sets of consecutive values $\{1, 2, 3, 4\}$ and $\{6, 7, 8\}$. Consequently, the corresponding MAX_SIZE_SET_OF_CONSECUTIVE_VAR constraint holds since the cardinality of the largest set of variables is 6.

Typical

MAX < |VARIABLES|
|VARIABLES| > 0
range(VARIABLES.var) > 1

Symmetries

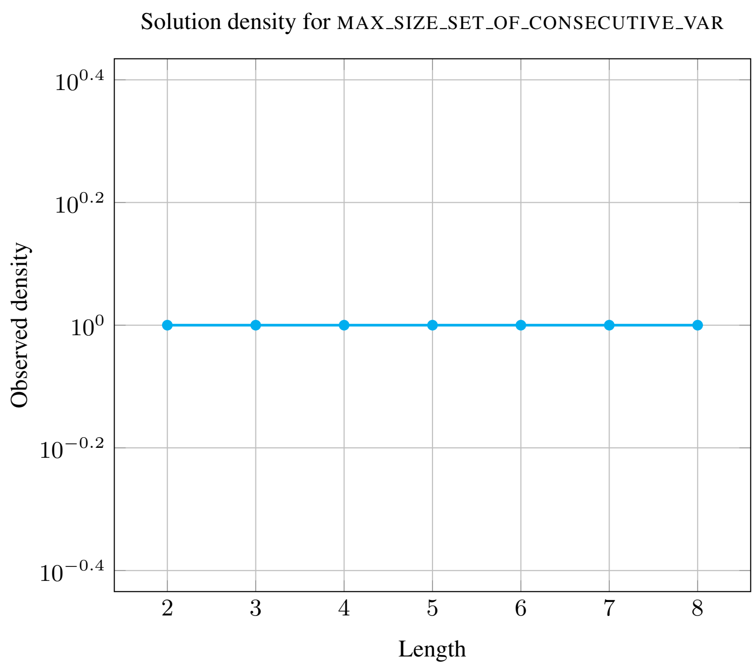
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: MAX determined by VARIABLES.

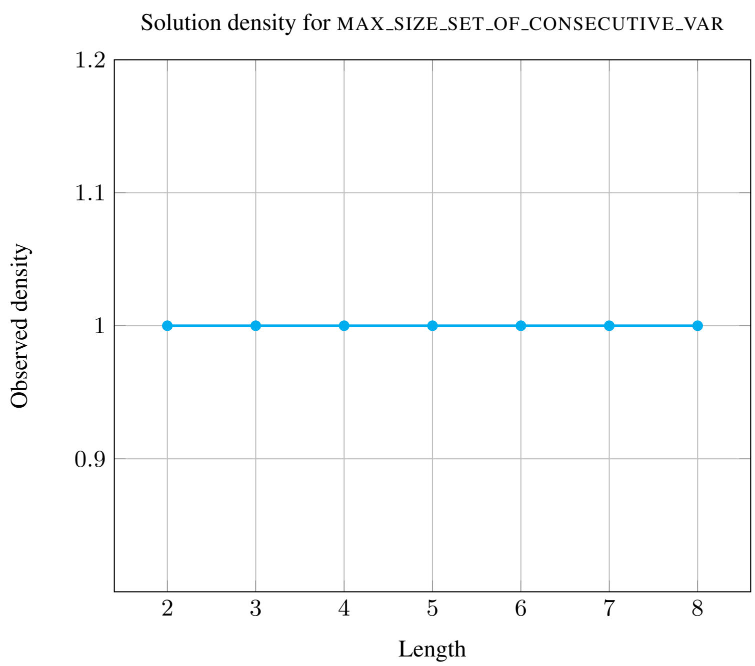
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for MAX_SIZE_SET_OF_CONSECUTIVE_VAR: domains $0..n$



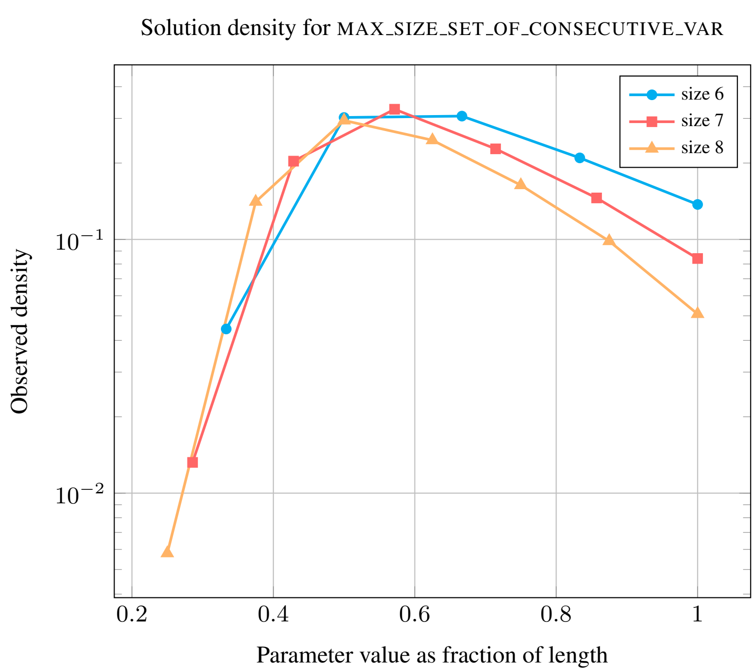
Solution density for MAX_SIZE_SET_OF_CONSECUTIVE_VAR
Observed density
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
2 3 4 5 6 7 8
Length

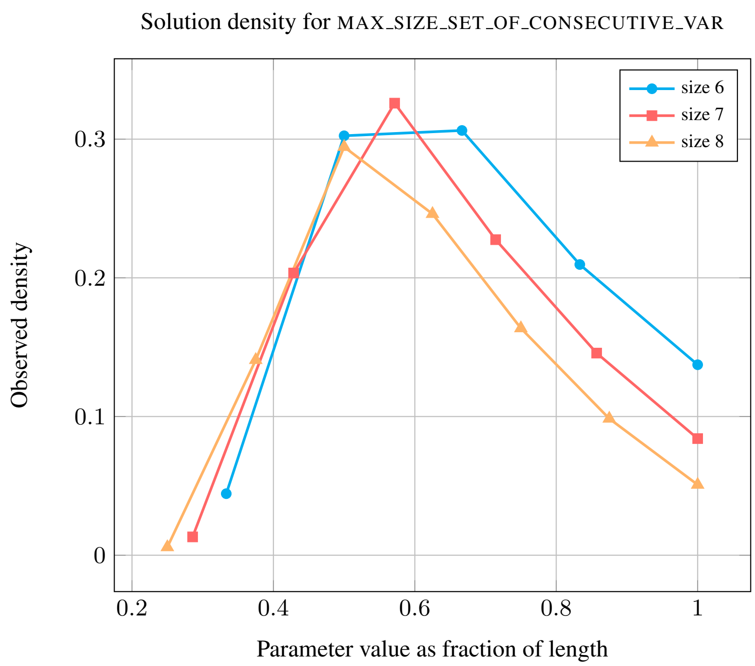


Solution density for MAX_SIZE_SET_OF_CONSECUTIVE_VAR
Observed density
1.2
1.1
1
0.9
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	2	-	-	-	-	-	-
	2	7	30	168	720	5220	27720	249480
	3	-	34	240	3080	35580	426720	6059760
	4	-	-	217	2260	36030	683550	12672940
	5	-	-	-	1716	24660	477162	10592848
	6	-	-	-	-	16159	305634	7044632
	7	-	-	-	-	-	176366	4239424
	8	-	-	-	-	-	-	2187637

Solution count for MAX_SIZE_SET_OF_CONSECUTIVE_VAR: domains $0..n$





See also **common keyword:** NSET_OF_CONSECUTIVE_VALUES *(consecutive values)*.

Keywords **characteristic of a constraint:** consecutive values, maximum.

constraint arguments: pure functional dependency.

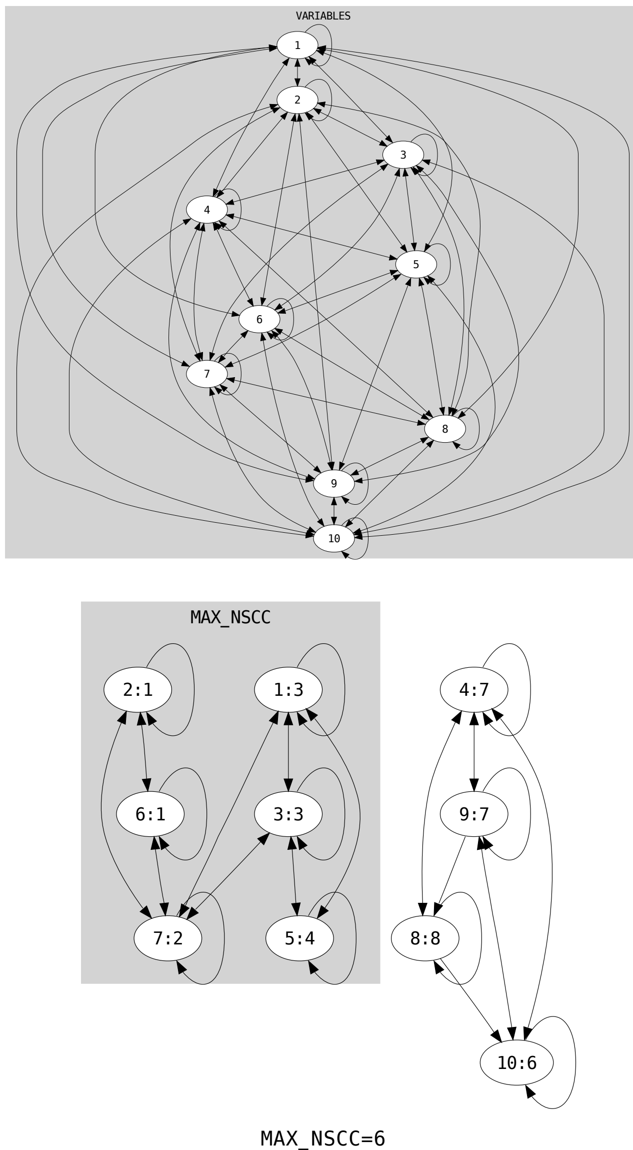
constraint type: value constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	abs(variables1.var − variables2.var) ≤ 1
Graph property(ies)	**MAX_NSCC** = MAX

Graph model Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.

Parts (A) and (B) of Figure 5.552 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **MAX_NSCC** graph property, we show the largest strongly connected component of the final graph.



(A)

(B)

Figure 5.552: Initial and final graph of the MAX_SIZE_SET_OF_CONSECUTIVE_VAR constraint

5.252 MAXIMUM

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint MAXIMUM(`MAX`, `VARIABLES`)

Synonym MAX.

Arguments

`MAX`	:	`dvar`
`VARIABLES`	:	`collection(var−dvar)`

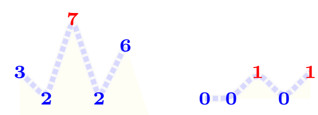
Restrictions

$|$`VARIABLES`$| > 0$
`required(VARIABLES, var)`

Purpose `MAX` is the maximum value of the collection of domain variables `VARIABLES`.

Example

(7, ⟨3, 2, 7, 2, 6⟩)
(1, ⟨0, 0, 1, 0, 1⟩)



The first MAXIMUM constraint holds since its first argument `MAX` = 7 is fixed to the maximum value of the collection ⟨3, 2, 7, 2, 6⟩.

Typical

$|$`VARIABLES`$| > 1$
`range(VARIABLES.var)` > 1

Typical model `nval(VARIABLES.var)` > 2

Symmetries

- Items of `VARIABLES` are permutable.
- All occurrences of two distinct values of `VARIABLES.var` can be swapped.
- One and the same constant can be added to `MAX` as well as to the `var` attribute of all items of `VARIABLES`.

Arg. properties

- Functional dependency: `MAX` determined by `VARIABLES`.
- Aggregate: `MAX(max)`, `VARIABLES(union)`.

Usage In some project scheduling problems one has to introduce dummy activities that correspond, for example, to the completion time of a given set of activities. In this context one can use the MAXIMUM constraint to get the maximum completion time of a set of tasks.

Remark Note that MAXIMUM is a constraint and not just a function that computes the maximum value of a collection of variables: potential values of `MAX` influence the variables of `VARIABLES`, and reciprocally potential values that can be assigned to variables of `VARIABLES` influence `MAX`.

The MAXIMUM constraint is called MAX in **JaCoP** (`http://www.jacop.eu/`).

Algorithm A filtering algorithm for the MAXIMUM constraint is described in [29].

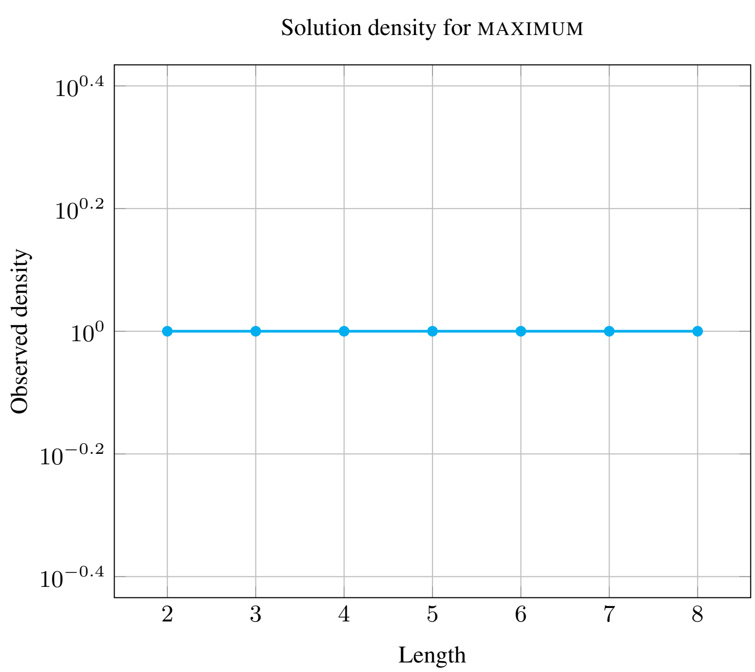
The MAXIMUM constraint is entailed if all the following conditions hold:

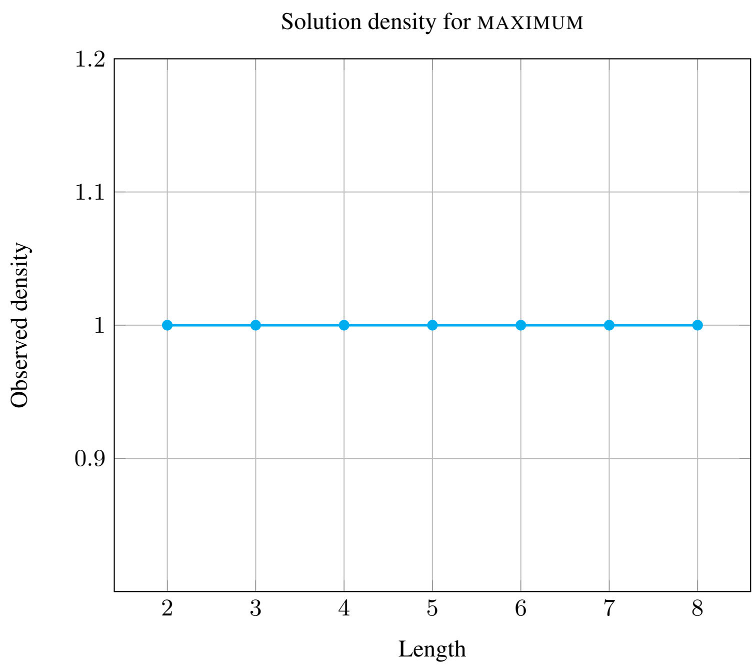
1. `MAX` is fixed.
2. At least one variable of `VARIABLES` is assigned value `MAX`.
3. All variables of `VARIABLES` have their maximum values less than or equal to value `MAX`.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

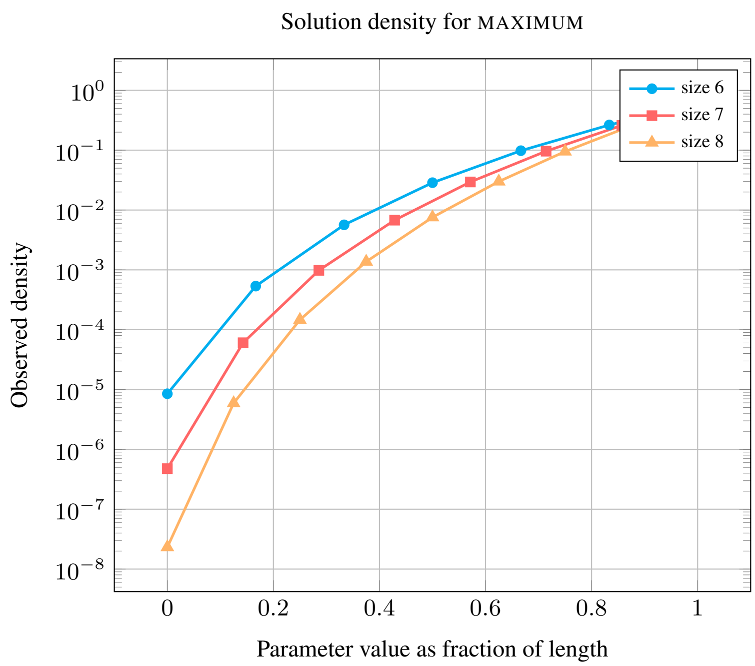
Number of solutions for MAXIMUM: domains $0..n$

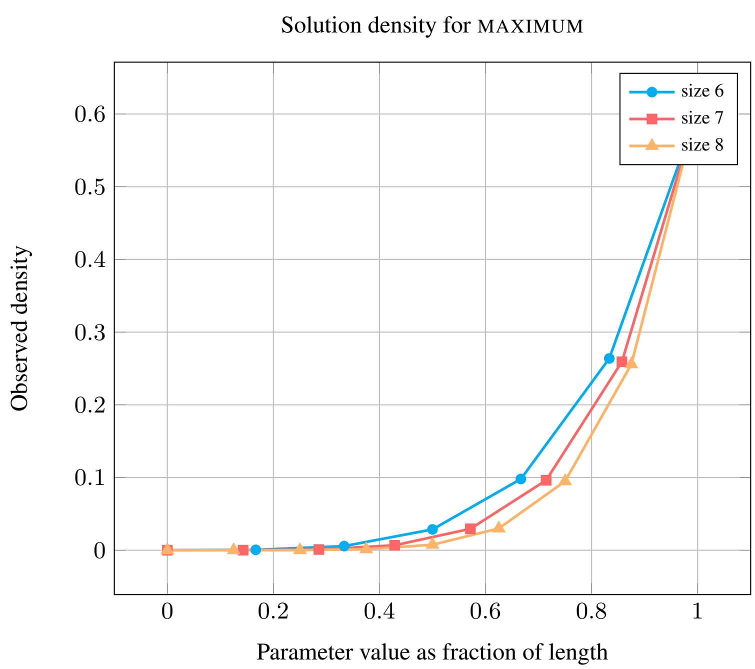




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	1	1	1	1	1	1	1
	1	3	7	15	31	63	127	255
	2	5	19	65	211	665	2059	6305
	3	-	37	175	781	3367	14197	58975
	4	-	-	369	2101	11529	61741	325089
	5	-	-	-	4651	31031	201811	1288991
	6	-	-	-	-	70993	543607	4085185
	7	-	-	-	-	-	1273609	11012415
	8	-	-	-	-	-	-	26269505

Solution count for MAXIMUM: domains $0..n$





Systems MAX in **Choco**, MAX in **Gecode**, MAX in **JaCoP**, MAXIMUM in **MiniZinc**, MAXIMUM in **SICStus**.

See also

common keyword: MINIMUM *(order constraint)*.

comparison swapped: MINIMUM.

generalisation: MAXIMUM_MODULO *(*`variable` *replaced by* `variable` mod `constant`*)*.

implied by: OR.

implies: BETWEEN_MIN_MAX, IN.

soft variant: OPEN_MAXIMUM *(open constraint)*.

specialisation: MAX_N *(maximum or order* `n` *replaced by absolute maximum)*.

uses in its reformulation: TREE_RANGE.

Keywords

characteristic of a constraint: maximum, automaton, automaton without counters, reified automaton constraint.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: centered cyclic(1) constraint network(1).

constraint type: order constraint.

filtering: glue matrix, arc-consistency, entailment.

modelling: balanced assignment, functional dependency.

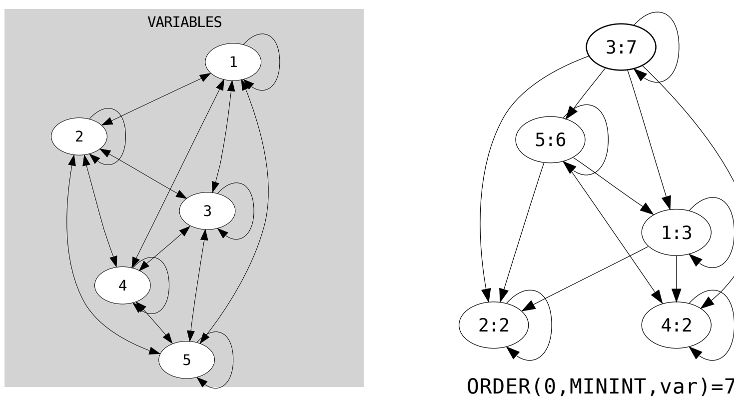
Cond. implications

MAXIMUM($\mathtt{MAX}, \mathtt{VARIABLES}$)
with $\mathtt{first}(\mathtt{VARIABLES.var}) < \mathtt{MAX}$
and $\mathtt{last}(\mathtt{VARIABLES.var}) < \mathtt{MAX}$
implies HIGHEST_PEAK($\mathtt{HEIGHT}, \mathtt{VARIABLES}$).

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	$\bigvee\left(\begin{array}{l}\texttt{variables1.key} = \texttt{variables2.key},\\ \texttt{variables1.var} > \texttt{variables2.var}\end{array}\right)$
Graph property(ies)	**ORDER**(0, MININT, var) = MAX

Graph model We use a similar definition that the one that was utilised for the MINIMUM constraint. Within the arc constraint, we replace the comparison operator $<$ by $>$.

Parts (A) and (B) of Figure 5.553 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **ORDER** graph property, the vertex of rank 0 (without considering the loops) of the final graph is outlined with a thick circle.



(A) **(B)**

Figure 5.553: Initial and final graph of the MAXIMUM constraint

Automaton Figure 5.554 depicts the automaton associated with the MAXIMUM constraint. Let $\texttt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each pair $(\texttt{MAX}, \texttt{VAR}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\texttt{MAX} > \texttt{VAR}_i \Leftrightarrow S_i = 0) \wedge (\texttt{MAX} = \texttt{VAR}_i \Leftrightarrow S_i = 1) \wedge (\texttt{MAX} < \texttt{VAR}_i \Leftrightarrow S_i = 2)$.

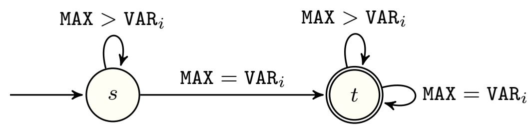


Figure 5.554: Counter free automaton of the MAXIMUM constraint

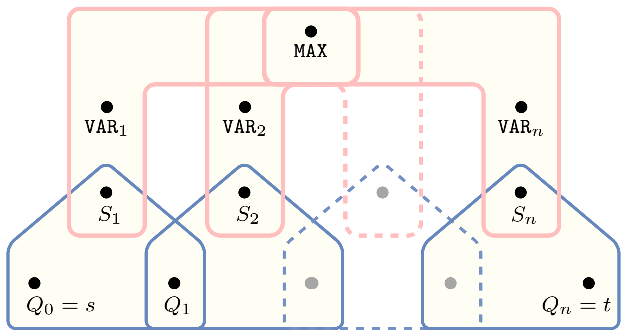


Figure 5.555: Hypergraph of the reformulation corresponding to the counter free automaton of the MAXIMUM constraint

Figure 5.556 depicts a second counter free non deterministic automaton associated with the MAXIMUM constraint, where the argument MAX is also part of the sequence passed to the automaton.

Figure 5.558 depicts a third deterministic automaton with one counter associated with the MAXIMUM constraint, where the argument MAX is unified to the final value of the counter.

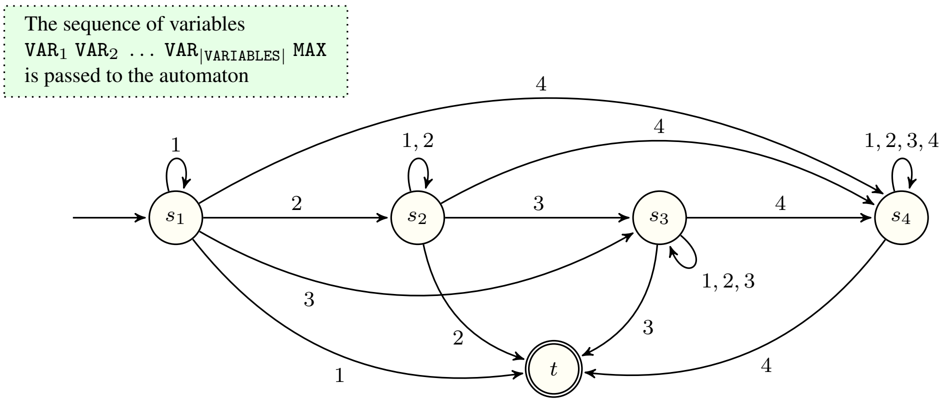


Figure 5.556: Counter free non deterministic automaton of the MAXIMUM(MAX, VARIABLES) constraint assuming that the union of the domain of the variables is the set $\{1, 2, 3, 4\}$ and that the elements of VARIABLES are first passed to the automaton followed by MAX (state s_i means that no value strictly greater than value i was found and that value i was already encountered at least once)

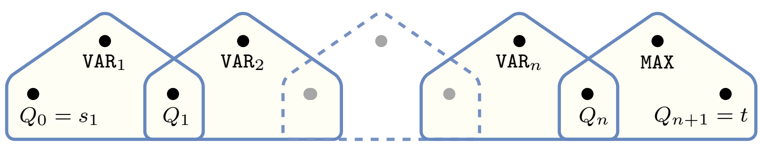


Figure 5.557: Hypergraph of the reformulation corresponding to the counter free non deterministic automaton of the MAXIMUM constraint

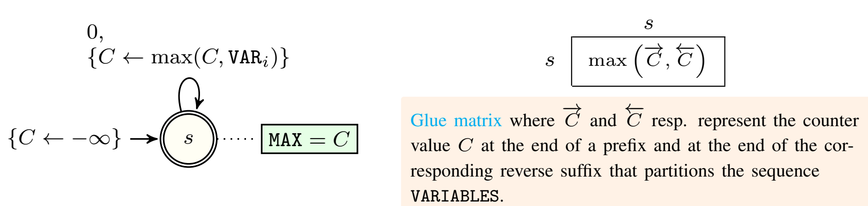


Figure 5.558: Automaton (with one counter) of the MAXIMUM constraint and its glue constraint

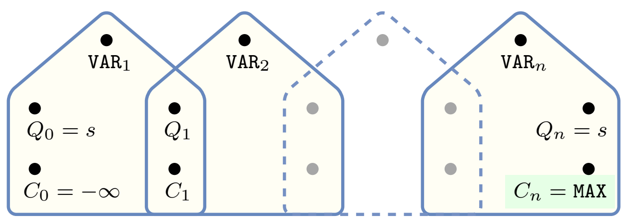


Figure 5.559: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the MAXIMUM constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.253 MAXIMUM_MODULO

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

Derived from MAXIMUM.

Constraint

MAXIMUM_MODULO(MAX, VARIABLES, M)

Arguments

MAX	:	dvar
VARIABLES	:	collection(var−dvar)
M	:	int

Restrictions

$|$VARIABLES$| > 0$
M > 0
required(VARIABLES, var)

Purpose

MAX is a maximum value of the collection of domain variables VARIABLES according to the following partial ordering: $(X \bmod \mathtt{M}) < (Y \bmod \mathtt{M})$.

Example

$(5, \langle 9, 1, 7, 6, 5 \rangle, 3)$

The MAXIMUM_MODULO constraint holds since its first argument MAX is set to value 5, where $5 \bmod 3 = 2$ is greater than or equal to all the expressions $9 \bmod 3 = 0$, $1 \bmod 3 = 1$, $7 \bmod 3 = 1$ and $6 \bmod 3 = 0$.

Typical

M > 1
M $<$maxval(VARIABLES.var)
$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1

Symmetry

Items of VARIABLES are permutable.

Arg. properties

Functional dependency: MAX determined by VARIABLES and M.

See also

comparison swapped: MINIMUM_MODULO.

specialisation: MAXIMUM (variable mod constant *replaced by* variable).

Keywords

characteristic of a constraint: modulo, maximum.

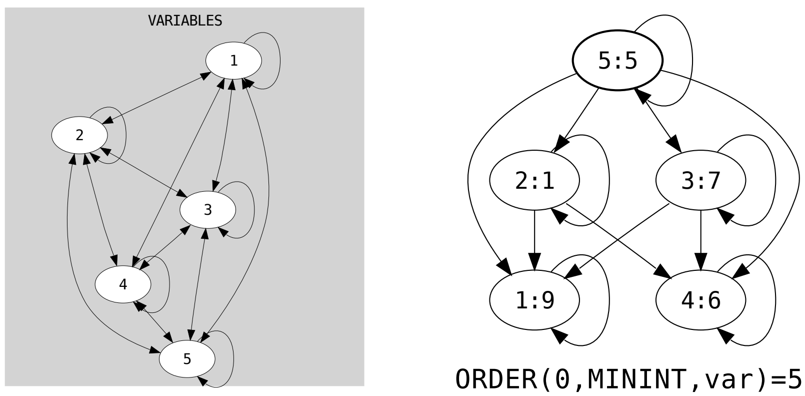
constraint arguments: pure functional dependency.

constraint type: order constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \texttt{variables1.key} = \texttt{variables2.key}, \\ \texttt{variables1.var} \bmod \texttt{M} > \texttt{variables2.var} \bmod \texttt{M} \end{array} \right)$
Graph property(ies)	**ORDER**(0, MININT, var) = MAX

Graph model Parts (A) and (B) of Figure 5.560 respectively show the initial and final graph associated with the **Example** slot. Since we use the **ORDER** graph property, the vertex of rank 0 (without considering the loops) of the final graph is outlined with a thick circle.



(A) **(B)**

Figure 5.560: Initial and final graph of the MAXIMUM_MODULO constraint

5.254 MEET_SBOXES

▶ ▷ ◁ ◀ DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [349]

Constraint MEET_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonym MEET.

Types

VARIABLES : collection(v−dvar)
INTEGERS : collection(v−int)
POSITIVES : collection(v−int)

Arguments

K : int
DIMS : sint
OBJECTS : collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES : collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

$|$VARIABLES$| \geq 1$
$|$INTEGERS$| \geq 1$
$|$POSITIVES$| \geq 1$
required(VARIABLES, v)
$|$VARIABLES$| =$ K
required(INTEGERS, v)
$|$INTEGERS$| =$ K
required(POSITIVES, v)
$|$POSITIVES$| =$ K
POSITIVES.v > 0
K > 0
DIMS ≥ 0
DIMS $<$ K
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
OBJECTS.oid ≥ 1
OBJECTS.oid $\leq |$OBJECTS$|$
OBJECTS.sid ≥ 1
OBJECTS.sid $\leq |$SBOXES$|$
$|$SBOXES$| \geq 1$
required(SBOXES, [sid, t, l])
SBOXES.sid ≥ 1
SBOXES.sid $\leq |$SBOXES$|$
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i \neq j$, O_i and O_j meet with respect to a set of dimensions depicted by `DIMS`. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a `K`-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id `sid`, shift offset `t`, and sizes `l`. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier `oid`, shape id `sid` and origin `x`.

Two objects O_i and object O_j *meet* with respect to a set of dimensions depicted by `DIMS` if and only if the two following conditions hold:

- For all shifted box s_i associated with O_i and for all shifted box s_j associated with O_j there exists a dimension $d \in$ `DIMS` such that (1) the start of s_i in dimension d is greater than or equal to the end of s_j in dimension d, or (2) the start of s_j in dimension d is greater than or equal to the end of s_i in dimension d (i.e., there is no overlap between the shifted box of O_i and the shifted box of O_j).
- There exists a shifted box s_i of O_i and there exists a shifted box s_j of O_j such that for all dimensions d (1) the end of s_i in dimension d is greater than or equal to the start of s_j in dimension d, and (2) the end of s_j in dimension d is greater than or equal to the start of s_i in dimension d (i.e., at least two shifted box of O_i and O_j are in contact).

Example

$$\left(\begin{array}{l} 2, \{0,1\}, \\ \left\langle \begin{array}{lll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 3,2\rangle, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,1\rangle, \\ \texttt{oid}-3 & \texttt{sid}-4 & \texttt{x}-\langle 3,4\rangle \end{array} \right\rangle, \\ \left\langle \begin{array}{lll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 1,0\rangle & \texttt{l}-\langle 1,3\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,2\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,1\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-4 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle \end{array} \right\rangle \end{array}\right)$$

Figure 5.561 shows the objects of the example. Since all the pairs of objects meet the MEET_SBOXES constraint holds.

Typical

$|$`OBJECTS`$| > 1$

Symmetries

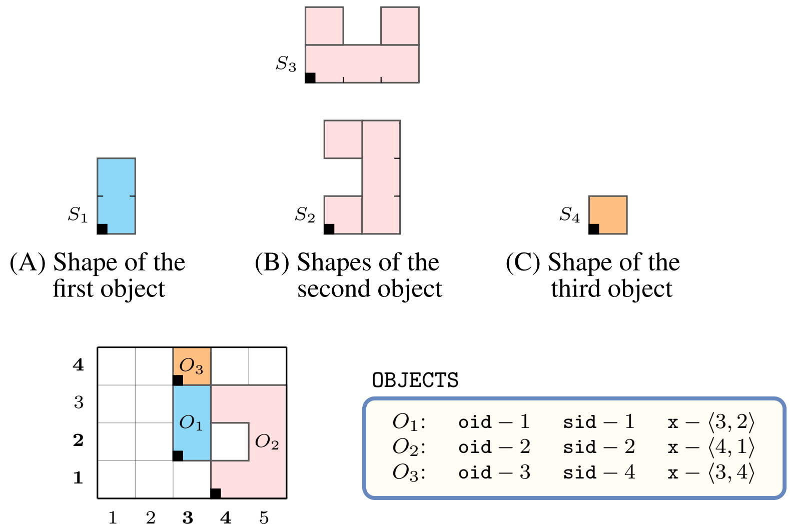
- Items of `OBJECTS` are permutable.
- Items of `SBOXES` are permutable.
- Items of `OBJECTS.x`, `SBOXES.t` and `SBOXES.l` are permutable (*same permutation used*).

Arg. properties

Suffix-contractible wrt. `OBJECTS`.

Remark

One of the eight relations of the *Region Connection Calculus* [349].



(A) Shape of the first object

(B) Shapes of the second object

(C) Shape of the third object

(D) Three objects for which each pair of objects meet

Figure 5.561: (D) the three pairwise meeting objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_1, S_2, S_4; (A), (B), (C) shapes S_1, S_2, S_3 and S_4 are respectively made up from 1, 3, 3 and 1 disjoint shifted box.

See also **common keyword:** CONTAINS_SBOXES, COVEREDBY_SBOXES, COVERS_SBOXES, DISJOINT_SBOXES, EQUAL_SBOXES, INSIDE_SBOXES *(rcc8)*, NON_OVERLAP_SBOXES *(geometrical constraint,logic)*, OVERLAP_SBOXES *(rcc8)*.

Keywords **constraint type:** logic.

geometry: geometrical constraint, rcc8.

Logic

- $\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D})$
- $\texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D}) + \texttt{S1.l}(\texttt{D})$
- $$\texttt{non\_overlap\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=} \exists \texttt{D} \in \texttt{Dims} \bigvee \left(\begin{array}{l} \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \leq \texttt{origin}\left(\begin{array}{l}\texttt{O2},\\ \texttt{S2},\\ \texttt{D}\end{array}\right), \\ \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) \leq \texttt{origin}\left(\begin{array}{l}\texttt{O1},\\ \texttt{S1},\\ \texttt{D}\end{array}\right) \end{array} \right)$$
- $$\texttt{meet\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=} \exists \texttt{D} \in \texttt{Dims} \bigvee \left(\begin{array}{l} \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) = \texttt{origin}(\texttt{O2},\texttt{S2},\texttt{D}), \\ \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) = \texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \end{array} \right)$$
- $$\texttt{meet\_objects}(\texttt{Dims},\texttt{O1},\texttt{O2}) \stackrel{\text{def}}{=} \bigwedge \left(\begin{array}{l} \forall \texttt{S1} \in \texttt{sboxes}([\texttt{O1.sid}]) \\ \quad \forall \texttt{S2} \in \texttt{sboxes}\left(\left[\ \texttt{O2.sid}\ \right]\right) \\ \quad \texttt{non\_overlap\_sboxes}\left(\begin{array}{l}\texttt{Dims},\\ \texttt{O1},\\ \texttt{S1},\\ \texttt{O2},\\ \texttt{S2}\end{array}\right), \\ \exists \texttt{S1} \in \texttt{sboxes}([\texttt{O1.sid}]) \\ \quad \exists \texttt{S2} \in \texttt{sboxes}\left(\left[\ \texttt{O2.sid}\ \right]\right) \\ \quad \texttt{meet\_sboxes}\left(\begin{array}{l}\texttt{Dims},\\ \texttt{O1},\\ \texttt{S1},\\ \texttt{O2},\\ \texttt{S2}\end{array}\right) \end{array} \right)$$
- $$\texttt{all\_meet}(\texttt{Dims},\texttt{OIDS}) \stackrel{\text{def}}{=} \forall \texttt{O1} \in \texttt{objects}(\texttt{OIDS})\ \forall \texttt{O2} \in \texttt{objects}(\texttt{OIDS})\ \begin{array}{l}\texttt{O1.oid} < \\ \texttt{O2.oid}\end{array} \Rightarrow \texttt{meet\_objects}\left(\begin{array}{l}\texttt{Dims},\\ \texttt{O1},\\ \texttt{O2}\end{array}\right)$$
- $\texttt{all\_meet}(\texttt{DIMENSIONS},\texttt{OIDS})$

5.255 MIN_DECREASING_SLOPE

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Motivated by time series.

Constraint MIN_DECREASING_SLOPE(MIN, VARIABLES)

Arguments

MIN : dvar
VARIABLES : collection(var−dvar)

Restrictions

MIN ≥ 0
MIN $<$range(VARIABLES.var)
required(VARIABLES, var)
$|$VARIABLES$| > 0$

Purpose Given a sequence of variables VARIABLES $= V_1, V_2, \ldots, V_n$, sets MIN to 0 if $\nexists i \in [1, n-1] | V_i > V_{i+1}$, otherwise sets MIN to $\min_{i\in[1,n-1]|V_i>V_{i+1}} (V_i - V_{i+1})$.

Example

$(2, \langle 1, 1, 5, 8, 6, 2, 4, 1, 5\rangle)$
$(0, \langle 1, 1, 1, 3, 4, 7, 7, 7, 9\rangle)$
$(9, \langle 1, 1, 9, 0, 4, 7, 7, 7, 9\rangle)$

The first MIN_DECREASING_SLOPE constraint holds since the sequence 1 1 5 8 6 2 4 1 5 contains two decreasing subsequences 8 6 2 and 4 1 and the minimum slope is equal to $\min(8-6, 6-2, 4-1) = 2$ as shown on Figure 5.562.

Typical

MIN > 1
$|$VARIABLES$| > 2$
range(VARIABLES.var) > 2

Typical model nval(VARIABLES.var) > 2

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: MIN determined by VARIABLES.

Usage Getting the minimum slope over the decreasing sequences of time series.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for MIN_DECREASING_SLOPE: domains $0..n$

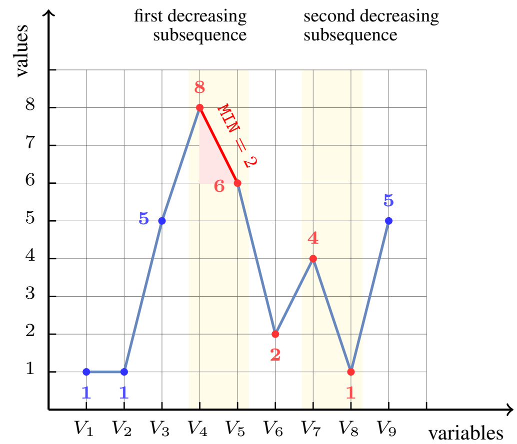
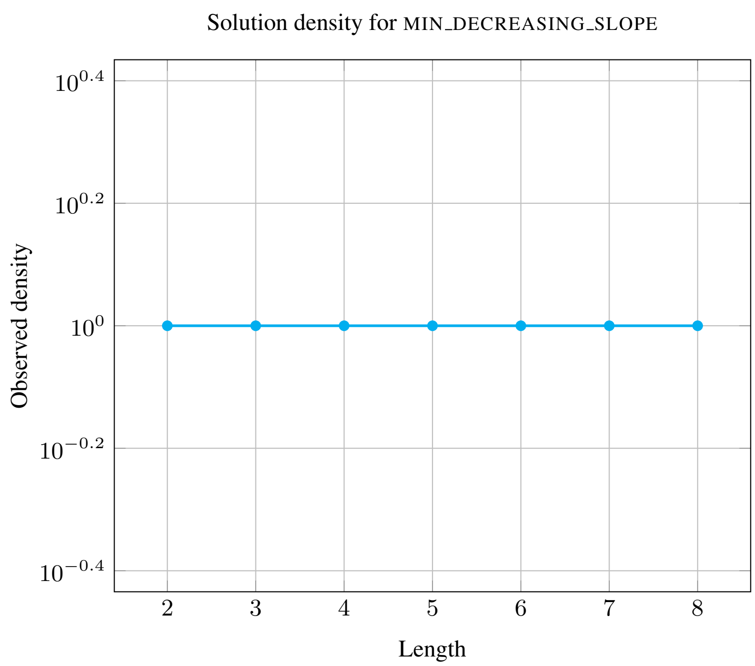
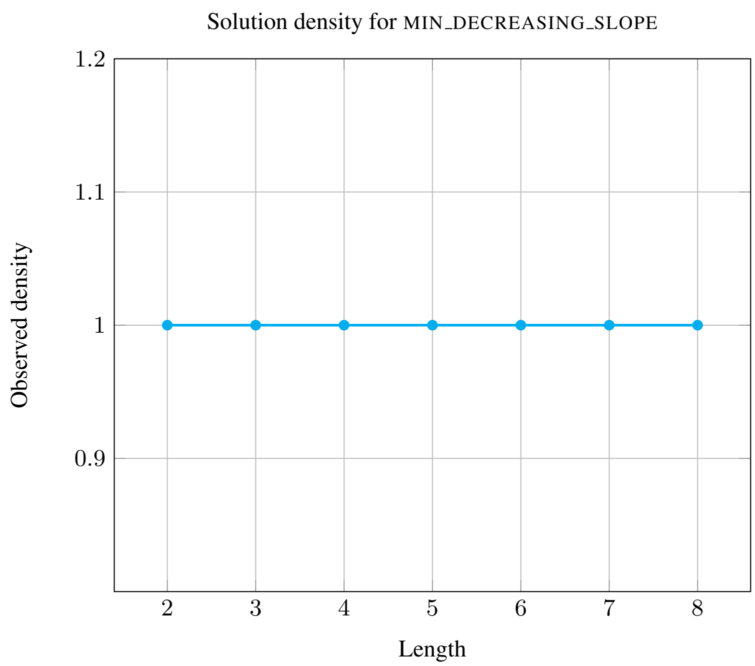


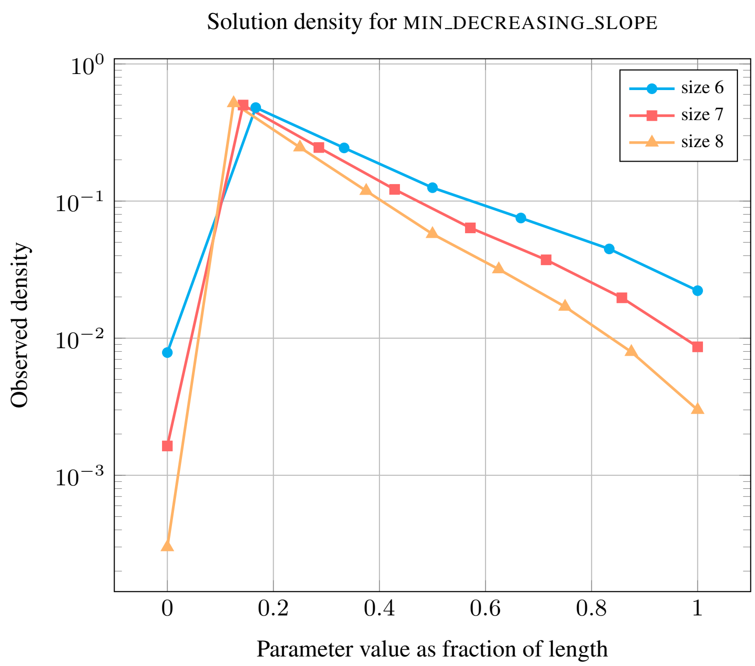
Figure 5.562: Illustration of the first example of the **Example** slot: a sequence of nine variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9 respectively fixed to values 1, 1, 5, 8, 6, 2, 4, 1, 5 and the corresponding minimum slope on the strictly decreasing subsequences 8 6 2 and 4 1 ($\mathtt{MIN} = 2$)

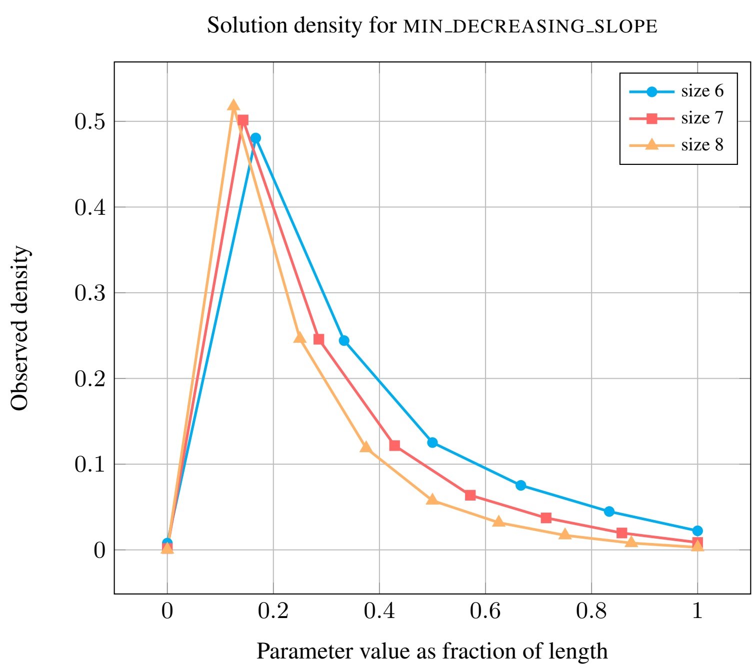




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	6	20	70	252	924	3432	12870
	1	2	22	256	3512	56537	1051936	22280084
	2	1	14	145	1864	28728	515372	10601773
	3	-	8	98	1062	14729	255076	5106480
	4	-	-	56	704	8853	133672	2475484
	5	-	-	-	382	5266	78198	1369232
	6	-	-	-	-	2612	41330	730161
	7	-	-	-	-	-	18136	341618
	8	-	-	-	-	-	-	129019

Solution count for MIN_DECREASING_SLOPE: domains $0..n$





Keywords **characteristic of a constraint:** automaton, automaton with counters.
combinatorial object: sequence.
constraint arguments: reverse of a constraint, pure functional dependency.

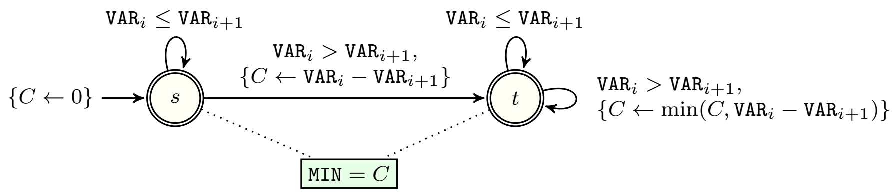
filtering: glue matrix.

modelling: functional dependency.

Cond. implications

MIN_DECREASING_SLOPE(MIN, VARIABLES)
  with range(VARIABLES.var) $=$ MIN $+ 1$
implies MAX_DECREASING_SLOPE(MAX, VARIABLES)
  when range(VARIABLES.var) $=$ MAX $+ 1$.

Automaton Figure 5.563 depicts the automaton associated with the MIN_DECREASING_SLOPE constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and S_i: $(\texttt{VAR}_i \leq \texttt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow S_i = 1)$.



Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	t
s	0	$\overleftarrow{C}$
t	$\overrightarrow{C}$	$\min(\overrightarrow{C}, \overleftarrow{C})$

Figure 5.563: Automaton for the MIN_DECREASING_SLOPE constraint and its glue matrix (note that the reverse of MIN_DECREASING_SLOPE is MIN_INCREASING_SLOPE)

5.256 MIN_DIST_BETWEEN_INFLEXION

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Derived from INFLEXION

Constraint MIN_DIST_BETWEEN_INFLEXION(MINDIST, VARIABLES)

Arguments

```
MINDIST   : int
VARIABLES : collection(var-dvar)
```

Restrictions

$\texttt{MINDIST} \geq 0$
$\texttt{MINDIST} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose

Given an integer value MINDIST and a sequence of variables VARIABLES enforce MINDIST to be greater than or equal to the smallest distance between two consecutive inflexions in the sequence VARIABLES, or to |VARIABLES| if no more than one inflexion exists.

An *inflexion* of a sequence of variables VARIABLES is a set of consecutive variables $V_i, V_{i+1}, \ldots, V_{j-1}, V_j$ $(i+1<j)$ such that one of the following conditions holds:

- $V_i < V_{i+1} \wedge V_{i+1} = \cdots = V_{j-1} \wedge V_{j-1} > V_j$,
- $V_i > V_{i+1} \wedge V_{i+1} = \cdots = V_{j-1} \wedge V_{j-1} < V_j$.

In this context, the index j is the *position* of the inflexion (i.e., the first instant when the inflexion is discovered when scanning the sequence of variables VARIABLES from left to right. The *distance between two consecutive inflexions* is the absolute value of the difference of their corresponding positions.

Example

$(2, \langle 2, 2, 3, 3, 2, 2, 1, 4, 4, 3\rangle)$

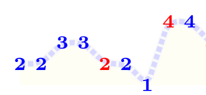


Figure 5.564 shows the three inflexions associated with the sequence 2, 2, 3, 3, 2, 2, 1, 4, 4, 3 and their respective positions 5, 8 and 10 in red. The MIN_DIST_BETWEEN_INFLEXION constraint holds since its first argument MINDIST $=2$ is greater than or equal to the smallest distance 2 between two consecutive inflexions of the sequence of variables VARIABLES.

Typical

$\texttt{MINDIST} > 1$
$|\texttt{VARIABLES}| > 3$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

Typical model

$\texttt{nval}(\texttt{VARIABLES.var}) > 2$

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

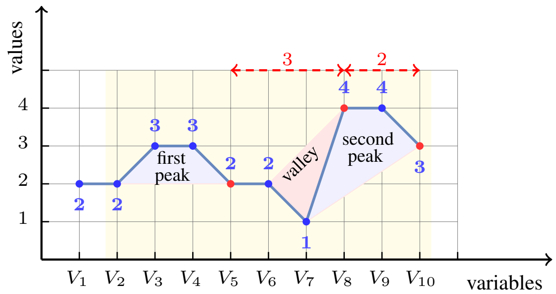
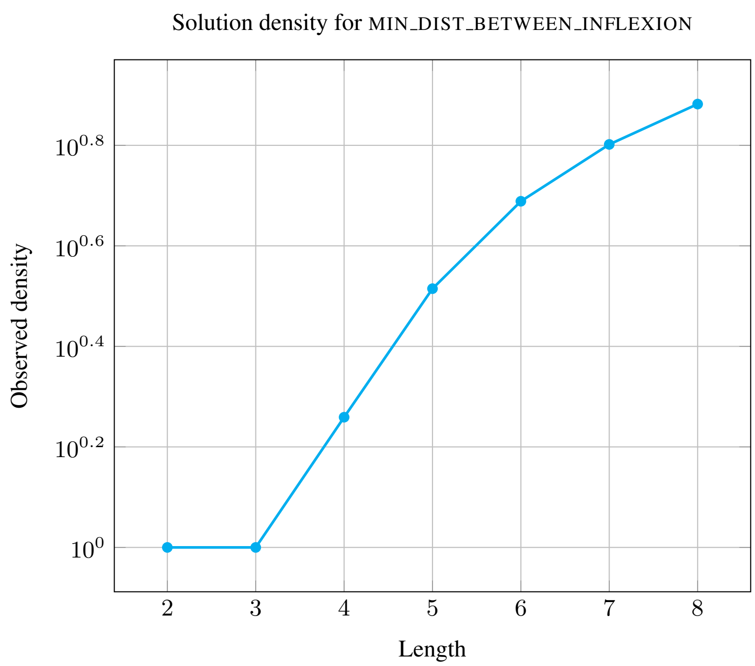


Figure 5.564: Illustration of the **Example** slot: a sequence of ten variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10} respectively fixed to values 2, 2, 3, 3, 2, 2, 1, 4, 4, 3 and its three inflexions, two peaks and one valley; each red point denotes an instant where a new inflexion is discovered while scanning the sequence from left to right; as shown by the rightmost arrow, the minimum distance between two consecutive inflexions is equal to 2.

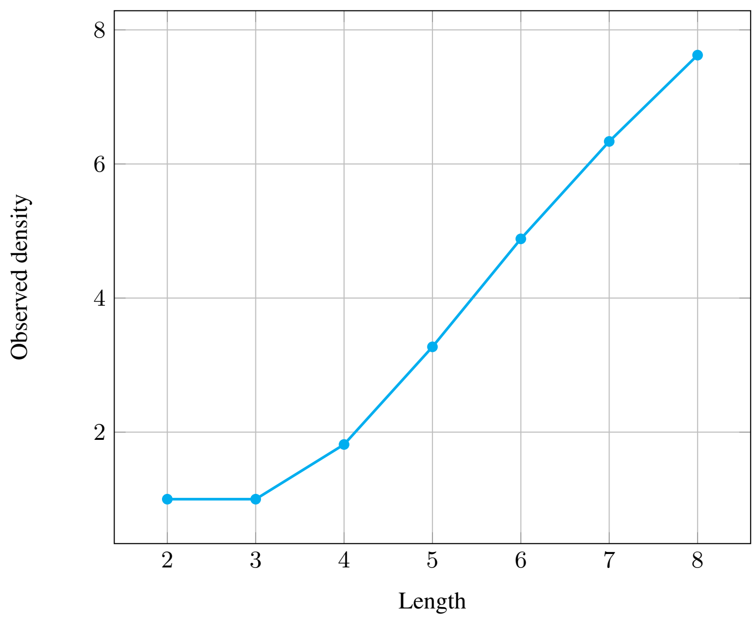
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	1135	25444	574483	13287476	328156407

Number of solutions for MIN_DIST_BETWEEN_INFLEXION: domains $0..n$



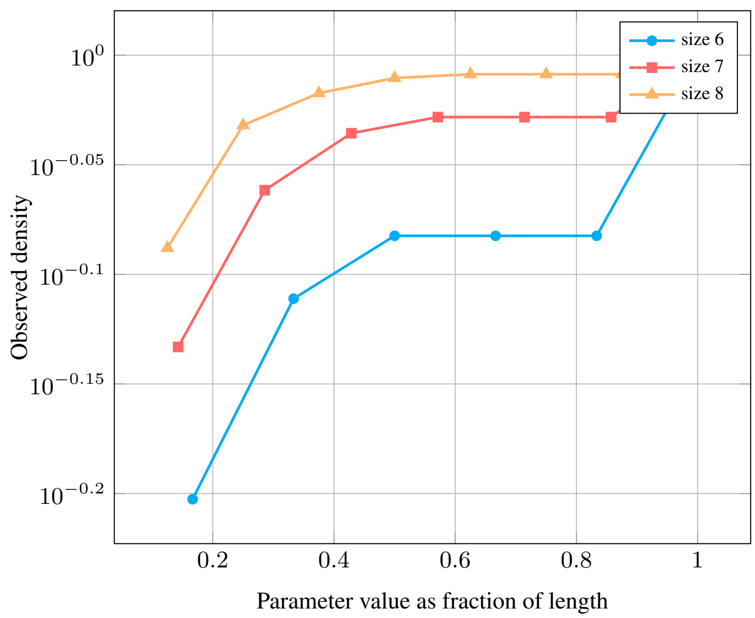
Solution density for MIN_DIST_BETWEEN_INFLEXION



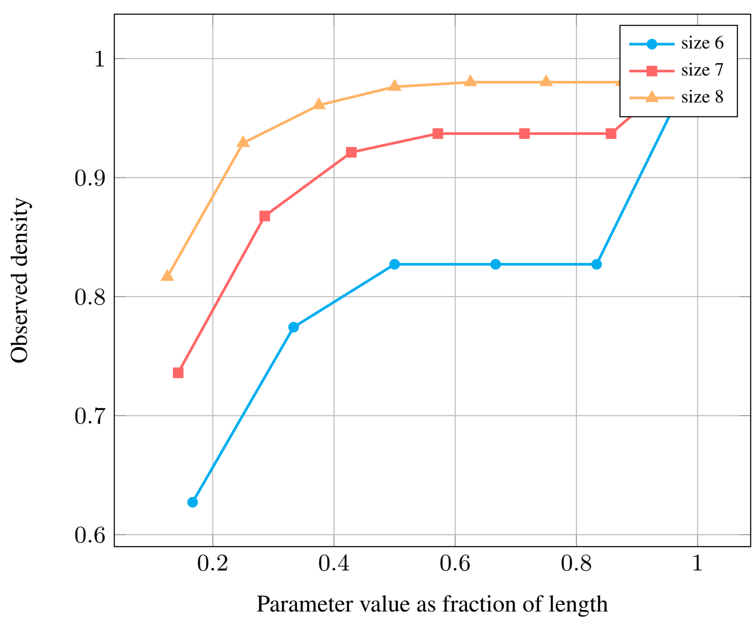
Length (n)		2	3	4	5	6	7	8
Total		9	64	1135	25444	574483	13287476	328156407
Parameter value	1	-	-	170	3598	73794	1543512	35152278
	2	9	-	170	4690	91098	1819764	39992562
	3	-	64	170	4690	97314	1932012	41360676
	4	-	-	625	4690	97314	1965012	42025560
	5	-	-	-	7776	97314	1965012	42192870
	6	-	-	-	-	117649	1965012	42192870
	7	-	-	-	-	-	2097152	42192870
	8	-	-	-	-	-	-	43046721

Solution count for MIN_DIST_BETWEEN_INFLEXION: domains $0..n$

Solution density for MIN_DIST_BETWEEN_INFLEXION



Solution density for MIN_DIST_BETWEEN_INFLEXION



See also **common keyword:** INFLEXION, LONGEST_DECREASING_SEQUENCE, LONGEST_INCREASING_SEQUENCE, PEAK, VALLEY *(sequence)*.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(3).

Automaton Figure 5.565 depicts the automaton associated with the MIN_DIST_BETWEEN_INFLEXION constraint.

STATE SEMANTICS

s	: stationary mode	$(=^*)$
$i0$	: increasing mode (no inflexion yet found)	$(< \{< \mid =\}^*)$
$d0$	: decreasing mode (no inflexion yet found)	$(> \{> \mid =\}^*)$
$i1$	: increasing mode (at least one inflexion already found)	$(< \{< \mid =\}^*)$
$d1$	: decreasing mode (at least one inflexion already found)	$(> \{> \mid =\}^*)$

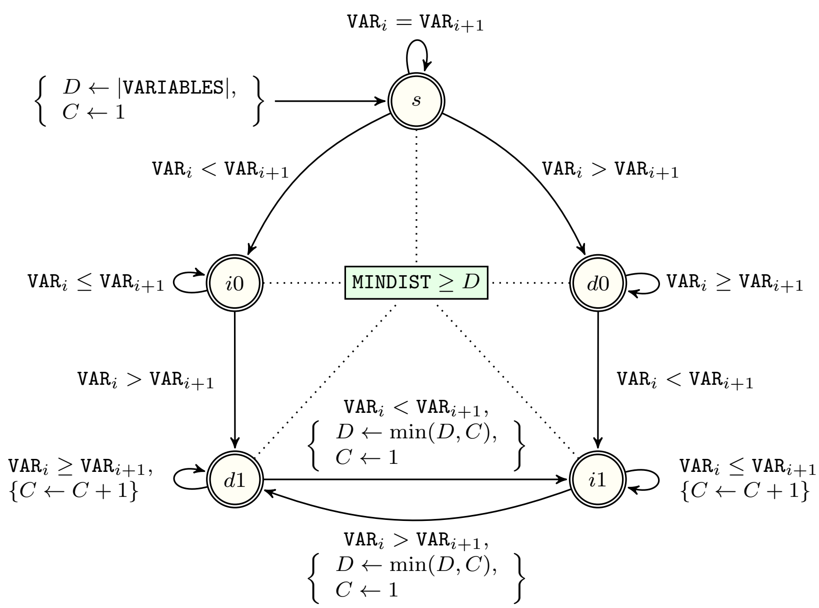


Figure 5.565: Automaton of the MIN_DIST_BETWEEN_INFLEXION constraint (state s means that we are in *stationary* mode, state $i0$ means that we are in *increasing* mode and that we did not yet found any inflexion, state $d0$ means that we are in *decreasing* mode and that we did not yet found any inflexion, state $i1$ means that we are in *increasing* mode and that we already found at least one inflexion, state $d1$ means that we are in *decreasing* mode and that we already found at least one inflexion, the minimum distance between two consecutive inflexions is updated each time we switch from $i1$ to $d1$ mode – or conversely from $d1$ to $i1$ mode – and the counter D is updated accordingly)

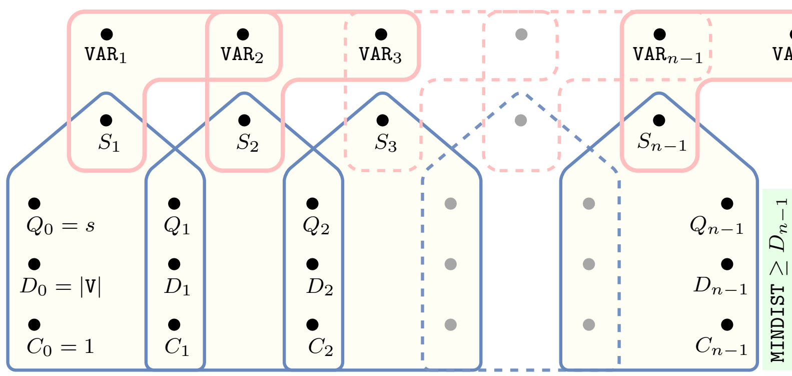


Figure 5.566: Hypergraph of the reformulation corresponding to the automaton of the MIN_DIST_BETWEEN_INFLEXION constraint where V is a shortcut for VARIABLES (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.257 MIN_INCREASING_SLOPE

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Motivated by time series.

Constraint MIN_INCREASING_SLOPE(MIN, VARIABLES)

Arguments

MIN : dvar
VARIABLES : collection(var−dvar)

Restrictions

MIN ≥ 0
MIN $<$range(VARIABLES.var)
required(VARIABLES, var)
$|$VARIABLES$| > 0$

Purpose Given a sequence of variables VARIABLES $= V_1, V_2, \ldots, V_n$, sets MIN to 0 if $\nexists i \in [1, n-1] | V_i < V_{i+1}$, otherwise sets MIN to $\min_{i \in [1,n-1] | V_i < V_{i+1}} (V_{i+1} - V_i)$.

Example

$(3, \langle 1, 1, 5, 8, 6, 2, 2, 1, 5 \rangle)$
$(0, \langle 8, 8, 2, 0, 0 \rangle)$
$(9, \langle 1, 1, 0, 9, 6 \rangle)$

The first MIN_INCREASING_SLOPE constraint holds since the sequence 1 1 5 8 6 2 2 1 5 contains two increasing subsequences 1 5 8 and 1 5 and the minimum slope is equal to $\min(5-1, 8-5, 5-1) = 3$ as shown on Figure 5.567.

Typical

MIN > 1
$|$VARIABLES$| > 2$
range(VARIABLES.var) > 2

Typical model nval(VARIABLES.var) > 2

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: MIN determined by VARIABLES.

Usage Getting the minimum slope over the increasing sequences of time series.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for MIN_INCREASING_SLOPE: domains $0..n$

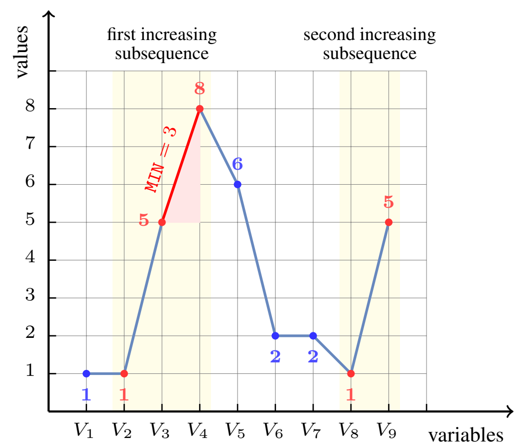
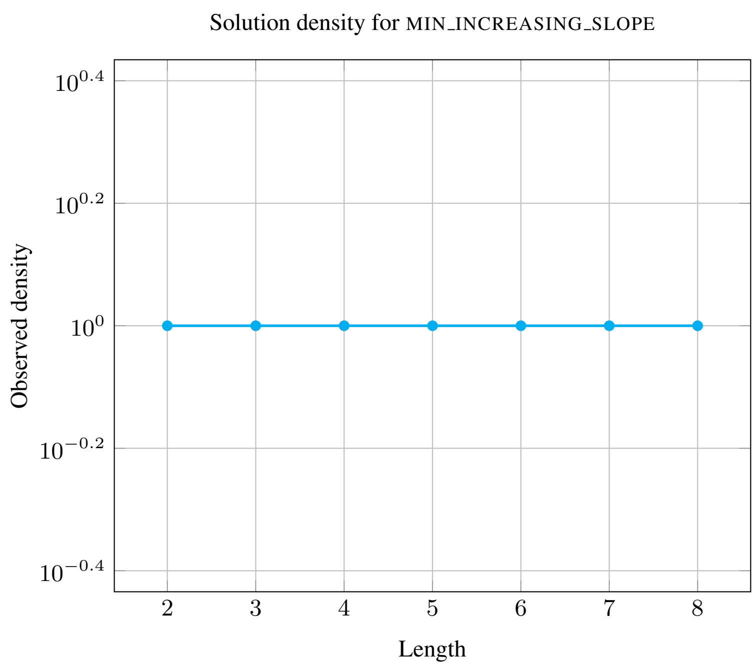
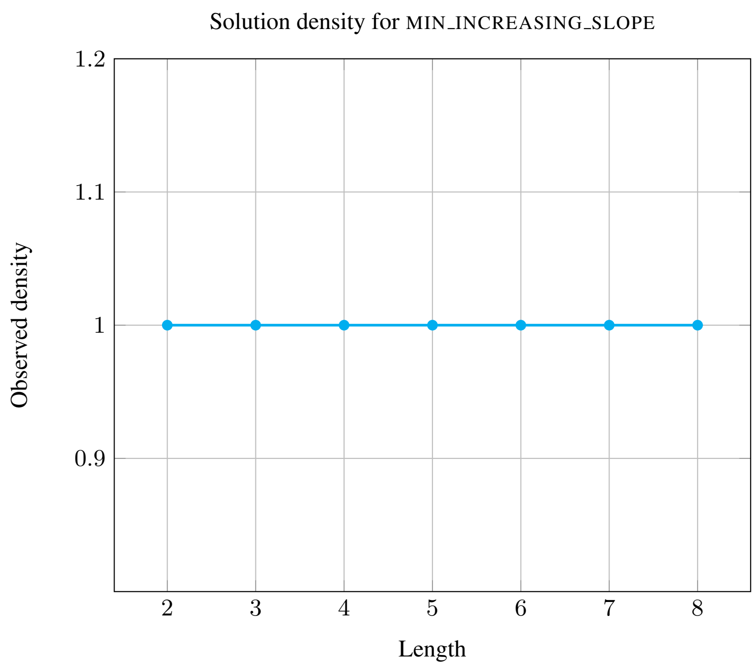


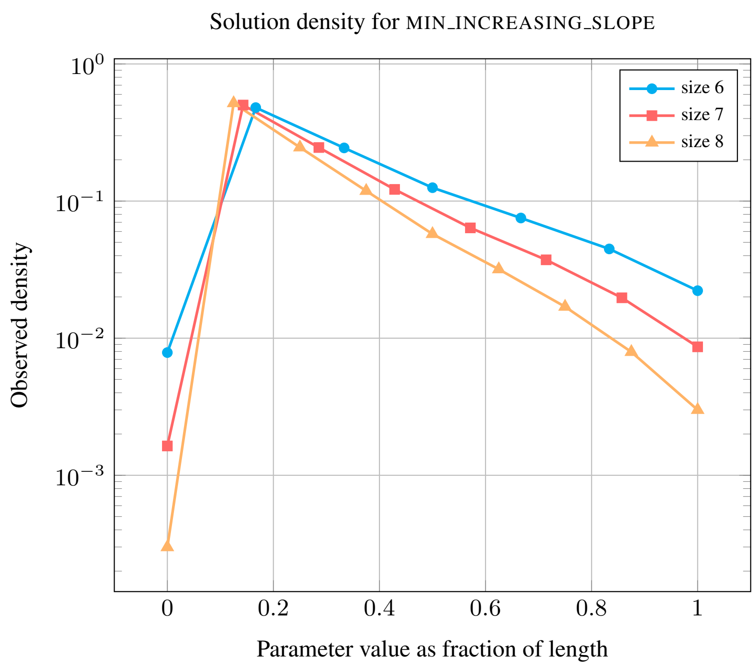
Figure 5.567: Illustration of the first example of the **Example** slot: a sequence of nine variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9 respectively fixed to values 1, 1, 5, 8, 6, 2, 2, 1, 5 and the corresponding minimum slope on the strictly increasing subsequences 1 5 8 and 1 5 ($\mathtt{MIN} = 3$)

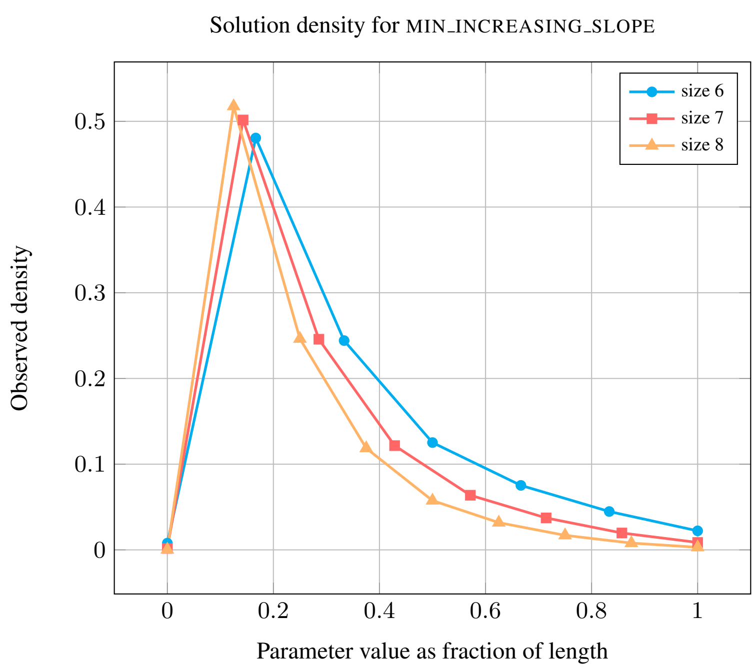




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	6	20	70	252	924	3432	12870
	1	2	22	256	3512	56537	1051936	22280084
	2	1	14	145	1864	28728	515372	10601773
	3	-	8	98	1062	14729	255076	5106480
	4	-	-	56	704	8853	133672	2475484
	5	-	-	-	382	5266	78198	1369232
	6	-	-	-	-	2612	41330	730161
	7	-	-	-	-	-	18136	341618
	8	-	-	-	-	-	-	129019

Solution count for MIN_INCREASING_SLOPE: domains $0..n$





Keywords **characteristic of a constraint:** automaton, automaton with counters.
combinatorial object: sequence.
constraint arguments: reverse of a constraint, pure functional dependency.

filtering: glue matrix.

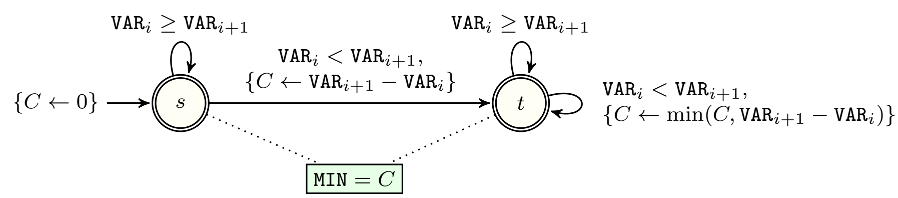
modelling: functional dependency.

Cond. implications

MIN_INCREASING_SLOPE(MIN, VARIABLES)
with range(VARIABLES.var) $=$ MIN $+ 1$
implies MAX_INCREASING_SLOPE(MAX, VARIABLES)
when range(VARIABLES.var) $=$ MAX $+ 1$.

Automaton

Figure 5.568 depicts the automaton associated with the MIN_INCREASING_SLOPE constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i \geq \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \land (\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1)$.



Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s	t
s	0	$\overleftarrow{C}$
t	$\overrightarrow{C}$	$\min(\overrightarrow{C}, \overleftarrow{C})$

Figure 5.568: Automaton for the MIN_INCREASING_SLOPE constraint and its glue matrix (note that the reverse of MIN_INCREASING_SLOPE is MIN_DECREASING_SLOPE)

5.258 MIN_INDEX

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint MIN_INDEX(MIN_INDEX, VARIABLES)

Arguments

MIN_INDEX : dvar
VARIABLES : collection(index−int, var−dvar)

Restrictions

$|\texttt{VARIABLES}| > 0$
$\texttt{MIN\_INDEX} \geq 0$
$\texttt{MIN\_INDEX} \leq |\texttt{VARIABLES}|$
required(VARIABLES, [index, var])
$\texttt{VARIABLES.index} \geq 1$
$\texttt{VARIABLES.index} \leq |\texttt{VARIABLES}|$
distinct(VARIABLES, index)

Purpose

MIN_INDEX is one of the indices of the collection of variables VARIABLES corresponding to its minimum value.

Example

$$\left(2, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{var}-3, \\ \texttt{index}-2 & \texttt{var}-2, \\ \texttt{index}-3 & \texttt{var}-7, \\ \texttt{index}-4 & \texttt{var}-2, \\ \texttt{index}-5 & \texttt{var}-6 \end{array} \right\rangle \right)$$
$$\left(4, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{var}-3, \\ \texttt{index}-2 & \texttt{var}-2, \\ \texttt{index}-3 & \texttt{var}-7, \\ \texttt{index}-4 & \texttt{var}-2, \\ \texttt{index}-5 & \texttt{var}-6 \end{array} \right\rangle \right)$$

The attribute var = 2 of the second and fourth items of the collection VARIABLES is the minimum value over values $3, 2, 7, 2, 6$. Consequently, both MIN_INDEX constraints hold since their first arguments MIN_INDEX are respectively set to 2 and 4.

Typical

$|\texttt{VARIABLES}| > 0$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

Symmetries

- Items of VARIABLES are permutable.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Usage Within the context of scheduling, assume the variables of the VARIABLES collection correspond to the starts of a set of tasks. Then MIN_INDEX gives the indexes of those tasks that can be scheduled first.

See also **comparison swapped:** MAX_INDEX.

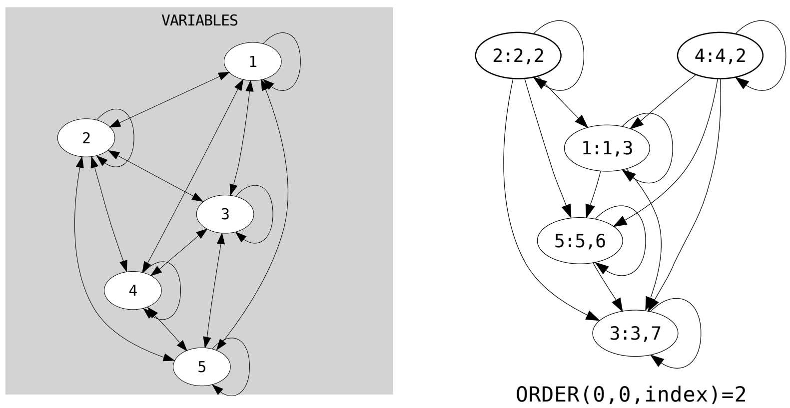
Keywords **characteristic of a constraint:** minimum.

constraint type: order constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	$\bigvee\left(\begin{array}{l}\texttt{variables1.key} = \texttt{variables2.key},\\ \texttt{variables1.var} < \texttt{variables2.var}\end{array}\right)$
Graph property(ies)	$\mathbf{ORDER}(0, 0, \texttt{index}) = \texttt{MIN\_INDEX}$

Graph model Parts (A) and (B) of Figure 5.569 respectively show the initial and final graph associated with the two examples of the **Example** slot. Since we use the **ORDER** graph property, the vertices of rank 0 (without considering the loops) of the final graph are outlined with a thick circle.



(A) (B)

Figure 5.569: Initial and final graph of the MIN_INDEX constraint

5.259 MIN_N

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [29]

Constraint MIN_N(MIN, RANK, VARIABLES)

Arguments

MIN : dvar
RANK : int
VARIABLES : collection(var−dvar)

Restrictions

$|\mathtt{VARIABLES}| > 0$
$\mathtt{RANK} \geq 0$
$\mathtt{RANK} < |\mathtt{VARIABLES}|$
required(VARIABLES, var)

Purpose MIN is the minimum value of rank RANK (i.e., the $\mathtt{RANK}^{th}$ smallest distinct value, identical values are merged) of the collection of domain variables VARIABLES. The minimum value has rank 0.

Example $(3, 1, \langle 3, 1, 7, 1, 6\rangle)$

The MIN_N constraint holds since its first argument $\mathtt{MIN} = 3$ is fixed to the second (i.e., $\mathtt{RANK} + 1$) smallest distinct value of the collection $\langle 3, 1, 7, 1, 6\rangle$. Note that identical values are only counted once: this is why the minimum of order 1 is 3 instead of 1.

Typical

$\mathtt{RANK} > 0$
$\mathtt{RANK} < 3$
$|\mathtt{VARIABLES}| > 1$
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES are permutable.
- One and the same constant can be added to MIN as well as to the var attribute of all items of VARIABLES.

Arg. properties Functional dependency: MIN determined by RANK and VARIABLES.

Algorithm [29].

Reformulation The constraint AMONG_VAR(1, ⟨MIN⟩, VARIABLES) enforces MIN to be assigned one of the values of VARIABLES. The constraint NVALUE(NVAL, VARIABLES) provides a hand on the number of distinct values assigned to the variables of VARIABLES. By associating to each variable V_i ($i \in [1, |\texttt{VARIABLES}|]$) of the VARIABLES collection a *rank* variable $R_i \in [0, |\texttt{VARIABLES}| - 1]$ with the reified constraint $R_i = \texttt{RANK} \Leftrightarrow V_i = \texttt{MIN}$, the inequality $R_i < \texttt{NVAL}$, and by creating for each pair of variables V_i, V_j ($i, j < i \in [1, |\texttt{VARIABLES}|]$) the reified constraints

$V_i < V_j \Leftrightarrow R_i < R_j$,
$V_i = V_j \Leftrightarrow R_i = R_j$,
$V_i > V_j \Leftrightarrow R_i > R_j$,

one can reformulate the MIN_N constraint in term of $3 \cdot \frac{|\texttt{VARIABLES}| \cdot (|\texttt{VARIABLES}| - 1)}{2} + 1$ reified constraints.

See also

comparison swapped: MAX_N.

generalisation: MINIMUM *(absolute minimum replaced by minimum or order* n*)*.

used in reformulation: AMONG_VAR, NVALUE.

Keywords

characteristic of a constraint: rank, minimum, maxint, automaton, automaton with array of counters.

constraint arguments: pure functional dependency.

constraint type: order constraint.

modelling: functional dependency.

Cond. implications

- MIN_N(MIN, RANK, VARIABLES)
  implies ATLEAST(N, VARIABLES, MIN)
  when $\texttt{N} = 1$.
- MIN_N(MIN, RANK, VARIABLES)
  with $\texttt{RANK} = 1$
  and $\texttt{minval}(\texttt{VARIABLES.var}) = 1$
  implies MINIMUM_GREATER_THAN(VAR1, VAR2, VARIABLES).

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	$\bigvee\left(\begin{array}{l}\texttt{variables1.key} = \texttt{variables2.key},\\ \texttt{variables1.var} < \texttt{variables2.var}\end{array}\right)$
Graph property(ies)	**ORDER**(RANK, MAXINT, var) = MIN

Graph model Parts (A) and (B) of Figure 5.570 respectively show the initial and final graph associated with the **Example** slot. Since we use the **ORDER** graph property, the vertex of rank 1 (without considering the loops) of the final graph is shown in grey.

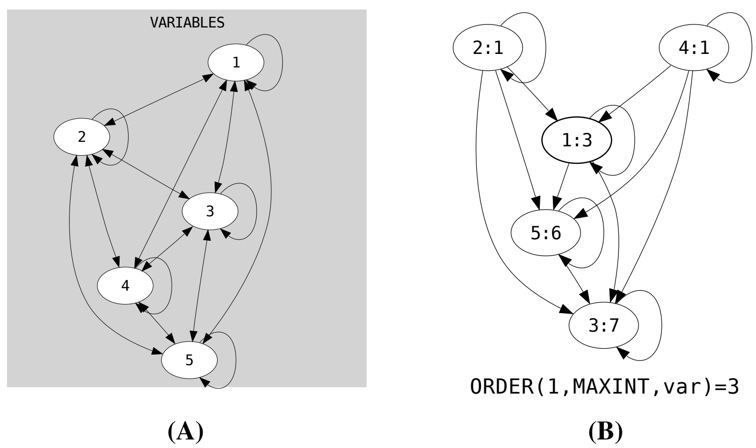


Figure 5.570: Initial and final graph of the MIN_N constraint

Automaton Figure 5.571 depicts the automaton associated with the MIN_N constraint. Figure 5.571 depicts the automaton associated with the MIN_N constraint. To each item of the collection VARIABLES corresponds a signature variable S_i that is equal to 1.

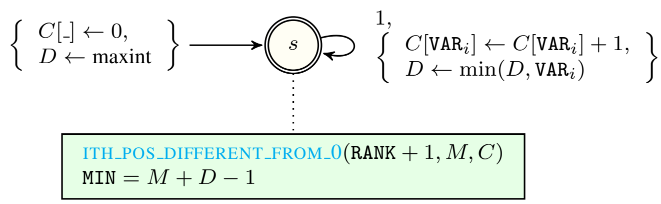


Figure 5.571: Automaton of the MIN_N constraint

5.260 MIN_NVALUE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint MIN_NVALUE(MIN, VARIABLES)

Arguments

MIN : dvar
VARIABLES : collection(var−dvar)

Restrictions

MIN ≥ 1
MIN $\leq$ |VARIABLES|
required(VARIABLES, var)

Purpose MIN is the minimum number of times that the same value is taken by the variables of the collection VARIABLES.

Example

$(2, \langle 9, 1, 7, 1, 1, 7, 7, 7, 7, 9\rangle)$
$(5, \langle 8, 8, 8, 8, 8\rangle)$
$(2, \langle 1, 8, 1, 8, 1\rangle)$

In the first example, values $1, 7, 9$ are respectively used $3, 5, 2$ times. So the minimum number of time MIN that a same value occurs is 2. Consequently the corresponding MIN_NVALUE constraint holds.

Typical

$2 *$ MIN $\leq$ |VARIABLES|
|VARIABLES| > 1
range(VARIABLES.var) > 1

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Functional dependency: MIN determined by VARIABLES.

Usage This constraint may be used in order to replace a set of COUNT or AMONG constraints were one would have to generate explicitly one constraint for each potential value. Also useful for constraining the number of occurrences of the less used value without knowing this value in advance and without giving explicitly a lower limit on the number of occurrences of each value as it is done in the GLOBAL_CARDINALITY constraint.

Reformulation Assume that VARIABLES is not empty. Let α and β respectively denote the smallest and largest possible values that can be assigned to the variables of the VARIABLES collection. Let the variables $O_\alpha, O_{\alpha+1}, \ldots, O_\beta$ respectively correspond to the number of occurrences of values $\alpha, \alpha+1, \ldots, \beta$ within the variables of the VARIABLES collection. The

MIN_NVALUE constraint can be expressed as the conjunction of the following two constraints:

GLOBAL_CARDINALITY (VARIABLES,
$\langle$val $-\ \alpha$ noccurrence $-\ O_\alpha$,
val $-\ \alpha+1$ noccurrence $-\ O_{\alpha+1}$,
$\dots$
val $-\ \beta$ noccurrence $-\ O_\beta\rangle$),

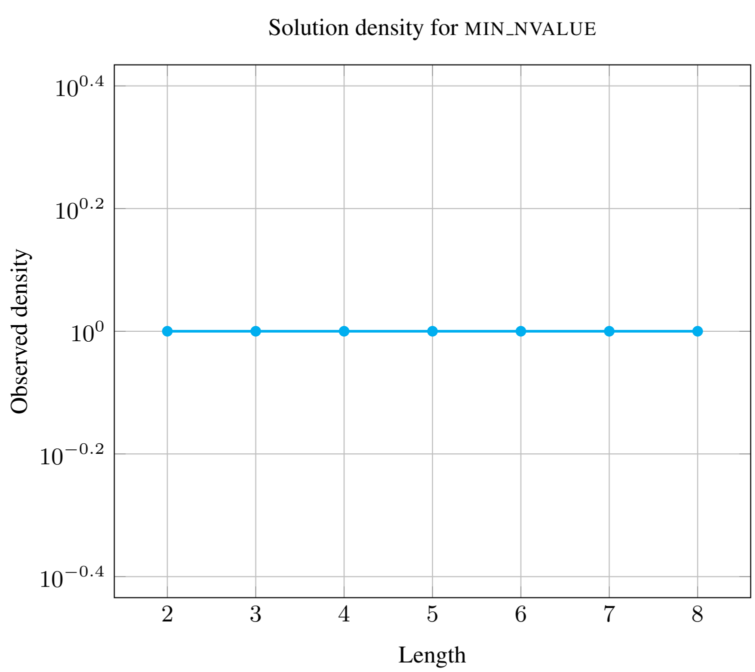
MIN_N(MIN, $1, \langle 0, O_\alpha, O_{\alpha+1}, \dots, O_\beta\rangle$).

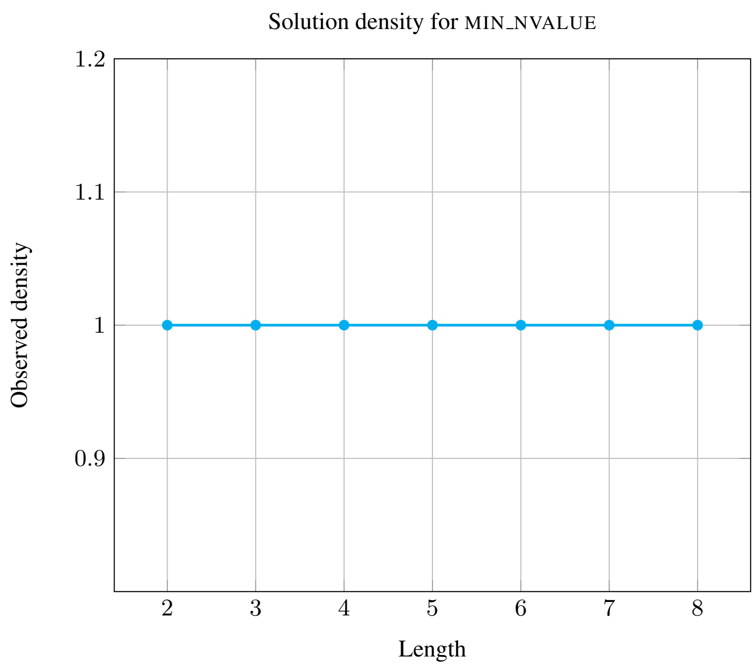
We use a MIN_N constraint (with its RANK parameter set to 1) instead of a MINIMUM constraint in order to discard the smallest value 0.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

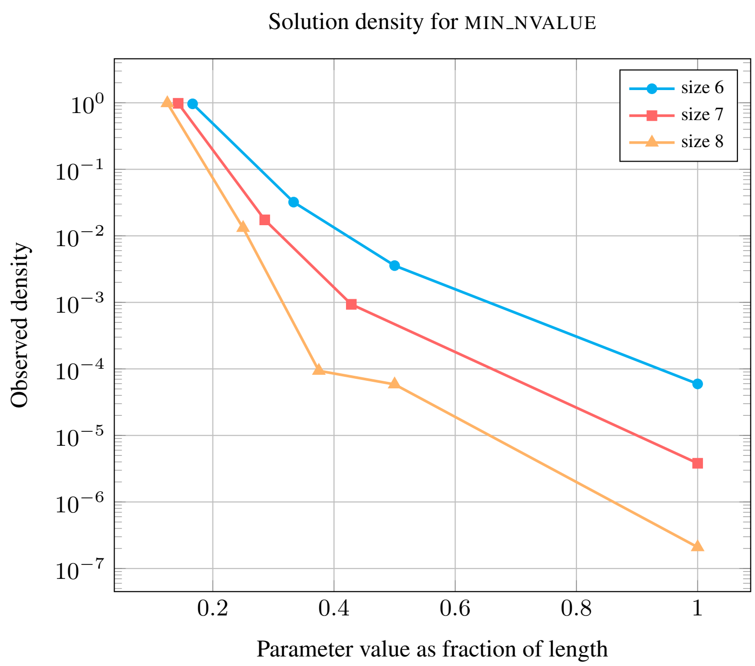
Number of solutions for MIN_NVALUE: domains $0..n$

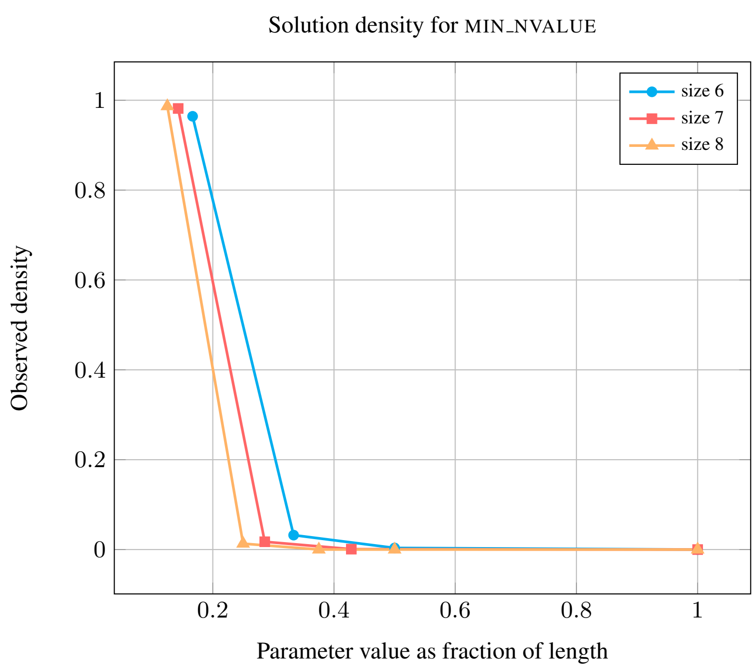




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	6	60	560	7470	113442	2058728	42473664
	2	3	-	60	300	3780	36456	566496
	3	-	4	-	-	420	1960	4032
	4	-	-	5	-	-	-	2520
	5	-	-	-	6	-	-	-
	6	-	-	-	-	7	-	-
	7	-	-	-	-	-	8	-
	8	-	-	-	-	-	-	9

Solution count for MIN_NVALUE: domains $0..n$





See also **common keyword:** AMONG *(counting constraint)*, COUNT, GLOBAL_CARDINALITY *(value constraint,counting constraint)*, MAX_NVALUE, NVALUE *(counting constraint)*.

Keywords

application area: assignment.

characteristic of a constraint: minimum, automaton, automaton with array of counters.

constraint arguments: pure functional dependency.

constraint type: value constraint, counting constraint.

final graph structure: equivalence.

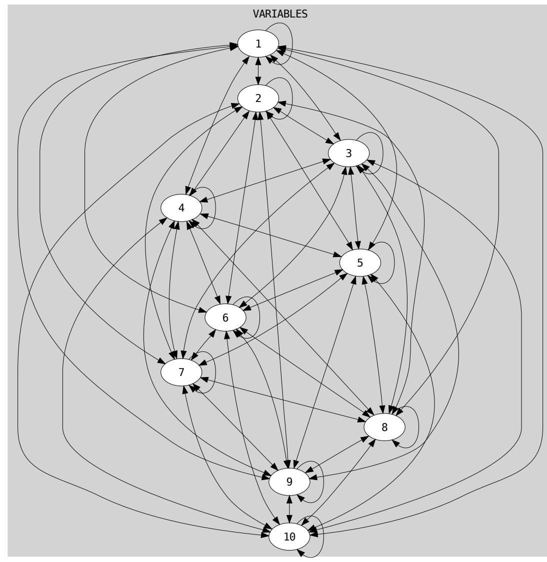
modelling: minimum number of occurrences, functional dependency.

Cond. implications

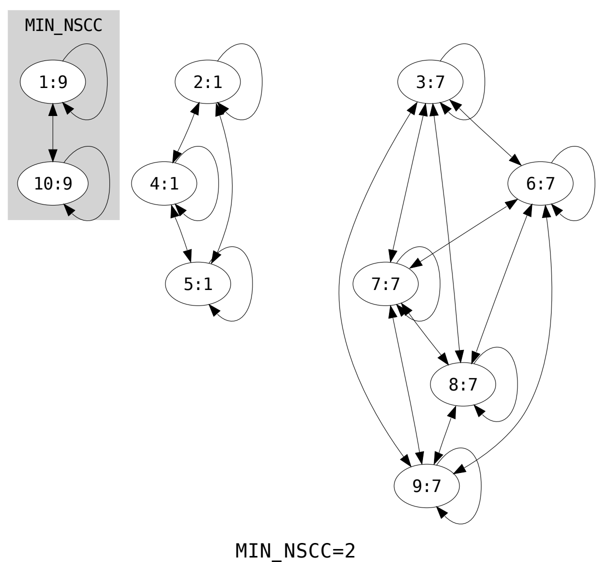
MIN_NVALUE($\mathtt{MIN}, \mathtt{VARIABLES}$)
with $\mathtt{MIN} < |\mathtt{VARIABLES}|$
implies ATLEAST_NVALUE($\mathtt{NVAL}, \mathtt{VARIABLES}$)
when $\mathtt{NVAL} = 2$.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**MIN_NSCC** = MIN

Graph model Parts (A) and (B) of Figure 5.572 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **MIN_NSCC** graph property, we show the smallest strongly connected component of the final graph associated with the first example of the **Example** slot.



(A)



(B)

Figure 5.572: Initial and final graph of the MIN_NVALUE constraint

Automaton Figure 5.573 depicts the automaton associated with the MIN_NVALUE constraint. To each item of the collection `VARIABLES` corresponds a signature variable S_i that is equal to 0.

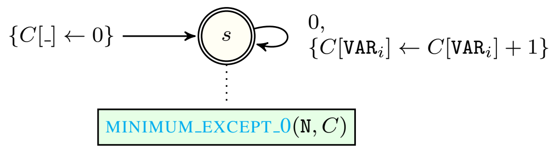


Figure 5.573: Automaton of the MIN_NVALUE constraint

5.261 MIN_SIZE_FULL_ZERO_STRETCH

DESCRIPTION LINKS AUTOMATON

Origin

Derived from the unit commitment problem

Constraint

MIN_SIZE_FULL_ZERO_STRETCH(MINSIZE, VARIABLES)

Arguments

MINSIZE : int
VARIABLES : collection(var−dvar)

Restrictions

MINSIZE ≥ 0
MINSIZE $\leq$ |VARIABLES|
required(VARIABLES, var)

Purpose

Given an integer MINSIZE and a sequence of variables VARIABLES enforce MINSIZE to be greater than or equal to the size of the smallest full stretch of zero of VARIABLES or to |VARIABLES| if no full stretch of zero exists.

A *stretch of zero* is a maximum sequence of zero, while a *full stretch of zero* is a stretch of zero that is neither located at the leftmost nor at the rightmost border of the sequence of variables VARIABLES. The *size of a stretch of zero* is the number of zero of the stretch.

Example

$(2, \langle 0, 2, 0, 0, 0, 2, 1, 0, 0, 3\rangle)$

Figure 5.574 shows the smallest full stretch of zero associated with the example. The MIN_SIZE_FULL_ZERO_STRETCH constraint holds since the size of the smallest full stretch of zero of the sequence 0 2 0 0 0 2 1 0 0 3 is greater than or equal to 2.

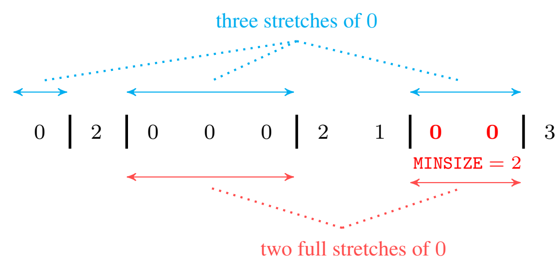


Figure 5.574: Illustration of the **Example** slot: smallest full stretch of zero in bold and red (MINSIZE $= 2$); note that the leftmost stretch of zero of size 1 is ignored since it is located at one of the two extremities of the sequence 0 2 0 0 0 2 1 0 0 3.

Typical

|VARIABLES| > 2
range(VARIABLES.var) > 1
|VARIABLES|−AMONG_DIFF_0(VARIABLES.var) > 1

Typical model

ATLEAST$(2, \texttt{VARIABLES}, 0)$

Symmetries

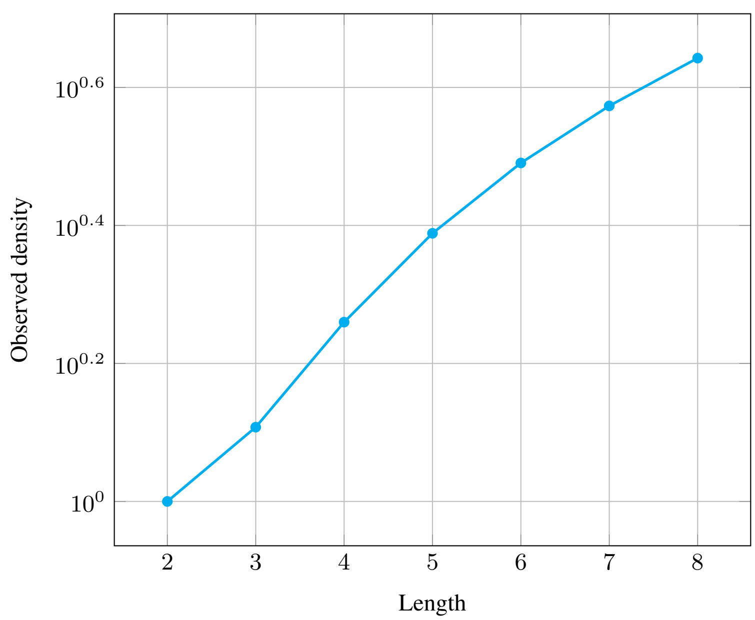
- Items of `VARIABLES` can be reversed.
- An occurrence of a value of `VARIABLES.var` that is different from 0 can be replaced by any other value that is also different from 0.

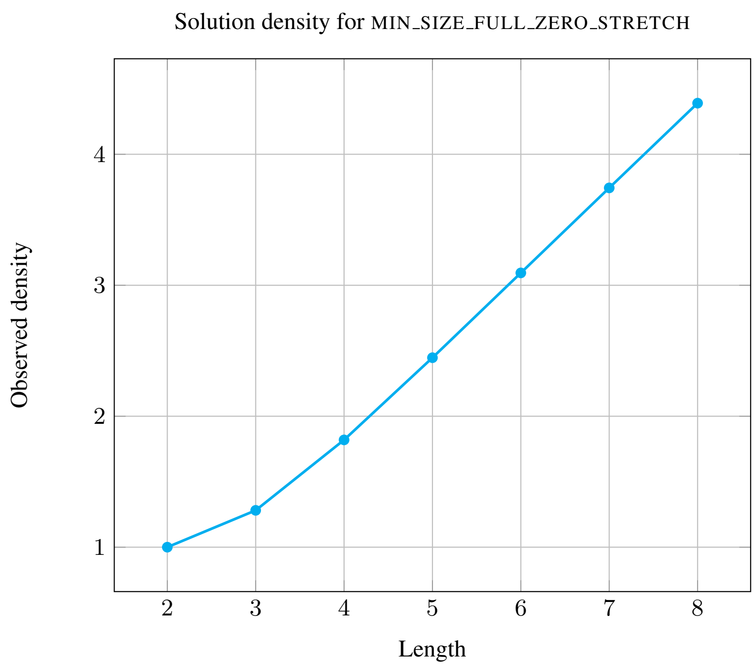
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	82	1137	19026	364033	7850291	188987201

Number of solutions for MIN_SIZE_FULL_ZERO_STRETCH: domains $0..n$

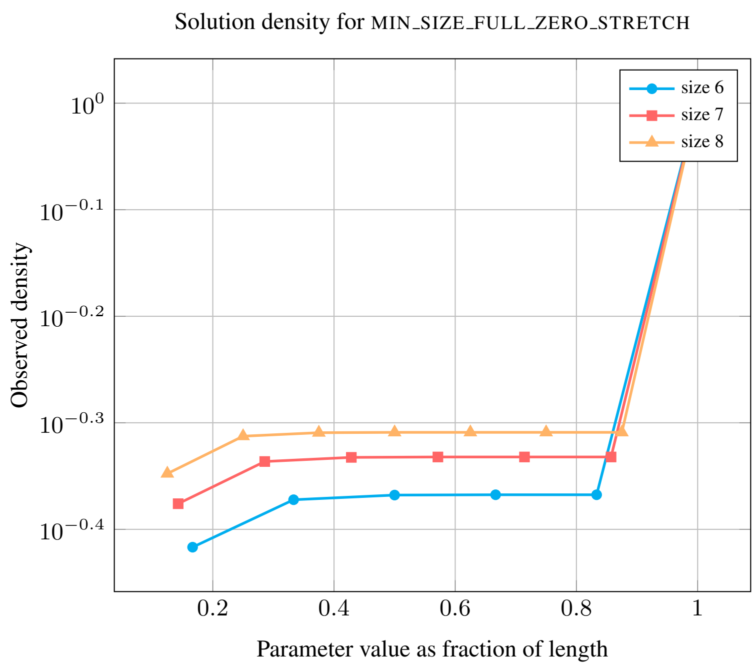
Solution density for MIN_SIZE_FULL_ZERO_STRETCH

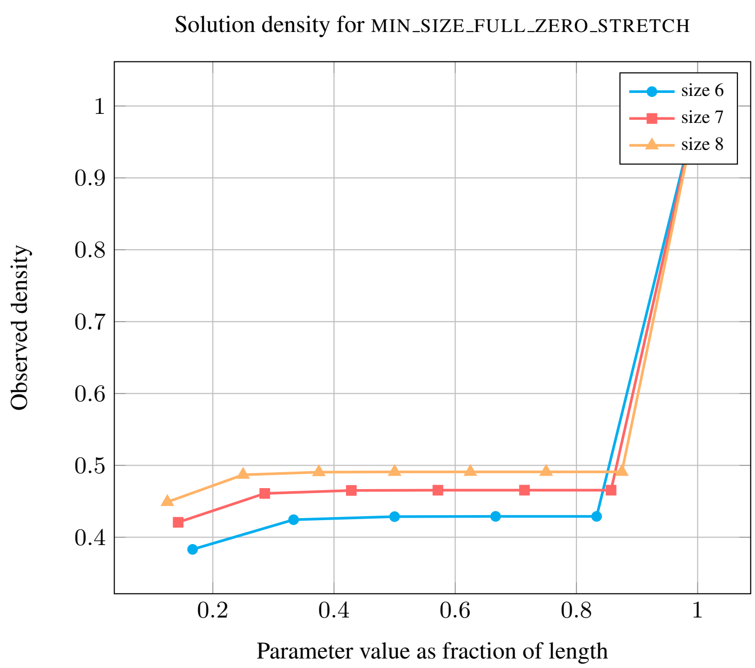




Length (n)		2	3	4	5	6	7	8
Total		9	82	1137	19026	364033	7850291	188987201
Parameter value	1	-	9	160	2575	45072	882441	19330432
	2	9	9	176	2875	49932	966672	20958912
	3	-	64	176	2900	50436	975394	21117888
	4	-	-	625	2900	50472	976178	21132416
	5	-	-	-	7776	50472	976227	21133568
	6	-	-	-	-	117649	976227	21133632
	7	-	-	-	-	-	2097152	21133632
	8	-	-	-	-	-	-	43046721

Solution count for MIN_SIZE_FULL_ZERO_STRETCH: domains $0..n$





See also **common keyword:** STRETCH_PATH *(sequence)*.

Keywords **characteristic of a constraint:** joker value, automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint network structure: alpha-acyclic constraint network(3).

Automaton Figure 5.575 depicts the automaton associated with the MIN_SIZE_FULL_ZERO_STRETCH constraint.

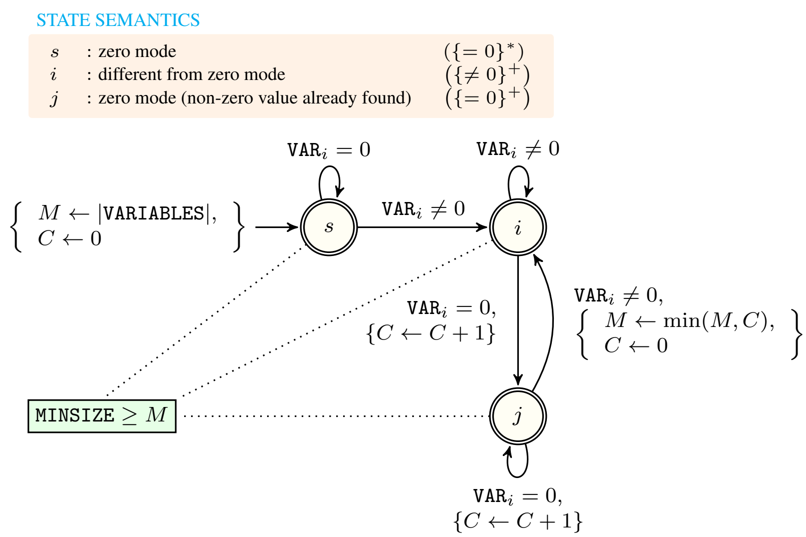


Figure 5.575: Automaton of the MIN_SIZE_FULL_ZERO_STRETCH constraint

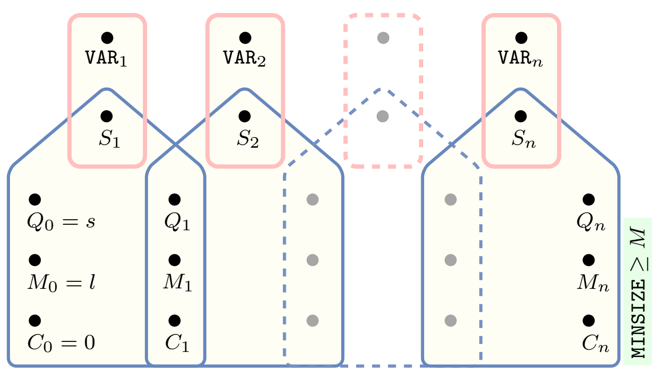


Figure 5.576: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the MIN_SIZE_FULL_ZERO_STRETCH constraint where $l = |\mathtt{VARIABLES}|$ (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

5.262 MIN_SIZE_SET_OF_CONSECUTIVE_VAR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint MIN_SIZE_SET_OF_CONSECUTIVE_VAR(MIN, VARIABLES)

Arguments

```
MIN       : dvar
VARIABLES : collection(var-dvar)
```

Restrictions

$\texttt{MIN} \geq 1$
$\texttt{MIN} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose MIN is the size of the smallest set of variables of the collection VARIABLES that all take their values in a set of consecutive values.

Example

$(4, \langle 3, 1, 3, 7, 4, 1, 2, 8, 7, 6\rangle)$
$(4, \langle 3, 1, 3, 2\rangle)$

In the first example, the two parts $3, 1, 3, 4, 1, 2$ and $7, 8, 7, 6$ take respectively their values in the two following sets of consecutive values $\{1, 2, 3, 4\}$ and $\{6, 7, 8\}$. Consequently, the corresponding MIN_SIZE_SET_OF_CONSECUTIVE_VAR constraint holds since the cardinality of the smallest set of variables is 4.

Typical

$\texttt{MIN} > 1$
$\texttt{MIN} < |\texttt{VARIABLES}|$
$|\texttt{VARIABLES}| > 0$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

Symmetries

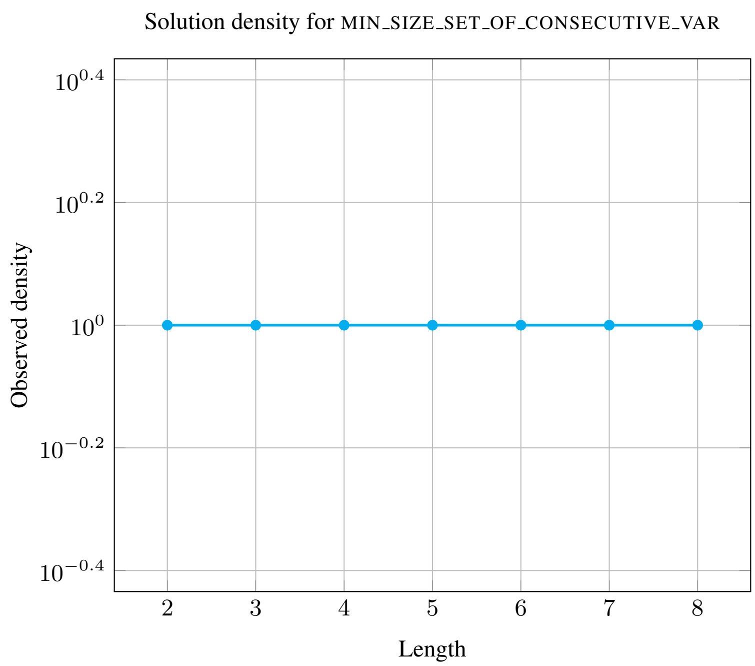
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

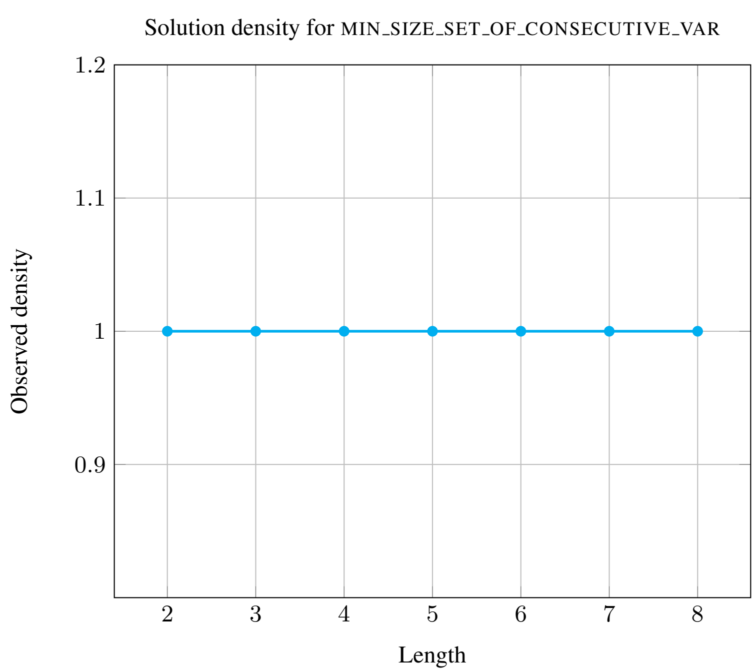
Arg. properties Functional dependency: MIN determined by VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for MIN_SIZE_SET_OF_CONSECUTIVE_VAR: domains $0..n$

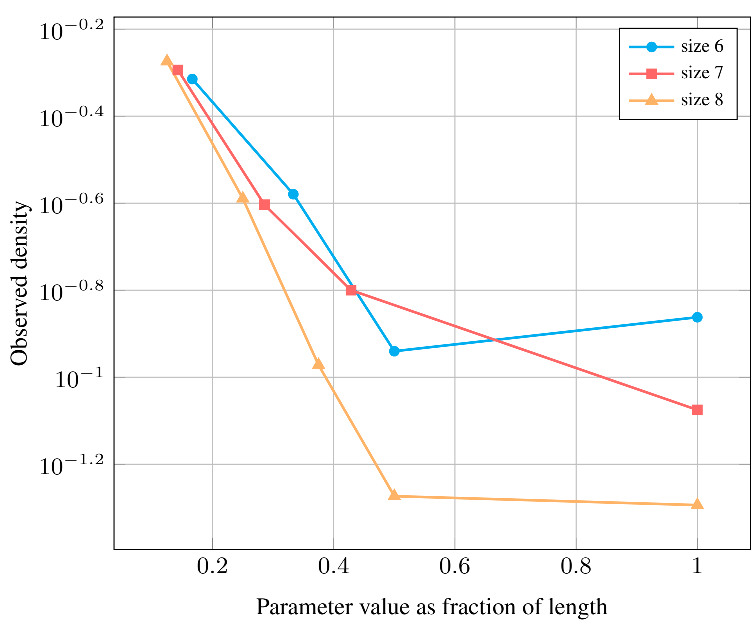

Solution density for MIN_SIZE_SET_OF_CONSECUTIVE_VAR
Observed density
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
2 3 4 5 6 7 8
Length

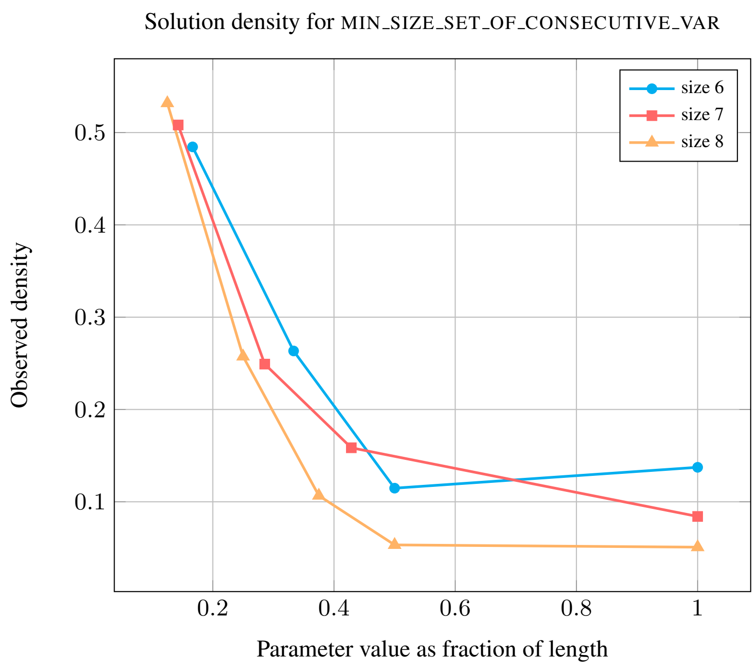

Solution density for MIN_SIZE_SET_OF_CONSECUTIVE_VAR
Observed density
1.2
1.1
1
0.9
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	2	30	276	3580	57000	1065834	22894984
	2	7	-	132	2480	30990	522522	11080412
	3	-	34	-	-	13500	332430	4590208
	4	-	-	217	-	-	-	2293480
	5	-	-	-	1716	-	-	-
	6	-	-	-	-	16159	-	-
	7	-	-	-	-	-	176366	-
	8	-	-	-	-	-	-	2187637

Solution count for MIN_SIZE_SET_OF_CONSECUTIVE_VAR: domains $0..n$

Solution density for MIN_SIZE_SET_OF_CONSECUTIVE_VAR





See also **common keyword:** NSET_OF_CONSECUTIVE_VALUES *(consecutive values)*.

Keywords **application area:** assignment.

characteristic of a constraint: consecutive values, minimum.

constraint arguments: pure functional dependency.

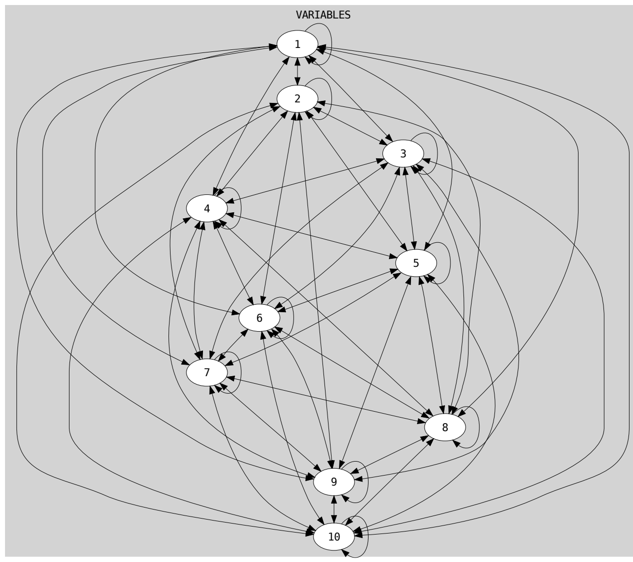
constraint type: value constraint.

modelling: functional dependency.

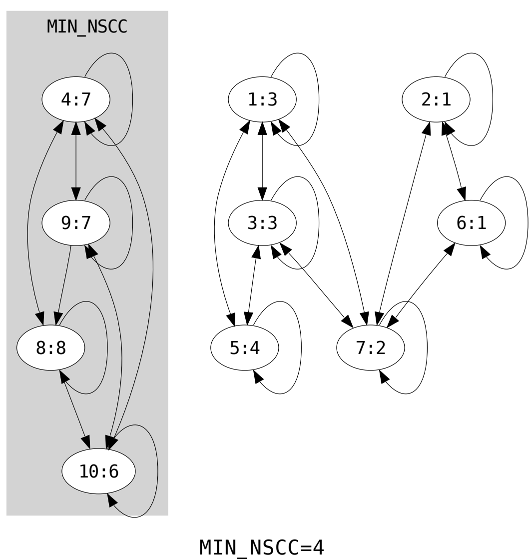
Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	abs(variables1.var − variables2.var) ≤ 1
Graph property(ies)	**MIN_NSCC** = MIN

Graph model Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.

Parts (A) and (B) of Figure 5.577 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **MIN_NSCC** graph property, we show the smallest strongly connected component of the final graph.



(A)



(B)

Figure 5.577: Initial and final graph of the MIN_SIZE_SET_OF_CONSECUTIVE_VAR constraint

5.263 MIN_SURF_PEAK

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin derived from PEAK

Constraint MIN_SURF_PEAK(MIN_SURF, VARIABLES)

Arguments

MIN_SURF : dvar
VARIABLES : collection(var−dvar)

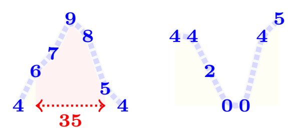
Restrictions

MIN_SURF ≥ 0
MIN_SURF $\leq$sum(VARIABLES.var)
required(VARIABLES, var)

Purpose Given a sequence VARIABLES constraint MIN_SURF to be fixed to the smallest surface of the different peaks, or to 0 if no peak exists.

Example

$(12, \langle 4, 4, 2, 2, 3, 5, 5, 6, 3, 1, 1, 2, 2, 2, 2, 2, 2, 1\rangle)$
$(35, \langle 4, 6, 7, 9, 8, 5, 4\rangle)$
$(0, \langle 4, 4, 2, 0, 0, 4, 5\rangle)$



The first MIN_SURF_PEAK constraint holds since the sequence 4 4 2 2 3 5 5 6 3 1 1 2 2 2 2 2 2 1 contains two peaks of respective surface 22 and 12 (see Figure 5.578) and since its argument MIN_SURF is fixed to the smallest value 12.

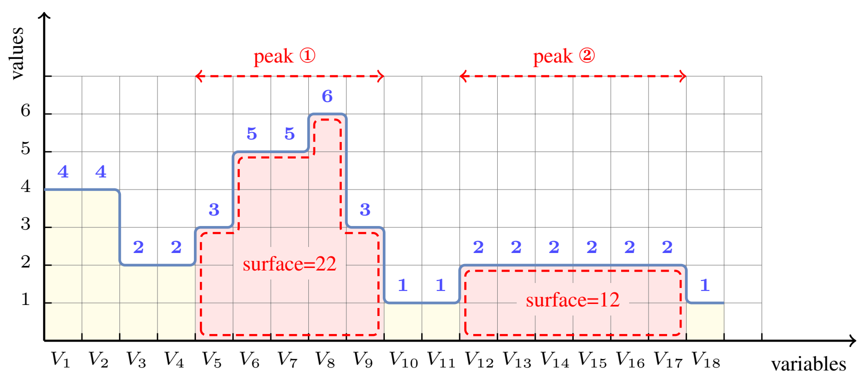


Figure 5.578: Illustration of the first example of the **Example** slot: a sequence of eighteen variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18} respectively fixed to values 4, 4, 2, 2, 3, 5, 5, 6, 3, 1, 1, 2, 2, 2, 2, 2, 2, 1 and its two peaks of surface 22 and 12.

Typical

$\mathtt{MIN\_SURF} > 1$
$|\mathtt{VARIABLES}| > 2$

Typical model

$\mathtt{nval}(\mathtt{VARIABLES.var}) > 2$

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Functional dependency: MIN_SURF determined by VARIABLES.

See also

common keyword: MIN_WIDTH_PEAK, PEAK *(sequence)*.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

modelling: functional dependency.

Automaton Figure 5.579 depicts the automaton associated with the MIN_SURF_PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

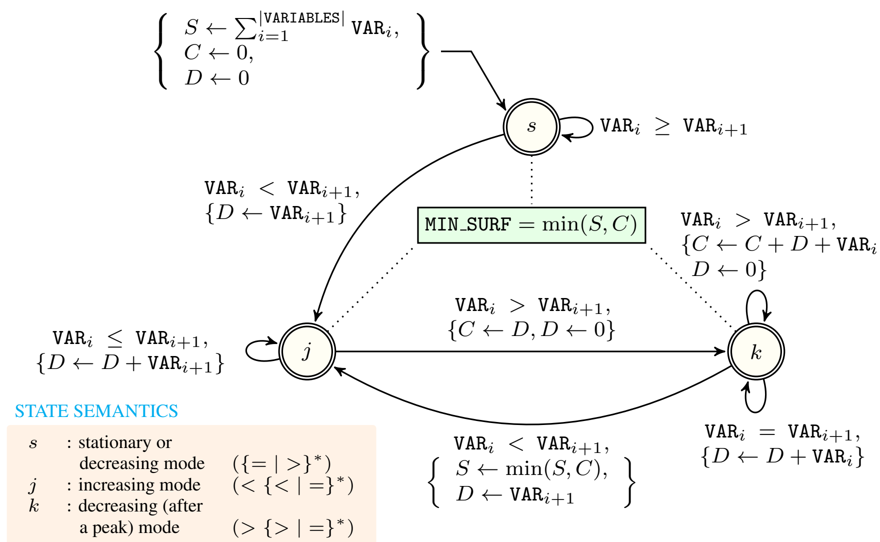


Figure 5.579: Automaton of the MIN_SURF_PEAK constraint: the start of the first potential peak is discovered while triggering the transition from s to j, the top of a peak is discovered while triggering the transition from j to k, the end of a peak and the start of the next potential peak are discovered while triggering the transition from k to j; the counters S, C and D respectively stand for *min_surface*, *current_surface* and *descending_plateau_surface*.

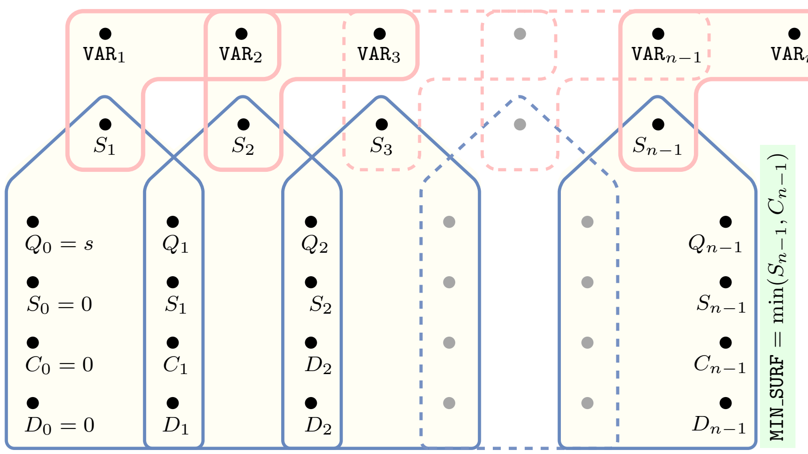


Figure 5.580: Hypergraph of the reformulation corresponding to the automaton of the MIN_SURF_PEAK constraint

5.264 MIN_WIDTH_PEAK

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin derived from PEAK

Constraint MIN_WIDTH_PEAK(MIN_WIDTH, VARIABLES)

Synonym MIN_BASE_PEAK.

Arguments

MIN_WIDTH : dvar
VARIABLES : collection(var−dvar)

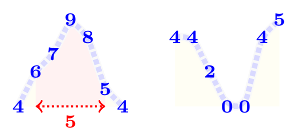
Restrictions

MIN_WIDTH ≥ 0
MIN_WIDTH $\leq$ |VARIABLES| $- 2$
required(VARIABLES, var)

Purpose Given a sequence VARIABLES constraint MIN_WIDTH to be fixed to the width of the smallest peak, or to 0 if no peak exists.

Example

$(5, \langle 4, 4, 2, 2, 3, 5, 5, 6, 3, 1, 1, 2, 2, 2, 2, 2, 2, 1\rangle)$
$(5, \langle 4, 6, 7, 9, 8, 5, 4\rangle)$
$(0, \langle 4, 4, 2, 0, 0, 4, 5\rangle)$



The first MIN_WIDTH_PEAK constraint holds since the sequence 4 4 2 2 3 5 5 6 3 1 1 2 2 2 2 2 2 1 contains two peaks of respective width 5 and 6 (see Figure 5.581) and since its argument MIN_WIDTH is fixed to the smallest value 5.

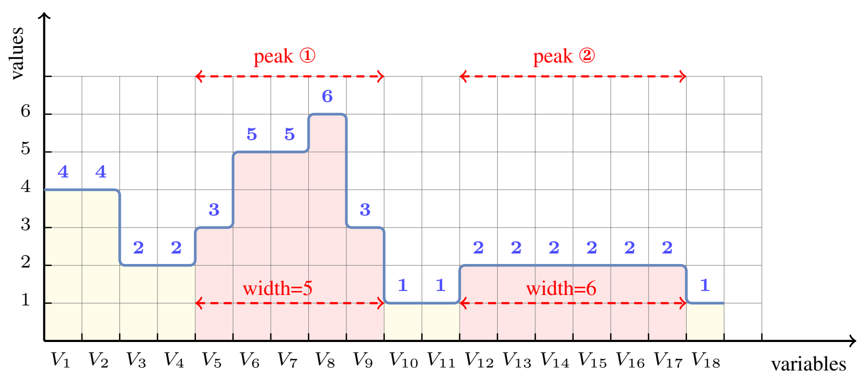


Figure 5.581: Illustration of the first example of the **Example** slot: a sequence of eighteen variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18} respectively fixed to values 4, 4, 2, 2, 3, 5, 5, 6, 3, 1, 1, 2, 2, 2, 2, 2, 2, 1 and its two peaks of width 5 and 6.

Typical

MIN_WIDTH > 1
$|$VARIABLES$| > 2$

Typical model

nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

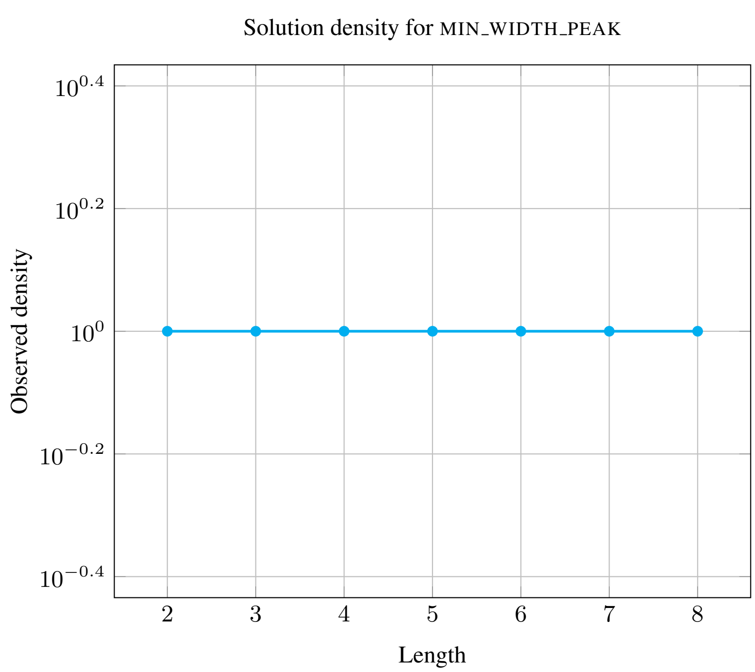
Arg. properties

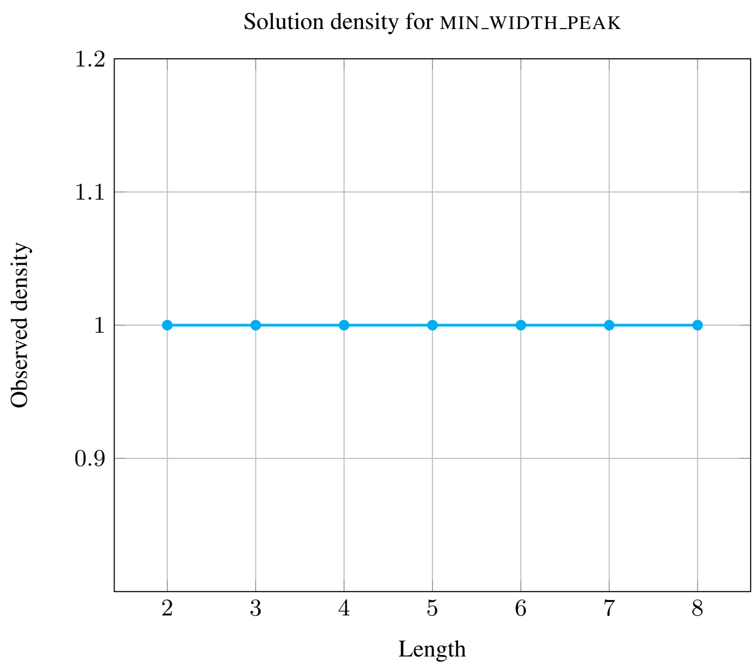
Functional dependency: MIN_WIDTH determined by VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

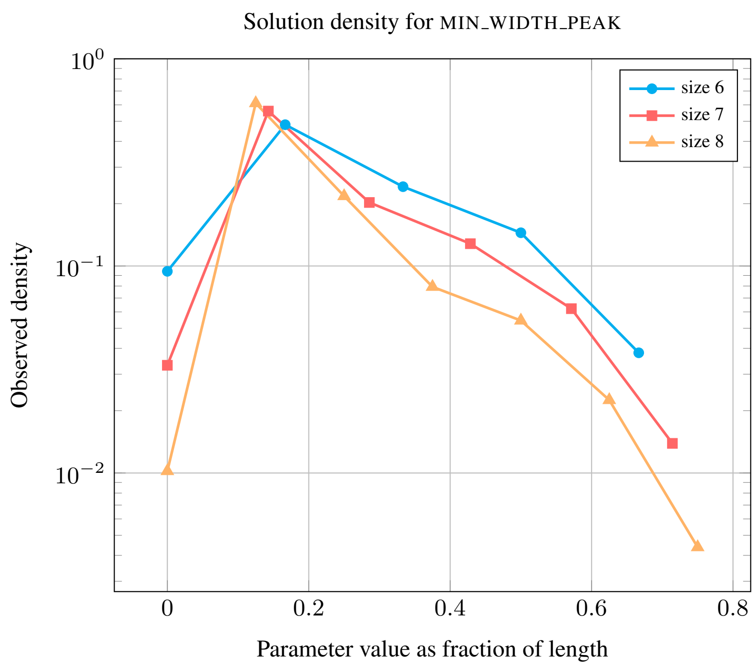
Number of solutions for MIN_WIDTH_PEAK: domains $0..n$

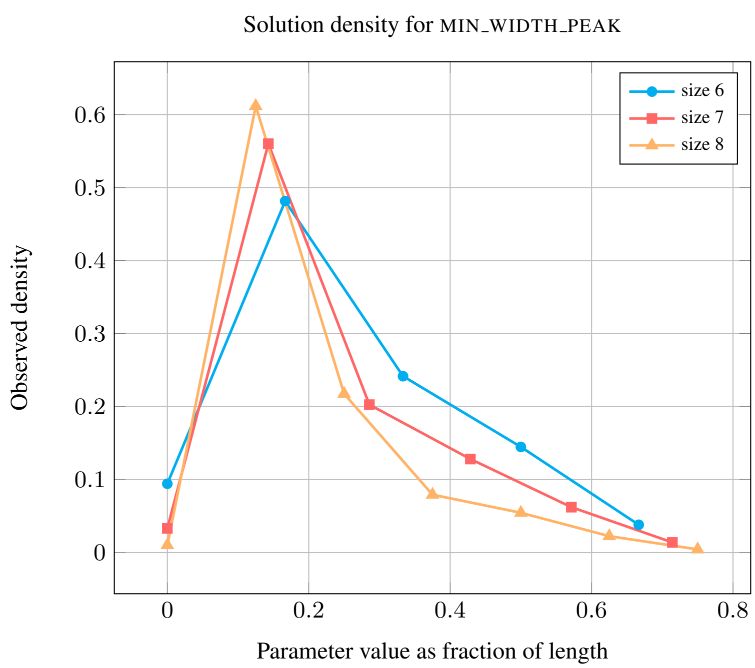




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	9	50	295	1792	11088	69498	439791
	1	-	14	230	3205	56637	1174398	26327058
	2	-	-	100	2100	28420	424928	9363060
	3	-	-	-	679	17024	268722	3413256
	4	-	-	-	-	4480	130452	2345982
	5	-	-	-	-	-	29154	968946
	6	-	-	-	-	-	-	188628

Solution count for MIN_WIDTH_PEAK: domains $0..n$





See also **common keyword:** MIN_SURF_PEAK, MIN_WIDTH_PLATEAU, PEAK *(sequence)*.

Keywords **characteristic of a constraint:** automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

filtering: glue matrix.

modelling: functional dependency.

Automaton Figure 5.582 depicts the automaton associated with the MIN_WIDTH_PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

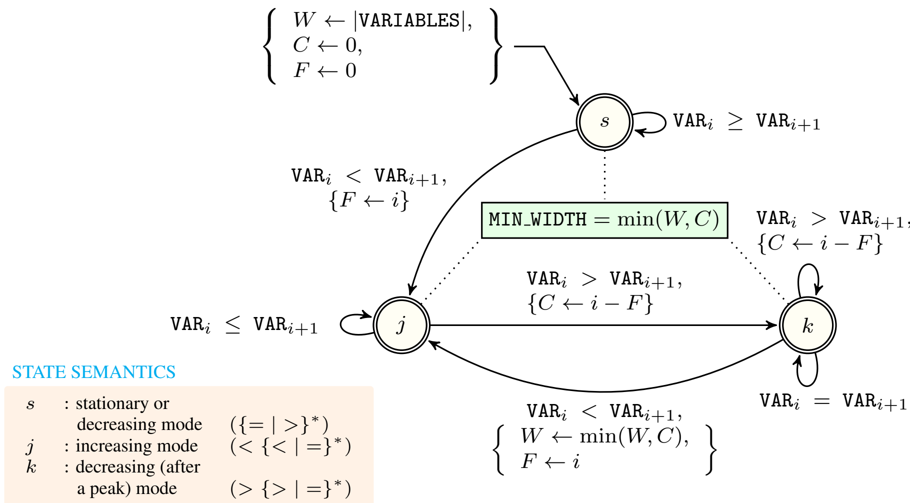


Figure 5.582: Automaton of the MIN_WIDTH_PEAK constraint: the start of the first potential peak is discovered while triggering the transition from s to j, the top of a peak is discovered while triggering the transition from j to k, the end of a peak and the start of the next potential peak are discovered while triggering the transition from k to j; the counters W, C and F respectively stand for *min_width*, *current* and *first*.

Glue matrix where $\overrightarrow{W}$, $\overrightarrow{C}$, $\overrightarrow{F}$ and $\overleftarrow{W}$, $\overleftarrow{C}$, $\overleftarrow{F}$ resp. represent the counters values W, C, F at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES; $\overrightarrow{\texttt{MIN\_WIDTH}}$ (resp. $\overleftarrow{\texttt{MIN\_WIDTH}}$) stands for $\min(\overrightarrow{W},\overrightarrow{C})$ (resp. $\min(\overleftarrow{W},\overleftarrow{C})$).

	$s\ (\{>\mid=\}^*)$	$j\ (<\{<\mid=\}^*)$	$k\ (>\{>\mid=\}^*)$
$s\ (\{>\mid=\}^*)$	0	$\overleftarrow{\texttt{MIN\_WIDTH}}$	$\overleftarrow{\texttt{MIN\_WIDTH}}$
$j\ (<\{<\mid=\}^*)$	$\overrightarrow{\texttt{MIN\_WIDTH}}$	$\min\begin{pmatrix}\overrightarrow{W},\\ n-\overrightarrow{F}-\overleftarrow{F},\\ \overleftarrow{W}\end{pmatrix}$	$\min\begin{pmatrix}\overrightarrow{\texttt{MIN\_WIDTH}},\\ n-\overrightarrow{F}-\overleftarrow{F},\\ \overleftarrow{\texttt{MIN\_WIDTH}}\end{pmatrix}$
$k\ (>\{>\mid=\}^*)$	$\overrightarrow{\texttt{MIN\_WIDTH}}$	$\min\begin{pmatrix}\overrightarrow{\texttt{MIN\_WIDTH}},\\ n-\overrightarrow{F}-\overleftarrow{F},\\ \overleftarrow{\texttt{MIN\_WIDTH}}\end{pmatrix}$	$\min\begin{pmatrix}\overrightarrow{\texttt{MIN\_WIDTH}},\\ \overleftarrow{\texttt{MIN\_WIDTH}}\end{pmatrix}$

Figure 5.583: Glue matrix associated with the automaton of the MIN_WIDTH_PEAK constraint, where n stands for |VARIABLES|

MIN_WIDTH_PEAK(MIN_WIDTH $=5$, VARIABLES $=\langle 4,6,7,9,8,5,4\rangle$)

	4	6	7	9		9	8	5	4	
		<	<	<			<	<	<	
i	0	1	2	3	:	3	2	1	0	i
$\overrightarrow{Q_i}$	s	j	j	j	:	j	j	j	s	$\overleftarrow{Q_i}$
$\overrightarrow{W_i}$	7	7	7	7	:	7	7	7	7	$\overleftarrow{W_i}$
$\overrightarrow{C_i}$	0	0	0	0	:	0	0	0	0	$\overleftarrow{C_i}$
$\overrightarrow{F_i}$	0	1	1	1	:	1	1	1	0	$\overleftarrow{F_i}$
$\overrightarrow{\texttt{MIN\_WIDTH}_i}$	0	0	0	0	:	0	0	0	0	$\overleftarrow{\texttt{MIN\_WIDTH}_i}$

MIN_WIDTH_PEAK$\begin{pmatrix}\overrightarrow{\texttt{MIN_WIDTH}_3}=0,\\ \langle 4,6,7,9\rangle\end{pmatrix}$ MIN_WIDTH_PEAK$\begin{pmatrix}\overleftarrow{\texttt{MIN_WIDTH}_3}=0,\\ \langle 4,5,8,9\rangle\end{pmatrix}$

glue matrix entry associated with the state pair (j,j):
$\texttt{MIN\_WIDTH}=\min(\overrightarrow{W_3},|\texttt{VARIABLES}|-\overrightarrow{F_3}-\overleftarrow{F_3},\overleftarrow{W_3})=\min(7,7-1-1,7)=5$

Figure 5.584: Illustrating the use of the state pair (j,j) of the glue matrix for linking MIN_WIDTH with the counters variables obtained after reading the prefix $4,6,7,9$ and corresponding suffix $9,8,5,4$ of the sequence $4,6,7,9,8,5,4$; note that the suffix $9,8,5,4$ (in pink) is proceed in reverse order; the left (resp. right) table shows the initialisation (for $i=0$) and the evolution (for $i>0$) of the state of the automaton and its counters W, C and F upon reading the prefix $4,6,7,9$ (resp. the reverse suffix $4,5,8,9$).

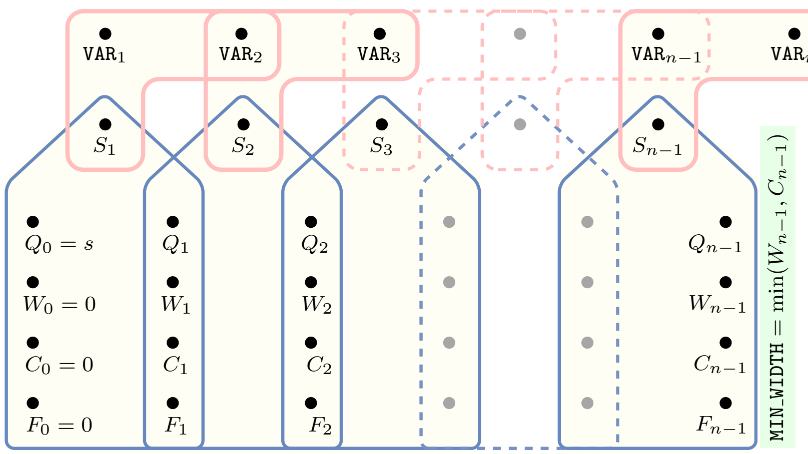


Figure 5.585: Hypergraph of the reformulation corresponding to the automaton of the MIN_WIDTH_PEAK constraint

5.265 MIN_WIDTH_PLATEAU

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin

Derived from PEAK.

Constraint

MIN_WIDTH_PLATEAU(MIN_WIDTH, VARIABLES)

Arguments

MIN_WIDTH : dvar
VARIABLES : collection(var−dvar)

Restrictions

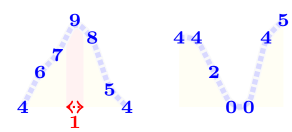
MIN_WIDTH ≥ 0
MIN_WIDTH $\leq$ |VARIABLES| $- 2$
required(VARIABLES, var)

Purpose

Given a sequence VARIABLES constraint MIN_WIDTH to be fixed to the width of the smallest plateau, or to 0 if no peak exists. A *plateau* corresponds to the highest portion of a peak.

Example

$(3, \langle 4, 4, 2, 2, 3, 5, 6, 6, 6, 1, 1, 2, 2, 2, 2, 2, 2, 1\rangle)$
$(1, \langle 4, 6, 7, 9, 8, 5, 4\rangle)$
$(0, \langle 4, 4, 2, 0, 0, 4, 5\rangle)$



The first MIN_WIDTH_PLATEAU constraint holds since the sequence 4 4 2 2 3 5 6 6 6 1 1 2 2 2 2 2 2 1 contains two peaks with two plateaux of respective width 3 and 6 (see Figure 5.586) and since its argument MIN_WIDTH is fixed to the smallest value 3.

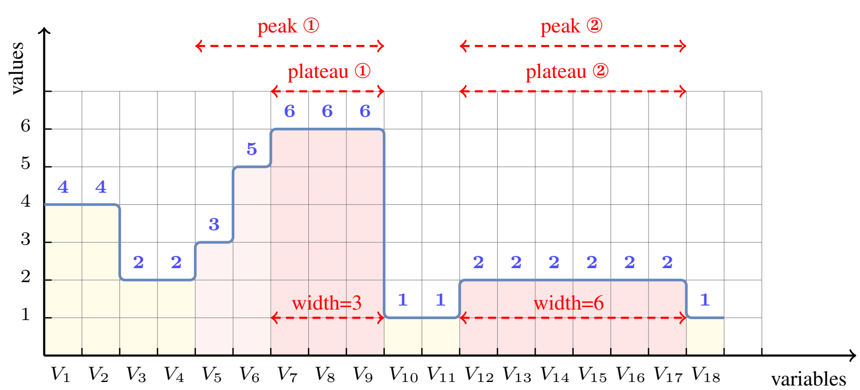


Figure 5.586: Illustration of the first example of the **Example** slot: a sequence of eighteen variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18} respectively fixed to values 4, 4, 2, 2, 3, 5, 6, 6, 6, 1, 1, 2, 2, 2, 2, 2, 2, 1 and its two peaks with two plateaux of width 3 and 6.

Typical

$|\texttt{VARIABLES}| > 2$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

Typical model

$\texttt{nval}(\texttt{VARIABLES.var}) > 2$

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

Functional dependency: MIN_WIDTH determined by VARIABLES.

See also

common keyword: MIN_WIDTH_PEAK *(sequence)*.

Keywords

characteristic of a constraint: automaton, automaton with counters.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

modelling: functional dependency.

Automaton Figure 5.587 depicts the automaton associated with the MIN_WIDTH_PLATEAU constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

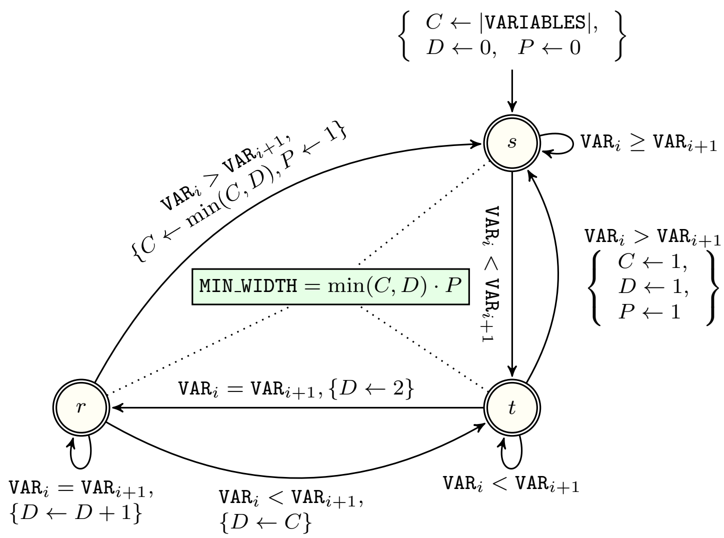


Figure 5.587: Automaton for the MIN_WIDTH_PLATEAU constraint; states s, t and r respectively correspond to a *decreasing or stationary state*, to an *increasing state* and to a *potential plateau*; counters C, D and P respectively correspond to the size of the smallest plateau encountered so far, to the size of the current potential plateau and to the fact that we already encountered at least one plateau; the transition from r to t cancels out the effect of D on the final result by resetting D to the current value of C.

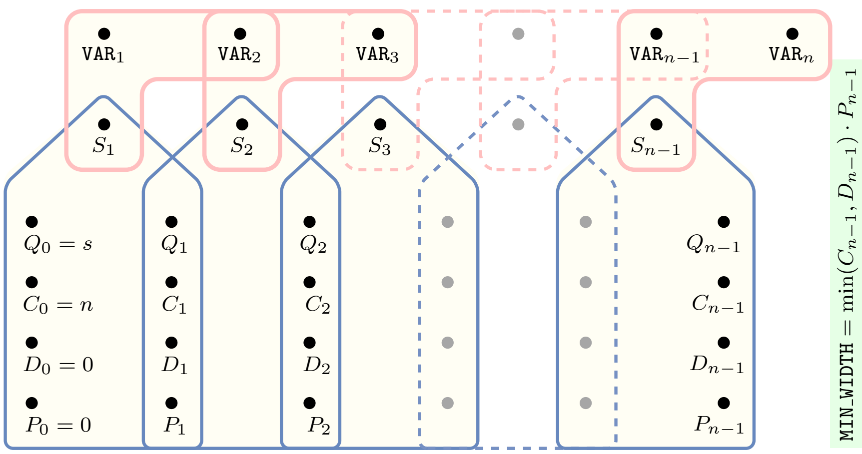


Figure 5.588: Hypergraph of the reformulation corresponding to the automaton of the MIN_WIDTH_PLATEAU constraint

5.266 MIN_WIDTH_VALLEY

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin derived from VALLEY

Constraint MIN_WIDTH_VALLEY(MIN_WIDTH, VARIABLES)

Synonym MIN_BASE_VALLEY.

Arguments

MIN_WIDTH : dvar
VARIABLES : collection(var−dvar)

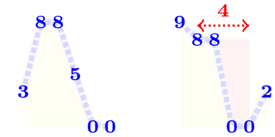
Restrictions

$\text{MIN\_WIDTH} \geq 0$
$\text{MIN\_WIDTH} \leq |\text{VARIABLES}| - 2$
required(VARIABLES, var)

Purpose Given a sequence VARIABLES constraint MIN_WIDTH to be fixed to the width of the smallest valley, or to 0 if no valley exists.

Example

$(5, \langle 3, 3, 5, 5, 4, 2, 2, 3, 4, 6, 6, 5, 5, 5, 5, 5, 5, 6\rangle)$
$(0, \langle 3, 8, 8, 5, 0, 0\rangle)$
$(4, \langle 9, 8, 8, 0, 0, 2\rangle)$



The first MIN_WIDTH_VALLEY constraint holds since the sequence 3 3 5 5 4 2 2 3 4 6 6 5 5 5 5 5 5 6 contains two valleys of respective width 5 and 6 (see Figure 5.589) and since its argument MIN_WIDTH is fixed to the smallest value 5.

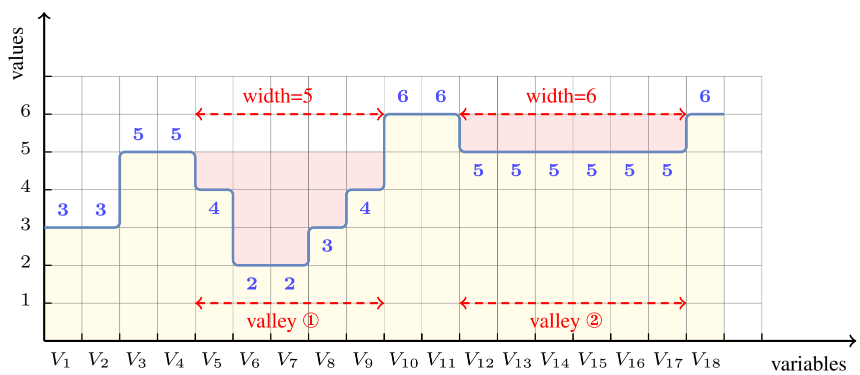


Figure 5.589: Illustration of the first example of the **Example** slot: a sequence of eighteen variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18} respectively fixed to values 3, 3, 5, 5, 4, 2, 2, 3, 4, 6, 6, 5, 5, 5, 5, 5, 5, 6 and its two valleys of width 5 and 6.

Typical

MIN_WIDTH > 1
$|$VARIABLES$| > 2$

Typical model

nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

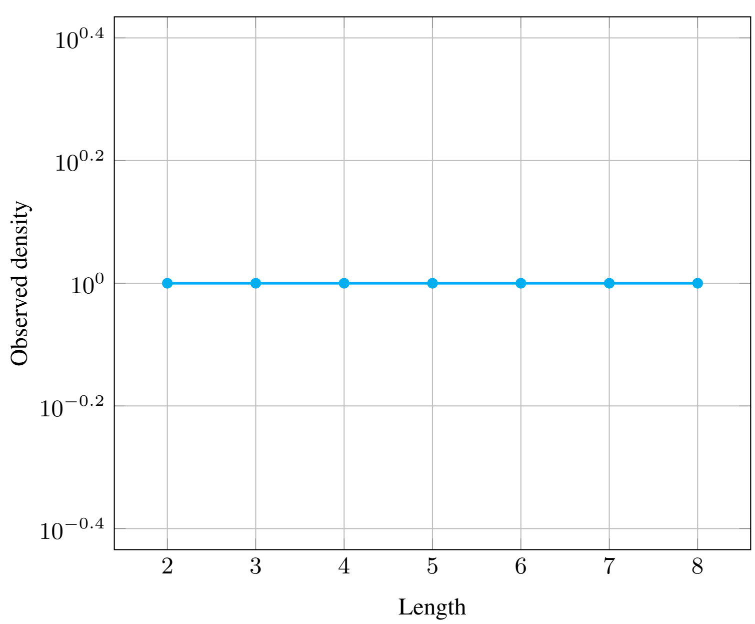
Functional dependency: MIN_WIDTH determined by VARIABLES.

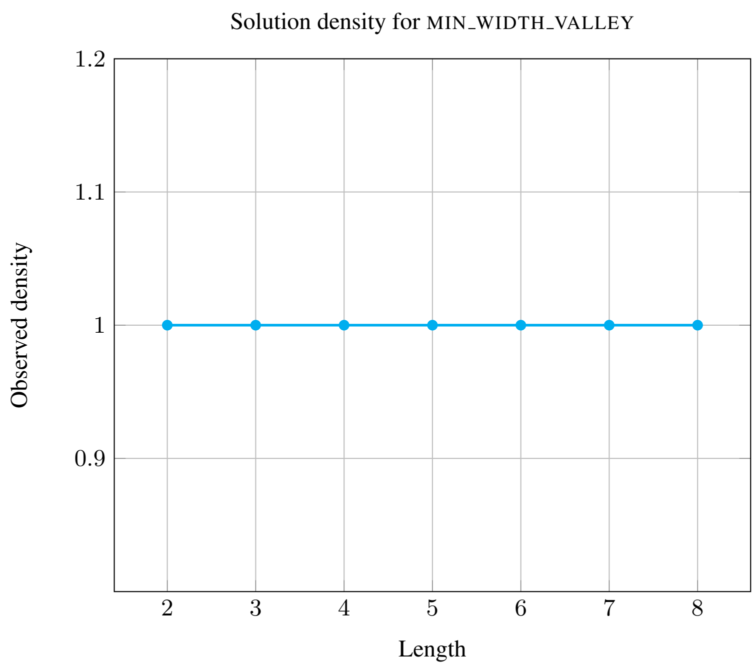
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for MIN_WIDTH_VALLEY: domains $0..n$

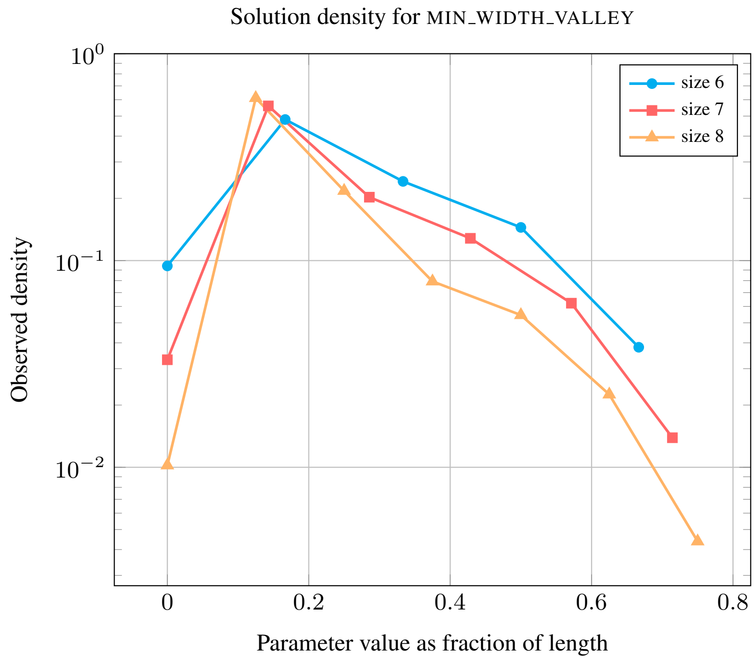
Solution density for MIN_WIDTH_VALLEY

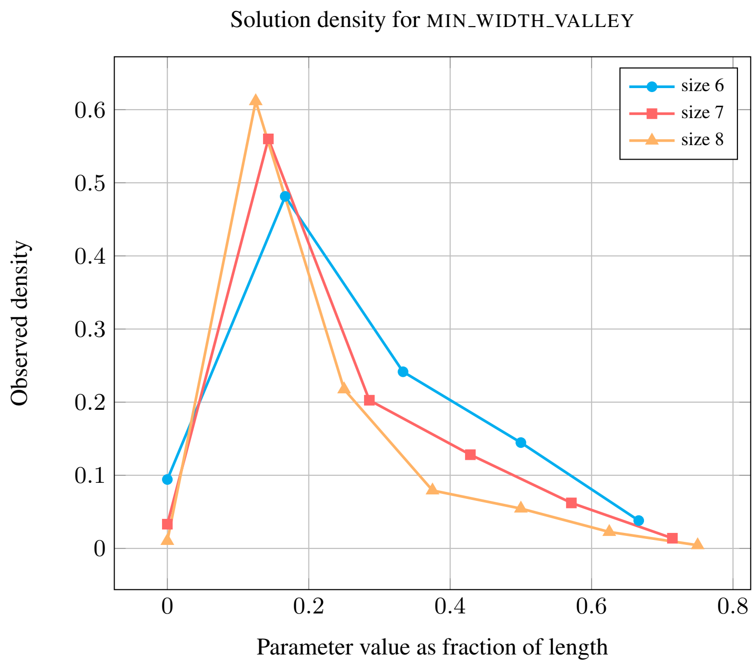




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	9	50	295	1792	11088	69498	439791
	1	-	14	230	3205	56637	1174398	26327058
	2	-	-	100	2100	28420	424928	9363060
	3	-	-	-	679	17024	268722	3413256
	4	-	-	-	-	4480	130452	2345982
	5	-	-	-	-	-	29154	968946
	6	-	-	-	-	-	-	188628

Solution count for MIN_WIDTH_VALLEY: domains $0..n$





See also **common keyword:** VALLEY (*sequence*).

Keywords **characteristic of a constraint:** automaton, automaton with counters.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

filtering: glue matrix.

modelling: functional dependency.

Automaton Figure 5.590 depicts the automaton associated with the MIN_WIDTH_VALLEY constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

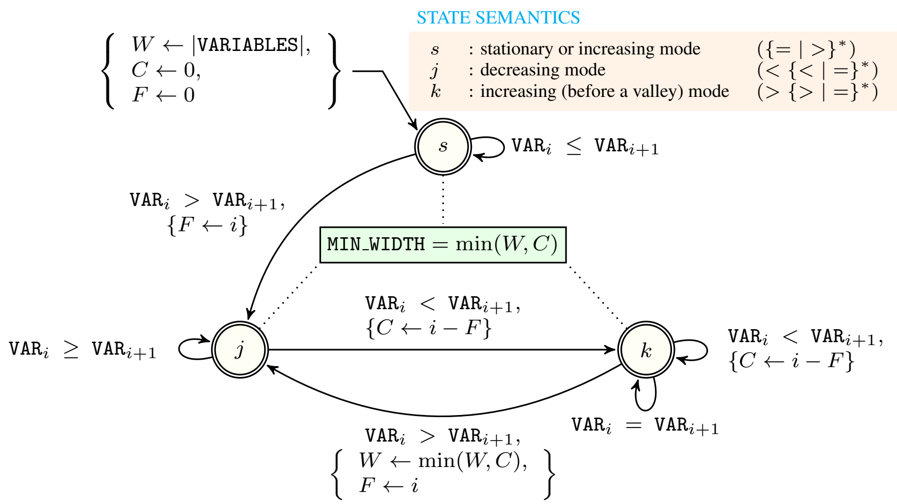


Figure 5.590: Automaton of the MIN_WIDTH_VALLEY constraint: the start of the first potential valley is discovered while triggering the transition from s to j, the bottom of a valley is discovered while triggering the transition from j to k, the end of a valley and the start of the next potential valley are discovered while triggering the transition from k to j; the counters W, C and F respectively stand for *min_width*, *current* and *first*.

Glue matrix where $\overrightarrow{W}$, $\overrightarrow{C}$, $\overrightarrow{F}$ and $\overleftarrow{W}$, $\overleftarrow{C}$, $\overleftarrow{F}$ resp. represent the counters values W, C, F at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES; $\overrightarrow{\mathtt{MIN\_WIDTH}}$ (resp. $\overleftarrow{\mathtt{MIN\_WIDTH}}$) stands for $\min(\overrightarrow{W}, \overrightarrow{C})$ (resp. $\min(\overleftarrow{W}, \overleftarrow{C})$).

	$s\ (\{<\mid=\}^*)$	$j\ (>\{>\mid=\}^*)$	$k\ (<\{<\mid=\}^*)$
$s\ (\{<\mid=\}^*)$	0	$\overleftarrow{\mathtt{MIN\_WIDTH}}$	$\overleftarrow{\mathtt{MIN\_WIDTH}}$
$j\ (>\{>\mid=\}^*)$	$\overrightarrow{\mathtt{MIN\_WIDTH}}$	$\min\begin{pmatrix}\overrightarrow{W},\\ n-\overrightarrow{F}-\overleftarrow{F},\\ \overleftarrow{W}\end{pmatrix}$	$\min\begin{pmatrix}\overrightarrow{\mathtt{MIN\_WIDTH}},\\ n-\overrightarrow{F}-\overleftarrow{F},\\ \overleftarrow{\mathtt{MIN\_WIDTH}}\end{pmatrix}$
$k\ (<\{<\mid=\}^*)$	$\overrightarrow{\mathtt{MIN\_WIDTH}}$	$\min\begin{pmatrix}\overrightarrow{\mathtt{MIN\_WIDTH}},\\ n-\overrightarrow{F}-\overleftarrow{F},\\ \overleftarrow{\mathtt{MIN\_WIDTH}}\end{pmatrix}$	$\min\begin{pmatrix}\overrightarrow{\mathtt{MIN\_WIDTH}},\\ \overleftarrow{\mathtt{MIN\_WIDTH}}\end{pmatrix}$

Figure 5.591: Glue matrix associated with the automaton of the MIN_WIDTH_VALLEY constraint, where n stands for $|\mathtt{VARIABLES}|$

MIN_WIDTH_VALLEY(MIN_WIDTH $= 5$, VARIABLES $= \langle 6, 4, 3, 1, 2, 5, 6\rangle$)

	6	4	3	1		1	2	5	6	
	$>$	$>$	$>$			$>$	$>$	$>$		
i	0	1	2	3	:	3	2	1	0	i
$\overrightarrow{Q_i}$	s	j	j	j	:	j	j	j	s	$\overleftarrow{Q_i}$
$\overrightarrow{W_i}$	7	7	7	7	:	7	7	7	7	$\overleftarrow{W_i}$
$\overrightarrow{C_i}$	0	0	0	0	:	0	0	0	0	$\overleftarrow{C_i}$
$\overrightarrow{F_i}$	0	1	1	1	:	1	1	1	0	$\overleftarrow{F_i}$
$\overrightarrow{\texttt{MIN\_WIDTH}_i}$	0	0	0	0	:	0	0	0	0	$\overleftarrow{\texttt{MIN\_WIDTH}_i}$

MIN_WIDTH_VALLEY$\left(\overrightarrow{\texttt{MIN_WIDTH}_3} = 0, \langle 6, 4, 3, 1\rangle\right)$ MIN_WIDTH_VALLEY$\left(\overleftarrow{\texttt{MIN_WIDTH}_3} = 0, \langle 6, 5, 2, 1\rangle\right)$

glue matrix entry associated with the state pair (j, j):

$\texttt{MIN\_WIDTH} = \min(\overrightarrow{W_3}, |\texttt{VARIABLES}| - \overrightarrow{F_3} - \overleftarrow{F_3}, \overleftarrow{W_3}) = \min(7, 7 - 1 - 1, 7) = 5$

Figure 5.592: Illustrating the use of the state pair (j, j) of the glue matrix for linking MIN_WIDTH with the counters variables obtained after reading the prefix $6, 4, 3, 1$ and corresponding suffix $1, 2, 5, 6$ of the sequence $6, 4, 3, 1, 2, 5, 6$; note that the suffix $1, 2, 5, 6$ (in pink) is proceed in reverse order; the left (resp. right) table shows the initialisation (for $i = 0$) and the evolution (for $i > 0$) of the state of the automaton and its counters W, C and F upon reading the prefix $6, 4, 3, 1$ (resp. the reverse suffix $6, 5, 2, 1$).

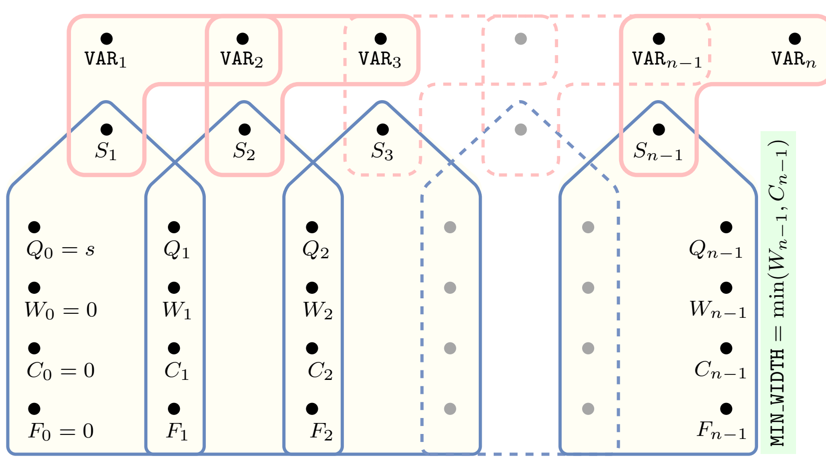


Figure 5.593: Hypergraph of the reformulation corresponding to the automaton of the MIN_WIDTH_VALLEY constraint

5.267 MINIMUM

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint MINIMUM(MIN, VARIABLES)

Synonym MIN.

Arguments

MIN	:	dvar
VARIABLES	:	collection(var−dvar)

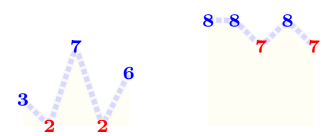
Restrictions

$|$VARIABLES$| > 0$
required(VARIABLES, var)

Purpose MIN is the minimum value of the collection of domain variables VARIABLES.

Example

$(2, \langle 3, 2, 7, 2, 6\rangle)$
$(7, \langle 8, 8, 7, 8, 7\rangle)$



The first MINIMUM constraint holds since its first argument MIN $= 2$ is set to the minimum value of the collection $\langle 3, 2, 7, 2, 6\rangle$.

Typical

$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to MIN as well as to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: MIN determined by VARIABLES.
- Aggregate: MIN(min), VARIABLES(union).

Usage In some project scheduling problems one has to introduce dummy activities that correspond, for example, to the starting time of a given set of activities. In this context one can use the MINIMUM constraint to get the minimum starting time of a set of tasks.

Remark Note that MINIMUM is a constraint and not just a function that computes the minimum value of a collection of variables: potential values of MIN influence the variables of VARIABLES, and reciprocally potential values that can be assigned to variables of VARIABLES influence MIN.

The MINIMUM constraint is called MIN in **JaCoP** (http://www.jacop.eu/).

Algorithm A filtering algorithm for the MINIMUM constraint is described in [29].

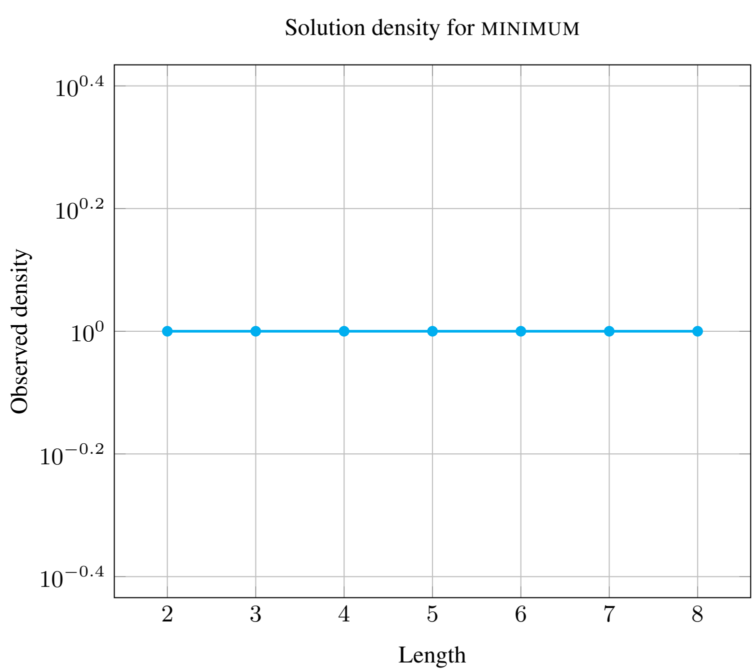
The MINIMUM constraint is entailed if all the following conditions hold:

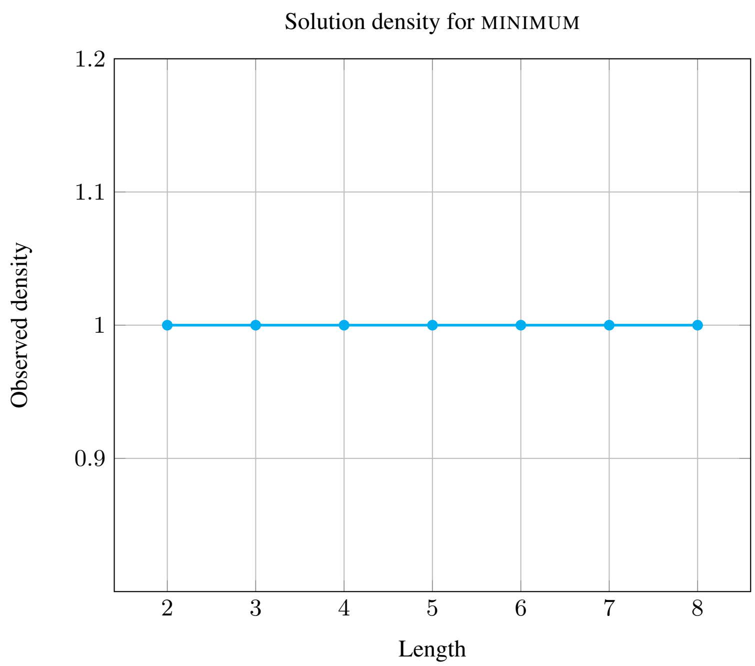
1. `MIN` is fixed.
2. At least one variable of `VARIABLES` is assigned value `MIN`.
3. All variables of `VARIABLES` have their minimum values greater than or equal to value `MIN`.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

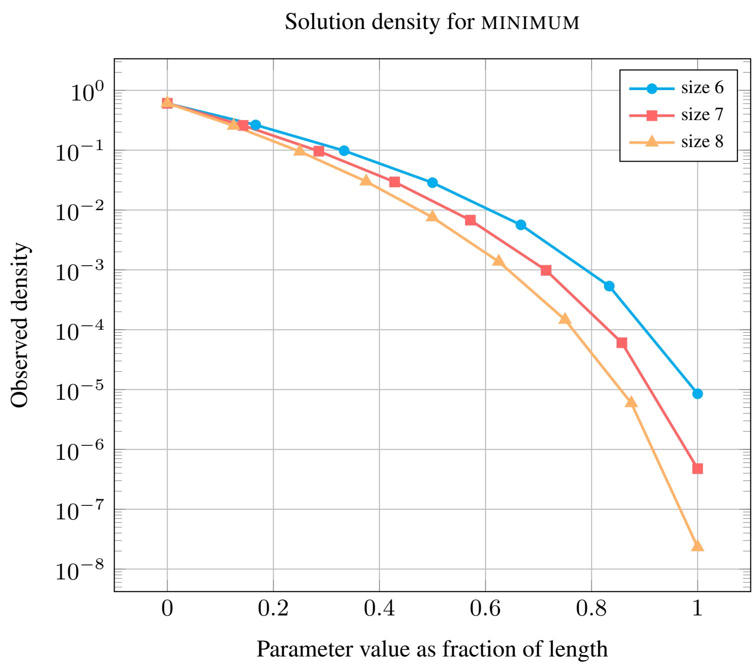
Number of solutions for MINIMUM: domains $0..n$

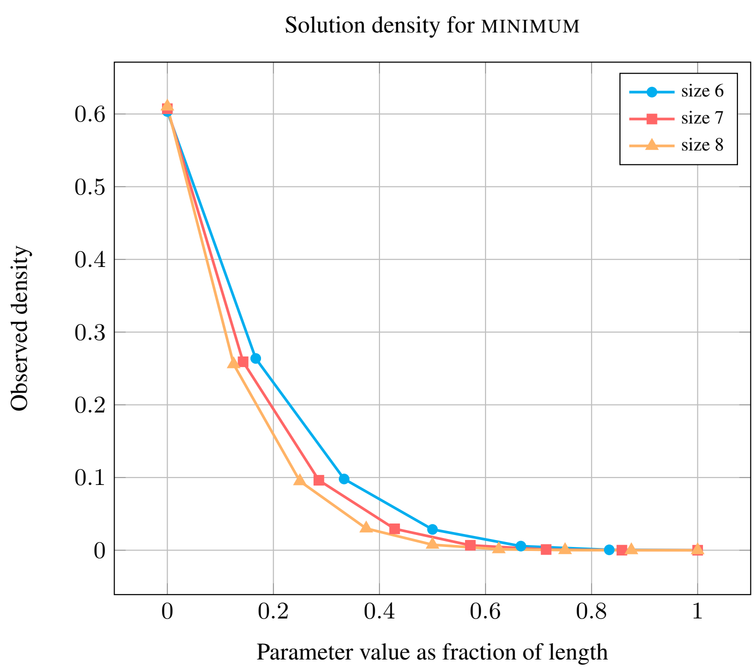




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	5	37	369	4651	70993	1273609	26269505
	1	3	19	175	2101	31031	543607	11012415
	2	1	7	65	781	11529	201811	4085185
	3	-	1	15	211	3367	61741	1288991
	4	-	-	1	31	665	14197	325089
	5	-	-	-	1	63	2059	58975
	6	-	-	-	-	1	127	6305
	7	-	-	-	-	-	1	255
	8	-	-	-	-	-	-	1

Solution count for MINIMUM: domains $0..n$





Systems MIN in **Choco**, MIN in **Gecode**, MIN in **JaCoP**, MINIMUM in **MiniZinc**, MINIMUM in **SICStus**.

Used in MINIMUM_GREATER_THAN, NEXT_ELEMENT, NEXT_GREATER_ELEMENT.

See also

common keyword: MAXIMUM *(order constraint)*.

comparison swapped: MAXIMUM.

generalisation: MINIMUM_MODULO *(*`variable` *replaced by* `variable` mod `constant`*)*.

implied by: AND.

implies: BETWEEN_MIN_MAX, IN.

soft variant: MINIMUM_EXCEPT_0 *(value 0 is ignored)*, OPEN_MINIMUM *(open constraint)*.

specialisation: MIN_N *(minimum or order* `n` *replaced by absolute minimum)*.

uses in its reformulation: CYCLE.

Keywords

characteristic of a constraint: minimum, maxint, automaton, automaton without counters, reified automaton constraint.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: centered cyclic(1) constraint network(1).

constraint type: order constraint.

filtering: glue matrix, arc-consistency, entailment.

modelling: functional dependency.

Cond. implications

MINIMUM(MIN, VARIABLES)
with $\texttt{first}(\texttt{VARIABLES.var}) > \texttt{MIN}$
and $\texttt{last}(\texttt{VARIABLES.var}) > \texttt{MIN}$
implies DEEPEST_VALLEY(DEPTH, VARIABLES).

Arc input(s)	VARIABLES
Arc generator	*CLIQUE* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	$\bigvee\begin{pmatrix} \texttt{variables1.key} = \texttt{variables2.key}, \\ \texttt{variables1.var} < \texttt{variables2.var} \end{pmatrix}$
Graph property(ies)	**ORDER**(0, MAXINT, var) = MIN

Graph model The condition variables1.key = variables2.key holds if and only if variables1 and variables2 corresponds to the same vertex. It is used in order to enforce to keep all the vertices of the initial graph. **ORDER**(0, MAXINT, var) refers to the source vertices of the graph, i.e., those vertices that do not have any predecessor.

Parts (A) and (B) of Figure 5.594 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **ORDER** graph property, the vertices of rank 0 (without considering the loops) of the final graph are outlined with a thick circle.

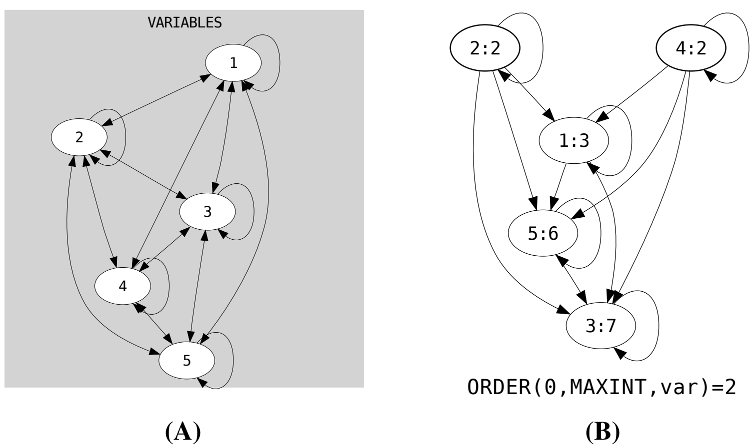


Figure 5.594: Initial and final graph of the MINIMUM constraint

Automaton Figure 5.595 depicts a first counter free deterministic automaton associated with the MINIMUM constraint. Let $\texttt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each pair $(\texttt{MIN}, \texttt{VAR}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\texttt{MIN} < \texttt{VAR}_i \Leftrightarrow S_i = 0) \wedge (\texttt{MIN} = \texttt{VAR}_i \Leftrightarrow S_i = 1) \wedge (\texttt{MIN} > \texttt{VAR}_i \Leftrightarrow S_i = 2)$.

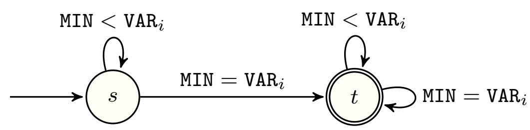


Figure 5.595: Counter free automaton of the MINIMUM constraint

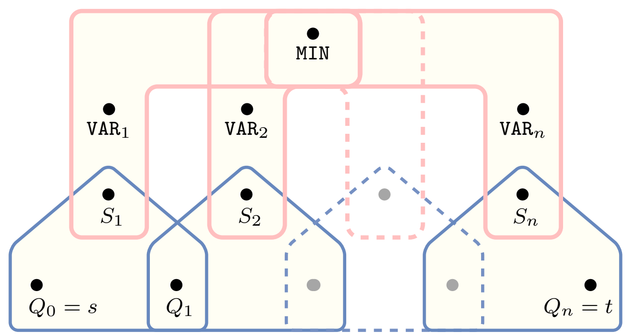


Figure 5.596: Hypergraph of the reformulation corresponding to the counter free automaton of the MINIMUM constraint

Figure 5.597 depicts a second counter free non deterministic automaton associated with the MINIMUM constraint, where the argument MIN is also part of the sequence passed to the automaton.

Figure 5.599 depicts a third deterministic automaton with one counter associated with the MINIMUM constraint, where the argument MIN is unified to the final value of the counter.

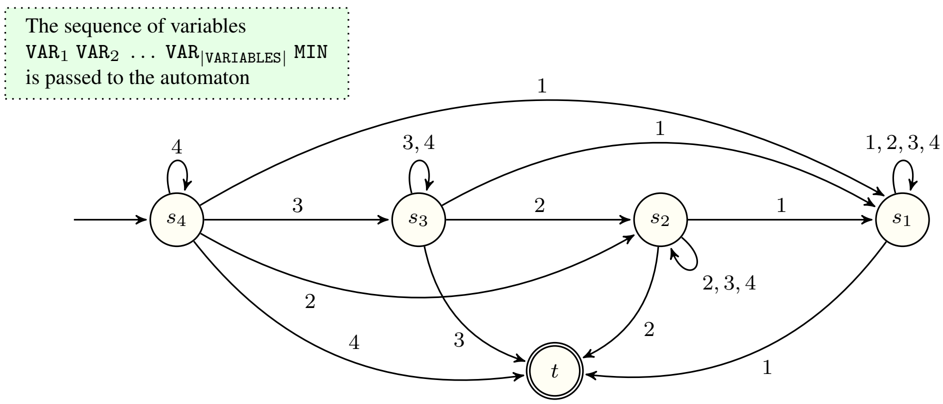


Figure 5.597: Counter free non deterministic automaton of the MINIMUM(MIN, VARIABLES) constraint assuming that the union of the domain of the variables is the set $\{1, 2, 3, 4\}$ and that the elements of VARIABLES are first passed to the automaton followed by MIN (state s_i means that no value strictly less than value i was found and that value i was already encountered at least once)

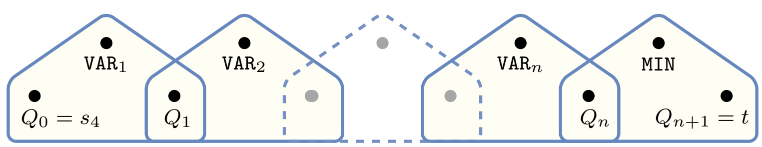


Figure 5.598: Hypergraph of the reformulation corresponding to the counter free non deterministic automaton of the MINIMUM constraint

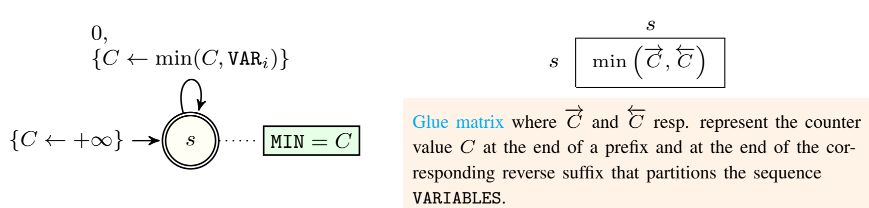


Figure 5.599: Automaton (with one counter) of the MINIMUM constraint and its glue matrix

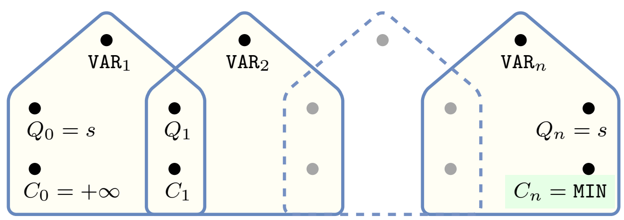


Figure 5.600: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the MINIMUM constraint: since all states variables $Q_0, Q_1, \ldots, Q_n$ are fixed to the unique state s of the automaton, the transitions constraints share only the counter variable C and the constraint network is Berge-acyclic

5.268 MINIMUM_EXCEPT_0

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin

Derived from MINIMUM.

Constraint

MINIMUM_EXCEPT_0(MIN, VARIABLES, DEFAULT)

Arguments

MIN	:	dvar
VARIABLES	:	collection(var−dvar)
DEFAULT	:	int

Restrictions

MIN > 0
MIN $\leq$ DEFAULT
$|$VARIABLES$| > 0$
required(VARIABLES, var)
VARIABLES.var ≥ 0
VARIABLES.var $\leq$ DEFAULT
DEFAULT > 0

Purpose

All variables of the collection VARIABLES are assigned a value that belongs to interval $[0, \mathtt{DEFAULT}]$. MIN is the minimum value of the collection of domain variables VARIABLES, ignoring all variables that take 0 as value. When all variables of the collection VARIABLES are assigned value 0, MIN is set to the default value DEFAULT.

Example

$(3, \langle 3, 7, 6, 7, 4, 7 \rangle, 1000000)$
$(2, \langle 3, 2, 0, 7, 2, 6 \rangle, 1000000)$
$(1000000, \langle 0, 0, 0, 0, 0, 0 \rangle, 1000000)$

The three examples of the MINIMUM_EXCEPT_0 constraint respectively hold since:

- Within the first example, MIN is set to the minimum value 3 of the collection $\langle 3, 7, 6, 7, 4, 7 \rangle$.
- Within the second example, MIN is set to the minimum value 2 (ignoring value 0) of the collection $\langle 3, 2, 0, 7, 2, 6 \rangle$.
- Finally within the third example, MIN is set to the default value 1000000 since all items of the collection $\langle 0, 0, 0, 0, 0, 0 \rangle$ are set to 0.

Typical

$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1
ATLEAST(1, VARIABLES, 0)

Typical model

nval(VARIABLES.var) > 2
ATLEAST(2, VARIABLES, 0)

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.

Arg. properties

Functional dependency: MIN determined by VARIABLES and DEFAULT.

Remark

The joker value 0 makes sense only because we restrict the variables of the VARIABLES collection to take non-negative values.

Reformulation

By (1) associating to each variable V_i $(i \in [1, |\mathtt{VARIABLES}|])$ of the VARIABLES collection a *rank* variable $R_i \in [0, |\mathtt{VARIABLES}|-1]$ with the reified constraint $R_i = 1 \Leftrightarrow V_i = \mathtt{MIN}$, and by creating for each pair of variables V_i, V_j $(i, j < i \in [1, |\mathtt{VARIABLES}|])$ the reified constraints

$V_i < V_j \Leftrightarrow R_i < R_j$,
$V_i = V_j \Leftrightarrow R_i = R_j$,
$V_i > V_j \Leftrightarrow R_i > R_j$,

and by (2) creating the reified constraint

$V_1 = 0 \wedge V_2 = 0 \wedge \cdots \wedge V_n = 0 \Rightarrow \mathtt{MIN} = \mathtt{DEFAULT}$,

one can reformulate the MINIMUM_EXCEPT_0 constraint in term of $3 \cdot \frac{|\mathtt{VARIABLES}| \cdot (|\mathtt{VARIABLES}|-1)}{2} + 2$ reified constraints.

See also

hard version: MINIMUM *(value* 0 *is not ignored any more)*.

Keywords

characteristic of a constraint: joker value, minimum, automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: centered cyclic(1) constraint network(1).

constraint type: order constraint.

modelling: functional dependency.

Cond. implications

MINIMUM_EXCEPT_0(MIN, VARIABLES, DEFAULT)
with maxval(VARIABLES.var) $<$ DEFAULT
implies ATMOST(N, VARIABLES, VALUE).

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• variables1.var $\neq 0$ • variables2.var $\neq 0$ • $\bigvee \begin{pmatrix} \texttt{variables1.key} = \texttt{variables2.key}, \\ \texttt{variables1.var} < \texttt{variables2.var} \end{pmatrix}$
Graph property(ies)	**ORDER**(0, DEFAULT, var) = MIN

Graph model Because of the first two conditions of the arc constraint, all vertices that correspond to 0 will be removed from the final graph.

Parts (A) and (B) of Figure 5.601 respectively show the initial and final graph of the second example of the **Example** slot. Since we use the **ORDER** graph property, the vertices of rank 0 (without considering the loops) of the final graph are outlined with a thick circle.

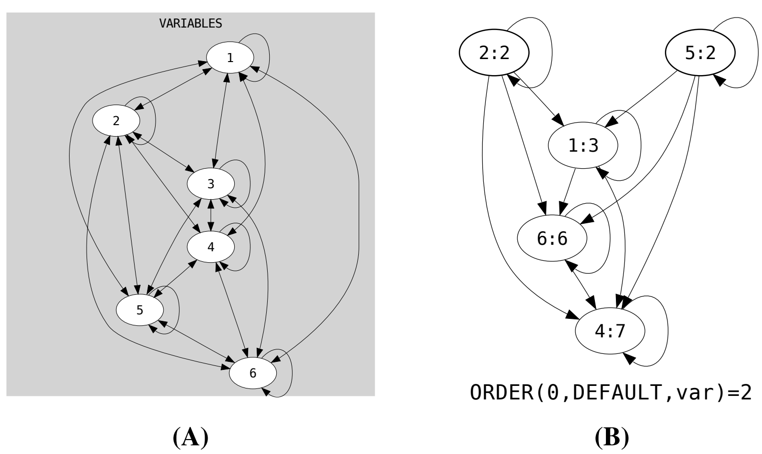


Figure 5.601: Initial and final graph of the MINIMUM_EXCEPT_0 constraint

Since the graph associated with the third example does not contain any vertex, **ORDER** returns the default value DEFAULT.

Automaton Figure 5.602 depicts the automaton associated with the MINIMUM_EXCEPT_0 constraint. Let $\texttt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each pair $(\texttt{MIN}, \texttt{VAR}_i)$ corresponds a signature variable S_i as well as the following signature constraint:

$((\texttt{VAR}_i = 0) \wedge (\texttt{MIN} \neq \texttt{DEFAULT})) \Leftrightarrow S_i = 0 \wedge$

$((\texttt{VAR}_i = 0) \wedge (\texttt{MIN} = \texttt{DEFAULT})) \Leftrightarrow S_i = 1 \wedge$

$((\texttt{VAR}_i \neq 0) \wedge (\texttt{MIN} = \texttt{VAR}_i)) \Leftrightarrow S_i = 2 \wedge$

$((\texttt{VAR}_i \neq 0) \wedge (\texttt{MIN} < \texttt{VAR}_i)) \Leftrightarrow S_i = 3 \wedge$

$((\texttt{VAR}_i \neq 0) \wedge (\texttt{MIN} > \texttt{VAR}_i)) \Leftrightarrow S_i = 4.$

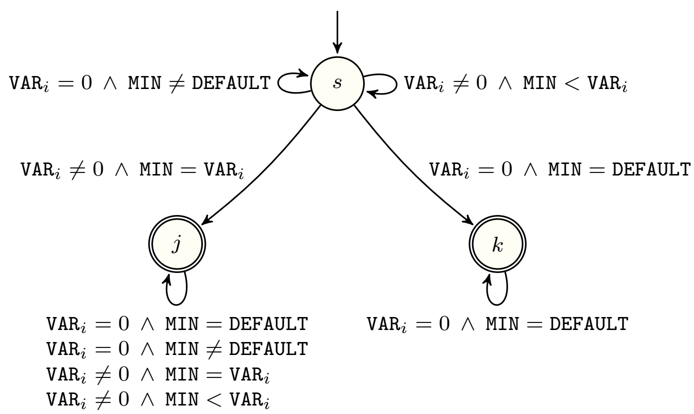


Figure 5.602: Automaton of the MINIMUM_EXCEPT_0 constraint

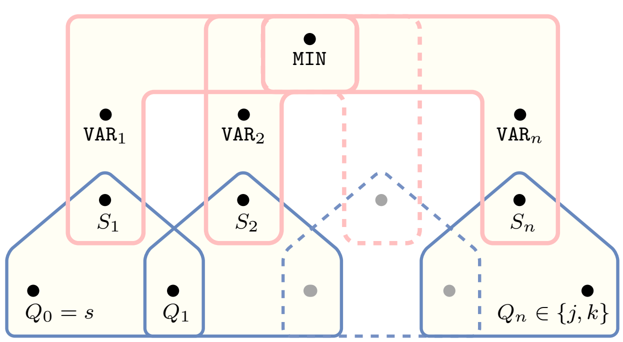


Figure 5.603: Hypergraph of the reformulation corresponding to the automaton of the MINIMUM_EXCEPT_0 constraint

5.269 MINIMUM_GREATER_THAN

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint MINIMUM_GREATER_THAN(VAR1, VAR2, VARIABLES)

Arguments

VAR1 : dvar
VAR2 : dvar
VARIABLES : collection(var−dvar)

Restrictions

VAR1 $>$ VAR2
$|$VARIABLES$| > 0$
required(VARIABLES, var)

Purpose VAR1 is the smallest value strictly greater than VAR2 of the collection of variables VARIABLES: this concretely means that there exists at least one variable of VARIABLES that takes a value strictly greater than VAR2.

Example $(5, 3, \langle 8, 5, 3, 8\rangle)$

The MINIMUM_GREATER_THAN constraint holds since value 5 is the smallest value strictly greater than value 3 among values $8, 5, 3$ and 8.

Typical

$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

Symmetry Items of VARIABLES are permutable.

Arg. properties Aggregate: VAR1(min), VAR2(id), VARIABLES(union).

Reformulation Let $V_1, V_2, \ldots, V_{|\texttt{VARIABLES}|}$ denote the variables of the collection of variables VARIABLES. By creating the extra variables M and $U_1, U_2, \ldots, U_{|\texttt{VARIABLES}|}$, the MINIMUM_GREATER_THAN constraint can be expressed in term of the following constraints:

1. MAXIMUM(M, VARIABLES),
2. VAR1 $>$ VAR2,
3. VAR1 $\leq M$,
4. $V_i \leq$ VAR2 $\Rightarrow U_i = M \quad (i \in [1, |\texttt{VARIABLES}|])$,
5. $V_i >$ VAR2 $\Rightarrow U_i = V_i \quad (i \in [1, |\texttt{VARIABLES}|])$,

6. MINIMUM(VAR1, $\langle U_1, U_2, \ldots, U_{|\text{VARIABLES}|}\rangle$).

See also

common keyword: NEXT_GREATER_ELEMENT *(order constraint)*.

implied by: NEXT_GREATER_ELEMENT.

related: NEXT_ELEMENT *(identify an element in a table)*.

Keywords

characteristic of a constraint: minimum, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: order constraint.

Derived Collection	col(ITEM−collection(var−dvar), [item(var − VAR2)])
Arc input(s)	ITEM VARIABLES
Arc generator	*PRODUCT*↦collection(item, variables)
Arc arity	2
Arc constraint(s)	item.var < variables.var
Graph property(ies)	**NARC**> 0
Sets	SUCC ↦ [source, variables]
Constraint(s) on sets	MINIMUM(VAR1, variables)

Graph model

Similar to the NEXT_GREATER_ELEMENT constraint, except that there is no order on the variables of the collection VARIABLES.

Parts (A) and (B) of Figure 5.604 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. The source and the sinks of the final graph respectively correspond to the variable VAR2 and to the variables of the VARIABLES collection that are strictly greater than VAR2. VAR1 is set to the smallest value of the var attribute of the sinks of the final graph.

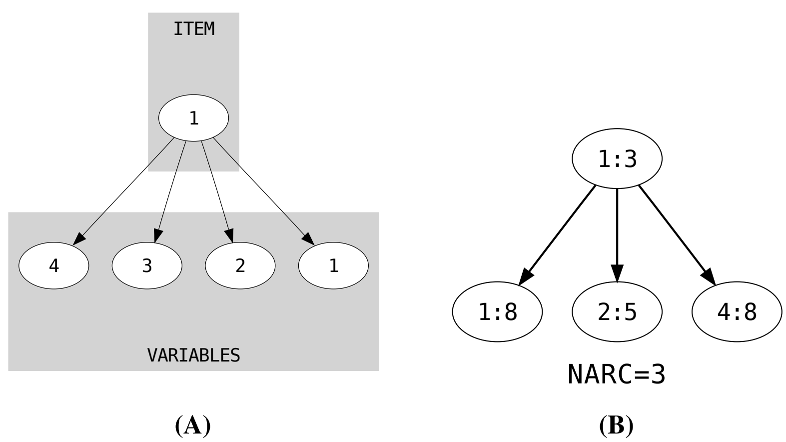


Figure 5.604: Initial and final graph of the MINIMUM_GREATER_THAN constraint

Automaton Figure 5.605 depicts the automaton associated with the MINIMUM_GREATER_THAN constraint. Let $\mathtt{VAR}_i$ be the i^{th} variable of the VARIABLES collection. To each triple $(\mathtt{VAR1}, \mathtt{VAR2}, \mathtt{VAR}_i)$ corresponds a signature variable S_i as well as the following signature constraint:

$((\mathtt{VAR}_i < \mathtt{VAR1}) \wedge (\mathtt{VAR}_i \leq \mathtt{VAR2})) \Leftrightarrow S_i = 0 \wedge$

$((\mathtt{VAR}_i = \mathtt{VAR1}) \wedge (\mathtt{VAR}_i \leq \mathtt{VAR2})) \Leftrightarrow S_i = 1 \wedge$

$((\mathtt{VAR}_i > \mathtt{VAR1}) \wedge (\mathtt{VAR}_i \leq \mathtt{VAR2})) \Leftrightarrow S_i = 2 \wedge$

$((\mathtt{VAR}_i < \mathtt{VAR1}) \wedge (\mathtt{VAR}_i > \mathtt{VAR2})) \Leftrightarrow S_i = 3 \wedge$

$((\mathtt{VAR}_i = \mathtt{VAR1}) \wedge (\mathtt{VAR}_i > \mathtt{VAR2})) \Leftrightarrow S_i = 4 \wedge$

$((\mathtt{VAR}_i > \mathtt{VAR1}) \wedge (\mathtt{VAR}_i > \mathtt{VAR2})) \Leftrightarrow S_i = 5.$

The automaton is constructed in order to fulfil the following conditions:

- We look for an item of the VARIABLES collection such that $\mathtt{var}_i = \mathtt{VAR1}$ and $\mathtt{var}_i > \mathtt{VAR2}$,
- There should not exist any item of the VARIABLES collection such that $\mathtt{var}_i < \mathtt{VAR1}$ and $\mathtt{var}_i > \mathtt{VAR2}$.

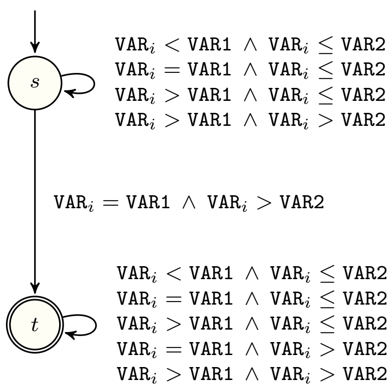


Figure 5.605: Automaton of the MINIMUM_GREATER_THAN constraint

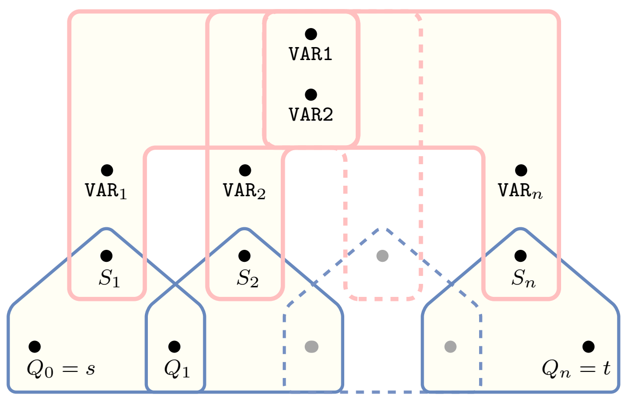


Figure 5.606: Hypergraph of the reformulation corresponding to the counter free non deterministic automaton of the MINIMUM_GREATER_THAN constraint

5.270 MINIMUM_MODULO

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from MINIMUM.

Constraint MINIMUM_MODULO(MIN, VARIABLES, M)

Arguments

```
MIN       : dvar
VARIABLES : collection(var-dvar)
M         : int
```

Restrictions

$|\mathtt{VARIABLES}| > 0$
$\mathtt{M} > 0$
required(VARIABLES, var)

Purpose MIN is a minimum value of the collection of domain variables VARIABLES according to the following partial ordering: $(X \bmod \mathtt{M}) < (Y \bmod \mathtt{M})$.

Example

$(6, \langle 9, 1, 7, 6, 5\rangle, 3)$
$(9, \langle 9, 1, 7, 6, 5\rangle, 3)$

The MINIMUM_MODULO constraints hold since MIN is respectively set to values 6 and 9, where $6 \bmod 3 = 0$ and $9 \bmod 3 = 0$ are both less than or equal to all the expressions $9 \bmod 3 = 0$, $1 \bmod 3 = 1$, $7 \bmod 3 = 1$, $6 \bmod 3 = 0$, and $5 \bmod 3 = 2$.

Typical

$|\mathtt{VARIABLES}| > 1$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 1$
$\mathtt{M} > 1$
$\mathtt{M} < \mathtt{maxval}(\mathtt{VARIABLES.var})$

Symmetry Items of VARIABLES are permutable.

Arg. properties Functional dependency: MIN determined by VARIABLES and M.

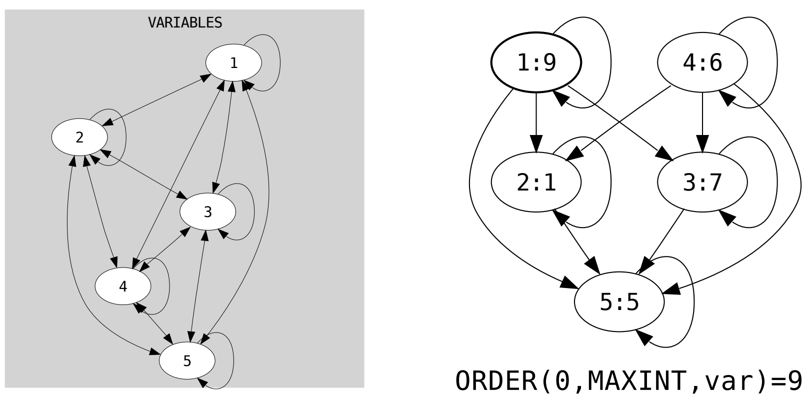
See also **comparison swapped:** MAXIMUM_MODULO.
specialisation: MINIMUM *(*variable mod constant *replaced by* variable*)*.

Keywords **characteristic of a constraint:** modulo, maxint, minimum.
constraint arguments: pure functional dependency.
constraint type: order constraint.
modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	$\bigvee \left(\begin{array}{l} \texttt{variables1.key} = \texttt{variables2.key}, \\ \texttt{variables1.var} \bmod \texttt{M} < \texttt{variables2.var} \bmod \texttt{M} \end{array} \right)$
Graph property(ies)	**ORDER**(0, MAXINT, var) = MIN

Graph model We use a similar definition that the one that was utilised for the MINIMUM constraint. Within the arc constraint we replace the condition $X < Y$ by the condition $(X \bmod \texttt{M}) < (Y \bmod \texttt{M})$.

Parts (A) and (B) of Figure 5.607 respectively show the initial and final graph associated with the second example of the **Example** slot. Since we use the **ORDER** graph property, the vertex of rank 0 (without considering the loops) associated with value 9 is outlined with a thick circle.



(A) **(B)**

Figure 5.607: Initial and final graph of the MINIMUM_MODULO constraint

5.271 MINIMUM_WEIGHT_ALLDIFFERENT

DESCRIPTION LINKS GRAPH

Origin [182]

Constraint MINIMUM_WEIGHT_ALLDIFFERENT(VARIABLES, MATRIX, COST)

Synonyms MINIMUM_WEIGHT_ALLDIFF, MINIMUM_WEIGHT_ALLDISTINCT, MIN_WEIGHT_ALLDIFF, MIN_WEIGHT_ALLDIFFERENT, MIN_WEIGHT_ALLDISTINCT.

Arguments

VARIABLES	:	collection(var−dvar)
MATRIX	:	collection(i−int, j−int, c−int)
COST	:	dvar

Restrictions

$|\texttt{VARIABLES}| > 0$
required(VARIABLES, var)
$\texttt{VARIABLES.var} \geq 1$
$\texttt{VARIABLES.var} \leq |\texttt{VARIABLES}|$
required(MATRIX, [i, j, c])
increasing_seq(MATRIX, [i, j])
$\texttt{MATRIX.i} \geq 1$
$\texttt{MATRIX.i} \leq |\texttt{VARIABLES}|$
$\texttt{MATRIX.j} \geq 1$
$\texttt{MATRIX.j} \leq |\texttt{VARIABLES}|$
$|\texttt{MATRIX}| = |\texttt{VARIABLES}| * |\texttt{VARIABLES}|$

Purpose

All variables of the VARIABLES collection should take a distinct value located within interval $[1, |\texttt{VARIABLES}|]$. In addition COST is equal to the sum of the costs associated with the fact that we assign value i to variable j. These costs are given by the matrix MATRIX.

Example

$$\left(\begin{array}{l}\langle 2,3,1,4\rangle, \\ \left\langle\begin{array}{lll} \mathtt{i}-1 & \mathtt{j}-1 & \mathtt{c}-4, \\ \mathtt{i}-1 & \mathtt{j}-2 & \mathtt{c}-1, \\ \mathtt{i}-1 & \mathtt{j}-3 & \mathtt{c}-7, \\ \mathtt{i}-1 & \mathtt{j}-4 & \mathtt{c}-0, \\ \mathtt{i}-2 & \mathtt{j}-1 & \mathtt{c}-1, \\ \mathtt{i}-2 & \mathtt{j}-2 & \mathtt{c}-0, \\ \mathtt{i}-2 & \mathtt{j}-3 & \mathtt{c}-8, \\ \mathtt{i}-2 & \mathtt{j}-4 & \mathtt{c}-2, \\ \mathtt{i}-3 & \mathtt{j}-1 & \mathtt{c}-3, \\ \mathtt{i}-3 & \mathtt{j}-2 & \mathtt{c}-2, \\ \mathtt{i}-3 & \mathtt{j}-3 & \mathtt{c}-1, \\ \mathtt{i}-3 & \mathtt{j}-4 & \mathtt{c}-6, \\ \mathtt{i}-4 & \mathtt{j}-1 & \mathtt{c}-0, \\ \mathtt{i}-4 & \mathtt{j}-2 & \mathtt{c}-0, \\ \mathtt{i}-4 & \mathtt{j}-3 & \mathtt{c}-6, \\ \mathtt{i}-4 & \mathtt{j}-4 & \mathtt{c}-5 \end{array}\right\rangle, 17\end{array}\right)$$

The MINIMUM_WEIGHT_ALLDIFFERENT constraint holds since the cost 17 corresponds to the sum $\mathtt{MATRIX}[(1-1)\cdot 4+2].\mathtt{c}+\mathtt{MATRIX}[(2-1)\cdot 4+3].\mathtt{c}+\mathtt{MATRIX}[(3-1)\cdot 4+1].\mathtt{c}+\mathtt{MATRIX}[(4-1)\cdot 4+4].\mathtt{c} = \mathtt{MATRIX}[2].\mathtt{c}+\mathtt{MATRIX}[7].\mathtt{c}+\mathtt{MATRIX}[9].\mathtt{c}+\mathtt{MATRIX}[16].\mathtt{c} = 1+8+3+5$.

All solutions

Figure 5.608 gives all solutions to the following non ground instance of the MINIMUM_WEIGHT_ALLDIFFERENT constraint:
$\mathtt{V}_1 \in [2,4]$, $\mathtt{V}_2 \in [2,3]$, $\mathtt{V}_3 \in [1,6]$, $\mathtt{V}_4 \in [2,5]$, $\mathtt{V}_5 \in [2,3]$, $\mathtt{V}_6 \in [1,6]$, $\mathtt{V} \in [0,25]$,
MINIMUM_WEIGHT_ALLDIFFERENT$(\langle \mathtt{V}_1,\mathtt{V}_2,\mathtt{V}_3,\mathtt{V}_4,\mathtt{V}_5,\mathtt{V}_6\rangle,$
$\langle 1\,1\,5,\ 1\,2\,0,\ 1\,3\,1,\ 1\,4\,1,\ 1\,5\,3,\ 1\,6\,0,$
$2\,1\,2,\ 2\,2\,7,\ 2\,3\,0,\ 2\,4\,2,\ 2\,5\,5,\ 2\,6\,1,$
$3\,1\,3,\ 3\,2\,3,\ 3\,3\,6,\ 3\,4\,6,\ 3\,5\,0,\ 3\,6\,9,$
$4\,1\,4,\ 4\,2\,3,\ 4\,3\,0,\ 4\,4\,0,\ 4\,5\,0,\ 4\,6\,2,$
$5\,1\,2,\ 5\,2\,0,\ 5\,3\,6,\ 5\,4\,3,\ 5\,5\,7,\ 5\,6\,2,$
$6\,1\,5,\ 6\,2\,4,\ 6\,3\,5,\ 6\,4\,4,\ 6\,5\,5,\ 6\,6\,4\rangle, \mathtt{C})$.

Typical

$|\mathtt{VARIABLES}| > 1$
$\mathrm{range}(\mathtt{MATRIX.c}) > 1$
$\mathtt{MATRIX.c} > 0$

Arg. properties

Functional dependency: COST determined by VARIABLES and MATRIX.

Algorithm

The Hungarian method for the assignment problem [254] can be used for evaluating the bounds of the COST variable. A filtering algorithm is described in [388]. It can be used for handling both side of the MINIMUM_WEIGHT_ALLDIFFERENT constraint:

- Evaluating a lower bound of the COST variable and pruning the variables of the VARIABLES collection in order to not exceed the maximum value of COST.
- Evaluating an upper bound of the COST variable and pruning the variables of the VARIABLES collection in order to not be under the minimum value of COST.

Systems

ALL_DIFFERENT in **SICStus**, ALL_DISTINCT in **SICStus**.

① $(\langle 4,2,1,5,3,6\rangle,\ \mathbf{21})$
② $(\langle 4,3,1,5,2,6\rangle,\ \mathbf{8})$
③ $(\langle 4,3,6,5,2,1\rangle,\ \mathbf{15})$

$$\begin{array}{cc} & \begin{array}{cccccc}1&2&3&4&5&6\end{array} \\ \begin{array}{c}① \\ \mathbf{21}\end{array} & \begin{pmatrix} V_1\ 5 & 0 & 1 & \mathbf{1} & 3 & 0 \\ V_2\ 2 & \mathbf{7} & 0 & 2 & 5 & 1 \\ V_3\ \mathbf{3} & 3 & 6 & 6 & 0 & 9 \\ V_4\ 4 & 3 & 0 & 0 & \mathbf{0} & 2 \\ V_5\ 2 & 0 & \mathbf{6} & 3 & 7 & 2 \\ V_6\ 5 & 4 & 5 & 4 & 5 & \mathbf{4} \end{pmatrix}\end{array} \quad \begin{array}{cc} & \begin{array}{cccccc}1&2&3&4&5&6\end{array} \\ \begin{array}{c}② \\ \mathbf{8}\end{array} & \begin{pmatrix} V_1\ 5 & 0 & 1 & \mathbf{1} & 3 & 0 \\ V_2\ 2 & 7 & \mathbf{0} & 2 & 5 & 1 \\ V_3\ \mathbf{3} & 3 & 6 & 6 & 0 & 9 \\ V_4\ 4 & 3 & 0 & 0 & \mathbf{0} & 2 \\ V_5\ 2 & \mathbf{0} & 6 & 3 & 7 & 2 \\ V_6\ 5 & 4 & 5 & 4 & 5 & \mathbf{4} \end{pmatrix}\end{array} \quad \begin{array}{cc} & \begin{array}{cccccc}1&2&3&4&5&6\end{array} \\ \begin{array}{c}③ \\ \mathbf{15}\end{array} & \begin{pmatrix} V_1\ 5 & 0 & 1 & \mathbf{1} & 3 & 0 \\ V_2\ 2 & 7 & \mathbf{0} & 2 & 5 & 1 \\ V_3\ 3 & 3 & 6 & 6 & 0 & \mathbf{9} \\ V_4\ 4 & 3 & 0 & 0 & \mathbf{0} & 2 \\ V_5\ 2 & \mathbf{0} & 6 & 3 & 7 & 2 \\ V_6\ \mathbf{5} & 4 & 5 & 4 & 5 & 4 \end{pmatrix}\end{array}$$

Figure 5.608: All solutions corresponding to the non ground example of the MINIMUM_WEIGHT_ALLDIFFERENT constraint of the **All solutions** slot

See also

attached to cost variant: ALLDIFFERENT.

common keyword: GLOBAL_CARDINALITY_WITH_COSTS *(cost filtering constraint,weighted assignment)*, SUM_OF_WEIGHTS_OF_DISTINCT_VALUES *(weighted assignment)*, WEIGHTED_PARTIAL_ALLDIFF *(cost filtering constraint,weighted assignment)*.

Keywords

application area: assignment.

characteristic of a constraint: core.

filtering: cost filtering constraint, Hungarian method for the assignment problem.

final graph structure: one_succ.

modelling: cost matrix, functional dependency.

problems: weighted assignment.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

Arc constraint(s) variables1.var = variables2.key

Graph property(ies)
- **NTREE** = 0
- **SUM_WEIGHT_ARC** $\left(\texttt{MATRIX} \left[\sum \left(\begin{array}{l} (\texttt{variables1.key} - 1) * |\texttt{VARIABLES}|, \\ \texttt{variables1.var} \end{array} \right) \right] \texttt{.c} \right) = \texttt{COST}$

Graph model Since each variable takes one value, and because of the arc constraint variables1 = variables.key, each vertex of the initial graph belongs to the final graph and has exactly one successor. Therefore the sum of the out-degrees of the vertices of the final graph is equal to the number of vertices of the final graph. Since the sum of the in-degrees is equal to the sum of the out-degrees, it is also equal to the number of vertices of the final graph. Since **NTREE** = 0, each vertex of the final graph belongs to a circuit. Therefore each vertex of the final graph has at least one predecessor. Since we saw that the sum of the in-degrees is equal to the number of vertices of the final graph, each vertex of the final graph has exactly one predecessor. We conclude that the final graph consists of a set of vertex-disjoint elementary circuits.

Finally the graph constraint expresses that the COST variable is equal to the sum of the elementary costs associated with each variable-value assignment. All these elementary costs are recorded in the MATRIX collection. More precisely, the cost c_{ij} is recorded in the attribute c of the $((i-1) \cdot |\texttt{VARIABLES}| + j)^{th}$ entry of the MATRIX collection. This is ensured by the increasing restriction that enforces that the items of the MATRIX collection are sorted in lexicographically increasing order according to attributes i and j.

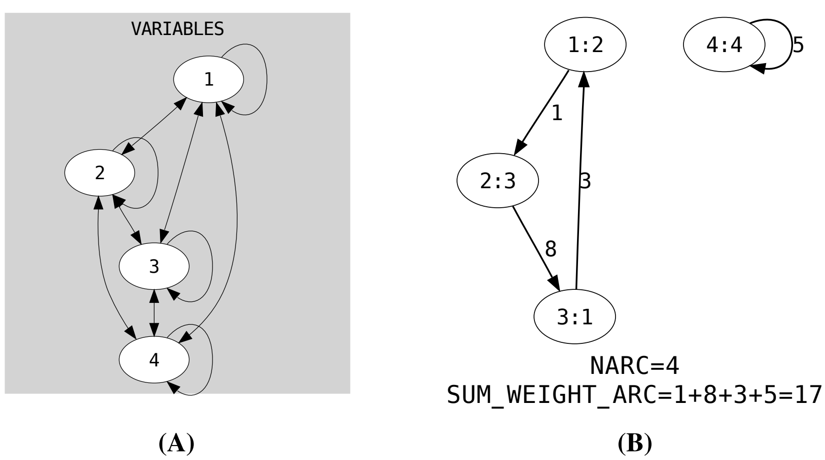


Figure 5.609: Initial and final graph of the MINIMUM_WEIGHT_ALLDIFFERENT constraint

Parts (A) and (B) of Figure 5.609 respectively show the initial and final graph associated with the **Example** slot. Since we use the **SUM_WEIGHT_ARC** graph property, the arcs of the final graph are stressed in bold. We also indicate their corresponding weights.

5.272 MULTI_GLOBAL_CONTIGUITY

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Derived from GLOBAL_CONTIGUITY.

Constraint MULTI_GLOBAL_CONTIGUITY(VARIABLES)

Synonym MULTI_CONTIGUITY.

Argument VARIABLES : collection(var−dvar)

Restrictions required(VARIABLES, var)
VARIABLES.var ≥ 0

Purpose Enforce all variables of the VARIABLES collection to be assigned a value greater than or equal to 0. In addition, each value v strictly greater than 0 should appear contiguously.

Example $(\langle 0, 2, 2, 1, 1, 0, 0, 5\rangle)$

The MULTI_GLOBAL_CONTIGUITY constraint holds since the sequence 0 2 2 1 1 0 0 5 contains no more than one group of contiguous 1, no more than one group of contiguous 2, and no more than one group of contiguous 5.

Typical $|\text{VARIABLES}| > 3$

Typical model nval(VARIABLES.var) > 2
ATLEAST(2, VARIABLES, 0)

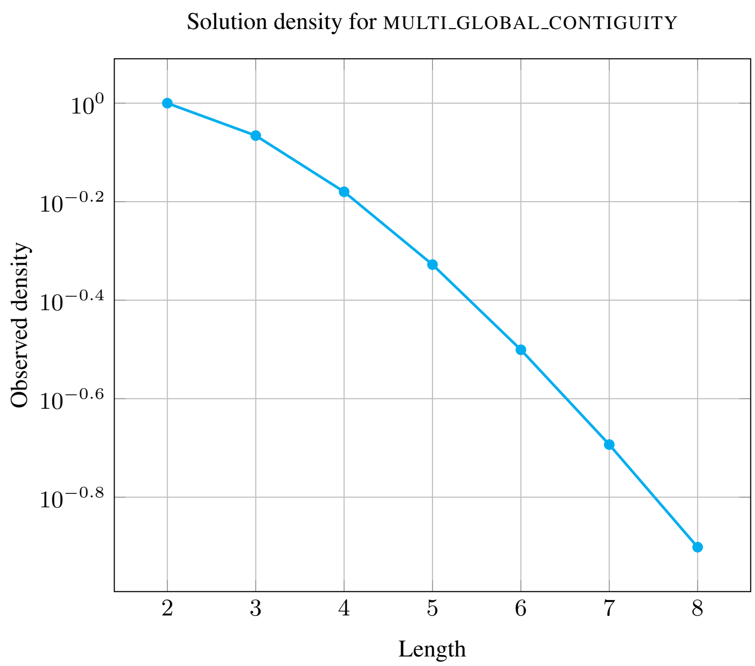
Symmetry Items of VARIABLES can be reversed.

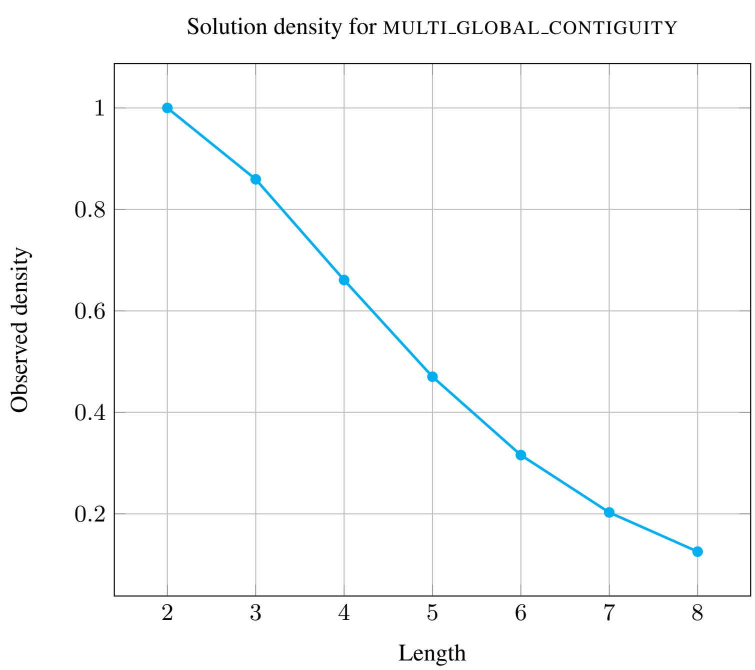
Arg. properties Contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	55	413	3656	37147	425069	5400481

Number of solutions for MULTI_GLOBAL_CONTIGUITY: domains $0..n$





See also **common keyword:** GROUP *(sequence)*.

implied by: ALL_EQUAL, ALLDIFFERENT, ALLDIFFERENT_EXCEPT_0, DECREASING, GLOBAL_CONTIGUITY, INCREASING.

Keywords **combinatorial object:** sequence.
constraint type: predefined constraint.

5.273 MULTI_INTER_DISTANCE

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin [313]

Constraint MULTI_INTER_DISTANCE(VARIABLES, LIMIT, DIST)

Synonyms MULTI_ALL_MIN_DISTANCE, MULTI_ALL_MIN_DIST, SLIDING_ATMOST, ATMOST_SLIDING.

Arguments

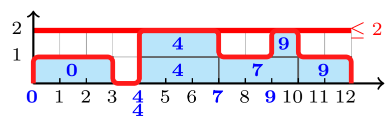
VARIABLES	:	collection(var−dvar)
LIMIT	:	int
DIST	:	int

Restrictions

required(VARIABLES, var)
LIMIT > 0
DIST > 0

Purpose Enforce that at most LIMIT variables of the collection VARIABLES are assigned values in any set consisting of DIST consecutive integer values.

Example $(\langle 4, 0, 9, 4, 7\rangle, 2, 3)$



The MULTI_INTER_DISTANCE constraint holds since, for each set of DIST $= 3$ consecutive values, no more than LIMIT $= 2$ variables of the VARIABLES collection $\langle 4, 0, 9, 4, 7\rangle$ are assigned a value from that set:

- At most two, in fact one, variables of the VARIABLES collection are assigned a value from the set $\{0, 1, 2\}$.
- At most two, in fact zero, variables of the VARIABLES collection are assigned a value from the set $\{1, 2, 3\}$.
- At most two, in fact two, variables of the VARIABLES collection are assigned a value from the set $\{2, 3, 4\}$.
- At most two, in fact two, variables of the VARIABLES collection are assigned a value from the set $\{3, 4, 5\}$.
- At most two, in fact two, variables of the VARIABLES collection are assigned a value from the set $\{4, 5, 6\}$.
- At most two, in fact one, variables of the VARIABLES collection are assigned a value from the set $\{5, 6, 7\}$.
- At most two, in fact one, variables of the VARIABLES collection are assigned a value from the set $\{6, 7, 8\}$.
- At most two, in fact two, variables of the VARIABLES collection are assigned a value from the set $\{7, 8, 9\}$.

Typical

LIMIT > 1
LIMIT $<$ |VARIABLES|
DIST > 1
DIST $<$range(VARIABLES.var)

Symmetries

- Items of VARIABLES are permutable.
- One and the same constant can be added to the var attribute of all items of VARIABLES.
- LIMIT can be increased.
- MINDIST can be decreased to any value ≥ 1.

Arg. properties

Contractible wrt. VARIABLES.

Usage

The MULTI_INTER_DISTANCE constraint was tested for scheduling tasks that all have the same fixed duration in the context of air traffic management.

Algorithm

P. Ouellet and C.-G. Quimper came up with a cubic time complexity algorithm achieving bound-consistency in [313].

See also

generalisation: CUMULATIVE *(*line segment*, of same length, replaced by* line segment*)*.

specialisation: ALL_MIN_DIST *(*LIMIT *parameter set to* 1*)*, CARDINALITY_ATMOST *(window of* DIST *consecutive values replaced by window of size* 1*)*.

Keywords

application area: air traffic management.

constraint type: predefined constraint, value constraint, scheduling constraint.

filtering: bound-consistency.

modelling: at most.

5.274 MULTIPLE

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic.

Constraint MULTIPLE(X, Y, C)

Arguments

X : dvar
Y : dvar
C : int

Restrictions

$\mathtt{X} \neq 0$
$\mathtt{Y} \neq 0$
$\mathtt{C} > 0$

Purpose Enforce $\max(|\mathtt{X}|, |\mathtt{Y}|) = \mathtt{C} \cdot \min(|\mathtt{X}|, |\mathtt{Y}|)$, (with $|\mathtt{X}| \neq 0$ and $|\mathtt{Y}| \neq 0$).

Example $(8, -2, 4)$

The MULTIPLE constraint holds since $\max(|8|, |-2|) = 4 \cdot \min(|8|, |-2|)$.

Typical $\mathtt{C} > 1$

Arg. properties Functional dependency: C determined by X and Y.

Keywords

constraint arguments: binary constraint.
constraint type: predefined constraint, arithmetic constraint.
modelling: functional dependency.

5.275 NAND

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint NAND(VAR, VARIABLES)

Synonym CLAUSE.

Arguments

VAR : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\texttt{VAR} \geq 0$
$\texttt{VAR} \leq 1$
$|\texttt{VARIABLES}| \geq 2$
required(VARIABLES, var)
$\texttt{VARIABLES.var} \geq 0$
$\texttt{VARIABLES.var} \leq 1$

Purpose Let VARIABLES be a collection of 0-1 variables $\texttt{VAR}_1, \texttt{VAR}_2, \ldots, \texttt{VAR}_n$ $(n \geq 2)$. Enforce $\texttt{VAR} = \neg(\texttt{VAR}_1 \wedge \texttt{VAR}_2 \wedge \cdots \wedge \texttt{VAR}_n)$.

Example

$(1, \langle 0, 0\rangle)$
$(1, \langle 0, 1\rangle)$
$(1, \langle 1, 0\rangle)$
$(0, \langle 1, 1\rangle)$
$(1, \langle 1, 0, 1\rangle)$

Symmetry Items of VARIABLES are permutable.

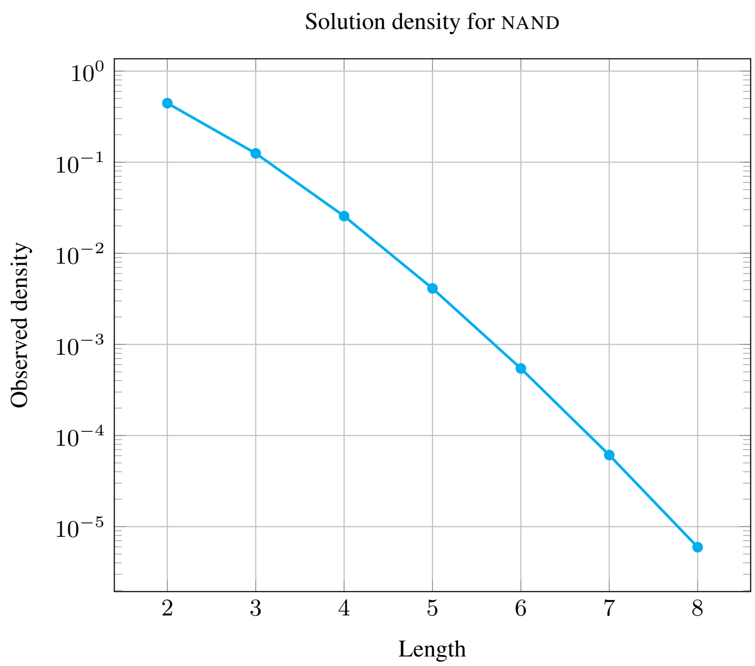
Arg. properties

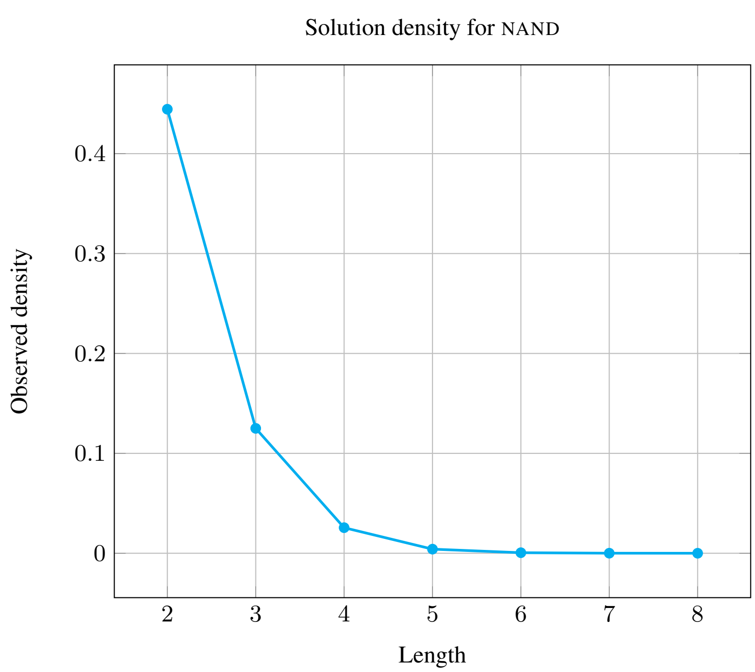
- Functional dependency: VAR determined by VARIABLES.
- Contractible wrt. VARIABLES when $\texttt{VAR} = 0$.
- Extensible wrt. VARIABLES when $\texttt{VAR} = 1$.
- Aggregate: VAR($\vee$), VARIABLES(union).

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	4	8	16	32	64	128	256

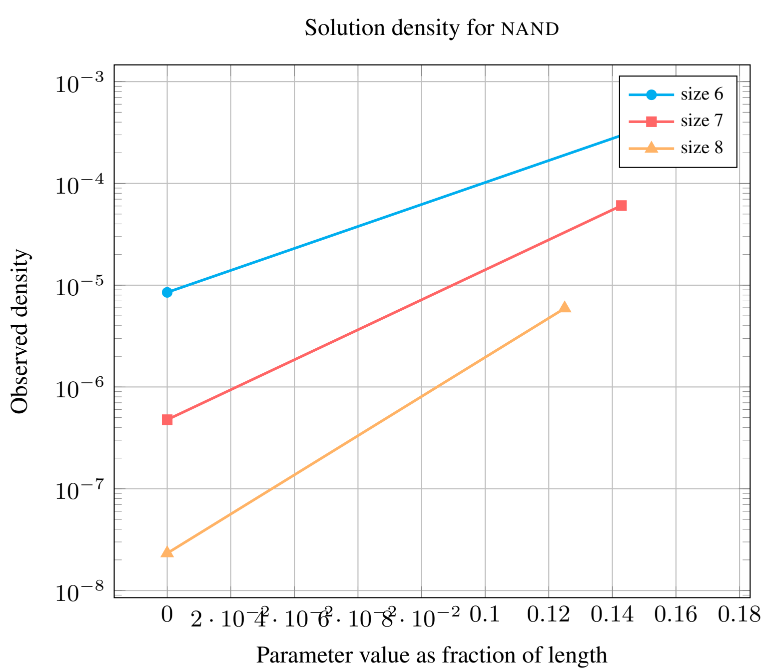
Number of solutions for NAND: domains $0..n$

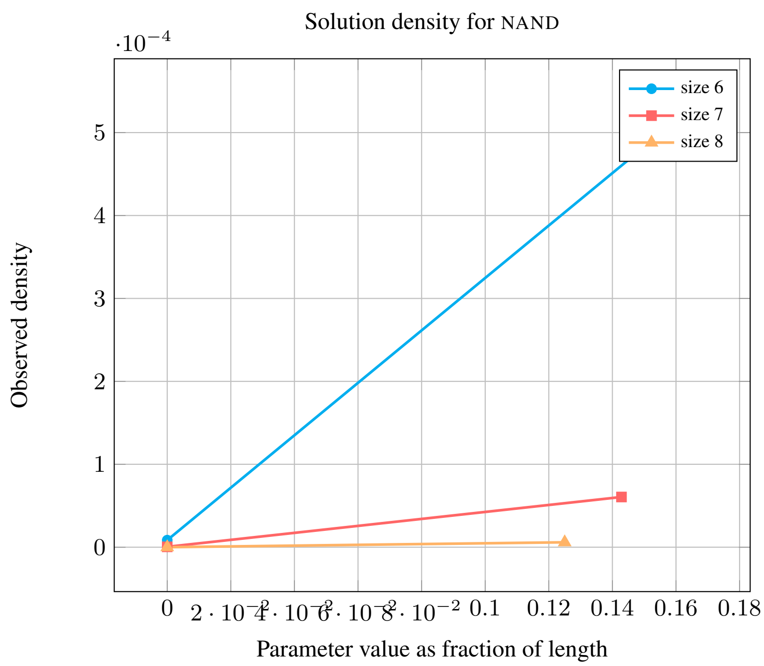

Solution density for NAND
Observed density
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
2
3
4
5
6
7
8
Length


Solution density for NAND
Observed density
0.4
0.3
0.2
0.1
0
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		4	8	16	32	64	128	256
Parameter value	0	1	1	1	1	1	1	1
	1	3	7	15	31	63	127	255

Solution count for NAND: domains $0..n$





Systems CLAUSE in **Choco**, CLAUSE in **Gecode**, #/\ in **SICStus**.

See also **common keyword:** AND, EQUIVALENT, IMPLY, NOR, OR, XOR *(Boolean constraint)*.

implies: ATLEAST_NVALUE.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: functional dependency.

Cond. implications NAND(VAR, VARIABLES)
with $|\texttt{VARIABLES}| > 2$
implies SOME_EQUAL(VARIABLES).

Automaton Figure 5.610 depicts the automaton associated with the NAND constraint. To the first argument VAR of the NAND constraint corresponds the first signature variable. To each variable $\mathtt{VAR}_i$ of the second argument VARIABLES of the NAND constraint corresponds the next signature variable. There is no signature constraint.

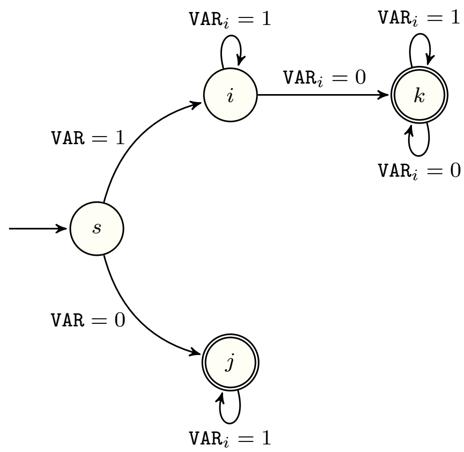


Figure 5.610: Automaton of the NAND constraint

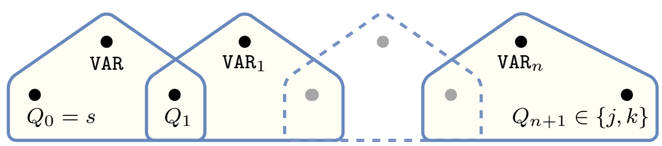


Figure 5.611: Hypergraph of the reformulation corresponding to the automaton of the NAND constraint

5.276 NCLASS

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from NVALUE.

Constraint NCLASS(NCLASS, VARIABLES, PARTITIONS)

Type VALUES : collection(val−int)

Arguments

NCLASS : dvar
VARIABLES : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

|VALUES| ≥ 1
required(VALUES, val)
distinct(VALUES, val)
NCLASS ≥ 0
NCLASS $\leq$ min(|VARIABLES|, |PARTITIONS|)
NCLASS $\leq$range(VARIABLES.var)
required(VARIABLES, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2

Purpose Number of partitions of the collection PARTITIONS such that at least one value is assigned to at least one variable of the collection VARIABLES.

Example $(2, \langle 3, 2, 7, 2, 6\rangle, \langle \mathtt{p} - \langle 1, 3\rangle, \mathtt{p} - \langle 4\rangle, \mathtt{p} - \langle 2, 6\rangle\rangle)$

Note that the values of $\langle 3, 2, 7, 2, 6\rangle$ occur within partitions p − $\langle 1, 3\rangle$ and p − $\langle 2, 6\rangle$ but not within p − $\langle 4\rangle$. Consequently, the NCLASS constraint holds since its first argument NCLASS is set to value 2.

Typical

NCLASS > 1
NCLASS $<$ |VARIABLES|
NCLASS $<$range(VARIABLES.var)
|VARIABLES| $>$ |PARTITIONS|

Symmetries

- Items of VARIABLES are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- All occurrences of two distinct tuples of values in VARIABLES.var or PARTITIONS.p.val can be swapped; all occurrences of a tuple of values in VARIABLES.var or PARTITIONS.p.val can be renamed to any unused tuple of values.

Arg. properties

- Functional dependency: NCLASS determined by VARIABLES and PARTITIONS.
- Extensible wrt. VARIABLES when NCLASS $=$ |PARTITIONS|.

Algorithm

[29, 46].

See also

related: NEQUIVALENCE *(*variable $\in$ partition *replaced by* variable mod constant*)*, NINTERVAL *(*variable $\in$ partition *replaced by* variable/constant*)*, NPAIR *(*variable $\in$ partition *replaced by* pair *of* variables*)*.

specialisation: NVALUE *(*variable $\in$ partition *replaced by* variable*)*.

used in graph description: IN_SAME_PARTITION.

Keywords

characteristic of a constraint: partition.

constraint arguments: pure functional dependency.

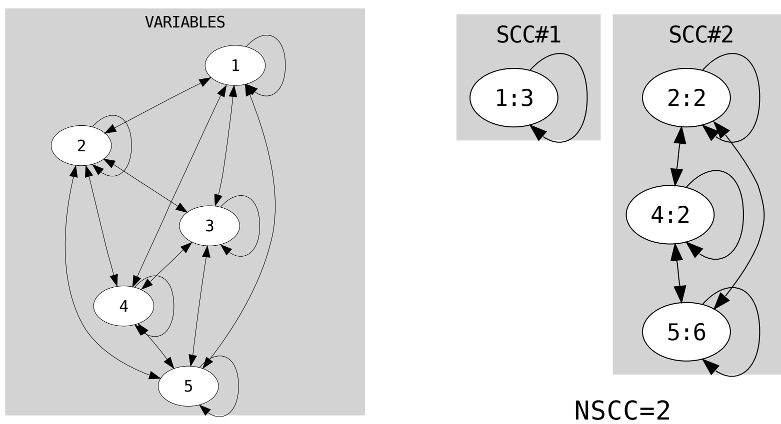
constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	**NSCC** = NCLASS

Graph model Parts (A) and (B) of Figure 5.612 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a class of values that was assigned to some variables of the VARIABLES collection. We effectively use two classes of values that respectively correspond to values $\{3\}$ and $\{2, 6\}$. Note that we do not consider value 7 since it does not belong to the different classes of values we gave: all corresponding arc constraints do not hold.



(A) **(B)**

Figure 5.612: Initial and final graph of the NCLASS constraint

5.277 NEQ

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic.

Constraint NEQ(VAR1, VAR2)

Synonym REL.

Arguments

VAR1	:	dvar
VAR2	:	dvar

Purpose Enforce the fact that two variables are not equal.

Example $(1, 8)$

The NEQ constraint holds since 1 is not equal to 8.

Symmetries

- Arguments are permutable w.r.t. permutation (VAR1, VAR2).
- A value in VAR1 or VAR2 can be renamed to any unused value.

Systems NEQ in **Choco**, REL in **Gecode**, #\= in **SICStus**.

See also

common keyword: GEQ, LEQ *(binary constraint,arithmetic constraint)*.

generalisation: NEQ_CST (constant *added*), NOT_ALL_EQUAL.

implied by: GT, LT.

negation: EQ.

system of constraints: ALLDIFFERENT.

Keywords

constraint arguments: binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.278 NEQ_CST

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic.

Constraint NEQ_CST(VAR1, VAR2, CST2)

Arguments

VAR1 : dvar
VAR2 : dvar
CST2 : int

Purpose Enforce the fact that the first variable is different from the sum of the second variable and the constant.

Example $(8, 2, 7)$

The NEQ_CST constraint holds since 8 is different from $2 + 7$.

Typical

$\mathtt{CST2} \neq 0$
$\mathtt{VAR1} \neq \mathtt{VAR2} + \mathtt{CST2}$

Symmetries

- Arguments are permutable w.r.t. permutation (VAR1) (VAR2, CST2).
- One and the same constant can be added to VAR1 and VAR2.
- One and the same constant can be added to VAR1 and CST2.

See also **negation:** EQ_CST.

specialisation: NEQ (constant *removed*).

Keywords **characteristic of a constraint:** disequality.

constraint arguments: binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.279 NEQUIVALENCE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin

Derived from NVALUE.

Constraint

NEQUIVALENCE(NEQUIV, M, VARIABLES)

Arguments

NEQUIV	:	dvar
M	:	int
VARIABLES	:	collection(var−dvar)

Restrictions

required(VARIABLES, var)
NEQUIV $\geq$ min(1, |VARIABLES|)
NEQUIV $\leq$ min(M, |VARIABLES|)
NEQUIV $\leq$range(VARIABLES.var)
M > 0

Purpose

NEQUIV is the number of distinct rests obtained by dividing the variables of the collection VARIABLES by M.

Example

$(2, 3, \langle 3, 2, 5, 6, 15, 3, 3\rangle)$

Since the expressions $3 \bmod 3 = 0$, $2 \bmod 3 = 2$, $5 \bmod 3 = 2$, $6 \bmod 3 = 0$, $15 \bmod 3 = 0$, $3 \bmod 3 = 0$, and $3 \bmod 3 = 0$ involve two distinct values (values 0 and 2), the first argument NEQUIV of the NEQUIVALENCE constraint is set to value 2.

Typical

NEQUIV > 1
NEQUIV $<$ |VARIABLES|
NEQUIV $<$range(VARIABLES.var)
M > 1
M $<$maxval(VARIABLES.var)

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value u of VARIABLES.var can be replaced by any other value v such that v is congruent to u modulo M.

Arg. properties

- Functional dependency: NEQUIV determined by M and VARIABLES.
- Contractible wrt. VARIABLES when NEQUIV $= 1$ and |VARIABLES| > 0.
- Contractible wrt. VARIABLES when NEQUIV $=$ |VARIABLES|.
- Extensible wrt. VARIABLES when NEQUIV $=$ M.

Algorithm

Since constraints $X = Y$ and $X \equiv Y \pmod{M}$ are similar, one should also use a similar algorithm as the one [29, 46] provided for constraint NVALUE.

See also

related: NCLASS *(*`variable` mod `constant` *replaced by* `variable` $\in$ `partition`*)*, NINTERVAL *(*`variable` mod `constant` *replaced by* `variable/constant`*)*, NPAIR *(*`variable` mod `constant` *replaced by* `pair` *of* `variables`*)*.

specialisation: NVALUE *(*`variable` mod `constant` *replaced by* `variable`*)*.

Keywords

constraint arguments: pure functional dependency.

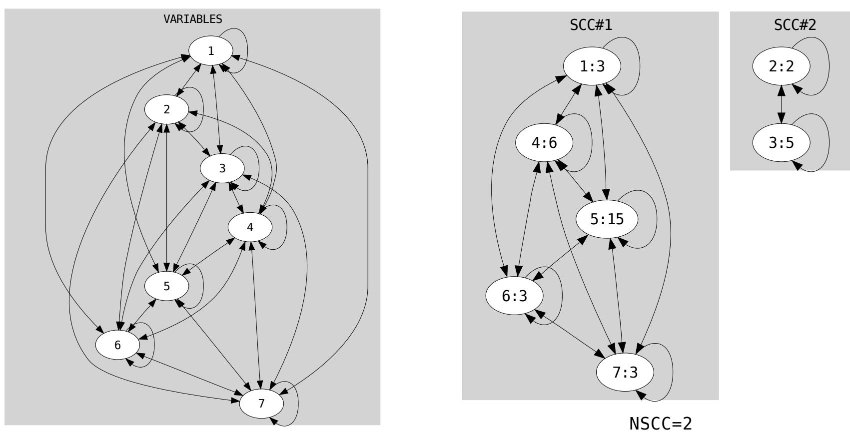
constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, functional dependency.

Arc input(s)	`VARIABLES`
Arc generator	$CLIQUE \mapsto$ `collection`(`variables1`, `variables2`)
Arc arity	2
Arc constraint(s)	`variables1.var` mod `M` = `variables2.var` mod `M`
Graph property(ies)	**NSCC**= `NEQUIV`

Graph model Parts (A) and (B) of Figure 5.613 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to one equivalence class: We have two equivalence classes that respectively correspond to values $\{3, 6, 15\}$ and $\{2, 5\}$.



(A) **(B)**

Figure 5.613: Initial and final graph of the NEQUIVALENCE constraint

5.280 NEXT_ELEMENT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint NEXT_ELEMENT(THRESHOLD, INDEX, TABLE, VAL)

Arguments

THRESHOLD	:	dvar
INDEX	:	dvar
TABLE	:	collection(index−int, value−dvar)
VAL	:	dvar

Restrictions

INDEX ≥ 1
INDEX $\leq$ |TABLE|
THRESHOLD $<$ INDEX
required(TABLE, [index, value])
|TABLE| > 0
TABLE.index ≥ 1
TABLE.index $\leq$ |TABLE|
distinct(TABLE, index)

Purpose INDEX is the smallest entry of TABLE strictly greater than THRESHOLD containing value VAL.

Example

$$\left(2, 3, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{value}-1, \\ \texttt{index}-2 & \texttt{value}-8, \\ \texttt{index}-3 & \texttt{value}-9, \\ \texttt{index}-4 & \texttt{value}-5, \\ \texttt{index}-5 & \texttt{value}-9 \end{array} \right\rangle, 9\right)$$

The NEXT_ELEMENT constraint holds since 3 is the smallest entry located after entry 2 that contains value 9.

Typical

|TABLE| > 1
range(TABLE.value) > 1

Usage Originally introduced for modelling the fact that a nucleotide has to be consumed as soon as possible at cycle INDEX after a given cycle represented by variable THRESHOLD.

See also **related:** MINIMUM_GREATER_THAN *(identify an element in a table)*, NEXT_GREATER_ELEMENT *(allow to iterate over the values of a table)*.

Keywords **characteristic of a constraint:** minimum, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint network structure: centered cyclic(3) constraint network(1).

constraint type: data constraint.

modelling: table.

Derived Collection	$\texttt{col}\left(\begin{array}{l}\texttt{ITEM}-\texttt{collection}(\texttt{index}-\texttt{dvar},\texttt{value}-\texttt{dvar}),\\ [\texttt{item}(\texttt{index}-\texttt{THRESHOLD},\texttt{value}-\texttt{VAL})]\end{array}\right)$
Arc input(s)	ITEM TABLE
Arc generator	$PRODUCT \mapsto$ collection(item, table)
Arc arity	2
Arc constraint(s)	• item.index < table.index • item.value = table.value
Graph property(ies)	**NARC** > 0
Sets	SUCC $\mapsto$ [source, variables − col(VARIABLES−collection(var−dvar), [item(var − TABLE.index)])]
Constraint(s) on sets	MINIMUM(INDEX, variables)

Graph model Parts (A) and (B) of Figure 5.614 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

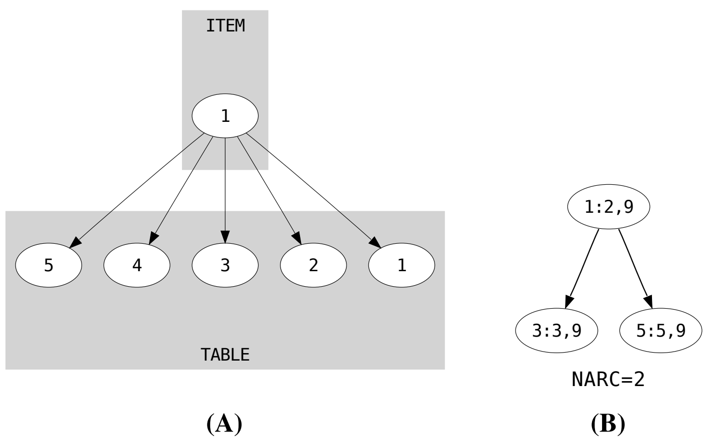


Figure 5.614: Initial and final graph of the NEXT_ELEMENT constraint

Automaton Figure 5.615 depicts the automaton associated with the NEXT_ELEMENT constraint. Let $\mathtt{I}_k$ and $\mathtt{V}_k$ respectively be the `index` and the `value` attributes of the k^{th} item of the `TABLE` collections. To each quintuple $(\mathtt{THRESHOLD}, \mathtt{INDEX}, \mathtt{VAL}, \mathtt{I}_k, \mathtt{V}_k)$ corresponds a signature variable S_k as well as the following signature constraint:

$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k < \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow S_k = 0 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k < \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow S_k = 1 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k = \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow S_k = 2 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k = \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow S_k = 3 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k > \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow S_k = 4 \wedge$$
$$((\mathtt{I}_k \leq \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k > \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow S_k = 5 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k < \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow S_k = 6 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k < \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow S_k = 7 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k = \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow S_k = 8 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k = \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow S_k = 9 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k > \mathtt{INDEX}) \wedge (\mathtt{V}_k = \mathtt{VAL})) \Leftrightarrow S_k = 10 \wedge$$
$$((\mathtt{I}_k > \mathtt{THRESHOLD}) \wedge (\mathtt{I}_k > \mathtt{INDEX}) \wedge (\mathtt{V}_k \neq \mathtt{VAL})) \Leftrightarrow S_k = 11.$$

The automaton is constructed in order to fulfil the following conditions:

- We look for an item of the `TABLE` collection such that $\mathtt{INDEX}_i > \mathtt{THRESHOLD}$ and $\mathtt{INDEX}_i = \mathtt{INDEX}$ and $\mathtt{VALUE}_i = \mathtt{VAL}$,
- There should not exist any item of the `TABLE` collection such that $\mathtt{INDEX}_i > \mathtt{THRESHOLD}$ and $\mathtt{INDEX}_i < \mathtt{INDEX}$ and $\mathtt{VALUE}_i = \mathtt{VAL}$.

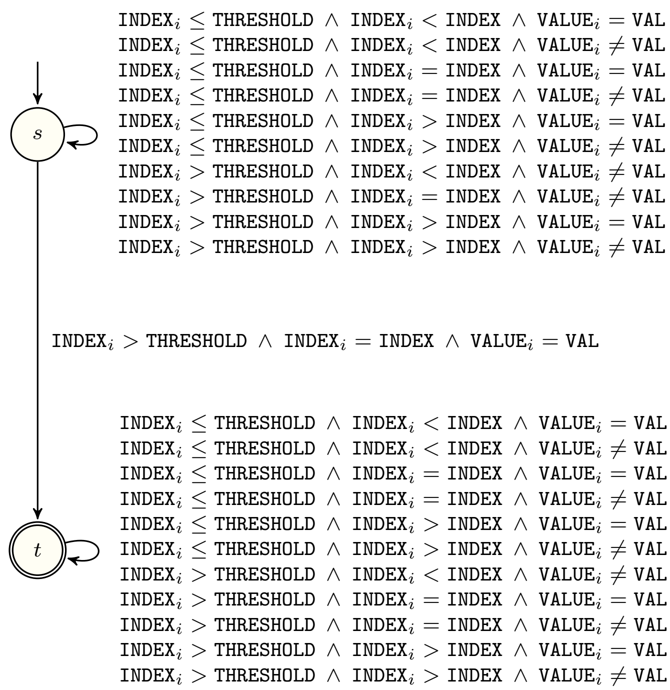


Figure 5.615: Automaton of the NEXT_ELEMENT constraint

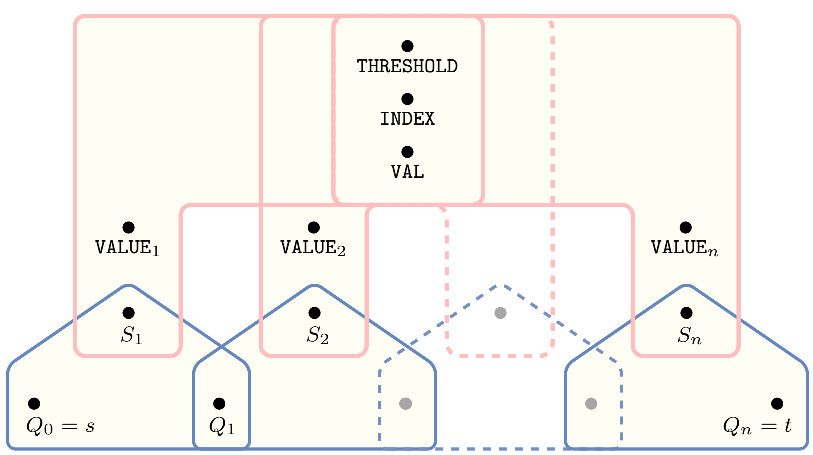


Figure 5.616: Hypergraph of the reformulation corresponding to the automaton of the NEXT_ELEMENT constraint

5.281 NEXT_GREATER_ELEMENT

DESCRIPTION LINKS GRAPH

Origin M. Carlsson

Constraint NEXT_GREATER_ELEMENT(VAR1, VAR2, VARIABLES)

Arguments

VAR1 : dvar
VAR2 : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\texttt{VAR1} < \texttt{VAR2}$
$|\texttt{VARIABLES}| > 0$
required(VARIABLES, var)

Purpose VAR2 is the value strictly greater than VAR1 located at the smallest possible entry of the table TABLE. In addition, the variables of the collection VARIABLES are sorted in strictly increasing order.

Example $(7, 8, \langle 3, 5, 8, 9\rangle)$

The NEXT_GREATER_ELEMENT constraint holds since:

- VAR2 is fixed to the first value 8 strictly greater than $\texttt{VAR1} = 7$,
- The var attributes of the items of the collection VARIABLES are sorted in strictly increasing order.

Typical

$|\texttt{VARIABLES}| > 1$
$\text{range}(\texttt{VARIABLES.var}) > 1$

Usage Originally introduced in [106] for modelling the fact that a nucleotide has to be consumed as soon as possible at cycle VAR2 after a given cycle VAR1.

Remark Similar to the MINIMUM_GREATER_THAN constraint, except for the fact that the var attributes are sorted.

Reformulation Let $V_1, V_2, \ldots, V_{|\texttt{VARIABLES}|}$ denote the variables of the collection of variables VARIABLES. By creating the extra variables M and $U_1, U_2, \ldots, U_{|\texttt{VARIABLES}|}$, the NEXT_GREATER_ELEMENT constraint can be expressed in term of the following constraints:

1. $V_1 < V_2 < \cdots < V_{|\texttt{VARIABLES}|}$
2. MAXIMUM(M, VARIABLES),
3. $\texttt{VAR2} > \texttt{VAR1}$,
4. $\texttt{VAR2} \leq M$,

5. $V_i \leq \texttt{VAR1} \Rightarrow U_i = M \;\; (i \in [1, |\texttt{VARIABLES}|])$,
6. $V_i > \texttt{VAR1} \Rightarrow U_i = V_i \;\; (i \in [1, |\texttt{VARIABLES}|])$,
7. MINIMUM$(\texttt{VAR2}, \langle U_1, U_2, \ldots, U_{|\texttt{VARIABLES}|}\rangle)$.

See also

common keyword: MINIMUM_GREATER_THAN *(order constraint)*.

implies: MINIMUM_GREATER_THAN.

related: NEXT_ELEMENT *(allow to iterate over the values of a table)*.

Keywords

characteristic of a constraint: minimum, derived collection.

constraint type: order constraint, data constraint.

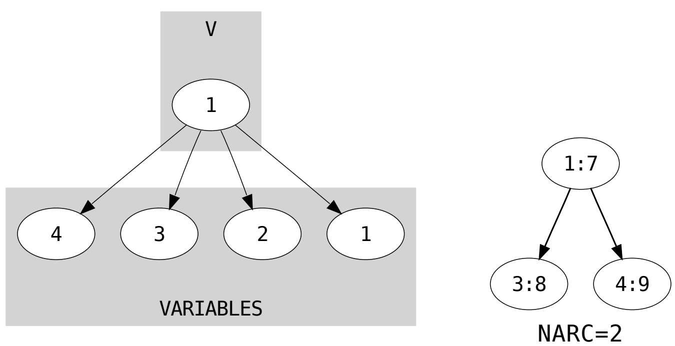
modelling: table.

Derived Collection	$\texttt{col}(\texttt{V}-\texttt{collection}(\texttt{var}-\texttt{dvar}), [\texttt{item}(\texttt{var}-\texttt{VAR1})])$

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto \texttt{collection}(\texttt{variables1}, \texttt{variables2})$
Arc arity	2
Arc constraint(s)	$\texttt{variables1.var} < \texttt{variables2.var}$
Graph property(ies)	$\mathbf{NARC} = \lvert\texttt{VARIABLES}\rvert - 1$

Arc input(s)	V VARIABLES
Arc generator	$PRODUCT \mapsto \texttt{collection}(\texttt{v}, \texttt{variables})$
Arc arity	2
Arc constraint(s)	$\texttt{v.var} < \texttt{variables.var}$
Graph property(ies)	$\mathbf{NARC} > 0$
Sets	$\mathsf{SUCC} \mapsto [\texttt{source}, \texttt{variables}]$
Constraint(s) on sets	$\textsc{minimum}(\texttt{VAR2}, \texttt{variables})$

Graph model Parts (A) and (B) of Figure 5.617 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.617: Initial and final graph of the NEXT_GREATER_ELEMENT constraint

Signature Since the first graph constraint uses the $PATH$ arc generator on the VARIABLES collection, the number of arcs of the corresponding initial graph is equal to $\lvert\texttt{VARIABLES}\rvert - 1$. Therefore the maximum number of arcs of the final graph is equal to $\lvert\texttt{VARIABLES}\rvert - 1$. For this reason we can rewrite $\mathbf{NARC} = \lvert\texttt{VARIABLES}\rvert - 1$ to $\mathbf{NARC} \geq \lvert\texttt{VARIABLES}\rvert - 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.282 NINTERVAL

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from NVALUE.

Constraint NINTERVAL(NVAL, VARIABLES, SIZE_INTERVAL)

Arguments

NVAL	:	dvar
VARIABLES	:	collection(var−dvar)
SIZE_INTERVAL	:	int

Restrictions

NVAL $\geq$ min(1, |VARIABLES|)
NVAL $\leq$ |VARIABLES|
required(VARIABLES, var)
SIZE_INTERVAL > 0

Purpose Consider the intervals of the form [SIZE_INTERVAL $\cdot$ k, SIZE_INTERVAL $\cdot$ k + SIZE_INTERVAL − 1] where k is an integer. NVAL is the number of intervals for which at least one value is assigned to at least one variable of the collection VARIABLES.

Example $(2, \langle 3, 1, 9, 1, 9\rangle, 4)$

In the example, the third argument SIZE_INTERVAL $=$ 4 defines the following family of intervals $[4 \cdot k, 4 \cdot k + 3]$, where k is an integer. Values 3, 1, 9, 1 and 9 are respectively located within intervals $[0, 3]$, $[0, 3]$, $[8, 11]$, $[0, 3]$ and $[8, 11]$. Since we only use the two intervals $[0, 3]$ and $[8, 11]$ the first argument of the NINTERVAL constraint is set to value 2.

Typical

NVAL > 1
NVAL $<$ |VARIABLES|
SIZE_INTERVAL > 1
SIZE_INTERVAL $<$range(VARIABLES.var)
(nval(VARIABLES.var) + SIZE_INTERVAL − 1)/SIZE_INTERVAL $<$ NVAL

Symmetries

- Items of VARIABLES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties

- Functional dependency: NVAL determined by VARIABLES and SIZE_INTERVAL.
- Contractible wrt. VARIABLES when NVAL $= 1$ and |VARIABLES| > 0.
- Contractible wrt. VARIABLES when NVAL $=$ |VARIABLES|.

Usage The NINTERVAL constraint is useful for counting the number of actually used periods, no matter how many time each period is used. A period can, for example, stand for a hour or for a day.

Algorithm [29, 46].

See also **related:** NCLASS *(*`variable`/`constant` *replaced by* `variable` $\in$ `partition`*)*, NEQUIVALENCE *(*`variable`/`constant` *replaced by* `variable` mod `constant`*)*, NPAIR *(*`variable`/`constant` *replaced by* `pair` *of* `variables`*)*.

specialisation: NVALUE *(*`variable`/`constant` *replaced by* `variable`*)*.

Keywords **constraint arguments:** pure functional dependency.

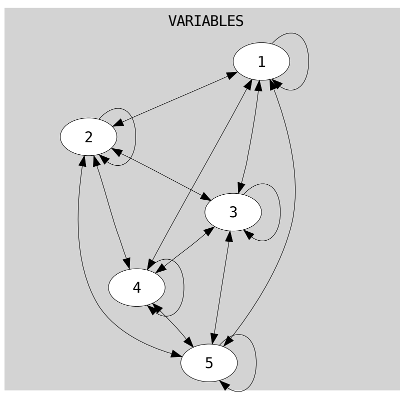
constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

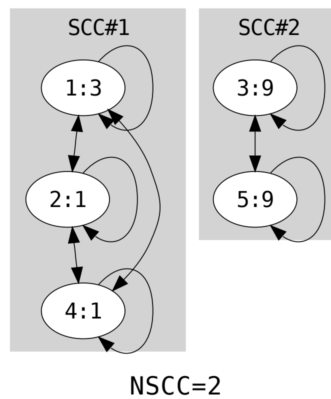
modelling: number of distinct equivalence classes, interval, functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL
Graph property(ies)	**NSCC** = NVAL

Graph model Parts (A) and (B) of Figure 5.618 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to those values of an interval that are assigned to some variables of the VARIABLES collection. The values 1, 3 and the value 9, which respectively correspond to intervals $[0, 3]$ and $[8, 11]$, are assigned to the variables of the VARIABLES collection.



(A)



(B)

Figure 5.618: Initial and final graph of the NINTERVAL constraint

5.283 NO_PEAK

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from PEAK.

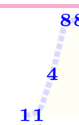
Constraint NO_PEAK(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions |VARIABLES| > 0
required(VARIABLES, var)

Purpose A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \dots, V_m$ is a *peak* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k > V_{k+1}$. The total number of peaks of the sequence of variables VARIABLES is equal to 0.

Example $(\langle 1, 1, 4, 8, 8 \rangle)$



The NO_PEAK constraint holds since the sequence 1 1 4 8 8 does not contain any peak.

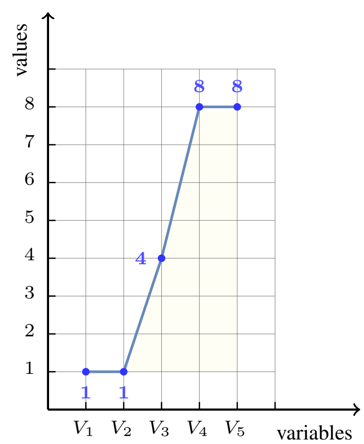


Figure 5.619: Illustration of the **Example** slot: a sequence of five variables V_1, V_2, V_3, V_4, V_5 respectively fixed to values 1, 1, 4, 8, 8 without any peak

Typical |VARIABLES| > 3
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

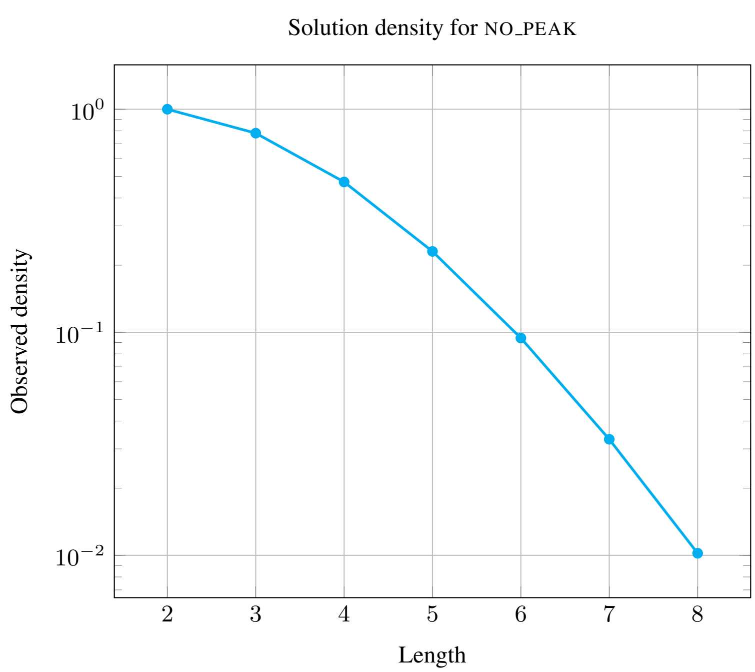
Arg. properties

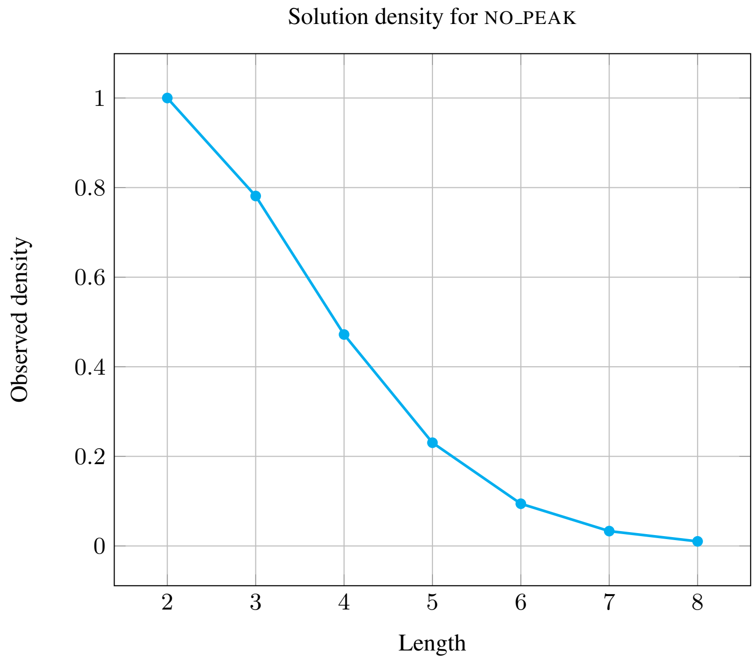
Contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	50	295	1792	11088	69498	439791

Number of solutions for NO_PEAK: domains $0..n$





See also

comparison swapped: NO_VALLEY.

generalisation: PEAK *(introduce a* `variable` *counting the number of peaks)*.

implied by: DECREASING, INCREASING.

implies: ALL_EQUAL_PEAK_MAX.

related: VALLEY.

Keywords

characteristic of a constraint: automaton, automaton without counters, automaton with same input symbol, reified automaton constraint.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(1).

Automaton Figure 5.620 depicts the automaton associated with the NO_PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: stationary/decreasing mode	$(\{> \mid =\}^*)$
t	: increasing mode	$(< \{< \mid =\}^*)$

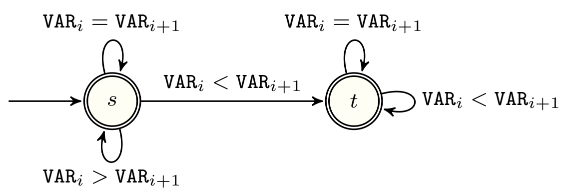


Figure 5.620: Automaton of the NO_PEAK constraint

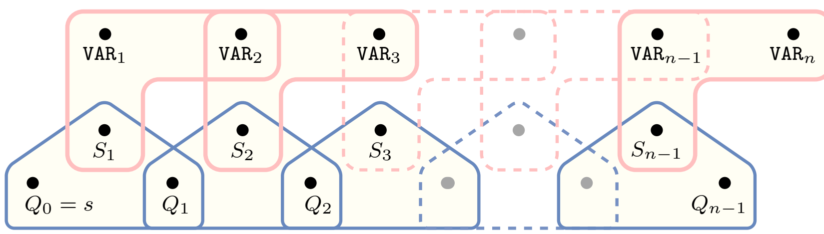


Figure 5.621: Hypergraph of the reformulation corresponding to the automaton of the NO_PEAK constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.284 NO_VALLEY

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from VALLEY.

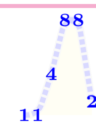
Constraint NO_VALLEY(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions $|\text{VARIABLES}| > 0$
required(VARIABLES, var)

Purpose A variable V_k $(1 < k < m)$ of the sequence of variables VARIABLES $= V_1, \dots, V_m$ is a *valley* if and only if there exists an i $(1 < i \leq k)$ such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k < V_{k+1}$. The total number of valleys of the sequence of variables VARIABLES is equal to 0.

Example $(\langle 1, 1, 4, 8, 8, 2\rangle)$



The NO_VALLEY constraint holds since the sequence 1 1 4 8 8 2 does not contain any valley.

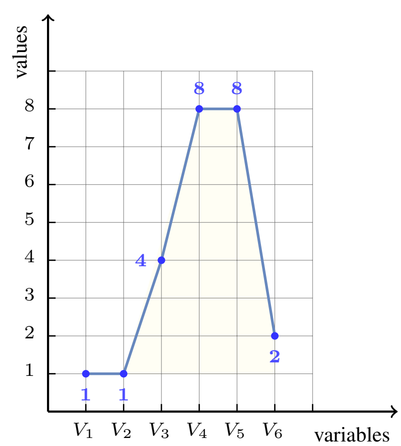


Figure 5.622: Illustration of the **Example** slot: a sequence of five variables V_1, V_2, V_3, V_4, V_5, V_6 respectively fixed to values 1, 1, 4, 8, 8, 2 without any valley

Typical $|\text{VARIABLES}| > 3$
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

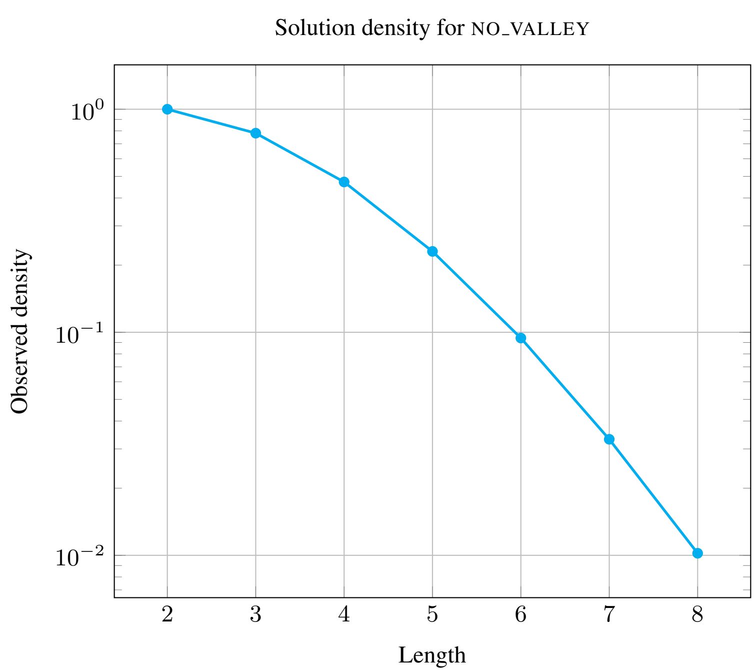
Arg. properties

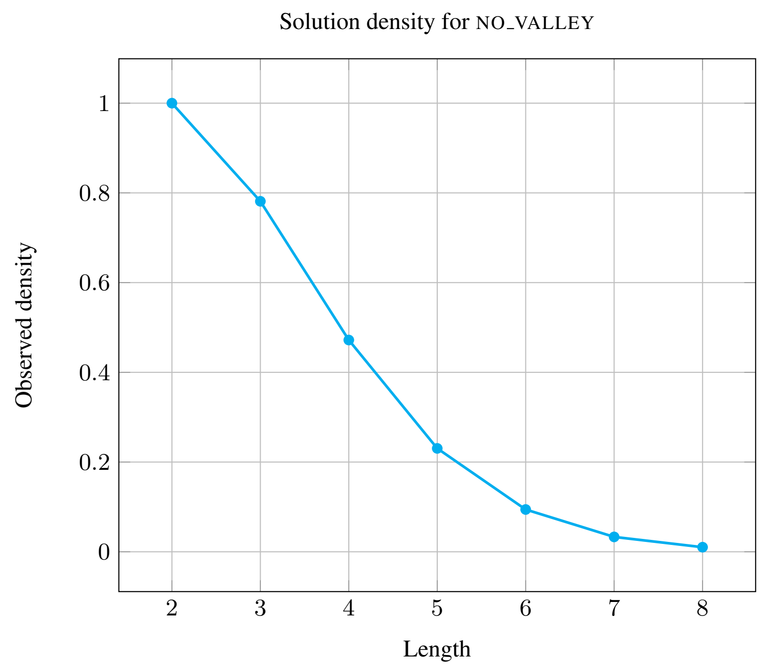
Contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	50	295	1792	11088	69498	439791

Number of solutions for NO_VALLEY: domains $0..n$





See also

comparison swapped: NO_PEAK.

generalisation: VALLEY *(introduce a* `variable` *counting the number of valleys)*.

implied by: DECREASING, GLOBAL_CONTIGUITY, INCREASING.

implies: ALL_EQUAL_VALLEY_MIN.

related: PEAK.

Keywords

characteristic of a constraint: automaton, automaton without counters, automaton with same input symbol, reified automaton constraint.

combinatorial object: sequence.

constraint network structure: sliding cyclic(1) constraint network(1).

Automaton Figure 5.623 depicts the automaton associated with the NO_VALLEY constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

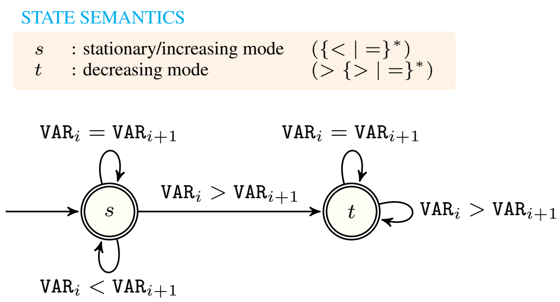


Figure 5.623: Automaton of the NO_VALLEY constraint

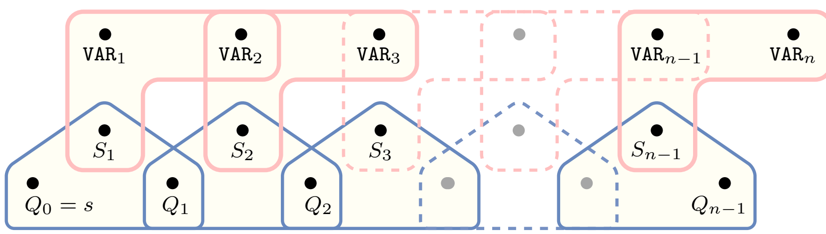


Figure 5.624: Hypergraph of the reformulation corresponding to the automaton of the NO_VALLEY constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

5.285 NON_OVERLAP_SBOXES

DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [42]

Constraint NON_OVERLAP_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonyms NON_OVERLAP, NON_OVERLAPPING.

Types

VARIABLES : collection(v−dvar)
INTEGERS : collection(v−int)
POSITIVES : collection(v−int)

Arguments

K : int
DIMS : sint
OBJECTS : collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES : collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

$|\text{VARIABLES}| \geq 1$
$|\text{INTEGERS}| \geq 1$
$|\text{POSITIVES}| \geq 1$
required(VARIABLES, v)
$|\text{VARIABLES}| = \text{K}$
required(INTEGERS, v)
$|\text{INTEGERS}| = \text{K}$
required(POSITIVES, v)
$|\text{POSITIVES}| = \text{K}$
$\text{POSITIVES.v} > 0$
$\text{K} > 0$
$\text{DIMS} \geq 0$
$\text{DIMS} < \text{K}$
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
$\text{OBJECTS.oid} \geq 1$
$\text{OBJECTS.oid} \leq |\text{OBJECTS}|$
$\text{OBJECTS.sid} \geq 1$
$\text{OBJECTS.sid} \leq |\text{SBOXES}|$
required(SBOXES, [sid, t, l])
$\text{SBOXES.sid} \geq 1$
$\text{SBOXES.sid} \leq |\text{SBOXES}|$

Purpose

Holds if, for each pair of objects (O_i, O_j), $i < j$, O_i and O_j do not overlap with respect to a set of dimensions depicted by `DIMS`. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a `K`-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id `sid`, shift offset `t`, and sizes `l`. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier `oid`, shape id `sid` and origin `x`.

An object O_i *does not overlap* an object O_j with respect to a set of dimensions depicted by `DIMS` if and only if, for all shifted box s_i associated with O_i and for all shifted box s_j associated with O_j, there exists a dimension $d \in$ `DIMS` such that the start of s_i in dimension d is greater than or equal to the end of s_j in dimension d, or the start of s_j in dimension d is greater than or equal to the end of s_i in dimension d.

Example

$$\left(\begin{array}{l} 2, \{0,1\}, \\ \left\langle\begin{array}{lll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 4,1\rangle, \\ \texttt{oid}-2 & \texttt{sid}-3 & \texttt{x}-\langle 2,2\rangle, \\ \texttt{oid}-3 & \texttt{sid}-4 & \texttt{x}-\langle 5,4\rangle \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-1 & \texttt{t}-\langle 1,0\rangle & \texttt{l}-\langle 1,3\rangle, \\ \texttt{sid}-1 & \texttt{t}-\langle 0,2\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,1\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 2,1\rangle & \texttt{l}-\langle 1,1\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle, \\ \texttt{sid}-4 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,1\rangle \end{array}\right\rangle \end{array}\right)$$

Figure 5.625 shows the objects of the example. Since O_1 and O_2 do not overlap, since O_1 and O_3 do not overlap, and since O_2 and O_3 also do not overlap, the NON_OVERLAP_SBOXES constraint holds.

Typical

$|\texttt{OBJECTS}| > 1$

Symmetries

- Items of `OBJECTS` are permutable.
- Items of `SBOXES` are permutable.
- Items of `OBJECTS.x`, `SBOXES.t` and `SBOXES.l` are permutable (*same permutation used*).
- `SBOXES.l.v` can be decreased to any value ≥ 1.

Arg. properties

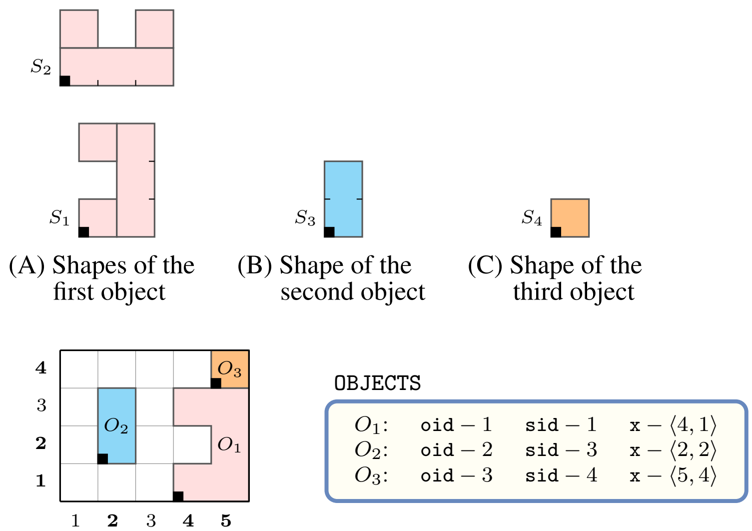
Suffix-contractible wrt. `OBJECTS`.

Remark

In addition from preventing objects to overlap, the DISJOINT_SBOXES constraint also enforces that borders and corners of objects are not directly in contact.

See also

common keyword: CONTAINS_SBOXES, COVEREDBY_SBOXES, COVERS_SBOXES (*geometrical constraint between shifted boxes*), DIFFN (*geometrical constraint,non-overlapping*), DISJOINT_SBOXES, EQUAL_SBOXES (*geometrical constraint*



(A) Shapes of the first object

(B) Shape of the second object

(C) Shape of the third object

(D) Three objects where O_1 does neither overlap O_2 nor O_3 and where O_2 and O_3 also do not overlap

Figure 5.625: (D) the three pairwise non-overlapping objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_1, S_3, S_4; (A), (B), (C) shapes S_1, S_2, S_3 and S_4 are respectively made up from 3, 3, 1 and 1 disjoint shifted box.

between shifted boxes), GEOST, GEOST_TIME *(geometrical constraint,non-overlapping)*, INSIDE_SBOXES, MEET_SBOXES, OVERLAP_SBOXES *(geometrical constraint between shifted boxes)*, VISIBLE *(geometrical constraint)*.

implied by: DISJOINT_SBOXES.

Keywords

constraint type: logic.

geometry: geometrical constraint, non-overlapping.

Logic

- $\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D})$
- $\texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D}) + \texttt{S1.l}(\texttt{D})$
- $\texttt{non\_overlap\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=}$
  $\exists \texttt{D} \in \texttt{Dims}$
  $\bigvee \left(\begin{array}{l} \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \leq \\ \texttt{origin}\left(\begin{array}{l}\texttt{O2},\\ \texttt{S2},\\ \texttt{D}\end{array}\right) \\ \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) \leq \\ \texttt{origin}\left(\begin{array}{l}\texttt{O1},\\ \texttt{S1},\\ \texttt{D}\end{array}\right) \end{array} , \right)$
- $\texttt{non\_overlap\_objects}(\texttt{Dims},\texttt{O1},\texttt{O2}) \stackrel{\text{def}}{=}$
  $\forall \texttt{S1} \in \texttt{sboxes}([\texttt{O1.sid}])$
  $\forall \texttt{S2} \in \texttt{sboxes}\left(\left[\ \texttt{O2.sid} \ \right] \right)$
  $\texttt{non\_overlap\_sboxes}\left(\begin{array}{l}\texttt{Dims},\\ \texttt{O1},\\ \texttt{S1},\\ \texttt{O2},\\ \texttt{S2}\end{array}\right)$
- $\texttt{all\_non\_overlap}(\texttt{Dims},\texttt{OIDS}) \stackrel{\text{def}}{=}$
  $\forall \texttt{O1} \in \texttt{objects}(\texttt{OIDS})$
  $\forall \texttt{O2} \in \texttt{objects}\left(\ \texttt{OIDS} \ \right)$
  $\begin{array}{l}\texttt{O1.oid} < \\ \texttt{O2.oid}\end{array} \Rightarrow$
  $\texttt{non\_overlap\_objects}\left(\begin{array}{l}\texttt{Dims},\\ \texttt{O1},\\ \texttt{O2}\end{array}\right)$
- $\texttt{all\_non\_overlap}(\texttt{DIMENSIONS},\texttt{OIDS})$

5.286 NOR

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint NOR(VAR, VARIABLES)

Synonym CLAUSE.

Arguments

VAR : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\texttt{VAR} \geq 0$
$\texttt{VAR} \leq 1$
$|\texttt{VARIABLES}| \geq 2$
required(VARIABLES, var)
$\texttt{VARIABLES.var} \geq 0$
$\texttt{VARIABLES.var} \leq 1$

Purpose Let VARIABLES be a collection of 0-1 variables $\texttt{VAR}_1, \texttt{VAR}_2, \ldots, \texttt{VAR}_n$ $(n \geq 2)$. Enforce $\texttt{VAR} = \neg(\texttt{VAR}_1 \vee \texttt{VAR}_2 \vee \cdots \vee \texttt{VAR}_n)$.

Example

$(1, \langle 0, 0\rangle)$
$(0, \langle 0, 1\rangle)$
$(0, \langle 1, 0\rangle)$
$(0, \langle 1, 1\rangle)$
$(0, \langle 1, 0, 1\rangle)$

Symmetry Items of VARIABLES are permutable.

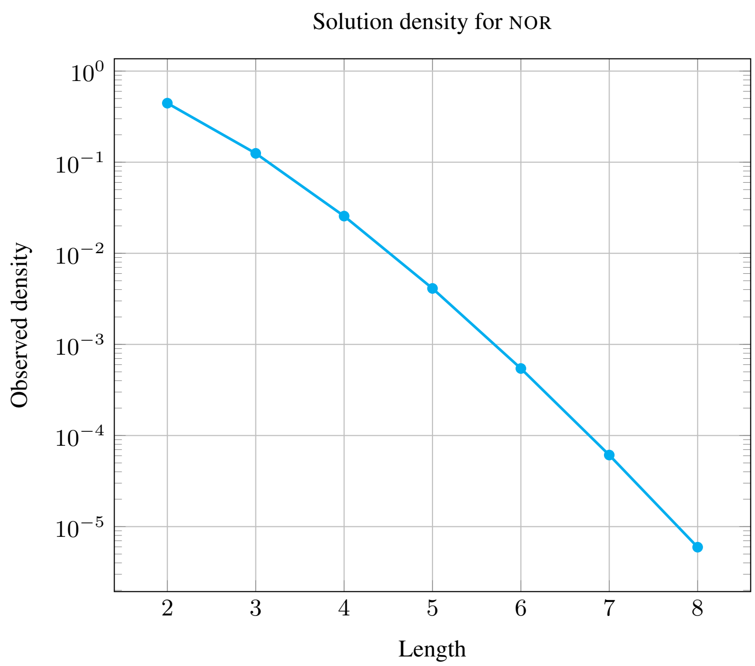
Arg. properties

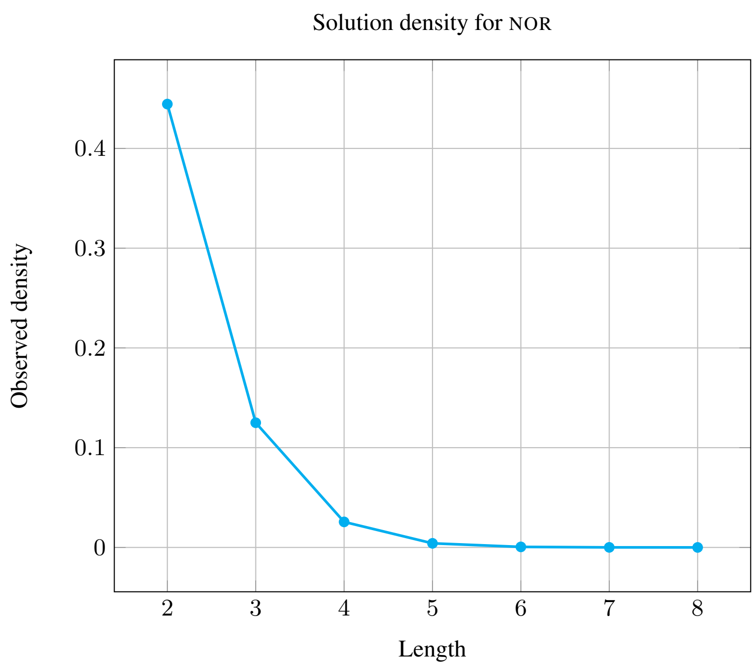
- Functional dependency: VAR determined by VARIABLES.
- Contractible wrt. VARIABLES when $\texttt{VAR} = 1$.
- Extensible wrt. VARIABLES when $\texttt{VAR} = 0$.
- Aggregate: VAR($\wedge$), VARIABLES(union).

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	4	8	16	32	64	128	256

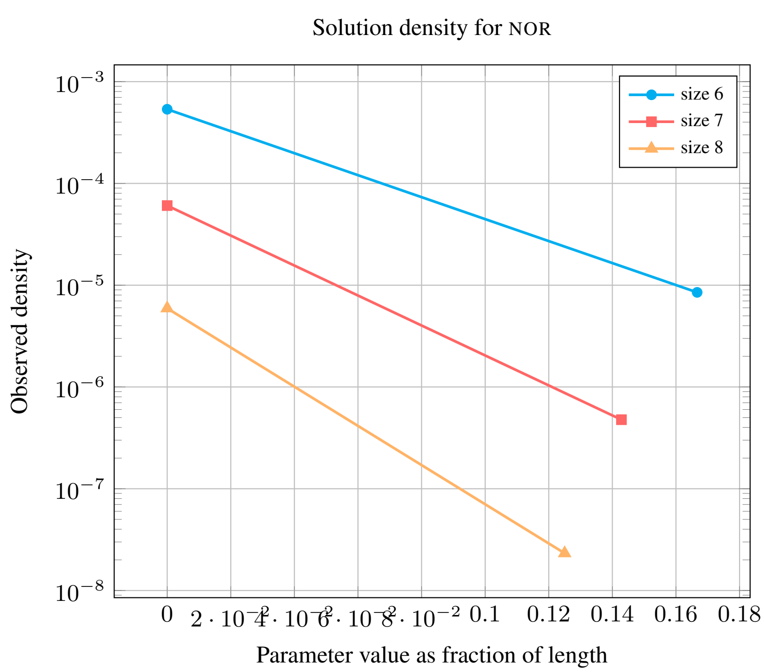
Number of solutions for NOR: domains $0..n$

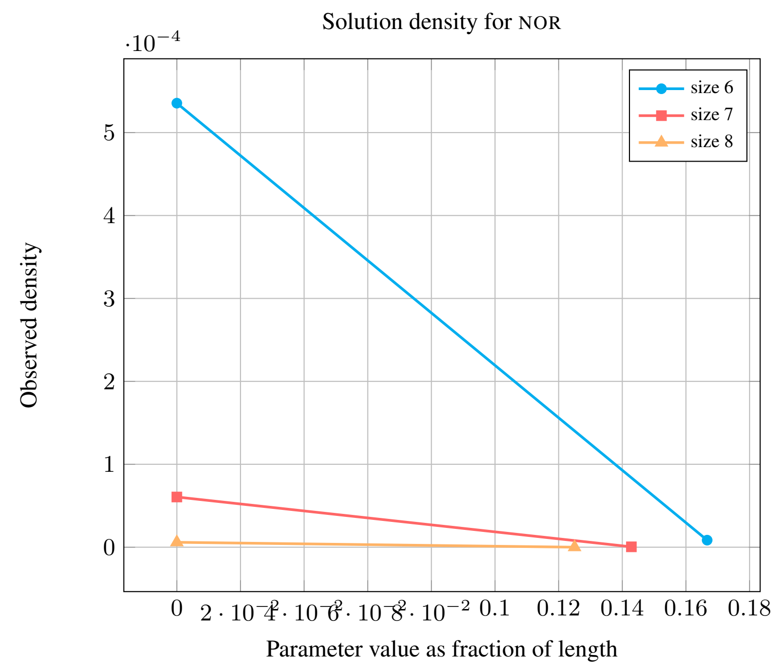

Solution density for NOR
Observed density
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
2 3 4 5 6 7 8
Length


Solution density for NOR
Observed density
0.4
0.3
0.2
0.1
0
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		4	8	16	32	64	128	256
Parameter value	0	3	7	15	31	63	127	255
	1	1	1	1	1	1	1	1

Solution count for NOR: domains $0..n$





Systems REIFIEDXNOR in **Choco**, CLAUSE in **Gecode**, #\/ in **SICStus**.

See also **common keyword:** AND, EQUIVALENT, IMPLY, NAND, OR, XOR *(Boolean constraint)*.

implies: ATLEAST_NVALUE, SOFT_ALL_EQUAL_MIN_CTR.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: functional dependency.

Cond. implications NOR(VAR, VARIABLES)
with $|\texttt{VARIABLES}| > 2$
implies SOME_EQUAL(VARIABLES).

Automaton Figure 5.626 depicts the automaton associated with the NOR constraint. To the first argument VAR of the NOR constraint corresponds the first signature variable. To each variable $\mathtt{VAR}_i$ of the second argument VARIABLES of the NOR constraint corresponds the next signature variable. There is no signature constraint.

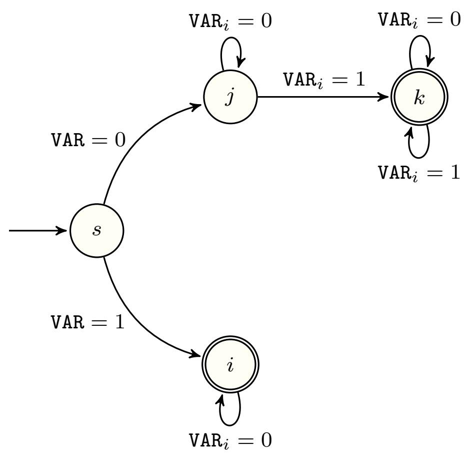


Figure 5.626: Automaton of the NOR constraint

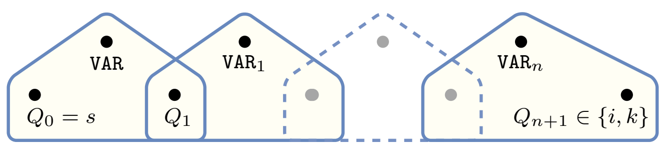


Figure 5.627: Hypergraph of the reformulation corresponding to the automaton of the NOR constraint

5.287 NOT_ALL_EQUAL

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin CHIP

Constraint NOT_ALL_EQUAL(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions required(VARIABLES, var)
|VARIABLES| >1

Purpose The variables of the collection VARIABLES should take more than a single value.

Example $(\langle 3,1,3,3,3\rangle)$

The NOT_ALL_EQUAL constraint holds since the collection $\langle 3,1,3,3,3\rangle$ involves more than one value (i.e., values 1 and 3).

Typical |VARIABLES| >2
nval(VARIABLES.var) >2

Symmetries
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Extensible wrt. VARIABLES.

Algorithm If the intersection of the domains of the variables of the VARIABLES collection is empty the NOT_ALL_EQUAL constraint is entailed. Otherwise, when only a single variable V remains not fixed, remove the unique value (unique since the constraint is not entailed) taken by the other variables from the domain of V.

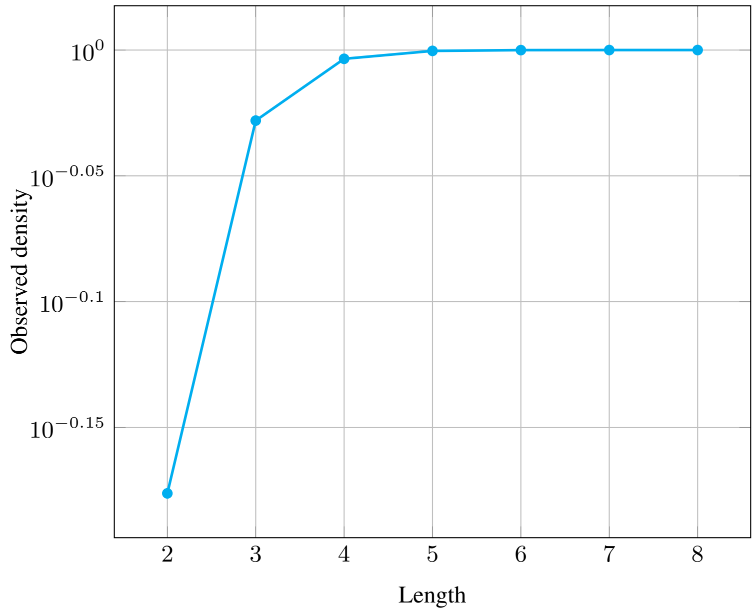
Reformulation The NOT_ALL_EQUAL(VARIABLES) constraint can be expressed as ATLEAST_NVALUE(2, VARIABLES).

Counting

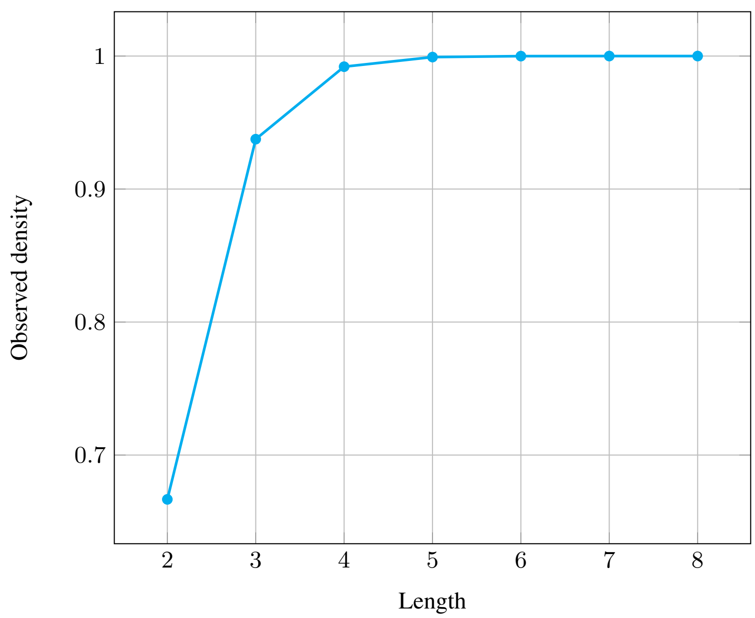
Length (n)	2	3	4	5	6	7	8
Solutions	6	60	620	7770	117642	2097144	43046712

Number of solutions for NOT_ALL_EQUAL: domains $0..n$

Solution density for NOT_ALL_EQUAL



Solution density for NOT_ALL_EQUAL



Systems REL in **Gecode**.

See also **generalisation:** NVALUE *(introduce a variable for counting the number of distinct values)*.

implied by: ALLDIFFERENT.

negation: ALL_EQUAL.

specialisation: NEQ *(when go down to two variables)*.

used in reformulation: ATLEAST_NVALUE.

Keywords

characteristic of a constraint: disequality, automaton, automaton without counters, reified automaton constraint.

constraint network structure: sliding cyclic(1) constraint network(1).

constraint type: value constraint.

filtering: arc-consistency.

final graph structure: equivalence.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSCC**> 1

Graph model Parts (A) and (B) of Figure 5.628 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The NOT_ALL_EQUAL holds since the final graph contains more than one strongly connected component.

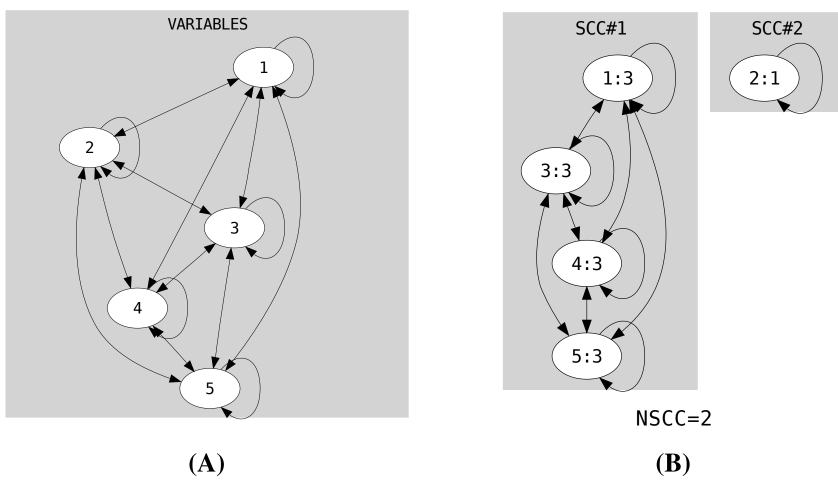


Figure 5.628: Initial and final graph of the NOT_ALL_EQUAL constraint

Automaton Figure 5.629 depicts the automaton associated with the NOT_ALL_EQUAL constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and S_i: $\texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow S_i$.

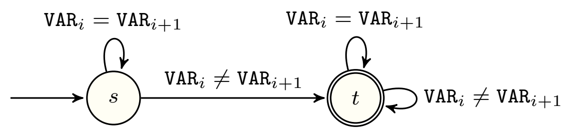


Figure 5.629: Automaton of the NOT_ALL_EQUAL constraint

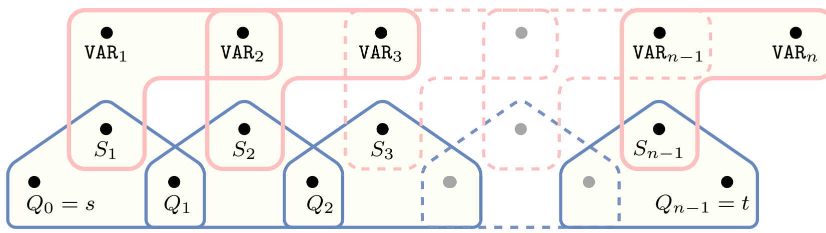


Figure 5.630: Hypergraph of the reformulation corresponding to the automaton of the NOT_ALL_EQUAL constraint

5.288 NOT_IN

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from IN.

Constraint NOT_IN(VAR, VALUES)

Arguments

VAR : dvar
VALUES : collection(val−int)

Restrictions

required(VALUES, val)
distinct(VALUES, val)

Purpose Enforce VAR to be assigned a value different from the values of the VALUES collection.

Example $(2, \langle 1, 3\rangle)$

The constraint NOT_IN holds since the value of its first argument VAR $= 2$ does not occur within the collection $\langle 1, 3\rangle$.

Typical $|\text{VALUES}| > 1$

Symmetries

- Items of VALUES are permutable.
- One and the same constant can be added to VAR as well as to the val attribute of all items of VALUES.

Arg. properties Contractible wrt. VALUES.

Remark Entailment occurs immediately after posting this constraint and removing all values in VALUES from VAR.

Systems NOTMEMBER in **Choco**, REL in **Gecode**.

Used in GROUP.

See also **negation:** IN.

Keywords **characteristic of a constraint:** disequality, automaton, automaton without counters, reified automaton constraint, derived collection.

constraint arguments: unary constraint.

constraint network structure: centered cyclic(1) constraint network(1).

constraint type: value constraint.

filtering: arc-consistency, entailment.

modelling: excluded, domain definition.

Derived Collection $\texttt{col}(\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}), [\texttt{item}(\texttt{var}-\texttt{VAR})])$

Arc input(s) VARIABLES VALUES

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{variables}, \texttt{values})$

Arc arity 2

Arc constraint(s) $\texttt{variables.var} = \texttt{values.val}$

Graph property(ies) $\mathbf{NARC} = 0$

Graph model Figure 5.631 shows the initial graph associated with the **Example** slot. Since we use the $\mathbf{NARC} = 0$ graph property the corresponding final graph is empty.

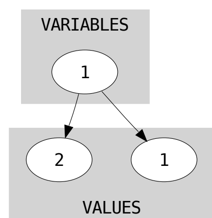


Figure 5.631: Initial graph of the NOT_IN constraint (the final graph is empty)

Signature Since 0 is the smallest number of arcs of the final graph we can rewrite $\mathbf{NARC} = 0$ to $\mathbf{NARC} \leq 0$. This leads to simplify $\overline{\underline{\mathbf{NARC}}}$ to $\underline{\mathbf{NARC}}$.

Automaton Figure 5.632 depicts the automaton associated with the NOT_IN constraint. Let $\texttt{VAL}_i$ be the val attribute of the i^{th} item of the VALUES collection. To each pair $(\texttt{VAR}, \texttt{VAL}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $\texttt{VAR} = \texttt{VAL}_i \Leftrightarrow S_i$.

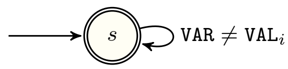


Figure 5.632: Automaton of the NOT_IN constraint

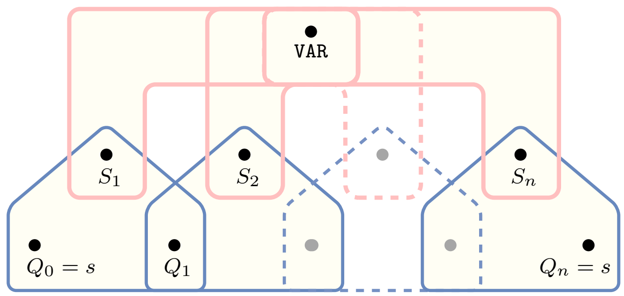


Figure 5.633: Hypergraph of the reformulation corresponding to the automaton of the NOT_IN constraint

5.289 NPAIR

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

Derived from NVALUE.

Constraint

NPAIR(NPAIRS, PAIRS)

Arguments

NPAIRS : dvar
PAIRS : collection(x−dvar, y−dvar)

Restrictions

$\texttt{NPAIRS} \geq \texttt{min}(1, |\texttt{PAIRS}|)$
$\texttt{NPAIRS} \leq |\texttt{PAIRS}|$
$\texttt{required}(\texttt{PAIRS}, [\texttt{x}, \texttt{y}])$

Purpose

NPAIRS is the number of distinct pairs of values assigned to the pairs of variables of the collection PAIRS.

Example

$$\left(2, \left\langle \begin{array}{ll} \texttt{x}-3 & \texttt{y}-1, \\ \texttt{x}-1 & \texttt{y}-5, \\ \texttt{x}-3 & \texttt{y}-1, \\ \texttt{x}-3 & \texttt{y}-1, \\ \texttt{x}-1 & \texttt{y}-5 \end{array} \right\rangle \right)$$

The NPAIR constraint holds since its first argument NPAIRS $= 2$ is set to the number of distinct pairs $\langle \texttt{x}-3\ \texttt{y}-1\rangle$ and $\langle \texttt{x}-1\ \texttt{y}-5\rangle$ of its second argument PAIRS.

Typical

$\texttt{NPAIRS} > 1$
$\texttt{NPAIRS} < |\texttt{PAIRS}|$
$|\texttt{PAIRS}| > 1$
$\texttt{range}(\texttt{PAIRS.x}) > 1$
$\texttt{range}(\texttt{PAIRS.y}) > 1$

Symmetries

- Items of PAIRS are permutable.
- Attributes of PAIRS are permutable w.r.t. permutation (x, y) (*permutation applied to all items*).
- All occurrences of two distinct tuples of values of NPAIRS can be swapped; all occurrences of a tuple of values of NPAIRS can be renamed to any unused tuple of values.

Arg. properties

- Functional dependency: NPAIRS determined by PAIRS.
- Contractible wrt. PAIRS when $\texttt{NPAIRS} = 1$ and $|\texttt{PAIRS}| > 0$.
- Contractible wrt. PAIRS when $\texttt{NPAIRS} = |\texttt{PAIRS}|$.

Remark

This is an example of a *number of distinct values* constraint where there is more than one attribute that is associated with each vertex of the final graph.

See also

related: NCLASS (`pair` *of* `variables` *replaced by* `variable` $\in$ `partition`), NEQUIVALENCE (`pair` *of* `variables` *replaced by* `variable` mod `constant`), NINTERVAL (`pair` *of* `variables` *replaced by* `variable`/`constant`).

specialisation: NVALUE (`pair` *of* `variables` *replaced by* `variable`).

Keywords

characteristic of a constraint: pair.

constraint arguments: pure functional dependency.

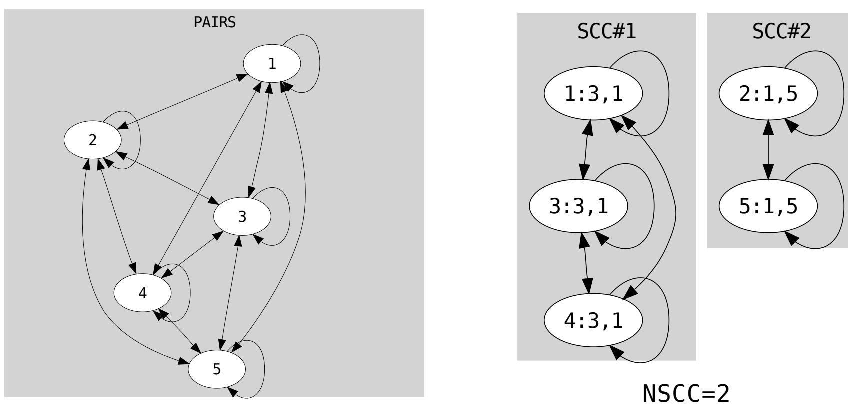
constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, functional dependency.

Arc input(s)	PAIRS
Arc generator	$CLIQUE \mapsto$ collection(pairs1, pairs2)
Arc arity	2
Arc constraint(s)	• pairs1.x = pairs2.x • pairs1.y = pairs2.y
Graph property(ies)	**NSCC** = NPAIRS

Graph model Parts (A) and (B) of Figure 5.634 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a pair of values that is assigned to some pairs of variables of the PAIRS collection. In our example we have the following pairs of values: $\langle$x − 3 y − 1$\rangle$ and $\langle$x − 1 y − 5$\rangle$.



(A) **(B)**

Figure 5.634: Initial and final graph of the NPAIR constraint

5.290 NSET_OF_CONSECUTIVE_VALUES

DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint NSET_OF_CONSECUTIVE_VALUES(N, VARIABLES)

Arguments

N : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\mathtt{N} \geq 1$
$\mathtt{N} \leq |\mathtt{VARIABLES}|$
required(VARIABLES, var)

Purpose N is the number of set of consecutive values used by the variables of the collection VARIABLES.

Example

$(2, \langle 3, 1, 7, 1, 1, 2, 8\rangle)$
$(7, \langle 3, 1, 5, 7, 9, 11, 13\rangle)$
$(1, \langle 3, 3, 3, 3, 3, 3, 3\rangle)$

In the first example, the two parts $3, 1, 1, 1, 2$ and $7, 8$ take respectively their values in the following sets of consecutive values $\{1, 2, 3\}$ and $\{7, 8\}$. Consequently, the corresponding NSET_OF_CONSECUTIVE_VALUES constraint holds since its first argument $\mathtt{N} = 2$ is set to the number of sets of consecutive values.

Typical

$\mathtt{N} > 1$
$|\mathtt{VARIABLES}| > 1$
$\mathrm{range}(\mathtt{VARIABLES.var}) > 1$

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

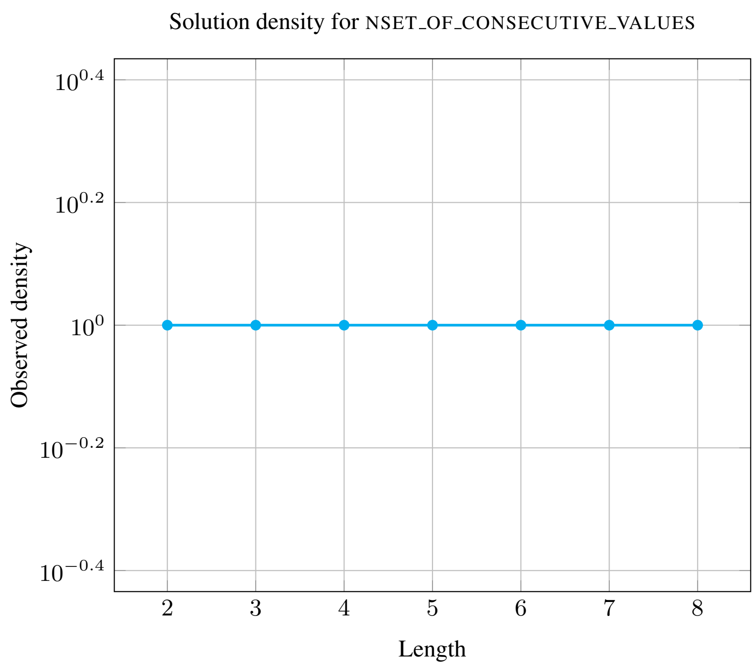
Arg. properties Functional dependency: N determined by VARIABLES.

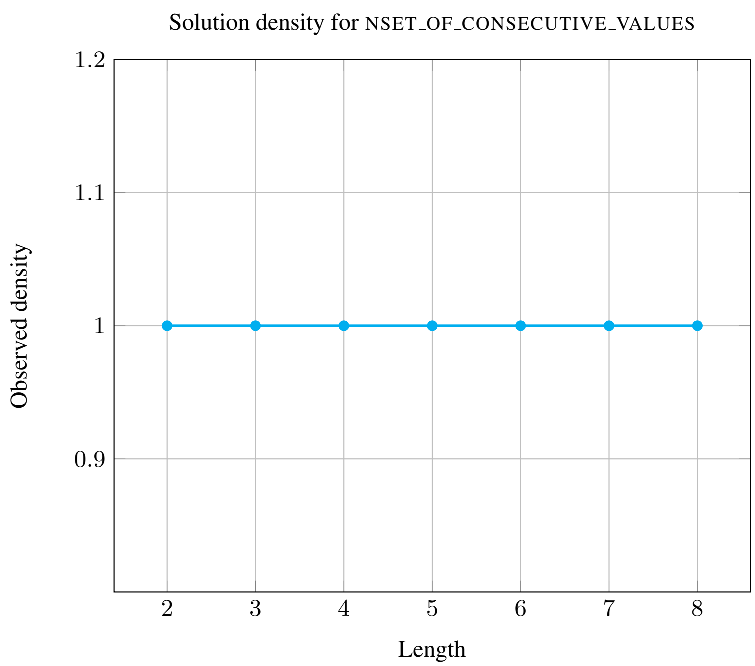
Usage Used for specifying the fact that the values have to be used in a compact way is achieved by setting N to 1.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for NSET_OF_CONSECUTIVE_VALUES: domains $0..n$

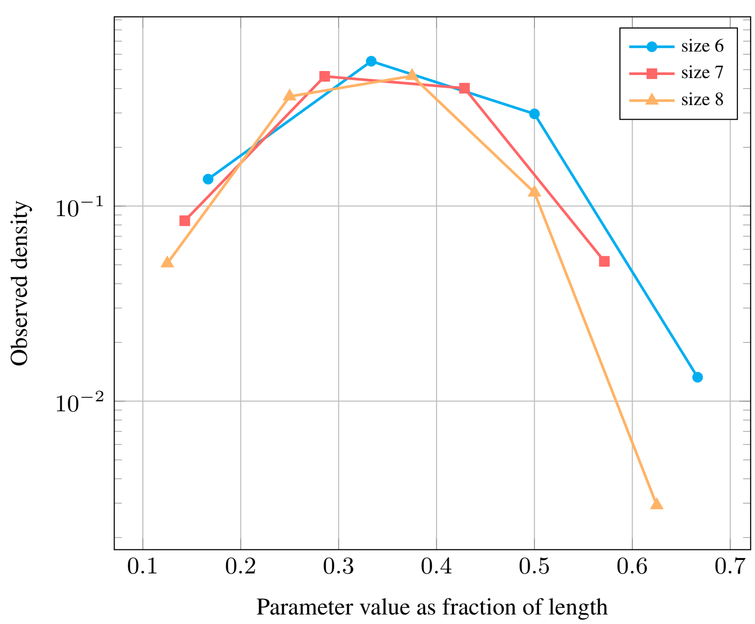

Solution density for NSET_OF_CONSECUTIVE_VALUES
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
Observed density
2 3 4 5 6 7 8
Length

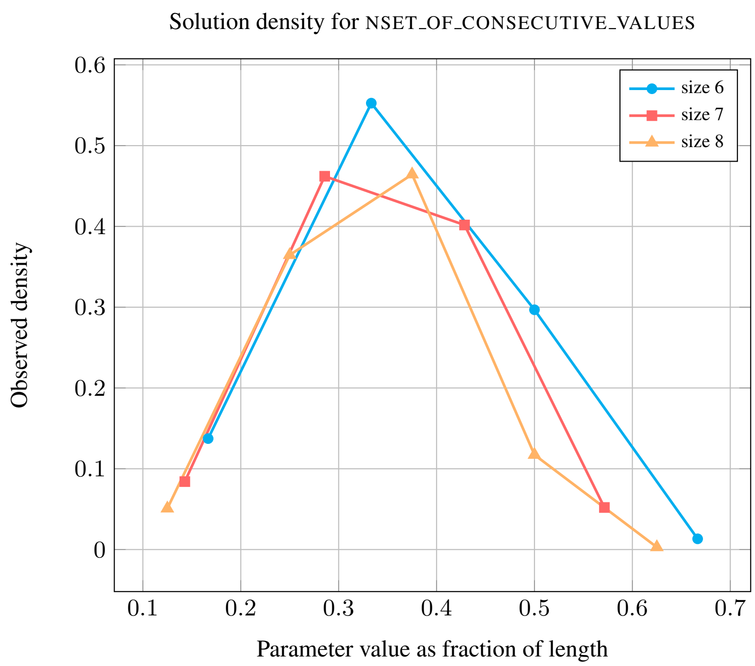

Solution density for NSET_OF_CONSECUTIVE_VALUES
1.2
1.1
1
0.9
Observed density
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	7	34	217	1716	16159	176366	2187637
	2	2	30	372	4740	65010	969066	15695624
	3	-	-	36	1320	34920	842520	19989900
	4	-	-	-	-	1560	109200	5047560
	5	-	-	-	-	-	-	126000

Solution count for NSET_OF_CONSECUTIVE_VALUES: domains $0..n$

Solution density for NSET_OF_CONSECUTIVE_VALUES





See also **common keyword:** MAX_SIZE_SET_OF_CONSECUTIVE_VAR, MIN_SIZE_SET_OF_CONSECUTIVE_VAR *(consecutive values)*.

Keywords **characteristic of a constraint:** consecutive values.

constraint arguments: pure functional dependency.

constraint type: value constraint.

final graph structure: strongly connected component.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	abs(variables1.var − variables2.var) ≤ 1
Graph property(ies)	**NSCC** = N

Graph model Since the arc constraint is symmetric each strongly connected component of the final graph corresponds exactly to one connected component of the final graph.

Parts (A) and (B) of Figure 5.635 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NSCC** graph property, we show the two strongly connected components of the final graph.

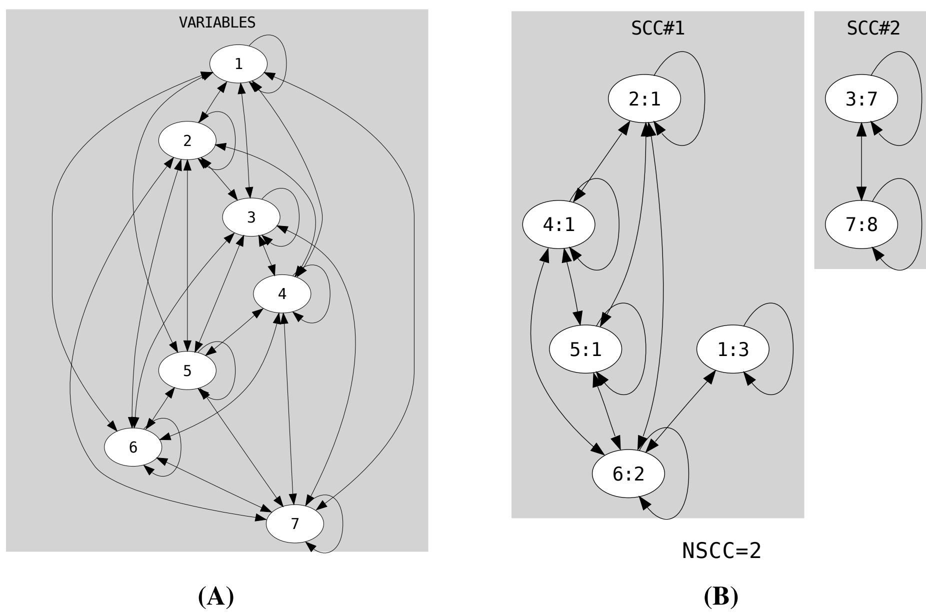


Figure 5.635: Initial and final graph of the NSET_OF_CONSECUTIVE_VALUES constraint

5.291 NUMBER_DIGIT

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic.

Constraint NUMBER_DIGIT(N, VARIABLES, B)

Arguments

N	:	dvar
VARIABLES	:	collection(var−dvar)
B	:	int

Restrictions

$\mathtt{N} \geq 0$
$|\mathtt{VARIABLES}| \geq 1$
$|\mathtt{VARIABLES}| \leq 9$
$\mathtt{VARIABLES.var} \geq 0$
$\mathtt{VARIABLES.var} \leq \mathtt{B} - 1$
$\mathtt{B} \geq 2$
$\mathtt{B} \leq 10$

Purpose Enforce N to be equal to the digits of VARIABLES in base B.

Example $(1234, \langle 1, 2, 3, 4 \rangle, 10)$

The NUMBER_DIGIT constraint holds since 1234 is equal to $1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10 + 4$.

Arg. properties Functional dependency: N determined by VARIABLES and B.

Keywords **constraint arguments:** pure functional dependency.
constraint type: predefined constraint, arithmetic constraint.
modelling: functional dependency.

5.292 NVALUE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [314]

Constraint NVALUE(NVAL, VARIABLES)

Synonyms CARDINALITY_ON_ATTRIBUTES_VALUES, VALUES.

Arguments

NVAL	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

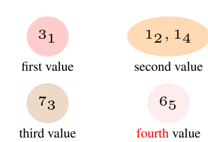
required(VARIABLES, var)
NVAL $\geq$ min(1, |VARIABLES|)
NVAL $\leq$ |VARIABLES|
NVAL $\leq$range(VARIABLES.var)

Purpose NVAL is the number of distinct values taken by the variables of the collection VARIABLES.

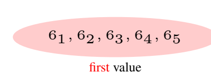
Example

$(4, \langle 3, 1, 7, 1, 6\rangle)$
$(1, \langle 6, 6, 6, 6, 6\rangle)$
$(5, \langle 6, 3, 0, 2, 9\rangle)$

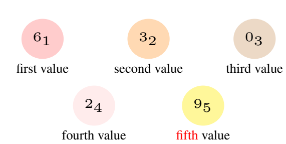
- The first NVALUE constraint holds since its first argument NVAL $=4$ is set to the number of distinct values occurring within the collection $\langle 3, 1, 7, 1, 6\rangle$.



- The second NVALUE constraint holds since its first argument NVAL $=1$ is set to the number of distinct values occurring within the collection $\langle 6, 6, 6, 6, 6\rangle$.



- The third NVALUE constraint holds since its first argument NVAL $=5$ is set to the number of distinct values occurring within the collection $\langle 6, 3, 0, 2, 9\rangle$.



All solutions Figure 5.636 gives all solutions to the following non ground instance of the NVALUE constraint: $\mathtt{N} \in [1, 2]$, $\mathtt{V}_1 \in [2, 4]$, $\mathtt{V}_2 \in [1, 2]$, $\mathtt{V}_3 \in [2, 4]$, NVALUE(N, $\langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3\rangle$).

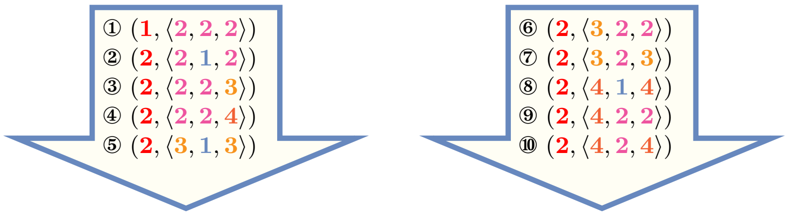


Figure 5.636: All solutions corresponding to the non ground example of the NVALUE constraint of the **All solutions** slot

Typical

`NVAL` > 1
`NVAL` $<$ |`VARIABLES`|
|`VARIABLES`| > 1

Symmetries

- Items of `VARIABLES` are permutable.
- All occurrences of two distinct values of `VARIABLES.var` can be swapped; all occurrences of a value of `VARIABLES.var` can be renamed to any unused value.

Arg. properties

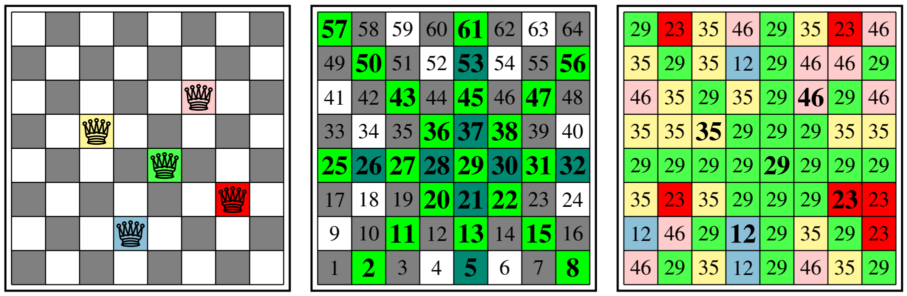
- Functional dependency: `NVAL` determined by `VARIABLES`.
- Contractible wrt. `VARIABLES` when `NVAL` $= 1$ and |`VARIABLES`| > 0.
- Contractible wrt. `VARIABLES` when `NVAL` $=$ |`VARIABLES`|.

Usage

The NVALUE constraint allows relaxing the ALLDIFFERENT constraint by restricting its first argument `NVAL` to be close, but not necessarily equal, to the number of variables of the `VARIABLES` collection.

A classical example from the early 1850s is the *dominating queens* chess puzzle problem: Place a number of queens on an n by n chessboard in such a way that all cells of the chessboard are either attacked by a queen or are occupied by a queen. A queen can attack all cells located on the same column, on the same row or on the same diagonal. Part (A) of Figure 5.637 illustrates a set of five queens which together attack all of the cells of an 8 by 8 chessboard. The *dominating queens* problem can be modelled by just one NVALUE constraint:

- We first label the different cells of the chessboard from 1 to n^2.
- We then associate to each cell c of the chessboard a domain variable. Its initial domain is set to the labels of the cells that can attack cell c. For instance, in the context of an 8 by 8 chessboard, the initial domain of V_{29} will be set to {2,5,8,11,13,15,20..22,25..32,36..38,43,45,47,50,53,56,57,61} (see the green cells of part (B) of Figure 5.637).
- Finally, we post the constraint NVALUE$(Q, \langle \mathtt{var} - V_1, \mathtt{var} - V_2, \ldots, \mathtt{var} - V_{n^2} \rangle)$ where Q is a domain variable in $[1, n^2]$ that gives the total number of queens used for controlling all cells of the chessboard. Note that variable Q should be passed to a minimisation procedure to get the smallest possible number of queens. For the solution depicted by Part (A) of Figure 5.637, the label in each cell of Part (C) of Figure 5.637 gives the value assigned to the corresponding variable. Note that,



(A) (B) (C)

Figure 5.637: Modelling the *dominating queens* problem with a single NVALUE constraint; (A) a solution to the dominating queens problem, (B) the initial domain (in bold) of the variable associated with cell 29: in a solution the value j assigned to the variable associated with cell i represents the label of the cell attacking cell i (i.e. in a solution one of the selected queens is located on cell j), (C) the value of each cell in the model with a single NVALUE constraint corresponding to the solution depicted in (A).

since a given cell can be attacked by several queens, we have also other assignments corresponding to the solution depicted by Part (A) of Figure 5.637.

To conclude note that, since we are only interested to restrict the maximum number of distinct values, we may replace the NVALUE constraint by the ATMOST_NVALUE constraint.

The NVALUE constraint occurs also in many practical applications. In the context of timetabling one wants to set up a limit on the maximum number of activity types it is possible to perform. For frequency allocation problems, one optimisation criterion is to minimise the number of distinct frequencies that you use all over the entire network.

The NVALUE constraint generalises several constraints like:

- ALLDIFFERENT(`VARIABLES`): in order to get the ALLDIFFERENT constraint, one has to set `NVAL` to the total number of variables.
- NOT_ALL_EQUAL(`VARIABLES`): in order to get the NOT_ALL_EQUAL constraint, one has to set the minimum value of `NVAL` to 2.
- ALL_EQUAL(`VARIABLES`): in order to get the ALL_EQUAL constraint, one has to set the maximum value of `NVAL` to 1.

Remark

This constraint appears in [314, page 339] under the name of *Cardinality on Attributes Values*. The NVALUE constraint is called VALUES in **JaCoP** (`http://www.jacop.eu/`). A constraint called K_DIFF enforcing that a set of variables takes at least k distinct values appears in the PhD thesis of J.-C. Régin [352].

It was shown in [75] that, finding out whether a NVALUE constraint has a solution or not is NP-hard. This was achieved by reduction from 3-SAT. In the same article, it is also shown, by reduction from minimum hitting set cardinality, that computing a sharp lower bound on `NVAL` is NP-hard.

Both reformulations of the COLOURED_CUMULATIVE constraint and of the COLOURED_CUMULATIVES constraint use the NVALUE constraint.

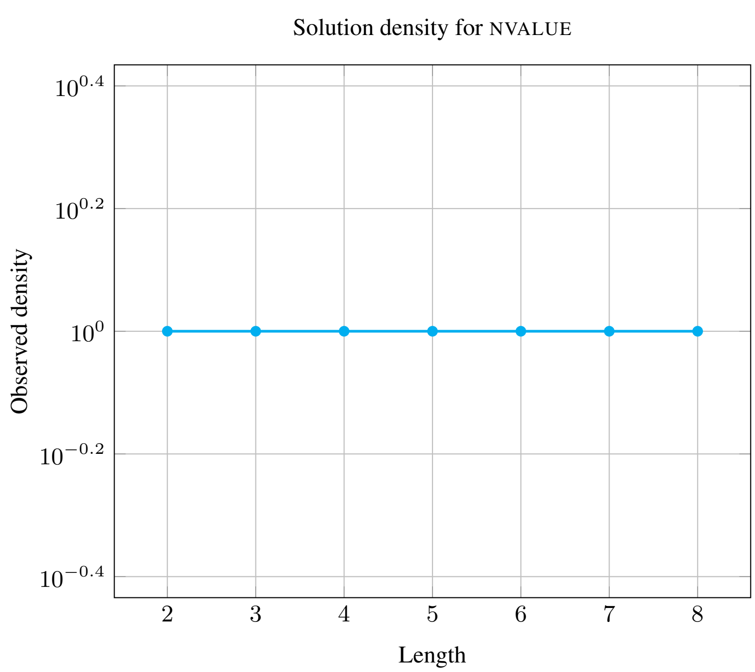
Algorithm A first filtering algorithm for the NVALUE constraint was described in [29]. Assuming that the minimum value of variable NVAL is not constrained at all, two algorithms that both achieve bound-consistency were provided one year later in [46]. Under the same assumption, algorithms that partially take into account holes in the domains of the variables of the VARIABLES collection are described in [46, 68].

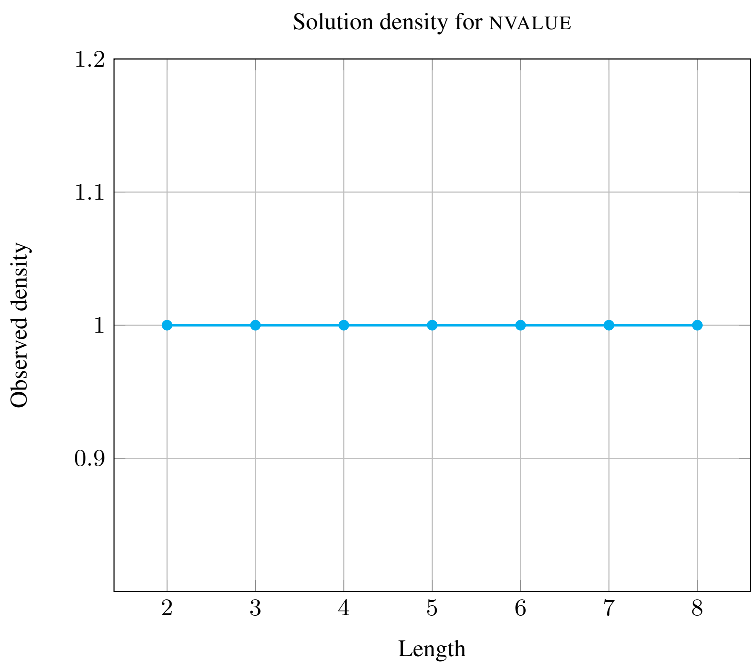
Reformulation A model, involving linear inequalities constraints, preserving bound-consistency was introduced in [79].

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

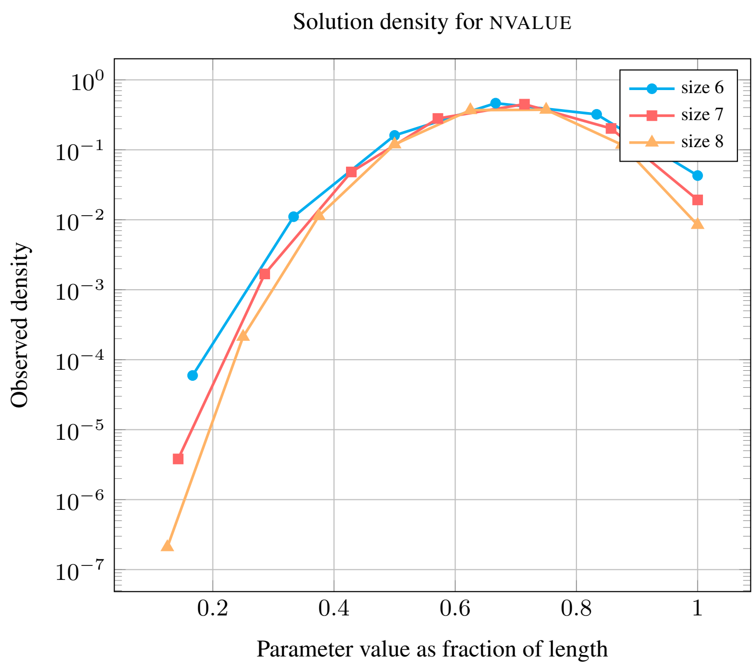
Number of solutions for NVALUE: domains $0..n$

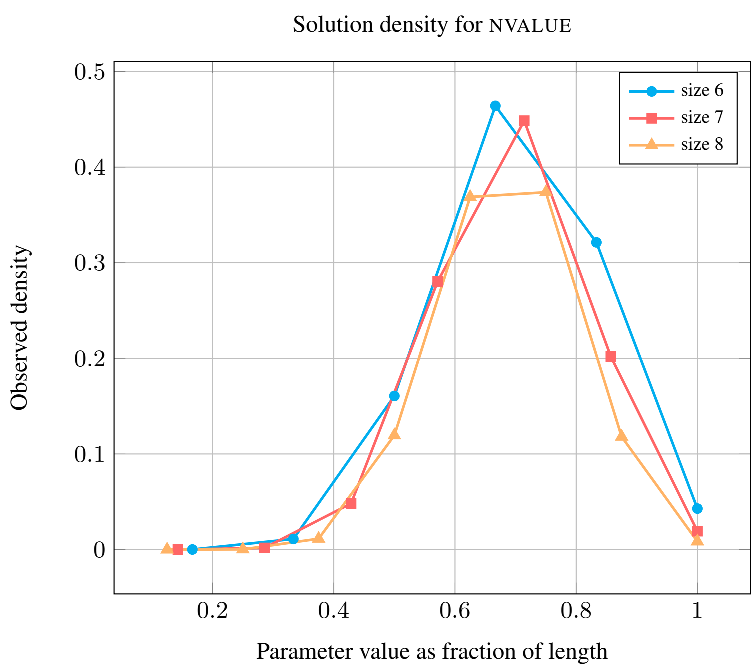




Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	3	4	5	6	7	8	9
	2	6	36	140	450	1302	3528	9144
	3	-	24	360	3000	18900	101136	486864
	4	-	-	120	3600	54600	588000	5143824
	5	-	-	-	720	37800	940800	15876000
	6	-	-	-	-	5040	423360	16087680
	7	-	-	-	-	-	40320	5080320
	8	-	-	-	-	-	-	362880

Solution count for NVALUE: domains $0..n$





Systems NVALUES in **Gecode**, NVALUE in **MiniZinc**, NVALUE in **SICStus**.

Used in TRACK.

See also

assignment dimension added: ASSIGN_AND_NVALUES.

common keyword: AMONG, AMONG_DIFF_0, COUNT, GLOBAL_CARDINALITY, MAX_NVALUE, MIN_NVALUE *(counting constraint)*, NVALUES_EXCEPT_0 *(counting constraint,number of distinct values)*.

cost variant: SUM_OF_WEIGHTS_OF_DISTINCT_VALUES *(introduce a weight for each value and replace number of distinct values by sum of weights associated with distinct values)*.

generalisation: NCLASS *(*`variable` *replaced by* `variable` $\in$ `partition`*)*, NEQUIVALENCE *(*`variable` *replaced by* `variable` mod `constant`*)*, NINTERVAL *(*`variable` *replaced by* `variable`/`constant`*)*, NPAIR *(*`variable` *replaced by* `pair` *of* `variables`*)*, NVALUES *(replace an equality with the number of distinct values by a comparison with the number of distinct values)*, NVECTOR *(variable replaced by vector)*.

implied by: INCREASING_NVALUE.

implies: ATLEAST_NVALUE *(*$=$ `NVAL` *replaced by* $\geq$ `NVAL`*)*, ATMOST_NVALUE *(*$=$ `NVAL` *replaced by* $\leq$ `NVAL`*)*.

related: BALANCE *(restriction on how balanced an assignment is)*, COLOURED_CUMULATIVE *(restrict number of distinct colours on each maximum clique of the interval graph associated with the tasks)*, COLOURED_CUMULATIVES *(restrict number of distinct colours on each maximum clique of the interval graph associated with the tasks assigned to the same machine)*, INCREASING_NVALUE_CHAIN, K_ALLDIFFERENT *(necessary condition for two overlapping* ALLDIFFERENT *constraints)*, SOFT_ALLDIFFERENT_VAR.

shift of concept: NVALUE_ON_INTERSECTION.

soft variant: NVALUES_EXCEPT_0 *(value* 0 *is ignored)*.

specialisation: ALL_EQUAL *(enforce to have one single value)*, ALLDIFFERENT *(enforce a number of distinct values equal to the number of variables)*, NOT_ALL_EQUAL *(enforce to have at least two distinct values)*.

uses in its reformulation: CONSECUTIVE_VALUES, CYCLE, MIN_N.

Keywords

characteristic of a constraint: core, automaton, automaton with array of counters.

complexity: 3-SAT, minimum hitting set cardinality.

constraint arguments: pure functional dependency.

constraint type: counting constraint, value partitioning constraint.

filtering: bound-consistency, convex bipartite graph.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, number of distinct values, functional dependency.

problems: domination.

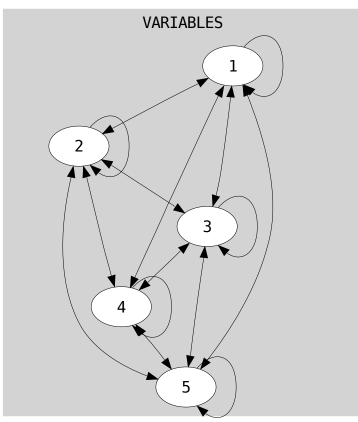
puzzles: dominating queens.

Cond. implications

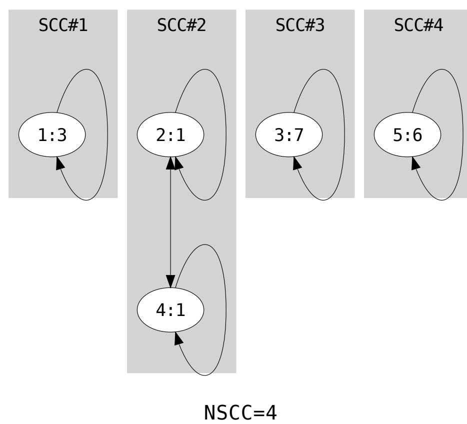
NVALUE(`NVAL`, `VARIABLES`)
with INCREASING(`VARIABLES`)
implies INCREASING_NVALUE(`NVAL`, `VARIABLES`).

Arc input(s)	VARIABLES
Arc generator	*CLIQUE* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSCC** = NVAL
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.638 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The 4 following values 1, 3, 6 and 7 are used by the variables of the VARIABLES collection.



(A)



(B)

Figure 5.638: Initial and final graph of the NVALUE constraint

Automaton Figure 5.639 depicts the automaton associated with the NVALUE constraint. To each item of the collection `VARIABLES` corresponds a signature variable S_i that is equal to 0.

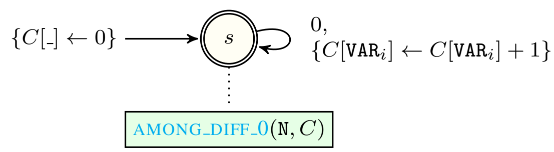


Figure 5.639: Automaton of the NVALUE constraint

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint $\text{NVALUE}(0, \langle 0, 0, 0, 0\rangle)$ hold?

B. Does the constraint $\text{NVALUE}(3, \langle 1, 2, 3\rangle)$ hold?

C. Does the constraint $\text{NVALUE}(3, \langle 1, 2, 3, 3\rangle)$ hold?

[a] Hint: go back to the definition of NVALUE.

EXERCISE 2 (finding all solutions)[a]

Give all the solutions to the constraint:

$$\begin{cases} N \in \{1, 5\}, \\ V_1 \in [3, 5], \quad V_2 \in [3, 4], \quad V_3 \in [2, 5], \\ V_4 \in [3, 3], \quad V_5 \in [3, 4], \quad V_6 \in [3, 7], \\ \text{NVALUE}(N, \langle V_1, V_2, V_3, V_4, V_5, V_6\rangle). \end{cases}$$

[a] Hint: identify the smallest and largest possible values of N, enumerate solutions in lexicographic order.

EXERCISE 3 (identifying infeasible values wrt the at most side)[a]

Identify all variable-value pairs (V_i, val) $(1 \leq i \leq 6)$, such that the following constraint has no solution when variable V_i is assigned value val:

$$\begin{cases} N \in [0, 2], \\ V_1 \in [2, 4], \quad V_2 \in [2, 5], \quad V_3 \in [4, 5], \\ V_4 \in [4, 7], \quad V_5 \in [5, 8], \quad V_6 \in [6, 9], \\ \text{NVALUE}(N, \langle V_1, V_2, V_3, V_4, V_5, V_6\rangle). \end{cases}$$

[a] Hint: are variables equivalent wrt a given value?

EXERCISE 4 (identifying infeasible variable-value pairs wrt the at least side)[a]

Identify all variable-value pairs (V_i, val) $(1 \leq i \leq 6)$, such that the following constraint has no solution when variable V_i is assigned value val:

$$\begin{cases} N \in [5,6], \\ V_1 \in [2,4], \quad V_2 \in [2,3], \quad V_3 \in [4,5], \\ V_4 \in [2,3], \quad V_5 \in [2,3], \quad V_6 \in [5,6], \\ \text{NVALUE}(N, \langle V_1, V_2, V_3, V_4, V_5, V_6 \rangle). \end{cases}$$

[a]Hint: find out how to compute the maximum number of distinct values.

EXERCISE 5 (variable-based degree of violation)[a]

Compute the variable-based degree of violation[b] of the following constraints:

A. $\text{NVALUE}(4, \langle 2, 2, 2, 2 \rangle)$,

B. $\text{NVALUE}(3, \langle 3, 1, 5, 2, 3 \rangle)$.

[a]Hint: take advantage of the functional dependency.

[b]Given a constraint for which all variables are fixed, the *variable-based degree of violation* is the minimum number of variables to assign differently in order to satisfy the constraint.

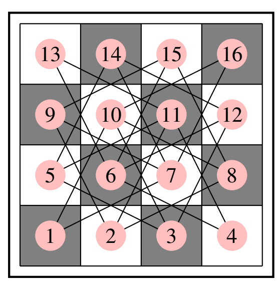
EXERCISE 6 (variations of dominating knights)[a]

A. Provide a model involving only one NVALUE constraint for showing that the cardinality of the dominating set[b] of the knight graph of a 4 by 4 chessboard does not exceed 4.

B. Show how to modify your model for also considering the fact that each knight must be protected by at least one other knight. Show that the number of required knights does not exceed 6.

[a]Hint: model the knight graph with a set of variables; in a domination problem whats matters for each vertex v is which vertices attack v.

[b]Given a graph G a *dominating set* $\mathcal{D}$ is a subset of the vertices of G such that every vertex of G either belongs to $\mathcal{D}$ or is adjacent to a vertex of G.



4×4 knight graph

SOLUTION TO EXERCISE 1

A. *No, since* $\langle 0,0,0,0\rangle$ *contains just one distinct value (and not* 0 *distinct values as stated by the first argument).*

B. *Yes, since* $\langle 1,2,3\rangle$ *contains* 3 *distinct values as stated by the first argument.*

C. *Yes, since* $\langle 1,2,3,3\rangle$ *contains* 3 *distinct values as stated by the the first argument.*

SOLUTION TO EXERCISE 2

the seven solutions

$N, \langle V_1, V_2, V_3, V_4, V_5, V_6\rangle$

① $(1, \langle 3,3,3,3,3,3\rangle)$
② $(5, \langle 5,3,2,3,4,6\rangle)$
③ $(5, \langle 5,3,2,3,4,7\rangle)$
④ $(5, \langle 5,4,2,3,3,6\rangle)$
⑤ $(5, \langle 5,4,2,3,3,7\rangle)$
⑥ $(5, \langle 5,4,2,3,4,6\rangle)$
⑦ $(5, \langle 5,4,2,3,4,7\rangle)$

A. *Value* 3 *being the only common value to variables* V_1, V_2, V_3, V_4, V_5, V_6, *we get a single solution where* N *is set to* 1, *i.e. solution ①.*

B. *A matching of cardinality* 5 *is given by* $V_1=5$, $V_2=3$, $V_3=2$, $V_4=3$, $V_5=4$, $V_6=6$. *It is maximum since variables* V_1, V_2, V_3, V_4, V_5 *have to be assigned one of the four values* 2, 3, 4 *and* 5, *and since values* 6 *and* 7 *can only be assigned to variable* V_6. *In any maximum matching we have that:*

(a) *Since variable* V_6 *is the only variable that can be assigned values* 6 *or* 7, *we have* $V_6=6$ *or* $V_6=7$.

(b) *Since variable* V_3 *is the only variable that can be assigned value* 2, *we have* $V_3=2$.

(c) *Now that* V_3 *is assigned value* 2 *and that* V_6 *is assigned values* 6 *or* 7, *variable* V_1 *is the only variable that can be assigned value* 5, *we have* $V_1=5$.

Finally combining the fact that variables V_2, V_5 *have to be assigned a distinct value in* $\{3,4\}$ *and variable* V_6 *a value in* $\{6,7\}$ *we obtain the remaining six solutions ②, ③, ④, ⑤, ⑥, ⑦.*

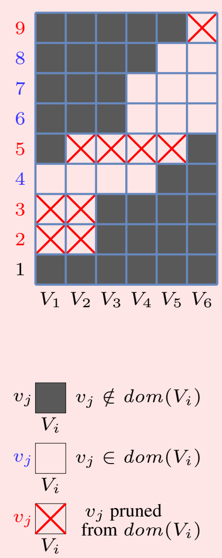
SOLUTION TO EXERCISE 3

The constraint forces that at most two distinct values are assigned to variables $V_1, V_2, \ldots, V_6$, i.e. there is no restriction coming from N on the minimum number of distinct values. In this context, a value val assigned to one of the variables V_i $(1 \leq i \leq 6)$ can be assigned to any other variable V_i without increasing the number of distinct values. Consequently a value val that is not removed (resp. removed) from a variable V_i $(1 \leq i \leq 6)$ can also not be removed (resp. removed) from a variable V_j $(j \neq i, 1 \leq j \leq 6)$. Let us successively study the values that can not be removed and the values that can be removed from $V_1, V_2, \ldots, V_6$.

A. [*FEASIBLE VALUES*]

Consider the three solutions
NVALUE$(2, \langle 4, 4, 4, 4, 6, 6\rangle)$,
NVALUE$(2, \langle 4, 4, 4, 4, 7, 7\rangle)$,
NVALUE$(2, \langle 4, 4, 4, 4, 8, 8\rangle)$.

All values used in the previous solutions (i.e., values 4, 6, 7, 8) can not be removed from $V_1, V_2, \ldots, V_6$.



B. [*INFEASIBLE VALUES*]

We now show that 2 cannot be assigned to any variable. If 2 can be used then we assign 2 to all variables that have 2 in their domains, i.e., V_1 and V_2. Now in order not to exceed two distinct values, the remaining variables V_3, V_4, V_5, V_6 must have a value in common, which is not the case. We can show in the same way that values 3, 5 and 9 cannot be assigned to any variable.

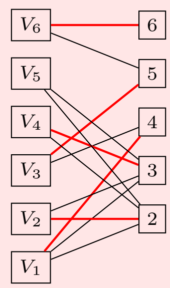
Finally since $V_1, V_2, \ldots, V_6$ do not have any value in common, N can only be equal to 2.

SOLUTION TO EXERCISE 4

The constraint forces that at least five distinct values are assigned to variables $V_1, V_2, \ldots, V_6$, i.e. there is no restriction coming from N on the maximum number of distinct values. Consequently identifying infeasible variable-value pairs is equivalent to finding edges that do not belong to any matching[a] *of cardinality greater than or equal to the minimum value of N in the variable-value bipartite graph $\mathcal{G}(V, E)$ associated with the* NVALUE *constraint (the vertices V of $\mathcal{G}$ are defined by the variables $V_1, V_2, \ldots, V_6$ and by the values $2, 3, \ldots, 6$, while the edges E are defined by the pairs (V_i, val), $(1 \leq i \leq 6)$ such that $val \in dom(V_i)$).*

A. [*MAXIMUM MATCHING*]

The solution nvalue$(5, \langle 4, 2, 5, 3, 2, 6\rangle)$ corresponds to a matching of cardinality 5 shown in red on the variable-value graph. This matching is maximum since $|dom(V_1) \cup dom(V_2) \cup \cdots \cup dom(V_6)| = 5$. Therefore N can only be equal to 5.

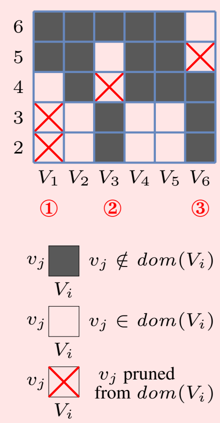


B. [*INFEASIBLE EDGES*]

① *Since $|dom(V_2) \cup dom(V_4)| = 2$, V_1 must be assigned value 4 in any maximum matching.*

② *Since V_1 must be assigned value 4 in any maximum matching and since $dom(V_3) = \{4, 5\}$, V_3 must be assigned value 5 in any maximum matching.*

③ *Since V_3 must be assigned value 5 in any maximum matching and since $dom(V_6) = \{5, 6\}$, V_6 must be assigned value 6 in any maximum matching.*



C. *Finally, V_2, V_4, V_5 must be assigned two distinct values from $\{2, 3\}$ in any maximum matching.*

---

[a] A *matching* of a graph $\mathcal{G}$ is a set of edges of $\mathcal{G}$ such that no two edges have a vertex in common.

SOLUTION TO EXERCISE 5

For a violated NVALUE *constraint it is always possible to change a single variable to get a feasible solution. This is done by setting the first argument of the* NVALUE *constraint to the number of distinct values occurring in the second argument.*

A. *The degree of violation is equal to* 1 *since the first argument needs to be set to* 1 *in order to obtain a solution.*

$$\text{NVALUE}(\overset{1}{\mathbf{4}}, \langle 2, 2, 2, 2\rangle)$$

B. *The degree of violation is equal to* 1 *since the first argument needs to be set to* 4 *in order to obtain a solution.*

$$\text{NVALUE}(\overset{4}{\mathbf{3}}, \langle 3, 1, 5, 2, 3\rangle)$$

Note that in this example we have other possibilities such as

$$\text{NVALUE}(3, \langle 3, 1, \overset{1}{\mathbf{5}}, 2, 3\rangle)$$

SOLUTION TO EXERCISE 6

A. *Each vertex of the* 4×4 *knight graph is represented by a variable whose domain is set to the labels of its adjacent vertices as well as to its own label. Consequently we get the following* 16 *variables with their corresponding initial domains:*

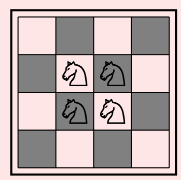
$V_1 \in \{1,7,10\}$, $V_2 \in \{2,8,9,11\}$, $V_3 \in \{3,5,10,12\}$,
$V_4 \in \{4,6,11\}$, $V_5 \in \{3,5,11,14\}$, $V_6 \in \{4,6,12,13,15\}$,
$V_7 \in \{1,7,9,14,16\}$, $V_8 \in \{2,8,10,15\}$, $V_9 \in \{2,7,9,15\}$,
$V_{10} \in \{1,3,8,10,16\}$, $V_{11} \in \{2,4,5,11,13\}$, $V_{12} \in \{3,6,12,14\}$,
$V_{13} \in \{6,11,13\}$, $V_{14} \in \{5,7,12,14\}$, $V_{15} \in \{6,8,9,15\}$,
$V_{16} \in \{7,10,16\}$.

We introduce a variable $N \in \{1,2,3,4\}$ *that provides the number of knights actually used and state the following constraint:*

$$\text{nvalue}(N, \langle V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16} \rangle)$$

In the previous constraint the assignment $V_i = j$ *means that a knight is located on vertex* j *and that vertex* j *attacks vertex* i. *Consequently the total number of distinct values in* $\langle V_1, V_2, \ldots, V_{16} \rangle$ *is equal to the total number of used knights. The assignment*

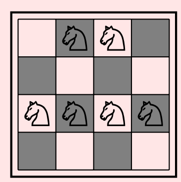
$N = 4$,
$V_1 = 7$, $V_2 = 11$, $V_3 = 10$, $V_4 = 6$,
$V_5 = 11$, $V_6 = 6$, $V_7 = 7$, $V_8 = 10$,
$V_9 = 7$, $V_{10} = 10$, $V_{11} = 11$, $V_{12} = 6$,
$V_{13} = 6$, $V_{14} = 7$, $V_{15} = 6$, $V_{16} = 7$



corresponds to the solution depicted on the right.

B. *Since a knight cannot protect itself, we only need to remove from the initial domain of each variable the label corresponding to its cell. The assignment*

$N = 6$,
$V_1 = 7$, $V_2 = 8$, $V_3 = 5$, $V_4 = 6$,
$V_5 = 14$, $V_6 = 15$, $V_7 = 14$, $V_8 = 15$,
$V_9 = 7$, $V_{10} = 8$, $V_{11} = 5$, $V_{12} = 6$,
$V_{13} = 6$, $V_{14} = 5$, $V_{15} = 6$, $V_{16} = 7$



corresponds to the solution depicted on the right.

5.293 NVALUE_ON_INTERSECTION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from COMMON and NVALUE.

Constraint NVALUE_ON_INTERSECTION(NVAL, VARIABLES1, VARIABLES2)

Arguments

NVAL	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)

Restrictions

required(VARIABLES1, var)
required(VARIABLES2, var)
NVAL ≥ 0
NVAL $\leq$ |VARIABLES1|
NVAL $\leq$ |VARIABLES2|
NVAL $\leq$range(VARIABLES1.var)
NVAL $\leq$range(VARIABLES2.var)

Purpose NVAL is the number of distinct values that both occur in the VARIABLES1 and VARIABLES2 collections.

Example $(2, \langle 1, 9, 1, 5 \rangle, \langle 2, 1, 9, 9, 6, 9 \rangle)$

Note that the two collections $\langle 1, 9, 1, 5 \rangle$ and $\langle 2, 1, 9, 9, 6, 9 \rangle$ share two values in common (i.e., values 1 and 9). Consequently the NVALUE_ON_INTERSECTION constraint holds since its first argument NVAL is set to 2.

Typical

NVAL > 0
NVAL $<$ |VARIABLES1|
NVAL $<$ |VARIABLES2|
NVAL $<$range(VARIABLES1.var)
NVAL $<$range(VARIABLES2.var)
|VARIABLES1| > 1
|VARIABLES2| > 1

Symmetries

- Arguments are permutable w.r.t. permutation (NVAL) (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Arg. properties

- Functional dependency: NVAL determined by VARIABLES1 and VARIABLES2.
- Contractible wrt. VARIABLES1 when NVAL $= 0$.
- Contractible wrt. VARIABLES2 when NVAL $= 0$.

See also

common keyword: ALLDIFFERENT_ON_INTERSECTION, COMMON, SAME_INTERSECTION *(constraint on the intersection)*.

root concept: NVALUE.

Keywords

constraint arguments: pure functional dependency.

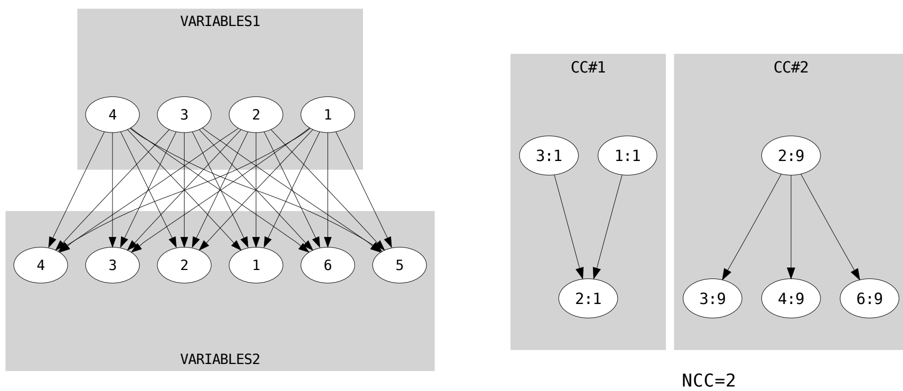
constraint type: counting constraint, constraint on the intersection.

final graph structure: connected component.

modelling: number of distinct values, functional dependency.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	$\mathbf{NCC}$ = NVAL

Graph model Parts (A) and (B) of Figure 5.640 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NCC}$ graph property we show the connected components of the final graph. The variable NVAL is equal to this number of connected components. Note that all the vertices corresponding to the variables that take values 5, 2 or 6 were removed from the final graph since there is no arc for which the associated equality constraint holds.



(A) **(B)**

Figure 5.640: Initial and final graph of the NVALUE_ON_INTERSECTION constraint

5.294 NVALUES

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Inspired by NVALUE and COUNT.

Constraint NVALUES(VARIABLES, RELOP, LIMIT)

Arguments

VARIABLES : collection(var−dvar)
RELOP : atom
LIMIT : dvar

Restrictions

required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Let N be the number of distinct values assigned to the variables of the VARIABLES collection. Enforce condition N RELOP LIMIT to hold.

Example $(\langle 4, 5, 5, 4, 1, 5\rangle, =, 3)$

The NVALUES constraint holds since the number of distinct values occurring within the collection $\langle 4, 5, 5, 4, 1, 5\rangle$ is equal (i.e., RELOP is set to $=$) to its third argument LIMIT $= 3$.

Typical

$|$VARIABLES$| > 1$
LIMIT > 1
LIMIT $<$ $|$VARIABLES$|$
RELOP $\in [=, <, \geq, >, \leq]$

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties

- Contractible wrt. VARIABLES when RELOP $\in [<, \leq]$.
- Contractible wrt. VARIABLES when RELOP $\in [=]$, LIMIT $= 1$ and $|$VARIABLES$| > 0$.
- Contractible wrt. VARIABLES when RELOP $\in [=]$ and LIMIT $=$ $|$VARIABLES$|$.
- Extensible wrt. VARIABLES when RELOP $\in [\geq, >]$.

Usage Used in the **Constraint(s) on sets** slot for defining some constraints like ASSIGN_AND_NVALUES, CIRCUIT_CLUSTER or COLOURED_CUMULATIVE.

Reformulation The NVALUES(VARIABLES, RELOP , LIMIT) constraint can be expressed in term of the conjunction NVALUE(NV, VARIABLES) $\wedge$ NV RELOP LIMIT.

Systems NVALUES in **Gecode**.

Used in ASSIGN_AND_NVALUES, CIRCUIT_CLUSTER, COLOURED_CUMULATIVE, COLOURED_CUMULATIVES.

See also **assignment dimension added:** ASSIGN_AND_NVALUES.

common keyword: NVALUES_EXCEPT_0 *(counting constraint,number of distinct values)*.

specialisation: NVALUE *(replace a comparison with the number of distinct values by an equality with the number of distinct values)*.

Keywords **constraint type:** counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

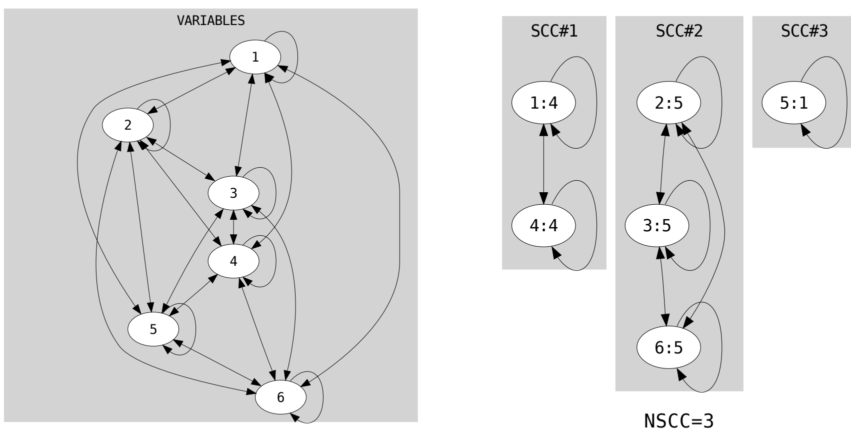
modelling: number of distinct equivalence classes, number of distinct values.

problems: domination.

Cond. implications NVALUES(VARIABLES, RELOP, LIMIT)
with minval(VARIABLES.var) > 0
implies NVALUES_EXCEPT_0(VARIABLES, RELOP, LIMIT).

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*↦collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSCC** RELOP LIMIT
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.641 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value that is assigned to some variables of the VARIABLES collection. The 3 following values 1, 4 and 5 are used by the variables of the VARIABLES collection.



(A) **(B)**

Figure 5.641: Initial and final graph of the NVALUES constraint

5.295 NVALUES_EXCEPT_0

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from NVALUES.

Constraint NVALUES_EXCEPT_0(VARIABLES, RELOP, LIMIT)

Arguments

VARIABLES	:	collection(var−dvar)
RELOP	:	atom
LIMIT	:	dvar

Restrictions

required(VARIABLES, var)
RELOP $\in [=, \neq, <, \geq, >, \leq]$

Purpose Let N be the number of distinct values, different from 0, assigned to the variables of the VARIABLES collection. Enforce condition N RELOP LIMIT to hold.

Example $(\langle 4, 5, 5, 4, 0, 1\rangle, =, 3)$

The NVALUES_EXCEPT_0 constraint holds since the number of distinct values, different from 0, occurring within the collection $\langle 4, 5, 5, 4, 0, 1\rangle$ is equal (i.e., RELOP is set to $=$) to its third argument LIMIT $= 3$.

Typical

$|\mathtt{VARIABLES}| > 1$
LIMIT > 1
LIMIT $< |\mathtt{VARIABLES}|$
ATLEAST(1, VARIABLES, 0)
RELOP $\in [=, <, \geq, >, \leq]$

Typical model ATLEAST(2, VARIABLES, 0)

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var that are both different from 0 can be swapped; all occurrences of a value of VARIABLES.var that is different from 0 can be renamed to any unused value that is also different from 0.

Arg. properties

- Contractible wrt. VARIABLES when RELOP $\in [<, \leq]$.
- Extensible wrt. VARIABLES when RELOP $\in [\geq, >]$.

Reformulation The NVALUES_EXCEPT_0($\langle V_1, V_2, \ldots, V_{|\mathtt{VARIABLES}|}\rangle$, RELOP , LIMIT) constraint can be expressed in term of the conjunction NVALUE($NV1$, $\langle 0, V_1, V_2, \ldots, V_{|\mathtt{VARIABLES}|}\rangle$) $\wedge$ $NV1 - 1$ RELOP LIMIT.

Used in CYCLE_OR_ACCESSIBILITY.

See also **common keyword:** ASSIGN_AND_NVALUES *(number of distinct values)*, NVALUE, NVALUES *(counting constraint,number of distinct values)*.

Keywords **characteristic of a constraint:** joker value.

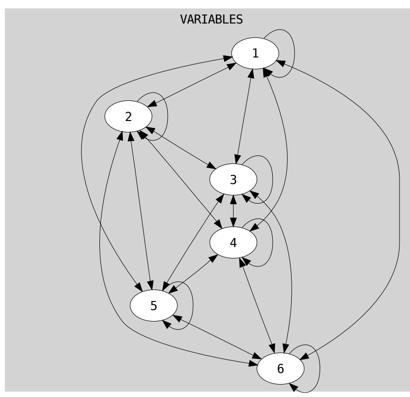
constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component.

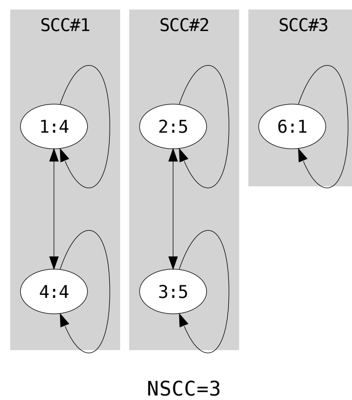
modelling: number of distinct values.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• variables1.var $\neq 0$ • variables1.var $=$ variables2.var
Graph property(ies)	**NSCC** RELOP LIMIT

Graph model Parts (A) and (B) of Figure 5.642 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a value distinct from 0 that is assigned to some variables of the VARIABLES collection. Beside value 0, the 3 following values 1, 4 and 5 are assigned to the variables of the VARIABLES collection.



(A)



(B)

Figure 5.642: Initial and final graph of the NVALUES_EXCEPT_0 constraint

5.296 NVECTOR

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Introduced by G. Chabert as a generalisation of NVALUE

Constraint NVECTOR(NVEC, VECTORS)

Synonyms NVECTORS, NPOINT, NPOINTS.

Type

VECTOR : collection(var−dvar)

Arguments

NVEC : dvar
VECTORS : collection(vec − VECTOR)

Restrictions

$|\text{VECTOR}| \geq 1$
$\text{NVEC} \geq \min(1, |\text{VECTORS}|)$
$\text{NVEC} \leq |\text{VECTORS}|$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose

NVEC is the number of distinct tuples of values taken by the vectors of the collection VECTORS. Two tuples of values $\langle A_1, A_2, \ldots, A_m\rangle$ and $\langle B_1, B_2, \ldots, B_m\rangle$ are *distinct* if and only if there exist an integer $i \in [1, m]$ such that $A_i \neq B_i$.

Example

$$\left(2, \left\langle \begin{array}{l} \text{vec} - \langle 5,6\rangle, \\ \text{vec} - \langle 5,6\rangle, \\ \text{vec} - \langle 9,3\rangle, \\ \text{vec} - \langle 5,6\rangle, \\ \text{vec} - \langle 9,3\rangle \end{array} \right\rangle \right)$$

The NVECTOR constraint holds since its first argument NVEC = 2 is set to the number of distinct tuples of values (i.e., tuples $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$) occurring within the collection VECTORS. Figure 5.643 depicts with a thick rectangle a possible initial domain for each of the five vectors and with a grey circle each tuple of values of the corresponding solution.

Typical

$|\text{VECTOR}| > 1$
$\text{NVEC} > 1$
$\text{NVEC} < |\text{VECTORS}|$
$|\text{VECTORS}| > 1$

Symmetries

- Items of VECTORS are permutable.
- Items of VECTORS.vec are permutable (*same permutation used*).
- All occurrences of two distinct tuples of values of VECTORS.vec can be swapped; all occurrences of a tuple of values of VECTORS.vec can be renamed to any unused tuple of values.

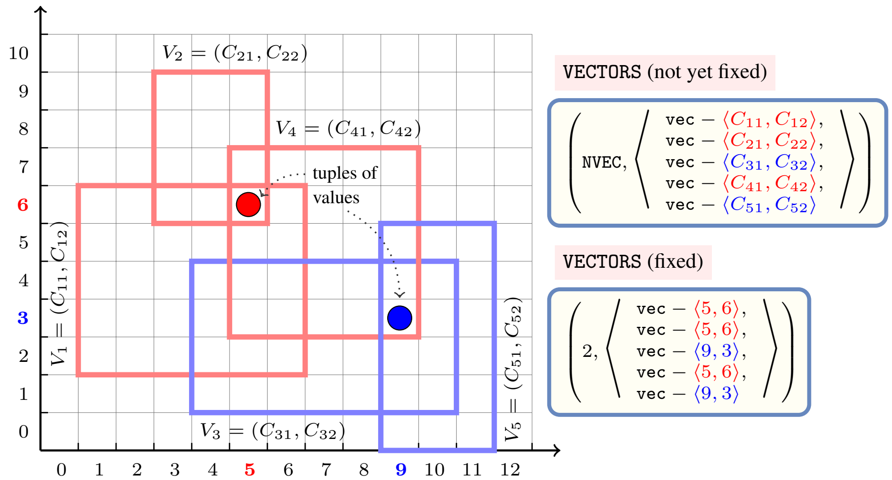


Figure 5.643: Possible initial domains ($C_{11} \in [1,6]$, $C_{12} \in [2,6]$, $C_{21} \in [3,5]$, $C_{22} \in [6,9]$, $C_{31} \in [4,10]$, $C_{32} \in [1,4]$, $C_{41} \in [5,9]$, $C_{42} \in [3,7]$, $C_{51} \in [9,11]$, $C_{52} \in [0,5]$) and solution corresponding to the **Example** slot: we have two distinct vectors (NVEC $= 2$)

Arg. properties

- Functional dependency: NVEC determined by VECTORS.
- Contractible wrt. VECTORS when NVEC $= 1$ and $|$VECTORS$| > 0$.
- Contractible wrt. VECTORS when NVEC $= |$VECTORS$|$.

Remark

It was shown in [118, 117] that, finding out whether a NVECTOR constraint has a solution or not is NP-hard (i.e., the restriction to the rectangle case and to the atmost side of the NVECTOR were considered for this purpose). This was achieved by reduction from the rectangle clique partition problem.

Reformulation

Assume the collection VECTORS is not empty (otherwise NVEC $= 0$). In this context, let n and m respectively denote the number of vectors of the collection VECTORS and the number of components of each vector. Furthermore, let $\alpha_i = \min(\underline{C_{1i}}, \underline{C_{2i}}, \ldots, \underline{C_{ni}})$, $\beta_i = \max(\overline{C_{1i}}, \overline{C_{2i}}, \ldots, \overline{C_{ni}})$, $\gamma_i = \beta_i - \alpha_i + 1$, $(i \in [1, m])$. By associating to each vector

$$\langle C_{k1}, C_{k2}, \ldots, C_{km}\rangle, \quad (k \in [1, n])$$

a variable

$$D_k = \sum_{1 \le i \le m} \left(\left(\prod_{i < j \le m} \gamma_j \right) \cdot (C_{ki} - \alpha_i) \right),$$

the constraint

NVECTOR(NVEC,
⟨vec − ⟨$C_{11}, C_{12}, \ldots, C_{1m}$⟩,
vec − ⟨$C_{21}, C_{22}, \ldots, C_{2m}$⟩,
....
vec − ⟨$C_{n1}, C_{n2}, \ldots, C_{nm}$⟩⟩)

can be expressed in term of the constraint
NVALUE(NVEC, $\langle D_1, D_2, \ldots, D_n\rangle$).

Note that the previous reformulation does not work anymore if the variables have a continuous domain, or if an overflow occurs while propagating the equality constraint $D_k = \sum_{1\leq i\leq m}\left(\left(\prod_{i<j\leq m}\gamma_j\right)\cdot(C_{ki}-\alpha_i)\right)$ (i.e., the number of components m is too big).

When using this reformulation with respect to the **Example** slot we first introduce $D_1 = 1\cdot 6-3+(4\cdot 5-20)) = 3$, $D_2 = 1\cdot 6-3+(4\cdot 5-20)) = 3$, $D_3 = 1\cdot 3-3+(4\cdot 9-20)) = 16$, $D_4 = 1\cdot 6-3+(4\cdot 5-20)) = 3$, $D_5 = 1\cdot 3-3+(4\cdot 9-20)) = 16$ and then get the constraint NVALUE$(2, \langle 3, 3, 16, 3, 16\rangle)$.

See also

common keyword: LEX_EQUAL, ORDERED_ATLEAST_NVECTOR, ORDERED_ATMOST_NVECTOR *(vector)*.

generalisation: NVECTORS *(replace an equality with the number of distinct vectors by a comparison with the number of distinct nvectors)*.

implied by: ORDERED_NVECTOR.

implies: ATLEAST_NVECTOR *($=$ NVEC replaced by $\geq$ NVEC)*, ATMOST_NVECTOR *($=$ NVEC replaced by $\leq$ NVEC)*.

specialisation: NVALUE *(vector replaced by* variable*)*.

Keywords

application area: SLAM problem.

characteristic of a constraint: vector.

complexity: rectangle clique partition.

constraint arguments: pure functional dependency.

constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes, functional dependency.

problems: domination.

Arc input(s)	VECTORS
Arc generator	$CLIQUE \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_EQUAL(vectors1.vec, vectors2.vec)
Graph property(ies)	**NSCC** = NVEC
Graph class	EQUIVALENCE

Graph model

Parts (A) and (B) of Figure 5.644 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 2 following tuple of values $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$ are used by the vectors of the VECTORS collection.

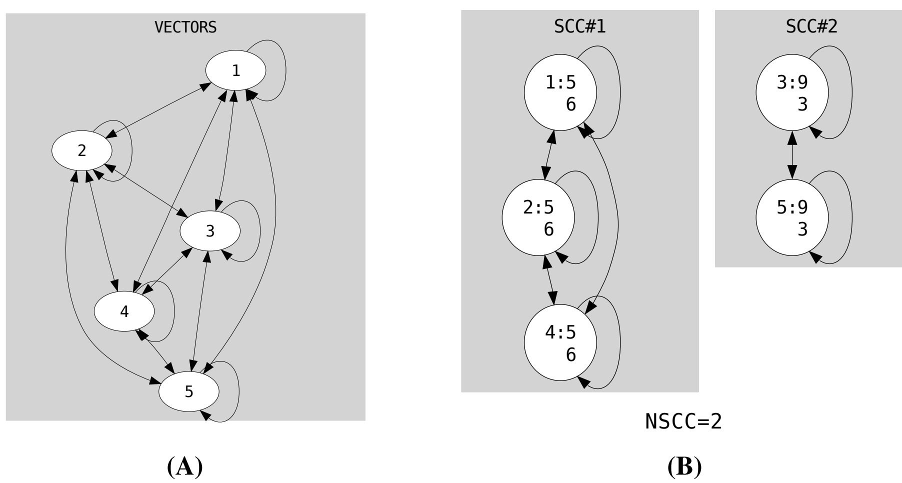


Figure 5.644: Initial and final graph of the NVECTOR constraint

5.297 NVECTORS

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin

Inspired by NVECTOR and COUNT.

Constraint

NVECTORS(VECTORS, RELOP, LIMIT)

Synonym

NPOINTS.

Type

VECTOR : collection(var−dvar)

Arguments

VECTORS : collection(vec − VECTOR)
RELOP : atom
LIMIT : dvar

Restrictions

$|\text{VECTOR}| \geq 1$
required(VECTORS, vec)
same_size(VECTORS, vec)
$\text{RELOP} \in [=, \neq, <, \geq, >, \leq]$

Purpose

Let N be the number of distinct tuples of values taken by the vectors of the VECTORS collection. Enforce condition N RELOP LIMIT to hold.

Example

$$\left(\left\langle\begin{array}{l}\text{vec} - \langle 5,6\rangle,\\ \text{vec} - \langle 5,6\rangle,\\ \text{vec} - \langle 9,3\rangle,\\ \text{vec} - \langle 5,6\rangle,\\ \text{vec} - \langle 9,3\rangle\end{array}\right\rangle, =, 2\right)$$

The NVECTORS constraint holds since the number of distinct tuples of values (i.e., tuples $\langle 5,6\rangle$ and $\langle 9,3\rangle$) occurring within the collection VECTORS is equal (i.e., RELOP is set to $=$) to its third argument $\text{LIMIT} = 2$.

Typical

$|\text{VECTOR}| > 1$
$|\text{VECTORS}| > 1$
$\text{RELOP} \in [=, <, \geq, >, \leq]$
$\text{LIMIT} > 1$
$\text{LIMIT} < |\text{VECTORS}|$

Symmetries

- Items of VECTORS are permutable.
- Items of VECTORS.vec are permutable (*same permutation used*).
- All occurrences of two distinct values of VECTORS.vec can be swapped; all occurrences of a value of VECTORS.vec can be renamed to any unused value.

Arg. properties

- Contractible wrt. VECTORS when RELOP $\in [<, \leq]$.
- Extensible wrt. VECTORS when RELOP $\in [\geq, >]$.

Reformulation

The NVECTORS(VECTORS, RELOP , LIMIT) constraint can be expressed in term of the conjunction NVECTOR(NV, VECTORS) $\wedge$ NV RELOP LIMIT.

See also

specialisation: NVECTOR *(replace a comparison with the number of distinct vectors by an equality with the number of distinct vectors)*.

Keywords

characteristic of a constraint: vector.

constraint type: counting constraint, value partitioning constraint.

final graph structure: strongly connected component, equivalence.

modelling: number of distinct equivalence classes.

problems: domination.

Arc input(s)	VECTORS
Arc generator	$CLIQUE \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_EQUAL(vectors1.vec, vectors2.vec)
Graph property(ies)	**NSCC** RELOP LIMIT
Graph class	EQUIVALENCE

Graph model Parts (A) and (B) of Figure 5.645 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 2 following tuple of values $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$ are used by the vectors of the VECTORS collection.

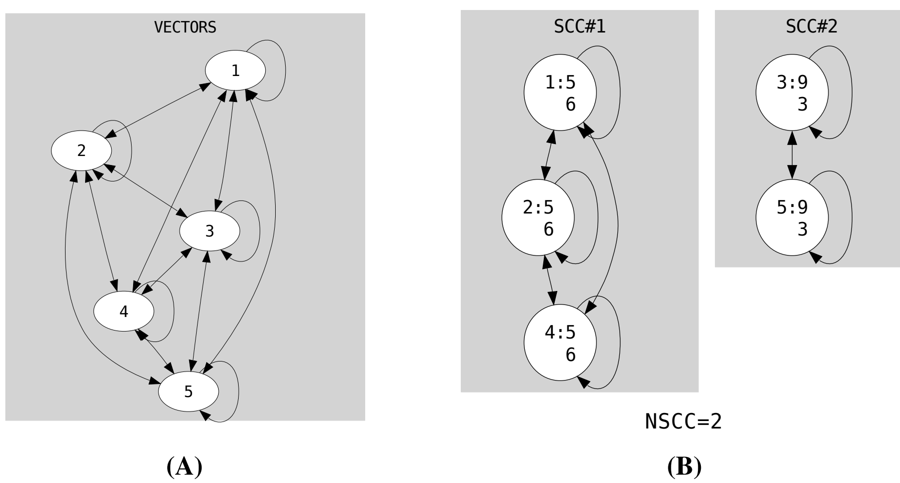


Figure 5.645: Initial and final graph of the NVECTORS constraint

5.298 NVISIBLE_FROM_END

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Derived from NVISIBLE_FROM_START

Constraint NVISIBLE_FROM_END(N, VARIABLES)

Synonyms NVISIBLE, NVISIBLE_FROM_RIGHT.

Arguments

```
N         : dvar
VARIABLES : collection(var−dvar)
```

Restrictions

required(VARIABLES, var)
$\mathtt{N} \geq \mathtt{min}(1, |\mathtt{VARIABLES}|)$
$\mathtt{N} \leq |\mathtt{VARIABLES}|$

Purpose The i^{th} ($1 \leq i \leq |\mathtt{VARIABLES}|$) variable of the sequence VARIABLES is *visible* if and only if all variables after the i^{th} variable are strictly smaller than the i^{th} variable itself. N is the total number of visible variables of the sequence of variables VARIABLES.

Example

$(2, \langle 1, 6, 2, 1, 4, 8, 2\rangle)$
$(1, \langle 3, 6, 2, 1, 4, 8, 8\rangle)$
$(7, \langle 9, 8, 7, 5, 4, 3, 2\rangle)$

The first NVISIBLE_FROM_END constraint holds since the sequence 1 6 2 1 4 8 2 contains two visible items that respectively correspond to the seventh and sixth items.

Typical

$|\mathtt{VARIABLES}| > 2$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 2$

Typical model $\mathtt{nval}(\mathtt{VARIABLES.var}) > 2$

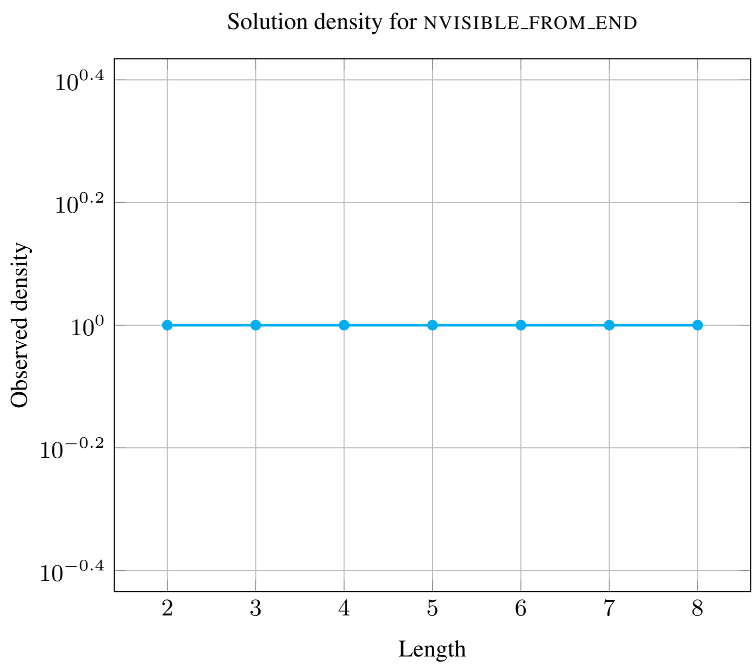
Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

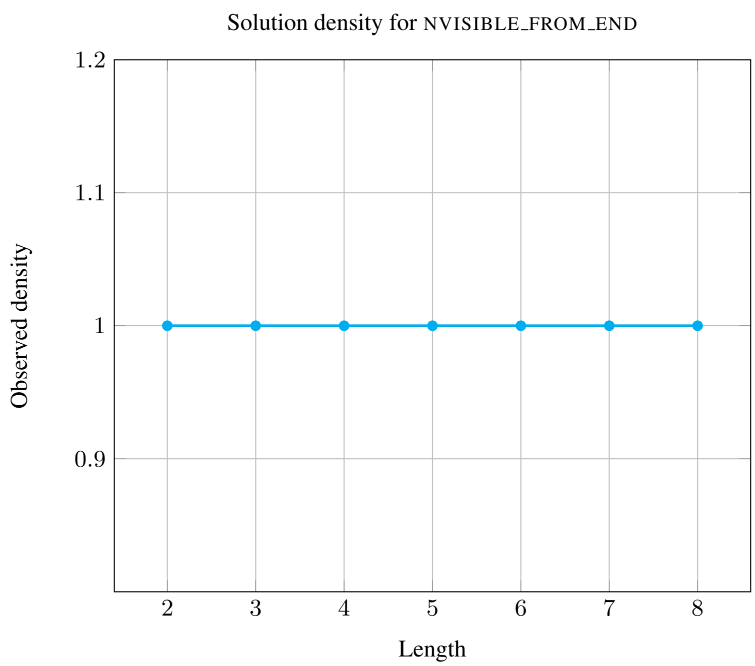
Arg. properties Functional dependency: N determined by VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

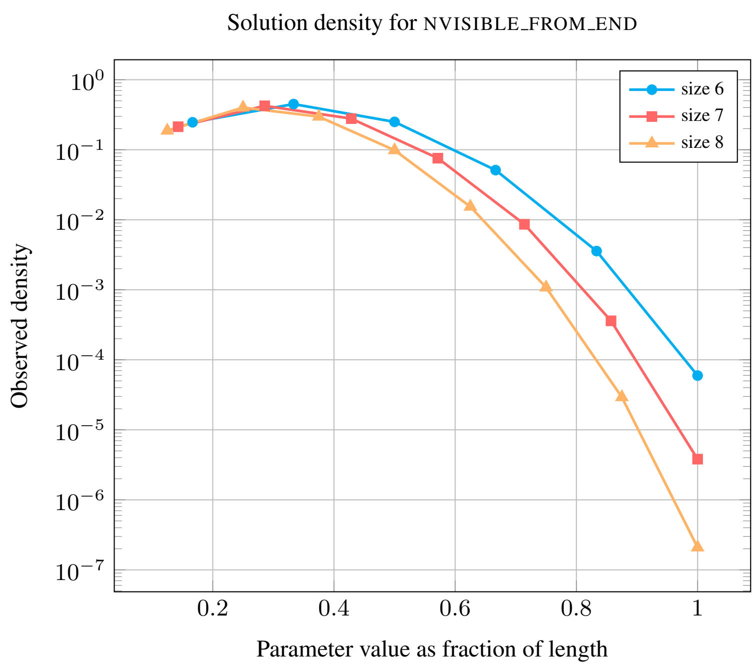
Number of solutions for NVISIBLE_FROM_END: domains $0..n$

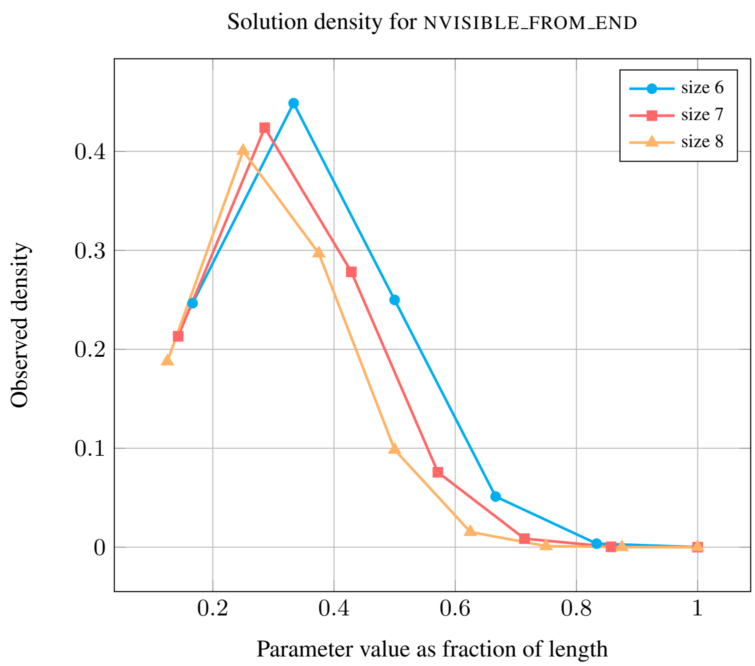

Solution density for NVISIBLE_FROM_END
Observed density
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
2 3 4 5 6 7 8
Length


Solution density for NVISIBLE_FROM_END
Observed density
1.2
1.1
1
0.9
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	6	30	225	2275	29008	446964	8080425
	2	3	30	305	3675	52794	889056	17238570
	3	-	4	90	1610	29400	583548	12780180
	4	-	-	5	210	6020	158760	4238367
	5	-	-	-	6	420	18060	661500
	6	-	-	-	-	7	756	46410
	7	-	-	-	-	-	8	1260
	8	-	-	-	-	-	-	9

Solution count for NVISIBLE_FROM_END: domains $0..n$





See also **implies:** ATLEAST_NVALUE.

related: NVISIBLE_FROM_START *(count from the start of the sequence rather than from the end)*.

Keywords **combinatorial object:** sequence.

constraint arguments: pure functional dependency.

modelling: functional dependency.

Automaton Figure 5.646 depicts the automaton associated with the NVISIBLE_FROM_END constraint.

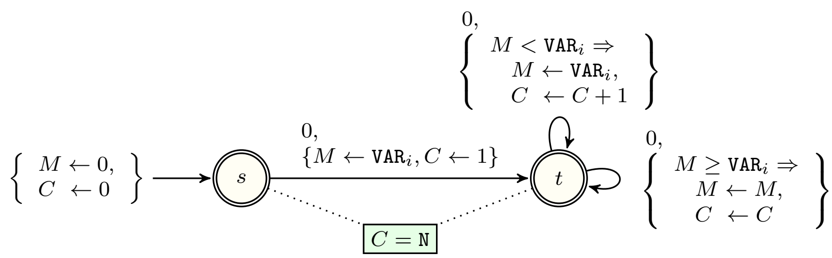


Figure 5.646: Automaton of the NVISIBLE_FROM_END constraint with two counters M and C, where M records the largest value encountered so far, and C the number of visible values from the right hand side of the sequence $\mathtt{VAR}_1$, $\mathtt{VAR}_2$, ..., $\mathtt{VAR}_n$ (i.e., the sequence $\mathtt{VAR}_n$, $\mathtt{VAR}_{n-1}$, ..., $\mathtt{VAR}_1$ is passed to the automaton)

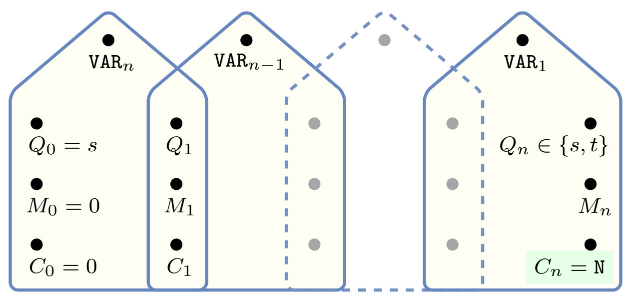


Figure 5.647: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the NVISIBLE_FROM_END constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

5.299 NVISIBLE_FROM_START

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Derived from a puzzle called skyscraper

Constraint NVISIBLE_FROM_START(N, VARIABLES)

Synonyms NVISIBLE, NVISIBLE_FROM_LEFT.

Arguments

```
N          : dvar
VARIABLES  : collection(var−dvar)
```

Restrictions

required(VARIABLES, var)
$\mathtt{N} \geq \mathtt{min}(1, |\mathtt{VARIABLES}|)$
$\mathtt{N} \leq |\mathtt{VARIABLES}|$

Purpose The i^{th} $(1 \leq i \leq |\mathtt{VARIABLES}|)$ variable of the sequence VARIABLES is *visible* if and only if all variables before the i^{th} variable are strictly smaller than the i^{th} variable itself. N is the total number of visible variables of the sequence of variables VARIABLES.

Example

$(3, \langle 1, 6, 2, 1, 4, 8, 2 \rangle)$
$(1, \langle 8, 6, 2, 1, 4, 8, 2 \rangle)$
$(7, \langle 0, 2, 3, 5, 6, 7, 9 \rangle)$

The first NVISIBLE_FROM_START constraint holds since the sequence 1 6 2 1 4 8 2 contains three visible items that respectively correspond to the first, second and sixth items.

Typical

$|\mathtt{VARIABLES}| > 2$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 2$

Typical model $\mathtt{nval}(\mathtt{VARIABLES.var}) > 2$

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

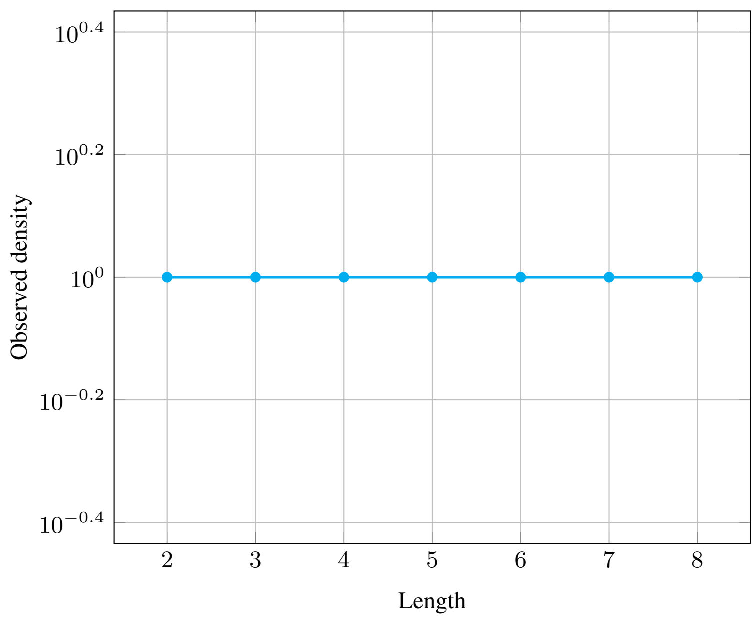
Arg. properties Functional dependency: N determined by VARIABLES.

Counting

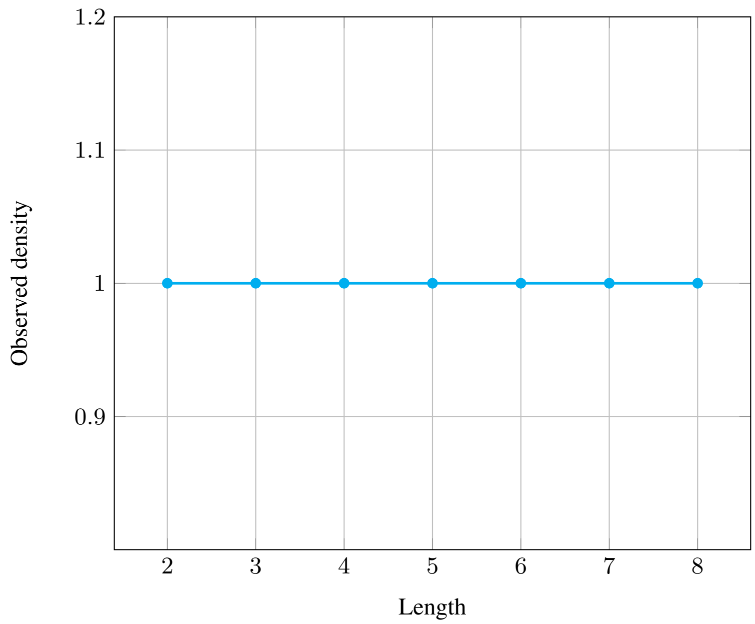
Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for NVISIBLE_FROM_START: domains $0..n$

Solution density for NVISIBLE_FROM_START



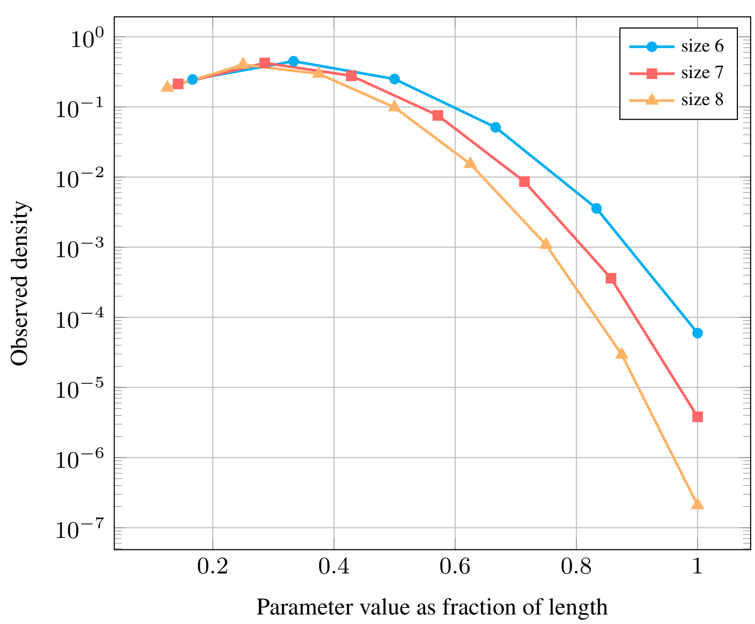
Solution density for NVISIBLE_FROM_START

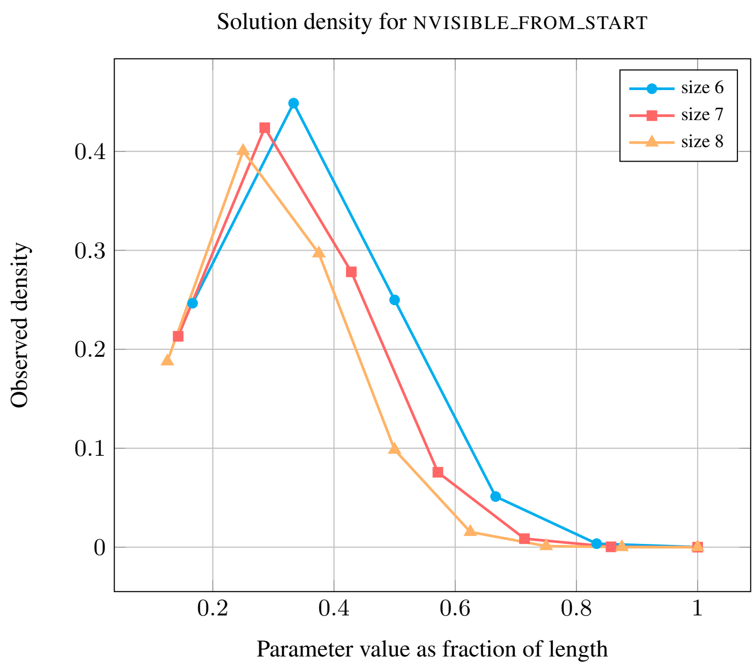


Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	6	30	225	2275	29008	446964	8080425
	2	3	30	305	3675	52794	889056	17238570
	3	-	4	90	1610	29400	583548	12780180
	4	-	-	5	210	6020	158760	4238367
	5	-	-	-	6	420	18060	661500
	6	-	-	-	-	7	756	46410
	7	-	-	-	-	-	8	1260
	8	-	-	-	-	-	-	9

Solution count for NVISIBLE_FROM_START: domains $0..n$

Solution density for NVISIBLE_FROM_START





See also

implied by: INCREASING_NVALUE.

implies: ATLEAST_NVALUE.

related: NVISIBLE_FROM_END *(count from the end of the sequence rather than from the start).*

Keywords

combinatorial object: sequence.

constraint arguments: pure functional dependency.

modelling: functional dependency.

Automaton Figure 5.648 depicts the automaton associated with the NVISIBLE_FROM_START constraint.

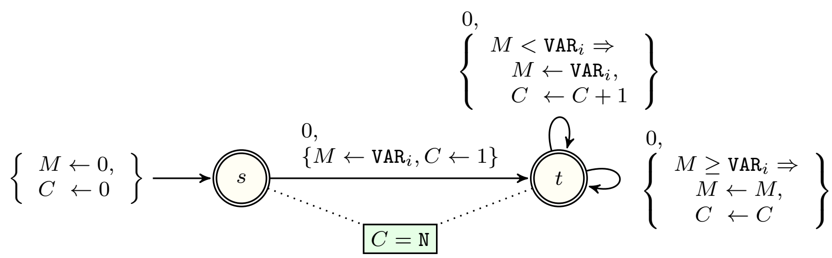


Figure 5.648: Automaton of the NVISIBLE_FROM_START constraint with two counters M and C, where M records the largest value encountered so far, and C the number of visible values from the left hand side of the sequence $\mathtt{VAR}_1$, $\mathtt{VAR}_2$, ..., $\mathtt{VAR}_n$

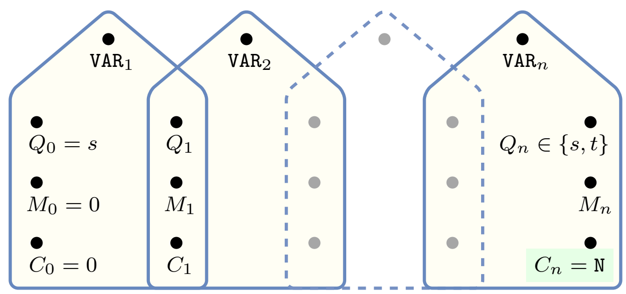


Figure 5.649: Hypergraph of the reformulation corresponding to the automaton (with two counters) of the NVISIBLE_FROM_START constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

5.300 OPEN_ALLDIFFERENT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [438]

Constraint OPEN_ALLDIFFERENT(S, VARIABLES)

Synonyms OPEN_ALLDIFF, OPEN_ALLDISTINCT, OPEN_DISTINCT.

Arguments

S	:	svar
VARIABLES	:	collection(var−dvar)

Restrictions

$S \geq 1$
$S \leq |VARIABLES|$
required(VARIABLES, var)

Purpose Let $\mathcal{V}$ be the variables of the collection VARIABLES for which the corresponding position belongs to the set S. Positions are numbered from 1. Enforce all variables of $\mathcal{V}$ to take distinct values.

Example $(\{2, 3, 4\}, \langle 9, 1, 9, 3\rangle)$

The OPEN_ALLDIFFERENT constraint holds since the last three (i.e., $S = \{2, 3, 4\}$) values of the collection $\langle 9, 1, 9, 3\rangle$ are distinct.

Typical $|VARIABLES| > 2$

Symmetry All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Suffix-contractible wrt. VARIABLES.

Usage In their article [438], W.-J. van Hoeve and J.-C. Régin motivate the OPEN_ALLDIFFERENT constraint by the following scheduling problem. Consider a set of activities (where each activity has a fixed duration 1 and a start variable) that can be processed on two factory lines such that all the activities that will be processed on a given line must be pairwise distinct. This can be modelled by using one OPEN_ALLDIFFERENT constraint for each line, involving all the start variables as well as a set variable whose final value specifies the set of activities assigned to that specific factory line.

Note that this can also be directly modelled by a single DIFFN constraint. This is done by introducing an assignment variable for each activity. The initial domain of each assignment variable consists of two values that respectively correspond to the two factory lines.

Algorithm A slight adaptation of the flow model that handles the original GLOBAL_CARDINALITY constraint [353] is described in [438]. The rightmost part of Figure 3.29 illustrates this flow model.

See also

common keyword: SIZE_MAX_SEQ_ALLDIFFERENT, SIZE_MAX_STARTING_SEQ_ALLDIFFERENT *(all different,disequality)*.

generalisation: OPEN_GLOBAL_CARDINALITY *(control the number of occurrence of each active value*[14] *with a counter variable)*, OPEN_GLOBAL_CARDINALITY_LOW_UP *(control the number of occurrence of each active value with an interval)*.

hard version: ALLDIFFERENT.

used in graph description: IN_SET.

Keywords

characteristic of a constraint: all different, disequality.

constraint arguments: constraint involving set variables.

constraint type: open constraint, soft constraint, value constraint.

filtering: flow.

[14] An *active value* corresponds to a value occuring at a position mentionned in the set S.

Arc input(s) VARIABLES

Arc generator $CLIQUE \mapsto$ collection(variables1, variables2)

Arc arity 2

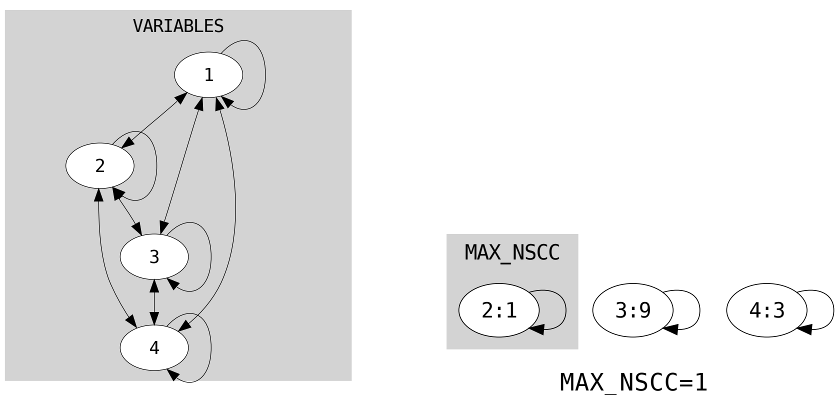
Arc constraint(s)
- variables1.var = variables2.var
- IN_SET(variables1.key, S)
- IN_SET(variables2.key, S)

Graph property(ies) **MAX_NSCC** ≤ 1

Graph class ONE_SUCC

Graph model We generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one. Variables for which the corresponding position does not belong to the set S are removed from the final graph by the second and third conditions of the arc-constraint.

Parts (A) and (B) of Figure 5.650 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph. The OPEN_ALLDIFFERENT holds since all the strongly connected components have at most one vertex: a value is used at most once.



(A) **(B)**

Figure 5.650: Initial and final graph of the OPEN_ALLDIFFERENT constraint

5.301 OPEN_AMONG

DESCRIPTION LINKS GRAPH

Origin

Derived from AMONG and OPEN_GLOBAL_CARDINALITY.

Constraint

OPEN_AMONG(S, NVAR, VARIABLES, VALUES)

Arguments

S	:	svar
NVAR	:	dvar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int)

Restrictions

$\mathtt{S} \geq 1$
$\mathtt{S} \leq |\mathtt{VARIABLES}|$
$\mathtt{NVAR} \geq 0$
$\mathtt{NVAR} \leq |\mathtt{VARIABLES}|$
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)

Purpose

Let $\mathcal{V}$ be the variables of the collection VARIABLES for which the corresponding position belongs to the set S. Positions are numbered from 1. NVAR is the number of variables of $\mathcal{V}$ that take their values in VALUES.

Example

$(\{2,3,4,5\}, 3, \langle 8,5,5,4,1\rangle, \langle 1,5,8\rangle)$

The OPEN_AMONG constraint holds since within the last four values (i.e., $\mathtt{S} = \{2,3,4,5\}$) of $\langle 8,5,5,4,1\rangle$ exactly 3 values belong to the set of values $\{1,5,8\}$.

Typical

$\mathtt{NVAR} > 0$
$\mathtt{NVAR} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$
$|\mathtt{VALUES}| > 1$
$|\mathtt{VARIABLES}| > |\mathtt{VALUES}|$

Symmetries

- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value in VALUES.val (resp. not in VALUES.val).

Arg. properties

- Functional dependency: NVAR determined by S, VARIABLES and VALUES.
- Suffix-contractible wrt. VARIABLES when $\mathtt{NVAR} = 0$.

See also

common keyword: OPEN_ATLEAST, OPEN_ATMOST *(open constraint,value constraint)*, OPEN_GLOBAL_CARDINALITY *(open constraint,counting constraint)*.

hard version: AMONG.

used in graph description: IN_SET.

Keywords

constraint arguments: constraint involving set variables.

constraint type: open constraint, value constraint, counting constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$SELF$ ↦ collection(variables)
Arc arity	1
Arc constraint(s)	• IN(variables.var, VALUES) • IN_SET(variables.key, S)
Graph property(ies)	**NARC** = NVAR

Graph model The arc constraint corresponds to the conjunction of unary constraints IN(variables.var, VALUES) and IN_SET(variables.key, S) defined in this catalogue. Consequently we employ the $SELF$ arc generator in order to produce an initial graph with a single loop on each vertex.

Parts (A) and (B) of Figure 5.651 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

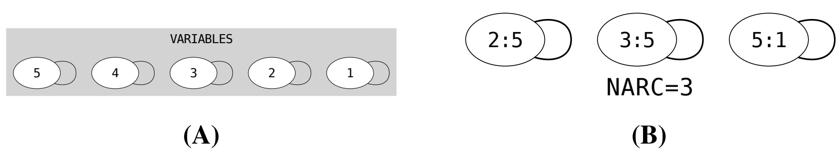


Figure 5.651: Initial and final graph of the OPEN_AMONG constraint

5.302 OPEN_ATLEAST

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from ATLEAST and OPEN_GLOBAL_CARDINALITY.

Constraint OPEN_ATLEAST(S, N, VARIABLES, VALUE)

Arguments

S	:	svar
N	:	int
VARIABLES	:	collection(var−dvar)
VALUE	:	int

Restrictions

$\mathtt{S} \geq 1$
$\mathtt{S} \leq |\mathtt{VARIABLES}|$
$\mathtt{N} \geq 0$
$\mathtt{N} \leq |\mathtt{VARIABLES}|$
required(VARIABLES, var)

Purpose Let $\mathcal{V}$ be the variables of the collection VARIABLES for which the corresponding position belongs to the set S. Positions are numbered from 1. At least N variables of $\mathcal{V}$ are assigned value VALUE.

Example $(\{2,3,4\}, 2, \langle 4,2,4,4\rangle, 4)$

The OPEN_ATLEAST constraint holds since, within the last three (i.e., $\mathtt{S} = \{2,3,4\}$) values of the collection $\langle 4,2,4,4\rangle$, at least $\mathtt{N} = 2$ values are equal to value $\mathtt{VALUE} = 4$.

Typical

$\mathtt{N} > 0$
$\mathtt{N} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$

Symmetries

- N can be decreased to any value ≥ 0.
- An occurrence of a value of VARIABLES.var that is different from VALUE can be replaced by any other value.

Arg. properties Suffix-extensible wrt. VARIABLES.

See also **common keyword:** OPEN_AMONG, OPEN_GLOBAL_CARDINALITY *(open constraint,value constraint)*.

comparison swapped: OPEN_ATMOST.

hard version: ATLEAST.

used in graph description: IN_SET.

Keywords **constraint arguments:** constraint involving set variables.

constraint type: open constraint, value constraint.

modelling: at least.

Arc input(s)	VARIABLES
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	• variables.var = VALUE • IN_SET(variables.key, S)
Graph property(ies)	**NARC** ≥ N

Graph model Since each arc constraint involves only one vertex (VALUE is fixed), we employ the *SELF* arc generator in order to produce a graph with a single loop on each vertex. Variables for which the corresponding position does not belong to the set S are removed from the final graph by the second condition of the arc-constraint.

Parts (A) and (B) of Figure 5.652 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

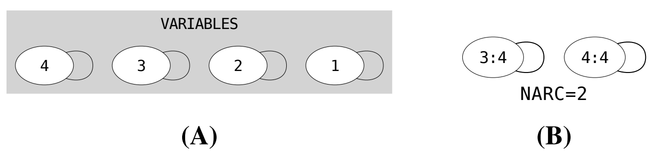


Figure 5.652: Initial and final graph of the OPEN_ATLEAST constraint

5.303 OPEN_ATMOST

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from ATMOST and OPEN_GLOBAL_CARDINALITY.

Constraint OPEN_ATMOST(S, N, VARIABLES, VALUE)

Arguments

S	:	svar
N	:	int
VARIABLES	:	collection(var−dvar)
VALUE	:	int

Restrictions

$\mathtt{S} \geq 1$
$\mathtt{S} \leq |\mathtt{VARIABLES}|$
$\mathtt{N} \geq 0$
required(VARIABLES, var)

Purpose Let $\mathcal{V}$ be the variables of the collection VARIABLES for which the corresponding position belongs to the set S. Positions are numbered from 1. At most N variables of $\mathcal{V}$ are assigned value VALUE.

Example $(\{2, 3, 4\}, 1, \langle 2, 2, 4, 5\rangle, 2)$

The OPEN_ATMOST constraint holds since, within the last three (i.e., $\mathtt{S} = \{2, 3, 4\}$) values of the collection $\langle 2, 2, 4, 5\rangle$, at most $\mathtt{N} = 1$ value is equal to value $\mathtt{VALUE} = 2$.

Typical

$\mathtt{N} > 0$
$\mathtt{N} < |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$

Symmetries

- N can be increased.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that is different from VALUE.

Arg. properties Suffix-contractible wrt. VARIABLES.

See also

common keyword: OPEN_AMONG, OPEN_GLOBAL_CARDINALITY *(open constraint,value constraint)*.

comparison swapped: OPEN_ATLEAST.

hard version: ATMOST.

used in graph description: IN_SET.

Keywords

constraint arguments: constraint involving set variables.

constraint type: open constraint, value constraint.

modelling: at most.

Arc input(s)	VARIABLES
Arc generator	$SELF \mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	• variables.var = VALUE • IN_SET(variables.key, S)
Graph property(ies)	**NARC** $\leq$ N

Graph model

Since each arc constraint involves only one vertex (VALUE is fixed), we employ the *SELF* arc generator in order to produce a graph with a single loop on each vertex. Variables for which the corresponding position does not belong to the set S are removed from the final graph by the second condition of the arc-constraint.

Parts (A) and (B) of Figure 5.653 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loops of the final graph are stressed in bold.

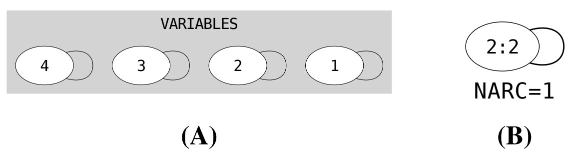


Figure 5.653: Initial and final graph of the OPEN_ATMOST constraint

5.304 OPEN_GLOBAL_CARDINALITY

DESCRIPTION LINKS GRAPH

Origin [438]

Constraint OPEN_GLOBAL_CARDINALITY(S, VARIABLES, VALUES)

Synonyms OPEN_GCC, OGCC.

Arguments

S : svar
VARIABLES : collection(var−dvar)
VALUES : collection(val−int, noccurrence−dvar)

Restrictions

$\texttt{S} \geq 1$
$\texttt{S} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)
required(VALUES, [val, noccurrence])
distinct(VALUES, val)
$\texttt{VALUES.noccurrence} \geq 0$
$\texttt{VALUES.noccurrence} \leq |\texttt{VARIABLES}|$

Purpose

Each value VALUES$[i]$.val ($1 \leq i \leq |\texttt{VALUES}|$) should be taken by exactly VALUES$[i]$.noccurrence variables of the VARIABLES collection for which the corresponding position belongs to the set S. Positions are numbered from 1.

Example

$$\left(\begin{array}{l}\{2,3,4\},\\ \langle 3,3,8,6\rangle,\\ \left\langle\begin{array}{ll}\texttt{val}-3 & \texttt{noccurrence}-1,\\ \texttt{val}-5 & \texttt{noccurrence}-0,\\ \texttt{val}-6 & \texttt{noccurrence}-1\end{array}\right\rangle\end{array}\right)$$

The OPEN_GLOBAL_CARDINALITY constraint holds since:

- Values 3, 5 and 6 respectively occur 1, 0 and 1 times within the collection $\langle 3,3,8,6\rangle$ (the first item 3 of $\langle 3,3,8,6\rangle$ is ignored since value 1 does not belong to the first argument $S = \{2,3,4\}$ of the OPEN_GLOBAL_CARDINALITY constraint).
- No constraint was specified for value 8.

Typical

$|\texttt{VARIABLES}| > 1$
range(VARIABLES.var) > 1
$|\texttt{VALUES}| > 1$
range(VALUES.noccurrence) > 1
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$

Symmetries

- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.

Usage

In their article [438], W.-J. van Hoeve and J.-C. Régin motivate the OPEN_GLOBAL_CARDINALITY constraint by the following scheduling problem. Consider a set of activities (where each activity has a fixed duration 1 and a start variable) that can be processed on two factory lines such that all the activities that will be processed on a given line must be pairwise distinct. This can be modelled by using one OPEN_GLOBAL_CARDINALITY constraint for each line, involving all the start variables as well as a set variable whose final value specifies the set of activities assigned to that specific factory line.

Note that this can also be directly modelled by a single DIFFN constraint. This is done by introducing an assignment variable for each activity. The initial domain of each assignment variable consists of two values that respectively correspond to the two factory lines.

Remark

In their article [438], W.-J. van Hoeve and J.-C. Régin consider the case where we have no counter variables for the values, but rather some lower and upper bounds (i.e., in fact the OPEN_GLOBAL_CARDINALITY_LOW_UP constraint).

Algorithm

A slight adaptation of the flow model that handles the original GLOBAL_CARDINALITY constraint [353] is described in [438].

See also

common keyword: GLOBAL_CARDINALITY_LOW_UP *(assignment,counting constraint)*, OPEN_AMONG *(open constraint,counting constraint)*, OPEN_ATLEAST, OPEN_ATMOST *(open constraint,value constraint)*.

hard version: GLOBAL_CARDINALITY.

specialisation: OPEN_ALLDIFFERENT *(each active value[15] should occur at most once)*, OPEN_GLOBAL_CARDINALITY_LOW_UP *(*variable *replaced by* fixed interval*)*.

used in graph description: IN_SET.

Keywords

application area: assignment.

constraint arguments: constraint involving set variables.

constraint type: open constraint, value constraint, counting constraint.

filtering: flow.

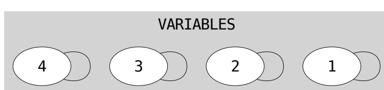
[15] An *active value* corresponds to a value occuring at a position mentionned in the set S.

For all items of VALUES:

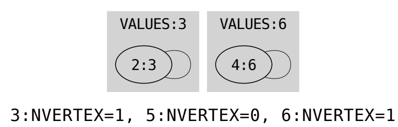
Arc input(s)	VARIABLES
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	• variables.var = VALUES.val • IN_SET(variables.key, S)
Graph property(ies)	**NVERTEX** = VALUES.noccurrence

Graph model Since we want to express one unary constraint for each value we use the "For all items of VALUES" iterator. The only difference with the graph model of the GLOBAL_CARDINALITY constraint is the arc constraint where we also specify that the position of the considered variable should belong to the first argument S.

Part (A) of Figure 5.654 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.654 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to those variables of the VARIABLES collection for which the index belongs to S (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold.



(A)



(B)

Figure 5.654: Initial and final graph of the OPEN_GLOBAL_CARDINALITY constraint

5.305 OPEN_GLOBAL_CARDINALITY_LOW_UP

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [438]

Constraint OPEN_GLOBAL_CARDINALITY_LOW_UP(S, VARIABLES, VALUES)

Arguments

S	:	svar
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int, omin−int, omax−int)

Restrictions

$\texttt{S} \geq 1$
$\texttt{S} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)
$|\texttt{VALUES}| > 0$
required(VALUES, [val, omin, omax])
distinct(VALUES, val)
$\texttt{VALUES.omin} \geq 0$
$\texttt{VALUES.omax} \leq |\texttt{VARIABLES}|$
$\texttt{VALUES.omin} \leq \texttt{VALUES.omax}$

Purpose Each value $\texttt{VALUES}[i].\texttt{val}$ $(1 \leq i \leq |\texttt{VALUES}|)$ should be taken by at least $\texttt{VALUES}[i].\texttt{omin}$ and at most $\texttt{VALUES}[i].\texttt{omax}$ variables of the VARIABLES collection for which the corresponding position belongs to the set S. Positions are numbered from 1.

Example

$$\left(\begin{array}{l}\{2,3,4\},\\ \langle 3,3,8,6\rangle,\\ \left\langle\begin{array}{lll}\texttt{val}-3 & \texttt{omin}-1 & \texttt{omax}-3,\\ \texttt{val}-5 & \texttt{omin}-0 & \texttt{omax}-1,\\ \texttt{val}-6 & \texttt{omin}-1 & \texttt{omax}-2\end{array}\right\rangle\end{array}\right)$$

The OPEN_GLOBAL_CARDINALITY_LOW_UP constraint holds since:

- Values 3, 5 and 6 are respectively used 1 $(1 \leq 1 \leq 3)$, 0 $(0 \leq 0 \leq 1)$ and 1 $(1 \leq 1 \leq 2)$ times within the collection $\langle 3,3,8,6\rangle$ (the first item 3 of $\langle 3,3,8,6\rangle$ is ignored since value 1 does not belong to the first argument $S = \{2,3,4\}$ of the OPEN_GLOBAL_CARDINALITY_LOW_UP constraint).
- No constraint was specified for value 8.

Typical

$|\texttt{VARIABLES}| > 1$
$\text{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VALUES}| > 1$
$\texttt{VALUES.omin} \leq |\texttt{VARIABLES}|$
$\texttt{VALUES.omax} > 0$
$\texttt{VALUES.omax} \leq |\texttt{VARIABLES}|$
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$

Symmetries

- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.

Usage

In their article [438], W.-J. van Hoeve and J.-C. Régin motivate the OPEN_GLOBAL_CARDINALITY_LOW_UP constraint by the following scheduling problem. Consider a set of activities (where each activity has a fixed duration 1 and a start variable) that can be processed on two factory lines such that all the activities that will be processed on a given line must be pairwise distinct. This can be modelled by using one OPEN_GLOBAL_CARDINALITY_LOW_UP constraint for each line, involving all the start variables as well as a set variable whose final value specifies the set of activities assigned to that specific factory line.

Note that this can also be directly modelled by a single DIFFN constraint. This is done by introducing an assignment variable for each activity. The initial domain of each assignment variable consists of two values that respectively correspond to the two factory lines.

Algorithm

A slight adaptation of the flow model that handles the original GLOBAL_CARDINALITY constraint [353] is described in [438].

See also

common keyword: GLOBAL_CARDINALITY *(assignment,counting constraint)*.

generalisation: OPEN_GLOBAL_CARDINALITY *(*fixed interval *replaced by* variable*)*.

hard version: GLOBAL_CARDINALITY_LOW_UP.

specialisation: OPEN_ALLDIFFERENT *(each active value*[16] *should occur at most once)*.

used in graph description: IN_SET.

Keywords

application area: assignment.

constraint arguments: constraint involving set variables.

constraint type: open constraint, value constraint, counting constraint.

filtering: flow.

[16] An *active value* corresponds to a value occuring at a position mentionned in the set S.

For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	$SELF \mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	• variables.var = VALUES.val • IN_SET(variables.key, S)
Graph property(ies)	• **NVERTEX** $\geq$ VALUES.omin • **NVERTEX** $\leq$ VALUES.omax

Graph model Since we want to express one unary constraint for each value we use the "For all items of VALUES" iterator. The only difference from the graph model of the GLOBAL_CARDINALITY_LOW_UP constraint is the arc constraint where we also specify that the position of the considered variable should belong to the first argument S.

Part (A) of Figure 5.655 shows the initial graphs associated with each value 3, 5 and 6 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.655 shows the two corresponding final graphs respectively associated with values 3 and 6 that are both assigned to the variables of the VARIABLES collection (since value 5 is not assigned to any variable of the VARIABLES collection the final graph associated with value 5 is empty). Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold.

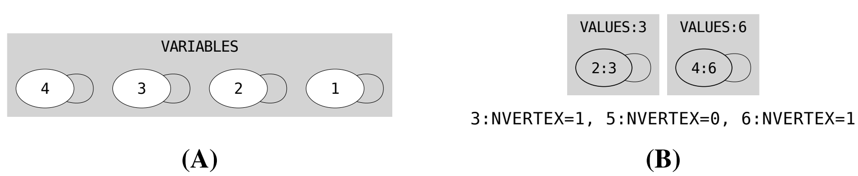


Figure 5.655: Initial and final graph of the OPEN_GLOBAL_CARDINALITY_LOW_UP constraint

5.306 OPEN_MAXIMUM

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Derived from MAXIMUM

Constraint OPEN_MAXIMUM(MAX, VARIABLES)

Arguments

MAX	:	dvar
VARIABLES	:	collection(var−dvar, bool−dvar)

Restrictions

$|\texttt{VARIABLES}| > 0$
required(VARIABLES, [var, bool])
VARIABLES.bool ≥ 0
VARIABLES.bool ≤ 1

Purpose MAX is the maximum value of the variables VARIABLES[i].var, $(1 \leq i \leq |\texttt{VARIABLES}|)$ for which VARIABLES[i].bool $= 1$ (at least one of the Boolean variables is set to 1).

Example

$$\left(5, \left\langle \begin{array}{ll} \texttt{var}-3 & \texttt{bool}-1, \\ \texttt{var}-1 & \texttt{bool}-0, \\ \texttt{var}-7 & \texttt{bool}-0, \\ \texttt{var}-5 & \texttt{bool}-1, \\ \texttt{var}-5 & \texttt{bool}-1 \end{array} \right\rangle \right)$$

The OPEN_MAXIMUM constraint holds since its first argument MAX $= 5$ is set to the maximum value of values $3, 1, 7, 5, 5$ for which the corresponding Boolean $1, 0, 0, 1, 1$ is set to 1 (i.e., values $3, 5, 5$).

Typical

$|\texttt{VARIABLES}| > 1$
range(VARIABLES.var) > 1

Symmetries

- Items of VARIABLES are permutable.
- One and the same constant can be added to MAX as well as to the var attribute of all items of VARIABLES.

See also

comparison swapped: OPEN_MINIMUM.

hard version: MAXIMUM.

used in graph description: IN_SET.

Keywords

characteristic of a constraint: maximum, automaton, automaton without counters, reified automaton constraint.

constraint network structure: centered cyclic(1) constraint network(1).

constraint type: order constraint, open constraint, open automaton constraint.

Automaton Figure 5.656 depicts the automaton associated with the OPEN_MAXIMUM constraint. Let $\mathtt{VAR}_i, \mathtt{B}_i$ be the i^{th} item of the VARIABLES collection. To each triple $(\mathtt{MAX}, \mathtt{VAR}_i, \mathtt{B}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\mathtt{B}_i = 1 \wedge \mathtt{MAX} < \mathtt{VAR}_i \Leftrightarrow S_i = 0) \wedge (\mathtt{B}_i = 1 \wedge \mathtt{MAX} = \mathtt{VAR}_i \Leftrightarrow S_i = 1) \wedge (\mathtt{B}_i = 1 \wedge \mathtt{MAX} > \mathtt{VAR}_i \Leftrightarrow S_i = 2) \wedge (\mathtt{B}_i = 0 \wedge \mathtt{MAX} < \mathtt{VAR}_i \Leftrightarrow S_i = 3) \wedge (\mathtt{B}_i = 0 \wedge \mathtt{MAX} = \mathtt{VAR}_i \Leftrightarrow S_i = 4) \wedge (\mathtt{B}_i = 0 \wedge \mathtt{MAX} > \mathtt{VAR}_i \Leftrightarrow S_i = 5)$.

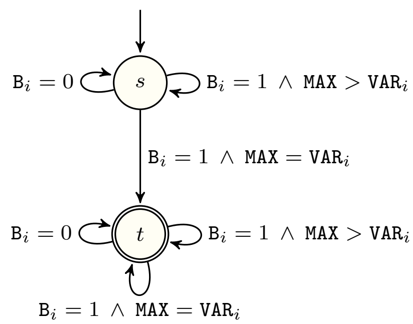


Figure 5.656: Automaton of the OPEN_MAXIMUM constraint

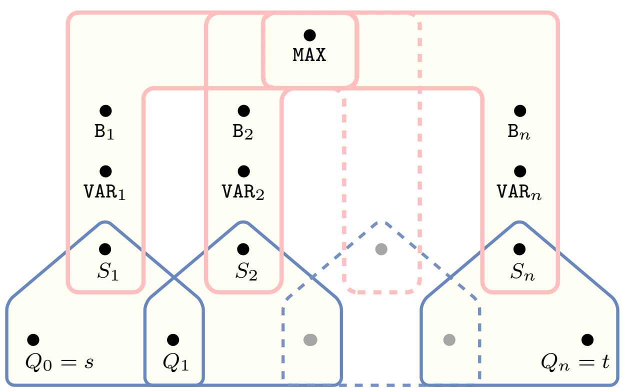


Figure 5.657: Hypergraph of the reformulation corresponding to the automaton of the OPEN_MAXIMUM constraint

5.307 OPEN_MINIMUM

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Derived from MINIMUM

Constraint OPEN_MINIMUM(MIN, VARIABLES)

Arguments

MIN	:	dvar
VARIABLES	:	collection(var−dvar, bool−dvar)

Restrictions

$|\texttt{VARIABLES}| > 0$
required(VARIABLES, [var, bool])
$\texttt{VARIABLES.bool} \geq 0$
$\texttt{VARIABLES.bool} \leq 1$

Purpose

MIN is the minimum value of the variables $\texttt{VARIABLES}[i].\texttt{var}$, $(1 \leq i \leq |\texttt{VARIABLES}|)$ for which $\texttt{VARIABLES}[i].\texttt{bool} = 1$ (at least one of the Boolean variables is set to 1).

Example

$$\left(3, \left\langle \begin{array}{ll} \texttt{var}-3 & \texttt{bool}-1, \\ \texttt{var}-1 & \texttt{bool}-0, \\ \texttt{var}-7 & \texttt{bool}-0, \\ \texttt{var}-5 & \texttt{bool}-1, \\ \texttt{var}-5 & \texttt{bool}-1 \end{array} \right\rangle \right)$$

The OPEN_MINIMUM constraint holds since its first argument $\texttt{MIN} = 3$ is set to the minimum value of values $3, 1, 7, 5, 5$ for which the corresponding Boolean $1, 0, 0, 1, 1$ is set to 1 (i.e., values $3, 5, 5$).

Typical

$|\texttt{VARIABLES}| > 1$
range(VARIABLES.var) > 1

Symmetries

- Items of VARIABLES are permutable.
- One and the same constant can be added to MIN as well as to the var attribute of all items of VARIABLES.

Remark The OPEN_MINIMUM constraint is used in the reformulation of the TREE_RANGE constraint.

See also

comparison swapped: OPEN_MAXIMUM.

hard version: MINIMUM.

used in graph description: IN_SET.

uses in its reformulation: TREE_RANGE.

Keywords

characteristic of a constraint: minimum, automaton, automaton without counters, reified automaton constraint.

constraint network structure: centered cyclic(1) constraint network(1).

constraint type: order constraint, open constraint, open automaton constraint.

Automaton Figure 5.658 depicts the automaton associated with the OPEN_MINIMUM constraint. Let $\texttt{VAR}_i, \texttt{B}_i$ be the i^{th} item of the VARIABLES collection. To each triple $(\texttt{MIN}, \texttt{VAR}_i, \texttt{B}_i)$ corresponds a signature variable S_i as well as the following signature constraint: $(\texttt{B}_i = 1 \wedge \texttt{MIN} < \texttt{VAR}_i \Leftrightarrow S_i = 0) \wedge (\texttt{B}_i = 1 \wedge \texttt{MIN} = \texttt{VAR}_i \Leftrightarrow S_i = 1) \wedge (\texttt{B}_i = 1 \wedge \texttt{MIN} > \texttt{VAR}_i \Leftrightarrow S_i = 2) \wedge (\texttt{B}_i = 0 \wedge \texttt{MIN} < \texttt{VAR}_i \Leftrightarrow S_i = 3) \wedge (\texttt{B}_i = 0 \wedge \texttt{MIN} = \texttt{VAR}_i \Leftrightarrow S_i = 4) \wedge (\texttt{B}_i = 0 \wedge \texttt{MIN} > \texttt{VAR}_i \Leftrightarrow S_i = 5)$.

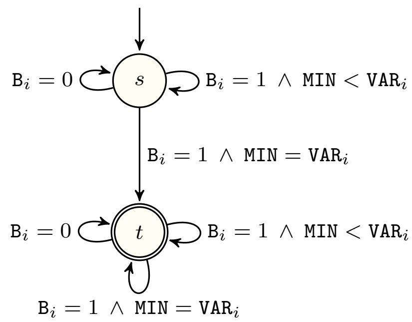


Figure 5.658: Automaton of the OPEN_MINIMUM constraint

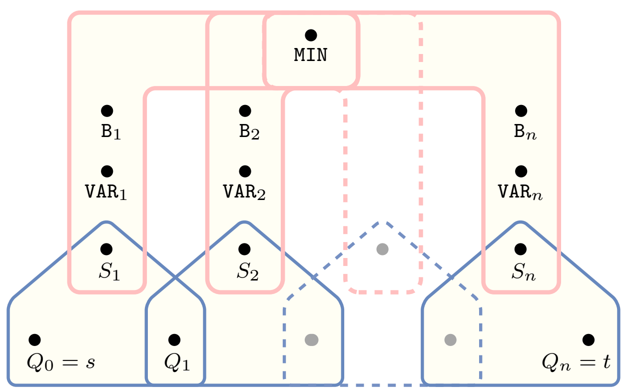


Figure 5.659: Hypergraph of the reformulation corresponding to the automaton of the OPEN_MINIMUM constraint

5.308 OPPOSITE_SIGN

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint OPPOSITE_SIGN(VAR1, VAR2)

Arguments

VAR1	:	dvar
VAR2	:	dvar

Restriction

Purpose Enforce the fact that the product of the first and second variables is less than or equal to 0.

Example $(6, -3)$

The OPPOSITE_SIGN constraint holds since 6 and -3 do not have the same sign.

Typical VAR1 $\neq 0$

Symmetry Arguments are permutable w.r.t. permutation (VAR1, VAR2).

See also **comparison swapped:** SAME_SIGN.

implies (if swap arguments): ABS_VALUE.

Keywords **constraint arguments:** binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.309 OR

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint $\text{OR}(\texttt{VAR}, \texttt{VARIABLES})$

Synonym REL.

Arguments

$\texttt{VAR} : \texttt{dvar}$
$\texttt{VARIABLES} : \texttt{collection}(\texttt{var}-\texttt{dvar})$

Restrictions

$\texttt{VAR} \geq 0$
$\texttt{VAR} \leq 1$
$|\texttt{VARIABLES}| \geq 2$
$\texttt{required}(\texttt{VARIABLES}, \texttt{var})$
$\texttt{VARIABLES.var} \geq 0$
$\texttt{VARIABLES.var} \leq 1$

Purpose Let VARIABLES be a collection of 0-1 variables $\texttt{VAR}_1, \texttt{VAR}_2, \ldots, \texttt{VAR}_n$ $(n \geq 2)$. Enforce $\texttt{VAR} = \texttt{VAR}_1 \vee \texttt{VAR}_2 \vee \cdots \vee \texttt{VAR}_n$.

Example

$(0, \langle 0, 0\rangle)$
$(1, \langle 0, 1\rangle)$
$(1, \langle 1, 0\rangle)$
$(1, \langle 1, 1\rangle)$
$(1, \langle 1, 0, 1\rangle)$

Symmetry Items of VARIABLES are permutable.

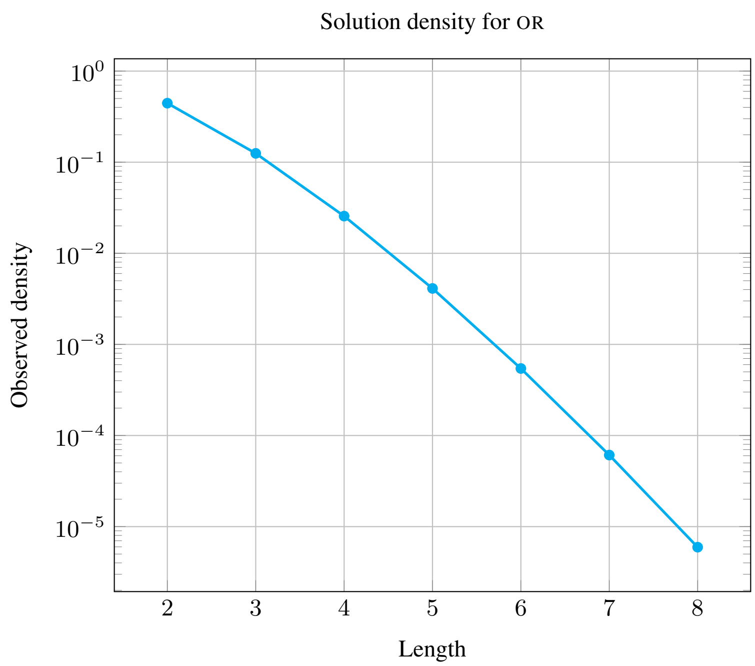
Arg. properties

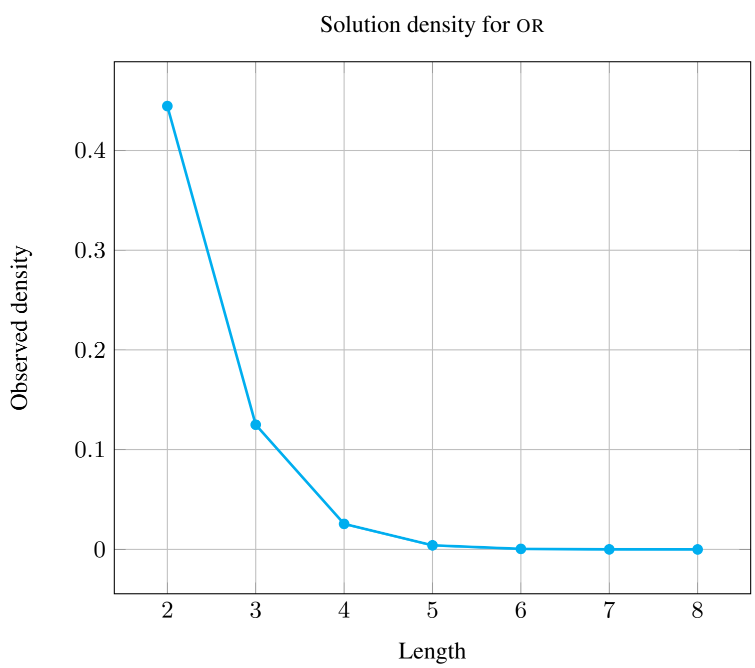
- Functional dependency: VAR determined by VARIABLES.
- Contractible wrt. VARIABLES when $\texttt{VAR} = 0$.
- Extensible wrt. VARIABLES when $\texttt{VAR} = 1$.
- Aggregate: $\texttt{VAR}(\vee)$, $\texttt{VARIABLES}(\texttt{union})$.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	4	8	16	32	64	128	256

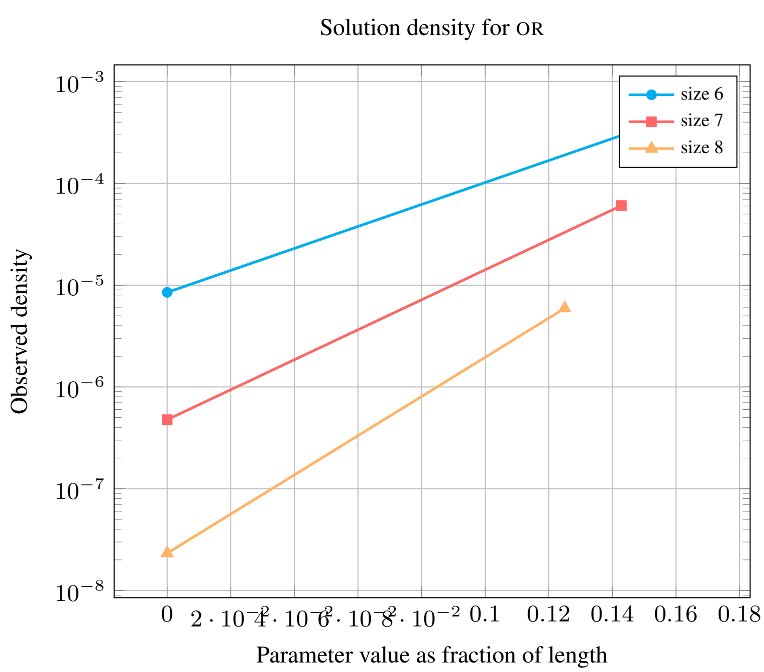
Number of solutions for OR: domains $0..n$

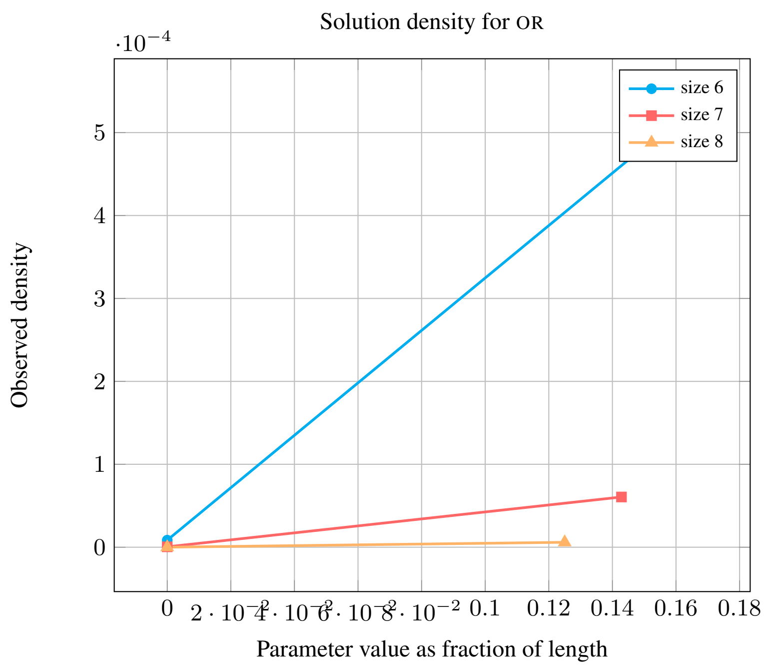

Solution density for OR
Observed density
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
2
3
4
5
6
7
8
Length


Solution density for OR
Observed density
0.4
0.3
0.2
0.1
0
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		4	8	16	32	64	128	256
Parameter value	0	1	1	1	1	1	1	1
	1	3	7	15	31	63	127	255

Solution count for OR: domains $0..n$





Systems REIFIEDOR in **Choco**, REL in **Gecode**, ORBOOL in **JaCoP**, #\/ in **SICStus**.

See also **common keyword:** AND, CLAUSE_OR, EQUIVALENT, IMPLY, NAND, NOR, XOR *(Boolean constraint)*.

implies: ATLEAST_NVALUE, MAXIMUM.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: disjunction, functional dependency.

Cond. implications

- OR(VAR, VARIABLES)
  with $|\mathtt{VARIABLES}| > 2$
  implies SOME_EQUAL(VARIABLES).

- OR(VAR, VARIABLES)
  with $\mathtt{VAR} = 0$
  implies NOR(VAR, VARIABLES)
  when $\mathtt{VAR} = 1$.

- OR(VAR, VARIABLES)
  with $\mathtt{VAR} = 1$
  implies NOR(VAR, VARIABLES)
  when $\mathtt{VAR} = 0$.

Automaton Figure 5.660 depicts a first deterministic automaton without counter associated with the OR constraint. To the first argument VAR of the OR constraint corresponds the first signature variable. To each variable $\mathtt{VAR}_i$ of the second argument VARIABLES of the OR constraint corresponds the next signature variable. There is no signature constraint.

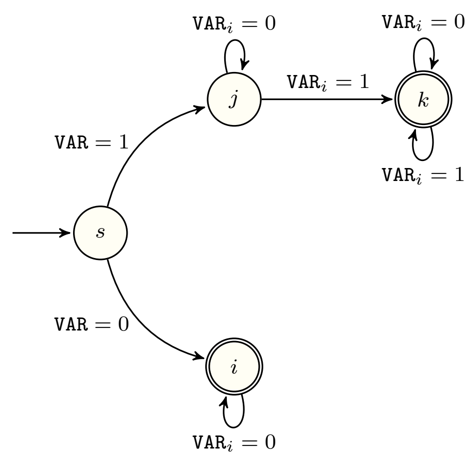


Figure 5.660: Counter free automaton of the OR constraint

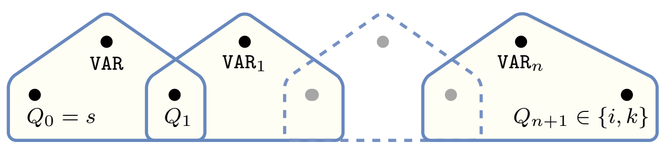


Figure 5.661: Hypergraph of the reformulation corresponding to the automaton of the OR constraint

Figure 5.662 depicts a second deterministic automaton with one counter associated with the OR constraint, where the argument VAR is unified to the final value of the counter.

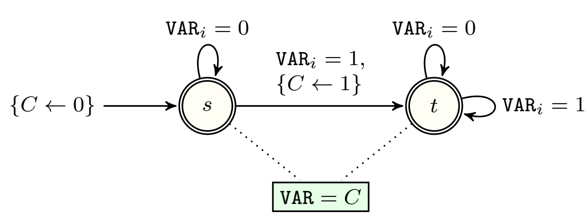


Figure 5.662: Automaton (with one counter) of the OR constraint

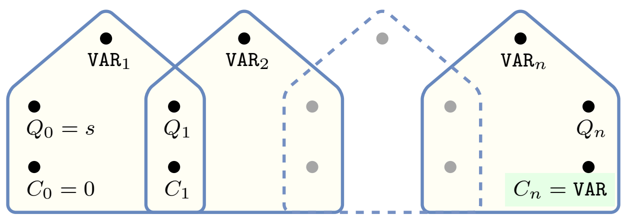


Figure 5.663: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the OR constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_n)

5.310 ORCHARD

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [235]

Constraint ORCHARD(NROW, TREES)

Arguments

NROW : dvar
TREES : collection(index−int, x−dvar, y−dvar)

Restrictions

NROW ≥ 0
TREES.index ≥ 1
TREES.index $\leq$ |TREES|
required(TREES, [index, x, y])
distinct(TREES, index)
TREES.x ≥ 0
TREES.y ≥ 0

Purpose

Orchard problem [235]:

> "*Your aid I want, Nine trees to plant, In rows just half a score, And let there be, In each row, three—Solve this: I ask no more!*"

Example

$$\left(10,\left\langle\begin{array}{lll}\texttt{index}-1 & \texttt{x}-0 & \texttt{y}-0,\\ \texttt{index}-2 & \texttt{x}-4 & \texttt{y}-0,\\ \texttt{index}-3 & \texttt{x}-8 & \texttt{y}-0,\\ \texttt{index}-4 & \texttt{x}-2 & \texttt{y}-4,\\ \texttt{index}-5 & \texttt{x}-4 & \texttt{y}-4,\\ \texttt{index}-6 & \texttt{x}-6 & \texttt{y}-4,\\ \texttt{index}-7 & \texttt{x}-0 & \texttt{y}-8,\\ \texttt{index}-8 & \texttt{x}-4 & \texttt{y}-8,\\ \texttt{index}-9 & \texttt{x}-8 & \texttt{y}-8\end{array}\right\rangle\right)$$

The 10 alignments of 3 trees correspond to the following triples of trees: $(1,2,3)$, $(1,4,8)$, $(1,5,9)$, $(2,4,7)$, $(2,5,8)$, $(2,6,9)$, $(3,5,7)$, $(3,6,8)$, $(4,5,6)$, $(7,8,9)$. Figure 5.664 shows the 9 trees and the 10 alignments corresponding to the example.

Typical

NROW > 0
|TREES| > 3

Symmetries

- Items of TREES are permutable.
- Attributes of TREES are permutable w.r.t. permutation (index) (x, y) (*permutation applied to all items*).
- One and the same constant can be added to the x attribute of all items of TREES.
- One and the same constant can be added to the y attribute of all items of TREES.

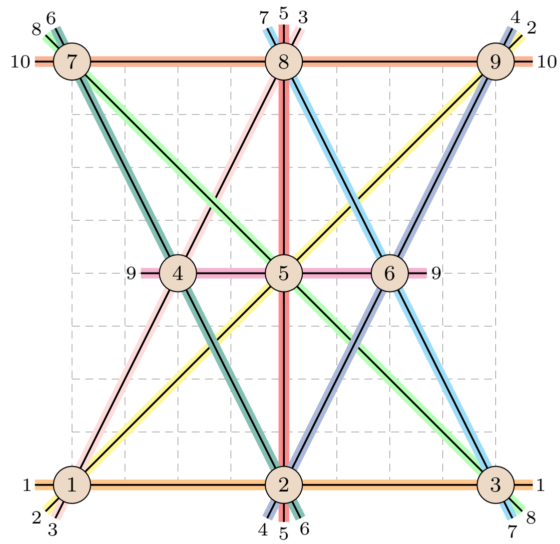


Figure 5.664: Nine trees with 10 alignments of 3 trees

Arg. properties Functional dependency: `NROW` determined by `TREES`.

Keywords **characteristic of a constraint:** hypergraph.

constraint arguments: pure functional dependency.

geometry: geometrical constraint, alignment.

modelling: functional dependency.

Arc input(s)	TREES
Arc generator	$CLIQUE(<) \mapsto$ collection(trees1, trees2, trees3)
Arc arity	3
Arc constraint(s)	$\sum\begin{pmatrix} \texttt{trees1.x} * \texttt{trees2.y} - \texttt{trees1.x} * \texttt{trees3.y}, \\ \texttt{trees1.y} * \texttt{trees3.x} - \texttt{trees1.y} * \texttt{trees2.x}, \\ \texttt{trees2.x} * \texttt{trees3.y} - \texttt{trees2.y} * \texttt{trees3.x} \end{pmatrix} = 0$
Graph property(ies)	**NARC**= NROW

Graph model The arc generator $CLIQUE(<)$ with an arity of three is used in order to generate all the arcs of the directed hypergraph. Each arc is an ordered triple of trees. We use the restriction $<$ in order to generate a single arc for each set of three trees. This is required, since otherwise we would count more than once a given alignment of three trees. The formula used within the arc constraint expresses the fact that the three points of respective coordinates $(\texttt{trees}_1.\texttt{x}, \texttt{trees}_1.\texttt{y})$, $(\texttt{trees}_2.\texttt{x}, \texttt{trees}_2.\texttt{y})$ and $(\texttt{trees}_3.\texttt{x}, \texttt{trees}_3.\texttt{y})$ are aligned. It corresponds to the development of the expression:

$$\begin{vmatrix} \texttt{trees}_1.\texttt{x} & \texttt{trees}_2.\texttt{y} & 1 \\ \texttt{trees}_2.\texttt{x} & \texttt{trees}_2.\texttt{y} & 1 \\ \texttt{trees}_3.\texttt{x} & \texttt{trees}_3.\texttt{y} & 1 \end{vmatrix} = 0$$

5.311 ORDER

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Derived from SORT_PERMUTATION

Constraint ORDER(VECTORS, PERMUTATION)

Type

VECTOR : collection(var−dvar)

Arguments

VECTORS : collection(vec − VECTOR)
PERMUTATION : collection(var−dvar)

Restrictions

$|\mathtt{VECTOR}| \geq 1$
$|\mathtt{VECTORS}| \geq 1$
required(VECTORS, vec)
same_size(VECTORS, vec)
required(PERMUTATION, var)
PERMUTATION.var ≥ 1
PERMUTATION.var $\leq$ |PERMUTATION|
|PERMUTATION| = |VECTORS|

Purpose Given a collection of distinct VECTORS, enforces PERMUTATION.var$[i]$ to be equal to the position of vector VECTORS.vec$[i]$ within the sorted vectors of the collection VECTORS.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{l}\mathtt{vec}-\langle 1,1,2,2\rangle,\\ \mathtt{vec}-\langle 2,1,2,1\rangle,\\ \mathtt{vec}-\langle 2,1,1,1\rangle,\\ \mathtt{vec}-\langle 1,1,1,2\rangle,\\ \mathtt{vec}-\langle 1,2,2,1\rangle,\\ \mathtt{vec}-\langle 1,1,1,1\rangle,\\ \mathtt{vec}-\langle 2,2,1,1\rangle,\\ \mathtt{vec}-\langle 2,1,1,2\rangle\end{array}\right\rangle,\\ \langle 3,7,5,2,4,1,8,6\rangle\end{array}\right)$$

The ORDER constraint holds since:

- The vector $\langle 1,1,2,2\rangle$ is in the third position of the sorted collection VECTORS,
- The vector $\langle 2,1,2,1\rangle$ is in the seventh position of the sorted collection VECTORS,
- The vector $\langle 2,1,1,1\rangle$ is in the fifth position of the sorted collection VECTORS,
- The vector $\langle 1,1,1,2\rangle$ is in the second position of the sorted collection VECTORS,
- The vector $\langle 1,2,2,1\rangle$ is in the fourth position of the sorted collection VECTORS,
- The vector $\langle 1,1,1,1\rangle$ is in the first position of the sorted collection VECTORS,
- The vector $\langle 2,2,1,1\rangle$ is in the eigth position of the sorted collection VECTORS,

- The vector $\langle 2, 1, 1, 2\rangle$ is in the sixth position of the sorted collection VECTORS.

Typical

$|\texttt{VECTOR}| > 1$
$|\texttt{VECTORS}| > 1$

Arg. properties

Functional dependency: PERMUTATION determined by VECTORS.

See also

common keyword: SORT_PERMUTATION *(sort, permutation)*.

Keywords

characteristic of a constraint: sort.

combinatorial object: permutation.

constraint type: predefined constraint.

modelling: functional dependency.

5.312 ORDERED_ATLEAST_NVECTOR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Conjoin ATLEAST_NVECTOR and LEX_CHAIN_LESSEQ.

Constraint ORDERED_ATLEAST_NVECTOR(NVEC, VECTORS)

Synonyms ORDERED_ATLEAST_NVECTORS, ORDERED_ATLEAST_NPOINT, ORDERED_ATLEAST_NPOINTS.

Type

VECTOR : collection(var−dvar)

Arguments

NVEC : dvar
VECTORS : collection(vec − VECTOR)

Restrictions

$|\text{VECTOR}| \geq 1$
$\text{NVEC} \geq 0$
$\text{NVEC} \leq |\text{VECTORS}|$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose

Enforces the following two conditions:

1. The number of distinct tuples of values taken by the vectors of the collection VECTORS is greater than or equal to NVEC. Two tuples of values $\langle A_1, A_2, \ldots, A_m\rangle$ and $\langle B_1, B_2, \ldots, B_m\rangle$ are *distinct* if and only if there exist an integer $i \in [1, m]$ such that $A_i \neq B_i$.
2. For each pair of consecutive vectors VECTOR_i and VECTOR_{i+1} of the VECTORS collection we have that VECTOR_i is lexicographically less than or equal to VECTOR_{i+1}. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically less than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is lexicographically less than or equal to $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$$\left(2, \left\langle \begin{array}{l} \texttt{vec} - \langle 5,6\rangle, \\ \texttt{vec} - \langle 5,6\rangle, \\ \texttt{vec} - \langle 5,6\rangle, \\ \texttt{vec} - \langle 9,3\rangle, \\ \texttt{vec} - \langle 9,4\rangle \end{array} \right\rangle \right)$$

The ORDERED_ATLEAST_NVECTOR constraint holds since:

1. The collection VECTORS involves at least 2 distinct tuples of values (i.e., in fact the 3 distinct tuples $\langle 5,6\rangle$, $\langle 9,3\rangle$ and $\langle 9,4\rangle$).
2. The vectors of the collection VECTORS are sorted in increasing lexicographical order.

Typical

$|\texttt{VECTOR}| > 1$
$\texttt{NVEC} > 0$
$\texttt{NVEC} < |\texttt{VECTORS}|$
$|\texttt{VECTORS}| > 1$

Symmetry

NVEC can be decreased to any value ≥ 0.

Reformulation

The ORDERED_ATLEAST_NVECTOR constraint can be reformulated as a conjunction of a ATLEAST_NVECTOR and a LEX_CHAIN_LESSEQ constraints.

See also

common keyword: NVECTOR *(vector)*.

comparison swapped: ORDERED_ATMOST_NVECTOR.

implied by: ORDERED_NVECTOR *($\geq$ NVEC replaced by $=$ NVEC)*.

implies: ATLEAST_NVECTOR, LEX_CHAIN_LESSEQ *(NVEC of constraint ORDERED_ATLEAST_NVECTOR removed)*.

used in graph description: LEX_LESS, LEX_LESSEQ.

Keywords

characteristic of a constraint: vector.

constraint type: counting constraint, order constraint.

symmetry: symmetry.

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESSEQ(vectors1.vec, vectors2.vec)
Graph property(ies)	**NARC** $= \|$VECTORS$\| - 1$

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESS(vectors1.vec, vectors2.vec)
Graph property(ies)	**NCC** $\geq$ NVEC

Graph model Parts (A) and (B) of Figure 5.665 respectively show the initial and final graph of the second graph constraint associated with the **Example** slot. Since we use the **NCC** graph property in this second graph constraint, we show the different connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 3 following tuple of values $\langle 5, 6\rangle$, $\langle 9, 3\rangle$ and $\langle 9, 4\rangle$ are used by the vectors of the VECTORS collection.

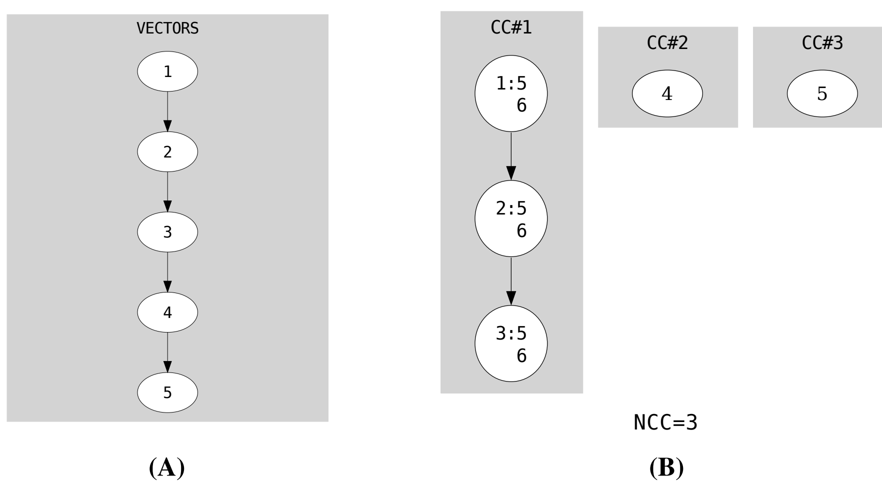


Figure 5.665: Initial and final graph of the ORDERED_ATLEAST_NVECTOR constraint

5.313 ORDERED_ATMOST_NVECTOR

DESCRIPTION LINKS GRAPH

Origin

Conjoin ATMOST_NVECTOR and LEX_CHAIN_LESSEQ.

Constraint

ORDERED_ATMOST_NVECTOR(NVEC, VECTORS)

Synonyms

ORDERED_ATMOST_NVECTORS, ORDERED_ATMOST_NPOINT, ORDERED_ATMOST_NPOINTS.

Type

VECTOR : collection(var−dvar)

Arguments

NVEC : dvar
VECTORS : collection(vec − VECTOR)

Restrictions

$|\mathtt{VECTOR}| \geq 1$
$\mathtt{NVEC} \geq \mathtt{min}(1, |\mathtt{VECTORS}|)$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose

Enforces the following two conditions:

1. The number of distinct tuples of values taken by the vectors of the collection VECTORS is less than or equal to NVEC. Two tuples of values $\langle A_1, A_2, \ldots, A_m\rangle$ and $\langle B_1, B_2, \ldots, B_m\rangle$ are *distinct* if and only if there exist an integer $i \in [1, m]$ such that $A_i \neq B_i$.
2. For each pair of consecutive vectors $\mathtt{VECTOR}_i$ and $\mathtt{VECTOR}_{i+1}$ of the VECTORS collection we have that $\mathtt{VECTOR}_i$ is lexicographically less than or equal to $\mathtt{VECTOR}_{i+1}$. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically less than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is lexicographically less than or equal to $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$$\left(3, \left\langle \begin{array}{l} \mathtt{vec} - \langle 5, 6\rangle, \\ \mathtt{vec} - \langle 5, 6\rangle, \\ \mathtt{vec} - \langle 5, 6\rangle, \\ \mathtt{vec} - \langle 9, 3\rangle, \\ \mathtt{vec} - \langle 9, 3\rangle \end{array} \right\rangle\right)$$

The ORDERED_ATMOST_NVECTOR constraint holds since:

1. The collection VECTORS involves at most 3 distinct tuples of values (i.e., in fact the 2 distinct tuples $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$).
2. The vectors of the collection VECTORS are sorted in increasing lexicographical order.

Typical

$|\mathtt{VECTOR}| > 1$
$\mathtt{NVEC} > 1$
$\mathtt{NVEC} < |\mathtt{VECTORS}|$
$|\mathtt{VECTORS}| > 1$

Symmetry

NVEC can be increased.

Arg. properties

Contractible wrt. VECTORS.

Reformulation

The ORDERED_ATMOST_NVECTOR constraint can be reformulated as a conjunction of a ATMOST_NVECTOR and a LEX_CHAIN_LESSEQ constraints.

See also

common keyword: NVECTOR *(vector)*.

comparison swapped: ORDERED_ATLEAST_NVECTOR.

implied by: ORDERED_NVECTOR *($\leq$ NVEC replaced by $=$ NVEC)*.

implies: ATMOST_NVECTOR, LEX_CHAIN_LESSEQ *(NVEC of constraint ORDERED_ATMOST_NVECTOR removed)*.

used in graph description: LEX_LESS, LEX_LESSEQ.

Keywords

characteristic of a constraint: vector.

constraint type: counting constraint, order constraint.

symmetry: symmetry.

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESSEQ(vectors1.vec, vectors2.vec)
Graph property(ies)	**NARC**$= \vert$VECTORS$\vert - 1$

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESS(vectors1.vec, vectors2.vec)
Graph property(ies)	**NCC**$\leq$ NVEC

Graph model Parts (A) and (B) of Figure 5.666 respectively show the initial and final graph of the second graph constraint associated with the **Example** slot. Since we use the **NCC** graph property in this second graph constraint, we show the different connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 2 following tuple of values $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$ are used by the vectors of the VECTORS collection.

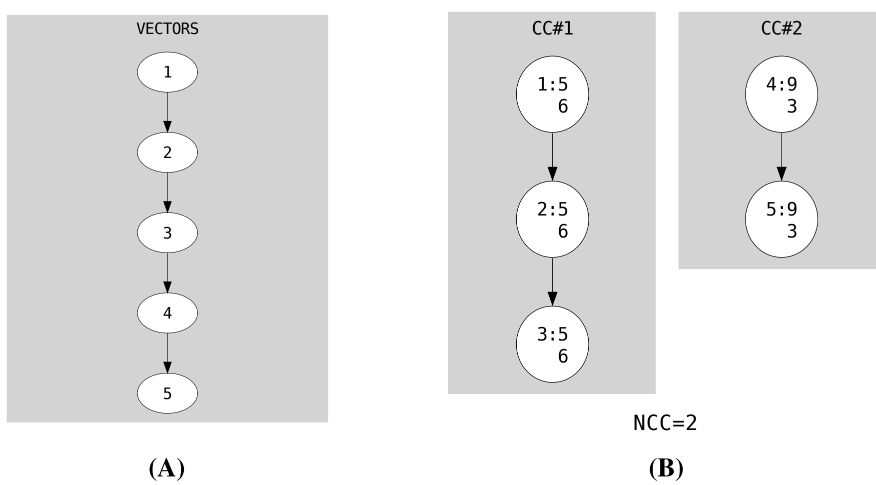


Figure 5.666: Initial and final graph of the ORDERED_ATMOST_NVECTOR constraint

5.314 ORDERED_GLOBAL_CARDINALITY

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [323]

Constraint ORDERED_GLOBAL_CARDINALITY(VARIABLES, VALUES)

Usual name ORDGCC

Synonym ORDERED_GCC.

Arguments

VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int, omax−int)

Restrictions

required(VARIABLES, var)
|VALUES| > 0
required(VALUES, [val, omax])
increasing_seq(VALUES, [val])
VALUES.omax ≥ 0
VALUES.omax ≤ |VARIABLES|

Purpose

For each $i \in [1, |\mathtt{VALUES}|]$, the values of the corresponding set of values VALUES[j].val ($i \leq j \leq |\mathtt{VALUES}|$) should be taken by at most VALUES[i].omax variables of the VARIABLES collection.
From that previous definition, the omax attributes are decreasing.

Example

$$\left(\begin{array}{l}\langle 2,0,1,0,0\rangle,\\ \langle \mathtt{val}-0\ \mathtt{omax}-5, \mathtt{val}-1\ \mathtt{omax}-3, \mathtt{val}-2\ \mathtt{omax}-1\rangle\end{array}\right)$$

The ORDERED_GLOBAL_CARDINALITY constraint holds since the values of the three sets of values $\{0,1,2\}$, $\{1,2\}$ and $\{2\}$ are respectively used no more than 5, 3 and 1 times within the collection $\langle 2,0,1,0,0\rangle$.

Symmetry Items of VARIABLES are permutable.

Arg. properties Contractible wrt. VALUES.

Usage

The ORDERED_GLOBAL_CARDINALITY can be used in order to restrict the way we assign the values of the VALUES collection to the variables of the VARIABLES collection. It expresses the fact that, when we use a value v, we implicitly also use all values that are less than or equal to v. As depicted by Figure 5.667 this is the case, for example, for a *soft cumulative* constraint where we want to control the shape of cumulative profile by providing for each instant i a variable h_i that gives the height of the cumulative profile at instant i. These variables h_i are passed as the first argument of the ORDERED_GLOBAL_CARDINALITY constraint. Then the omax attribute of the j-th item of the VALUES collection gives the maximum number of instants for which the height of the cumulative profile is greater than or equal to value VALUES[j].val. In Figure 5.667 we should have:

- no more than 1 height variable greater than or equal to 2,
- no more than 3 height variables greater than or equal to 1,
- no more than 5 height variables greater than or equal to 0.

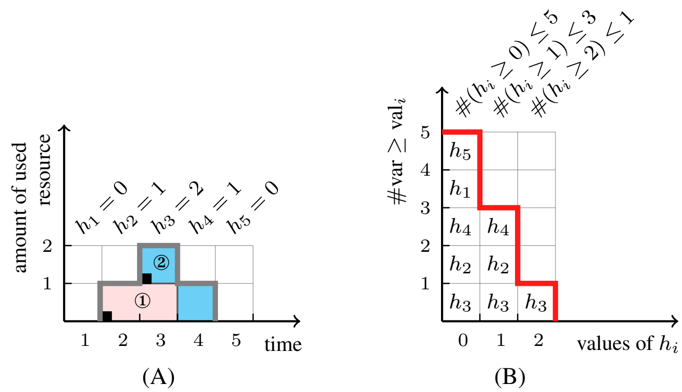


Figure 5.667: (A) A cumulative profile wrt two tasks ① and ②, and its corresponding height variables h_1, $h_2, \ldots, h_5$ giving at each instant how many resource is used (B) profile of value utilisation of the height variables (e.g., value 1 is assigned to variables h_3, h_2, h_4 and therefore used three times)

Remark

The original definition of the ORDERED_GLOBAL_CARDINALITY constraint mentions a third argument, namely the minimum number of occurrences of the smallest value. We omit it since it is redundant.

An other closely related constraint, the COST_ORDERED_GLOBAL_CARDINALITY constraint was introduced in [323] in order to model the fact that overloads costs may depend of the instant where they occur.

Algorithm

A filtering algorithm achieving arc-consistency in $O(|\texttt{VARIABLES}|+|\texttt{VALUES}|)$ is described in [323]. It is based on the equivalence between the following two statements:

1. the ORDERED_GLOBAL_CARDINALITY constraint has a solution,
2. all variables of the VARIABLES collection assigned to their respective minimum values correspond to a solution to the ORDERED_GLOBAL_CARDINALITY constraint.

Reformulation

The ORDERED_GLOBAL_CARDINALITY($\langle \texttt{var}-V_1, \texttt{var}-V_2, \ldots, \texttt{var}-V_{|\texttt{VARIABLES}|}\rangle$, $\langle \texttt{val}-v_1\ \texttt{omax}-o_1, \texttt{val}-v_2\ \texttt{omax}-o_2, \ldots, \texttt{val}-v_{|\texttt{VALUES}|}\ \texttt{omax}-o_{|\texttt{VALUES}|}\rangle$) constraint can be reformulated into a GLOBAL_CARDINALITY($\langle \texttt{var}-V_1, \texttt{var}-V_2, \ldots, \texttt{var}-V_{|\texttt{VARIABLES}|}\rangle$, $\langle \texttt{val}-v_1\ \texttt{noccurrence}-N_1, \texttt{val}-v_2\ \texttt{noccurrence}-N_2, \ldots, \texttt{val}-v_{|\texttt{VALUES}|}\ \texttt{noccurrence}-N_{|\texttt{VALUES}|}\rangle$) and $|\texttt{VALUES}|$ sliding linear inequalities constraints of the form:

$$\begin{aligned} N_1 + N_2 + \cdots + N_{|\texttt{VALUES}|} &\leq o_1, \\ N_2 + \cdots + N_{|\texttt{VALUES}|} &\leq o_2, \\ \cdots\cdots\cdots\cdots\cdots\cdots &, \\ N_{|\texttt{VALUES}|} &\leq o_{|\texttt{VALUES}|}. \end{aligned}$$

However, with the next example, T. Petit and J.-C. Régin have shown that this reformulation hinders propagation:

1. $V_1 \in \{0,1\}$, $V_2 \in \{0,1\}$, $V_3 \in \{0,1,2\}$, $V_4 \in \{2,3\}$, $V_5 \in \{2,3\}$.
2. GLOBAL_CARDINALITY($\langle V_1, V_2, V_3, V_4, V_5 \rangle$, $\langle \texttt{val} - 1\ \texttt{noccurrence} - N_1, \texttt{val} - 2\ \texttt{noccurrence} - N_2, \texttt{val} - 3\ \texttt{noccurrence} - N_3 \rangle$),
3. $N_1 + N_2 + N_3 \leq 3 \wedge N_2 + N_3 \leq 2 \wedge N_3 \leq 2$.

The previous reformulation does not remove value 2 from the domain of variable V_3.

See also

related: CUMULATIVE *(controlling the shape of the cumulative profile for breaking symmetry)*, GLOBAL_CARDINALITY_LOW_UP, INCREASING_GLOBAL_CARDINALITY *(the order is imposed on the main variables, and not on the count variables)*.

root concept: GLOBAL_CARDINALITY.

Keywords

application area: assignment.

constraint type: value constraint, order constraint.

filtering: arc-consistency.

For all items of VALUES:

Arc input(s) VARIABLES

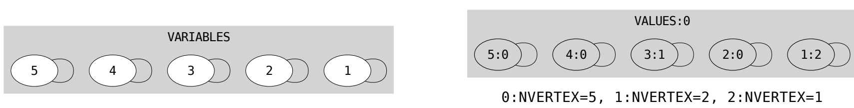
Arc generator $SELF \mapsto$collection(variables)

Arc arity 1

Arc constraint(s) variables.var $\geq$ VALUES.val

Graph property(ies) **NVERTEX** $\leq$ VALUES.omax

Graph model Since we want to express one unary constraint for each value we use the "For all items of VALUES" iterator. Part (A) of Figure 5.668 shows the initial graphs associated with each value 0, 1 and 2 of the VALUES collection of the **Example** slot. Part (B) of Figure 5.668 shows the corresponding final graph associated with value 0. Since we use the **NVERTEX** graph property, the vertices of the final graph is stressed in bold.



(A) **(B)**

Figure 5.668: Initial and final graph of the ORDERED_GLOBAL_CARDINALITY constraint

5.315 ORDERED_NVECTOR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from NVECTOR.

Constraint ORDERED_NVECTOR(NVEC, VECTORS)

Synonyms ORDERED_NVECTORS, ORDERED_NPOINT, ORDERED_NPOINTS.

Type

VECTOR : collection(var−dvar)

Arguments

NVEC : dvar
VECTORS : collection(vec − VECTOR)

Restrictions

$|\texttt{VECTOR}| \geq 1$
$\texttt{NVEC} \geq \texttt{min}(1, |\texttt{VECTORS}|)$
$\texttt{NVEC} \leq |\texttt{VECTORS}|$
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose

Enforces the following two conditions:

1. NVEC is the number of distinct tuples of values assigned to the vectors of the collection VECTORS. Two tuples of values $\langle A_1, A_2, \ldots, A_m\rangle$ and $\langle B_1, B_2, \ldots, B_m\rangle$ are *distinct* if and only if there exist an integer $i \in [1, m]$ such that $A_i \neq B_i$.
2. For each pair of consecutive vectors $\texttt{VECTOR}_i$ and $\texttt{VECTOR}_{i+1}$ of the VECTORS collection we have that $\texttt{VECTOR}_i$ is lexicographically less than or equal to $\texttt{VECTOR}_{i+1}$. Given two vectors, $\vec{X}$ and $\vec{Y}$ of n components, $\langle X_0, \ldots, X_{n-1}\rangle$ and $\langle Y_0, \ldots, Y_{n-1}\rangle$, $\vec{X}$ is *lexicographically less than or equal to* $\vec{Y}$ if and only if $n = 0$ or $X_0 < Y_0$ or $X_0 = Y_0$ and $\langle X_1, \ldots, X_{n-1}\rangle$ is lexicographically less than or equal to $\langle Y_1, \ldots, Y_{n-1}\rangle$.

Example

$$\left(2, \left\langle \begin{array}{l} \texttt{vec} - \langle 5, 6\rangle, \\ \texttt{vec} - \langle 5, 6\rangle, \\ \texttt{vec} - \langle 5, 6\rangle, \\ \texttt{vec} - \langle 9, 3\rangle, \\ \texttt{vec} - \langle 9, 3\rangle \end{array} \right\rangle \right)$$

The ORDERED_NVECTOR constraint holds since:

1. Its first argument $\texttt{NVEC} = 2$ is set to the number of distinct tuples of values (i.e., tuples $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$) occurring within the collection VECTORS.
2. The vectors of the collection VECTORS are sorted in increasing lexicographical order.

Typical

$|\texttt{VECTOR}| > 1$
$\texttt{NVEC} > 1$
$\texttt{NVEC} < |\texttt{VECTORS}|$
$|\texttt{VECTORS}| > 1$

Arg. properties

- Functional dependency: NVEC determined by VECTORS.
- Contractible wrt. VECTORS when $\texttt{NVEC} = 1$ and $|\texttt{VECTORS}| > 0$.
- Contractible wrt. VECTORS when $\texttt{NVEC} = |\texttt{VECTORS}|$.

Reformulation

The ORDERED_NVECTOR constraint can be reformulated as a conjunction of a NVECTOR and a LEX_CHAIN_LESSEQ constraints.

See also

implies: LEX_CHAIN_LESSEQ *(*NVEC *of constraint* ORDERED_NVECTOR *removed)*, NVECTOR, ORDERED_ATLEAST_NVECTOR *(*$=$ NVEC *replaced by* $\geq$ NVEC*)*, ORDERED_ATMOST_NVECTOR *(*$=$ NVEC *replaced by* $\leq$ NVEC*)*.

related: INCREASING_NVALUE_CHAIN.

root concept: INCREASING_NVALUE.

used in graph description: LEX_LESS, LEX_LESSEQ.

Keywords

characteristic of a constraint: vector.

constraint type: counting constraint, order constraint.

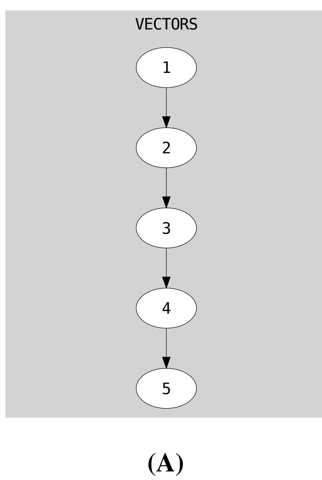
modelling: functional dependency.

symmetry: symmetry.

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESSEQ(vectors1.vec, vectors2.vec)
Graph property(ies)	**NARC**$=$ |VECTORS| -1

Arc input(s)	VECTORS
Arc generator	$PATH \mapsto$ collection(vectors1, vectors2)
Arc arity	2
Arc constraint(s)	LEX_LESS(vectors1.vec, vectors2.vec)
Graph property(ies)	**NCC**$=$ NVEC

Graph model Parts (A) and (B) of Figure 5.669 respectively show the initial and final graph of the second graph constraint associated with the **Example** slot. Since we use the **NCC** graph property in this second graph constraint, we show the different connected components of the final graph. Each strongly connected component corresponds to a tuple of values that is assigned to some vectors of the VECTORS collection. The 2 following tuple of values $\langle 5, 6\rangle$ and $\langle 9, 3\rangle$ are used by the vectors of the VECTORS collection.



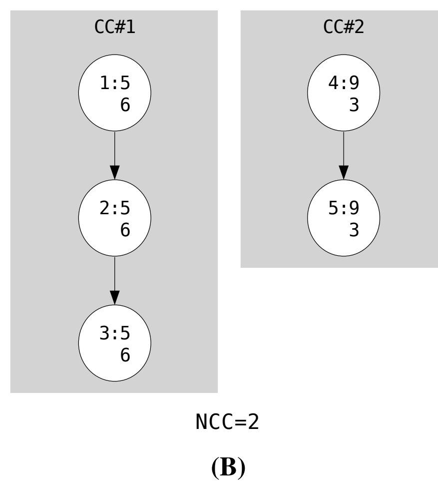


Figure 5.669: Initial and final graph of the ORDERED_NVECTOR constraint

5.316 ORTH_LINK_ORI_SIZ_END

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Used by several constraints between orthotopes

Constraint ORTH_LINK_ORI_SIZ_END(ORTHOTOPE)

Argument ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Restrictions
|ORTHOTOPE| >0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori $\leq$ ORTHOTOPE.end

Purpose Enforce for each item of the ORTHOTOPE collection the constraint ori + siz = end.

Example (⟨ori − 2 siz − 2 end − 4, ori − 1 siz − 3 end − 4⟩)

The ORTH_LINK_ORI_SIZ_END constraint holds since the two items ⟨ori − 2 siz − 2 end − 4⟩ and ⟨ori − 1 siz − 3 end − 4⟩ respectively verify the conditions $2+2=4$ and $1+3=4$.

Typical
|ORTHOTOPE| >1
ORTHOTOPE.siz >0

Symmetries
- Items of ORTHOTOPE are permutable.
- One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPE.
- One and the same constant can be added to the siz and end attributes of all items of ORTHOTOPE.

Arg. properties
- Functional dependency: ORTHOTOPE.ori determined by ORTHOTOPE.siz and ORTHOTOPE.end.
- Functional dependency: ORTHOTOPE.siz determined by ORTHOTOPE.ori and ORTHOTOPE.end.
- Functional dependency: ORTHOTOPE.end determined by ORTHOTOPE.ori and ORTHOTOPE.siz.
- Contractible wrt. ORTHOTOPE.

Usage Used in the **Arc constraint(s)** slot for defining some constraints like DIFFN, PLACE_IN_PYRAMID or ORTHS_ARE_CONNECTED.

Used in DIFFN, ORTH_ON_THE_GROUND, ORTH_ON_TOP_OF_ORTH, ORTHS_ARE_CONNECTED, TWO_ORTH_ARE_IN_CONTACT, TWO_ORTH_COLUMN, TWO_ORTH_DO_NOT_OVERLAP, TWO_ORTH_INCLUDE.

Keywords **constraint arguments:** pure functional dependency.

constraint type: decomposition.

geometry: orthotope.

modelling: functional dependency.

Arc input(s)	ORTHOTOPE
Arc generator	$SELF \mapsto$ collection(orthotope)
Arc arity	1
Arc constraint(s)	orthotope.ori + orthotope.siz = orthotope.end
Graph property(ies)	$\mathbf{NARC}$= |ORTHOTOPE|

Graph model Parts (A) and (B) of Figure 5.670 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{NARC}$ graph property, the loops of the final graph are stressed in bold.

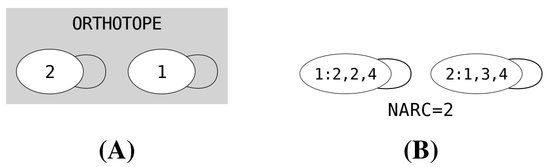


Figure 5.670: Initial and final graph of the ORTH_LINK_ORI_SIZ_END constraint

Signature Since we use the $SELF$ arc generator on the ORTHOTOPE collection the number of arcs of the initial graph is equal to |ORTHOTOPE|. Therefore the maximum number of arcs of the final graph is also equal to |ORTHOTOPE|. For this reason we can rewrite the graph property $\mathbf{NARC}$ = |ORTHOTOPE| to $\mathbf{NARC} \geq$ |ORTHOTOPE| and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.317 ORTH_ON_THE_GROUND

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Used for defining PLACE_IN_PYRAMID.

Constraint ORTH_ON_THE_GROUND(ORTHOTOPE, VERTICAL_DIM)

Arguments

ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)
VERTICAL_DIM : int

Restrictions

|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori $\leq$ ORTHOTOPE.end
VERTICAL_DIM ≥ 1
VERTICAL_DIM $\leq$ |ORTHOTOPE|
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE)

Purpose The ori attribute of the VERTICAL_DIMth item of the ORTHOTOPES collection should be fixed to one.

Example (⟨ori − 1 siz − 2 end − 3, ori − 2 siz − 3 end − 5⟩ , 1)

The ORTH_ON_THE_GROUND constraint holds since the ori attribute of its 1^{th} item ⟨ori − 1 siz − 2 end − 3⟩ (i.e., 1^{th} item since VERTICAL_DIM $= 1$) is set to one.

Typical

|ORTHOTOPE| > 1
ORTHOTOPE.siz > 0

Used in PLACE_IN_PYRAMID.

Keywords **geometry:** geometrical constraint, orthotope.

Arc input(s)	ORTHOTOPE
Arc generator	*SELF* $\mapsto$ collection(orthotope)
Arc arity	1
Arc constraint(s)	• orthotope.key = VERTICAL_DIM • orthotope.ori = 1
Graph property(ies)	**NARC** = 1

Graph model Parts (A) and (B) of Figure 5.671 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the loop of the final graph is stressed in bold.

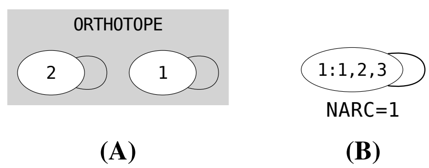


Figure 5.671: Initial and final graph of the ORTH_ON_THE_GROUND constraint

Signature Since all the key attributes of the ORTHOTOPES collection are distinct, because of the first condition of the arc constraint, and since we use the *SELF* arc generator the final graph contains at most one arc. Therefore we can rewrite the graph property **NARC** $=1$ to **NARC** ≥ 1 and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.318 ORTH_ON_TOP_OF_ORTH

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Used for defining PLACE_IN_PYRAMID.

Constraint ORTH_ON_TOP_OF_ORTH(ORTHOTOPE1, ORTHOTOPE2, VERTICAL_DIM)

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments

ORTHOTOPE1 : ORTHOTOPE
ORTHOTOPE2 : ORTHOTOPE
VERTICAL_DIM : int

Restrictions

$|\texttt{ORTHOTOPE}| > 0$
require_at_least(2, ORTHOTOPE, [ori, siz, end])
$\texttt{ORTHOTOPE.siz} \geq 0$
$\texttt{ORTHOTOPE.ori} \leq \texttt{ORTHOTOPE.end}$
$|\texttt{ORTHOTOPE1}| = |\texttt{ORTHOTOPE2}|$
$\texttt{VERTICAL\_DIM} \geq 1$
$\texttt{VERTICAL\_DIM} \leq |\texttt{ORTHOTOPE1}|$
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE1)
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE2)

Purpose

ORTHOTOPE1 is located on top of ORTHOTOPE2 which concretely means:

- In each dimension different from VERTICAL_DIM the projection of ORTHOTOPE1 is included in the projection of ORTHOTOPE2.
- In the dimension VERTICAL_DIM the origin of ORTHOTOPE1 coincide with the end of ORTHOTOPE2.

Example

$$\left(\begin{array}{l} \langle \texttt{ori}-5\ \texttt{siz}-2\ \texttt{end}-7, \texttt{ori}-3\ \texttt{siz}-3\ \texttt{end}-6 \rangle, \\ \langle \texttt{ori}-3\ \texttt{siz}-5\ \texttt{end}-8, \texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3 \rangle, 2 \end{array} \right)$$

As illustrated by Figure 5.672 the orthotope ORTHOTOPE1 (rectangle R1 coloured in pink) is on top of ORTHOTOPE2 (rectangle R2 coloured in blue) according to the hypothesis that the vertical dimension corresponds to dimension 2 (i.e., $\texttt{VERTICAL\_DIM} = 2$). This stands from the fact that the following conditions hold:

- $\texttt{ORTHOTOPE2}[2].\texttt{ori} + \texttt{ORTHOTOPE2}[2].\texttt{siz} = 1 + 2 = \texttt{ORTHOTOPE1}[2].\texttt{ori}$,
- $\texttt{ORTHOTOPE2}[1].\texttt{ori} = 3 \leq \texttt{ORTHOTOPE1}[1].\texttt{ori} = 5$,
- $\texttt{ORTHOTOPE1}[1].\texttt{end} = 7 \leq \texttt{ORTHOTOPE2}[1].\texttt{end} = 8$.

Consequently, the ORTH_ON_TOP_OF_ORTH constraint holds.

Typical

$|\texttt{ORTHOTOPE}| > 1$
$\texttt{ORTHOTOPE.siz} > 0$

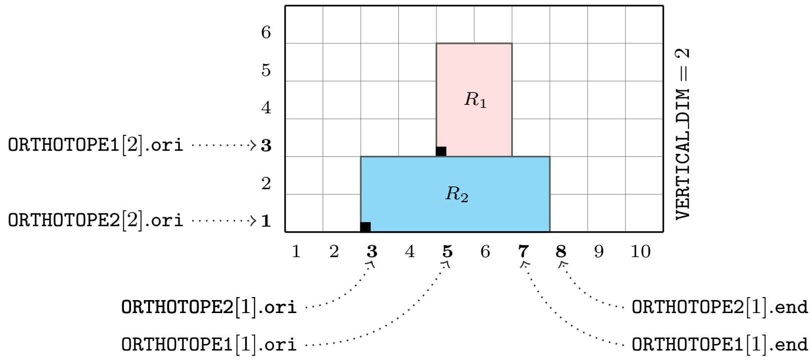


Figure 5.672: Illustration of the relation *on top of* of the **Example** slot (R_1 on top of R_2 wrt dimension VERTICAL_DIM $= 2$)

Used in PLACE_IN_PYRAMID.

Keywords **constraint type:** logic.

geometry: geometrical constraint, non-overlapping, orthotope.

Arc input(s)	ORTHOTOPE1 ORTHOTOPE2
Arc generator	$PRODUCT(=) \mapsto$collection(orthotope1, orthotope2)
Arc arity	2
Arc constraint(s)	• orthotope1.key $\neq$ VERTICAL_DIM • orthotope2.ori $\leq$ orthotope1.ori • orthotope1.end $\leq$ orthotope2.end
Graph property(ies)	**NARC**$= \lvert$ORTHOTOPE1$\rvert - 1$

Arc input(s)	ORTHOTOPE1 ORTHOTOPE2
Arc generator	$PRODUCT(=) \mapsto$collection(orthotope1, orthotope2)
Arc arity	2
Arc constraint(s)	• orthotope1.key $=$ VERTICAL_DIM • orthotope1.ori $=$ orthotope2.end
Graph property(ies)	**NARC**$= 1$

Graph model The first and second graph constraints respectively express the first and second conditions stated in the **Purpose** slot defining the ORTH_ON_TOP_OF_ORTH constraint.

Parts (A) and (B) of Figure 5.673 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

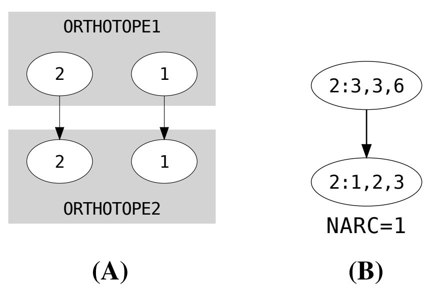


Figure 5.673: Initial and final graph of the ORTH_ON_TOP_OF_ORTH constraint

Signature Consider the second graph constraint. Since all the key attributes of the ORTHOTOPE1 collection are distinct, because of the arc constraint orthotope1.key $=$ VERTICAL_DIM, and since we use the $PRODUCT(=)$ arc generator the final graph contains at most one arc. Therefore we can rewrite the graph property **NARC** $= 1$ to **NARC** ≥ 1 and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

5.319 ORTHS_ARE_CONNECTED

DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint ORTHS_ARE_CONNECTED(ORTHOTOPES)

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Argument ORTHOTOPES : collection(orth − ORTHOTOPE)

Restrictions

$|\text{ORTHOTOPE}| > 0$
require_at_least(2, ORTHOTOPE, [ori, siz, end])
$\text{ORTHOTOPE.siz} > 0$
$\text{ORTHOTOPE.ori} \leq \text{ORTHOTOPE.end}$
required(ORTHOTOPES, orth)
same_size(ORTHOTOPES, orth)

Purpose

There should be a single group of connected orthotopes. Two orthotopes touch each other (i.e., are connected) if they overlap in all dimensions except one, and if, for the dimension where they do not overlap, the distance between the two orthotopes is equal to 0.

Example

$$\left(\left\langle\begin{array}{l}\text{orth} - \langle\text{ori} - 2\ \text{siz} - 4\ \text{end} - 6, \text{ori} - 2\ \text{siz} - 2\ \text{end} - 4\rangle,\\ \text{orth} - \langle\text{ori} - 1\ \text{siz} - 2\ \text{end} - 3, \text{ori} - 4\ \text{siz} - 3\ \text{end} - 7\rangle,\\ \text{orth} - \langle\text{ori} - 6\ \text{siz} - 3\ \text{end} - 9, \text{ori} - 1\ \text{siz} - 2\ \text{end} - 3\rangle,\\ \text{orth} - \langle\text{ori} - 6\ \text{siz} - 2\ \text{end} - 8, \text{ori} - 3\ \text{siz} - 2\ \text{end} - 5\rangle\end{array}\right\rangle\right)$$

Figure 5.674 shows the rectangles associated with the example. One can note that:

- Rectangle 2 touch rectangle 1,
- Rectangle 1 touch rectangle 2, rectangle 3 and rectangle 4,
- Rectangle 4 touch rectangle 1 and rectangle 3,
- Rectangle 3 touch rectangle 1 and rectangle 4.

Consequently, since we have a single group of connected rectangles, the ORTHS_ARE_CONNECTED constraint holds.

Typical

$|\text{ORTHOTOPE}| > 1$
$|\text{ORTHOTOPES}| > 1$

Symmetries

- Items of ORTHOTOPES are permutable.
- Items of ORTHOTOPES.orth are permutable (*same permutation used*).
- One and the same constant can be added to the ori and end attributes of all items of ORTHOTOPES.orth.

ORTHOTOPES (rectangles)

R_1:	$\langle \texttt{ori}-2\ \texttt{siz}-4\ \texttt{end}-6, \texttt{ori}-2\ \texttt{siz}-2\ \texttt{end}-4\rangle$
R_2:	$\langle \texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3, \texttt{ori}-4\ \texttt{siz}-3\ \texttt{end}-7\rangle$
R_3:	$\langle \texttt{ori}-6\ \texttt{siz}-3\ \texttt{end}-9, \texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3\rangle$
R_4:	$\langle \texttt{ori}-6\ \texttt{siz}-2\ \texttt{end}-8, \texttt{ori}-3\ \texttt{siz}-2\ \texttt{end}-5\rangle$

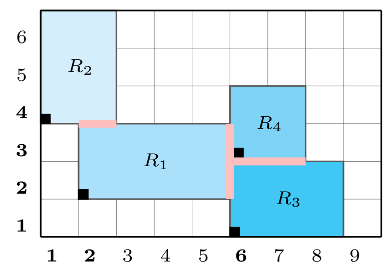


Figure 5.674: The four connected rectangles of the **Example** slot: contacts between rectangles are shown in pink

Usage In floor planning problem there is a typical constraint, that states that one should be able to access every room from any room.

See also **implies:** DIFFN.

used in graph description: ORTH_LINK_ORI_SIZ_END, TWO_ORTH_ARE_IN_CONTACT.

Keywords **geometry:** geometrical constraint, touch, contact, non-overlapping, orthotope.

Arc input(s)	ORTHOTOPES
Arc generator	$SELF \mapsto$ collection(orthotopes)
Arc arity	1
Arc constraint(s)	ORTH_LINK_ORI_SIZ_END(orthotopes.orth)
Graph property(ies)	**NARC**= |ORTHOTOPES|

Arc input(s)	ORTHOTOPES
Arc generator	$CLIQUE(\neq) \mapsto$ collection(orthotopes1, orthotopes2)
Arc arity	2
Arc constraint(s)	TWO_ORTH_ARE_IN_CONTACT(orthotopes1.orth, orthotopes2.orth)
Graph property(ies)	• **NVERTEX**= |ORTHOTOPES| • **NCC**= 1

Graph model Parts (A) and (B) of Figure 5.675 respectively show the initial and final graph associated with the **Example** slot.Since we use the **NVERTEX** graph property the vertices of the final graph are stressed in bold. Since we also use the **NCC** graph property we show the unique connected component of the final graph. An arc between two vertices indicates that two rectangles are in contact.

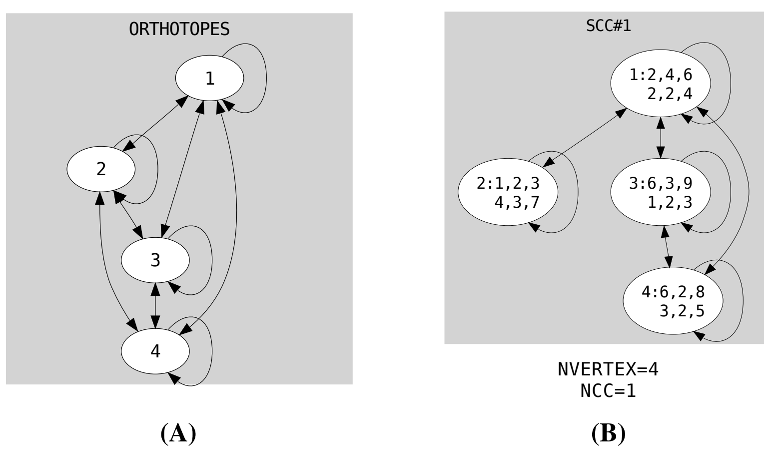


Figure 5.675: Initial and final graph of the ORTHS_ARE_CONNECTED constraint

Signature Since the first graph constraint uses the $SELF$ arc generator on the ORTHOTOPES collection the corresponding initial graph contains |ORTHOTOPES| arcs. Therefore the final graph of the first graph constraint contains at most |ORTHOTOPES| arcs and we can rewrite **NARC** = |ORTHOTOPES| to **NARC** $\geq$ |ORTHOTOPES|. So we can simplify $\overline{\textbf{NARC}}$ to $\underline{\textbf{NARC}}$.

Consider now the second graph constraint. Since its corresponding initial graph contains |ORTHOTOPES| vertices, its final graph has a maximum number of vertices also equal to $|\texttt{ORTHOTOPES}|$. Therefore we can rewrite $\mathbf{NVERTEX} = |\texttt{ORTHOTOPES}|$ to $\mathbf{NVERTEX} \geq |\texttt{ORTHOTOPES}|$ and simplify $\overline{\underline{\mathbf{NVERTEX}}}$ to $\overline{\mathbf{NVERTEX}}$. From the graph property $\mathbf{NVERTEX} = |\texttt{ORTHOTOPES}|$ and from the restriction $|\texttt{ORTHOTOPES}| > 0$ the final graph is not empty. Therefore it contains at least one connected component. So we can rewrite $\mathbf{NCC} = 1$ to $\mathbf{NCC} \leq 1$ and simplify $\overline{\underline{\mathbf{NCC}}}$ to $\underline{\mathbf{NCC}}$.

5.320 OVERLAP_SBOXES

▶ ▷ ◁ ◀ DESCRIPTION LINKS LOGIC

Origin Geometry, derived from [349]

Constraint OVERLAP_SBOXES(K, DIMS, OBJECTS, SBOXES)

Synonym OVERLAP.

Types

VARIABLES	:	collection(v−dvar)
INTEGERS	:	collection(v−int)
POSITIVES	:	collection(v−int)

Arguments

K	:	int
DIMS	:	sint
OBJECTS	:	collection(oid−int, sid−dvar, x − VARIABLES)
SBOXES	:	collection(sid−int, t − INTEGERS, l − POSITIVES)

Restrictions

$|\mathtt{VARIABLES}| \geq 1$
$|\mathtt{INTEGERS}| \geq 1$
$|\mathtt{POSITIVES}| \geq 1$
required(VARIABLES, v)
$|\mathtt{VARIABLES}| = \mathtt{K}$
required(INTEGERS, v)
$|\mathtt{INTEGERS}| = \mathtt{K}$
required(POSITIVES, v)
$|\mathtt{POSITIVES}| = \mathtt{K}$
$\mathtt{POSITIVES.v} > 0$
$\mathtt{K} > 0$
$\mathtt{DIMS} \geq 0$
$\mathtt{DIMS} < \mathtt{K}$
increasing_seq(OBJECTS, [oid])
required(OBJECTS, [oid, sid, x])
$\mathtt{OBJECTS.oid} \geq 1$
$\mathtt{OBJECTS.oid} \leq |\mathtt{OBJECTS}|$
$\mathtt{OBJECTS.sid} \geq 1$
$\mathtt{OBJECTS.sid} \leq |\mathtt{SBOXES}|$
$|\mathtt{SBOXES}| \geq 1$
required(SBOXES, [sid, t, l])
$\mathtt{SBOXES.sid} \geq 1$
$\mathtt{SBOXES.sid} \leq |\mathtt{SBOXES}|$
do_not_overlap(SBOXES)

Purpose

Holds if, for each pair of objects (O_i, O_j), $i < j$, O_i overlaps O_j with respect to a set of dimensions depicted by DIMS. O_i and O_j are objects that take a shape among a set of shapes. Each *shape* is defined as a finite set of shifted boxes, where each shifted box is described by a box in a K-dimensional space at a given offset (from the origin of the shape) with given sizes. More precisely, a *shifted box* is an entity defined by its shape id sid, shift offset t, and sizes l. Then, a shape is defined as the union of shifted boxes sharing the same shape id. An *object* is an entity defined by its unique object identifier oid, shape id sid and origin x.
An object O_i *overlaps* an object O_j with respect to a set of dimensions depicted by DIMS if and only if, there exists a shifted box s_i associated with O_i and there exists a shifted box s_j associated with O_j, such that (1) there exists a dimension $d \in$ DIMS where the end of O_i in dimension d is strictly greater than the start of O_j in dimension d, and (2) the end of O_j in dimension d is strictly greater than the start of O_i in dimension d.

Example

$$\left(\begin{array}{l} 2, \{0,1\}, \\ \left\langle\begin{array}{lll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 3,2\rangle, \\ \texttt{oid}-3 & \texttt{sid}-3 & \texttt{x}-\langle 2,4\rangle \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 4,5\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 3,3\rangle, \\ \texttt{sid}-3 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,1\rangle \end{array}\right\rangle \end{array}\right)$$

Figure 5.676 shows the objects of the example. Since O_1 overlaps both O_2 and O_3, and since O_2 overlaps O_3, the OVERLAP_SBOXES constraint holds.

Typical

$|\texttt{OBJECTS}| > 1$

Symmetries

- Items of OBJECTS are permutable.
- Items of SBOXES are permutable.
- Items of OBJECTS.x, SBOXES.t and SBOXES.l are permutable (*same permutation used*).
- SBOXES.l.v can be increased.

Arg. properties

Suffix-contractible wrt. OBJECTS.

Remark

One of the eight relations of the *Region Connection Calculus* [349].

See also

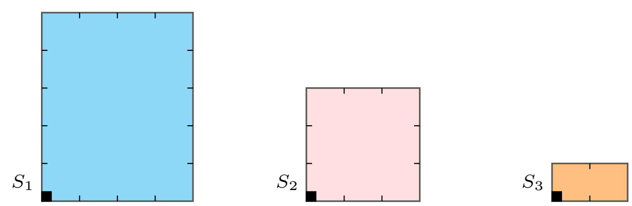
common keyword: CONTAINS_SBOXES, COVEREDBY_SBOXES, COVERS_SBOXES, DISJOINT_SBOXES, EQUAL_SBOXES, INSIDE_SBOXES, MEET_SBOXES (*rcc8*), NON_OVERLAP_SBOXES (*geometrical constraint,logic*).

Keywords

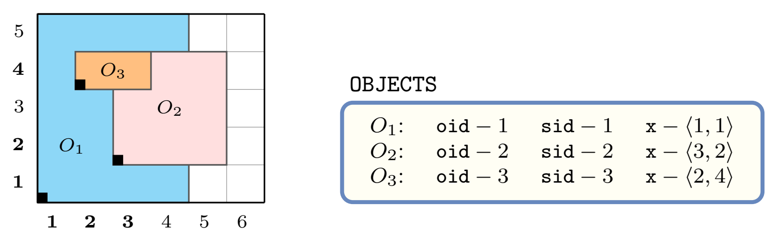
constraint type: logic.

geometry: geometrical constraint, rcc8.

miscellaneous: obscure.



(A) Shape of the first object (B) Shape of the second object (C) Shape of the third object



(D) Three objects where O_1 both overlaps O_2 and O_3 and where O_2 overlaps O_3

Figure 5.676: (D) the three mutually overlapping objects O_1, O_2, O_3 of the **Example** slot respectively assigned shapes S_1, S_2, S_3; (A), (B), (C) shapes S_1, S_2 and S_3 are made up from a single shifted box.

Logic

- $\texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D})$
- $\texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) \stackrel{\text{def}}{=} \texttt{O1.x}(\texttt{D}) + \texttt{S1.t}(\texttt{D}) + \texttt{S1.l}(\texttt{D})$
- $\texttt{overlap\_sboxes}(\texttt{Dims},\texttt{O1},\texttt{S1},\texttt{O2},\texttt{S2}) \stackrel{\text{def}}{=}$
  $\forall \texttt{D} \in \texttt{Dims}$
  $\bigwedge \left(\begin{array}{l} \texttt{end}(\texttt{O1},\texttt{S1},\texttt{D}) > \\ \texttt{origin}(\texttt{O2},\texttt{S2},\texttt{D}) \end{array}, \begin{array}{l} \texttt{end}(\texttt{O2},\texttt{S2},\texttt{D}) > \\ \texttt{origin}(\texttt{O1},\texttt{S1},\texttt{D}) \end{array} \right)$
- $\texttt{overlap\_objects}(\texttt{Dims},\texttt{O1},\texttt{O2}) \stackrel{\text{def}}{=}$
  $\forall \texttt{S1} \in \texttt{sboxes}([\texttt{O1.sid}])$
  $\exists \texttt{S2} \in \texttt{sboxes}\left(\left[\ \texttt{O2.sid} \ \right] \right)$
  $\texttt{overlap\_sboxes}\left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{S1}, \\ \texttt{O2}, \\ \texttt{S2} \end{array} \right)$
- $\texttt{all\_overlap}(\texttt{Dims},\texttt{OIDS}) \stackrel{\text{def}}{=}$
  $\forall \texttt{O1} \in \texttt{objects}(\texttt{OIDS})$
  $\forall \texttt{O2} \in \texttt{objects}(\texttt{OIDS})$
  $\begin{array}{l} \texttt{O1.oid} < \\ \texttt{O2.oid} \end{array} \Rightarrow$
  $\texttt{overlap\_objects}\left(\begin{array}{l} \texttt{Dims}, \\ \texttt{O1}, \\ \texttt{O2} \end{array} \right)$
- $\texttt{all\_overlap}(\texttt{DIMENSIONS},\texttt{OIDS})$

5.321 PATH

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from BINARY_TREE.

Constraint PATH(NPATH, NODES)

Arguments

NPATH : dvar
NODES : collection(index−int, succ−dvar)

Restrictions

$\texttt{NPATH} \geq 1$
$\texttt{NPATH} \leq |\texttt{NODES}|$
required(NODES, [index, succ])
$|\texttt{NODES}| > 0$
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
distinct(NODES, index)
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$

Purpose

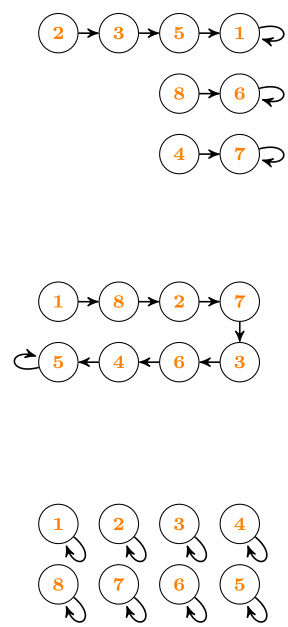
Cover the digraph G described by the NODES collection with NPATH paths in such a way that each vertex of G belongs to exactly one path.

Example

$$\left(\begin{array}{l} 3, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-3, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-7, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-6, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-6 \end{array} \right\rangle \end{array}\right)$$

$$\left(\begin{array}{l} 1, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-8, \\ \texttt{index}-2 & \texttt{succ}-7, \\ \texttt{index}-3 & \texttt{succ}-6, \\ \texttt{index}-4 & \texttt{succ}-5, \\ \texttt{index}-5 & \texttt{succ}-5, \\ \texttt{index}-6 & \texttt{succ}-4, \\ \texttt{index}-7 & \texttt{succ}-3, \\ \texttt{index}-8 & \texttt{succ}-2 \end{array} \right\rangle \end{array}\right)$$

$$\left(\begin{array}{l} 8, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-2, \\ \texttt{index}-3 & \texttt{succ}-3, \\ \texttt{index}-4 & \texttt{succ}-4, \\ \texttt{index}-5 & \texttt{succ}-5, \\ \texttt{index}-6 & \texttt{succ}-6, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-8 \end{array} \right\rangle \end{array}\right)$$



The first PATH constraint holds since its second argument corresponds to the 3 (i.e., the first argument of the PATH constraint) paths depicted by Figure 5.677.

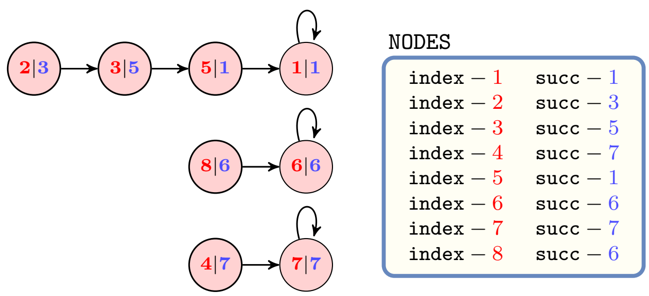


Figure 5.677: The three paths corresponding to the first example of the **Example** slot; each vertex contains the information `index`|`succ` where `succ` is the index of its successor in the path (by convention one of the extremities of a path points to itself).

All solutions

Figure 5.678 gives all solutions to the following non ground instance of the PATH constraint: NPATH $\in \{1,3\}$, $\mathtt{S}_1 \in [3,4]$, $\mathtt{S}_2 \in [1,2]$, $\mathtt{S}_3 \in [1,3]$, $\mathtt{S}_4 \in [2,4]$, PATH(NPATH, $\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4\rangle$).

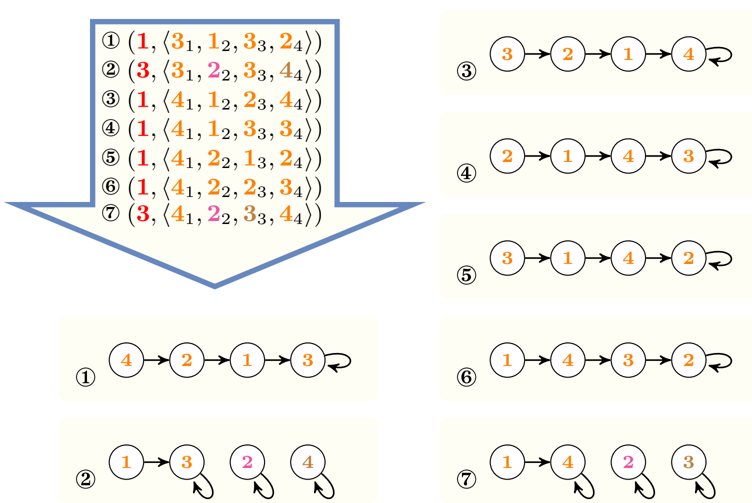


Figure 5.678: All solutions corresponding to the non ground example of the PATH constraint of the **All solutions** slot; in the left-hand side the `index` attributes are displayed as indices of the `succ` attribute, while in the right-hand side they are directly displayed within each node.

Typical

$\mathtt{NPATH} < |\mathtt{NODES}|$
$|\mathtt{NODES}| > 1$

Symmetry Items of NODES are permutable.

Arg. properties Functional dependency: NPATH determined by NODES.

Reformulation The PATH constraint can be expressed in term of (1) a set of $|\texttt{NODES}|^2$ reified constraints for avoiding circuit between more than one node and of (2) $|\texttt{NODES}|$ reified constraints and of one sum constraint for counting the paths and of (3) a set of $|\texttt{NODES}|^2$ reified constraints and of $|\texttt{NODES}|$ inequalities constraints for enforcing the fact that each vertex has at most two children.

1. For each vertex $\texttt{NODES}[i]$ $(i \in [1, |\texttt{NODES}|])$ of the NODES collection we create a variable R_i that takes its value within interval $[1, |\texttt{NODES}|]$. This variable represents the *rank* of vertex $\texttt{NODES}[i]$ within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices $\texttt{NODES}[i], \texttt{NODES}[j]$ $(i, j \in [1, |\texttt{NODES}|])$ of the NODES collection we create a reified constraint of the form $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Rightarrow R_i < R_j$. The purpose of this constraint is to express the fact that, if there is an arc from vertex $\texttt{NODES}[i]$ to another vertex $\texttt{NODES}[j]$, then R_i should be strictly less than R_j.

2. For each vertex $\texttt{NODES}[i]$ $(i \in [1, |\texttt{NODES}|])$ of the NODES collection we create a 0-1 variable B_i and state the following reified constraint $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[i].\texttt{index} \Leftrightarrow B_i$ in order to force variable B_i to be set to value 1 if and only if there is a loop on vertex $\texttt{NODES}[i]$. Finally we create a constraint $\texttt{NPATH} = B_1 + B_2 + \cdots + B_{|\texttt{NODES}|}$ for stating the fact that the number of paths is equal to the number of loops of the graph.

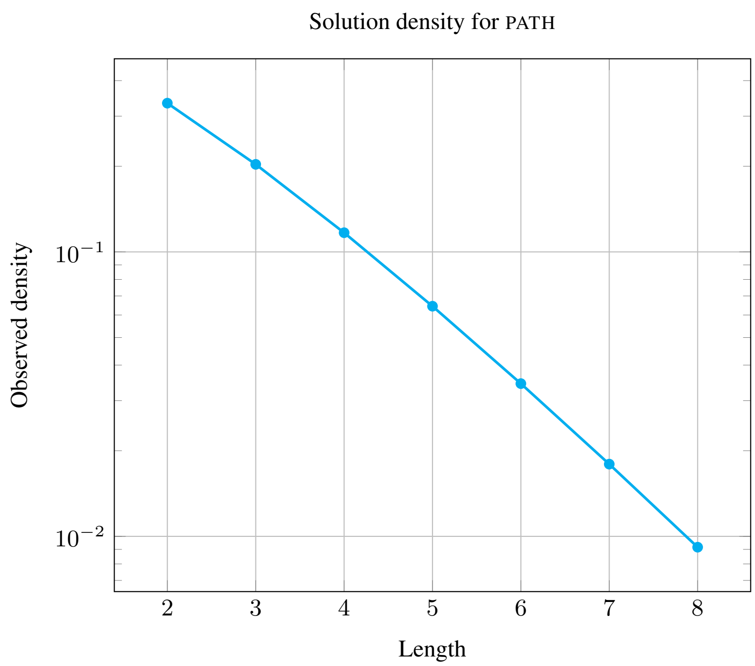
3. For each pair of vertices $\texttt{NODES}[i], \texttt{NODES}[j]$ $(i, j \in [1, |\texttt{NODES}|])$ of the NODES collection we create a 0-1 variable B_{ij} and state the following reified constraint $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Leftrightarrow B_{ij}$. Variable B_{ij} is set to value 1 if and only if there is an arc from $\texttt{NODES}[i]$ to $\texttt{NODES}[j]$. Then for each vertex $\texttt{NODES}[j]$ $(j \in [1, |\texttt{NODES}|])$ we create a constraint of the form $B_{1j} + B_{2j} + \cdots + B_{|\texttt{NODES}|j} \leq 1$.

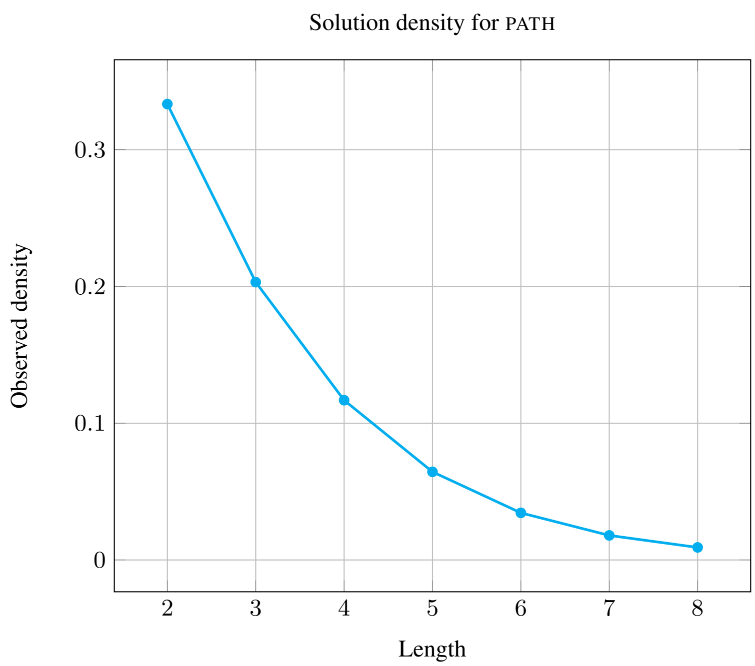
A second reformulation of the PATH constraint in term of two TREE constraints and one INVERSE_EXCEPT_LOOP constraint is described in the **Usage** slot of the INVERSE_EXCEPT_LOOP constraint.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	13	73	501	4051	37633	394353

Number of solutions for PATH: domains $0..n$

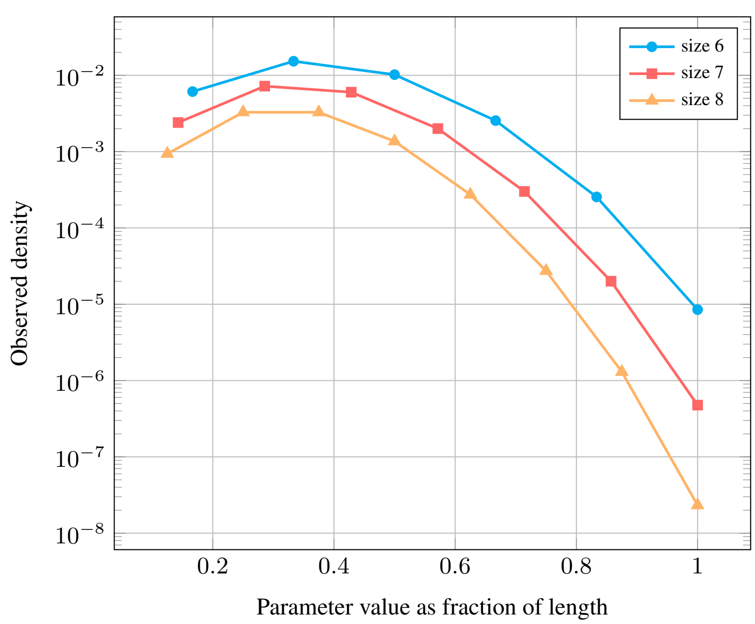

Solution density for PATH
Observed density
10^{-1}
10^{-2}
2
3
4
5
6
7
8
Length

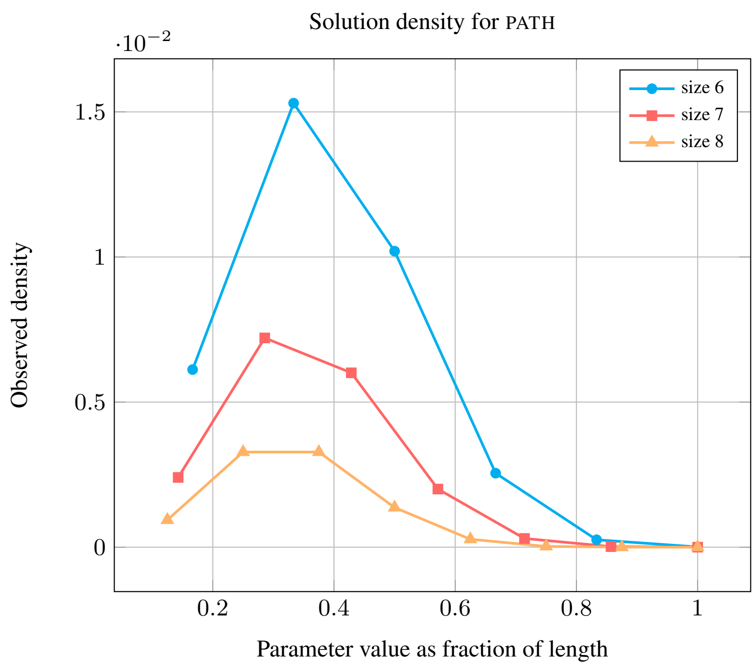

Solution density for PATH
Observed density
0.3
0.2
0.1
0
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		3	13	73	501	4051	37633	394353
Parameter value	1	2	6	24	120	720	5040	40320
	2	1	6	36	240	1800	15120	141120
	3	-	1	12	120	1200	12600	141120
	4	-	-	1	20	300	4200	58800
	5	-	-	-	1	30	630	11760
	6	-	-	-	-	1	42	1176
	7	-	-	-	-	-	1	56
	8	-	-	-	-	-	-	1

Solution count for PATH: domains $0..n$

Solution density for PATH





See also **common keyword:** CIRCUIT *(graph partitioning constraint, one_succ)*, DOM_REACHABILITY *(path)*, PATH_FROM_TO *(path, select an induced subgraph so that there is a path from a given vertex to an other given vertex)*, PROPER_CIRCUIT *(graph partitioning constraint, one_succ)*.

generalisation: BINARY_TREE *(at most one child replaced by at most two children)*, TEMPORAL_PATH *(vertices are located in time, and to each arc corresponds a precedence constraint)*, TREE *(at most one child replaced by no limit on the number of children)*.

implies: BINARY_TREE.

related: BALANCE_PATH *(counting number of paths versus controlling how balanced the paths are)*.

Keywords **combinatorial object:** path.

constraint type: graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck.

final graph structure: connected component, tree, one_succ.

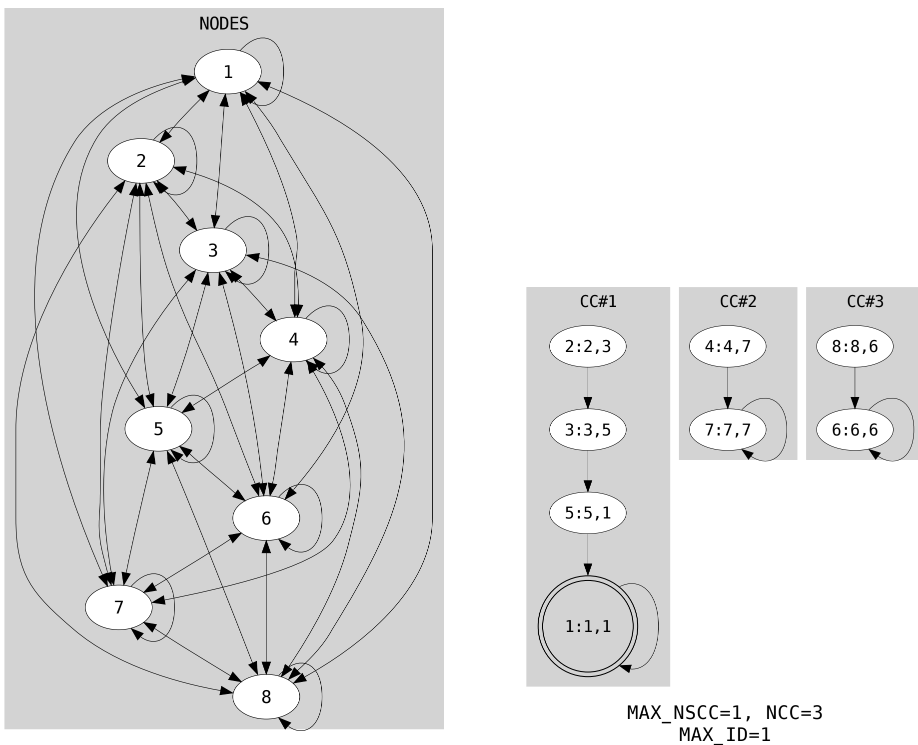
modelling: functional dependency.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NCC** = NPATH • **MAX_ID** ≤ 1
Graph class	ONE_SUCC

Graph model We use the same graph constraint as for the BINARY_TREE constraint, except that we replace the graph property **MAX_ID** ≤ 2, which constraints the maximum in-degree of the final graph to not exceed 2 by **MAX_ID** ≤ 1. **MAX_ID** does not consider loops: This is why we do not have any problem with the final node of each path.

Parts (A) and (B) of Figure 5.679 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NCC** graph property, we display the three connected components of the final graph. Each of them corresponds to a path. Since we use the **MAX_ID** graph property, we also show with a double circle a vertex that has a maximum number of predecessors.

The PATH constraint holds since all strongly connected components of the final graph have no more than one vertex, since NPATH = **NCC** $= 3$ and since **MAX_ID** $= 1$.



(A) **(B)**

Figure 5.679: Initial and final graph of the PATH constraint

5.322 PATH_FROM_TO

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [6]

Constraint PATH_FROM_TO(FROM, TO, NODES)

Usual name PATH

Arguments

FROM : int
TO : int
NODES : collection(index−int, succ−svar)

Restrictions

FROM ≥ 1
FROM $\leq$ |NODES|
TO ≥ 1
TO $\leq$ |NODES|
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

Purpose Select some arcs of a digraph G so that there is still a path between two given vertices of G.

Example

$$\left(4, 3, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\emptyset, \\ \texttt{index}-2 & \texttt{succ}-\emptyset, \\ \texttt{index}-3 & \texttt{succ}-\{5\}, \\ \texttt{index}-4 & \texttt{succ}-\{5\}, \\ \texttt{index}-5 & \texttt{succ}-\{2,3\} \end{array} \right\rangle \right)$$

The PATH_FROM_TO constraint holds since within the digraph G corresponding to the item of the NODES collection there is a path from vertex FROM $= 4$ to vertex TO $= 3$: this path starts from vertex 4, enters vertex 5, and ends up in vertex 3.

Typical

FROM $\neq$ TO
|NODES| > 2

Symmetry Items of NODES are permutable.

Remark Within [140] an undirected version of the PATH_FROM_TO constraint was proposed under the name UNDIRECTED_PATH.

See also

common keyword: DOM_REACHABILITY *(path)*, LINK_SET_TO_BOOLEANS *(constraint involving set variables)*, PATH, TEMPORAL_PATH *(path)*.

used in graph description: IN_SET.

Keywords

combinatorial object: path.

constraint arguments: constraint involving set variables.

constraint type: graph constraint.

filtering: linear programming.

Arc input(s)	NODES
Arc generator	*CLIQUE*↦collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.succ)
Graph property(ies)	**PATH_FROM_TO**(index, FROM, TO) = 1

Graph model Within the context of the **Example** slot, part (A) of Figure 5.680 shows the initial graph from which we choose to start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.680 gives the final graph associated with the **Example** slot. Since we use the **PATH_FROM_TO** graph property we show on the final graph the following information:

- The vertices that respectively correspond to the start and the end of the required path are stressed in bold.
- The arcs on the required path are also stressed in bold.

The PATH_FROM_TO constraint holds since there is a path from vertex 4 to vertex 3 (4 and 3 refer to the index attribute of a vertex).

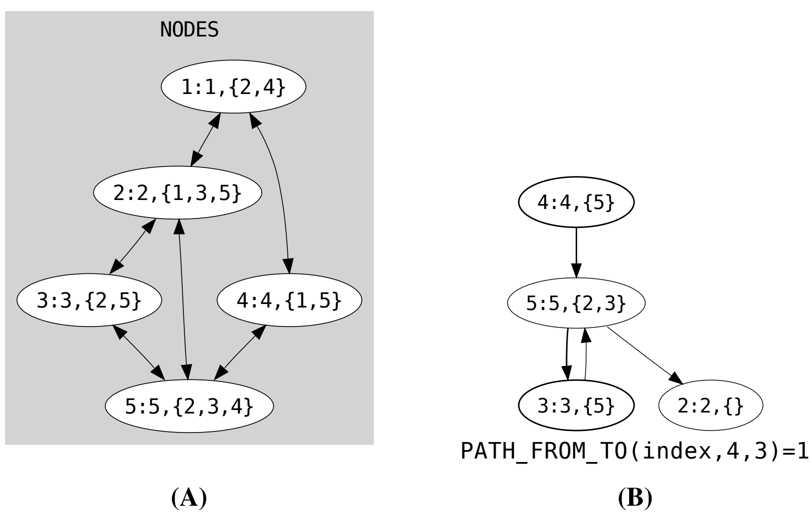


Figure 5.680: Initial and final graph of the PATH_FROM_TO set constraint

Signature Since the maximum value returned by the graph property **PATH_FROM_TO** is equal to 1 we can rewrite **PATH_FROM_TO**(index, FROM, TO) = 1 to **PATH_FROM_TO**(index, FROM, TO) ≥ 1. Therefore we simplify **PATH_FROM_TO** to **PATH_FROM_TO**.

5.323 PATTERN

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin [92]

Constraint $\text{PATTERN}(\mathtt{VARIABLES}, \mathtt{PATTERNS})$

Type $\mathtt{PATTERN} : \mathtt{collection}(\mathtt{var}-\mathtt{int})$

Arguments

$\mathtt{VARIABLES} : \mathtt{collection}(\mathtt{var}-\mathtt{dvar})$
$\mathtt{PATTERNS} : \mathtt{collection}(\mathtt{pat}-\mathtt{PATTERN})$

Restrictions

$\mathtt{required}(\mathtt{PATTERN}, \mathtt{var})$
$\mathtt{PATTERN.var} \geq 0$
$\text{CHANGE}(0, \mathtt{PATTERN}, =)$
$|\mathtt{PATTERN}| > 1$
$\mathtt{required}(\mathtt{VARIABLES}, \mathtt{var})$
$\mathtt{required}(\mathtt{PATTERNS}, \mathtt{pat})$
$|\mathtt{PATTERNS}| > 0$
$\mathtt{same\_size}(\mathtt{PATTERNS}, \mathtt{pat})$

Purpose

We quote the definition from the original article [92, page 157] introducing the PATTERN constraint:

> "*We call a k-pattern ($k > 1$) any sequence of k elements such that no two successive elements have the same value. Consider a set $V = \{v_1, v_2, \ldots, v_m\}$ and a sequence* $\mathbf{s} = s_1\ s_2\ \ldots\ s_n$ *of elements of V. In this context, a* stretch *is a maximum subsequence of variables of* $\mathbf{s}$ *which all have the same value. Consider now the sequence $v_{i1}\ v_{i2}\ \ldots\ v_{il}$ of the types of the successive stretches that appear in* $\mathbf{s}$*. Let $\mathcal{P}$ be a set of k-patterns.* $\mathbf{s}$ *satisfies $\mathcal{P}$ if and only if every subsequence of k elements in $v_{i1}\ v_{i2}\ \ldots, v_{il}$ belongs to $\mathcal{P}$.*"

Example

$$\left(\begin{array}{l}\langle 1,1,2,2,2,1,3,3\rangle, \\ \langle \mathtt{pat}-\langle 1,2,1\rangle, \mathtt{pat}-\langle 1,2,3\rangle, \mathtt{pat}-\langle 2,1,3\rangle\rangle\end{array}\right)$$

The PATTERN constraint holds since, as depicted by Figure 5.681, all its sequences of three consecutive stretches correspond to one of the 3-patterns given in the PATTERNS collection.

Typical

$|\mathtt{VARIABLES}| > 2$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 1$

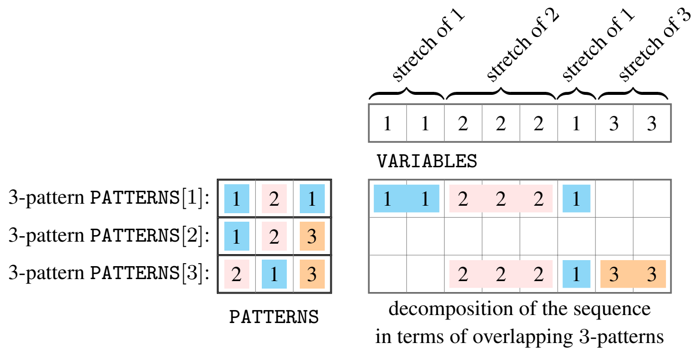


Figure 5.681: The sequence of the **Example** slot, its four stretches and the corresponding two 3-patterns 1 2 1 and 2 1 3

Symmetries

- Items of PATTERNS are permutable.
- Items of VARIABLES and PATTERNS.pat are simultaneously reversable.
- All occurrences of two distinct tuples of values in VARIABLES.var or PATTERNS.pat.var can be swapped; all occurrences of a tuple of values in VARIABLES.var or PATTERNS.pat.var can be renamed to any unused tuple of values.

Arg. properties

- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Usage

The PATTERN constraint was originally introduced within the context of staff scheduling. In this context, the value of the i^{th} variable of the VARIABLES collection corresponds to the type of shift performed by a person on the i^{th} day. A *stretch* is a maximum sequence of consecutive variables that are all assigned to the same value. The PATTERN constraint imposes that each sequence of k consecutive stretches belongs to a given list of patterns.

Remark

A generalisation of the PATTERN constraint to the REGULAR constraint enforcing the fact that a sequence of variables corresponds to a regular expression is presented in [317].

See also

common keyword: GROUP *(timetabling constraint)*, SLIDING_DISTRIBUTION *(sliding sequence constraint)*, STRETCH_CIRCUIT, STRETCH_PATH *(sliding sequence constraint,timetabling constraint)*, STRETCH_PATH_PARTITION *(sliding sequence constraint)*.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: timetabling constraint, sliding sequence constraint.

filtering: arc-consistency.

Automaton Taking advantage that all k-patterns have the same length k, it is straightforward to construct an automaton that only accepts solutions of the PATTERN constraint. Figure 5.682 depicts the automaton associated with the PATTERN constraint of the **Example** slot. The construction can be done in three steps:

- First, build a prefix tree of all the k-patterns. In the context of our example, this gives all arcs of Figure 5.682, except self loops and the arc from s_3 to s_7.
- Second, find out the transitions that exit a leave of the tree. For this purpose we remove the first symbol of the corresponding k-pattern, add at the end of the remaining k-pattern a symbol corresponding to a stretch value, and check whether the new pattern belongs or not to the set of k-patterns of the PATTERN constraint. When the new pattern belongs to the set of k-patterns we add a corresponding transition. For instance, in the context of our example, consider the leave s_3 that is associated with the 3-pattern $1, 2, 1$. We remove the first symbol 1 and get $2, 1$. We then try to successively add the stretch values 1, 2 and 3 to the end of $2, 1$ and check if the corresponding patterns $2, 1, 1$, $2, 1, 2$ and $2, 1, 3$ belong or not to our set of 3-patterns. Since only $2, 1, 3$ is a 3-pattern we add a new transition between the corresponding leaves of the prefix tree (i.e., a transition from s_3 to s_7).
- Third, in order to take into account that each value of a k-pattern corresponds in fact to a given stretch value (i.e., several consecutive values that are assigned the same value), we add a self loop to all non-source states with a transition label that corresponds to the transition label of their entering arcs.

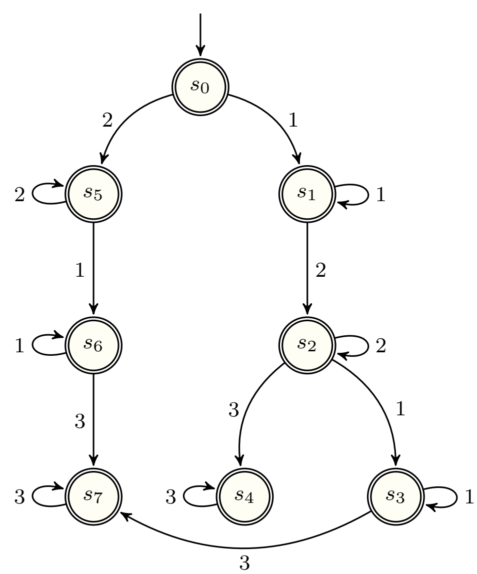


Figure 5.682: Automaton of the PATTERN constraint of the **Example** slot

5.324 PEAK

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin Derived from INFLEXION.

Constraint PEAK(N, VARIABLES)

Arguments

N : dvar
VARIABLES : collection(var−dvar)

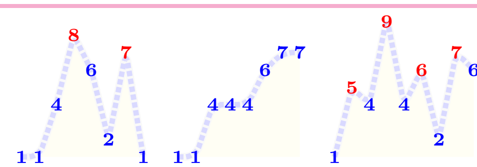
Restrictions

$\mathtt{N} \geq 0$
$2 * \mathtt{N} \leq \mathtt{max}(|\mathtt{VARIABLES}| - 1, 0)$
required(VARIABLES, var)

Purpose A variable V_k $(1 < k < m)$ of the sequence of variables $\mathtt{VARIABLES} = V_1, \ldots, V_m$ is a *peak* if and only if there exists an i (with $1 < i \leq k$) such that $V_{i-1} < V_i$ and $V_i = V_{i+1} = \cdots = V_k$ and $V_k > V_{k+1}$. N is the total number of peaks of the sequence of variables VARIABLES.

Example

$(2, \langle 1, 1, 4, 8, 6, 2, 7, 1 \rangle)$
$(0, \langle 1, 1, 4, 4, 4, 6, 7, 7 \rangle)$
$(4, \langle 1, 5, 4, 9, 4, 6, 2, 7, 6 \rangle)$



The first PEAK constraint holds since the sequence 1 1 4 8 6 2 7 1 contains two peaks that respectively correspond to the variables that are assigned to values 8 and 7.

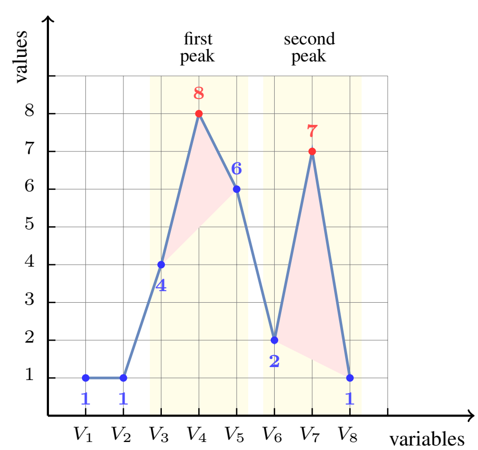


Figure 5.683: Illustration of the first example of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values 1, 1, 4, 8, 6, 2, 7, 1 and its corresponding two peaks ($\mathtt{N} = 2$)

All solutions

Figure 5.684 gives all solutions to the following non ground instance of the PEAK constraint: $\mathtt{N} \in [1,2]$, $\mathtt{V}_1 \in [1,2]$, $\mathtt{V}_2 = 2$, $\mathtt{V}_3 \in [1,2]$, $\mathtt{V}_4 \in [1,2]$, $\mathtt{V}_5 \in [2,3]$, PEAK$(\mathtt{N}, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4, \mathtt{V}_5 \rangle)$.

① $(\mathbf{1}, \langle 1, \mathbf{2}, 1, 1, 2\rangle)$
② $(\mathbf{1}, \langle 1, \mathbf{2}, 1, 1, 3\rangle)$
③ $(\mathbf{1}, \langle 1, \mathbf{2}, 1, 2, 2\rangle)$
④ $(\mathbf{1}, \langle 1, \mathbf{2}, 1, 2, 3\rangle)$
⑤ $(\mathbf{1}, \langle 1, \mathbf{2}, \mathbf{2}, 1, 2\rangle)$
⑥ $(\mathbf{1}, \langle 1, \mathbf{2}, \mathbf{2}, 1, 3\rangle)$

Figure 5.684: All solutions corresponding to the non ground example of the PEAK constraint of the **All solutions** slot where each peak is coloured in orange

Typical

$|\mathtt{VARIABLES}| > 2$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 1$

Typical model

$\mathtt{nval}(\mathtt{VARIABLES.var}) > 2$

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: N determined by VARIABLES.
- Contractible wrt. VARIABLES when $\mathtt{N} = 0$.

Usage

Useful for constraining the number of *peaks* of a sequence of domain variables.

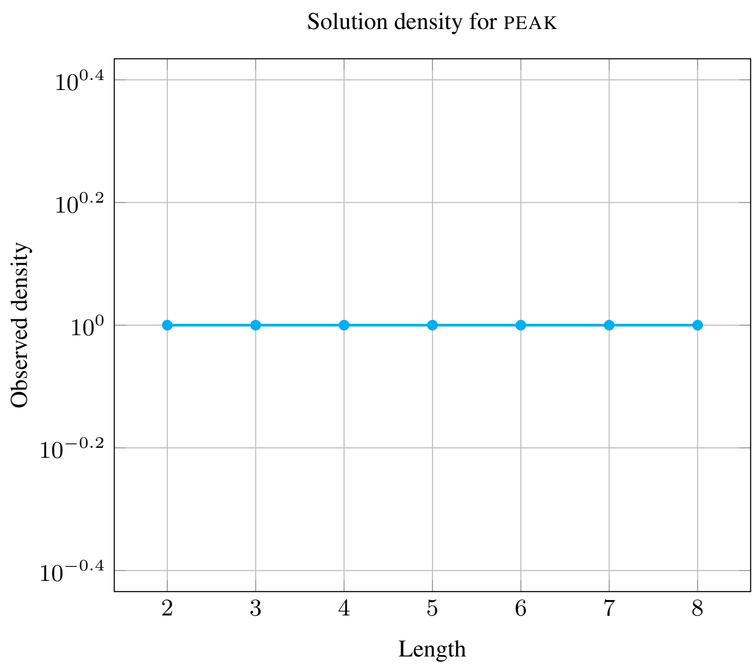
Remark

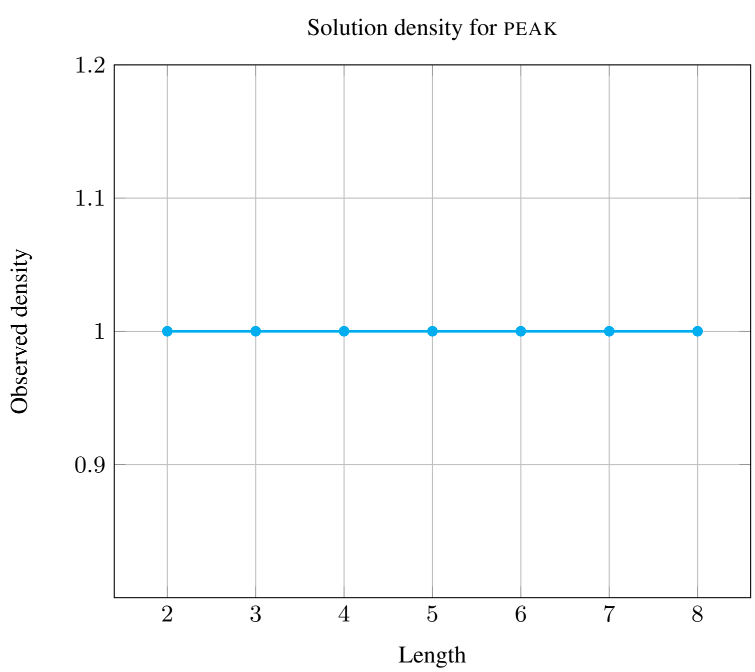
Since the arity of the arc constraint is not fixed, the PEAK constraint cannot be currently described with the graph-based representation. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

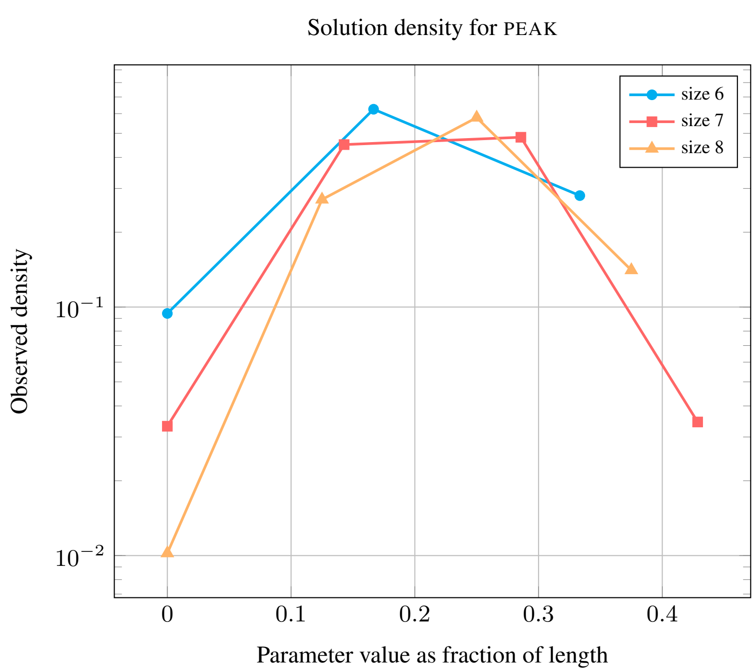
Number of solutions for PEAK: domains $0..n$

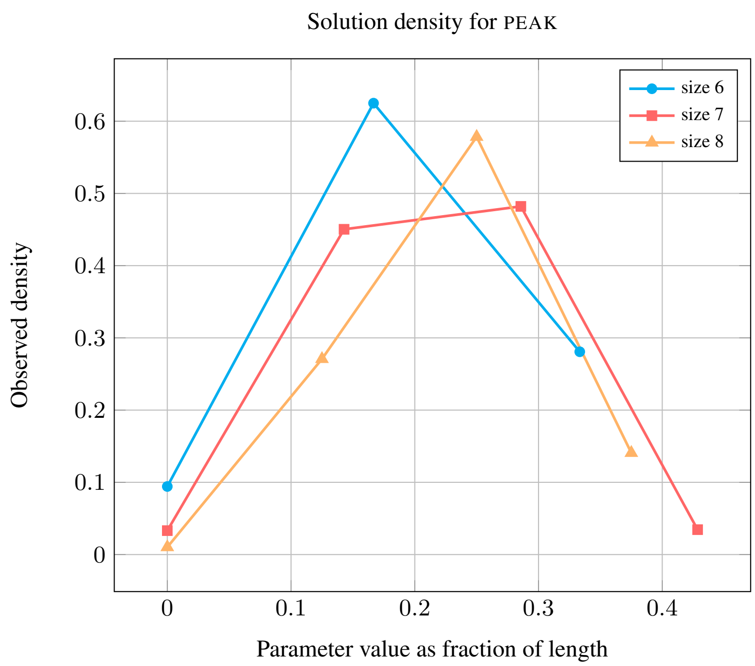

Solution density for PEAK
Observed density
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
2
3
4
5
6
7
8
Length


Solution density for PEAK
Observed density
1.2
1.1
1
0.9
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	9	50	295	1792	11088	69498	439791
	1	-	14	330	5313	73528	944430	11654622
	2	-	-	-	671	33033	1010922	24895038
	3	-	-	-	-	-	72302	6057270

Solution count for PEAK: domains $0..n$





See also

common keyword: HIGHEST_PEAK, INFLEXION, MIN_DIST_BETWEEN_INFLEXION, MIN_SURF_PEAK, MIN_WIDTH_PEAK *(sequence)*.

comparison swapped: VALLEY.

generalisation: BIG_PEAK *(a tolerance parameter is added for counting only big peaks)*.

related: ALL_EQUAL_PEAK, ALL_EQUAL_PEAK_MAX, DECREASING_PEAK, INCREASING_PEAK, NO_VALLEY.

specialisation: NO_PEAK *(the variable counting the number of peaks is set to* 0 *and removed)*.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

filtering: glue matrix.

modelling: functional dependency.

Cond. implications

- PEAK(N, VARIABLES)
  with $\mathtt{N} > 0$
  implies ATLEAST_NVALUE(NVAL, VARIABLES)
  when $\mathtt{NVAL} = 2$.
- PEAK(N, VARIABLES)
  implies INFLEXION(N, VARIABLES)
  when N = PEAK(VARIABLES.var)+VALLEY(VARIABLES.var).

Automaton Figure 5.685 depicts the automaton associated with the PEAK constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $(\mathtt{VAR}_i < \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\mathtt{VAR}_i = \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\mathtt{VAR}_i > \mathtt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: stationary/decreasing mode	$(\{> \mid =\}^*)$
u	: increasing mode	$(< \{< \mid =\}^*)$

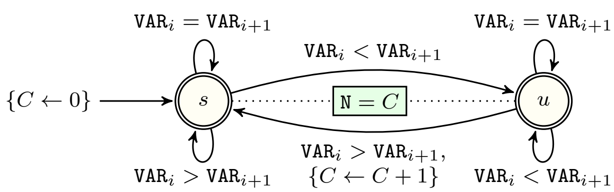


Figure 5.685: Automaton of the PEAK constraint

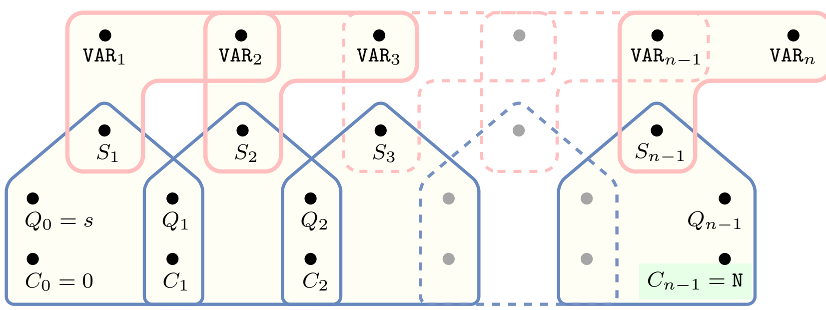


Figure 5.686: Hypergraph of the reformulation corresponding to the automaton of the PEAK constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

	s $(\{> \mid =\}^*)$	u $(< \{< \mid =\}^*)$
s $(\{> \mid =\}^*)$	$\overrightarrow{C} + \overleftarrow{C}$	$\overrightarrow{C} + \overleftarrow{C}$
u $(< \{< \mid =\}^*)$	$\overrightarrow{C} + \overleftarrow{C}$	$\overrightarrow{C} + 1 + \overleftarrow{C}$

Figure 5.687: Glue matrix of the PEAK constraint

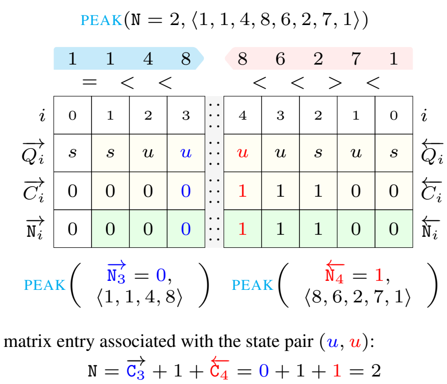


Figure 5.688: Illustrating the use of the state pair (u, u) of the glue matrix for linking N with the counters variables obtained after reading the prefix $1, 1, 4, 8$ and corresponding suffix $8, 6, 2, 7, 1$ of the sequence $1, 1, 4, 8, 6, 2, 7, 1$; note that the suffix $8, 6, 2, 7, 1$ (in pink) is proceed in reverse order; the left (resp. right) table shows the initialisation (for $i = 0$) and the evolution (for $i > 0$) of the state of the automaton and of its counter C upon reading the prefix $1, 1, 4, 8$ (resp. the suffix $1, 7, 2, 6, 8$).

5.325 PERIOD

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin N. Beldiceanu

Constraint PERIOD(PERIOD, VARIABLES, CTR)

Arguments

PERIOD : dvar
VARIABLES : collection(var−dvar)
CTR : atom

Restrictions

PERIOD ≥ 1
PERIOD $\leq |$VARIABLES$|$
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Let us note $V_0, V_1, \ldots, V_{m-1}$ the variables of the VARIABLES collection. PERIOD is the *period* of the sequence $V_0\ V_1 \ldots V_{m-1}$ according to constraint CTR. This means that PERIOD is the smallest natural number such that V_i CTR $V_{i+\texttt{PERIOD}}$ holds for all $i \in 0, 1, \ldots, m - \texttt{PERIOD} - 1$.

Example

$(3, \langle 1, 1, 4, 1, 1, 4, 1, 1 \rangle, =)$

The PERIOD constraint holds since, as depicted by Figure 5.689, its first argument PERIOD $= 3$ is equal (i.e., since CTR is set to $=$) to the period of the sequence 1 1 4 1 1 4 1 1.

1	**1**	**4**	1	1	4	1	1

Figure 5.689: A sequence of period 3

Typical

PERIOD > 1
PERIOD $< |$VARIABLES$|$
$|$VARIABLES$| > 2$
range(VARIABLES.var) > 1
CTR $\in [=]$

Symmetries

- Items of VARIABLES can be reversed.
- Items of VARIABLES can be shifted.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties

- Functional dependency: PERIOD determined by VARIABLES and CTR.
- Contractible wrt. VARIABLES when CTR $\in [=]$ and PERIOD $= 1$.
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Algorithm

When CTR corresponds to the equality constraint, a potentially incomplete filtering algorithm based on 13 deductions rules is described in [60]. The generalisation of these rules to the case where CTR is not the equality constraint is discussed.

See also

generalisation: PERIOD_VECTORS *(*variable *replaced by vector)*.

implies: PERIOD_EXCEPT_0.

soft variant: PERIOD_EXCEPT_0 *(value* 0 *can match any other value)*.

Keywords

combinatorial object: periodic, sequence.

constraint arguments: pure functional dependency.

constraint type: predefined constraint, timetabling constraint, scheduling constraint.

filtering: border.

modelling: functional dependency.

5.326 PERIOD_EXCEPT_0

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin

Derived from PERIOD.

Constraint

PERIOD_EXCEPT_0(PERIOD, VARIABLES, CTR)

Arguments

PERIOD	:	dvar
VARIABLES	:	collection(var−dvar)
CTR	:	atom

Restrictions

PERIOD ≥ 1
PERIOD $\leq$ |VARIABLES|
required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Let us note $V_0, V_1, \ldots, V_{m-1}$ the variables of the VARIABLES collection. PERIOD is the *period* of the sequence $V_0\ V_1 \ldots V_{m-1}$ according to constraint CTR. This means that PERIOD is the smallest natural number such that V_i CTR $V_{i+\texttt{PERIOD}} \vee V_i = 0 \vee V_{i+\texttt{PERIOD}} = 0$ holds for all $i \in 0, 1, \ldots, m - \texttt{PERIOD} - 1$.

Example

$(3, \langle 1, 1, 4, 1, 1, 0, 1, 1 \rangle, =)$

The PERIOD_EXCEPT_0 constraint holds since, as depicted by Figure 5.690, its first argument PERIOD $= 3$ is equal (i.e., since CTR is set to $=$) to the period of the sequence 1 1 4 1 1 0 1 1; value 0 is assumed to be equal to any other value.

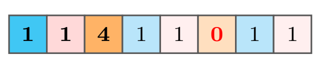


Figure 5.690: A sequence that has a period of 3 when we assume that value 0 can match to any other value

Typical

PERIOD > 1
PERIOD $<$ |VARIABLES|
|VARIABLES| > 2
range(VARIABLES.var) > 1
ATLEAST(1, VARIABLES, 0)
CTR $\in [=]$

Typical model

ATLEAST(2, VARIABLES, 0)

Symmetries

- Items of VARIABLES can be reversed.
- Items of VARIABLES can be shifted.
- All occurrences of two distinct values of VARIABLES.var that are both different from 0 can be swapped; all occurrences of a value of VARIABLES.var that is different from 0 can be renamed to any unused value that is also different from 0.

Arg. properties

- Functional dependency: PERIOD determined by VARIABLES and CTR.
- Contractible wrt. VARIABLES when CTR $\in [=]$ and PERIOD $= 1$.
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Usage

Useful for timetabling problems where a person should repeat some work pattern over an over except when he is unavailable for some reason. The value 0 represents the fact that he is unavailable, while the other values are used in the work pattern.

Algorithm

See [60].

See also

hard version: PERIOD.

implied by: PERIOD.

Keywords

characteristic of a constraint: joker value.

combinatorial object: periodic, sequence.

constraint arguments: pure functional dependency.

constraint type: predefined constraint, timetabling constraint, scheduling constraint.

modelling: functional dependency.

5.327 PERIOD_VECTORS

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin

Derived from PERIOD

Constraint

PERIOD_VECTORS(PERIOD, VECTORS, CTRS)

Types

VECTOR	:	collection(var−dvar)
CTR	:	atom

Arguments

PERIOD	:	dvar
VECTORS	:	collection(vec − VECTOR)
CTRS	:	collection(ctr − CTR)

Restrictions

$|\mathtt{VECTOR}| \geq 1$
required(VECTOR, var)
$\mathtt{CTR} \in [=, \neq, <, \geq, >, \leq]$
$\mathtt{PERIOD} \geq 1$
$\mathtt{PERIOD} \leq |\mathtt{VECTORS}|$
required(VECTORS, vec)
same_size(VECTORS, vec)
required(CTRS, ctr)
$|\mathtt{CTRS}| = |\mathtt{VECTOR}|$

Purpose

Let us note $\mathtt{VECTOR}_0, \mathtt{VECTOR}_1, \ldots, \mathtt{VECTOR}_{n-1}$ the vectors of the VECTORS collection, and d the number of components of each vector (all vectors have the same size). PERIOD is the *period* of the sequence of vectors $\mathtt{VECTOR}_0, \mathtt{VECTOR}_1, \ldots, \mathtt{VECTOR}_{n-1}$ according to constraints CTRS. This means that PERIOD is the smallest natural number such that $\forall i \in [0, n - \mathtt{PERIOD} - 1], \forall j \in [0, d - 1]$: $\mathtt{VECTOR}_i.\mathtt{vec}[j]\ \mathtt{CTRS}[j]\ \mathtt{VECTOR}_{i+\mathtt{PERIOD}}.\mathtt{vec}[j]$.

Example

$$\left(\begin{array}{l} 3, \left\langle\begin{array}{l} \mathtt{vec} - \langle 1,0\rangle, \\ \mathtt{vec} - \langle 1,5\rangle, \\ \mathtt{vec} - \langle 4,4\rangle, \\ \mathtt{vec} - \langle 1,0\rangle, \\ \mathtt{vec} - \langle 1,5\rangle, \\ \mathtt{vec} - \langle 4,4\rangle, \\ \mathtt{vec} - \langle 1,0\rangle, \\ \mathtt{vec} - \langle 1,5\rangle \end{array}\right\rangle, \\ \langle =, = \rangle \end{array}\right)$$

The PERIOD_VECTORS constraint holds since its first argument $\mathtt{PERIOD} = 3$ is equal (i.e., since CTRS is set to $\langle =, = \rangle$) to the period of the sequence $\mathtt{vec} - \langle 1,0\rangle$, $\mathtt{vec} - \langle 1,5\rangle$, $\mathtt{vec} - \langle 4,4\rangle$, $\mathtt{vec} - \langle 1,0\rangle$, $\mathtt{vec} - \langle 1,5\rangle$, $\mathtt{vec} - \langle 4,4\rangle$, $\mathtt{vec} - \langle 1,0\rangle$, $\mathtt{vec} - \langle 1,5\rangle$.

Typical

$\mathtt{CTR} \in [=]$
$|\mathtt{VECTOR}| > 1$
$\mathtt{PERIOD} > 1$
$\mathtt{PERIOD} < |\mathtt{VECTORS}|$
$|\mathtt{VECTORS}| > 2$

Symmetry

Items of VECTORS can be reversed.

Arg. properties

- Functional dependency: PERIOD determined by VECTORS and CTRS.
- Prefix-contractible wrt. VECTORS.
- Suffix-contractible wrt. VECTORS.

See also

specialisation: PERIOD (*vector replaced by* variable).

Keywords

characteristic of a constraint: vector.

combinatorial object: periodic, sequence.

constraint arguments: pure functional dependency.

constraint type: predefined constraint.

modelling: functional dependency.

5.328 PERMUTATION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from ALLDIFFERENT_CONSECUTIVE_VALUES.

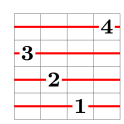
Constraint PERMUTATION(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restrictions
required(VARIABLES, var)
$\text{minval}(\texttt{VARIABLES.var}) = 1$
$\text{maxval}(\texttt{VARIABLES.var}) = |\texttt{VARIABLES}|$

Purpose Enforce all variables of the collection VARIABLES to take distinct values between 1 and the total number of variables.

Example $(\langle 3, 2, 1, 4\rangle)$



The PERMUTATION constraint holds since all the values 3, 2, 1 and 4 are distinct, and since they all belong to interval $[1, 4]$ where 4 is the total number of variables.

Typical $|\texttt{VARIABLES}| > 2$

Symmetries

- Items of VARIABLES are permutable.
- Two distinct values of VARIABLES.var can be swapped.

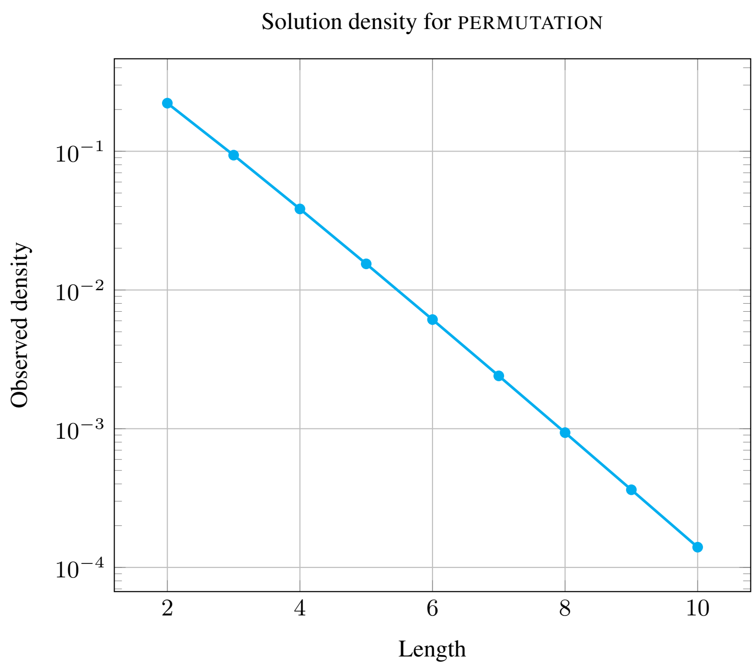
Usage See **Usage** slot of ALLDIFFERENT.

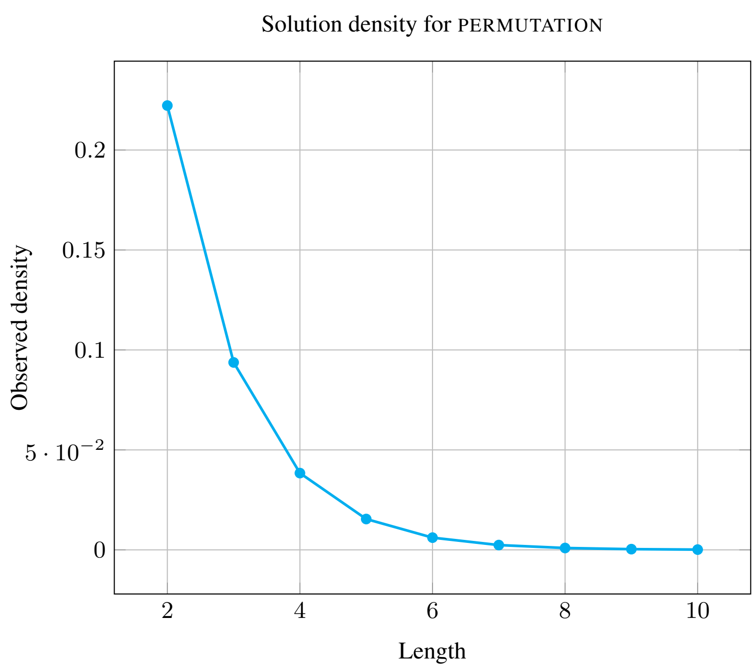
Algorithm See **Algorithm** slot of ALLDIFFERENT.

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	2	6	24	120	720	5040	40320	362880	3628800

Number of solutions for PERMUTATION: domains $0..n$





See also **implied by:** PROPER_CIRCUIT.

implies: ALLDIFFERENT_CONSECUTIVE_VALUES.

Keywords **characteristic of a constraint:** all different, disequality, sort based reformulation.

combinatorial object: permutation.

constraint type: value constraint.

final graph structure: one_succ.

Cond. implications

- PERMUTATION(VARIABLES)
  implies BALANCE(BALANCE, VARIABLES)
  when BALANCE $= 0$.

- PERMUTATION(VARIABLES)
  implies CHANGE(NCHANGE, VARIABLES, CTR)
  when NCHANGE $= |$VARIABLES$| - 1$
  and CTR $\in [\neq]$.

- PERMUTATION(VARIABLES)
  implies CIRCULAR_CHANGE(NCHANGE, VARIABLES, CTR)
  when NCHANGE $= |$VARIABLES$|$
  and CTR $\in [\neq]$.

- PERMUTATION(VARIABLES)
  implies LENGTH_LAST_SEQUENCE(LEN, VARIABLES)
  when LEN $= 1$.

- PERMUTATION(VARIABLES)
  implies LENGTH_FIRST_SEQUENCE(LEN, VARIABLES)
  when LEN $= 1$.

- PERMUTATION(VARIABLES)
  implies LONGEST_CHANGE(SIZE, VARIABLES, CTR)
  when SIZE $= |$VARIABLES$|$
  and CTR $\in [\neq]$.

- PERMUTATION(VARIABLES)
  implies MAX_N(MAX, RANK, VARIABLES)
  when MAX $= |$VARIABLES$| -$ RANK.

- PERMUTATION(VARIABLES)
  implies MIN_N(MIN, RANK, VARIABLES)
  when MIN $=$ RANK $+ 1$.

- PERMUTATION(VARIABLES)
  implies MIN_NVALUE(MIN, VARIABLES)
  when MIN $= 1$.

- PERMUTATION(VARIABLES)
  implies MIN_SIZE_FULL_ZERO_STRETCH(MINSIZE, VARIABLES)
  when MINSIZE $= |$VARIABLES$|$.

- PERMUTATION(VARIABLES)
  implies NINTERVAL(NVAL, VARIABLES, SIZE_INTERVAL)
  when NVAL $= (|$VARIABLES$| +$ SIZE_INTERVAL$)/$SIZE_INTERVAL.

- PERMUTATION(VARIABLES)
  implies RANGE_CTR(VARIABLES, CTR, R)
  when CTR $\in$ [$\leq$]
  and R $=$ |VARIABLES|.

- PERMUTATION(VARIABLES)
  implies SOFT_ALLDIFFERENT_CTR(C, VARIABLES).

- PERMUTATION(VARIABLES)
  implies SOFT_ALL_EQUAL_MAX_VAR(N, VARIABLES)
  when N $\leq$ |VARIABLES| $-$ 1.

- PERMUTATION(VARIABLES)
  implies SOFT_ALL_EQUAL_MIN_VAR(N, VARIABLES)
  when N $\geq$ |VARIABLES| $-$ 1.

- PERMUTATION(VARIABLES)
  implies SUM_CTR(VARIABLES, CTR, VAR)
  when CTR $\in$ [$=$]
  and VAR $=$ |VARIABLES| $*$ (|VARIABLES| $+$ 1)/2.

- PERMUTATION(VARIABLES)
  with |VARIABLES| $>$ 2
  and first(VARIABLES.var) $>$minval(VARIABLES.var)
  and last(VARIABLES.var) $>$minval(VARIABLES.var)
  implies DEEPEST_VALLEY(DEPTH, VARIABLES)
  when DEPTH $=$minval(VARIABLES.var).

- PERMUTATION(VARIABLES)
  with |VARIABLES| $>$ 2
  and first(VARIABLES.var) $=$ 1
  implies DEEPEST_VALLEY(DEPTH, VARIABLES)
  when DEPTH $=$ 2.

- PERMUTATION(VARIABLES)
  with |VARIABLES| $>$ 2
  and last(VARIABLES.var) $=$ 1
  implies DEEPEST_VALLEY(DEPTH, VARIABLES)
  when DEPTH $=$ 2.

- PERMUTATION(VARIABLES)
  with |VARIABLES| $>$ 2
  and first(VARIABLES.var) $<$maxval(VARIABLES.var)
  and last(VARIABLES.var) $<$maxval(VARIABLES.var)
  implies HIGHEST_PEAK(HEIGHT, VARIABLES)
  when HEIGHT $=$maxval(VARIABLES.var).

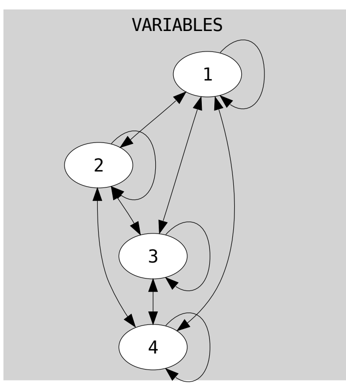
- PERMUTATION(VARIABLES)
  with |VARIABLES| $>$ 2
  and first(VARIABLES.var) $=$ |VARIABLES|
  implies HIGHEST_PEAK(HEIGHT, VARIABLES)
  when HEIGHT $=$ |VARIABLES| $-$ 1.

- PERMUTATION($\texttt{VARIABLES}$)
  with $|\texttt{VARIABLES}| > 2$
  and $\texttt{last}(\texttt{VARIABLES.var}) = |\texttt{VARIABLES}|$
  implies HIGHEST_PEAK($\texttt{HEIGHT}, \texttt{VARIABLES}$)
  when $\texttt{HEIGHT} = |\texttt{VARIABLES}| - 1$.

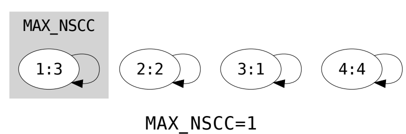
Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**MAX_NSCC** ≤ 1
Graph class	ONE_SUCC

Graph model We generate a *clique* with an *equality* constraint between each pair of vertices (including a vertex and itself) and state that the size of the largest strongly connected component should not exceed one. Finally the restrictions express the fact that all values are between 1 and the total number of variables.

Parts (A) and (B) of Figure 5.691 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected component of the final graph. The PERMUTATION holds since all the strongly connected components have at most one vertex: a value is used at most once.



(A)



(B)

Figure 5.691: Initial and final graph of the PERMUTATION constraint

5.329 PLACE_IN_PYRAMID

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint PLACE_IN_PYRAMID(ORTHOTOPES, VERTICAL_DIM)

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments

ORTHOTOPES : collection(orth − ORTHOTOPE)
VERTICAL_DIM : int

Restrictions

|ORTHOTOPE| >0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori $\leq$ ORTHOTOPE.end
required(ORTHOTOPES, orth)
same_size(ORTHOTOPES, orth)
VERTICAL_DIM ≥ 1
DIFFN(ORTHOTOPES)

Purpose

For each pair of orthotopes (O_1, O_2) of the collection ORTHOTOPES, O_1 and O_2 do not overlap (two orthotopes do not overlap if there exists at least one dimension where their projections do not overlap). In addition, each orthotope of the collection ORTHOTOPES should be supported by one other orthotope or by the ground. The vertical dimension is given by the parameter VERTICAL_DIM.

Example

$$\left(\left\langle\begin{array}{l}\texttt{orth}-\langle\texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3\rangle,\\ \texttt{orth}-\langle\texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3, \texttt{ori}-3\ \texttt{siz}-3\ \texttt{end}-6\rangle,\\ \texttt{orth}-\left\langle\begin{array}{lll}\texttt{ori}-5 & \texttt{siz}-6 & \texttt{end}-11,\\ \texttt{ori}-1 & \texttt{siz}-2 & \texttt{end}-3\end{array}\right\rangle,\\ \texttt{orth}-\langle\texttt{ori}-5\ \texttt{siz}-2\ \texttt{end}-7, \texttt{ori}-3\ \texttt{siz}-2\ \texttt{end}-5\rangle,\\ \texttt{orth}-\left\langle\begin{array}{lll}\texttt{ori}-8 & \texttt{siz}-3 & \texttt{end}-11,\\ \texttt{ori}-3 & \texttt{siz}-2 & \texttt{end}-5\end{array}\right\rangle,\\ \texttt{orth}-\left\langle\begin{array}{lll}\texttt{ori}-8 & \texttt{siz}-2 & \texttt{end}-10,\\ \texttt{ori}-5 & \texttt{siz}-2 & \texttt{end}-7\end{array}\right\rangle\end{array}\right\rangle, 2\right)$$

Figure 5.692 depicts the placement associated with the example, where the i^{th} item of the ORTHOTOPES collection is represented by the rectangle Ri. The PLACE_IN_PYRAMID constraint holds since the rectangles do not overlap and since rectangles R1, R2, R3, R4, R5, and R6 are respectively supported by the ground, R1, the ground, R3, R3, and R5.

Typical

|ORTHOTOPE| >1
ORTHOTOPE.siz >0
|ORTHOTOPES| >1

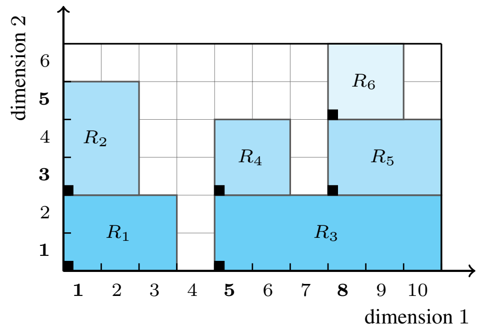


Figure 5.692: Solution corresponding to the **Example** slot

Symmetry Items of `ORTHOTOPES` are permutable.

Usage The DIFFN constraint is not enough if one wants to produce a placement where no orthotope floats in the air. This constraint is usually handled with a heuristic during the enumeration phase.

See also **used in graph description:** ORTH_ON_THE_GROUND, ORTH_ON_TOP_OF_ORTH.

Keywords **constraint type:** logic.

geometry: geometrical constraint, non-overlapping, orthotope.

Arc input(s) ORTHOTOPES

Arc generator $CLIQUE \mapsto$ collection(orthotopes1, orthotopes2)

Arc arity 2

Arc constraint(s)

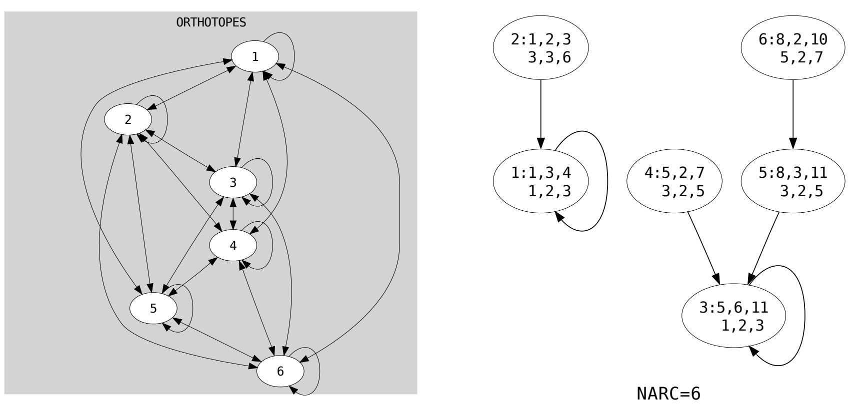
$$\bigvee\left(\begin{array}{l}\bigwedge\left(\begin{array}{l}\texttt{orthotopes1.key} = \texttt{orthotopes2.key},\\ \texttt{ORTH\_ON\_THE\_GROUND}(\texttt{orthotopes1.orth}, \texttt{VERTICAL\_DIM})\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{orthotopes1.key} \neq \texttt{orthotopes2.key},\\ \texttt{ORTH\_ON\_TOP\_OF\_ORTH}\left(\begin{array}{l}\texttt{orthotopes1.orth},\\ \texttt{orthotopes2.orth},\\ \texttt{VERTICAL\_DIM}\end{array}\right)\end{array}\right)\end{array}\right)$$

Graph property(ies) $\mathbf{NARC} = |\texttt{ORTHOTOPES}|$

Graph model The arc constraint of the graph constraint forces one of the following conditions:

- If the arc connects the same orthotope O then the ground directly supports O,
- Otherwise, if we have an arc from an orthotope O_1 to a distinct orthotope O_2, the condition is: O_1 is on top of O_2 (i.e., in all dimensions, except dimension VERTICAL_DIM, the projection of O_1 is included in the projection of O_2, while in dimension VERTICAL_DIM the projection of O_1 is located after the projection of O_2).

Parts (A) and (B) of Figure 5.693 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.693: Initial and final graph of the PLACE_IN_PYRAMID constraint

5.330 POLYOMINO

DESCRIPTION LINKS GRAPH

Origin

Inspired by [206].

Constraint

POLYOMINO($\texttt{CELLS}$)

Argument

$$\texttt{CELLS} : \texttt{collection}\begin{pmatrix}\texttt{index}-\texttt{int},\\ \texttt{right}-\texttt{dvar},\\ \texttt{left}-\texttt{dvar},\\ \texttt{up}-\texttt{dvar},\\ \texttt{down}-\texttt{dvar}\end{pmatrix}$$

Restrictions

$\texttt{CELLS.index} \geq 1$
$\texttt{CELLS.index} \leq |\texttt{CELLS}|$
$|\texttt{CELLS}| \geq 1$
$\texttt{required}(\texttt{CELLS}, [\texttt{index}, \texttt{right}, \texttt{left}, \texttt{up}, \texttt{down}])$
$\texttt{distinct}(\texttt{CELLS}, \texttt{index})$
$\texttt{CELLS.right} \geq 0$
$\texttt{CELLS.right} \leq |\texttt{CELLS}|$
$\texttt{CELLS.left} \geq 0$
$\texttt{CELLS.left} \leq |\texttt{CELLS}|$
$\texttt{CELLS.up} \geq 0$
$\texttt{CELLS.up} \leq |\texttt{CELLS}|$
$\texttt{CELLS.down} \geq 0$
$\texttt{CELLS.down} \leq |\texttt{CELLS}|$

Purpose

Enforce all cells of the collection CELLS to be connected and to form a single block. Each cell is defined by the following attributes:

1. The index attribute of the cell, which is an integer between 1 and the total number of cells, is unique for each cell.
2. The right attribute that is the index of the cell located immediately to the right of that cell (or 0 if no such cell exists).
3. The left attribute that is the index of the cell located immediately to the left of that cell (or 0 if no such cell exists).
4. The up attribute that is the index of the cell located immediately on top of that cell (or 0 if no such cell exists).
5. The down attribute that is the index of the cell located immediately above that cell (or 0 if no such cell exists).

This corresponds to a polyomino [206].

Example

$$\left(\left\langle\begin{array}{lllll}\texttt{index}-1 & \texttt{right}-0 & \texttt{left}-0 & \texttt{up}-2 & \texttt{down}-0,\\ \texttt{index}-2 & \texttt{right}-3 & \texttt{left}-0 & \texttt{up}-0 & \texttt{down}-1,\\ \texttt{index}-3 & \texttt{right}-0 & \texttt{left}-2 & \texttt{up}-4 & \texttt{down}-0,\\ \texttt{index}-4 & \texttt{right}-5 & \texttt{left}-0 & \texttt{up}-0 & \texttt{down}-3,\\ \texttt{index}-5 & \texttt{right}-0 & \texttt{left}-4 & \texttt{up}-0 & \texttt{down}-0\end{array}\right\rangle\right)$$

The POLYOMINO constraint holds since all the cells corresponding to the items of the CELLS collection form one single group of connected cells: the i^{th} ($i \in [1, 4]$) cell is connected to the $(i+1)^{th}$ cell. Figure 5.694 shows the corresponding polyomino.

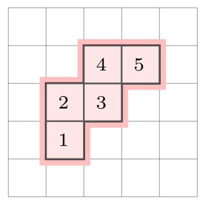


Figure 5.694: Polyomino corresponding to the **Example** slot where each cell contains the index of the corresponding item within the CELLS collection

Symmetries

- Items of CELLS are permutable.
- Attributes of CELLS are permutable w.r.t. permutation (index) (right, left) (up) (down) (*permutation applied to all items*).
- Attributes of CELLS are permutable w.r.t. permutation (index) (right) (left) (up, down) (*permutation applied to all items*).
- Attributes of CELLS are permutable w.r.t. permutation (index) (up, left, down, right) (*permutation applied to all items*).

Usage

Enumeration of polyominoes.

Keywords

combinatorial object: pentomino.

final graph structure: strongly connected component.

geometry: geometrical constraint.

puzzles: pentomino.

Arc input(s) CELLS

Arc generator $CLIQUE(\neq) \mapsto$ collection(cells1, cells2)

Arc arity 2

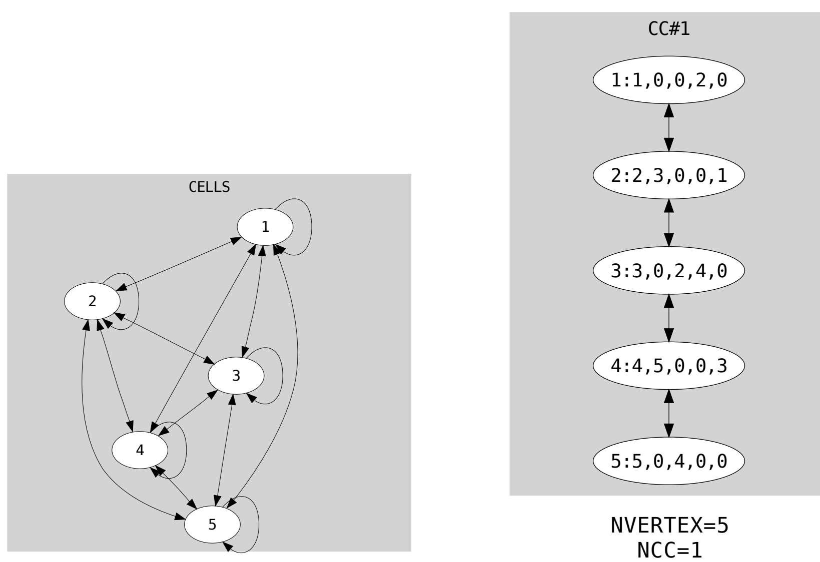
Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\bigwedge\left(\begin{array}{l}\texttt{cells1.right} = \texttt{cells2.index},\\ \texttt{cells2.left} = \texttt{cells1.index}\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{cells1.left} = \texttt{cells2.index},\\ \texttt{cells2.right} = \texttt{cells1.index}\end{array}\right),\\ \texttt{cells1.up} = \texttt{cells2.index} \wedge \texttt{cells2.down} = \texttt{cells1.index},\\ \texttt{cells1.down} = \texttt{cells2.index} \wedge \texttt{cells2.up} = \texttt{cells1.index}\end{array}\right)$$

Graph property(ies)
- **NVERTEX**$= |\texttt{CELLS}|$
- **NCC**$= 1$

Graph model The graph constraint models the fact that all the cells are connected. We use the $CLIQUE(\neq)$ arc generator in order to only consider connections between two distinct cells. The first graph property $\mathbf{NVERTEX} = |\texttt{CELLS}|$ avoid the case isolated cells, while the second graph property $\mathbf{NCC} = 1$ enforces to have a single group of connected cells.

Parts (A) and (B) of Figure 5.695 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NVERTEX** graph property the vertices of the final graph are stressed in bold. Since we also use the **NCC** graph property we show the unique connected component of the final graph. An arc between two vertices indicates that two cells are directly connected.



(A) **(B)**

Figure 5.695: Initial and final graph of the POLYOMINO constraint

Signature From the graph property $\mathbf{NVERTEX} = |\texttt{CELLS}|$ and from the restriction $|\texttt{CELLS}| \geq 1$ we have that the final graph is not empty. Therefore it contains at least one connected component. So we can rewrite $\mathbf{NCC} = 1$ to $\mathbf{NCC} \leq 1$ and simplify $\overline{\underline{\mathbf{NCC}}}$ to $\overline{\mathbf{NCC}}$.

5.331 POWER

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin [146]

Constraint POWER(X, N, Y)

Synonym XEXPYEQZ.

Arguments

X : dvar
N : dvar
Y : dvar

Restrictions

$X \geq 0$
$N \geq 0$
$Y \geq 0$

Purpose Enforce the fact that Y is equal to X^N.

Example $(2, 3, 8)$

The POWER constraint holds since 8 is equal to 2^3.

Typical

$X > 1$
$N > 1$
$N < 5$
$Y > 1$

Arg. properties Functional dependency: Y determined by X and N.

Algorithm In [146] a filtering algorithm for the POWER constraint was automatically derived from the algorithm that multiplies X by itself N times by using constructive disjunction and abstract interpretation in order to approximate the behaviour of the while loop of that algorithm.

Systems XEXPYEQZ in **JaCoP**.

See also **common keyword:** GCD *(abstract interpretation)*.

Keywords **constraint arguments:** ternary constraint, pure functional dependency.

constraint type: arithmetic constraint, predefined constraint.

filtering: abstract interpretation.

modelling: functional dependency.

5.332 PRECEDENCE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Scheduling

Constraint PRECEDENCE(TASKS)

Argument TASKS : collection(origin−dvar, duration−dvar)

Restrictions
required(TASKS, [origin, duration])
TASKS.duration ≥ 0

Purpose All consecutive pairs of tasks of the collection TASKS should be ordered (i.e., the end of the first task of a pair should be less than or equal to the start of the second task of the same pair).

Example

$$\left(\left\langle\begin{array}{ll} \texttt{origin}-1 & \texttt{duration}-3, \\ \texttt{origin}-4 & \texttt{duration}-0, \\ \texttt{origin}-5 & \texttt{duration}-2, \\ \texttt{origin}-8 & \texttt{duration}-1 \end{array}\right\rangle\right)$$

Since the tasks are ordered (i.e., $1+3 \leq 4$, $4+0 \leq 5$, $5+2 \leq 8$) the PRECEDENCE constraint holds.

Typical
$|\texttt{TASKS}| > 2$
TASKS.duration ≥ 1

Symmetries

- TASKS.duration can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin attribute of all items of TASKS.

Arg. properties Contractible wrt. TASKS.

See also
common keyword: INCREASING *(order constraint)*.
implies: DISJUNCTIVE.
implies (items to collection): LEX_CHAIN_LESSEQ.

Keywords
constraint type: decomposition, order constraint.
filtering: arc-consistency.

Arc input(s)	TASKS
Arc generator	$PATH \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	tasks1.origin + tasks1.duration $\leq$ tasks2.origin
Graph property(ies)	**NARC**$= \|$TASKS$\| - 1$

Graph model Since we are only interested by the constraints linking two consecutive items of the collection TASKS we use $PATH$ to generate the arcs of the initial graph.

Parts (A) and (B) of Figure 5.696 respectively show the initial and final graph of the first example of the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

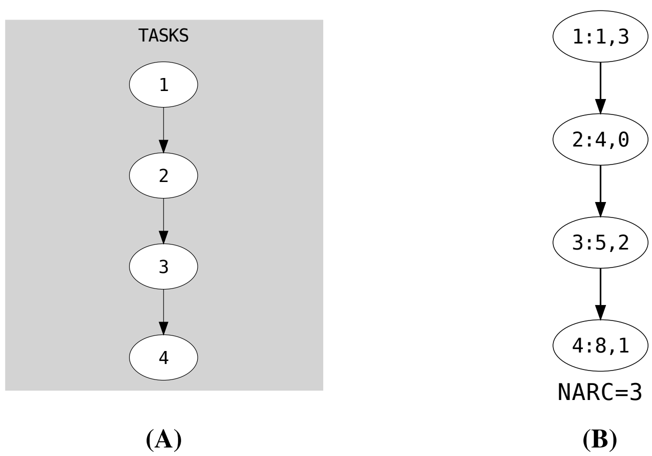


Figure 5.696: Initial and final graph of the PRECEDENCE constraint

5.333 PRODUCT_CTR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Arithmetic constraint.

Constraint PRODUCT_CTR(VARIABLES, CTR, VAR)

Arguments

VARIABLES	:	collection(var−dvar)
CTR	:	atom
VAR	:	dvar

Restrictions

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose Constraint the product of a set of domain variables. More precisely, let P denote the product of the variables of the VARIABLES collection. Enforce the following constraint to hold: P CTR VAR.

Example $(\langle 2, 1, 4\rangle, =, 8)$

The PRODUCT_CTR constraint holds since its last argument VAR $= 8$ is equal (i.e., CTR is set to $=$) to $2 \cdot 1 \cdot 4$.

Typical

$|\text{VARIABLES}| > 1$
$|\text{VARIABLES}| < 10$
range(VARIABLES.var) > 1
VARIABLES.var $\neq 0$
CTR $\in [=, <, \geq, >, \leq]$
VAR $\neq 0$

Symmetry Items of VARIABLES are permutable.

Arg. properties

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) > 0.
- Aggregate: VARIABLES(union), CTR(id), VAR($*$) when CTR $\in [=]$.

Used in CUMULATIVE_PRODUCT.

See also **common keyword:** RANGE_CTR, SUM_CTR *(arithmetic constraint)*.

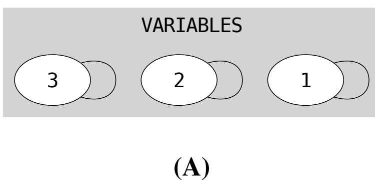
Keywords **characteristic of a constraint:** product.

constraint type: arithmetic constraint.

Arc input(s)	VARIABLES
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	TRUE
Graph property(ies)	**PROD**(VARIABLES, var) CTR VAR

Graph model Since we want to keep all the vertices of the initial graph we use the *SELF* arc generator together with the TRUE arc constraint. This predefined arc constraint always holds.

Parts (A) and (B) of Figure 5.697 respectively show the initial and final graph associated with the **Example** slot. Since we use the TRUE arc constraint both graphs are identical.



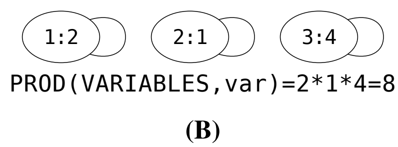


Figure 5.697: Initial and final graph of the PRODUCT_CTR constraint

5.334 PROPER_CIRCUIT

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Derived from CIRCUIT

Constraint PROPER_CIRCUIT(NODES)

Synonym CIRCUIT.

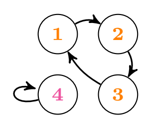
Argument NODES : collection(index−int, succ−dvar)

Restrictions
|NODES| > 1
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |NODES|

Purpose Enforce to cover a digraph G described by the NODES collection with one circuit visiting once a subset of the vertices of G.

Example

$$\left(\left\langle\begin{array}{ll}\text{index}-1 & \text{succ}-2,\\ \text{index}-2 & \text{succ}-3,\\ \text{index}-3 & \text{succ}-1,\\ \text{index}-4 & \text{succ}-4\end{array}\right\rangle\right)$$



The PROPER_CIRCUIT constraint holds since its NODES argument depicts the following circuit visiting successively the vertices 1, 2, 3 and 1 (i.e., node 4 is not visited).

All solutions Figure 5.698 gives all solutions to the following non ground instance of the PROPER_CIRCUIT constraint: $S_1 \in [2,4]$, $S_2 \in [1,2]$, $S_3 \in [1,4]$, $S_4 \in [2,4]$, PROPER_CIRCUIT($\langle 1\ S_1, 2\ S_2, 3\ S_3, 4\ S_4\rangle$).

Typical |NODES| > 2

Symmetry Items of NODES are permutable.

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	1	5	20	84	409	2365	16064	125664	1112073

Number of solutions for PROPER_CIRCUIT: domains $0..n$

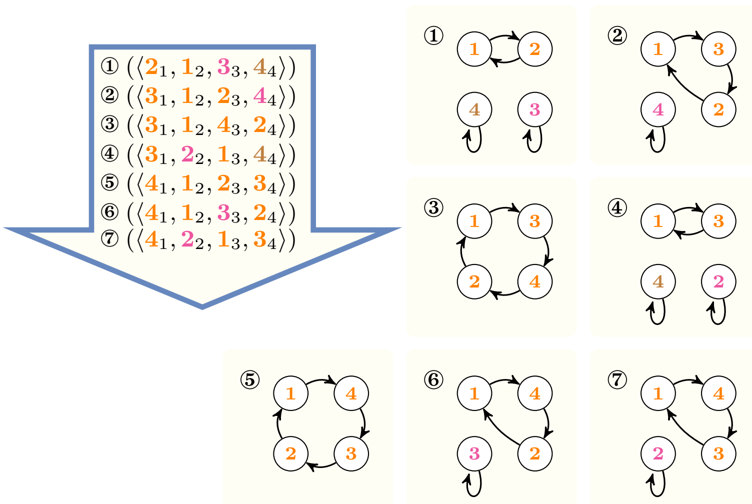
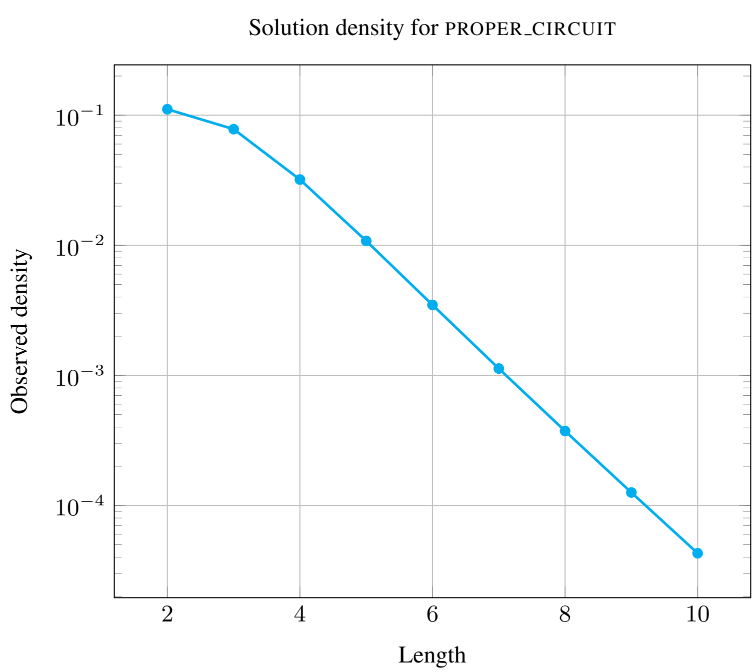
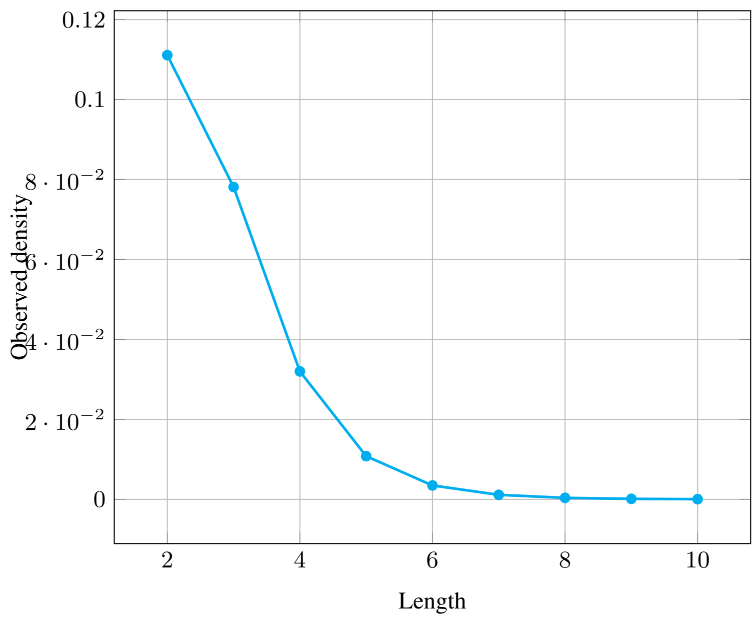


Figure 5.698: All solutions corresponding to the non ground example of the PROPER_CIRCUIT constraint of the **All solutions** slot; in the left-hand side the `index` attributes are displayed as indices of the `succ` attribute, while in the right-hand side they are directly displayed within each node.



Solution density for PROPER_CIRCUIT



See also

common keyword: ALLDIFFERENT *(permutation)*, CIRCUIT *(permutation, one_succ)*, PATH *(graph partitioning constraint, one_succ)*.

implied by: CIRCUIT.

implies: PERMUTATION, TWIN.

implies (items to collection): LEX_ALLDIFFERENT.

Keywords

combinatorial object: permutation.

constraint type: predefined constraint, graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck.

final graph structure: circuit, one_succ.

5.335 PROPER_FOREST

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from TREE, [52].

Constraint PROPER_FOREST(NTREES, NODES)

Arguments

NTREES : dvar
NODES : collection(index−int, neighbour−svar)

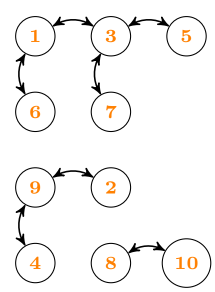
Restrictions

NTREES ≥ 0
required(NODES, [index, neighbour])
$|$NODES$|$ mod $2 = 0$
NODES.index ≥ 1
NODES.index $\leq |$NODES$|$
distinct(NODES, index)
NODES.neighbour ≥ 1
NODES.neighbour $\leq |$NODES$|$
NODES.neighbour $\neq$ NODES.index

Purpose Cover an undirected graph G by a set of NTREES trees (i.e., a *tree* is a connected graph without cycles that contains at least two vertices [114]) in such a way that each vertex of G belongs to one distinct tree.

Example

$$\left(3,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{neighbour}-\{3,6\},\\ \texttt{index}-2 & \texttt{neighbour}-\{9\},\\ \texttt{index}-3 & \texttt{neighbour}-\{1,5,7\},\\ \texttt{index}-4 & \texttt{neighbour}-\{9\},\\ \texttt{index}-5 & \texttt{neighbour}-\{3\},\\ \texttt{index}-6 & \texttt{neighbour}-\{1\},\\ \texttt{index}-7 & \texttt{neighbour}-\{3\},\\ \texttt{index}-8 & \texttt{neighbour}-\{10\},\\ \texttt{index}-9 & \texttt{neighbour}-\{2,4\},\\ \texttt{index}-10 & \texttt{neighbour}-\{8\}\end{array}\right\rangle\right)$$



The PROPER_FOREST constraint holds since the undirected graph associated with the items of the NODES collection corresponds to a forest containing NTREES $= 3$ trees: each tree respectively involves the vertices $\{1,3,5,6,7\}$, $\{2,4,9\}$ and $\{8,10\}$.

Typical

NTREES > 0
$|$NODES$| > 1$

Symmetry Items of NODES are permutable.

Arg. properties Functional dependency: NTREES determined by NODES.

Remark Extension to the *minimum spanning tree* constraint is described in [152, 360, 363].

Algorithm A filtering algorithm for the PROPER_FOREST constraint was proposed by N. Beldiceanu *et al.* in [52]. It achieves hybrid-consistency and its running time is dominated by the complexity of finding all edges that do not belong to any maximum cardinality matching in an undirected n-vertex, m-edge graph, i.e., $O(m \cdot n)$. A second filtering algorithm with a worst-case time complexity of $O(m \cdot p)$, where p is the number of maximum extreme sets of the graph, based on Gallai-Edmonds decomposition [190, 159] was proposed by R. Cymer in [140].

Systems TREE in **Choco**.

See also **common keyword:** TREE *(connected component,tree)*.

Keywords **characteristic of a constraint:** undirected graph.

constraint arguments: constraint involving set variables.

constraint type: graph constraint.

filtering: hybrid-consistency.

final graph structure: connected component, tree, no cycle, symmetric.

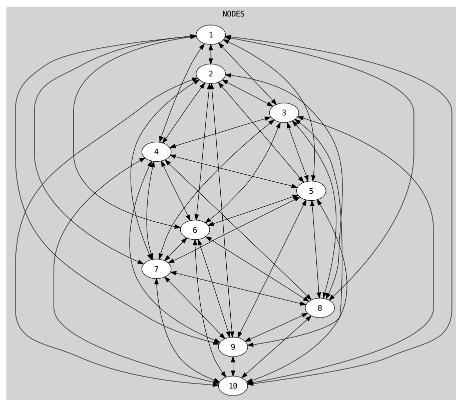
modelling: functional dependency.

Arc input(s)	NODES
Arc generator	$CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.neighbour)
Graph property(ies)	• **NVERTEX**$= ($**NARC**$+2 *$ NTREES$)/2$ • **NCC**$=$ NTREES • **NVERTEX**$= \|$NODES$\|$
Graph class	SYMMETRIC

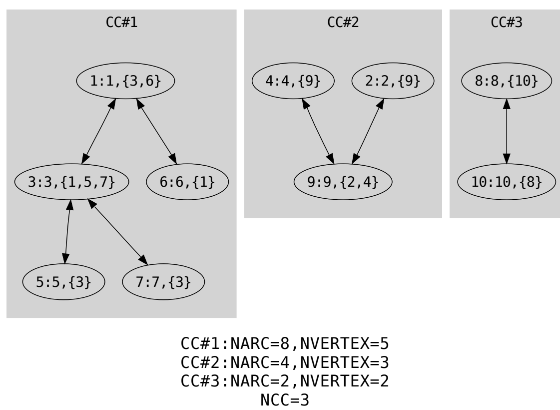
Graph model The graph constraint forces the following conditions:

- Each connected component of the final graph has n vertices and $2 \cdot (n-1)$ arcs. This is equivalent to the fact that each connected component has not any cycle.
- Since we use the $CLIQUE(\neq)$ arc-generator and since, by definition, the final graph does not contain any isolated vertex, each connected component of the final graph involves more than one vertex.
- The number of connected components of the final graph is equal to **NCC**.
- All the vertices of the initial graph belong to the final graph.
- The final graph is symmetric.

Parts (A) and (B) of Figure 5.699 respectively show the initial and final graph associated with the **Example** slot. For each connected component we display its number of arcs as well as its number of vertices. The PROPER_FOREST constraint holds since the final graph has NTREES $=$ **NCC** $= 3$ connected components and no cycle.



(A)



(B)

Figure 5.699: Initial and final graph of the PROPER_FOREST constraint

5.336 RANGE_CTR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Arithmetic constraint.

Constraint RANGE_CTR(VARIABLES, CTR, R)

Arguments

VARIABLES	:	collection(var−dvar)
CTR	:	atom
R	:	dvar

Restrictions

$|\texttt{VARIABLES}| > 0$
required(VARIABLES, var)
$\texttt{CTR} \in [=, \neq, <, \geq, >, \leq]$

Purpose Constraint the difference between the maximum value and the minimum value of a set of domain variables. More precisely, let RANGE denote the difference between the largest and the smallest variables of the VARIABLES collection plus one. Enforce the following constraint to hold: RANGE CTR R.

Example $(\langle 1, 9, 4\rangle, =, 9)$

The RANGE_CTR constraint holds since $\max(1, 9, 4) - \min(1, 9, 4) + 1$ is equal (i.e., CTR is set to $=$) to its last argument $\texttt{R} = 9$.

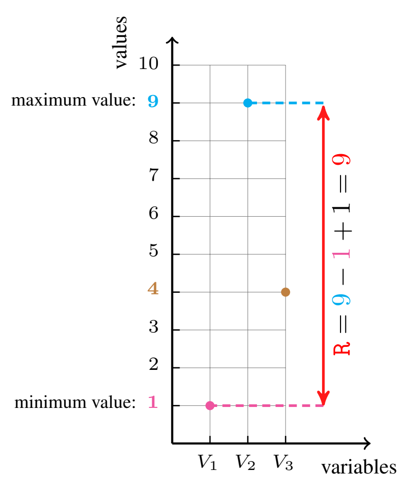


Figure 5.700: Illustration of the **Example** slot: three variables respectively fixed to values 1, 9 and 4, and their corresponding range $\texttt{R} = 9$

Typical

$|\texttt{VARIABLES}| > 1$
range$(\texttt{VARIABLES.var}) > 1$
$\texttt{CTR} \in [=, <, \geq, >, \leq]$

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Contractible wrt. VARIABLES when $\texttt{CTR} \in [<, \leq]$.
- Extensible wrt. VARIABLES when $\texttt{CTR} \in [\geq, >]$.

Used in SHIFT.

See also **common keyword:** PRODUCT_CTR, SUM_CTR *(arithmetic constraint)*.

Keywords **characteristic of a constraint:** range.

constraint type: arithmetic constraint.

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	TRUE
Graph property(ies)	**RANGE**(VARIABLES, var) CTR R

Graph model Since we want to keep all the vertices of the initial graph we use the *SELF* arc generator together with the TRUE arc constraint. This predefined arc constraint always holds.

Parts (A) and (B) of Figure 5.701 respectively show the initial and final graph associated with the **Example** slot. Since we use the TRUE arc constraint both graphs are identical.

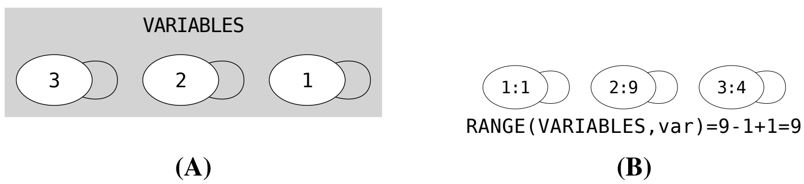


Figure 5.701: Initial and final graph of the RANGE_CTR constraint

5.337 RELAXED_SLIDING_SUM

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin CHIP

Constraint RELAXED_SLIDING_SUM(ATLEAST, ATMOST, LOW, UP, SEQ, VARIABLES)

Arguments

ATLEAST	:	int
ATMOST	:	int
LOW	:	int
UP	:	int
SEQ	:	int
VARIABLES	:	collection(var−dvar)

Restrictions

$\texttt{ATLEAST} \geq 0$
$\texttt{ATMOST} \geq \texttt{ATLEAST}$
$\texttt{ATMOST} \leq |\texttt{VARIABLES}| - \texttt{SEQ} + 1$
$\texttt{UP} \geq \texttt{LOW}$
$\texttt{SEQ} > 0$
$\texttt{SEQ} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose There are between ATLEAST and ATMOST sequences of SEQ consecutive variables of the collection VARIABLES such that the sum of the variables of the sequence is in [LOW, UP].

Example $(3, 4, 3, 7, 4, \langle 2, 4, 2, 0, 0, 3, 4\rangle)$

Within the sequence 2 4 2 0 0 3 4 we have exactly 3 subsequences of $\texttt{SEQ} = 4$ consecutive values such that their sums is located within the interval $[\texttt{LOW}, \texttt{UP}] = [3, 7]$: subsequences 4 2 0 0, 2 0 0 3 and 0 0 3 4. Consequently the RELAXED_SLIDING_SUM constraint holds since the number of such subsequences is located within the interval $[\texttt{ATLEAST}, \texttt{ATMOST}] = [3, 4]$.

Typical

$\texttt{SEQ} > 1$
$\texttt{SEQ} < |\texttt{VARIABLES}|$
$\text{range}(\texttt{VARIABLES.var}) > 1$
$\texttt{ATLEAST} > 0 \vee \texttt{ATMOST} < |\texttt{VARIABLES}| - \texttt{SEQ} + 1$

Symmetries

- ATLEAST can be decreased to any value ≥ 0.
- ATMOST can be increased to any value $\leq |\texttt{VARIABLES}| - \texttt{SEQ} + 1$.
- Items of VARIABLES can be reversed.

Algorithm [32].

See also

hard version: SLIDING_SUM.

used in graph description: SUM_CTR *(the sliding constraint)*.

Keywords

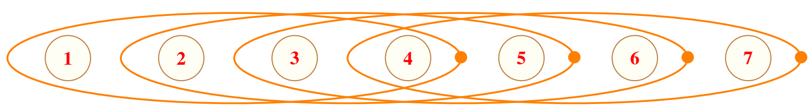
characteristic of a constraint: hypergraph.

combinatorial object: sequence.

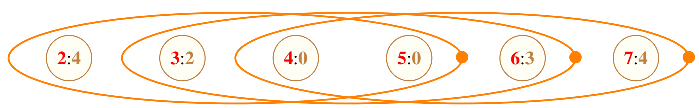
constraint type: sliding sequence constraint, soft constraint, relaxation.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$collection
Arc arity	SEQ
Arc constraint(s)	• SUM_CTR(collection, $\geq$, LOW) • SUM_CTR(collection, $\leq$, UP)
Graph property(ies)	• **NARC** $\geq$ ATLEAST • **NARC** $\leq$ ATMOST

Graph model Parts (A) and (B) of Figure 5.702 respectively show the initial and final graph associated with the **Example** slot. For each vertex of the graph we show its corresponding position within the collection of variables. The constraint associated with each arc corresponds to a conjunction of two sum_ctr constraints involving 4 consecutive variables. In Part (B), we did not put vertex 1 since the single arc constraint that mentions vertex 1 does not hold (i.e., the sum $2 + 4 + 2 + 0 = 8$ is not located in interval $[3, 7]$). However, the directed hypergraph contains 3 arcs, so the RELAXED_SLIDING_SUM constraint is satisfied since it was requested to have between 3 and 4 arcs.



(A)



(B)

Figure 5.702: (A) Initial and (B) final graph of the RELAXED_SLIDING_SUM$(3, 4, 3, 7, \mathbf{4}, \langle 2, \mathbf{4}, \mathbf{2}, \mathbf{0}, \mathbf{0}, \mathbf{3}, \mathbf{4}\rangle)$ constraint of the **Example** slot where each ellipse represents an hyperedge involving SEQ $=$ **4** vertices (e.g., the rightmost ellipse represents the constraint $0 + 0 + 3 + 4 \in [3, 7]$)

5.338 REMAINDER

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint REMAINDER(Q, D, R)

Synonyms MODULO, MOD.

Arguments

Q : dvar
D : dvar
R : dvar

Restrictions

$\mathtt{Q} \geq 0$
$\mathtt{D} > 0$
$\mathtt{R} \geq 0$
$\mathtt{R} < \mathtt{D}$

Purpose Enforce R to be equal to the remainder of the division of Q by D.

Example $(15, 2, 1)$

The REMAINDER constraint holds since 1 is the rest of the division of 15 by 2.

Arg. properties Functional dependency: R determined by Q and D.

Keywords

constraint arguments: ternary constraint, pure functional dependency.
constraint type: predefined constraint, arithmetic constraint.
modelling: functional dependency.

5.339 ROOTS

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [69]

Constraint ROOTS(S, T, VARIABLES)

Arguments

S	:	svar
T	:	svar
VARIABLES	:	collection(var−dvar)

Restrictions

$\texttt{S} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose

S is the set of indices of the variables in the collection VARIABLES taking their values in T; $\texttt{S} = \{i \mid \texttt{VARIABLES}[i].\texttt{var} \in \texttt{T}\}$. Positions are numbered from 1.

Example

$(\{2,4,5\}, \{2,3,8\}, \langle 1,3,1,2,3\rangle)$

The ROOTS constraint holds since values 2 and 3 in T occur in the collection $\langle 1,3,1,2,3\rangle$ only at positions $\texttt{S} = \{2,4,5\}$. The value $8 \in \texttt{T}$ does not occur within the collection $\langle 1,3,1,2,3\rangle$.

Typical

$|\texttt{VARIABLES}| > 1$
range(VARIABLES.var) > 1

Usage

Bessière et al. showed [69] that many counting and occurence constraints can be specified with two *global primitives*: ROOTS and RANGE. For example, the COUNT constraint can be decomposed into one ROOTS constraint: COUNT(VAL, VARS, OP, NVAR) iff ROOTS(S, {VAL}, VARS) $\wedge$ |S| OP NVAR.

ROOTS does not count but collects the set of variables using particular values. It provides then a way of channeling. ROOTS generalises, for example, the LINK_SET_TO_BOOLEANS constraint, LINK_SET_TO_BOOLEANS(S, BOOLEANS) iff ROOTS(S, {1}, BOOLEANS.bool), or may be used instead of the DOMAIN_CONSTRAINT.

Other examples of reformulations are given in [73].

Algorithm

In [72], Bessière *et al.* shows that enforcing hybrid-consistency on ROOTS is NP-hard. They consider the decomposition of ROOTS into a network of ternary constraints: $\forall i$, $i \in \texttt{S} \Rightarrow \texttt{VARIABLES}[i].\texttt{var} \in \texttt{T}$ and $\texttt{VARIABLES}[i].\texttt{var} \Rightarrow \texttt{T} \wedge i \in \texttt{S}$. Enforcing bound consistency on the decomposition achieves bound consistency on ROOTS. Enforcing hybrid consistency on the decomposition achieves at least bound consistency on ROOTS, until hybrid consistency in some special cases:

- $dom(\texttt{VARIABLES}[i].\texttt{var}) \subset \underline{\texttt{T}}$, $\forall i \in \underline{\texttt{S}}$,
- $dom(\texttt{VARIABLES}[i].\texttt{var}) \cap \overline{\texttt{T}} = \emptyset$, $\forall i \notin \overline{\texttt{S}}$,

- VARIABLES are ground,
- T is ground.

Enforcing hybrid consistency on the decomposition can be done in $O(nd)$ with $n = |\texttt{VARIABLES}|$ and d the maximum domain size of VARIABLES$[i]$.var and T.

Systems ROOTS in **Gecode**, ROOTS in **MiniZinc**.

See also **common keyword:** LINK_SET_TO_BOOLEANS *(constraint involving set variables)*.

related: AMONG *(can be expressed with* ROOTS*)*, ASSIGN_AND_NVALUES *(can be expressed with* ROOTS *and* RANGE*)*, ATLEAST, ATMOST *(can be expressed with* ROOTS*)*, COMMON *(can be expressed with* ROOTS *and* RANGE*)*, COUNT *(can be expressed with* ROOTS*)*, DOMAIN_CONSTRAINT, GLOBAL_CARDINALITY, GLOBAL_CONTIGUITY *(can be expressed with* ROOTS*)*, SYMMETRIC_ALLDIFFERENT, USES *(can be expressed with* ROOTS *and* RANGE*)*.

Keywords **characteristic of a constraint:** disequality.

constraint arguments: constraint involving set variables.

constraint type: counting constraint, value constraint, decomposition.

filtering: hybrid-consistency.

Derived Collection $\texttt{col}(\texttt{SETS}-\texttt{collection}(\texttt{s}-\texttt{svar},\texttt{t}-\texttt{svar}),[\texttt{item}(\texttt{s}-\texttt{S},\texttt{t}-\texttt{T})])$

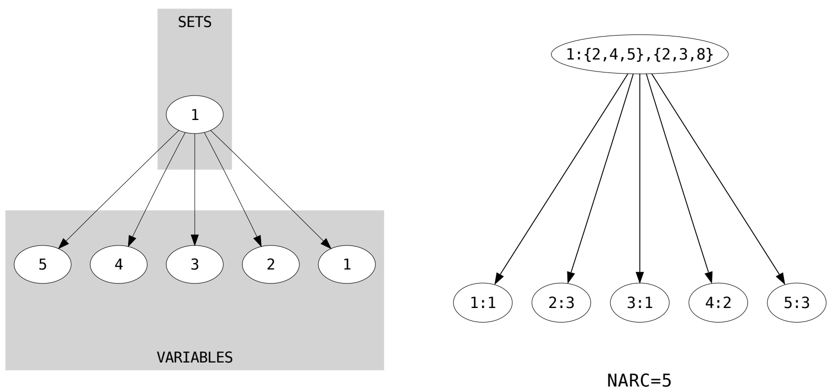
Arc input(s) SETS VARIABLES

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{sets},\texttt{variables})$

Arc arity 2

Arc constraint(s) $\text{IN\_SET}(\texttt{variables.key},\texttt{sets.s}) \Leftrightarrow$
$\text{IN\_SET}(\texttt{variables.var},\texttt{sets.t})$

Graph property(ies) $\mathbf{NARC} = |\texttt{VARIABLES}|$



(A) **(B)**

Figure 5.703: Initial and final graph of the ROOTS constraint

Graph model

5.340 SAME

DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint SAME(VARIABLES1, VARIABLES2)

Arguments

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

Restrictions

|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose The variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation.

Example $(\langle 1, 9, 1, 5, 2, 1\rangle, \langle 9, 1, 1, 1, 2, 5\rangle)$

The SAME constraint holds since values 1, 2, 5 and 9 have the same number of occurrences within both collections $\langle 1, 9, 1, 5, 2, 1\rangle$ and $\langle 9, 1, 1, 1, 2, 5\rangle$. Figure 5.704 illustrates this correspondence.

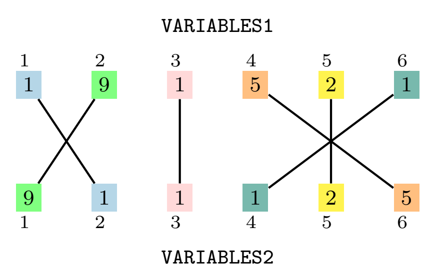


Figure 5.704: Illustration of the correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot

All solutions Figure 5.705 gives all solutions to the following non ground instance of the SAME constraint: $U_1 \in [0, 2]$, $U_2 \in [1, 2]$, $U_3 \in [1, 2]$, $V_1 \in [0, 1]$, $V_2 \in [2, 4]$, $V_3 \in [2, 3]$, SAME($\langle U_1, U_2, U_3\rangle, \langle V_1, V_2, V_3\rangle$).

Typical

|VARIABLES1| > 1
range(VARIABLES1.var) > 1
range(VARIABLES2.var) > 1

① $(\langle 0, 2, 2\rangle, \langle 0, 2, 2\rangle)$
② $(\langle 1, 2, 2\rangle, \langle 1, 2, 2\rangle)$
③ $(\langle 2, 1, 2\rangle, \langle 1, 2, 2\rangle)$
④ $(\langle 2, 2, 1\rangle, \langle 1, 2, 2\rangle)$

Figure 5.705: All solutions corresponding to the non ground example of the SAME constraint of the **All solutions** slot where identical values are coloured in the same way in both collections

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Arg. properties

Aggregate: VARIABLES1(union), VARIABLES2(union).

Usage

The SAME constraint can be used in the following contexts:

- Pairing problems taken from [54]. The organisation Doctors Without Borders has a list of doctors and a list of nurses, each of whom volunteered to go on one mission in the next year. Each volunteer specifies a list of possible dates and each mission involves one doctor and one nurse. The task is to produce a list of pairs such that each pair includes a doctor and a nurse who are available at the same date and each volunteer appears in exactly one pair. The problem is modelled by a SAME$(D = d_1, d_2, \ldots, d_m, N = n_1, n_2, \ldots, n_m)$ constraint where each doctor is represented by a domain variable in D and each nurse by a domain variable in N. For a given doctor or nurse the corresponding domain variable gives the dates when the person is available. When the number of nurses is different from the number of doctors we replace the SAME constraint by a USED_BY constraint.
- Timetabling problems where we wish to produce fair schedules for different persons is a second use of the SAME constraint. Assume we need to generate a plan over a period of D consecutive days for P persons. For each day d and each person p we need to decide whether person p works in the morning shift, in the afternoon shift, in the night shift or does not work at all on day d. In a fair schedule, the number of morning shifts should be the same for all the persons. The same condition holds for the afternoon and the night shifts as well as for the days off. We create for each person p the sequence of variables $v_{p,1}, v_{p,2}, \ldots, v_{p,D}$. $v_{p,D}$ is equal to one of $0, 1, 2$ and 3, depending on whether person p does not work, works in the morning, in the afternoon or during the night on day d. We can use $P-1$ SAME constraints to express the fact that $v_{1,1}, v_{1,2}, \ldots, v_{1,D}$ should be a permutation of $v_{p,1}, v_{p,2}, \ldots, v_{p,D}$ for each $(1 < p \leq P)$.
- The SAME constraint can also be used as a channelling constraint for modelling the following recurring pattern: given the number of 1s in each line and each column of

a 0-1 matrix $\mathcal{M}$ with n rows and m columns, reconstruct the matrix. This pattern usually occurs with additional constraints about compatible positions of the 1s, or about the overall shape reconstructed from all the 1's (e.g., convexity, connectivity). If we restrict ourselves to the basic pattern there is an $O(mn)$ algorithm for reconstructing a $m \cdot n$ matrix from its horizontal and vertical directions [189]. We show how to model this pattern with the SAME constraint. Let l_i $(1 \leq i \leq n)$ and c_j $(1 \leq j \leq m)$ denote respectively, the required number of 1s in the i^{th} row and the j^{th} column of $\mathcal{M}$. We number the entries of the matrix as shown in the left-hand side of 5.706. For row i we create l_i domain variables v_{ik} where $k \in [1, l_i]$. Similarly, for each column j we create c_j domain variables u_{jk} where $k \in [1, c_i]$. The domain of each variable contains the set of entries that belong to the row or column that the variable corresponds to. Thus, each domain variable represents a 1 that appears in the designated row or column. Let $\mathcal{V}$ be the set of variables corresponding to rows and $\mathcal{U}$ be the set of variables corresponding to columns. To make sure that each 1 is placed in a different entry, we impose the constraint ALLDIFFERENT$(\mathcal{U})$. In addition, the constraint SAME$(\mathcal{U}, \mathcal{V})$ enforces that the 1s exactly coincide on the rows and the columns. A solution is shown on the right-hand side of 5.706. Note that the SAME_AND_GLOBAL_CARDINALITY constraint allows one to model the matrix reconstruction problem without the additional ALLDIFFERENT constraint.

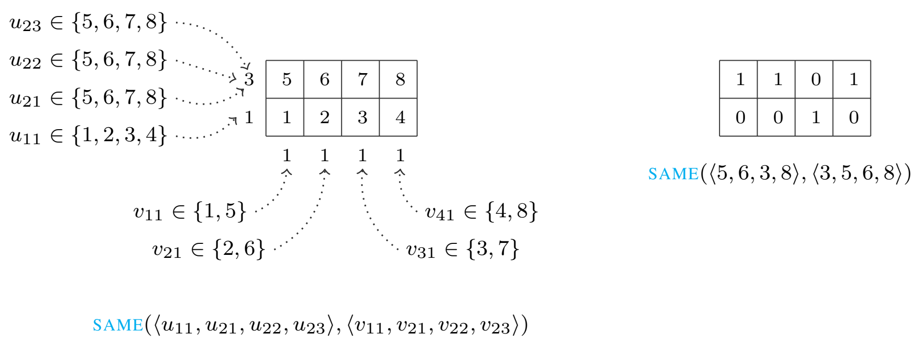


Figure 5.706: Modelling the 0-1 matrix reconstruction problem with the SAME constraint (variable u_{11} corresponds to the position of value 1 in the first row, variables u_{21}, u_{22}, u_{23} correspond to the position of value 1 in the second row, and variables v_{11}, v_{21}, v_{31}, v_{41} respectively to the positions of value 1 in the first, second, third and fourth columns)

Remark The SAME constraint is a relaxed version of the SORT constraint introduced in [308]. We do not enforce the second collection of variables to be sorted in increasing order.

If we interpret the collections `VARIABLES1` and `VARIABLES2` as two multisets variables [251], the SAME constraint can be considered as an equality constraint between two multisets variables.

The SAME constraint can be modelled by two GLOBAL_CARDINALITY constraints. For example, the SAME constraint

$$\text{SAME}\left(\begin{array}{l}\langle \texttt{var}-x_1, \texttt{var}-x_2\rangle,\\ \langle \texttt{var}-y_1, \texttt{var}-y_2\rangle,\end{array}\right)$$

where the union of the domains of the different variables is $\{1,2,3,4\}$ corresponds to the conjunction of the following two GLOBAL_CARDINALITY constraints:

$$\text{GLOBAL\_CARDINALITY}\left(\begin{array}{l}\langle\ \mathtt{var}-x_1,\mathtt{var}-x_2\ \rangle,\\ \left\langle\begin{array}{ll}\mathtt{val}-1 & \mathtt{noccurrence}-c_1,\\ \mathtt{val}-2 & \mathtt{noccurrence}-c_2,\\ \mathtt{val}-3 & \mathtt{noccurrence}-c_3,\\ \mathtt{val}-4 & \mathtt{noccurrence}-c_4\end{array}\right\rangle\end{array}\right)$$

$$\text{GLOBAL\_CARDINALITY}\left(\begin{array}{l}\langle\ \mathtt{var}-y_1,\mathtt{var}-y_2\ \rangle,\\ \left\langle\begin{array}{ll}\mathtt{val}-1 & \mathtt{noccurrence}-c_1,\\ \mathtt{val}-2 & \mathtt{noccurrence}-c_2,\\ \mathtt{val}-3 & \mathtt{noccurrence}-c_3,\\ \mathtt{val}-4 & \mathtt{noccurrence}-c_4\end{array}\right\rangle\end{array}\right)$$

As shown by the next example, the consistency for all variables of the two GLOBAL_CARDINALITY constraints does not implies consistency for the corresponding SAME constraint. This is the case, for example, when the domains of x_1, x_2, y_1 and y_2 is respectively equal to $\{1,2\}$, $\{3,4\}$, $\{1,2,3,4\}$ and $\{3,4\}$. The conjunction of the two GLOBAL_CARDINALITY constraints does not remove values 3 and 4 from y_1.

In his PhD thesis, W.-J. van Hoeve introduces a soft version of the SAME constraint where the cost is the minimum number of variables to assign differently in order to get back to a solution [434, page 78]. In the context of the SAME constraint this violation cost corresponds to the difference between the number of variables in VARIABLES1 and the number of values that both occur in VARIABLES1 and in VARIABLES2 (provided that one value of VARIABLES1 matches at most one value of VARIABLES2).

Algorithm In [53, 54, 55, 242], it is shown how to model this constraint by a flow network that enables to compute arc-consistency and bound-consistency. The rightmost part of Figure 3.31 illustrates this flow model. Unlike the networks used for ALLDIFFERENT and GLOBAL_CARDINALITY, the network now has three sets of nodes, so the algorithms are more complex, in particular the efficient bound-consistency algorithm.

More recently [138, 139] presents a second filtering algorithm also achieving arc-consistency based on a mapping of the solutions to the SAME constraint to perfect matchings in a bipartite intersection graph derived from the domain of the variables of the constraint in the following way. To each variable of the VARIABLES1 and VARIABLES2 collection corresponds a vertex of the intersection graph. There is an edge between a vertex associated with a variable of the VARIABLES1 collection and a vertex associated with a variable of the VARIABLES2 collection if and only if the corresponding variables have at least one value in common in their domains.

Reformulation The SAME(VARIABLES1, VARIABLES2) constraint can be reformulated as the conjunction SORT(VARIABLES1, SORTED_VARIABLES) $\wedge$ SORT(VARIABLES2, SORTED_VARIABLES).

Used in K_SAME.

See also **generalisation:** CORRESPONDENCE *(*PERMUTATION *parameter added)*, SAME_INTERVAL *(*variable *replaced by* variable/constant*)*, SAME_MODULO *(*variable *replaced by* variable mod constant*)*, SAME_PARTITION *(*variable *replaced by* variable $\in$ partition*)*.

implied by: LEX_EQUAL, SAME_AND_GLOBAL_CARDINALITY, SAME_AND_GLOBAL_CARDINALITY_LOW_UP, SORT.

implies: SAME_INTERSECTION, USED_BY.

related to a common problem: COLORED_MATRIX *(matrix reconstruction problem)*.

soft variant: SOFT_SAME_VAR *(variable-based violation measure)*.

system of constraints: K_SAME.

used in reformulation: SORT.

Keywords

characteristic of a constraint: sort based reformulation, automaton, automaton with array of counters.

combinatorial object: permutation, multiset.

constraint arguments: constraint between two collections of variables.

filtering: bipartite matching, flow, arc-consistency, bound-consistency, DFS-bottleneck.

modelling: channelling constraint, equality between multisets.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	• for all connected components: **NSOURCE**=**NSINK** • **NSOURCE**= |VARIABLES1| • **NSINK**= |VARIABLES2|

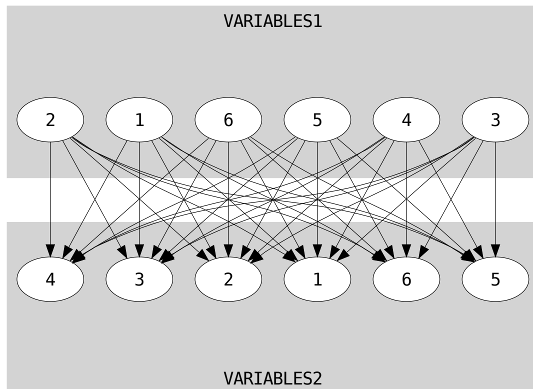
Graph model Parts (A) and (B) of Figure 5.707 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The SAME constraint holds since:

- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to |VARIABLES1|.
- The number of sinks of the final graph is equal to |VARIABLES2|.

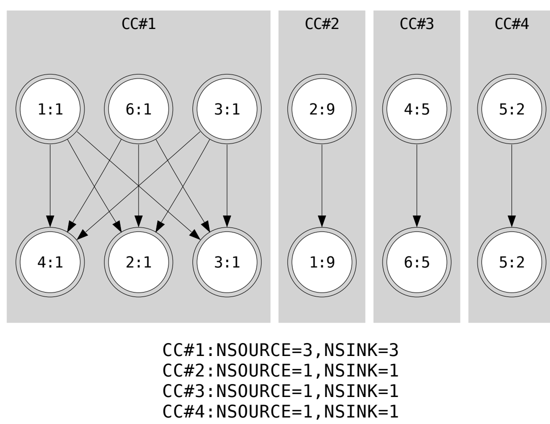
Signature Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the $PRODUCT$ arc generator on the collections VARIABLES1 and VARIABLES2, we have that the maximum number of sources and sinks of the final graph is respectively equal to |VARIABLES1| and |VARIABLES2|. Therefore we can rewrite **NSOURCE** $=$ |VARIABLES1| to **NSOURCE** $\geq$ |VARIABLES1| and simplify $\underline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite **NSINK** $=$ |VARIABLES2| to **NSINK** $\geq$ |VARIABLES2| and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.



(A)



(B)

Figure 5.707: Initial and final graph of the SAME constraint

Automaton To each item of the collection VARIABLES1 corresponds a signature variable S_i that is equal to 0. To each item of the collection VARIABLES2 corresponds a signature variable $S_{i+|\texttt{VARIABLES1}|}$ that is equal to 1.

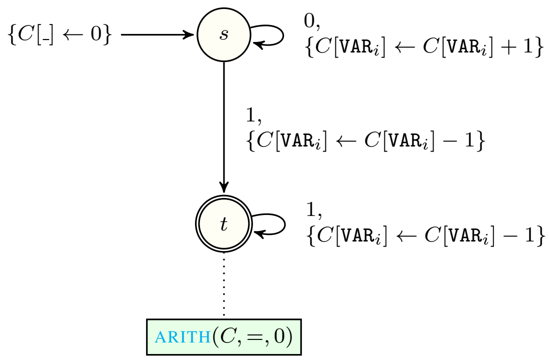


Figure 5.708: Automaton of the SAME constraint

5.341 SAME_AND_GLOBAL_CARDINALITY

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Conjoin SAME and GLOBAL_CARDINALITY

Constraint SAME_AND_GLOBAL_CARDINALITY(VARIABLES1, VARIABLES2, VALUES)

Synonyms SGCC, SAME_GCC, SAME_AND_GCC, SWC, SAME_WITH_CARDINALITIES.

Arguments

VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
VALUES	:	collection(val−int, noccurrence−dvar)

Restrictions

$|\mathtt{VARIABLES1}| = |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
required(VALUES, [val, noccurrence])
distinct(VALUES, val)
$\mathtt{VALUES.noccurrence} \geq 0$
$\mathtt{VALUES.noccurrence} \leq |\mathtt{VARIABLES1}|$

Purpose

The variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation. In addition, each value VALUES$[i]$.val (with $i \in [1, |\mathtt{VALUES}|]$) should be taken by exactly VALUES$[i]$.noccurrence variables of the VARIABLES1 collection. Finally, each variable of VARIABLES1 should be assigned a value of VALUES$[i]$.val (with $i \in [1, |\mathtt{VALUES}|]$).

Example

$$\left(\begin{array}{l} \langle 1,9,1,5,2,1\rangle, \\ \langle 9,1,1,1,2,5\rangle, \\ \left\langle\begin{array}{ll} \mathtt{val}-1 & \mathtt{noccurrence}-3, \\ \mathtt{val}-2 & \mathtt{noccurrence}-1, \\ \mathtt{val}-5 & \mathtt{noccurrence}-1, \\ \mathtt{val}-7 & \mathtt{noccurrence}-0, \\ \mathtt{val}-9 & \mathtt{noccurrence}-1 \end{array}\right\rangle \end{array}\right)$$

The SAME_AND_GLOBAL_CARDINALITY constraint holds since:

- The values 1, 9, 1, 5, 2, 1 assigned to VARIABLES1 correspond to a permutation of the values 9, 1, 1, 1, 2, 5 assigned to VARIABLES2.
- The values 1, 2, 5, 7 and 6 are respectively used 3, 1, 1, 0 and 1 times.

Typical

$|\mathtt{VARIABLES1}| > 1$
range(VARIABLES1.var) > 1
range(VARIABLES2.var) > 1
$|\mathtt{VALUES}| > 1$
range(VALUES.noccurrence) > 1
$|\mathtt{VARIABLES1}| > |\mathtt{VALUES}|$

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2) (VALUES).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of VALUES are permutable.
- An occurrence of a value of VARIABLES1.var or VARIABLES2.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.
- All occurrences of two distinct values in VARIABLES1.var, VARIABLES2.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES1.var, VARIABLES2.var or VALUES.val can be renamed to any unused value.

Arg. properties

Contractible wrt. VALUES.

Usage

See the SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraint.

Algorithm

The filtering algorithm presented in [56] can be reused for pruning the variables of the VARIABLES1 and the VARIABLES2 collection. This algorithm does not restrict the noccurrence variables of the VALUES collection.

See also

implies: GLOBAL_CARDINALITY, SAME.

related: K_ALLDIFFERENT *(two overlapping* ALLDIFFERENT *plus restriction on values)*.

specialisation: SAME_AND_GLOBAL_CARDINALITY_LOW_UP *(*variable *replaced by* fixed interval*)*.

Keywords

application area: assignment.

combinatorial object: permutation, multiset.

constraint arguments: constraint between two collections of variables.

constraint type: value constraint.

filtering: flow.

modelling: equality between multisets.

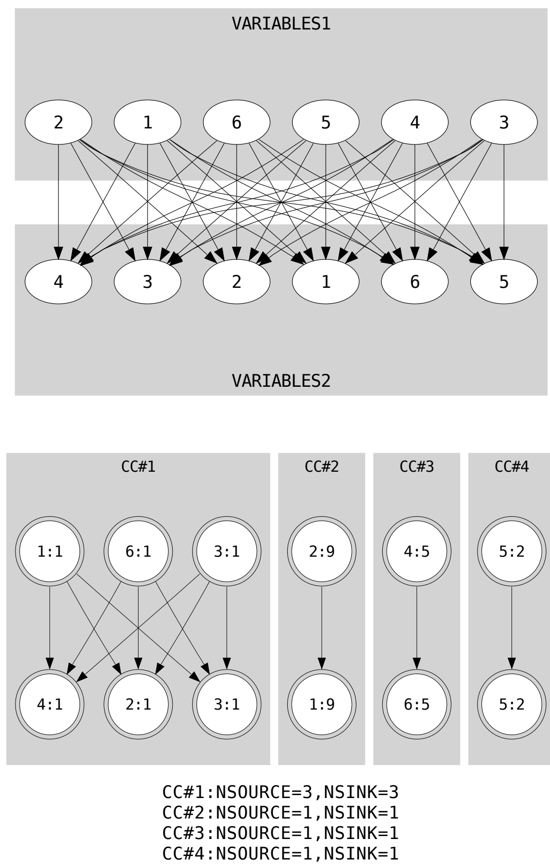
problems: demand profile.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	• for all connected components: **NSOURCE**=**NSINK** • **NSOURCE**= |VARIABLES1| • **NSINK**= |VARIABLES2|

For all items of VALUES:

Arc input(s)	VARIABLES1
Arc generator	*SELF* ↦ collection(variables)
Arc arity	1
Arc constraint(s)	variables.var = VALUES.val
Graph property(ies)	**NVERTEX**= VALUES.noccurrence

Graph model Parts (A) and (B) of Figure 5.709 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint.



(A) **(B)**

Figure 5.709: Initial and final graph of the SAME_AND_GLOBAL_CARDINALITY constraint

5.342 SAME_AND_GLOBAL_CARDINALITY_LOW_UP

DESCRIPTION LINKS GRAPH

Origin

Derived from SAME and GLOBAL_CARDINALITY_LOW_UP

Constraint

SAME_AND_GLOBAL_CARDINALITY_LOW_UP(VARIABLES1, VARIABLES2, VALUES)

Arguments

VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
VALUES	:	collection(val−int, omin−int, omax−int)

Restrictions

|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
required(VALUES, [val, omin, omax])
distinct(VALUES, val)
VALUES.omin ≥ 0
VALUES.omax ≤ |VARIABLES1|
VALUES.omin ≤ VALUES.omax

Purpose

The variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation. In addition, each value VALUES[i].val (with $i \in [1, |\text{VALUES}|]$) should be taken by at least VALUES[i].omin and at most VALUES[i].omax variables of the VARIABLES1 collection. Finally, each variable of VARIABLES1 should be assigned a value of VALUES[i].val (with $i \in [1, |\text{VALUES}|]$).

Example

$$\left(\begin{array}{l}\langle 1,9,1,5,2,1\rangle,\\ \langle 9,1,1,1,2,5\rangle,\\ \left\langle\begin{array}{lll}\text{val}-1 & \text{omin}-2 & \text{omax}-3,\\ \text{val}-2 & \text{omin}-1 & \text{omax}-1,\\ \text{val}-5 & \text{omin}-1 & \text{omax}-1,\\ \text{val}-7 & \text{omin}-0 & \text{omax}-2,\\ \text{val}-9 & \text{omin}-1 & \text{omax}-1\end{array}\right\rangle\end{array}\right)$$

The SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraint holds since:

- The values 1, 9, 1, 5, 2, 1 assigned to |VARIABLES1| correspond to a permutation of the values 9, 1, 1, 1, 2, 5 assigned to |VARIABLES2|.
- The values 1, 2, 5, 7 and 6 are respectively used 3 ($2 \le 3 \le 3$), 1 ($1 \le 1 \le 1$), 1 ($1 \le 1 \le 1$), 0 ($0 \le 0 \le 2$) and 1 ($1 \le 1 \le 1$) times.

Typical

$|\texttt{VARIABLES1}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$|\texttt{VALUES}| > 1$
$\texttt{VALUES.omin} \leq |\texttt{VARIABLES1}|$
$\texttt{VALUES.omax} > 0$
$\texttt{VALUES.omax} < |\texttt{VARIABLES1}|$
$|\texttt{VARIABLES1}| > |\texttt{VALUES}|$

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2) (VALUES).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var or VARIABLES2.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.
- Items of VALUES are permutable.
- VALUES.omin can be decreased to any value ≥ 0.
- VALUES.omax can be increased to any value $\leq |\texttt{VARIABLES1}|$.
- All occurrences of two distinct values in VARIABLES1.var, VARIABLES2.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES1.var, VARIABLES2.var or VALUES.val can be renamed to any unused value.

Arg. properties

Contractible wrt. VALUES.

Usage

The SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraint can be used for modelling the following assignment problem with a single constraint. The organisation Doctors Without Borders has a list of doctors and a list of nurses, each of whom volunteered to go on one rescue mission. Each volunteer specifies a list of possible dates and each mission should include one doctor and one nurse. In addition we have for each date the minimum and maximum number of missions that should be effectively done. The task is to produce a list of pairs such that each pair includes a doctor and a nurse who are available on the same date and each volunteer appears in exactly one pair so that for each day we build the required number of missions.

Algorithm

In [56], the flow network that was used to model the SAME constraint [53, 54] is extended to support the cardinalities. Figure 3.32 illustrates this flow model. Then, algorithms are developed to compute arc-consistency and bound-consistency.

See also

generalisation: SAME_AND_GLOBAL_CARDINALITY (`fixed interval` *replaced by* `variable`).

implies: GLOBAL_CARDINALITY_LOW_UP, GLOBAL_CARDINALITY_LOW_UP_NO_LOOP, SAME.

Keywords

application area: assignment.

combinatorial object: permutation, multiset.

constraint arguments: constraint between two collections of variables.

constraint type: value constraint.

filtering: bound-consistency, arc-consistency, flow.

modelling: equality between multisets.

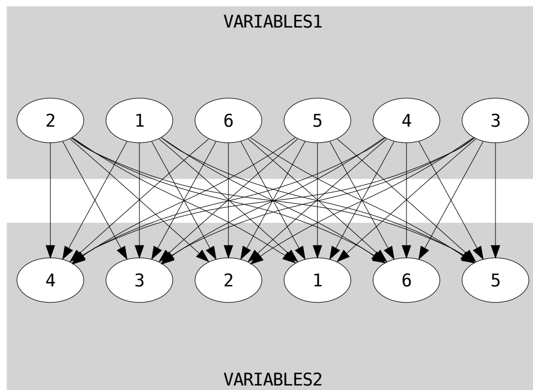
problems: demand profile.

Arc input(s)	`VARIABLES1` `VARIABLES2`
Arc generator	*PRODUCT*$\mapsto$`collection`(`variables1`, `variables2`)
Arc arity	2
Arc constraint(s)	`variables1.var` $=$ `variables2.var`
Graph property(ies)	• for all connected components: **NSOURCE**=**NSINK** • **NSOURCE**= |`VARIABLES1`| • **NSINK**= |`VARIABLES2`|

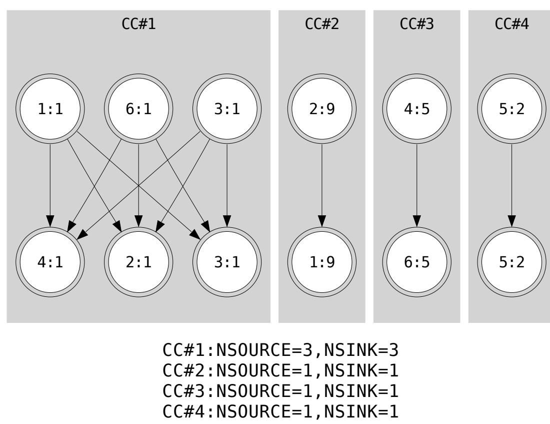
For all items of `VALUES`:

Arc input(s)	`VARIABLES1`
Arc generator	*SELF*$\mapsto$`collection`(`variables`)
Arc arity	1
Arc constraint(s)	`variables.var` $=$ `VALUES.val`
Graph property(ies)	• **NVERTEX**$\geq$ `VALUES.omin` • **NVERTEX**$\leq$ `VALUES.omax`

Graph model Parts (A) and (B) of Figure 5.710 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint.



(A)



(B)

Figure 5.710: Initial and final graph of the SAME_AND_GLOBAL_CARDINALITY_LOW_UP constraint

5.343 SAME_INTERSECTION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from SAME and COMMON.

Constraint SAME_INTERSECTION(VARIABLES1, VARIABLES2)

Arguments

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

Restrictions

required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose Each value, which occurs both in the VARIABLES1 and in the VARIABLES2 collections, has the same number of occurrences in VARIABLES1 as well as in VARIABLES2.

Example $(\langle 1,9,1,5,2,1\rangle, \langle 9,1,1,1,3,5,8\rangle)$

First note that the values, which occur both in VARIABLES1 $= \langle 1,9,1,5,2,1\rangle$ as well as in VARIABLES2 $= \langle 9,1,1,1,3,5,8\rangle$ correspond to values 1, 5, and 9. Consequently, the SAME_INTERSECTION constraint holds since these values 1, 5, and 9 have the same number of occurrences in both collections (i.e., they respectively occur 3, 1, and 1 times within VARIABLES1 and VARIABLES2).

Typical

$|\texttt{VARIABLES1}| > 1$
range(VARIABLES1.var) > 1
$|\texttt{VARIABLES2}| > 1$
range(VARIABLES2.var) > 1

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

See also **common keyword:** COMMON, NVALUE_ON_INTERSECTION *(constraint on the intersection)*.

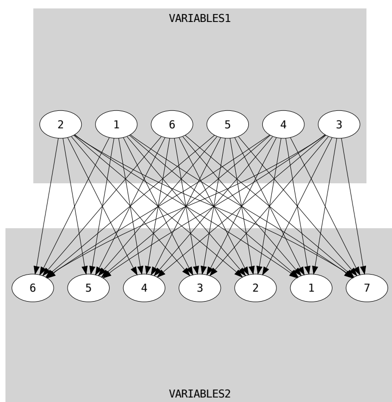
implied by: ALLDIFFERENT_ON_INTERSECTION, SAME.

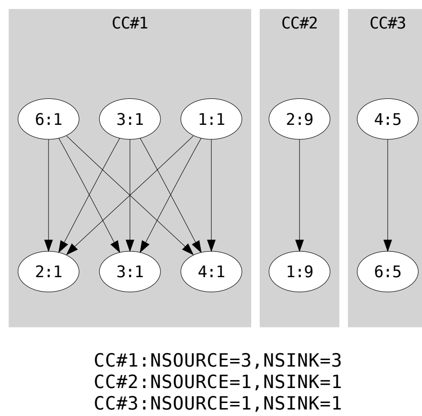
Keywords **constraint arguments:** constraint between two collections of variables.

constraint type: constraint on the intersection.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	for all connected components: **NSOURCE=NSINK**

Graph model Parts (A) and (B) of Figure 5.711 respectively show the initial and final graph associated with the **Example** slot. The SAME_INTERSECTION constraint holds since each connected component of the final graph has the same number of sources and sinks. Note that all the vertices corresponding to the variables that take values 2, 3 or 8 were removed from the final graph since there is no arc for which the associated equality constraint holds.





(A) **(B)**

Figure 5.711: Initial and final graph of the SAME_INTERSECTION constraint

5.344 SAME_INTERVAL

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from SAME.

Constraint SAME_INTERVAL(VARIABLES1, VARIABLES2, SIZE_INTERVAL)

Arguments

VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
SIZE_INTERVAL	:	int

Restrictions

|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
SIZE_INTERVAL > 0

Purpose Let N_i (respectively M_i) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval [SIZE_INTERVAL $\cdot$ i, SIZE_INTERVAL $\cdot$ i + SIZE_INTERVAL − 1. For all integer i we have $N_i = M_i$.

Example $(\langle 1, 7, 6, 0, 1, 7\rangle, \langle 8, 8, 8, 0, 1, 2\rangle, 3)$

In the example, the third argument SIZE_INTERVAL = 3 defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Consequently the values of the collection $\langle 1, 7, 6, 0, 1, 7\rangle$ are respectively located within intervals $[0, 2]$, $[6, 8]$, $[6, 8]$, $[0, 2]$, $[0, 2]$, $[6, 8]$. Therefore intervals $[0, 2]$ and $[6, 8]$ are respectively used 3 and 3 times. Similarly, the values of the collection $\langle 8, 8, 8, 0, 1, 2\rangle$ are respectively located within intervals $[6, 8]$, $[6, 8]$, $[6, 8]$, $[0, 2]$, $[0, 2]$, $[0, 2]$. As before intervals $[0, 2]$ and $[6, 8]$ are respectively used 3 and 3 times. Consequently the SAME_INTERVAL constraint holds. Figure 5.712 illustrates this correspondence.

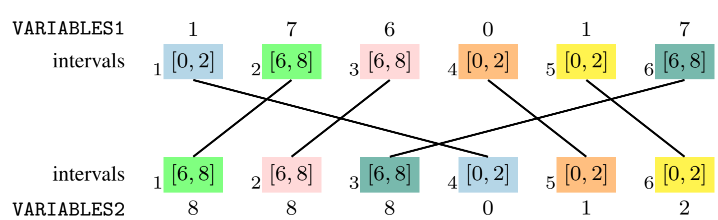


Figure 5.712: Illustration of the correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot

Typical

$|\texttt{VARIABLES1}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$\texttt{SIZE\_INTERVAL} > 1$
$\texttt{SIZE\_INTERVAL} < \texttt{range}(\texttt{VARIABLES1.var})$
$\texttt{SIZE\_INTERVAL} < \texttt{range}(\texttt{VARIABLES2.var})$

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2) (SIZE_INTERVAL).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties

Aggregate: VARIABLES1(union), VARIABLES2(union), SIZE_INTERVAL(id).

Algorithm

See algorithm of the SAME constraint.

Used in

K_SAME_INTERVAL.

See also

implies: USED_BY_INTERVAL.

soft variant: SOFT_SAME_INTERVAL_VAR *(variable-based violation measure)*.

specialisation: SAME *(*variable/constant *replaced by* variable*)*.

system of constraints: K_SAME_INTERVAL.

Keywords

characteristic of a constraint: sort based reformulation.

combinatorial object: permutation.

constraint arguments: constraint between two collections of variables.

modelling: interval.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL
Graph property(ies)	• for all connected components: **NSOURCE**=**NSINK** • **NSOURCE**= |VARIABLES1| • **NSINK**= |VARIABLES2|

Graph model Parts (A) and (B) of Figure 5.713 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The SAME_INTERVAL constraint holds since:

- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to |VARIABLES1|.
- The number of sinks of the final graph is equal to |VARIABLES2|.

Signature Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the $PRODUCT$ arc generator on the collections VARIABLES1 and VARIABLES2, we have that the maximum number of sources and sinks of the final graph is respectively equal to |VARIABLES1| and |VARIABLES2|. Therefore we can rewrite **NSOURCE** = |VARIABLES1| to **NSOURCE** $\geq$ |VARIABLES1| and simplify $\underline{\overline{\textbf{NSOURCE}}}$ to $\overline{\textbf{NSOURCE}}$. In a similar way, we can rewrite **NSINK** = |VARIABLES2| to **NSINK** $\geq$ |VARIABLES2| and simplify $\underline{\overline{\textbf{NSINK}}}$ to $\overline{\textbf{NSINK}}$.

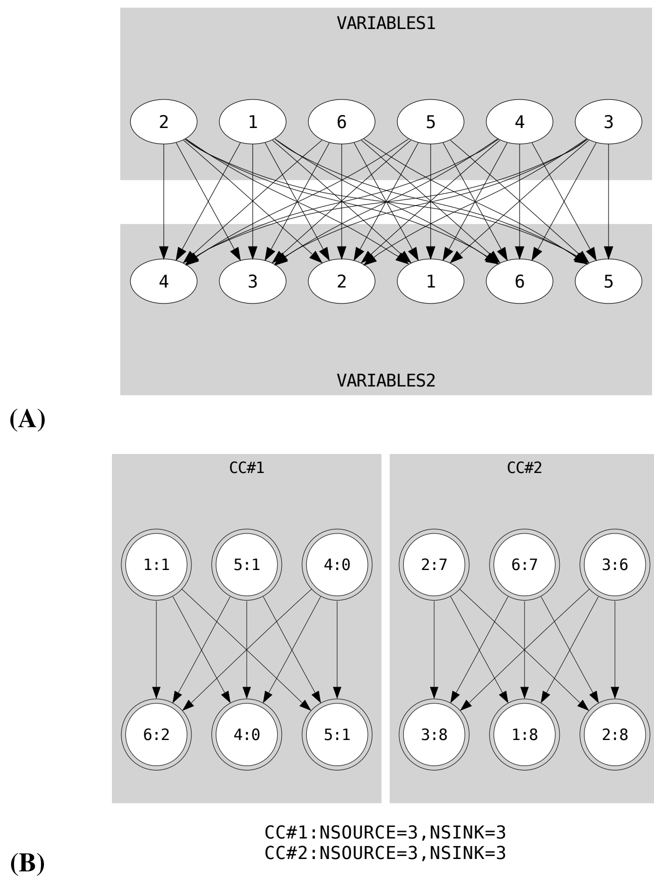


Figure 5.713: Initial and final graph of the SAME_INTERVAL constraint

5.345 SAME_MODULO

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin

Derived from SAME.

Constraint

SAME_MODULO(VARIABLES1, VARIABLES2, M)

Arguments

VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
M	:	int

Restrictions

|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
M > 0

Purpose

For each integer R in $[0, \mathtt{M}-1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have R as a rest when divided by M. For all R in $[0, \mathtt{M}-1]$ we have that $N1_R = N2_R$.

Example

$(\langle 1, 9, 1, 5, 2, 1\rangle, \langle 6, 4, 1, 1, 5, 5\rangle, 3)$

The values of the first collection $\langle 1, 9, 1, 5, 2, 1\rangle$ are respectively associated with the equivalence classes $1 \bmod 3 = 1$, $9 \bmod 3 = 0$, $1 \bmod 3 = 1$, $5 \bmod 3 = 2$, $2 \bmod 3 = 2$, $1 \bmod 3 = 1$. Therefore the equivalence classes 0, 1, and 2 are respectively used 1, 3, and 2 times. Similarly, the values of the second collection $\langle 6, 4, 1, 1, 5, 5\rangle$ are respectively associated with the equivalence classes $6 \bmod 3 = 0$, $4 \bmod 3 = 1$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $5 \bmod 3 = 2$, $5 \bmod 3 = 2$. Therefore the equivalence classes 0, 1, and 2 are respectively used 1, 3, and 2 times. Consequently the SAME_MODULO constraint holds. Figure 5.714 illustrates this correspondence.

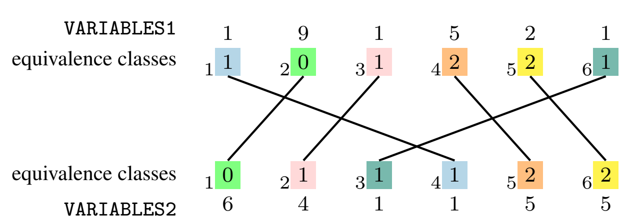


Figure 5.714: Illustration of the correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot

Typical

$|\texttt{VARIABLES1}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$\texttt{M} > 1$
$\texttt{M} < \texttt{maxval}(\texttt{VARIABLES1.var})$
$\texttt{M} < \texttt{maxval}(\texttt{VARIABLES2.var})$

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2) (M).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value u of VARIABLES.var can be replaced by any other value v such that v is congruent to u modulo M.

Arg. properties

Aggregate: VARIABLES1(union), VARIABLES2(union), M(id).

Used in

K_SAME_MODULO.

See also

implies: USED_BY_MODULO.

soft variant: SOFT_SAME_MODULO_VAR *(variable-based violation measure)*.

specialisation: SAME *(*variable mod constant *replaced by* variable*)*.

system of constraints: K_SAME_MODULO.

Keywords

characteristic of a constraint: sort based reformulation, modulo.

combinatorial object: permutation.

constraint arguments: constraint between two collections of variables.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var mod M = variables2.var mod M
Graph property(ies)	• for all connected components: **NSOURCE**=**NSINK** • **NSOURCE**= |VARIABLES1| • **NSINK**= |VARIABLES2|

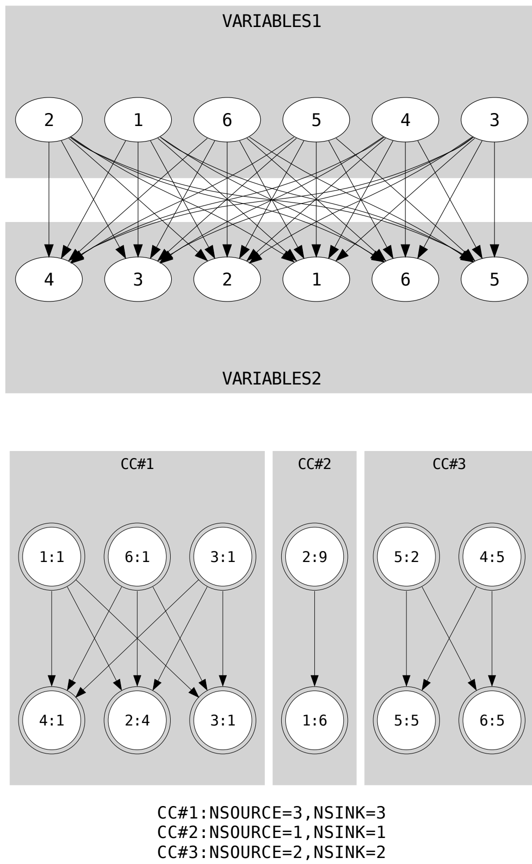
Graph model Parts (A) and (B) of Figure 5.715 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The SAME_MODULO constraint holds since:

- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to |VARIABLES1|.
- The number of sinks of the final graph is equal to |VARIABLES2|.

Signature Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the $PRODUCT$ arc generator on the collections VARIABLES1 and VARIABLES2, we have that the maximum number of sources and sinks of the final graph is respectively equal to |VARIABLES1| and |VARIABLES2|. Therefore we can rewrite $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$ to $\mathbf{NSOURCE} \geq |\texttt{VARIABLES1}|$ and simplify $\underline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.



(A)

(B)

Figure 5.715: Initial and final graph of the SAME_MODULO constraint

5.346 SAME_PARTITION

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from SAME.

Constraint SAME_PARTITION(VARIABLES1, VARIABLES2, PARTITIONS)

Type VALUES : collection(val−int)

Arguments
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions
|VALUES| ≥ 1
required(VALUES, val)
distinct(VALUES, val)
|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2

Purpose For each integer i in [1, |PARTITIONS|], let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that take their values in the i^{th} partition of the collection PARTITIONS. For all i in [1, |PARTITIONS|] we have $N1_i = N2_i$.

Example
$$\left(\begin{array}{l}\langle 1,2,6,3,1,2\rangle, \\ \langle 6,6,2,3,1,3\rangle, \\ \langle \mathtt{p}-\langle 1,3\rangle, \mathtt{p}-\langle 4\rangle, \mathtt{p}-\langle 2,6\rangle\rangle\end{array}\right)$$

The different values of the collection $\langle 1,2,6,3,1,2\rangle$ are respectively associated with the partitions p − $\langle 1,3\rangle$, p − $\langle 2,6\rangle$, p − $\langle 2,6\rangle$, p − $\langle 1,3\rangle$, p − $\langle 1,3\rangle$, and p − $\langle 2,6\rangle$. Therefore partitions p − $\langle 1,3\rangle$ and p − $\langle 2,6\rangle$ are respectively used 3 and 3 times. Similarly, the different values of the collection $\langle 6,6,2,3,1,3\rangle$ are respectively associated with the partitions p − $\langle 2,6\rangle$, p − $\langle 2,6\rangle$, p − $\langle 2,6\rangle$, p − $\langle 1,3\rangle$, p − $\langle 1,3\rangle$, and p − $\langle 1,3\rangle$. As before partitions p − $\langle 1,3\rangle$ and p − $\langle 2,6\rangle$ are respectively used 3 and 3 times. Consequently the SAME_PARTITION constraint holds. Figure 5.716 illustrates this correspondence.

Typical
|VARIABLES1| > 1
range(VARIABLES1.var) > 1
range(VARIABLES2.var) > 1
|VARIABLES1| > |PARTITIONS|
|VARIABLES2| > |PARTITIONS|

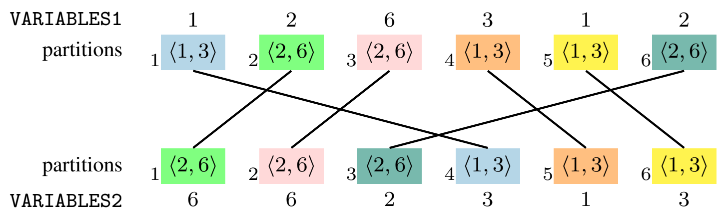


Figure 5.716: Illustration of the correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES1, VARIABLES2) (PARTITIONS).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Arg. properties

Aggregate: VARIABLES1(union), VARIABLES2(union), PARTITIONS(id).

Used in

K_SAME_PARTITION.

See also

implies: USED_BY_PARTITION.

soft variant: SOFT_SAME_PARTITION_VAR (*variable-based violation measure*).

specialisation: SAME (variable $\in$ partition *replaced by* variable).

system of constraints: K_SAME_PARTITION.

used in graph description: IN_SAME_PARTITION.

Keywords

characteristic of a constraint: sort based reformulation, partition.

combinatorial object: permutation.

constraint arguments: constraint between two collections of variables.

Arc input(s)	`VARIABLES1` `VARIABLES2`
Arc generator	$PRODUCT$ $\mapsto$ `collection`(`variables1`, `variables2`)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(`variables1.var`, `variables2.var`, `PARTITIONS`)
Graph property(ies)	• for all connected components: **NSOURCE**=**NSINK** • **NSOURCE**= |`VARIABLES1`| • **NSINK**= |`VARIABLES2`|

Graph model

Parts (A) and (B) of Figure 5.717 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. The SAME_PARTITION constraint holds since:

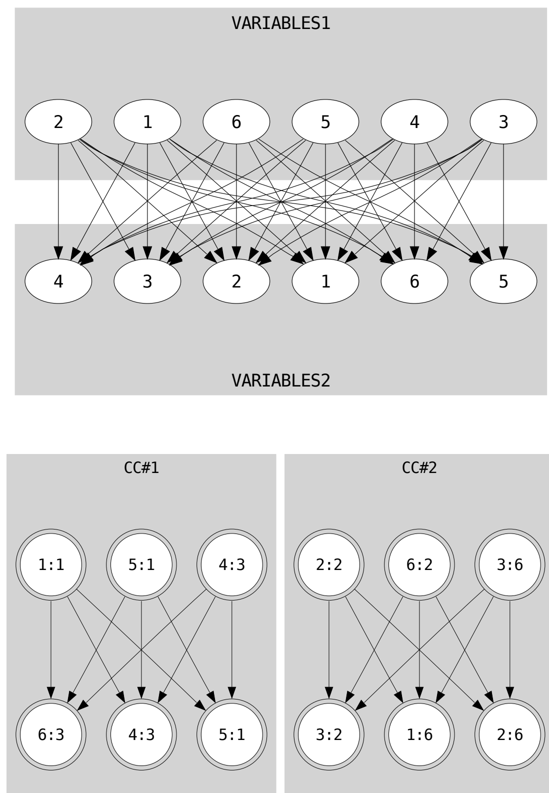
- Each connected component of the final graph has the same number of sources and of sinks.
- The number of sources of the final graph is equal to |`VARIABLES1`|.
- The number of sinks of the final graph is equal to |`VARIABLES2`|.

Signature

Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the $PRODUCT$ arc generator on the collections `VARIABLES1` and `VARIABLES2`, we have that the maximum number of sources and sinks of the final graph is respectively equal to |`VARIABLES1`| and |`VARIABLES2`|. Therefore we can rewrite $\mathbf{NSOURCE} = |\texttt{VARIABLES1}|$ to $\mathbf{NSOURCE} \geq |\texttt{VARIABLES1}|$ and simplify $\underline{\mathbf{NSOURCE}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite $\mathbf{NSINK} = |\texttt{VARIABLES2}|$ to $\mathbf{NSINK} \geq |\texttt{VARIABLES2}|$ and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.



(A)

CC#1:NSOURCE=3,NSINK=3
CC#2:NSOURCE=3,NSINK=3

(B)

Figure 5.717: Initial and final graph of the SAME_PARTITION constraint

5.347 SAME_REMAINDER

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin learning

Constraint SAME_REMAINDER(VARIABLES, Q, R)

Arguments

VARIABLES	:	collection(var−dvar)
Q	:	dvar
R	:	dvar

Restrictions

|VARIABLES| > 0
required(VARIABLES, [var])
VARIABLES.var ≥ 0
Q > 1
Q $\leq$maxval(VARIABLES.var)
R ≥ 0
R $<$ Q

Purpose All variables of the VARIABLES collection have the same remainder R when divided by Q.

Example

$(\langle 4, 6, 4, 8\rangle, 2, 0)$
$(\langle 4, 1, 4, 7\rangle, 3, 1)$

The first SAME_REMAINDER constraint holds since all variables are even, i.e. Q $= 2$ and R $= 0$. The second SAME_REMAINDER constraint holds since, when divided by Q $= 3$ all variables have a remainder R equal to 1.

Typical

|VARIABLES| > 2
Q < 10

Symmetry Items of VARIABLES are permutable.

Keywords **constraint type:** arithmetic constraint.

5.348 SAME_SIGN

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic.

Constraint SAME_SIGN(VAR1, VAR2)

Arguments

VAR1	:	dvar
VAR2	:	dvar

Restriction

Purpose Enforce the fact that the product of the first and second variables is greater than or equal to 0.

Example $(7, 1)$

The SAME_SIGN constraint holds since 7 and 1 have the same sign.

Typical

VAR1 $\neq 0$
VAR2 $\neq 0$

Symmetry Arguments are permutable w.r.t. permutation (VAR1, VAR2).

See also **comparison swapped:** OPPOSITE_SIGN.

implied by: DIVISIBLE_OR, EQ, SIGN_OF.

Keywords **constraint arguments:** binary constraint.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

5.349 SCALAR_PRODUCT

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic constraint.

Constraint SCALAR_PRODUCT(LINEARTERM, CTR, VAL)

Synonyms EQUATION, LINEAR, SUM_WEIGHT, WEIGHTEDSUM.

Arguments

LINEARTERM : collection(coeff−int, var−dvar)
CTR : atom
VAL : dvar

Restrictions

required(LINEARTERM, [coeff, var])
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose Constraint a linear term defined as the sum of products of coefficients and variables. More precisely, let S denote the sum of the product between a coefficient and its variable of the different items of the LINEARTERM collection. Enforce the following constraint to hold: S CTR VAL.

Example $(\langle \texttt{coeff}-1\ \texttt{var}-1, \texttt{coeff}-3\ \texttt{var}-1, \texttt{coeff}-1\ \texttt{var}-4\rangle, =, 8)$

The SCALAR_PRODUCT constraint holds since the condition $1\cdot 1+3\cdot 1+1\cdot 4=8$ is satisfied.

Typical

$|\texttt{LINEARTERM}| > 1$
range(LINEARTERM.coeff) > 1
range(LINEARTERM.var) > 1
CTR $\in [=, <, \geq, >, \leq]$

Symmetries

- Items of LINEARTERM are permutable.
- Attributes of LINEARTERM are permutable w.r.t. permutation (coeff, var) (*permutation not necessarily applied to all items*).

Arg. properties

- Contractible wrt. LINEARTERM when CTR $\in [<, \leq]$, minval(LINEARTERM.coeff) ≥ 0 and minval(LINEARTERM.var) ≥ 0.
- Extensible wrt. LINEARTERM when CTR $\in [\geq, >]$, minval(LINEARTERM.coeff) ≥ 0 and minval(LINEARTERM.var) ≥ 0.
- Aggregate: LINEARTERM(union), CTR(id), VAL(+).

Remark The SCALAR_PRODUCT constraint is called LINEAR in **Gecode** (http://www.gecode.org/). It is called SUM_WEIGHT in **JaCoP** (http://www.jacop.eu/). In the 2008 CSP solver competition the SCALAR_PRODUCT constraint was called WEIGHTEDSUM and required VAL to be fixed.

Algorithm Most filtering algorithms first merge multiple occurrences of identical variables in order to potentially make more deductions. When CTR corresponds to the *less than or equal to* constraint, a filtering algorithm achieving bound-consistency for the SCALAR_PRODUCT constraint with large numbers of variables is described in [214].

Systems EQUATION in **Choco**, LINEAR in **Gecode**, SUMWEIGHT in **JaCoP**, SCALAR_PRODUCT in **SICStus**.

See also **specialisation:** SUM_CTR *(arithmetic constraint where all coefficients are equal to* 1*)*.

Keywords **characteristic of a constraint:** sum.

constraint type: predefined constraint, arithmetic constraint.

filtering: duplicated variables.

5.350 SEQUENCE_FOLDING

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin J. Pearson

Constraint SEQUENCE_FOLDING(LETTERS)

Argument LETTERS : collection(index−int, next−dvar)

Restrictions
|LETTERS| ≥ 1
required(LETTERS, [index, next])
LETTERS.index ≥ 1
LETTERS.index ≤ |LETTERS|
increasing_seq(LETTERS, index)
LETTERS.next ≥ 1
LETTERS.next ≤ |LETTERS|

Purpose Express the fact that a sequence is folded in a way that no crossing occurs. A sequence is modelled by a collection of letters. For each letter l_1 of a sequence, we indicate the next letter l_2 located after l_1 that is directly in contact with l_1 (l_1 itself if such a letter does not exist).

Example

$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{next}-1,\\ \texttt{index}-2 & \texttt{next}-8,\\ \texttt{index}-3 & \texttt{next}-3,\\ \texttt{index}-4 & \texttt{next}-5,\\ \texttt{index}-5 & \texttt{next}-5,\\ \texttt{index}-6 & \texttt{next}-7,\\ \texttt{index}-7 & \texttt{next}-7,\\ \texttt{index}-8 & \texttt{next}-8,\\ \texttt{index}-9 & \texttt{next}-9\end{array}\right\rangle\right)$$

Figure 5.718 gives the folded sequence associated with the previous example. Each number represents the index of an item. The SEQUENCE_FOLDING constraint holds since no crossing occurs.

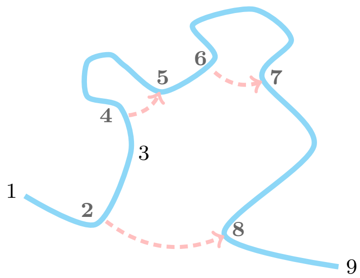
Typical
|LETTERS| > 2
range(LETTERS.next) > 1

Usage Motivated by RNA folding [178].

See also **implies (items to collection):** LEX_ALLDIFFERENT, LEX_CHAIN_LESS.

Keywords **application area:** bioinformatics.

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.



LETTERS

index − 1	next − 1
index − 2	next − 8
index − 3	next − 3
index − 4	next − 5
index − 5	next − 5
index − 6	next − 7
index − 7	next − 7
index − 8	next − 8
index − 9	next − 9

Figure 5.718: Folded sequence (in blue) of the **Example** slot: links from a letter to a distinct letter are represented by a dashed arc, while self-loops are not drawn

combinatorial object: sequence.

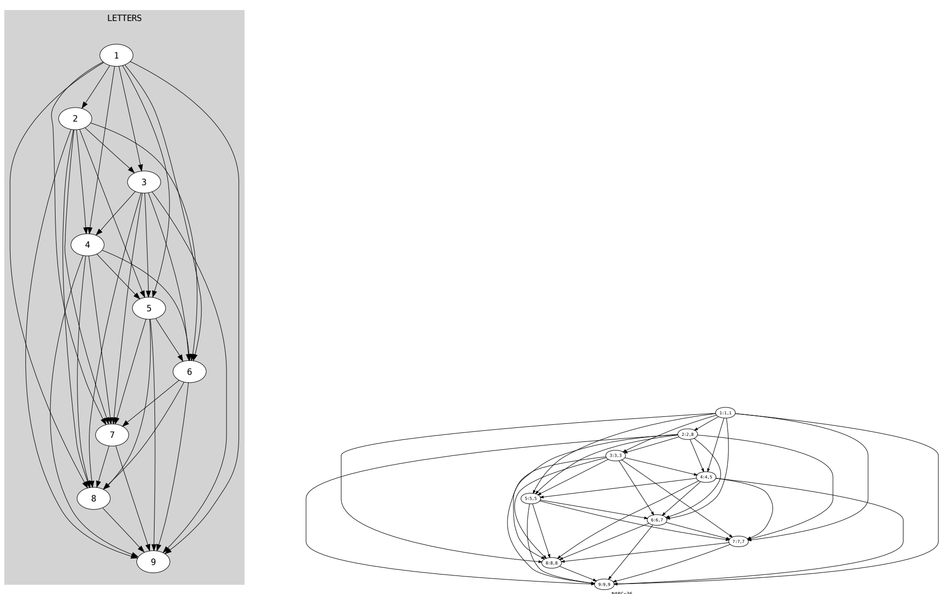
constraint type: decomposition.

geometry: geometrical constraint.

Arc input(s)	LETTERS
Arc generator	$SELF \mapsto$ collection(letters)
Arc arity	1
Arc constraint(s)	letters.next $\geq$ letters.index
Graph property(ies)	**NARC** $= \vert$LETTERS$\vert$

Arc input(s)	LETTERS
Arc generator	$CLIQUE(<) \mapsto$ collection(letters1, letters2)
Arc arity	2
Arc constraint(s)	$\bigvee\left(\begin{array}{l}\texttt{letters2.index} \geq \texttt{letters1.next},\\ \texttt{letters2.next} \leq \texttt{letters1.next}\end{array}\right)$
Graph property(ies)	**NARC** $= \vert$LETTERS$\vert * (\vert$LETTERS$\vert - 1)/2$

Graph model Parts (A) and (B) of Figure 5.719 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.719: Initial and final graph of the SEQUENCE_FOLDING constraint

Signature Consider the first graph constraint. Since we use the $SELF$ arc generator on the LETTERS collection the maximum number of arcs of the final graph is equal to $\vert$LETTERS$\vert$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{LETTERS}|$ to $\mathbf{NARC} \geq |\texttt{LETTERS}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Consider now the second graph constraint. Since we use the $CLIQUE(<)$ arc generator on the LETTERS collection the maximum number of arcs of the final graph is equal to $|\texttt{LETTERS}| \cdot (|\texttt{LETTERS}| - 1)/2$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{LETTERS}| \cdot (|\texttt{LETTERS}| - 1)/2$ to $\mathbf{NARC} \geq |\texttt{LETTERS}| \cdot (|\texttt{LETTERS}| - 1)/2$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 5.720 depicts the automaton associated with the SEQUENCE_FOLDING constraint. Consider the i^{th} and the j^{th} $(i < j)$ items of the collection $\texttt{LETTERS}$. Let $\texttt{INDEX}_i$ and $\texttt{NEXT}_i$ respectively denote the `index` and the `next` attributes of the i^{th} item of the collection $\texttt{LETTERS}$. Similarly, let $\texttt{INDEX}_j$ and $\texttt{NEXT}_j$ respectively denote the `index` and the `next` attributes of the j^{th} item of the collection $\texttt{LETTERS}$. To each quadruple $(\texttt{INDEX}_i, \texttt{NEXT}_i, \texttt{INDEX}_j, \texttt{NEXT}_j)$ corresponds a signature variable $S_{i,j}$, which takes its value in $\{0, 1, 2\}$, as well as the following signature constraint:

$(\texttt{INDEX}_i \leq \texttt{NEXT}_i) \wedge (\texttt{INDEX}_j \leq \texttt{NEXT}_j) \wedge (\texttt{NEXT}_i \leq \texttt{NEXT}_j) \Leftrightarrow S_{i,j} = 0 \wedge$

$(\texttt{INDEX}_i \leq \texttt{NEXT}_i) \wedge (\texttt{INDEX}_j \leq \texttt{NEXT}_j) \wedge (\texttt{NEXT}_i > \texttt{INDEX}_j) \wedge (\texttt{NEXT}_j \leq \texttt{NEXT}_i) \Leftrightarrow S_{i,j} = 1$.

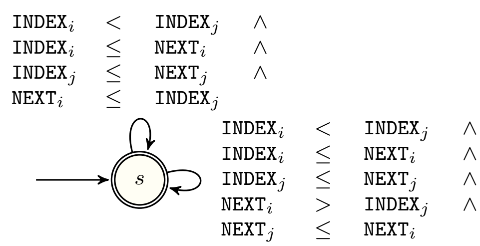


Figure 5.720: Automaton of the SEQUENCE_FOLDING constraint

5.351 SET_VALUE_PRECEDE

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin [269]

Constraint SET_VALUE_PRECEDE(S, T, VARIABLES)

Arguments

S : int
T : int
VARIABLES : collection(var−svar)

Restrictions

$\texttt{S} \neq \texttt{T}$
required(VARIABLES, var)

Purpose If there exists a set variable v_1 of VARIABLES such that S does not belong to v_1 and T does, then there also exists a set variable v_2 preceding v_1 such that S belongs to v_2 and T does not.

Example

$(2, 1, \langle \texttt{var} - \{0, 2\}, \texttt{var} - \{0, 1\}, \texttt{var} - \emptyset, \texttt{var} - \{1\}\rangle)$
$(0, 1, \langle \texttt{var} - \{0, 2\}, \texttt{var} - \{0, 1\}, \texttt{var} - \emptyset, \texttt{var} - \{1\}\rangle)$
$(0, 2, \langle \texttt{var} - \{0, 2\}, \texttt{var} - \{0, 1\}, \texttt{var} - \emptyset, \texttt{var} - \{1\}\rangle)$
$(0, 4, \langle \texttt{var} - \{0, 2\}, \texttt{var} - \{0, 1\}, \texttt{var} - \emptyset, \texttt{var} - \{1\}\rangle)$

The following examples are taken from [268, page 58]:

- The SET_VALUE_PRECEDE$(2, 1, \langle\{0, 2\}, \{0, 1\}, \{\}, \{1\}\rangle)$ constraint holds since the first occurrence of value 2 precedes the first occurrence of value 1 (i.e., the set $\{0, 2\}$ occurs before the set $\{0, 1\}$).
- The SET_VALUE_PRECEDE$(0, 1, \langle\{0, 2\}, \{0, 1\}, \{\}, \{1\}\rangle)$ constraint holds since the first occurrence of value 0 precedes the first occurrence of value 1 (i.e., the set $\{0, 2\}$ occurs before the set $\{0, 1\}$).
- The SET_VALUE_PRECEDE$(0, 2, \langle\{0, 2\}, \{0, 1\}, \{\}, \{1\}\rangle)$ constraint holds since "there is no set in $\langle\{0, 2\}, \{0, 1\}, \{\}, \{1\}\rangle$ that contains 2 but not 0".
- The SET_VALUE_PRECEDE$(0, 4, \langle\{0, 2\}, \{0, 1\}, \{\}, \{1\}\rangle)$ constraint holds since no set in $\langle\{0, 2\}, \{0, 1\}, \{\}, \{1\}\rangle$ contains value 4.

Typical

$\texttt{S} < \texttt{T}$
$|\texttt{VARIABLES}| > 1$

Arg. properties Suffix-contractible wrt. VARIABLES.

Algorithm A filtering algorithm for maintaining value precedence on a sequence of set variables is presented in [269]. Its complexity is linear to the number of variables of the collection VARIABLES.

Systems PRECEDE in **Gecode**.

See also **specialisation:** INT_VALUE_PRECEDE *(*`sequence` *of* `set variables` *replaced by* `sequence` *of* `domain variables`*)*.

Keywords **constraint arguments:** constraint involving set variables.

constraint type: order constraint.

symmetry: symmetry, indistinguishable values, value precedence.

5.352 SHIFT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint $\text{SHIFT}(\texttt{MIN\_BREAK}, \texttt{MAX\_RANGE}, \texttt{TASKS})$

Arguments

$\texttt{MIN\_BREAK} : \texttt{int}$
$\texttt{MAX\_RANGE} : \texttt{int}$
$\texttt{TASKS} : \texttt{collection}(\texttt{origin}-\texttt{dvar}, \texttt{end}-\texttt{dvar})$

Restrictions

$\texttt{MIN\_BREAK} > 0$
$\texttt{MAX\_RANGE} > 0$
$\texttt{required}(\texttt{TASKS}, [\texttt{origin}, \texttt{end}])$
$\texttt{TASKS.origin} < \texttt{TASKS.end}$

Purpose

The difference between the end of the last task of a *shift* and the origin of the first task of a *shift* should not exceed the quantity MAX_RANGE. Two tasks t_1 and t_2 belong to the *same shift* if at least one of the following conditions is true:

- Task t_2 starts after the end of task t_1 at a distance that is less than or equal to the quantity MIN_BREAK,
- Task t_1 starts after the end of task t_2 at a distance that is less than or equal to the quantity MIN_BREAK.
- Task t_1 overlaps task t_2.

Example

$$\left(6, 8, \left\langle \begin{array}{ll} \texttt{origin}-17 & \texttt{end}-20, \\ \texttt{origin}-7 & \texttt{end}-10, \\ \texttt{origin}-2 & \texttt{end}-4, \\ \texttt{origin}-21 & \texttt{end}-22, \\ \texttt{origin}-5 & \texttt{end}-6 \end{array} \right\rangle \right)$$

Figure 5.721 represents the different tasks of the example. Each task is drawn as a rectangle with its corresponding id attribute in the middle. We indicate the distance between two consecutive tasks of a same shift and note that it is less than or equal to $\texttt{MIN\_BREAK} = 6$. Since each shift has a range that is less than or equal to $\texttt{MAX\_RANGE} = 8$, the SHIFT constraint holds (the *range* of a shift is the difference between the end of the last task of the shift and the origin of the first task of the shift).

All solutions

Figure 5.722 gives all solutions to the following non ground instance of the SHIFT constraint: $\texttt{MIN\_BREAK} = 2$, $\texttt{MAX\_RANGE} = 5$, $\texttt{O}_1 \in [1, 2]$, $\texttt{E}_1 \in [1, 4]$, $\texttt{O}_2 \in [1, 4]$, $\texttt{E}_2 \in [1, 2]$, $\texttt{O}_3 \in [4, 7]$, $\texttt{E}_3 \in [4, 5]$, $\texttt{O}_4 \in [7, 9]$, $\texttt{E}_4 \in [0, 9]$, $\text{SHIFT}(\texttt{MIN\_BREAK}, \texttt{MAX\_RANGE}, \langle \texttt{O}_1\ \texttt{E}_1, \texttt{O}_2\ \texttt{E}_2, \texttt{O}_3\ \texttt{E}_3, \texttt{O}_4\ \texttt{E}_4 \rangle)$.

TASKS

①	`origin` − 17	`end` − 20	④	`origin` − 21	`end` − 22
②	`origin` − 7	`end` − 10	⑤	`origin` − 5	`end` − 6
③	`origin` − 2	`end` − 4			

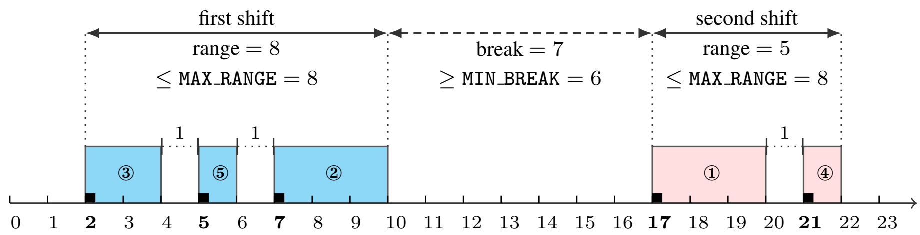


Figure 5.721: The two shifts of the **Example** slot

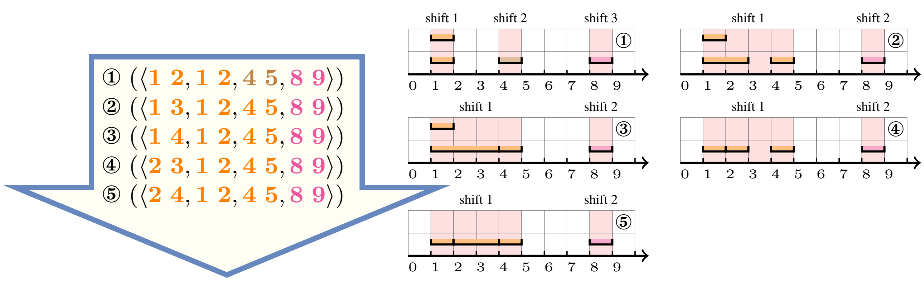


Figure 5.722: All solutions corresponding to the non ground example of the SHIFT constraint of the **All solutions** slot, where tasks belonging to the same shift are coloured with the same colour

Typical

$\texttt{MIN\_BREAK} > 1$
$\texttt{MAX\_RANGE} > 1$
$\texttt{MIN\_BREAK} < \texttt{MAX\_RANGE}$
$|\texttt{TASKS}| > 2$

Symmetries

- Items of TASKS are permutable.
- One and the same constant can be added to the `origin` attribute of all items of TASKS.

Usage

The shift constraint can be used in machine scheduling problems where one has to shut down a machine for maintenance purpose after a given maximum utilisation of that machine. In this case the MAX_RANGE parameter indicates the maximum possible utilisation of the machine before maintenance, while the MIN_BREAK parameter gives the minimum time needed for maintenance.

The shift constraint can also be used for timetabling problems where the rest period of a person can move in time. In this case MAX_RANGE indicates the maximum possible working time for a person, while `MIN_BREAK` specifies the minimum length of the break that follows a working time period.

See also **common keyword:** SLIDING_TIME_WINDOW *(temporal constraint)*.

used in graph description: RANGE_CTR.

Keywords **constraint type:** scheduling constraint, timetabling constraint, temporal constraint.

Arc input(s)	TASKS
Arc generator	$SELF \mapsto$ collection(tasks)
Arc arity	1
Arc constraint(s)	• tasks.end $\geq$ tasks.origin • tasks.end $-$ tasks.origin $\leq$ MAX_RANGE
Graph property(ies)	**NARC** $=$ |TASKS|

Arc input(s)	TASKS
Arc generator	$CLIQUE \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	$\bigvee\left(\begin{array}{l}\bigwedge\left(\begin{array}{l}\texttt{tasks2.origin} \geq \texttt{tasks1.end},\\ \texttt{tasks2.origin} - \texttt{tasks1.end} \leq \texttt{MIN\_BREAK}\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{tasks1.origin} \geq \texttt{tasks2.end},\\ \texttt{tasks1.origin} - \texttt{tasks2.end} \leq \texttt{MIN\_BREAK}\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{tasks2.origin} < \texttt{tasks1.end},\\ \texttt{tasks1.origin} < \texttt{tasks2.end}\end{array}\right)\end{array}\right)$
Sets	$\mathsf{CC} \mapsto \left[\texttt{variables} - \texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ \left[\begin{array}{l}\texttt{item}(\texttt{var} - \texttt{TASKS.origin}),\\ \texttt{item}(\texttt{var} - \texttt{TASKS.end})\end{array}\right]\end{array}\right)\right]$
Constraint(s) on sets	RANGE_CTR(variables, $\leq$, MAX_RANGE)

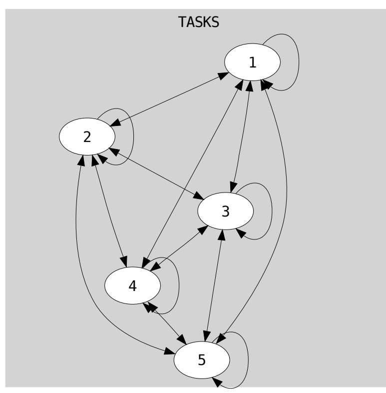
Graph model The first graph constraint forces the following two constraints between the attributes of each task:

- The end of a task should not be situated before its start,
- The duration of a task should not be greater than the MAX_RANGE parameter.

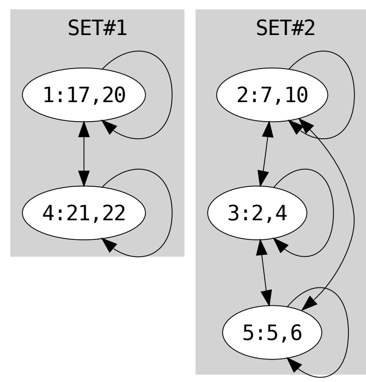
The second graph constraint decomposes the final graph in connected components where each component corresponds to a given shift. Finally, the **Constraint(s) on sets** slot restricts the stretch of each shift.

Parts (A) and (B) of Figure 5.723 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the set generator CC we show the two connected components of the final graph. They respectively correspond to the two shifts that are displayed in Figure 5.721.

Signature Consider the first graph constraint. Since we use the $SELF$ arc generator on the TASKS collection the maximum number of arcs of the final graph is equal to |TASKS|. Therefore we can rewrite the graph property $\mathbf{NARC} = |\texttt{TASKS}|$ to $\mathbf{NARC} \geq |\texttt{TASKS}|$ and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.



(A)



(B)

Figure 5.723: Initial and final graph of the SHIFT constraint

5.353 SIGN_OF

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic.

Constraint SIGN_OF(S, X)

Usual name SIGN

Arguments

S : dvar
X : dvar

Restrictions

$S \geq -1$
$S \leq 1$

Purpose

According to the value of the first variable S, restrict the sign of the second variable X:

- When $S = -1$, X should be negative (i.e., $X < 0$).
- When $S = 0$, X is also equal to 0.
- When $S = +1$, X should be positive (i.e., $X > 0$).

Example

$(-1, -8)$
$(0, 0)$
$(1, 8)$

- The first SIGN_OF constraint holds since $S = -1$ and $X = -8$ is negative.
- The second SIGN_OF constraint holds since $S = 0$ and $X = 0$ is neither negative, neither positive.
- The second SIGN_OF constraint holds since $S = +1$ and $X = 8$ is positive.

Typical

$S \neq 0$
$X \neq 0$

Arg. properties Functional dependency: S determined by X.

See also **implies:** SAME_SIGN, ZERO_OR_NOT_ZERO.

Keywords

constraint arguments: binary constraint, pure functional dependency.

constraint type: predefined constraint, arithmetic constraint.

filtering: arc-consistency.

modelling: functional dependency.

5.354 SIZE_MAX_SEQ_ALLDIFFERENT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint SIZE_MAX_SEQ_ALLDIFFERENT(SIZE, VARIABLES)

Synonyms SIZE_MAXIMAL_SEQUENCE_ALLDIFF, SIZE_MAXIMAL_SEQUENCE_ALLDISTINCT, SIZE_MAXIMAL_SEQUENCE_ALLDIFFERENT.

Arguments

SIZE	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

SIZE ≥ 0
SIZE $\leq$ |VARIABLES|
required(VARIABLES, var)

Purpose SIZE is the size of the maximal sequence (among all possible sequences of consecutive variables of the collection VARIABLES) for which the ALLDIFFERENT constraint holds.

Example

$(4, \langle 2, 2, 4, 5, 2, 7, 4\rangle)$
$(1, \langle 2, 2, 2, 2, 2, 2, 2\rangle)$
$(2, \langle 2, 2, 4, 4, 4, 7, 4\rangle)$
$(7, \langle 2, 0, 4, 6, 5, 7, 3\rangle)$

The first SIZE_MAX_SEQ_ALLDIFFERENT constraint holds since the constraint ALLDIFFERENT($\langle$var − 4, var − 5, var − 2, var − 7$\rangle$) holds and since the following three constraints do not hold:

- ALLDIFFERENT($\langle$var − 2, var − 2, var − 4, var − 5, var − 2$\rangle$),
- ALLDIFFERENT($\langle$var − 2, var − 4, var − 5, var − 2, var − 7$\rangle$),
- ALLDIFFERENT($\langle$var − 4, var − 5, var − 2, var − 7, var − 4$\rangle$).

Typical

SIZE > 2
SIZE $<$ |VARIABLES|
range(VARIABLES.var) > 1

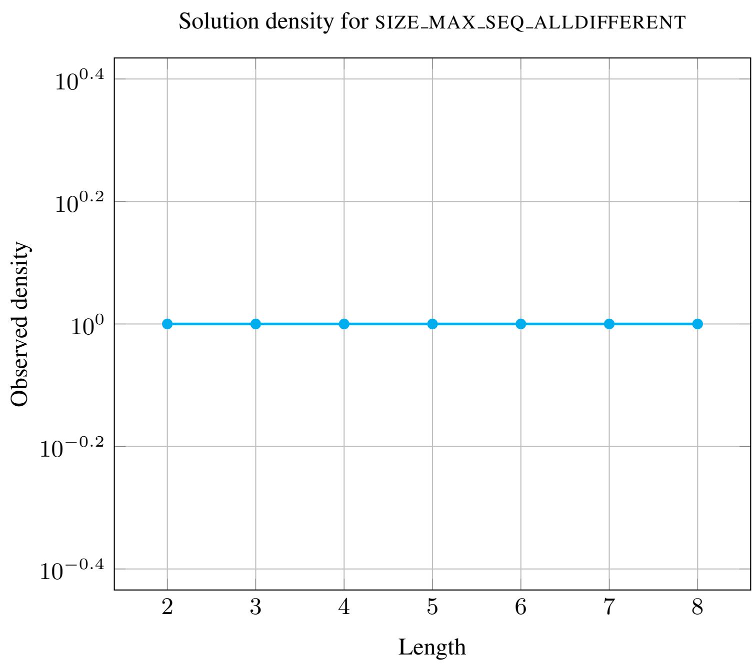
Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

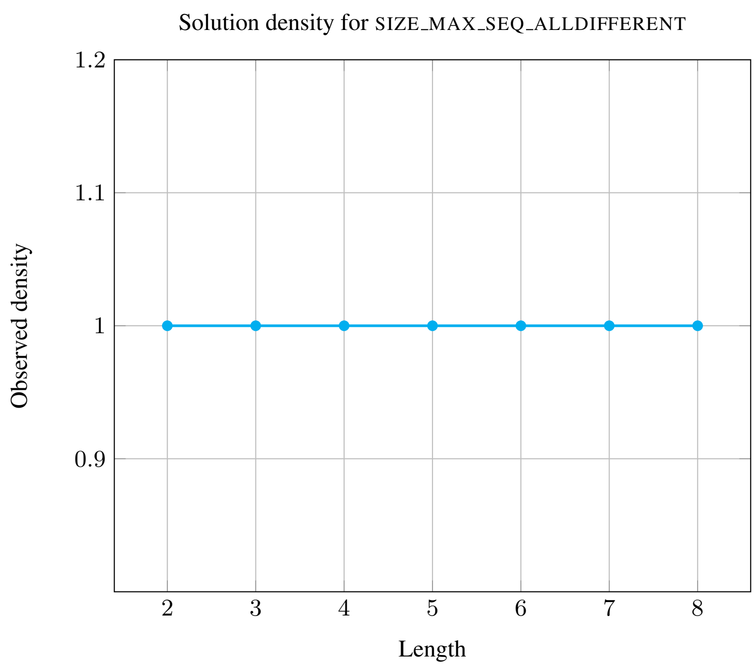
Arg. properties Functional dependency: SIZE determined by VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for SIZE_MAX_SEQ_ALLDIFFERENT: domains $0..n$

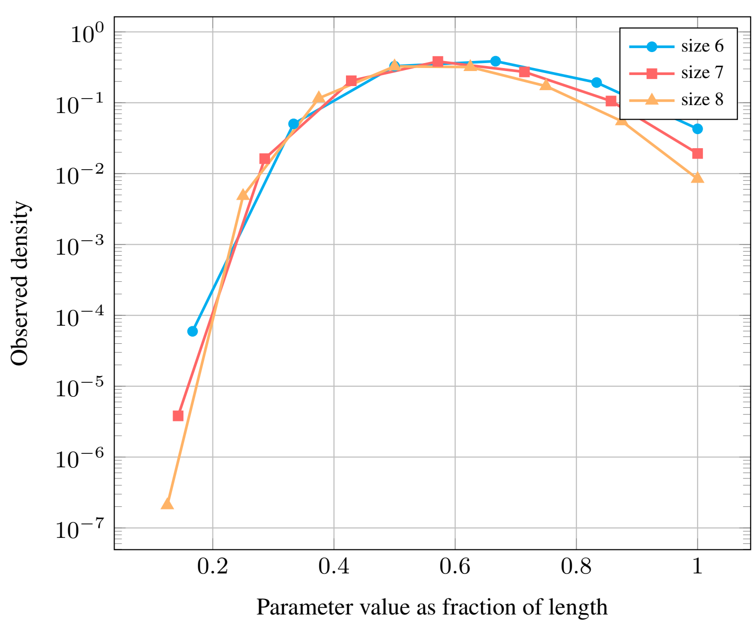


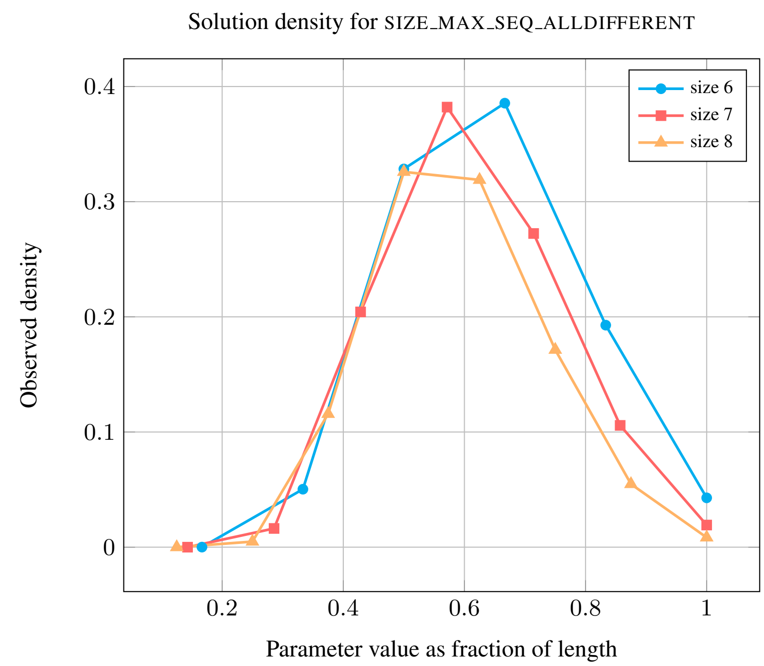


Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	3	4	5	6	7	8	9
	2	6	36	200	1050	5922	34104	208224
	3	-	24	300	3480	38640	428400	4981032
	4	-	-	120	2520	45360	801360	14028336
	5	-	-	-	720	22680	571200	13728960
	6	-	-	-	-	5040	221760	7378560
	7	-	-	-	-	-	40320	2358720
	8	-	-	-	-	-	-	362880

Solution count for SIZE_MAX_SEQ_ALLDIFFERENT: domains $0..n$

Solution density for SIZE_MAX_SEQ_ALLDIFFERENT





See also **common keyword:** ALLDIFFERENT, OPEN_ALLDIFFERENT, SIZE_MAX_STARTING_SEQ_ALLDIFFERENT *(all different,disequality)*.

implies: ATLEAST_NVALUE.

Keywords **characteristic of a constraint:** all different, disequality, hypergraph.

combinatorial object: sequence.

constraint arguments: pure functional dependency.

constraint type: sliding sequence constraint, conditional constraint.

modelling: functional dependency.

Arc input(s)	VARIABLES
Arc generator	$PATH\_N \mapsto$ collection
Arc arity	$*$
Arc constraint(s)	ALLDIFFERENT(collection)
Graph property(ies)	**NARC** $=$ SIZE

Graph model Note that this is an example of global constraint where the arc constraints do not have the same arity. However they correspond to the same type of constraint.

5.355 SIZE_MAX_STARTING_SEQ_ALLDIFFERENT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Inspired by SIZE_MAX_SEQ_ALLDIFFERENT.

Constraint SIZE_MAX_STARTING_SEQ_ALLDIFFERENT(SIZE, VARIABLES)

Synonyms SIZE_MAXIMAL_STARTING_SEQUENCE_ALLDIFF, SIZE_MAXIMAL_STARTING_SEQUENCE_ALLDISTINCT, SIZE_MAXIMAL_STARTING_SEQUENCE_ALLDIFFERENT.

Arguments

SIZE : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\texttt{SIZE} \geq 0$
$\texttt{SIZE} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose SIZE is the size of the maximal sequence (among all sequences of consecutive variables of the collection VARIABLES starting at position one) for which the ALLDIFFERENT constraint holds.

Example

$(4, \langle 9, 2, 4, 5, 2, 7, 4\rangle)$
$(7, \langle 9, 2, 4, 5, 1, 7, 8\rangle)$
$(6, \langle 9, 2, 4, 5, 1, 7, 9\rangle)$

The first SIZE_MAX_STARTING_SEQ_ALLDIFFERENT constraint holds since the constraint ALLDIFFERENT($\langle$var − 9, var − 2, var − 4, var − 5$\rangle$) holds and since ALLDIFFERENT($\langle$var − 9, var − 2, var − 4, var − 5, var − 2$\rangle$) does not hold.

Typical

$\texttt{SIZE} > 2$
$\texttt{SIZE} < |\texttt{VARIABLES}|$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

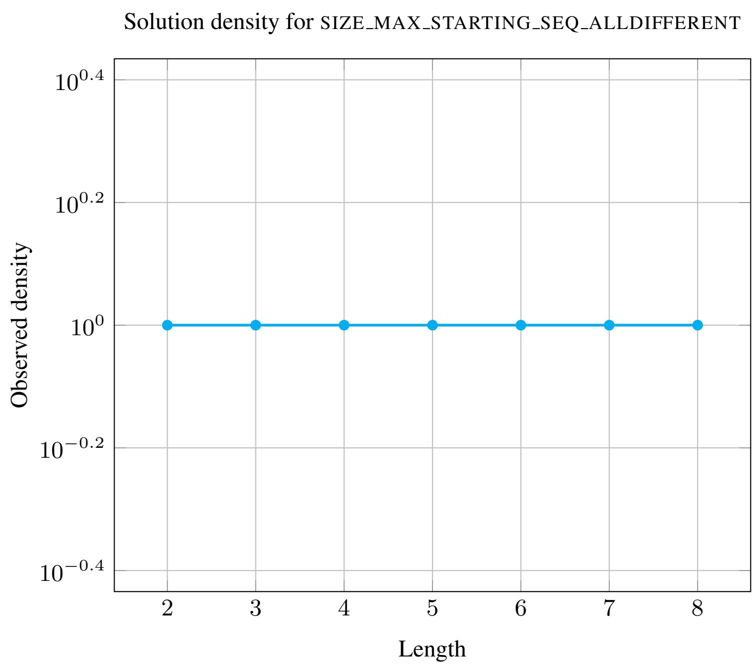
Arg. properties Functional dependency: SIZE determined by VARIABLES.

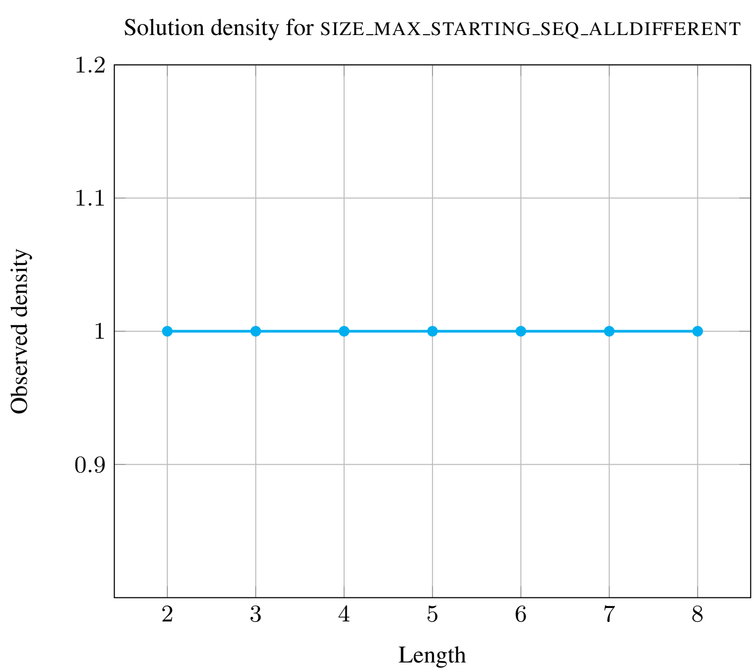
Remark A conditional constraint [296] with the specific structure that one can relax the constraints on the last variables of the collection VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

Number of solutions for SIZE_MAX_STARTING_SEQ_ALLDIFFERENT: domains $0..n$

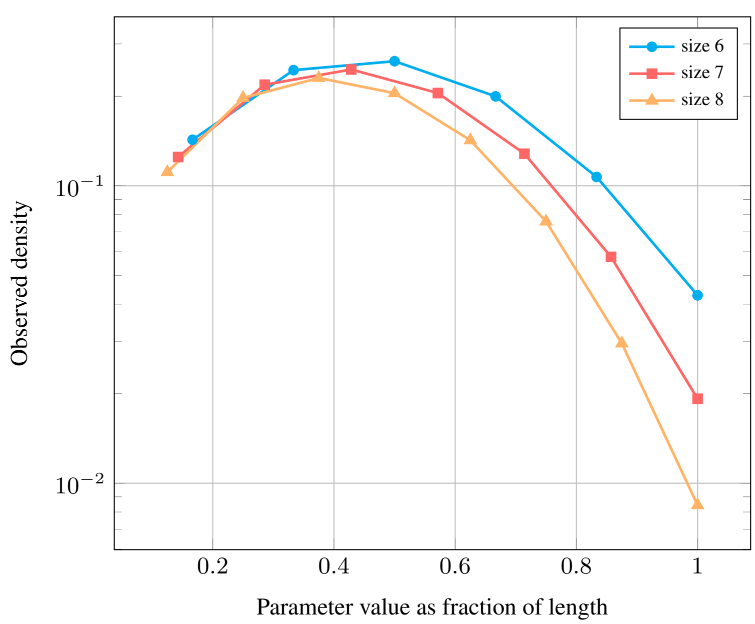

Solution density for SIZE_MAX_STARTING_SEQ_ALLDIFFERENT
Observed density
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
2 3 4 5 6 7 8
Length

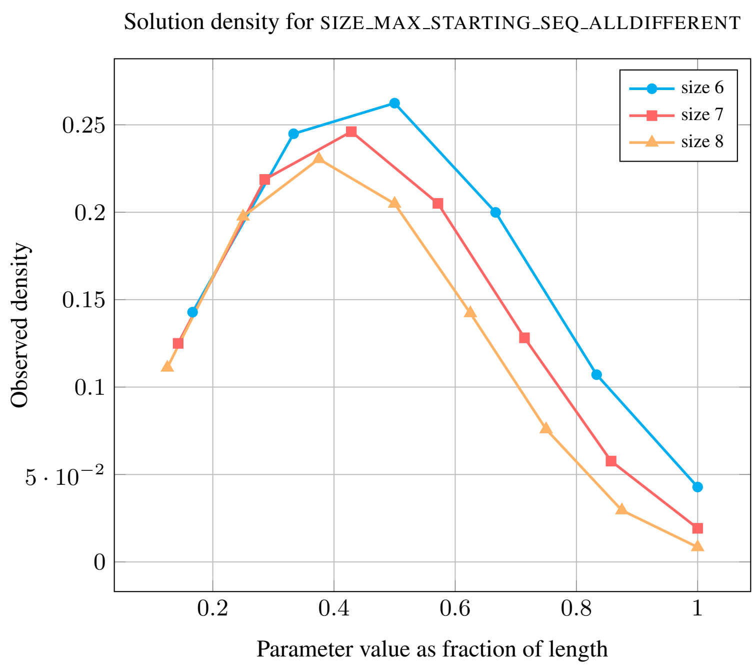

Solution density for SIZE_MAX_STARTING_SEQ_ALLDIFFERENT
Observed density
1.2
1.1
1
0.9
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	1	3	16	125	1296	16807	262144	4782969
	2	6	24	200	2160	28812	458752	8503056
	3	-	24	180	2160	30870	516096	9920232
	4	-	-	120	1440	23520	430080	8817984
	5	-	-	-	720	12600	268800	6123600
	6	-	-	-	-	5040	120960	3265920
	7	-	-	-	-	-	40320	1270080
	8	-	-	-	-	-	-	362880

Solution count for SIZE_MAX_STARTING_SEQ_ALLDIFFERENT: domains $0..n$

Solution density for SIZE_MAX_STARTING_SEQ_ALLDIFFERENT





See also **common keyword:** ALLDIFFERENT, OPEN_ALLDIFFERENT, SIZE_MAX_SEQ_ALLDIFFERENT *(all different,disequality)*.

implies: ATLEAST_NVALUE.

Keywords **characteristic of a constraint:** all different, disequality, hypergraph.

combinatorial object: sequence.

constraint arguments: pure functional dependency.

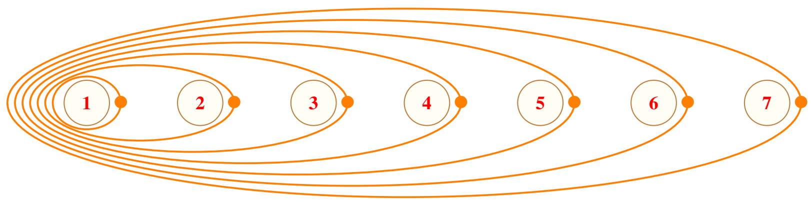
constraint type: sliding sequence constraint, open constraint, conditional constraint.

modelling: functional dependency.

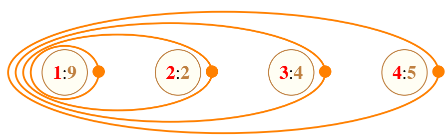
Arc input(s)	VARIABLES
Arc generator	*PATH_1* ↦ collection
Arc arity	$*$
Arc constraint(s)	ALLDIFFERENT(collection)
Graph property(ies)	**NARC** = SIZE

Graph model Note that this is an example where the arc constraints do not have the same arity. However they correspond to the same constraint.

Parts (A) and (B) of Figure 5.724 respectively show the initial and final graph associated with the first example of the **Example** slot.



(A)



(B)

Figure 5.724: (A) Initial and (B) final graph of the SIZE_MAX_STARTING_SEQ_ALLDIFFERENT$(4, \langle 9, 2, 4, 5, 2, 7, 4\rangle)$ constraint of the first example of the **Example** slot where each ellipse represents an hyperedge corresponding to an ALLDIFFERENT constraint (e.g., the fourth ellipse represents the constraint ALLDIFFERENT$\langle 9, 2, 4, 5\rangle$)

5.356 SLIDING_CARD_SKIP0

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint SLIDING_CARD_SKIP0(ATLEAST, ATMOST, VARIABLES, VALUES)

Arguments

ATLEAST : int
ATMOST : int
VARIABLES : collection(var−dvar)
VALUES : collection(val−int)

Restrictions

ATLEAST ≥ 0
ATLEAST $\leq$ |VARIABLES|
ATMOST ≥ 0
ATMOST $\leq$ |VARIABLES|
ATMOST $\geq$ ATLEAST
required(VARIABLES, var)
required(VALUES, val)
distinct(VALUES, val)
VALUES.val $\neq 0$

Purpose

Let n be the total number of variables of the collection VARIABLES. A *maximum non-zero set of consecutive variables* $X_i..X_j (1 \leq i \leq j \leq n)$ is defined in the following way:

- All variables $X_i, \ldots, X_j$ take a non-zero value,
- $i = 1$ or X_{i-1} is equal to 0,
- $j = n$ or X_{j+1} is equal to 0.

Enforces that each maximum non-zero set of consecutive variables of the collection VARIABLES contains at least ATLEAST and at most ATMOST values from the collection of values VALUES.

Example

$(2, 3, \langle 0, 7, 2, 9, 0, 0, 9, 4, 9 \rangle, \langle 7, 9 \rangle)$

The SLIDING_CARD_SKIP0 constraint holds since the two maximum non-zero set of consecutive values 7 2 9 and 9 4 9 of its third argument $\langle 0, 7, 2, 9, 0, 0, 9, 4, 9 \rangle$ take both 2 ($2 \in$ [ATLEAST, ATMOST] $= [2, 3]$) values within the set of values $\langle 7, 9 \rangle$.

Typical

|VARIABLES| > 1
|VALUES| > 0
|VARIABLES| $>$ |VALUES|
ATLEAST(1, VARIABLES, 0)
ATLEAST $> 0 \vee$ ATMOST $<$ |VARIABLES|

Symmetries

- ATLEAST can be decreased to any value ≥ 0.
- ATMOST can be increased to any value $\leq$ |VARIABLES|.
- Items of VARIABLES can be reversed.
- An occurrence of a value different from 0 of VARIABLES.var that belongs to VALUES.val (resp. does not belong to VALUES.val) can be replaced by any other value different from 0 in VALUES.val (resp. not in VALUES.val).

Usage

This constraint is useful in timetabling problems where the variables are interpreted as the type of job that a person does on consecutive days. Value 0 represents a rest day and one imposes a cardinality constraint on periods that are located between rest periods.

Remark

One cannot initially state a GLOBAL_CARDINALITY constraint since the rest days are not yet allocated. One can also not use an AMONG_SEQ constraint since it does not hold for the sequences of consecutive variables that contains at least one rest day.

See also

related: AMONG *(counting constraint on the full sequence)*, GLOBAL_CARDINALITY *(counting constraint for different values on the full sequence)*.

specialisation: AMONG_LOW_UP *(maximal sequences replaced by the full sequence)*.

Keywords

characteristic of a constraint: automaton, automaton with counters.

combinatorial object: sequence.

constraint network structure: alpha-acyclic constraint network(2).

constraint type: timetabling constraint, sliding sequence constraint.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2) $LOOP \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• variables1.var $\neq 0$ • variables2.var $\neq 0$
Sets	CC $\mapsto$ [variables]
Constraint(s) on sets	AMONG_LOW_UP(ATLEAST, ATMOST, variables, VALUES)

Graph model Note that the arc constraint will produce the different sequences of consecutive variables that do not contain any 0. The CC set generator produces all the connected components of the final graph.

Parts (A) and (B) of Figure 5.725 respectively show the initial and final graph associated with the **Example** slot. Since we use the set generator CC we show the two connected components of the final graph. Since these two connected components both contains between 2 and 3 variables that take their values in $\{7, 9\}$ the SLIDING_CARD_SKIP0 constraint holds.

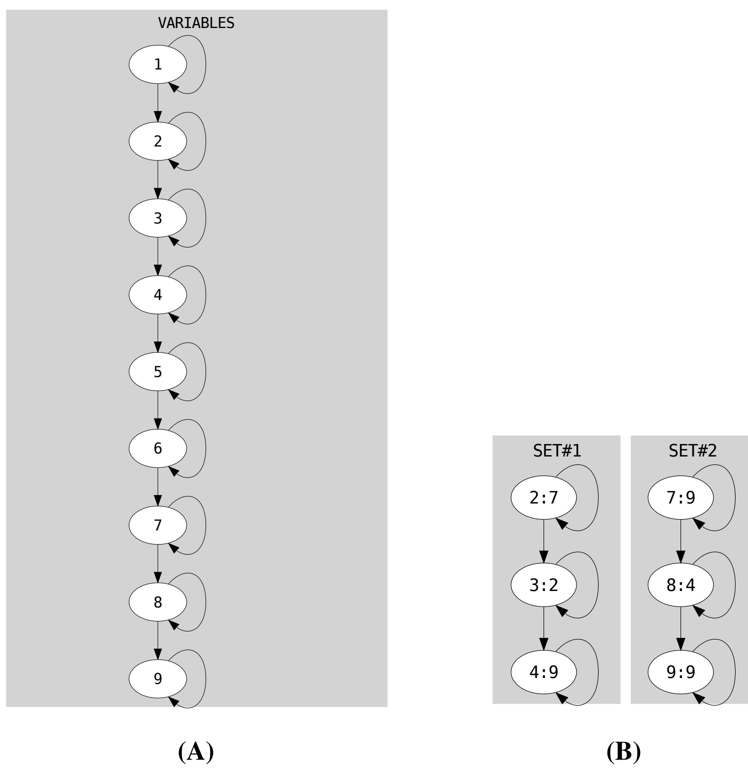


Figure 5.725: Initial and final graph of the SLIDING_CARD_SKIP0 constraint

Automaton Figure 5.726 depicts the automaton associated with the SLIDING_CARD_SKIP0 constraint. To each variable $\texttt{VAR}_i$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\texttt{VAR}_i$ and S_i:

$(\texttt{VAR}_i = 0) \Leftrightarrow S_i = 0 \wedge$

$(\texttt{VAR}_i \neq 0 \wedge \texttt{VAR}_i \notin \texttt{VALUES}) \Leftrightarrow S_i = 1 \wedge$

$(\texttt{VAR}_i \neq 0 \wedge \texttt{VAR}_i \in \texttt{VALUES}) \Leftrightarrow S_i = 2.$

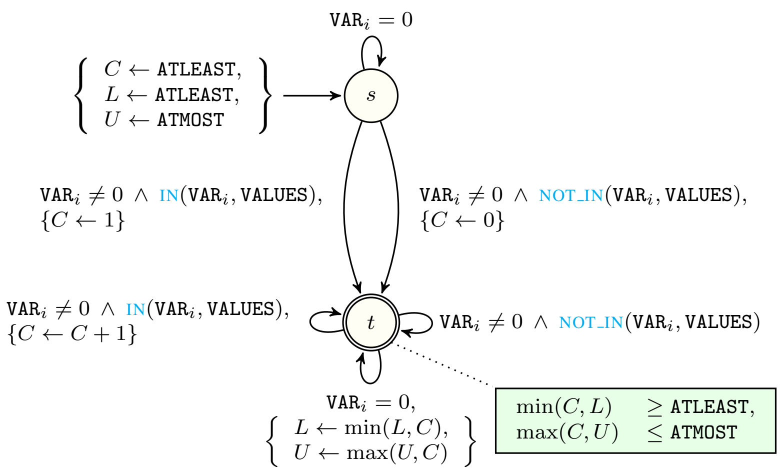


Figure 5.726: Automaton of the SLIDING_CARD_SKIP0 constraint

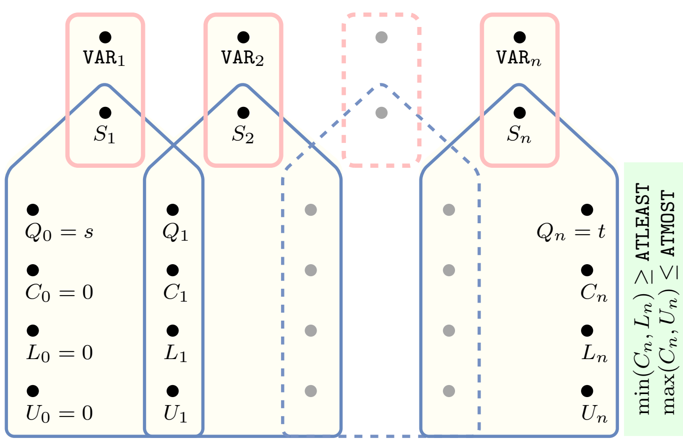


Figure 5.727: Hypergraph of the reformulation corresponding to the automaton of the SLIDING_CARD_SKIP0 constraint

5.357 SLIDING_DISTRIBUTION

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [362]

Constraint SLIDING_DISTRIBUTION(SEQ, VARIABLES, VALUES)

Arguments

SEQ	:	int
VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int, omin−int, omax−int)

Restrictions

SEQ >0
SEQ $\leq$ |VARIABLES|
required(VARIABLES, var)
|VALUES| >0
required(VALUES, [val, omin, omax])
distinct(VALUES, val)
VALUES.omin ≥ 0
VALUES.omax $\leq$ SEQ
VALUES.omin $\leq$ VALUES.omax

Purpose

For each sequence of SEQ consecutive variables of the VARIABLES collection, each value VALUES$[i]$.val ($1 \leq i \leq$ |VALUES|) should be taken by at least VALUES$[i]$.omin and at most VALUES$[i]$.omax variables.

Example

$$\left(\begin{array}{l} 4, \langle 0,5,0,6,5,0,0\rangle, \\ \left\langle\begin{array}{lll} \text{val}-0 & \text{omin}-1 & \text{omax}-2, \\ \text{val}-1 & \text{omin}-0 & \text{omax}-4, \\ \text{val}-4 & \text{omin}-0 & \text{omax}-4, \\ \text{val}-5 & \text{omin}-1 & \text{omax}-2, \\ \text{val}-6 & \text{omin}-0 & \text{omax}-2 \end{array}\right\rangle \end{array}\right)$$

The SLIDING_DISTRIBUTION constraint holds since:

- On the first sequence of 4 consecutive values 0 5 0 6 values 0, 1, 4, 5 and 6 are respectively used 2, 0, 0, 1 and 1 times.
- On the second sequence of 4 consecutive values 5 0 6 5 values 0, 1, 4, 5 and 6 are respectively used 1, 0, 0, 2 and 1 times.
- On the third sequence of 4 consecutive values 0 6 5 0 values 0, 1, 4, 5 and 6 are respectively used 2, 0, 0, 1 and 1 times.
- On the fourth sequence of 4 consecutive values 6 5 0 0 values 0, 1, 4, 5 and 6 are respectively used 2, 0, 0, 1 and 1 times.

Typical

SEQ >1
SEQ $<$ |VARIABLES|
|VARIABLES| $>$ |VALUES|

Symmetries

- Items of VARIABLES can be reversed.
- An occurrence of a value of VARIABLES.var that does not belong to VALUES.val can be replaced by any other value that also does not belong to VALUES.val.
- Items of VALUES are permutable.
- VALUES.omin can be decreased to any value ≥ 0.
- VALUES.omax can be increased to any value $\leq$ SEQ.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Arg. properties

- Contractible wrt. VARIABLES when SEQ $= 1$.
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.
- Contractible wrt. VALUES.

See also

common keyword: PATTERN, SLIDING_SUM, STRETCH_CIRCUIT, STRETCH_PATH *(sliding sequence constraint)*.

part of system of constraints: GLOBAL_CARDINALITY_LOW_UP.

specialisation: AMONG_SEQ *(individual values replaced by single set of values)*.

used in graph description: GLOBAL_CARDINALITY_LOW_UP.

Keywords

characteristic of a constraint: hypergraph.

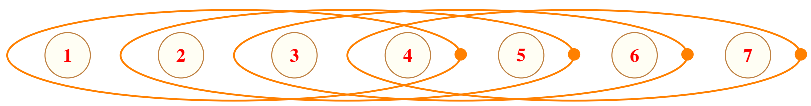
combinatorial object: sequence.

constraint type: decomposition, sliding sequence constraint, system of constraints.

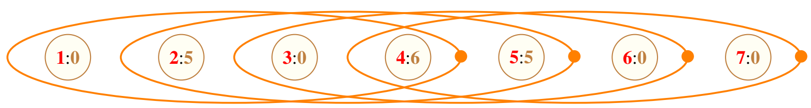
Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection
Arc arity	SEQ
Arc constraint(s)	GLOBAL_CARDINALITY_LOW_UP(collection, VALUES)
Graph property(ies)	**NARC** $= \vert$VARIABLES$\vert -$ SEQ $+ 1$

Graph model Note that the SLIDING_DISTRIBUTION constraint is a constraint where the arc constraints do not have an arity of 2.

Parts (A) and (B) of Figure 5.728 respectively show the initial and final graph associated with the **Example** slot. Since all arc constraints hold (i.e., because of the graph property **NARC** $= \vert$VARIABLES$\vert -$ SEQ $+ 1$) the final graph corresponds to the initial graph.



(A)



(B)

Figure 5.728: (A) Initial and (B) final graph of the SLIDING_DISTRIBUTION($4, \langle 0, 5, 0, 6, 5, 0, 0\rangle, \langle 0\ 1\ 2,\ 1\ 0\ 4,\ 4\ 0\ 4,\ 5\ 1\ 2,\ 6\ 0\ 2\rangle$) constraint of the **Example** slot where each ellipse represents an hyperedge involving SEQ $= 4$ vertices (to each ellipse corresponds a GLOBAL_CARDINALITY_LOW_UP constraint)

5.358 SLIDING_SUM

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin CHIP

Constraint SLIDING_SUM(LOW, UP, SEQ, VARIABLES)

Synonym SEQUENCE.

Arguments

LOW	:	int
UP	:	int
SEQ	:	int
VARIABLES	:	collection(var−dvar)

Restrictions

$\texttt{UP} \geq \texttt{LOW}$
$\texttt{SEQ} > 0$
$\texttt{SEQ} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose Constrains all sequences of SEQ consecutive variables of the collection VARIABLES so that the sum of the variables belongs to interval [LOW, UP].

Example $(3, 7, 4, \langle 1, 4, 2, 0, 0, 3, 4\rangle)$

The example considers all sliding sequences of SEQ $=$ 4 consecutive values of $\langle 1, 4, 2, 0, 0, 3, 4\rangle$ collection and constraints the sum to be in [LOW, UP] $= [3, 7]$. The SLIDING_SUM constraint holds since the sum associated with the corresponding subsequences 1 4 2 0, 4 2 0 0, 2 0 0 3, and 0 0 3 4 are respectively 7, 6, 5 and 7.

Typical

$\texttt{LOW} \geq 0$
$\texttt{UP} > 0$
$\texttt{SEQ} > 1$
$\texttt{SEQ} < |\texttt{VARIABLES}|$
$\texttt{VARIABLES.var} \geq 0$
$\texttt{UP} < \texttt{sum}(\texttt{VARIABLES.var})$

Symmetry Items of VARIABLES can be reversed.

Arg. properties

- Contractible wrt. VARIABLES when SEQ $= 1$.
- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Algorithm Beldiceanu and Carlsson [32] have proposed a first incomplete filtering algorithm for the SLIDING_SUM constraint. In 2008, Maher *et al.* showed in [284] that the SLIDING_SUM constraint has a solution "*if and only there are no negative cycles in the flow graph associated with the dual linear program*" that encodes the conjunction of inequalities. They derive a bound-consistency filtering algorithm from this fact.

Systems SLIDING_SUM in **MiniZinc**.

See also **common keyword:** SLIDING_DISTRIBUTION *(sliding sequence constraint)*.

part of system of constraints: SUM_CTR.

soft variant: RELAXED_SLIDING_SUM.

used in graph description: SUM_CTR.

Keywords **characteristic of a constraint:** hypergraph, sum.

combinatorial object: sequence.

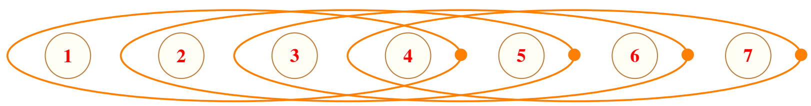
constraint type: decomposition, sliding sequence constraint, system of constraints.

filtering: linear programming, flow, bound-consistency.

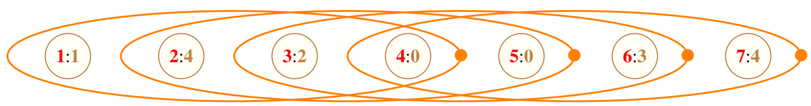
Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection
Arc arity	SEQ
Arc constraint(s)	• SUM_CTR(collection, $\geq$, LOW) • SUM_CTR(collection, $\leq$, UP)
Graph property(ies)	$\mathbf{NARC} = \lvert \mathtt{VARIABLES} \rvert - \mathtt{SEQ} + 1$

Graph model We use SUM_CTR as an arc constraint. SUM_CTR takes a collection of domain variables as its first argument.

Parts (A) and (B) of Figure 5.729 respectively show the initial and final graph associated with the **Example** slot. Since all arc constraints hold (i.e., because of the graph property $\mathbf{NARC} = \lvert \mathtt{VARIABLES} \rvert - \mathtt{SEQ} + 1$) the final graph corresponds to the initial graph.



(A)



(B)

Figure 5.729: (A) Initial and (B) final graph of the SLIDING_SUM$(3, 7, 4, \langle 1, 4, 2, 0, 0, 3, 4\rangle)$ constraint of the **Example** slot where each ellipse represents an hyperedge involving $\mathtt{SEQ} = 4$ vertices (e.g., the first ellipse represents the constraint $1 + 4 + 2 + 0 \in [3, 7]$)

Signature Since we use the $PATH$ arc generator with an arity of SEQ on the items of the VARIABLES collection, the expression $\lvert \mathtt{VARIABLES} \rvert - \mathtt{SEQ} + 1$ corresponds to the maximum number of arcs of the final graph. Therefore we can rewrite the graph property $\mathbf{NARC} = \lvert \mathtt{VARIABLES} \rvert - \mathtt{SEQ} + 1$ to $\mathbf{NARC} \geq \lvert \mathtt{VARIABLES} \rvert - \mathtt{SEQ} + 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.359 SLIDING_TIME_WINDOW

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint SLIDING_TIME_WINDOW(WINDOW_SIZE, LIMIT, TASKS)

Arguments

WINDOW_SIZE : int
LIMIT : int
TASKS : collection(origin−dvar, duration−dvar)

Restrictions

WINDOW_SIZE > 0
LIMIT ≥ 0
required(TASKS, [origin, duration])
TASKS.duration ≥ 0

Purpose For any time window of size WINDOW_SIZE, the intersection of all the tasks of the collection TASKS with this time window is less than or equal to a given limit LIMIT.

Example

$$\left(9, 6, \left\langle \begin{array}{ll} \texttt{origin}-10 & \texttt{duration}-3, \\ \texttt{origin}-5 & \texttt{duration}-1, \\ \texttt{origin}-6 & \texttt{duration}-2, \\ \texttt{origin}-14 & \texttt{duration}-2, \\ \texttt{origin}-2 & \texttt{duration}-2 \end{array} \right\rangle \right)$$

The lower part of Figure 5.730 indicates the different tasks on the time axis. Each task is drawn as a rectangle with its corresponding identifier in the middle. Finally the upper part of Figure 5.730 shows the different time windows and the respective contribution of the tasks in these time windows. Note that we only need to focus on those time windows starting at the start of one of the tasks. A line with two arrows depicts each time window. The two arrows indicate the start and the end of the time window. At the left of each time window we give its occupation. Since this occupation is always less than or equal to the limit 6, the SLIDING_TIME_WINDOW constraint holds.

Typical

WINDOW_SIZE > 1
LIMIT > 0
LIMIT $<$ sum(TASKS.duration)
$|$TASKS$| > 1$
TASKS.duration > 0

Symmetries

- WINDOW_SIZE can be decreased.
- LIMIT can be increased.
- Items of TASKS are permutable.
- One and the same constant can be added to the origin attribute of all items of TASKS.
- TASKS.duration can be decreased to any value ≥ 0.

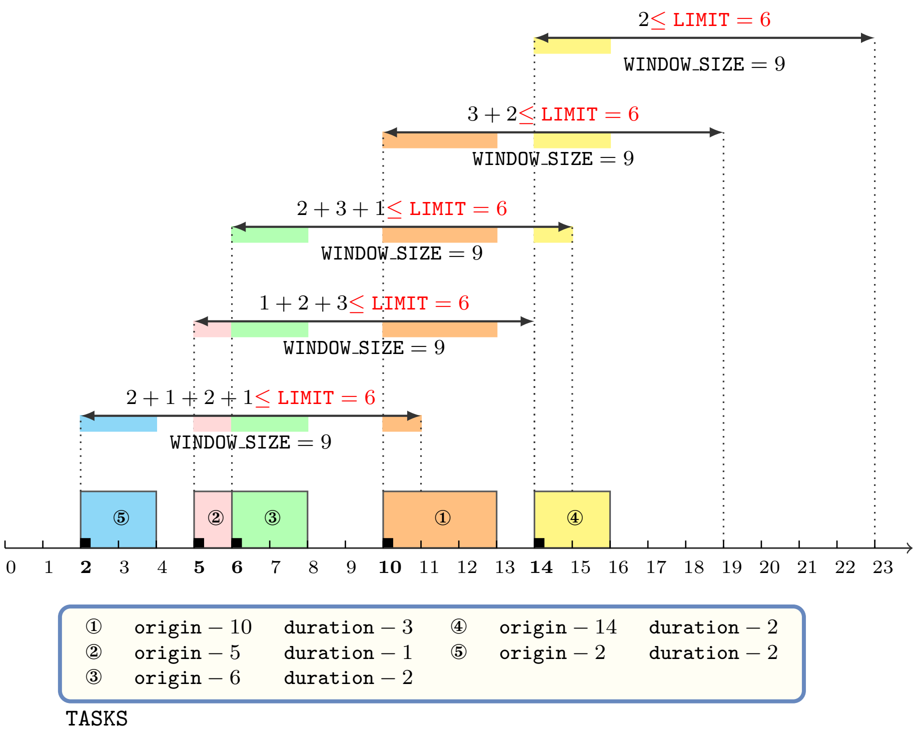


Figure 5.730: Time windows and their uses for the five tasks of the **Example** slot

Arg. properties Contractible wrt. TASKS.

Usage The SLIDING_TIME_WINDOW constraint is useful for timetabling problems in order to put an upper limit on the total work over sliding time windows.

Reformulation The SLIDING_TIME_WINDOW constraint can be expressed in term of a set of $|\text{TASKS}|^2$ reified constraints and of $|\text{TASKS}|$ linear inequalities constraints:

1. For each pair of tasks $\text{TASKS}[i], \text{TASKS}[j]$ $(i, j \in [1, |\text{TASKS}|])$ of the TASKS collection we create a variable $Inter_{ij}$ which is set to the intersection of $\text{TASKS}[j]$ with the time window $\mathcal{W}_i$ of size WINDOW_SIZE that starts at instant $\text{TASKS}[i].\text{origin}$:
   - If $i = j$ (i.e., $\text{TASKS}[i]$ and $\text{TASKS}[j]$ coincide):
     - $Inter_{ij} = \min(\text{TASKS}[i].\text{duration}, \text{WINDOW\_SIZE})$.
   - If $i \neq j$ and $\overline{\text{TASKS}[j].\text{origin}} + \overline{\text{TASKS}[j].\text{duration}} < \underline{\text{TASKS}[i].\text{origin}}$ (i.e., $\text{TASKS}[j]$ for sure ends before the time window $\mathcal{W}_i$):
     - $Inter_{ij} = 0$.
   - If $i \neq j$ and $\underline{\text{TASKS}[j].\text{origin}} > \overline{\text{TASKS}[i].\text{origin}} + \text{WINDOW\_SIZE} - 1$ (i.e., $\text{TASKS}[j]$ for sure starts after the time window $\mathcal{W}_i$):
     - $Inter_{ij} = 0$.
   - Otherwise (i.e., $\text{TASKS}[j]$ can potentially overlap the time window $\mathcal{W}_i$):

- $Inter_{ij} = \max(0, \min(\texttt{TASKS}[i].\texttt{origin} + \texttt{WINDOW\_SIZE}, \texttt{TASKS}[j].\texttt{origin} + \texttt{TASKS}[j].\texttt{duration}) - \max(\texttt{TASKS}[i].\texttt{origin}, \texttt{TASKS}[j].\texttt{origin}))$.

2. For each task $\texttt{TASKS}[i]$ $(i \in [1, |\texttt{TASKS}|])$ we create a linear inequality constraint $Inter_{i1} + Inter_{i2} + \cdots + Inter_{i|\texttt{TASKS}|} \leq \texttt{LIMIT}$.

See also

common keyword: SHIFT *(temporal constraint)*.

related: SLIDING_TIME_WINDOW_SUM *(sum of intersections of tasks with sliding time window replaced by sum of the points of intersecting tasks with sliding time window)*.

used in graph description: SLIDING_TIME_WINDOW_FROM_START.

Keywords

constraint type: sliding sequence constraint, temporal constraint.

Arc input(s)	TASKS
Arc generator	$CLIQUE \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.origin $\leq$ tasks2.origin • tasks2.origin $-$ tasks1.origin $<$ WINDOW_SIZE
Sets	SUCC $\mapsto$ [source, tasks]
Constraint(s) on sets	SLIDING_TIME_WINDOW_FROM_START(WINDOW_SIZE, LIMIT, tasks, source.origin)

Graph model We generate an arc from a task t_1 to a task t_2 if task t_2 does not start before task t_1 and if task t_2 intersects the time window that starts at the origin of task t_1. Each set generated by SUCC corresponds to all tasks that intersect in time the time window that starts at the origin of a given task.

Parts (A) and (B) of Figure 5.731 respectively show the initial and final graph associated with the **Example** slot. In the final graph, the successors of a given task t correspond to the set of tasks that do not start before task t and intersect the time window that starts at the origin of task t.

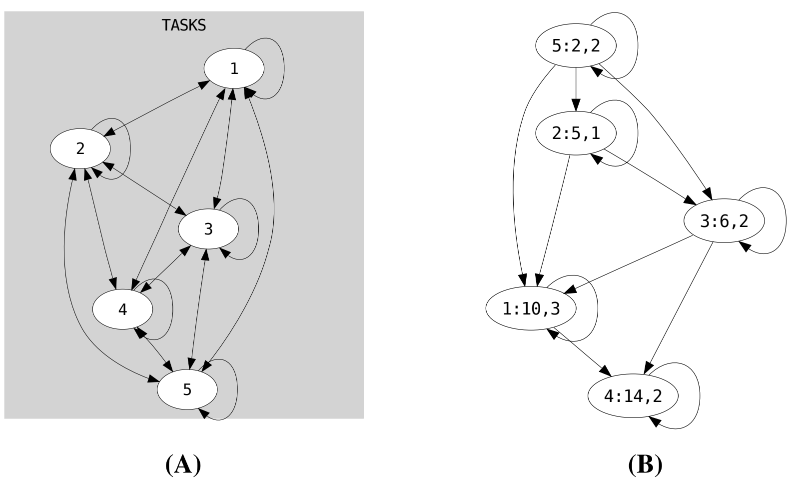


Figure 5.731: Initial and final graph of the SLIDING_TIME_WINDOW constraint

5.360 SLIDING_TIME_WINDOW_FROM_START

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Used for defining SLIDING_TIME_WINDOW.

Constraint SLIDING_TIME_WINDOW_FROM_START(WINDOW_SIZE, LIMIT, TASKS, START)

Arguments

WINDOW_SIZE	:	int
LIMIT	:	int
TASKS	:	collection(origin−dvar, duration−dvar)
START	:	dvar

Restrictions

$\texttt{WINDOW\_SIZE} > 0$
$\texttt{LIMIT} \geq 0$
required(TASKS, [origin, duration])
$\texttt{TASKS.duration} \geq 0$

Purpose The sum of the intersections of all the tasks of the TASKS collection with interval $[\texttt{START}, \texttt{START} + \texttt{WINDOW\_SIZE} - 1]$ is less than or equal to LIMIT.

Example

$$\left(9, 6, \left\langle \begin{array}{ll} \texttt{origin} - 10 & \texttt{duration} - 3, \\ \texttt{origin} - 5 & \texttt{duration} - 1, \\ \texttt{origin} - 6 & \texttt{duration} - 2 \end{array} \right\rangle, 5\right)$$

The intersections of tasks $\langle\texttt{id} - 1\ \texttt{origin} - 10\ \texttt{duration} - 3\rangle$, $\langle\texttt{id} - 2\ \texttt{origin} - 5\ \texttt{duration} - 1\rangle$, and $\langle\texttt{id} - 3\ \texttt{origin} - 6\ \texttt{duration} - 2\rangle$ with interval $[\texttt{START}, \texttt{START} + \texttt{WINDOW\_SIZE} - 1] = [5, 5 + 9 - 1] = [5, 13]$ are respectively equal to 3, 1, and 2 (i.e., the three tasks of the TASKS collection are in fact included within interval $[5, 13]$). Consequently, the SLIDING_TIME_WINDOW_FROM_START constraint holds since the sum $3 + 1 + 2$ of these intersections does not exceed the value of its second argument $\texttt{LIMIT} = 6$.

Typical

$\texttt{WINDOW\_SIZE} > 1$
$\texttt{LIMIT} > 0$
$\texttt{LIMIT} < \texttt{WINDOW\_SIZE}$
$|\texttt{TASKS}| > 1$
$\texttt{TASKS.duration} > 0$

Symmetries

- WINDOW_SIZE can be decreased.
- LIMIT can be increased.
- Items of TASKS are permutable.
- TASKS.duration can be decreased to any value ≥ 0.
- One and the same constant can be added to START as well as to the origin attribute of all items of TASKS.

Arg. properties Contractible wrt. TASKS.

Reformulation Similar to the reformulation of SLIDING_TIME_WINDOW.

Used in SLIDING_TIME_WINDOW.

Keywords **characteristic of a constraint:** derived collection.

constraint type: sliding sequence constraint, temporal constraint.

Derived Collection $\texttt{col}(\texttt{S}-\texttt{collection}(\texttt{var}-\texttt{dvar}),[\texttt{item}(\texttt{var}-\texttt{START})])$

Arc input(s)	S TASKS
Arc generator	$PRODUCT \mapsto \texttt{collection}(\texttt{s},\texttt{tasks})$
Arc arity	2
Arc constraint(s)	TRUE
Graph property(ies)	$\mathbf{SUM\_WEIGHT\_ARC}\left(\texttt{max}\left(0, \texttt{min}\left(\begin{array}{l}\texttt{s.var}+\texttt{WINDOW\_SIZE},\\ \texttt{tasks.origin}+\texttt{tasks.duration}\end{array}\right) - \texttt{max}(\texttt{s.var},\texttt{tasks.origin})\right)\right) \leq \texttt{LIMIT}$

Graph model Since we use the TRUE arc constraint the final and the initial graph are identical. The unique source of the final graph corresponds to the interval $[\texttt{START},\texttt{START}+\texttt{WINDOW\_SIZE}-1]$. Each sink of the final graph represents a given task of the TASKS collection. We associate to each arc the value given by the intersection of the task associated with one of the extremities of the arc with the time window $[\texttt{START},\texttt{START}+\texttt{WINDOW\_SIZE}-1]$. Finally, the graph property **SUM_WEIGHT_ARC** sums up all the valuations of the arcs and check that it does not exceed a given limit.

Parts (A) and (B) of Figure 5.732 respectively show the initial and final graph associated with the **Example** slot. To each arc of the final graph we associate the intersection of the corresponding sink task with interval $[\texttt{START},\texttt{START}+\texttt{WINDOW\_SIZE}-1]$. The constraint SLIDING_TIME_WINDOW_FROM_START holds since the sum of the previous intersections does not exceed LIMIT.

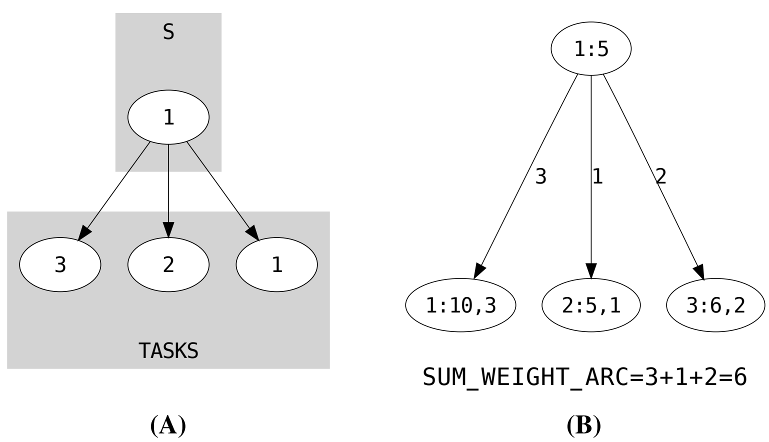


Figure 5.732: Initial and final graph of the SLIDING_TIME_WINDOW_FROM_START constraint

5.361 SLIDING_TIME_WINDOW_SUM

DESCRIPTION LINKS GRAPH

Origin

Derived from SLIDING_TIME_WINDOW.

Constraint

SLIDING_TIME_WINDOW_SUM(WINDOW_SIZE, LIMIT, TASKS)

Arguments

WINDOW_SIZE	:	int
LIMIT	:	int
TASKS	:	collection(origin−dvar, end−dvar, npoint−dvar)

Restrictions

WINDOW_SIZE > 0
LIMIT ≥ 0
required(TASKS, [origin, end, npoint])
TASKS.origin $\leq$ TASKS.end
TASKS.npoint ≥ 0

Purpose

For any time window of size WINDOW_SIZE, the sum of the points of the tasks of the collection TASKS that overlap that time window do not exceed a given limit LIMIT.

Example

$$\left(9, 16, \left\langle \begin{array}{lll} \texttt{origin}-10 & \texttt{end}-13 & \texttt{npoint}-2, \\ \texttt{origin}-5 & \texttt{end}-6 & \texttt{npoint}-3, \\ \texttt{origin}-6 & \texttt{end}-8 & \texttt{npoint}-4, \\ \texttt{origin}-14 & \texttt{end}-16 & \texttt{npoint}-5, \\ \texttt{origin}-2 & \texttt{end}-4 & \texttt{npoint}-6 \end{array} \right\rangle \right)$$

The lower part of Figure 5.733 indicates the different tasks on the time axis. Each task is drawn as a rectangle with its corresponding identifier in the middle. Finally the upper part of Figure 5.733 shows the different time windows and the respective contribution of the tasks in these time windows. A line with two arrows depicts each time window. The two arrows indicate the start and the end of the time window. At the right of each time window we give its occupation. Since this occupation is always less than or equal to the limit 16, the SLIDING_TIME_WINDOW_SUM constraint holds.

Typical

WINDOW_SIZE > 1
LIMIT > 0
LIMIT $<$sum(TASKS.npoint)
$|$TASKS$| > 1$
TASKS.origin $<$ TASKS.end
TASKS.npoint > 0

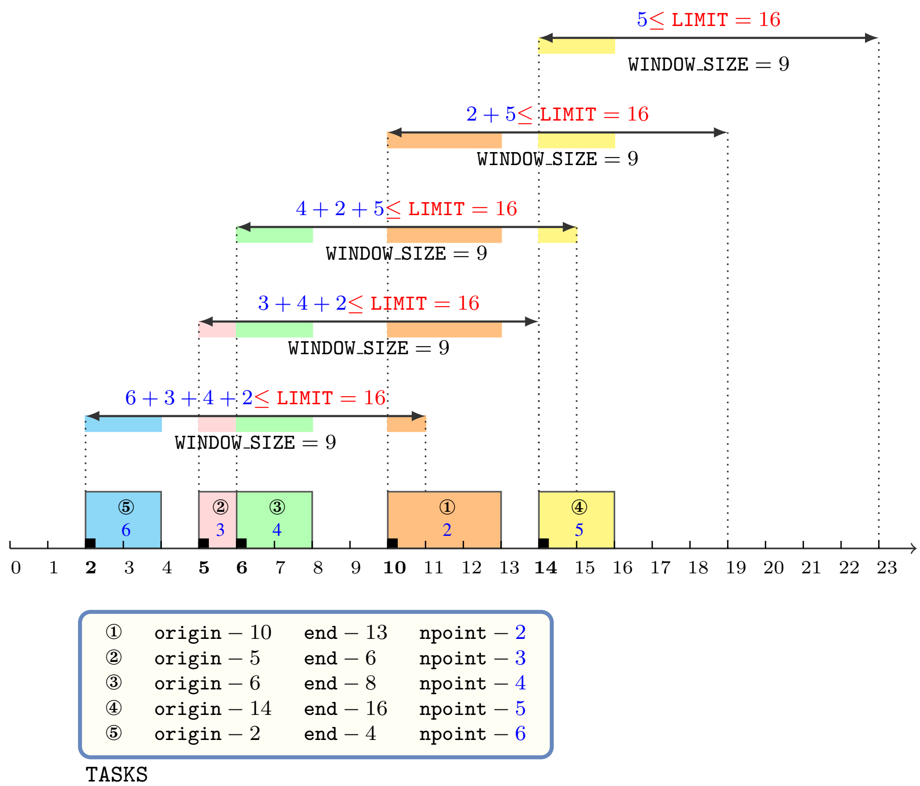


Figure 5.733: Time windows and their uses for the five tasks of the **Example** slot

Symmetries

- WINDOW_SIZE can be decreased.
- LIMIT can be increased.
- Items of TASKS are permutable.
- TASKS.npoint can be decreased to any value ≥ 0.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.

Arg. properties

Contractible wrt. TASKS.

Usage

This constraint may be used for timetabling problems in order to put an upper limit on the cumulated number of points in a shift.

Reformulation

The SLIDING_TIME_WINDOW_SUM constraint can be expressed in term of a set of $|\text{TASKS}|^2$ reified constraints and of $|\text{TASKS}|$ linear inequalities constraints:

1. For each pair of tasks $\text{TASKS}[i], \text{TASKS}[j]$ $(i, j \in [1, |\text{TASKS}|])$ of the TASKS collection we create a variable $Point_{ij}$ which is set to $\text{TASKS}[j].\text{npoint}$ if $\text{TASKS}[j]$ intersects the time window $\mathcal{W}_i$ of size WINDOW_SIZE that starts at instant $\text{TASKS}[i].\text{origin}$, or 0 otherwise:

- If $i = j$ (i.e., $\texttt{TASKS}[i]$ and $\texttt{TASKS}[j]$ coincide):
  - $Point_{ij} = \texttt{TASKS}[i].\texttt{npoint}$.
- If $i \neq j$ and $\overline{\texttt{TASKS}[j].\texttt{end}} < \underline{\texttt{TASKS}[i].\texttt{origin}}$ (i.e., $\texttt{TASKS}[j]$ for sure ends before the time window $\mathcal{W}_i$):
  - $Point_{ij} = 0$.
- If $i \neq j$ and $\underline{\texttt{TASKS}[j].\texttt{origin}} > \overline{\texttt{TASKS}[i].\texttt{origin}} + \texttt{WINDOW\_SIZE} - 1$ (i.e., $\texttt{TASKS}[j]$ for sure starts after the time window $\mathcal{W}_i$):
  - $Point_{ij} = 0$.
- Otherwise (i.e., $\texttt{TASKS}[j]$ can potentially overlap the time window $\mathcal{W}_i$):
  - $Point_{ij} = \min(1, \max(0, \min(\texttt{TASKS}[i].\texttt{origin} + \texttt{WINDOW\_SIZE}, \texttt{TASKS}[j].\texttt{end}) - \max(\texttt{TASKS}[i].\texttt{origin}, \texttt{TASKS}[j].\texttt{origin}))) \cdot \texttt{TASKS}[j].\texttt{npoint}$.

2. For each task $\texttt{TASKS}[i]$ ($i \in [1, |\texttt{TASKS}|]$) we create a linear inequality constraint $Point_{i1} + Point_{i2} + \cdots + Point_{i|\texttt{TASKS}|} \leq \texttt{LIMIT}$.

See also

related: SLIDING_TIME_WINDOW *(sum of the points of intersecting tasks with sliding time window replaced by sum of intersections of tasks with sliding time window)*.

used in graph description: SUM_CTR.

Keywords

characteristic of a constraint: time window, sum.

constraint type: sliding sequence constraint, temporal constraint.

Arc input(s)	TASKS
Arc generator	$SELF \mapsto$ collection(tasks)
Arc arity	1
Arc constraint(s)	tasks.origin $\leq$ tasks.end
Graph property(ies)	$\mathbf{NARC} = \|\mathtt{TASKS}\|$

Arc input(s)	TASKS
Arc generator	$CLIQUE \mapsto$ collection(tasks1, tasks2)
Arc arity	2
Arc constraint(s)	• tasks1.end $\leq$ tasks2.end • tasks2.origin $-$ tasks1.end $<$ WINDOW_SIZE $-$ 1
Sets	$\mathsf{SUCC} \mapsto \left[\begin{array}{l}\mathtt{source},\\ \mathtt{variables}-\mathtt{col}\left(\begin{array}{l}\mathtt{VARIABLES}-\mathtt{collection}(\mathtt{var}-\mathtt{dvar}),\\ [\mathtt{item}(\mathtt{var}-\mathtt{TASKS.npoint})]\end{array}\right)\end{array}\right]$
Constraint(s) on sets	SUM_CTR(variables, $\leq$, LIMIT)

Graph model We generate an arc from a task t_1 to a task t_2 if task t_2 does not end before the end of task t_1 and if task t_2 intersects the time window that starts at the last instant of task t_1. Each set generated by SUCC corresponds to all tasks that intersect in time the time window that starts at instant end $-$ 1, where end is the end of a given task.

Parts (A) and (B) of Figure 5.734 respectively show the initial and final graph associated with the **Example** slot. In the final graph, the successors of a given task t correspond to the set of tasks that both do not end before the end of task t, and intersect the time window that starts at the end $-$ 1 of task t.

Signature Consider the first graph constraint. Since we use the $SELF$ arc generator on the TASKS collection the maximum number of arcs of the final graph is equal to |TASKS|. Therefore we can rewrite $\mathbf{NARC} = |\mathtt{TASKS}|$ to $\mathbf{NARC} \geq |\mathtt{TASKS}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

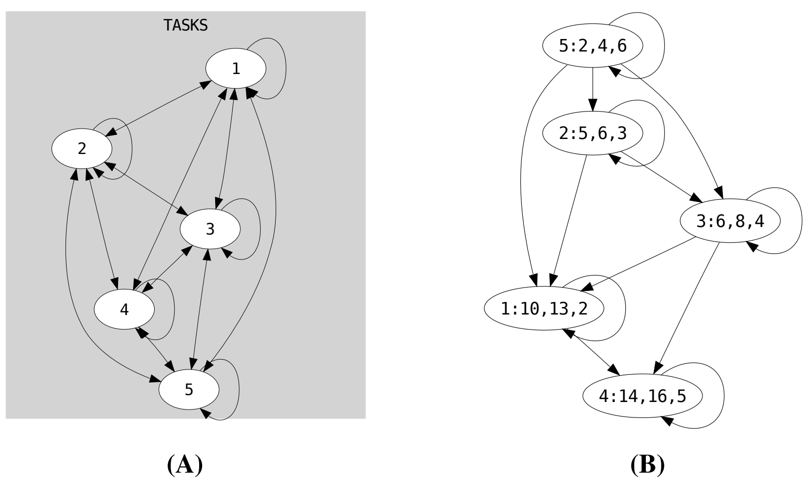


Figure 5.734: Initial and final graph of the SLIDING_TIME_WINDOW_SUM constraint

5.362 SMOOTH

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from CHANGE.

Constraint SMOOTH(NCHANGE, TOLERANCE, VARIABLES)

Arguments

```
NCHANGE    : dvar
TOLERANCE  : int
VARIABLES  : collection(var-dvar)
```

Restrictions

NCHANGE ≥ 0
NCHANGE $<$ |VARIABLES|
TOLERANCE ≥ 0
required(VARIABLES, var)

Purpose NCHANGE is the number of times that $|X - Y| >$ TOLERANCE holds; X and Y correspond to consecutive variables of the collection VARIABLES.

Example $(1, 2, \langle 1, 3, 4, 5, 2\rangle)$

In the example we have one change between values 5 and 2 since the difference in absolute value is greater than the tolerance (i.e., $|5 - 2| > 2$). Consequently the NCHANGE argument is fixed to 1 and the SMOOTH constraint holds.

Typical

TOLERANCE > 0
|VARIABLES| > 3
range(VARIABLES.var) > 1

Typical model nval(VARIABLES.var) > 2

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: NCHANGE determined by TOLERANCE and VARIABLES.
- Prefix-contractible wrt. VARIABLES when NCHANGE $= 0$.
- Suffix-contractible wrt. VARIABLES when NCHANGE $= 0$.
- Prefix-contractible wrt. VARIABLES when NCHANGE $=$ |VARIABLES| $- 1$.
- Suffix-contractible wrt. VARIABLES when NCHANGE $=$ |VARIABLES| $- 1$.

Usage This constraint is useful for the following problems:

- Assume that VARIABLES corresponds to the number of people that work on consecutive weeks. One may not normally increase or decrease too drastically the number of people from one week to the next week. With the SMOOTH constraint you can state a limit on the number of drastic changes.
- Assume you have to produce a set of orders, each order having a specific attribute. You want to generate the orders in such a way that there is not a too big difference between the values of the attributes of two consecutive orders. If you cannot achieve this on two given specific orders, this would imply a set-up or a cost. Again, with the SMOOTH constraint, you can control this kind of drastic changes.

Algorithm

A first incomplete algorithm is described in [32]. The sketch of a filtering algorithm for the conjunction of the SMOOTH and the STRETCH constraints based on dynamic programming achieving arc-consistency is mentioned by Lars Hellsten in [219, page 60].

Reformulation

The SMOOTH constraint can be reformulated with the SEQ_BIN constraint [321] that we now introduce. Given N a domain variable, X a sequence of domain variables, and C and B two binary constraints, SEQ_BIN(N, X, C, B) holds if (1) N is equal to the number of C-stretches in the sequence X, and (2) B holds on any pair of consecutive variables in X. A C-stretch is a generalisation of the notion of stretch introduced by G. Pesant [316], where the equality constraint is made explicit by replacing it by a binary constraint C, i.e., a C-stretch is a maximal length subsequence of X for which the binary constraint C is satisfied on consecutive variables. SMOOTH(NCHANGE, VARIABLES, TOLERANCE) can be reformulated as $\texttt{N} = \texttt{N1} - 1 \wedge$ SEQ_BIN($\texttt{N1}, \texttt{X}, |x_i - x_{i+1}| \leq \texttt{TOLERANCE}, \text{true}$), where true is the universal constraint.

See also

common keyword: CHANGE *(number of changes in a sequence with respect to a binary constraint)*.

related: DISTANCE.

Keywords

characteristic of a constraint: automaton, automaton with counters, non-deterministic automaton, non-deterministic automaton.

constraint arguments: pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2), Berge-acyclic constraint network.

constraint type: timetabling constraint.

filtering: glue matrix, dynamic programming.

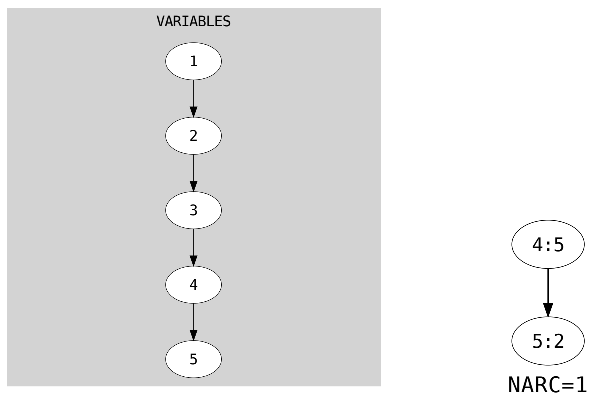
modelling: number of changes, functional dependency.

modelling exercises: n-Amazons.

puzzles: n-Amazons.

Arc input(s)	VARIABLES
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	abs(variables1.var − variables2.var) > TOLERANCE
Graph property(ies)	**NARC**= NCHANGE

Graph model Parts (A) and (B) of Figure 5.735 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.



(A) **(B)**

Figure 5.735: Initial and final graph of the SMOOTH constraint

Automaton Figure 5.736 depicts a first automaton that only accepts all the solutions to the SMOOTH constraint. This automaton uses a counter in order to record the number of satisfied constraints of the form $(|\texttt{VAR}_i - \texttt{VAR}_{i+1}|) > \texttt{TOLERANCE}$ already encountered. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and S_i: $(|\texttt{VAR}_i - \texttt{VAR}_{i+1}|) > \texttt{TOLERANCE} \Leftrightarrow S_i = 1$.

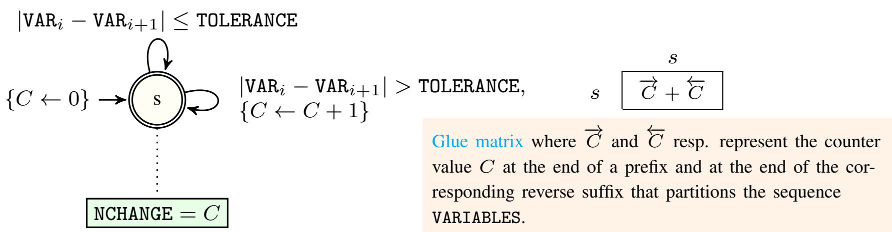


Figure 5.736: Automaton (with one counter) of the SMOOTH constraint and its glue matrix

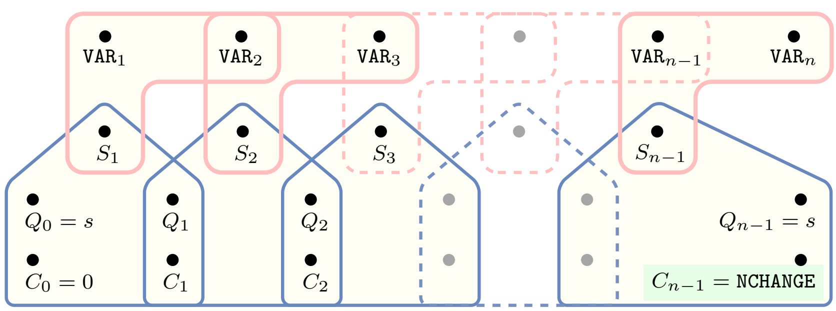


Figure 5.737: Hypergraph of the reformulation corresponding to the automaton (with one counter) of the SMOOTH constraint

Since the reformulation associated with the previous automaton is not Berge-acyclic, we now describe a second counter free automaton that also only accepts all the solutions to the SMOOTH constraint. Without loss of generality, assume that the collection of variables VARIABLES contains at least two variables (i.e., $|\texttt{VARIABLES}| \geq 2$). Let n, min, max, and $\mathcal{D}$ respectively denote the number of variables of the collection VARIABLES, the smallest value that can be assigned to the variables of VARIABLES, the largest value that can be assigned to the variables of VARIABLES, and the union of the domains of the variables of VARIABLES. Clearly, the maximum number of changes (i.e., the number of times the constraint $(|\texttt{VAR}_i - \texttt{VAR}_{i+1}|) > \texttt{TOLERANCE}$ $(1 \leq i < n)$ holds) cannot exceed the quantity $m = \min(n-1, \overline{\texttt{NCHANGE}})$. The $(m+1) \cdot |\mathcal{D}| + 2$ states of the automaton that only accepts all the solutions to the SMOOTH constraint are defined in the following way:

- We have an initial state labelled by s_I.
- We have $m \cdot |\mathcal{D}|$ intermediate states labelled by s_{ij} $(i \in \mathcal{D}, j \in [0, m])$. The first subscript i of state s_{ij} corresponds to the value currently encountered. The second

subscript j denotes the number of already encountered satisfied constraints of the form $(|\texttt{VAR}_k - \texttt{VAR}_{k+1}|) > \texttt{TOLERANCE}$ from the initial state s_I to the state s_{ij}.

- We have an accepting state labelled by s_F.

Four classes of transitions are respectively defined in the following way:

1. There is a transition, labelled by i from the initial state s_I to the state s_{i0}, $(i \in \mathcal{D})$.
2. There is a transition, labelled by j, from every state s_{ij}, $(i \in \mathcal{D}, j \in [0, m])$, to the accepting state s_F.
3. $\forall i \in \mathcal{D}$, $\forall j \in [0, m]$, $\forall k \in \mathcal{D} \cap [\max(min, i - \texttt{TOLERANCE}), \min(max, i + \texttt{TOLERANCE})]$ there is a transition labelled by k from s_{ij} to s_{kj} (i.e., the counter j does not change for values k that are too closed from value i).
4. $\forall i \in \mathcal{D}$, $\forall j \in [0, m-1]$, $\forall k \in \mathcal{D} \setminus [\max(min, i - \texttt{TOLERANCE}), \min(max, i + \texttt{TOLERANCE})]$ there is a transition labelled by k from s_{ij} to s_{kj+1} (i.e., the counter j is incremented by $+1$ for values k that are too far from i).

We have $|\mathcal{D}|$ transitions of type 1, $|\mathcal{D}| \cdot (m+1)$ transitions of type 2, and at least $|\mathcal{D}|^2 \cdot m$ transitions of types 3 and 4. Since the maximum value of m is equal to $n-1$, in the worst case we have at least $|\mathcal{D}|^2 \cdot (n-1)$ transitions. This leads to a worst case time complexity of $O(|\mathcal{D}|^2 \cdot n^2)$ if we use Pesant's algorithm for filtering the REGULAR constraint [317].

Figure 5.738 depicts the corresponding counter free non deterministic automaton associated with the SMOOTH constraint under the hypothesis that (1) all variables of VARIABLES are assigned a value in $\{0, 1, 2, 3\}$, (2) $|\texttt{VARIABLES}|$ is equal to 4, and (3) TOLERANCE is equal to 1.

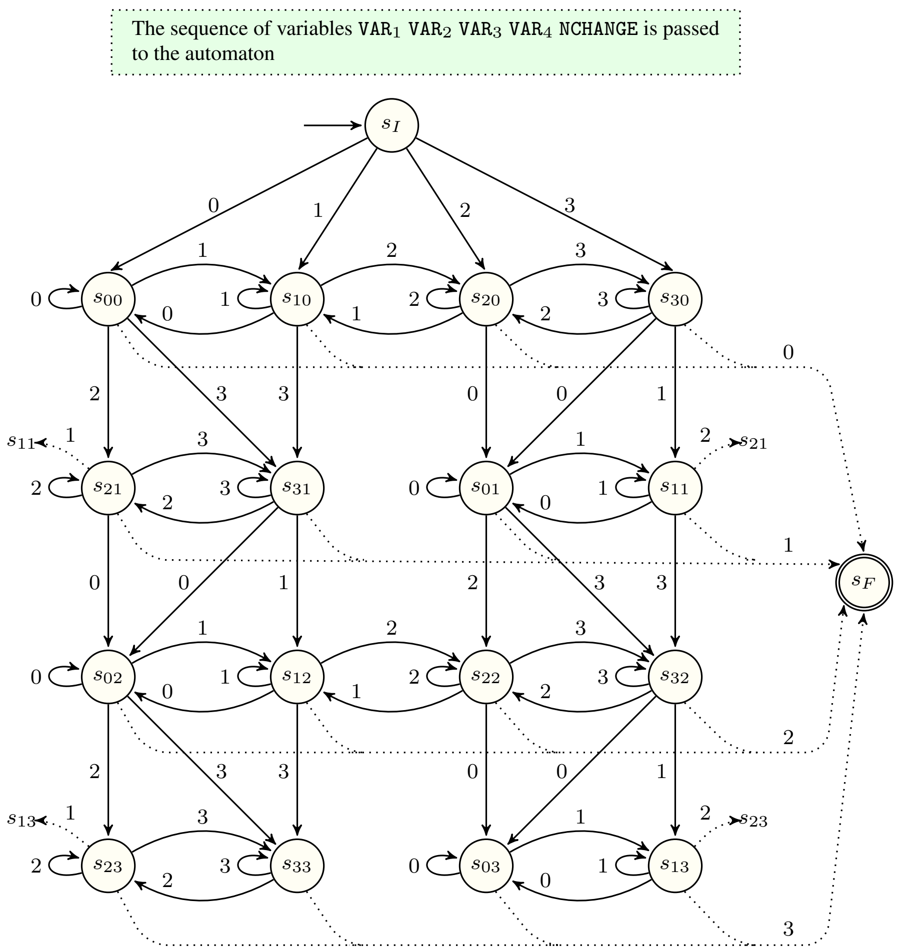


Figure 5.738: Counter free non deterministic automaton of the SMOOTH$(\mathtt{NCHANGE}, 1, \langle \mathtt{VAR}_1, \mathtt{VAR}_2, \mathtt{VAR}_3, \mathtt{VAR}_4\rangle)$ constraint assuming $\mathtt{VAR}_i \in [0, 3]$ $(1 \leq i \leq 3)$, with initial state s_I and accepting state s_F

5.363 SOFT_ALL_EQUAL_MAX_VAR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [158]

Constraint SOFT_ALL_EQUAL_MAX_VAR(N, VARIABLES)

Arguments

N : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\texttt{N} \geq 0$
$\texttt{N} \leq |\texttt{VARIABLES}|$
required(VARIABLES, var)

Purpose

Let M be the number of occurrences of the most often assigned value to the variables of the VARIABLES collection. N is less than or equal to the total number of variables of the VARIABLES collection minus M (i.e., N is less than or equal to the minimum number of variables that need to be reassigned in order to obtain a solution where all variables are assigned a same value).

Example

$(1, \langle 5, 1, 5, 5\rangle)$

Within the collection $\langle 5, 1, 5, 5\rangle$, 3 is the number of occurrences of the most assigned value. Consequently, the SOFT_ALL_EQUAL_MAX_VAR constraint holds since the argument $\texttt{N} = 1$ is less than or equal to the total number of variables 4 minus 3.

Typical

$\texttt{N} > 0$
$\texttt{N} < |\texttt{VARIABLES}|$
$\texttt{N} < |\texttt{VARIABLES}|/10 + 2$
$|\texttt{VARIABLES}| > 1$

Symmetries

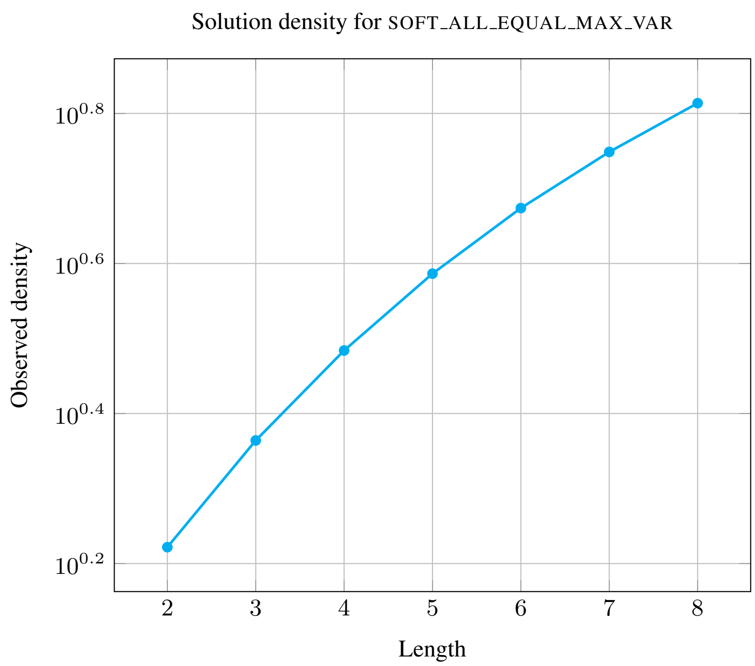
- N can be decreased to any value ≥ 0.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

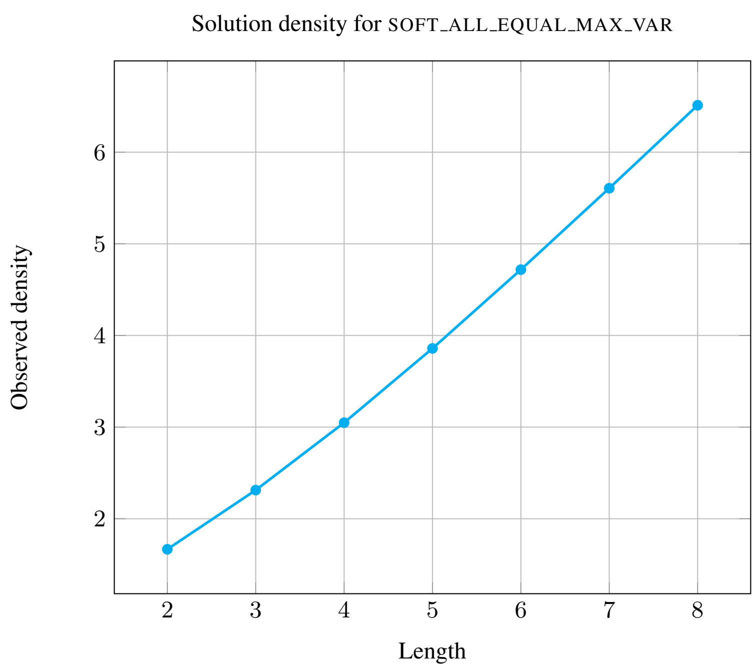
Algorithm [158].

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	15	148	1905	30006	555121	11758048	280310337

Number of solutions for SOFT_ALL_EQUAL_MAX_VAR: domains $0..n$

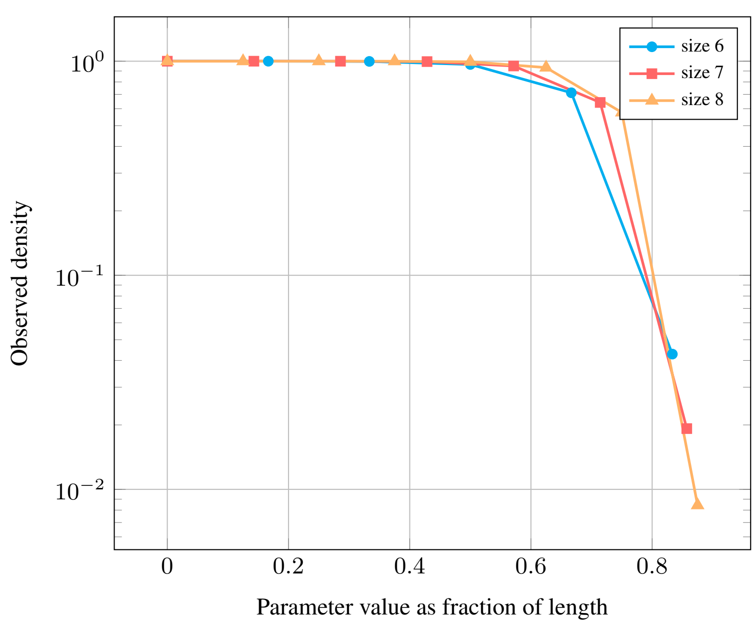

Solution density for SOFT_ALL_EQUAL_MAX_VAR
Observed density
$10^{0.8}$
$10^{0.6}$
$10^{0.4}$
$10^{0.2}$
2 3 4 5 6 7 8
Length

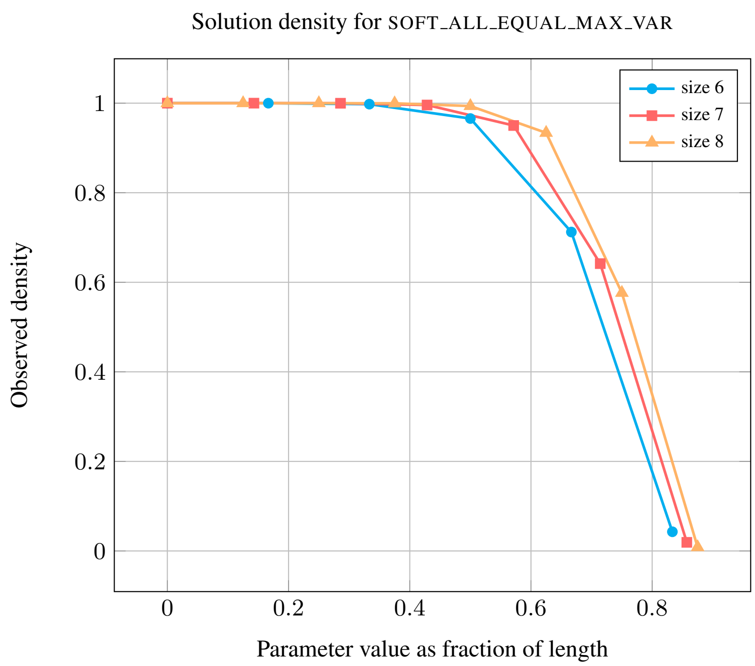

Solution density for SOFT_ALL_EQUAL_MAX_VAR
Observed density
6
5
4
3
2
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		15	148	1905	30006	555121	11758048	280310337
Parameter value	0	9	64	625	7776	117649	2097152	43046721
	1	6	60	620	7770	117642	2097144	43046712
	2	-	24	540	7620	117390	2096752	43046136
	3	-	-	120	6120	113610	2088520	43030008
	4	-	-	-	720	83790	1992480	42771960
	5	-	-	-	-	5040	1345680	40194000
	6	-	-	-	-	-	40320	24811920
	7	-	-	-	-	-	-	362880

Solution count for SOFT_ALL_EQUAL_MAX_VAR: domains $0..n$

Solution density for SOFT_ALL_EQUAL_MAX_VAR





See also

common keyword: SOFT_ALL_EQUAL_MIN_CTR, SOFT_ALL_EQUAL_MIN_VAR, SOFT_ALLDIFFERENT_CTR, SOFT_ALLDIFFERENT_VAR *(soft constraint)*.

hard version: ALL_EQUAL.

implied by: XOR.

related: ATMOST_NVALUE.

Keywords

constraint type: soft constraint, value constraint, relaxation, variable-based violation measure.

filtering: arc-consistency, bound-consistency.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**MAX_NSCC**$\leq$ |VARIABLES| − N

Graph model We generate an initial graph with binary *equalities* constraints between each vertex and its successors. The graph property states that N is less than or equal to the difference between the total number of vertices of the initial graph and the number of vertices of the largest strongly connected component of the final graph.

Parts (A) and (B) of Figure 5.739 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_NSCC** graph property we show one of the largest strongly connected components of the final graph.

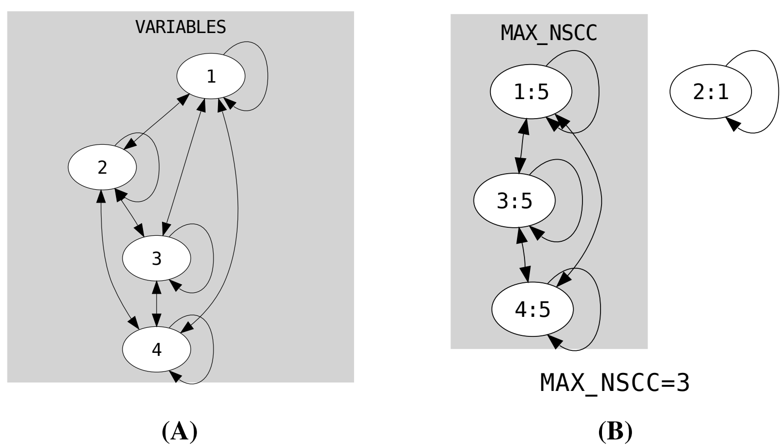


Figure 5.739: Initial and final graph of the SOFT_ALL_EQUAL_MAX_VAR constraint

5.364 SOFT_ALL_EQUAL_MIN_CTR

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [216]

Constraint SOFT_ALL_EQUAL_MIN_CTR(N, VARIABLES)

Synonyms SOFT_ALLDIFF_MAX_CTR, SOFT_ALLDIFFERENT_MAX_CTR, SOFT_ALLDISTINCT_MAX_CTR.

Arguments

N	:	int
VARIABLES	:	collection(var−dvar)

Restrictions

$\mathtt{N} \geq 0$
$\mathtt{N} \leq |\mathtt{VARIABLES}| * |\mathtt{VARIABLES}| - |\mathtt{VARIABLES}|$
required(VARIABLES, var)

Purpose Consider the *equality* constraints involving two distinct variables of the collection VARIABLES. Among the previous set of constraints, N is less than or equal to the number of *equality* constraints that hold.

Example $(6, \langle 5, 1, 5, 5\rangle)$

Within the collection $\langle 5, 1, 5, 5\rangle$ six equality constraints holds. Consequently, the SOFT_ALL_EQUAL_CTR constraint holds since the argument $\mathtt{N} = 6$ is less than or equal to the number of equality constraints that hold.

Typical

$\mathtt{N} > 0$
$\mathtt{N} < |\mathtt{VARIABLES}| * |\mathtt{VARIABLES}| - |\mathtt{VARIABLES}|$
$|\mathtt{VARIABLES}| > 1$

Symmetries

- N can be decreased to any value ≥ 0.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Remark It was shown in [216] that, finding out whether the SOFT_ALL_EQUAL_CTR constraint has a solution or not is NP-hard. This was achieved by reduction from 3-dimensional-matching. Hebrard *et al.* also identify a tractable class when no value occurs in more than two variables of the collection VARIABLES that is equivalent to the vertex matching problem. One year later, [158] shows how to achieve bound-consistency in polynomial time.

See also **common keyword:** SOFT_ALL_EQUAL_MAX_VAR, SOFT_ALL_EQUAL_MIN_VAR, SOFT_ALLDIFFERENT_CTR, SOFT_ALLDIFFERENT_VAR *(soft constraint)*.

hard version: ALL_EQUAL.

implied by: AND, BALANCE, EQUIVALENT, NOR.

related: ATMOST_NVALUE.

Keywords

complexity: 3-dimensional-matching.

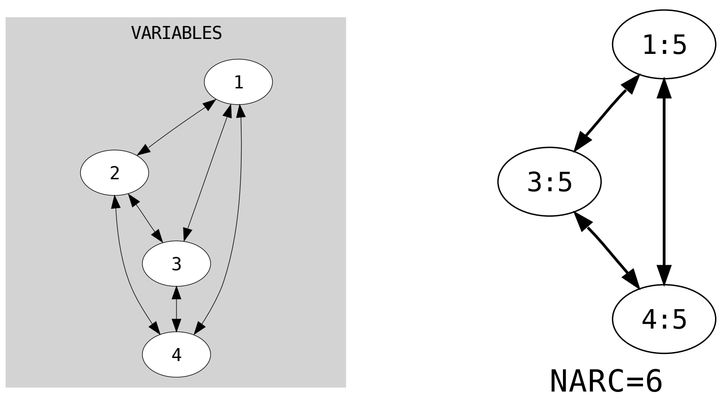
constraint type: soft constraint, value constraint, relaxation, decomposition-based violation measure.

filtering: bound-consistency.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE(\neq) \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NARC** $\geq$ N

Graph model We generate an initial graph with binary *equalities* constraints between each vertex and its successors. We use the arc generator $CLIQUE(\neq)$ in order to avoid considering *equality* constraints between the same variable. The graph property states that N is less than or equal to the number of *equalities* that hold in the final graph.

Parts (A) and (B) of Figure 5.740 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Six equality constraints remain in the final graph.



(A) (B)

Figure 5.740: Initial and final graph of the SOFT_ALL_EQUAL_MIN_CTR constraint

5.365 SOFT_ALL_EQUAL_MIN_VAR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [158]

Constraint SOFT_ALL_EQUAL_MIN_VAR(N, VARIABLES)

Arguments

N : dvar
VARIABLES : collection(var−dvar)

Restrictions

$\mathtt{N} \geq 0$
required(VARIABLES, var)

Purpose Let M be the number of occurrences of the most often assigned value to the variables of the VARIABLES collection. N is greater than or equal to the total number of variables of the VARIABLES collection minus M (i.e., N is greater than or equal to the minimum number of variables that need to be reassigned in order to obtain a solution where all variables are assigned a same value).

Example $(1, \langle 5, 1, 5, 5\rangle)$

Within the collection $\langle 5, 1, 5, 5\rangle$, 3 is the number of occurrences of the most assigned value. Consequently, the SOFT_ALL_EQUAL_MIN_VAR constraint holds since the argument $\mathtt{N} = 1$ is greater than or equal to the total number of variables 4 minus 3.

Typical

$\mathtt{N} > 0$
$\mathtt{N} < |\mathtt{VARIABLES}|$
$\mathtt{N} < |\mathtt{VARIABLES}|/10 + 2$
$|\mathtt{VARIABLES}| > 1$

Symmetries

- N can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Algorithm Let m denote the total number of potential values that can be assigned to the variables of the VARIABLES collection. In [158], E. Hebrard *et al.* provides an $O(m)$ filtering algorithm achieving arc-consistency on the SOFT_ALL_EQUAL_MIN_VAR constraint. The same paper also provides an algorithm with a lower complexity for achieving range consistency. Both algorithms are based on the following ideas:

- In a first phase, they both compute an *envelope* of the union $\mathcal{D}$ of the domains of the variables of the VARIABLES collection, i.e., an array A that indicates for each potential value v of $\mathcal{D}$, the maximum number of variables that could possibly be assigned value v. Let max_occ denote the maximum value over the entries of array A, and let $\mathcal{V}_{max\_occ}$ denote the set of values which all occur in max_occ variables of the VARIABLES collection. The quantity $|\texttt{VARIABLES}| - max\_occ$ is a lower bound of N.

- In a second phase, depending on the relative ordering between max_occ and the minimum value of $|\texttt{VARIABLES}| - \texttt{N}$, i.e., $|\texttt{VARIABLES}| - \overline{\texttt{N}}$, we have the three following cases:
  1. When $max\_occ < |\texttt{VARIABLES}| - \overline{\texttt{N}}$, the constraint SOFT_ALL_EQUAL_MIN_VAR simply fails since not enough variables of the VARIABLES collection can be assigned the same value.
  2. When $max\_occ = |\texttt{VARIABLES}| - \overline{\texttt{N}}$, the constraint SOFT_ALL_EQUAL_MIN_VAR can be satisfied. In this context, a value v can be removed from the domain of a variable V of the VARIABLES collection if and only if:
     (a) value v does not belong to $\mathcal{V}_{max\_occ}$,
     (b) the domain of variable V contains all values of $\mathcal{V}_{max\_occ}$.

     On the one hand, the first condition can be understand as the fact that value v is not a value that allows the constraint to have at least $|\texttt{VARIABLES}| - \overline{\texttt{N}}$ variables assigned the same value. On the other hand, the second condition can be interpreted as the fact that variable V is absolutely required in order to have at least $|\texttt{VARIABLES}| - \overline{\texttt{N}}$ variables assigned the same value.
  3. When $max\_occ > |\texttt{VARIABLES}| - \overline{\texttt{N}}$, the constraint SOFT_ALL_EQUAL_MIN_VAR can be satisfied, but no value can be pruned.

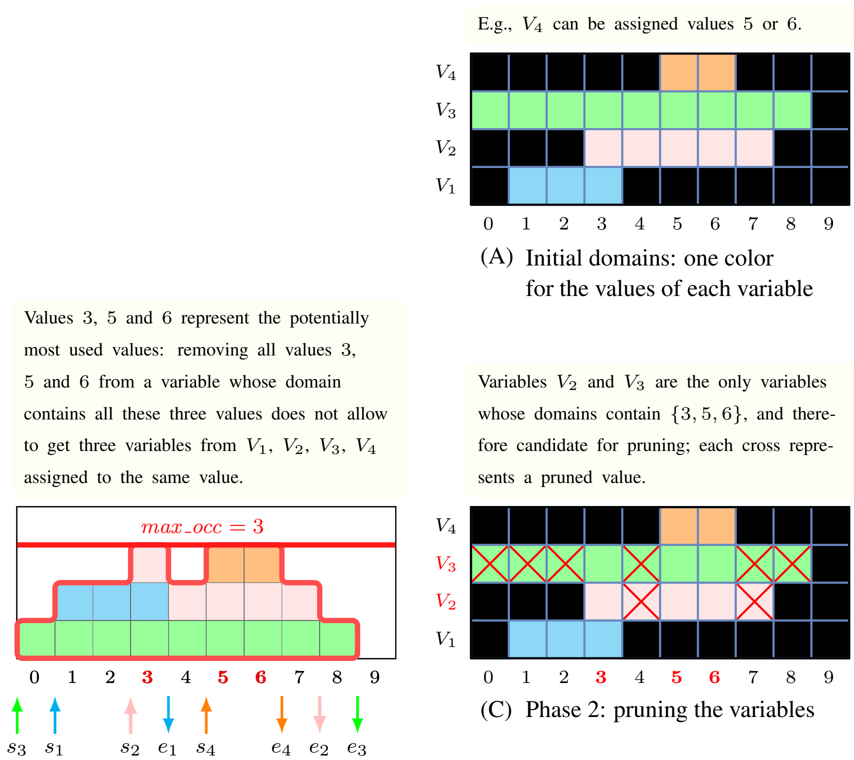
Note that, in the context of range consistency, the first phase of the filtering algorithm can be interpreted as a sweep algorithm were:

- On the one hand, the *sweep status* corresponds to the maximum number of occurrence of variables that can be assigned a given value.
- On the other hand, the *event point series* correspond to the minimum values of the variables of the VARIABLES collection as well as to the maximum values $(+1)$ of the same variables.

Figure 5.741 illustrates the previous filtering algorithm on an example where N is equal to 1, and where we have four variables V_1, V_2, V_3 and V_4 respectively taking their values within intervals $[1,3]$, $[3,7]$, $[0,8]$ and $[5,6]$ (see Part (A) of Figure 5.741, where the values of each variable are assigned a same colour that we retrieve in the other parts of Figure 5.741).

Part (B) of Figure 5.741 illustrates the first phase of the filtering algorithm, namely the computation of the envelope of the domains of variables V_1, V_2, V_3 and V_4. The *start events* s_1, s_2, s_3, s_4 (i.e., the events respectively associated with the minimum value of variables V_1, V_2, V_3, V_4) where the envelope is increased by 1 are represented by the character $\uparrow$. Similarly, the *end events* (i.e., the events e_1, e_2, e_3, e_4 respectively associated with the maximum value $(+1)$ of V_1, V_2, V_3, V_4 are represented by the character $\downarrow$). Since the highest peak of the envelope is equal to 3 we have that max_occ is equal to 3. The values that allow to reach this highest peak are equal to $\mathcal{V}_{max\_occ} = \{3,5,6\}$ (i.e., shown in red in Part (B) of Figure 5.741).

Finally, Part (C) of Figure 5.741 illustrates the second phase of the filtering algorithm. Since $max\_occ = 3$ is equal to $|\texttt{VARIABLES}| - \overline{\texttt{N}} = 4 - 1$ we remove from the variables



(A) Initial domains: one color for the values of each variable

(B) Phase 1: computing the domains envelope (in red) from the sorted start and end events s_3, s_1, s_2, e_1, s_4, e_4, e_2, e_3

(C) Phase 2: pruning the variables

Figure 5.741: Illustration of the two phases filtering algorithm of the SOFT_ALL_EQUAL_MIN_VAR$(1, \langle V_1, V_2, V_3, V_4\rangle)$ constraint with $V_1 \in [1, 3]$, $V_2 \in [3, 7]$, $V_3 \in [0, 8]$ and $V_4 \in [5, 6]$

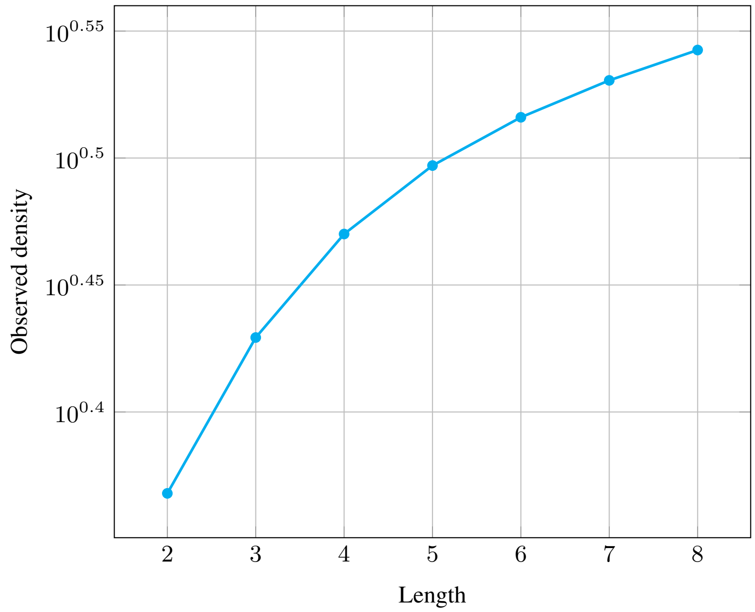
whose domains contain $\mathcal{V}_{max\_occ} = \{3, 5, 6\}$ (i.e., variables V_2 and V_3) all values not in $\mathcal{V}_{max\_occ} = \{3, 5, 6\}$ (i.e., values 4, 7 for variable V_2 and values 0, 1, 2, 4, 7, 8 for variable V_3).

Counting

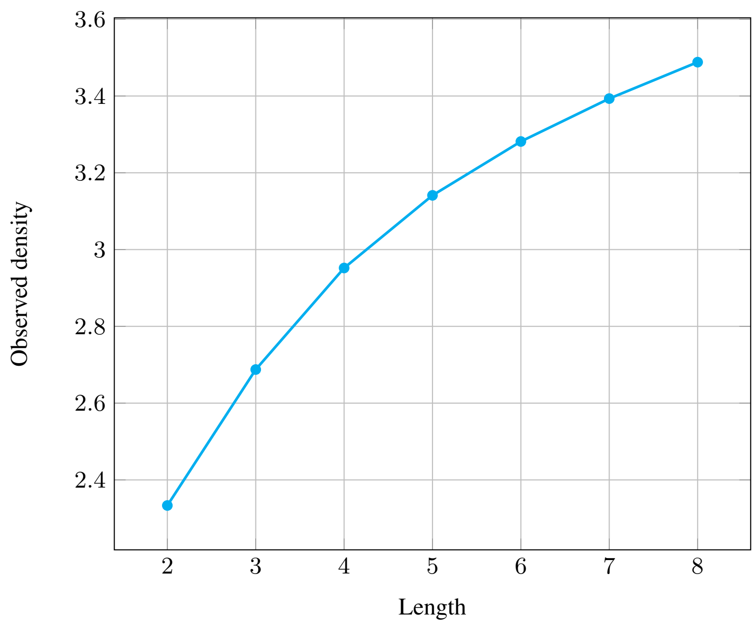
Length (n)	2	3	4	5	6	7	8
Solutions	21	172	1845	24426	386071	7116320	150156873

Number of solutions for SOFT_ALL_EQUAL_MIN_VAR: domains $0..n$

Solution density for SOFT_ALL_EQUAL_MIN_VAR



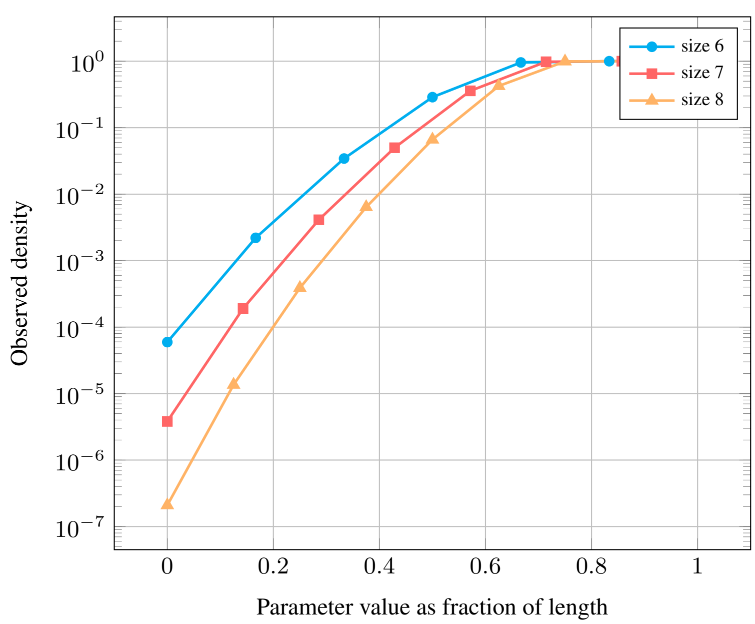
Solution density for SOFT_ALL_EQUAL_MIN_VAR

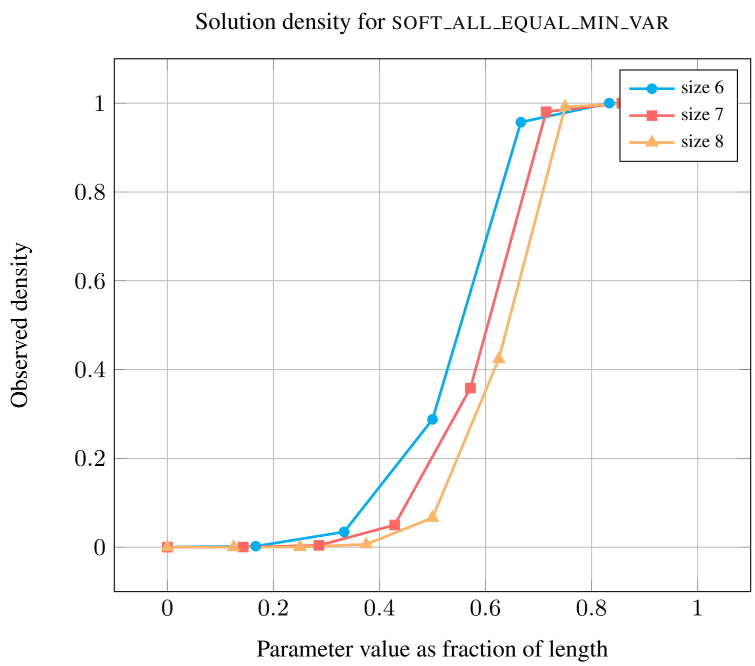


Length (n)		2	3	4	5	6	7	8
Total		21	172	1845	24426	386071	7116320	150156873
Parameter value	0	3	4	5	6	7	8	9
	1	9	40	85	156	259	400	585
	2	9	64	505	1656	4039	8632	16713
	3	-	64	625	7056	33859	104672	274761
	4	-	-	625	7776	112609	751472	2852721
	5	-	-	-	7776	117649	2056832	18234801
	6	-	-	-	-	117649	2097152	42683841
	7	-	-	-	-	-	2097152	43046721
	8	-	-	-	-	-	-	43046721

Solution count for SOFT_ALL_EQUAL_MIN_VAR: domains $0..n$

Solution density for SOFT_ALL_EQUAL_MIN_VAR





See also **common keyword:** SOFT_ALL_EQUAL_MAX_VAR, SOFT_ALL_EQUAL_MIN_CTR, SOFT_ALLDIFFERENT_CTR, SOFT_ALLDIFFERENT_VAR *(soft constraint)*.

hard version: ALL_EQUAL.

implied by: XOR.

related: ATMOST_NVALUE.

Keywords **constraint type:** soft constraint, value constraint, relaxation, variable-based violation measure.

filtering: arc-consistency, sweep.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	$\mathbf{MAX\_NSCC} \geq$ |VARIABLES| − N

Graph model We generate an initial graph with binary *equalities* constraints between each vertex and its successors. The graph property states that N is greater than or equal to the difference between the total number of vertices of the initial graph and the number of vertices of the largest strongly connected component of the final graph.

Parts (A) and (B) of Figure 5.742 respectively show the initial and final graph associated with the **Example** slot. Since we use the $\mathbf{MAX\_NSCC}$ graph property we show one of the largest strongly connected components of the final graph.

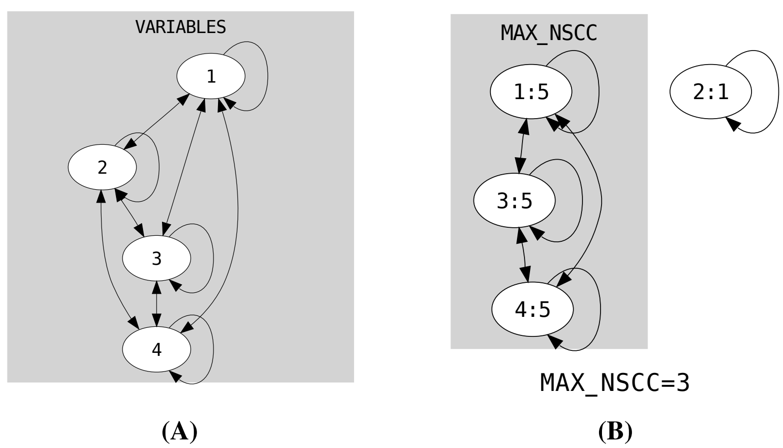


Figure 5.742: Initial and final graph of the SOFT_ALL_EQUAL_MIN_VAR constraint

5.366 SOFT_ALLDIFFERENT_CTR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [325]

Constraint SOFT_ALLDIFFERENT_CTR(C, VARIABLES)

Synonyms SOFT_ALLDIFF_CTR, SOFT_ALLDISTINCT_CTR, SOFT_ALLDIFF_MIN_CTR, SOFT_ALLDIFFERENT_MIN_CTR, SOFT_ALLDISTINCT_MIN_CTR, SOFT_ALL_EQUAL_MAX_CTR.

Arguments

C	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

$\texttt{C} \geq 0$
required(VARIABLES, var)

Purpose Consider the *disequality* constraints involving two distinct variables VARIABLES$[i]$.var and VARIABLES$[j]$.var $(i < j)$ of the collection VARIABLES. Among the previous set of constraints, C is greater than or equal to the number of *disequality* constraints that do not hold.

Example

$(4, \langle 5, 1, 9, 1, 5, 5\rangle)$
$(1, \langle 5, 1, 9, 1, 2, 6\rangle)$
$(0, \langle 5, 1, 9, 0, 2, 6\rangle)$

Within the collection $\langle 5, 1, 9, 1, 5, 5\rangle$ the first and fifth values, the first and sixth values, the second and fourth values, and the fifth and sixth values are identical. Consequently, the argument $\texttt{C} = 4$ is greater than or equal to the number of *disequality* constraints that do not hold (i.e, 4) and the SOFT_ALLDIFFERENT_CTR constraint holds.

Typical

$\texttt{C} > 0$
$\texttt{C} \leq |\texttt{VARIABLES}| * (|\texttt{VARIABLES}| - 1)/2$
$|\texttt{VARIABLES}| > 1$

Symmetries

- C can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Contractible wrt. VARIABLES.

Usage A soft ALLDIFFERENT constraint.

Remark

The SOFT_ALLDIFFERENT_CTR constraint is called SOFT_ALLDIFF_MIN_CTR or SOFT_ALL_EQUAL_MAX_CTR in [158].

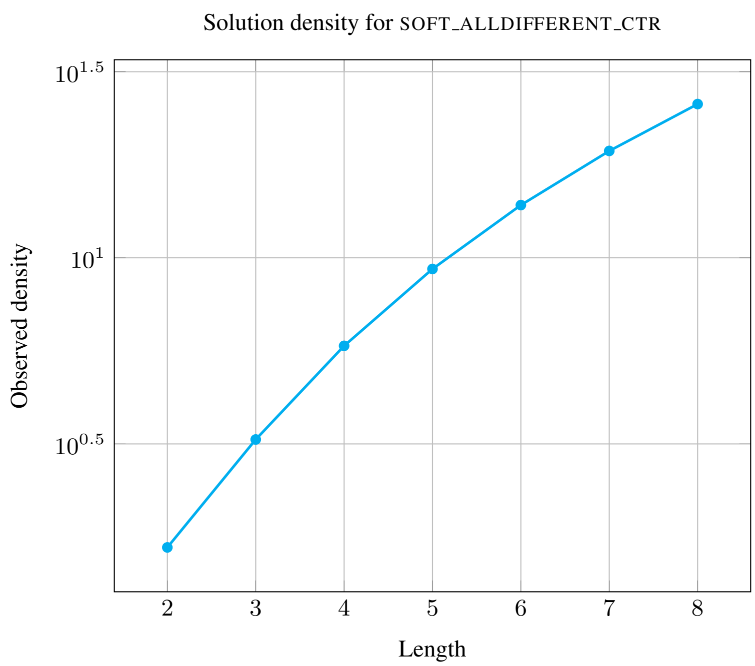
Algorithm

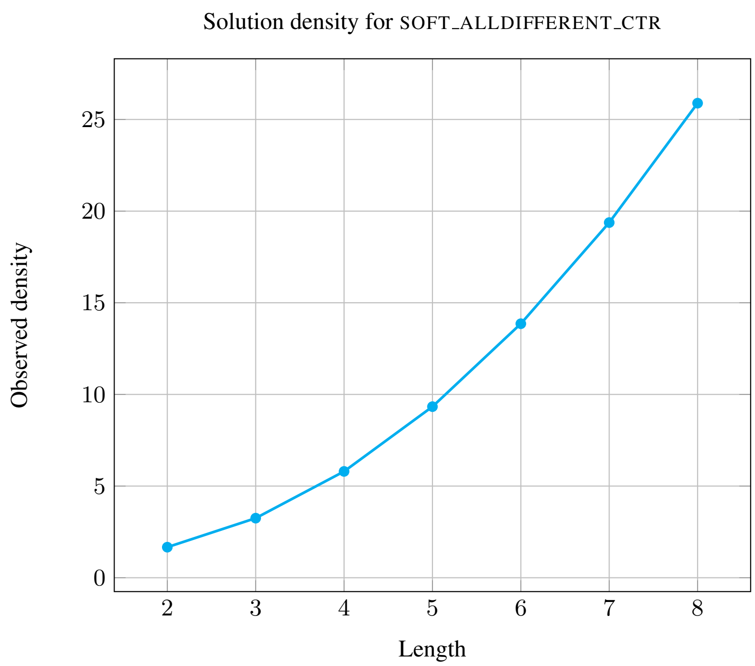
Since it focus on the soft aspect of the ALLDIFFERENT constraint, the original article [325] that introduces this constraint describes how to evaluate the minimum value of C and how to prune according to the maximum value of C. The corresponding filtering algorithm does not achieve arc-consistency. W.-J. van Hoeve [433] presents a new filtering algorithm that achieves arc-consistency. This algorithm is based on a reformulation into a minimum-cost flow problem.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	15	208	3625	72576	1630279	40632320	1114431777

Number of solutions for SOFT_ALLDIFFERENT_CTR: domains $0..n$

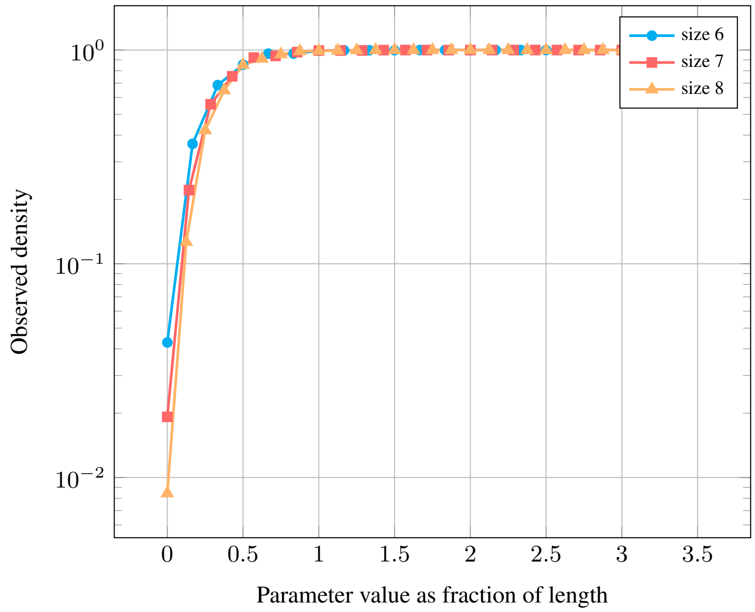



Solution density for SOFT_ALLDIFFERENT_CTR
Observed density
0
5
10
15
20
25
2
3
4
5
6
7
8
Length

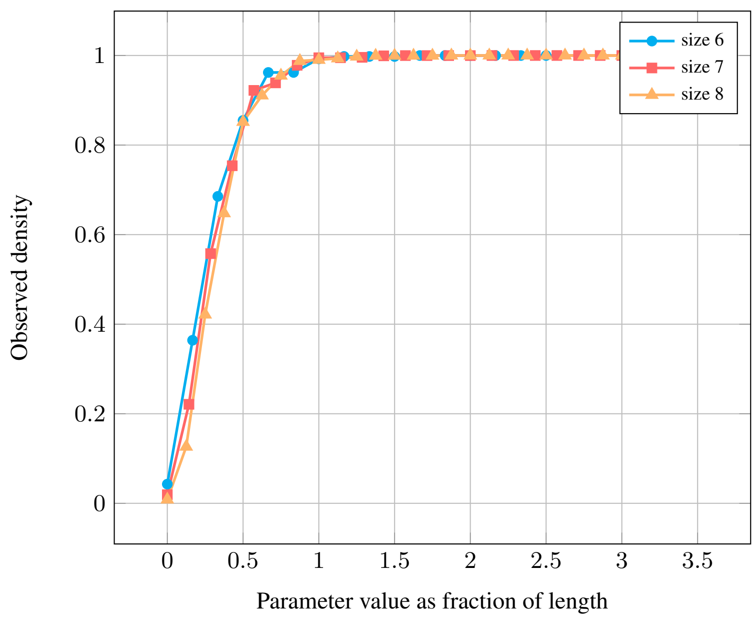
Length (n)		2	3	4	5	6	7	8
Total		15	208	3625	72576	1630279	40632320	1114431777
Parameter value	0	6	24	120	720	5040	40320	362880
	1	9	60	480	4320	42840	463680	5443200
	2	-	60	540	6120	80640	1169280	18144000
	3	-	64	620	7320	100590	1580880	27881280
	4	-	-	620	7620	113190	1933680	36666000
	5	-	-	620	7620	113190	1968960	39206160
	6	-	-	625	7770	116760	2051280	41111280
	7	-	-	-	7770	117390	2086560	42522480
	8	-	-	-	7770	117390	2086560	42628320
	9	-	-	-	7770	117390	2088520	42769440
	10	-	-	-	7776	117642	2095576	42938784
	11	-	-	-	-	117642	2096752	43023456
	12	-	-	-	-	117642	2096752	43025976
	13	-	-	-	-	117642	2096752	43030008
	14	-	-	-	-	117642	2096752	43030008
	15	-	-	-	-	117649	2097144	43044120
	16	-	-	-	-	-	2097144	43046136
	17	-	-	-	-	-	2097144	43046136
	18	-	-	-	-	-	2097144	43046136
	19	-	-	-	-	-	2097144	43046136
	20	-	-	-	-	-	2097144	43046136
	21	-	-	-	-	-	2097152	43046712
	22	-	-	-	-	-	-	43046712
	23	-	-	-	-	-	-	43046712
	24	-	-	-	-	-	-	43046712
	25	-	-	-	-	-	-	43046712
	26	-	-	-	-	-	-	43046712
	27	-	-	-	-	-	-	43046712
	28	-	-	-	-	-	-	43046721

Solution count for SOFT_ALLDIFFERENT_CTR: domains $0..n$

Solution density for SOFT_ALLDIFFERENT_CTR



Solution density for SOFT_ALLDIFFERENT_CTR



See also **common keyword:** SOFT_ALL_EQUAL_MAX_VAR, SOFT_ALL_EQUAL_MIN_CTR, SOFT_ALL_EQUAL_MIN_VAR, SOFT_ALLDIFFERENT_VAR (*soft constraint*).

hard version: ALLDIFFERENT.

implied by: EQUIVALENT, IMPLY.

implies: SOFT_ALLDIFFERENT_VAR.

related: ATMOST_NVALUE.

Keywords

characteristic of a constraint: all different, disequality.

constraint type: soft constraint, value constraint, relaxation, decomposition-based violation measure.

filtering: minimum cost flow.

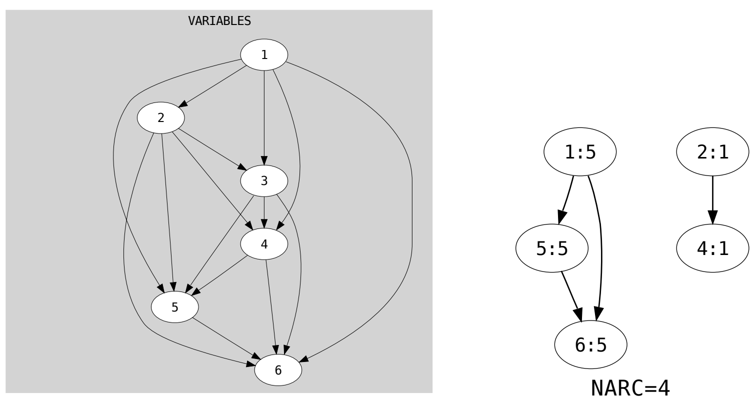
modelling: degree of diversity of a set of solutions.

modelling exercises: degree of diversity of a set of solutions.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE(<) \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NARC** $\leq$ C

Graph model We generate an initial graph with binary *equalities* constraints between each vertex and its successors. We use the arc generator $CLIQUE(<)$ in order to avoid counting twice the same *equality* constraint. The graph property states that C is greater than or equal to the number of *equalities* that hold in the final graph.

Parts (A) and (B) of Figure 5.743 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold. Since four equality constraints remain in the final graph the *cost* variable C is greater than or equal to 4.



(A) **(B)**

Figure 5.743: Initial and final graph of the SOFT_ALLDIFFERENT_CTR constraint

5.367 SOFT_ALLDIFFERENT_VAR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [325]

Constraint SOFT_ALLDIFFERENT_VAR(C, VARIABLES)

Synonyms SOFT_ALLDIFF_VAR, SOFT_ALLDISTINCT_VAR, SOFT_ALLDIFF_MIN_VAR, SOFT_ALLDIFFERENT_MIN_VAR, SOFT_ALLDISTINCT_MIN_VAR.

Arguments

C	:	dvar
VARIABLES	:	collection(var−dvar)

Restrictions

$\texttt{C} \geq 0$
required(VARIABLES, var)

Purpose C is greater than or equal to the minimum number of variables of the collection VARIABLES for which the value needs to be changed in order that all variables of VARIABLES take a distinct value.

Example

$(3, \langle 5, 1, 9, 1, 5, 5\rangle)$
$(1, \langle 5, 1, 9, 6, 5, 3\rangle)$
$(0, \langle 8, 1, 9, 6, 5, 3\rangle)$

Within the collection $\langle 5, 1, 9, 1, 5, 5\rangle$ of the first example, 3 and 2 items are respectively fixed to values 5 and 1. Therefore one must change the values of at least $(3 - 1) + (2 - 1) = 3$ items to get back to 6 distinct values. Consequently, the corresponding SOFT_ALLDIFFERENT_VAR constraint holds since its first argument C is greater than or equal to 3.

Typical

$\texttt{C} > 0$
$2 * \texttt{C} \leq |\texttt{VARIABLES}|$
$|\texttt{VARIABLES}| > 1$
SOME_EQUAL(VARIABLES)

Symmetries

- C can be increased.
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

Arg. properties Contractible wrt. VARIABLES.

Usage A soft ALLDIFFERENT constraint.

Remark Since it focus on the soft aspect of the ALLDIFFERENT constraint, the original article [325], which introduce this constraint, describes how to evaluate the minimum value of C and how to prune according to the maximum value of C.

The SOFT_ALLDIFFERENT_VAR constraint is called SOFT_ALLDIFF_MIN_VAR in [158].

Algorithm A first filtering algorithm presented in [325] achieves arc-consistency. A second filtering algorithm also achieving arc-consistency is described in [138, 139].

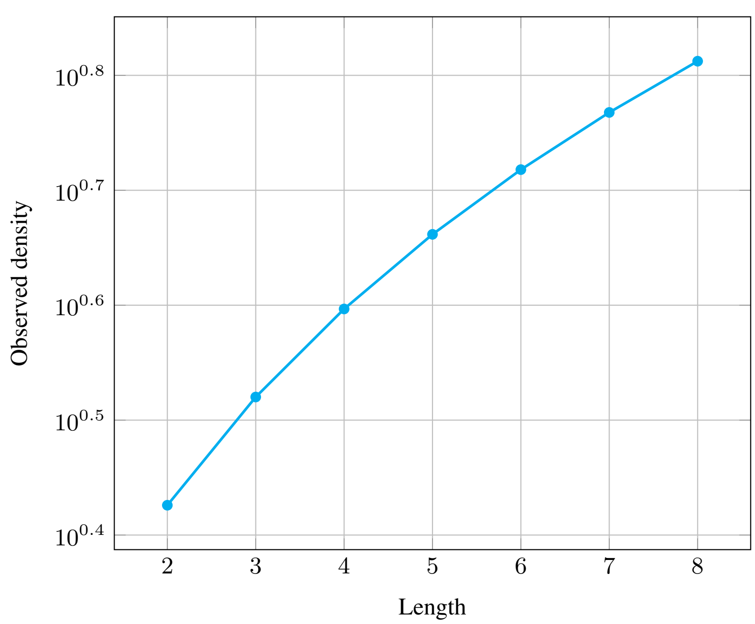
Reformulation By introducing a variable M that gives the number of distinct values used by variables of the collection VARIABLES, the SOFT_ALLDIFFERENT_VAR(C, VARIABLES) constraint can be expressed as a conjunction of the NVALUE(M, VARIABLES) constraint and of the linear constraint C $\geq$ |VARIABLES| $- M$.

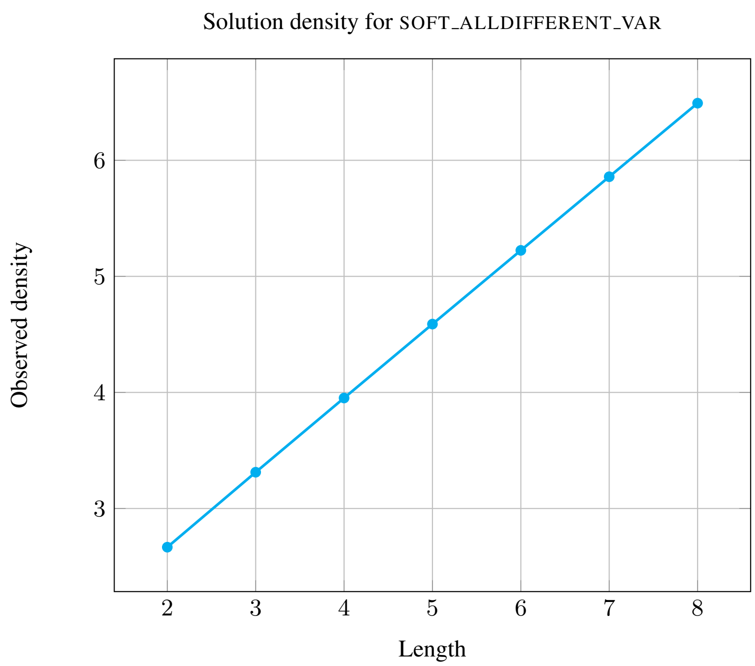
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	24	212	2470	35682	614600	12286024	279472266

Number of solutions for SOFT_ALLDIFFERENT_VAR: domains $0..n$

Solution density for SOFT_ALLDIFFERENT_VAR

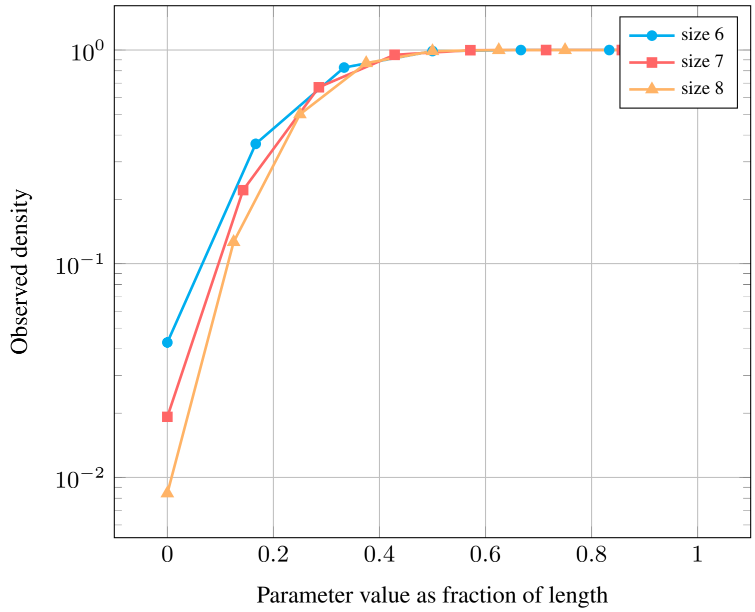




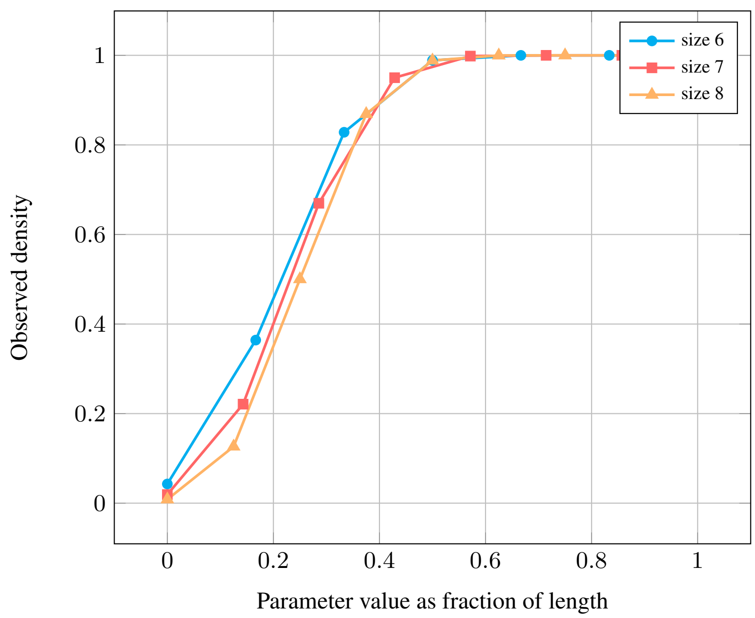
Length (n)		2	3	4	5	6	7	8
Total		24	212	2470	35682	614600	12286024	279472266
Parameter value	0	6	24	120	720	5040	40320	362880
	1	9	60	480	4320	42840	463680	5443200
	2	9	64	620	7320	97440	1404480	21530880
	3	-	64	625	7770	116340	1992480	37406880
	4	-	-	625	7776	117642	2093616	42550704
	5	-	-	-	7776	117649	2097144	43037568
	6	-	-	-	-	117649	2097152	43046712
	7	-	-	-	-	-	2097152	43046721
	8	-	-	-	-	-	-	43046721

Solution count for SOFT_ALLDIFFERENT_VAR: domains $0..n$

Solution density for SOFT_ALLDIFFERENT_VAR



Solution density for SOFT_ALLDIFFERENT_VAR



See also **common keyword:** SOFT_ALL_EQUAL_MAX_VAR, SOFT_ALL_EQUAL_MIN_CTR, SOFT_ALL_EQUAL_MIN_VAR, SOFT_ALLDIFFERENT_CTR, WEIGHTED_PARTIAL_ALLDIFF *(soft constraint)*.

hard version: ALLDIFFERENT.

implied by: ALL_MIN_DIST, ALLDIFFERENT_MODULO, SOFT_ALLDIFFERENT_CTR.

related: ATMOST_NVALUE, NVALUE.

Keywords

characteristic of a constraint: all different, disequality.

constraint type: soft constraint, value constraint, relaxation, variable-based violation measure.

filtering: bipartite matching.

final graph structure: strongly connected component, equivalence.

Arc input(s)	VARIABLES
Arc generator	*CLIQUE*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSCC**$\geq$ |VARIABLES| $-$ C

Graph model We generate a clique with binary *equalities* constraints between each pairs of vertices (this include an arc between a vertex and itself) and we state that C is equal to the difference between the total number of variables and the number of strongly connected components.

Parts (A) and (B) of Figure 5.744 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NSCC** graph property we show the different strongly connected components of the final graph. Each strongly connected component of the final graph includes all variables that take the same value. Since we have 6 variables and 3 strongly connected components the *cost* variable C is greater than or equal to $6 - 3$.

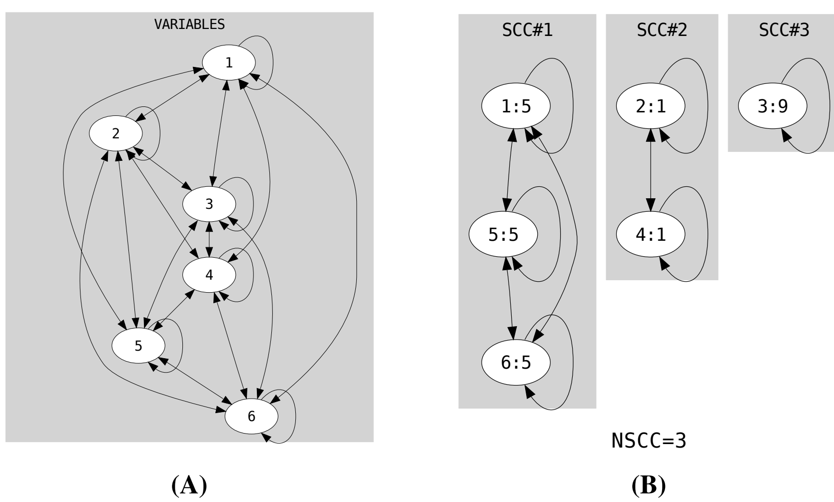


Figure 5.744: Initial and final graph of the SOFT_ALLDIFFERENT_VAR constraint

5.368 SOFT_CUMULATIVE

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Derived from CUMULATIVE

Constraint SOFT_CUMULATIVE(TASKS, LIMIT, INTERMEDIATE_LEVEL, SURFACE_ON_TOP)

Arguments

TASKS	:	collection(origin−dvar, duration−dvar, end−dvar, height−dvar)
LIMIT	:	int
INTERMEDIATE_LEVEL	:	int
SURFACE_ON_TOP	:	dvar

Restrictions

require_at_least(2, TASKS, [origin, duration, end])
required(TASKS, height)
TASKS.duration ≥ 0
TASKS.origin $\leq$ TASKS.end
TASKS.height ≥ 0
LIMIT ≥ 0
INTERMEDIATE_LEVEL ≥ 0
INTERMEDIATE_LEVEL $\leq$ LIMIT
SURFACE_ON_TOP ≥ 0

Purpose

Consider a set $\mathcal{T}$ of n tasks described by the TASKS collection, where $\texttt{origin}_j$, $\texttt{duration}_j$, $\texttt{end}_j$, $\texttt{height}_j$ are shortcuts for TASKS$[j]$.origin, TASKS$[j]$.duration, TASKS$[j]$.end, TASKS$[j]$.height. In addition let α and β respectively denote the earliest possible start over all tasks and the latest possible end over all tasks. The SOFT_CUMULATIVE constraint forces the three following conditions:

1. For each task TASKS$[j]$ $(1 \leq j \leq n)$ of $\mathcal{T}$ we have $\texttt{origin}_j + \texttt{duration}_j = \texttt{end}_j$.
2. At each point in time, the cumulated height of the set of tasks that overlap that point, does not exceed a given limit LIMIT (i.e., $\forall i \in [\alpha, \beta]$: $\sum_{j\in[1,n]|\texttt{origin}_j \leq i < \texttt{end}_j} \texttt{height}_j \leq \texttt{LIMIT}$).
3. The surface of the profile resource utilisation, which is greater than INTERMEDIATE_LEVEL, is equal to SURFACE_ON_TOP (i.e., $\sum_{i\in[\alpha,\beta]} \max(0, (\sum_{j\in[1,n]|\texttt{origin}_j \leq i < \texttt{end}_j} \texttt{height}_j) - \texttt{INTERMEDIATE\_LEVEL}) = \texttt{SURFACE\_ON\_TOP}$).

Example

$$\left(\left\langle\begin{array}{llll} \texttt{origin}-1 & \texttt{duration}-4 & \texttt{end}-5 & \texttt{height}-1, \\ \texttt{origin}-1 & \texttt{duration}-1 & \texttt{end}-2 & \texttt{height}-2, \\ \texttt{origin}-3 & \texttt{duration}-3 & \texttt{end}-6 & \texttt{height}-2 \end{array}\right\rangle, 3, 2, 3\right)$$

Figure 5.745 shows the cumulated profile associated with the example. To each task of the CUMULATIVE constraint corresponds a set of rectangles coloured with the same colour: the sum of the lengths of the rectangles corresponds to the duration of the task, while the height of the rectangles (i.e., all the rectangles associated with a task have the same height) corresponds to the resource consumption of the task. The SOFT_CUMULATIVE constraint holds since:

1. For each task we have that its end is equal to the sum of its origin and its duration.
2. At each point in time we do not have a cumulated resource consumption strictly greater than the upper limit LIMIT $= 3$ enforced by the second argument of the SOFT_CUMULATIVE constraint.
3. The surface of the cumulated profile located on top of the intermediate level INTERMEDIATE_LEVEL $= 2$ is equal to SURFACE_ON_TOP $= 3$.

TASKS

①	o − 1	d − 4	e − 5	h − 1
②	o − 1	d − 1	e − 2	h − 2
③	o − 3	d − 3	e − 6	h − 2

(o for origin, d for duration, e for end, h for height)

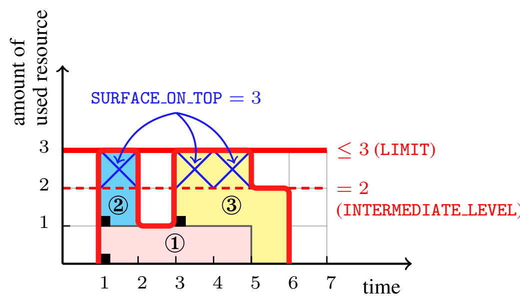


Figure 5.745: Resource consumption profile associated with the three tasks of the **Example** slot, where parts on top of the intermediate level 2 are marked by a cross

Typical

|TASKS| > 1
range(TASKS.origin) > 1
range(TASKS.duration) > 1
range(TASKS.end) > 1
range(TASKS.height) > 1
TASKS.duration > 0
TASKS.height > 0
LIMIT $<$ sum(TASKS.height)
INTERMEDIATE_LEVEL > 0
INTERMEDIATE_LEVEL $<$ LIMIT
SURFACE_ON_TOP > 0

Symmetries

- Items of TASKS are permutable.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.
- LIMIT can be increased.

Remark

The SOFT_CUMULATIVE constraint was initially introduced in **CHIP** [133] as a variant of the CUMULATIVE constraint. An extension of this constraint where one can restrict the surface on top of the intermediate level on different time intervals was first proposed in [322] and was generalised in [127].

See also **hard version:** CUMULATIVE.

Keywords **constraint type:** predefined constraint, soft constraint, scheduling constraint, resource constraint, temporal constraint, relaxation.

5.369 SOFT_SAME_INTERVAL_VAR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from SAME_INTERVAL

Constraint SOFT_SAME_INTERVAL_VAR(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)

Synonym SOFT_SAME_INTERVAL.

Arguments

C	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
SIZE_INTERVAL	:	int

Restrictions

$\texttt{C} \geq 0$
$\texttt{C} \leq |\texttt{VARIABLES1}|$
$|\texttt{VARIABLES1}| = |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\texttt{SIZE\_INTERVAL} > 0$

Purpose

Let N_i (respectively M_i) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\texttt{SIZE\_INTERVAL} \cdot i, \texttt{SIZE\_INTERVAL} \cdot i + \texttt{SIZE\_INTERVAL} - 1$. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all integer i we have $N_i = M_i$.

Example

$(4, \langle 9, 9, 9, 9, 9, 1\rangle, \langle 9, 1, 1, 1, 1, 8\rangle, 3)$

In the example, the fourth argument SIZE_INTERVAL = 3 defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Consequently the values of the collections $\langle 9, 9, 9, 9, 9, 1\rangle$ and $\langle 9, 1, 1, 1, 1, 8\rangle$ are respectively located within intervals $[9, 11]$, $[9, 11]$, $[9, 11]$, $[9, 11]$, $[9, 11]$, $[0, 2]$ and intervals $[9, 11]$, $[0, 2]$, $[0, 2]$, $[0, 2]$, $[0, 2]$, $[6, 8]$. Since there is a correspondence between two pairs of intervals we must unset at least $6 - 2$ items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). Consequently, the SOFT_SAME_INTERVAL_VAR constraint holds since its first argument C is set to $6 - 2$.

Typical

$\texttt{C} > 0$
$|\texttt{VARIABLES1}| > 1$
$\text{range}(\texttt{VARIABLES1.var}) > 1$
$\text{range}(\texttt{VARIABLES2.var}) > 1$
$\texttt{SIZE\_INTERVAL} > 1$
$\texttt{SIZE\_INTERVAL} < \text{range}(\texttt{VARIABLES1.var})$
$\texttt{SIZE\_INTERVAL} < \text{range}(\texttt{VARIABLES2.var})$

Symmetries

- Arguments are permutable w.r.t. permutation (C) (VARIABLES1, VARIABLES2) (SIZE_INTERVAL).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.
- An occurrence of a value of VARIABLES2.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Usage A soft SAME_INTERVAL constraint.

Algorithm See algorithm of the SOFT_SAME_VAR constraint.

See also **hard version:** SAME_INTERVAL.

implies: SOFT_USED_BY_INTERVAL_VAR.

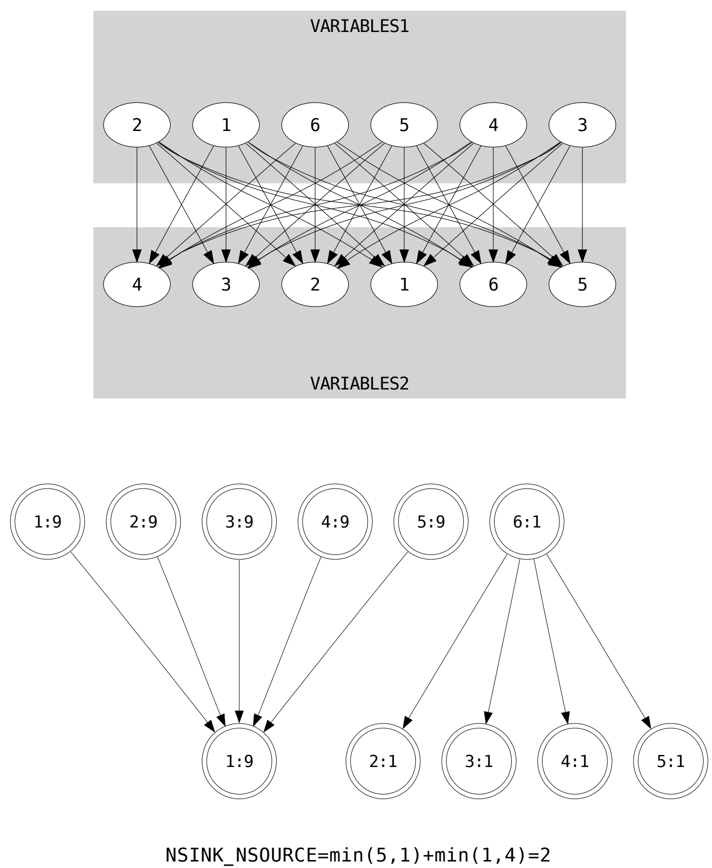
Keywords **constraint arguments:** constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.

modelling: interval.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL
Graph property(ies)	**NSINK_NSOURCE**= |VARIABLES1| − C

Graph model Parts (A) and (B) of Figure 5.746 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The SOFT_SAME_INTERVAL_VAR constraint holds since the cost 4 corresponds to the difference between the number of variables of VARIABLES1 and the sum over the different connected components of the minimum number of sources and sinks.



(A)

(B)

Figure 5.746: Initial and final graph of the SOFT_SAME_INTERVAL_VAR constraint

5.370 SOFT_SAME_MODULO_VAR

DESCRIPTION LINKS GRAPH

Origin

Derived from SAME_MODULO

Constraint

SOFT_SAME_MODULO_VAR(C, VARIABLES1, VARIABLES2, M)

Synonym

SOFT_SAME_MODULO.

Arguments

```
C          : dvar
VARIABLES1 : collection(var-dvar)
VARIABLES2 : collection(var-dvar)
M          : int
```

Restrictions

$\mathtt{C} \geq 0$
$\mathtt{C} \leq |\mathtt{VARIABLES1}|$
$|\mathtt{VARIABLES1}| = |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\mathtt{M} > 0$

Purpose

For each integer R in $[0, \mathtt{M}-1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have R as a rest when divided by M. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all R in $[0, \mathtt{M}-1]$ we have $N1_R = N2_R$.

Example

$(4, \langle 9, 9, 9, 9, 9, 1\rangle, \langle 9, 1, 1, 1, 1, 8\rangle, 3)$

In the example, the values of the collections $\langle 9, 9, 9, 9, 9, 1\rangle$ and $\langle 9, 1, 1, 1, 1, 8\rangle$ are respectively associated with the equivalence classes $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $1 \bmod 3 = 1$ and $9 \bmod 3 = 0$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $8 \bmod 3 = 2$. Since there is a correspondence between two pairs of equivalence classes we must unset at least $6 - 2$ items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). Consequently, the SOFT_SAME_MODULO_VAR constraint holds since its first argument C is set to $6 - 2$.

Typical

$\mathtt{C} > 0$
$|\mathtt{VARIABLES1}| > 1$
range(VARIABLES1.var) > 1
range(VARIABLES2.var) > 1
$\mathtt{M} > 1$
$\mathtt{M} <$ maxval(VARIABLES1.var)
$\mathtt{M} <$ maxval(VARIABLES2.var)

Symmetries

- Arguments are permutable w.r.t. permutation (C) (VARIABLES1, VARIABLES2) (M).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value u of VARIABLES1.var can be replaced by any other value v such that v is congruent to u modulo M.
- An occurrence of a value u of VARIABLES2.var can be replaced by any other value v such that v is congruent to u modulo M.

Usage A soft SAME_MODULO constraint.

Algorithm See algorithm of the SOFT_SAME_VAR constraint.

See also **hard version:** SAME_MODULO.

implies: SOFT_USED_BY_MODULO_VAR.

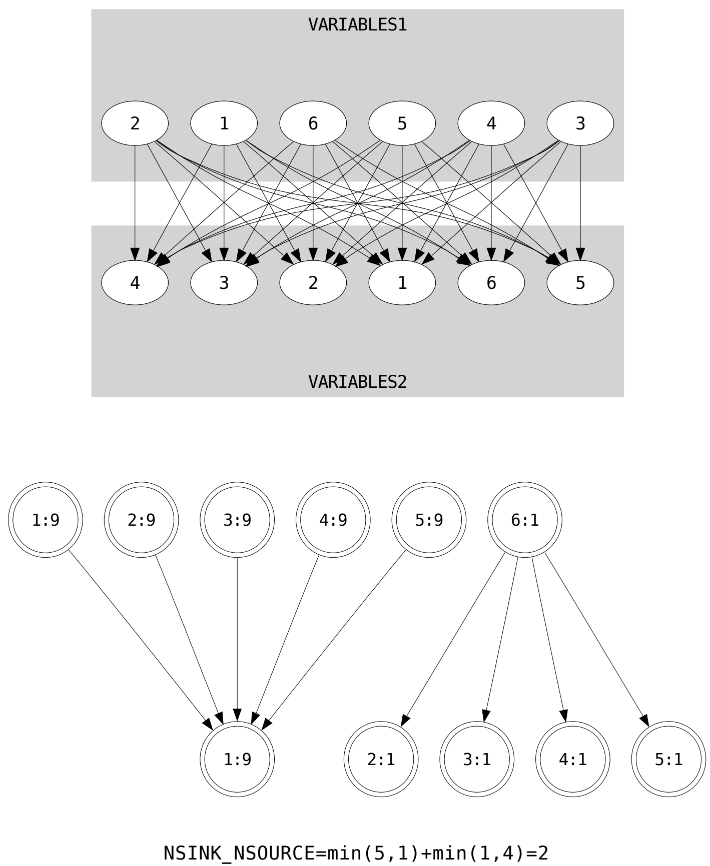
Keywords **characteristic of a constraint:** modulo.

constraint arguments: constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.

Arc input(s)	`VARIABLES1` `VARIABLES2`
Arc generator	*PRODUCT* $\mapsto$ `collection`(`variables1`, `variables2`)
Arc arity	2
Arc constraint(s)	`variables1.var` mod `M` = `variables2.var` mod `M`
Graph property(ies)	**NSINK_NSOURCE**= |`VARIABLES1`| − `C`

Graph model Parts (A) and (B) of Figure 5.747 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The SOFT_SAME_MODULO_VAR constraint holds since the cost 4 corresponds to the difference between the number of variables of `VARIABLES1` and the sum over the different connected components of the minimum number of sources and sinks.



(A)

(B)

Figure 5.747: Initial and final graph of the SOFT_SAME_MODULO_VAR constraint

5.371 SOFT_SAME_PARTITION_VAR

DESCRIPTION LINKS GRAPH

Origin Derived from SAME_PARTITION

Constraint SOFT_SAME_PARTITION_VAR(C, VARIABLES1, VARIABLES2, PARTITIONS)

Synonym SOFT_SAME_PARTITION.

Type

VALUES : collection(val−int)

Arguments

C : dvar
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

C ≥ 0
C $\leq$ |VARIABLES1|
|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
|PARTITIONS| ≥ 2
|VALUES| ≥ 1
required(VALUES, val)
distinct(VALUES, val)

Purpose

For each integer i in [1, |PARTITIONS|], let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that take their value in the i^{th} partition of the collection PARTITIONS. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all i in [1, |PARTITIONS|] we have $N1_i = N2_i$.

Example

$$\left(\begin{array}{l} 4, \langle 9,9,9,9,9,1\rangle, \\ \langle 9,1,1,1,1,8\rangle, \\ \langle \mathtt{p}-\langle 1,2\rangle, \mathtt{p}-\langle 9\rangle, \mathtt{p}-\langle 7,8\rangle\rangle \end{array}\right)$$

In the example, the values of the collections $\langle 9,9,9,9,9,1\rangle$ and $\langle 9,1,1,1,1,8\rangle$ are respectively associated with the partitions p − $\langle 9\rangle$, p − $\langle 9\rangle$, p − $\langle 9\rangle$, p − $\langle 9\rangle$, p − $\langle 9\rangle$, p − $\langle 1,2\rangle$ and p − $\langle 9\rangle$, p − $\langle 1,2\rangle$, p − $\langle 1,2\rangle$, p − $\langle 1,2\rangle$, p − $\langle 1,2\rangle$, p − $\langle 7,8\rangle$. Since there is a correspondence between two pairs of partitions we must unset at least $6-2$ items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). Consequently, the SOFT_SAME_PARTITION_VAR constraint holds since its first argument C is set to $6-2$.

Typical

$\mathtt{C} > 0$
$|\mathtt{VARIABLES1}| > 1$
$\mathtt{range}(\mathtt{VARIABLES1.var}) > 1$
$\mathtt{range}(\mathtt{VARIABLES2.var}) > 1$
$|\mathtt{VARIABLES1}| > |\mathtt{PARTITIONS}|$
$|\mathtt{VARIABLES2}| > |\mathtt{PARTITIONS}|$

Symmetries

- Arguments are permutable w.r.t. permutation (C) (VARIABLES1, VARIABLES2) (PARTITIONS).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Usage A soft SAME_PARTITION constraint.

Algorithm See algorithm of the SOFT_SAME_VAR constraint.

See also **hard version:** SAME_PARTITION.

implies: SOFT_USED_BY_PARTITION_VAR.

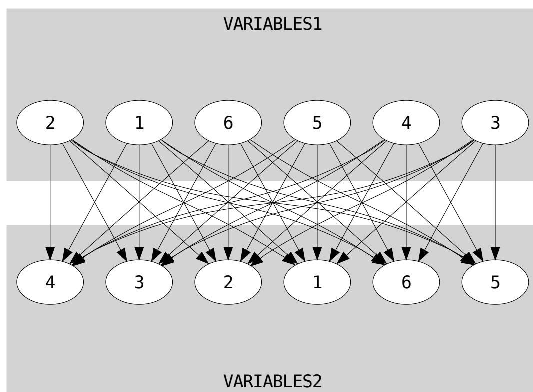
Keywords **characteristic of a constraint:** partition.

constraint arguments: constraint between two collections of variables.

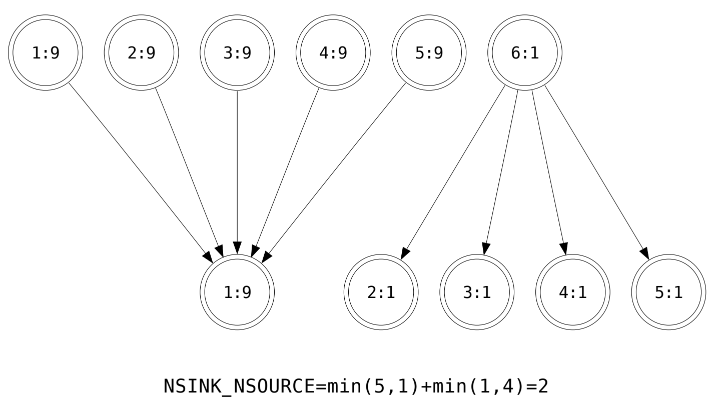
constraint type: soft constraint, relaxation, variable-based violation measure.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	**NSINK_NSOURCE**$=$ |VARIABLES1| $-$ C

Graph model Parts (A) and (B) of Figure 5.748 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The SOFT_SAME_PARTITION_VAR constraint holds since the cost 4 corresponds to the difference between the number of variables of VARIABLES1 and the sum over the different connected components of the minimum number of sources and sinks.



(A)



(B)

Figure 5.748: Initial and final graph of the SOFT_SAME_PARTITION_VAR constraint

5.372 SOFT_SAME_VAR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [434]

Constraint SOFT_SAME_VAR(C, VARIABLES1, VARIABLES2)

Synonym SOFT_SAME.

Arguments

C : dvar
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

Restrictions

$\texttt{C} \geq 0$
$\texttt{C} \leq |\texttt{VARIABLES1}|$
$|\texttt{VARIABLES1}| = |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that the variables of the VARIABLES2 collection correspond to the variables of the VARIABLES1 collection according to a permutation.

Example $(4, \langle 9, 9, 9, 9, 9, 1\rangle, \langle 9, 1, 1, 1, 1, 8\rangle)$

As illustrated by Figure 5.749, there is a correspondence between two pairs of values of the collections $\langle 9, 9, 9, 9, 9, 1\rangle$ and $\langle 9, 1, 1, 1, 1, 8\rangle$. Consequently, we must unset at least $6 - 2$ items (6 is the number of items of the VARIABLES1 and VARIABLES2 collections). The SOFT_SAME_VAR constraint holds since its first argument C is set to $6 - 2$.

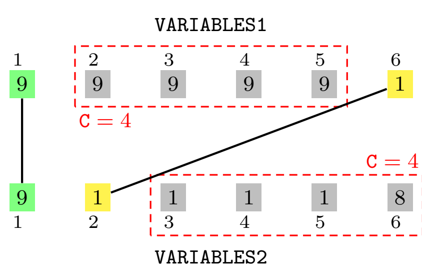


Figure 5.749: Illustration of the partial correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot, i.e., $\texttt{C} = 4$ items of the VARIABLES1 or of the VARIABLES2 collections need to be changed in order to have a full correspondence

Typical

$C > 0$
$|VARIABLES1| > 1$
$range(VARIABLES1.var) > 1$
$range(VARIABLES2.var) > 1$

Symmetries

- Arguments are permutable w.r.t. permutation (C) (VARIABLES1, VARIABLES2).
- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Usage

A soft SAME constraint.

Algorithm

A first filtering algorithm is described in [434, page 80]. A second filtering algorithm is presented in [138, 139].

See also

hard version: SAME.

implies: SOFT_USED_BY_VAR.

Keywords

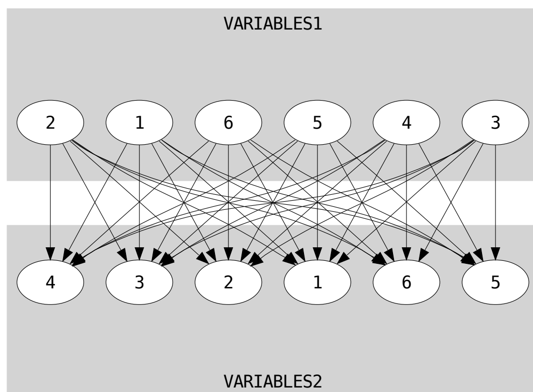
constraint arguments: constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.

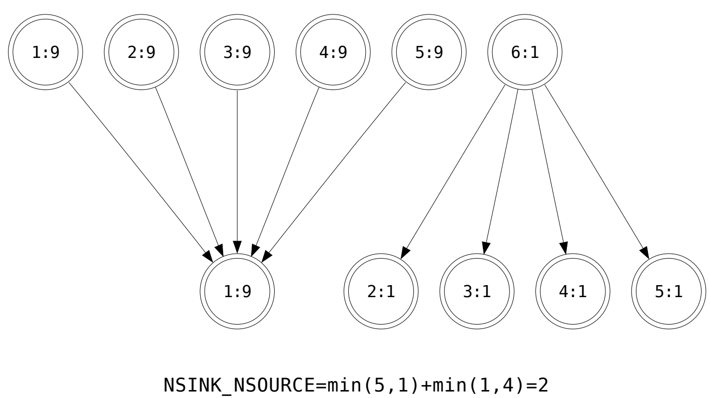
filtering: minimum cost flow, bipartite matching.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSINK_NSOURCE** = |VARIABLES1| − C

Graph model Parts (A) and (B) of Figure 5.750 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The SOFT_SAME_VAR constraint holds since the cost 4 corresponds to the difference between the number of variables of VARIABLES1 and the sum over the different connected components of the minimum number of sources and sinks.



(A)



(B)

Figure 5.750: Initial and final graph of the SOFT_SAME_VAR constraint

5.373 SOFT_USED_BY_INTERVAL_VAR

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY_INTERVAL.

Constraint SOFT_USED_BY_INTERVAL_VAR(C, VARIABLES1, VARIABLES2, SIZE_INTERVAL)

Synonym SOFT_USED_BY_INTERVAL.

Arguments

C	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
SIZE_INTERVAL	:	int

Restrictions

$\texttt{C} \geq 0$
$\texttt{C} \leq |\texttt{VARIABLES2}|$
$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\texttt{SIZE\_INTERVAL} > 0$

Purpose Let N_i (respectively M_i) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\texttt{SIZE\_INTERVAL} \cdot i, \texttt{SIZE\_INTERVAL} \cdot i + \texttt{SIZE\_INTERVAL} - 1]$. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all integer i we have $M_i > 0 \Rightarrow N_i \geq M_i$.

Example $(2, \langle 9, 1, 1, 8, 8 \rangle, \langle 9, 9, 9, 1 \rangle, 3)$

In the example, the fourth argument SIZE_INTERVAL $=$ 3 defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Consequently the values of the collections $\langle 9, 1, 1, 8, 8 \rangle$ and $\langle 9, 9, 9, 1 \rangle$ are respectively located within intervals $[9, 11]$, $[0, 2]$, $[0, 2]$, $[6, 8]$, $[6, 8]$ and intervals $[9, 11]$, $[9, 11]$, $[9, 11]$, $[0, 2]$. Since there is a correspondence between two pairs of intervals we must unset at least $4 - 2$ items (4 is the number of items of the VARIABLES2 collection). Consequently, the SOFT_USED_BY_INTERVAL_VAR constraint holds since its first argument C is set to $4 - 2$.

Typical

$\texttt{C} > 0$
$|\texttt{VARIABLES1}| > 1$
$|\texttt{VARIABLES2}| > 1$
range(VARIABLES1.var) > 1
range(VARIABLES2.var) > 1
$\texttt{SIZE\_INTERVAL} > 1$
SIZE_INTERVAL $<$range(VARIABLES1.var)
SIZE_INTERVAL $<$range(VARIABLES2.var)

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.
- An occurrence of a value of VARIABLES2.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Usage

A soft USED_BY_INTERVAL constraint.

See also

hard version: USED_BY_INTERVAL.

implied by: SOFT_SAME_INTERVAL_VAR.

Keywords

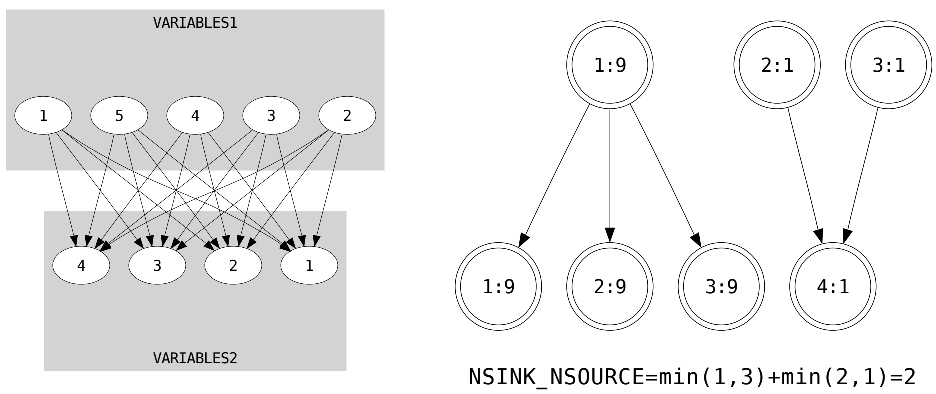
constraint arguments: constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.

modelling: interval.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var/SIZE_INTERVAL = variables2.var/SIZE_INTERVAL
Graph property(ies)	**NSINK_NSOURCE** = |VARIABLES2| − C

Graph model Parts (A) and (B) of Figure 5.751 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The SOFT_USED_BY_INTERVAL_VAR constraint holds since the cost 2 corresponds to the difference between the number of variables of VARIABLES2 and the sum over the different connected components of the minimum number of sources and sinks.



(A) **(B)**

Figure 5.751: Initial and final graph of the SOFT_USED_BY_INTERVAL_VAR constraint

5.374 SOFT_USED_BY_MODULO_VAR

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY_MODULO

Constraint SOFT_USED_BY_MODULO_VAR(C, VARIABLES1, VARIABLES2, M)

Synonym SOFT_USED_BY_MODULO.

Arguments

C	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
M	:	int

Restrictions

$C \geq 0$
$C \leq |VARIABLES2|$
$|VARIABLES1| \geq |VARIABLES2|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$M > 0$

Purpose

For each integer R in $[0, M-1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have R as a rest when divided by M. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all R in $[0, M-1]$ we have $N2_R > 0 \Rightarrow N1_R \geq N2_R$.

Example

$(2, \langle 9, 1, 1, 8, 8\rangle, \langle 9, 9, 9, 1\rangle, 3)$

In the example, the values of the collections $\langle 9, 1, 1, 8, 8\rangle$ and $\langle 9, 9, 9, 1\rangle$ are respectively associated with the equivalence classes $9 \bmod 3 = 0$, $1 \bmod 3 = 1$, $1 \bmod 3 = 1$, $8 \bmod 3 = 2$, $8 \bmod 3 = 2$ and $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $9 \bmod 3 = 0$, $1 \bmod 3 = 1$. Since there is a correspondence between two pairs of equivalence classes we must unset at least $4-2$ items (4 is the number of items of the VARIABLES2 collection). Consequently, the SOFT_USED_BY_MODULO_VAR constraint holds since its first argument C is set to $4-2$.

Typical

$C > 0$
$|VARIABLES1| > 1$
$|VARIABLES2| > 1$
range(VARIABLES1.var) > 1
range(VARIABLES2.var) > 1
$M > 1$
M $<$maxval(VARIABLES1.var)
M $<$maxval(VARIABLES2.var)

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value u of VARIABLES1.var can be replaced by any other value v such that v is congruent to u modulo M.
- An occurrence of a value u of VARIABLES2.var can be replaced by any other value v such that v is congruent to u modulo M.

Usage

A soft USED_BY_MODULO constraint.

See also

hard version: USED_BY_MODULO.

implied by: SOFT_SAME_MODULO_VAR.

Keywords

characteristic of a constraint: modulo.

constraint arguments: constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var mod M = variables2.var mod M
Graph property(ies)	**NSINK_NSOURCE** = |VARIABLES2| − C

Graph model Parts (A) and (B) of Figure 5.752 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The SOFT_USED_BY_MODULO_VAR constraint holds since the cost 2 corresponds to the difference between the number of variables of VARIABLES2 and the sum over the different connected components of the minimum number of sources and sinks.

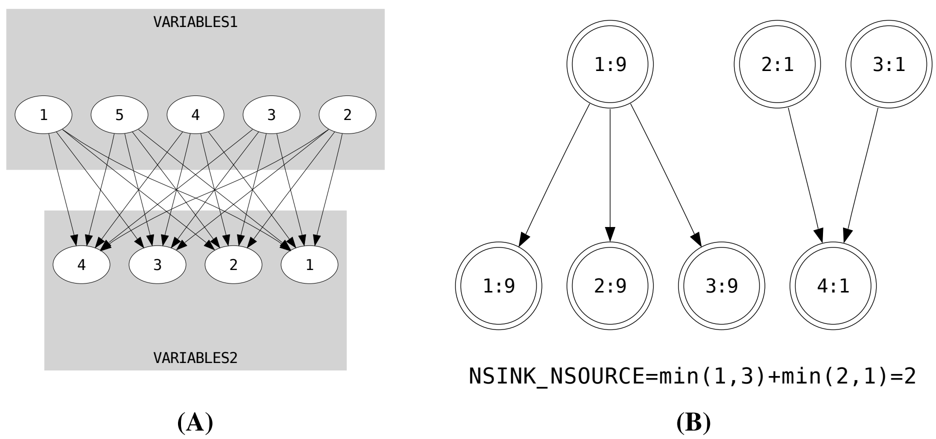


Figure 5.752: Initial and final graph of the SOFT_USED_BY_MODULO_VAR constraint

5.375 SOFT_USED_BY_PARTITION_VAR

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY_PARTITION.

Constraint SOFT_USED_BY_PARTITION_VAR(C, VARIABLES1, VARIABLES2, PARTITIONS)

Synonym SOFT_USED_BY_PARTITION.

Type

VALUES : collection(val−int)

Arguments

C : dvar
VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

$\texttt{C} \geq 0$
$\texttt{C} \leq |\texttt{VARIABLES2}|$
$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
$|\texttt{PARTITIONS}| \geq 2$
$|\texttt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)

Purpose

For each integer i in $[1, |\texttt{PARTITIONS}|]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that take their values in the i^{th} partition of the collection PARTITIONS. C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that for all i in $[1, |\texttt{PARTITIONS}|]$ we have $N2_i > 0 \Rightarrow N1_i \geq N2_i$.

Example

$$\left(\begin{array}{l} 2, \langle 9,1,1,8,8\rangle, \\ \langle 9,9,9,1\rangle, \\ \langle \texttt{p}-\langle 1,2\rangle, \texttt{p}-\langle 9\rangle, \texttt{p}-\langle 7,8\rangle\rangle \end{array}\right)$$

In the example, the values of the collections $\langle 9,1,1,8,8\rangle$ and $\langle 9,9,9,1\rangle$ are respectively associated with the partitions $\texttt{p}-\langle 9\rangle$, $\texttt{p}-\langle 1,2\rangle$, $\texttt{p}-\langle 1,2\rangle$, $\texttt{p}-\langle 7,8\rangle$, $\texttt{p}-\langle 7,8\rangle$ and $\texttt{p}-\langle 9\rangle$, $\texttt{p}-\langle 9\rangle$, $\texttt{p}-\langle 9\rangle$, $\texttt{p}-\langle 1,2\rangle$. Since there is a correspondence between two pairs of partitions we must unset at least $4-2$ items (4 is the number of items of the VARIABLES2 collection). Consequently, the SOFT_USED_BY_PARTITION_VAR constraint holds since its first argument C is set to $4-2$.

Typical

$\texttt{C} > 0$
$|\texttt{VARIABLES1}| > 1$
$|\texttt{VARIABLES2}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$|\texttt{VARIABLES1}| > |\texttt{PARTITIONS}|$
$|\texttt{VARIABLES2}| > |\texttt{PARTITIONS}|$

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Usage

A soft USED_BY_PARTITION constraint.

See also

hard version: USED_BY_PARTITION.

implied by: SOFT_SAME_PARTITION_VAR.

Keywords

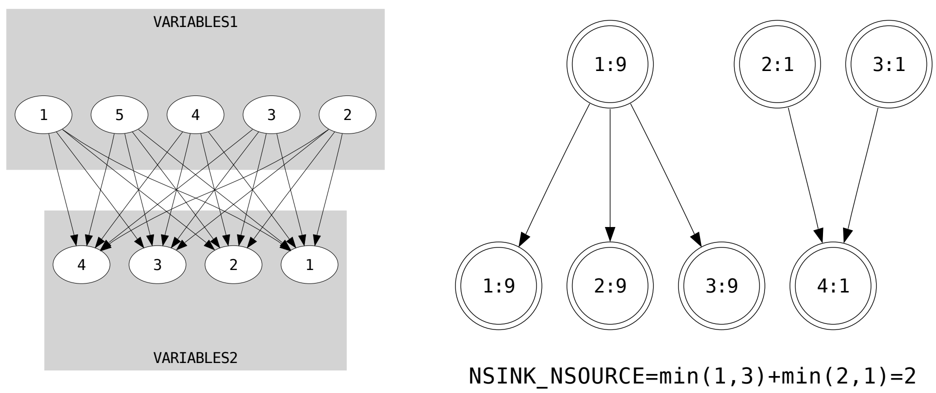
characteristic of a constraint: partition.

constraint arguments: constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT$ ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	**NSINK_NSOURCE** = |VARIABLES2| − C

Graph model Parts (A) and (B) of Figure 5.753 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The SOFT_USED_BY_PARTITION_VAR constraint holds since the cost 2 corresponds to the difference between the number of variables of VARIABLES2 and the sum over the different connected components of the minimum number of sources and sinks.



(A) **(B)**

Figure 5.753: Initial and final graph of the SOFT_USED_BY_PARTITION_VAR constraint

5.376 SOFT_USED_BY_VAR

DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY

Constraint SOFT_USED_BY_VAR(C, VARIABLES1, VARIABLES2)

Synonym SOFT_USED_BY.

Arguments

C	:	dvar
VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)

Restrictions

$\mathtt{C} \geq 0$
$\mathtt{C} \leq |\mathtt{VARIABLES2}|$
$|\mathtt{VARIABLES1}| \geq |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose C is the minimum number of values to change in the VARIABLES1 and VARIABLES2 collections so that all the values of the variables of collection VARIABLES2 are used by the variables of collection VARIABLES1.

Example $(2, \langle 9, 1, 1, 8, 8\rangle, \langle 9, 9, 9, 1\rangle)$

As illustrated by Figure 5.754, there is a correspondence between two pairs of values of the collections $\langle 9, 1, 1, 8, 8\rangle$ and $\langle 9, 9, 9, 1\rangle$. Consequently, we must unset at least $4 - 2$ items (4 is the number of items of the VARIABLES2 collection). The SOFT_USED_BY_VAR constraint holds since its first argument C is set to $4 - 2$.

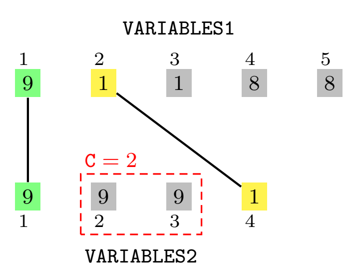


Figure 5.754: Illustration of the partial correspondence between the items of the VARIABLES2 $= \langle 9, 9, 9, 1\rangle$ and of the VARIABLES1 $= \langle 9, 1, 1, 8, 8\rangle$ collections of the **Example** slot, i.e., C $= 2$ items of the VARIABLES2 or of the VARIABLES1 collections need to be changed in order to cover all elements of VARIABLES2

Typical

$\mathtt{C} > 0$
$|\mathtt{VARIABLES1}| > 1$
$|\mathtt{VARIABLES2}| > 1$
$\mathtt{range}(\mathtt{VARIABLES1.var}) > 1$
$\mathtt{range}(\mathtt{VARIABLES2.var}) > 1$

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Usage

A soft USED_BY constraint.

Algorithm

A filtering algorithm achieving arc-consistency is described in [138, 139].

See also

hard version: USED_BY.

implied by: SOFT_SAME_VAR.

Keywords

constraint arguments: constraint between two collections of variables.

constraint type: soft constraint, relaxation, variable-based violation measure.

filtering: bipartite matching.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NSINK_NSOURCE** = |VARIABLES2| − C

Graph model Parts (A) and (B) of Figure 5.755 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK_NSOURCE** graph property, the source and sink vertices of the final graph are stressed with a double circle. The SOFT_USED_BY_VAR constraint holds since the cost 2 corresponds to the difference between the number of variables of VARIABLES2 and the sum over the different connected components of the minimum number of sources and sinks.

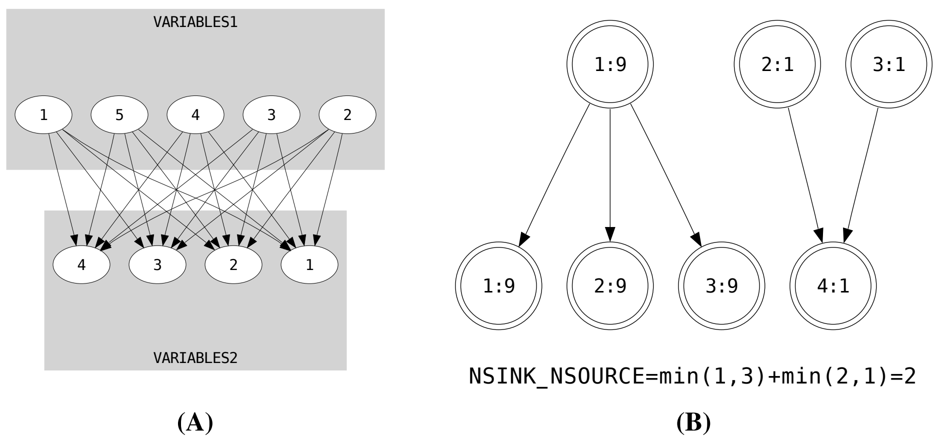


Figure 5.755: Initial and final graph of the SOFT_USED_BY_VAR constraint

5.377 SOME_EQUAL

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from ALLDIFFERENT

Constraint SOME_EQUAL(VARIABLES)

Synonyms SOME_EQ, NOT_ALLDIFFERENT, NOT_ALLDIFF, NOT_ALLDISTINCT, NOT_DISTINCT.

Argument VARIABLES : collection(var−dvar)

Restrictions
required(VARIABLES, var)
$|$VARIABLES$| > 1$

Purpose Enforce at least two variables of the collection VARIABLES to be assigned the same value.

Example $(\langle 1, 4, 1, 6\rangle)$

The SOME_EQUAL constraint holds since the first and the third variables are both assigned the same value 1.

Typical
$|$VARIABLES$| > 2$
nval(VARIABLES.var) > 2

Symmetries
- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped; all occurrences of a value of VARIABLES.var can be renamed to any unused value.

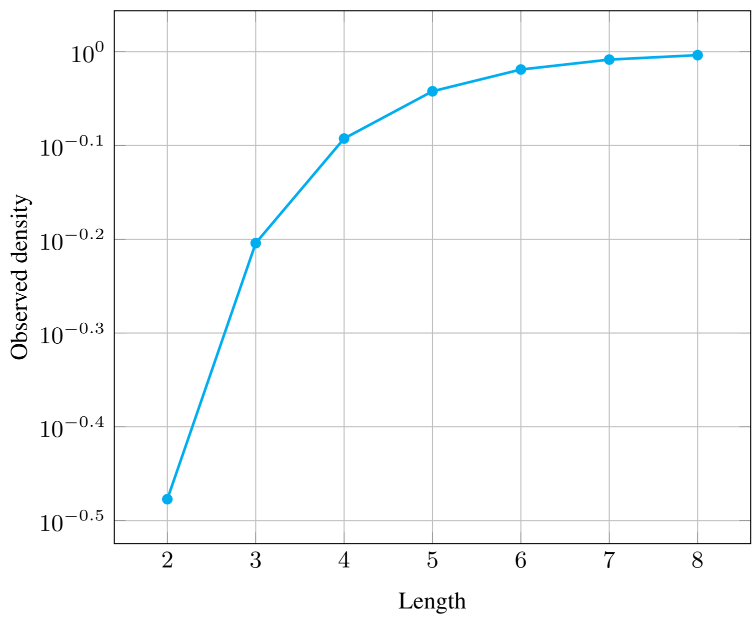
Arg. properties Extensible wrt. VARIABLES.

Counting

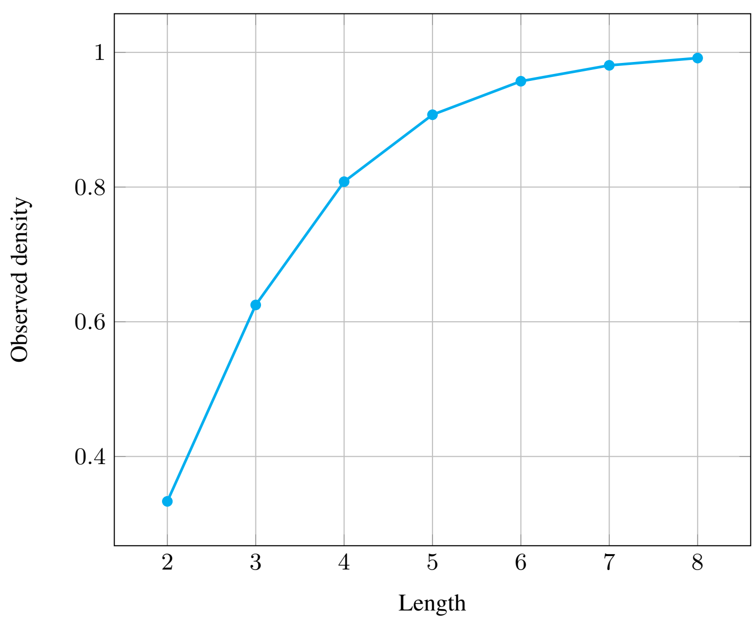
Length (n)	2	3	4	5	6	7	8
Solutions	3	40	505	7056	112609	2056832	42683841

Number of solutions for SOME_EQUAL: domains $0..n$

Solution density for SOME_EQUAL



Solution density for SOME_EQUAL



Used in SOFT_ALLDIFFERENT_VAR.

See also **negation:** ALLDIFFERENT.

Keywords

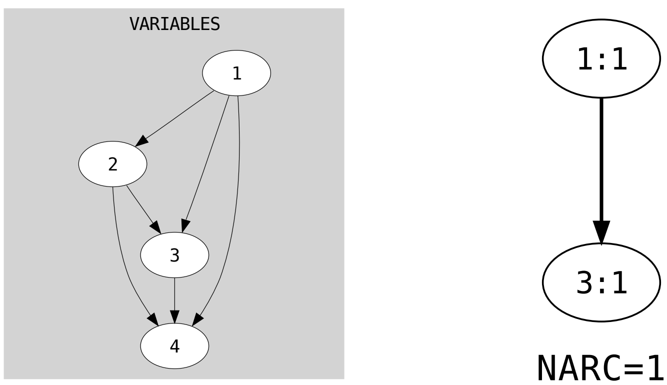
characteristic of a constraint: sort based reformulation.

constraint type: value constraint.

Arc input(s)	VARIABLES
Arc generator	$CLIQUE(<) \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	**NARC** > 0

Graph model We generate a *clique* with an equality constraint between each pair of distinct vertices and state that the number of arcs of the final graph should be strictly greater than 0.

Parts (A) and (B) of Figure 5.756 respectively show the initial and final graph associated with the **Example** slot. The SOME_EQUAL constraint holds since the final graph has at one arc, i.e. two variables are assigned the same value.



(A) **(B)**

Figure 5.756: Initial and final graph of the SOME_EQUAL constraint

5.378 SORT

DESCRIPTION LINKS GRAPH

Origin [308]

Constraint SORT(VARIABLES1, VARIABLES2)

Synonyms SORTEDNESS, SORTED, SORTING.

Arguments

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

Restrictions

|VARIABLES1| = |VARIABLES2|
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose First, the variables of the collection VARIABLES2 correspond to a permutation of the variables of VARIABLES1. Second, the variables of VARIABLES2 are sorted in increasing order.

Example $(\langle 1, 9, 1, 5, 2, 1\rangle, \langle 1, 1, 1, 2, 5, 9\rangle)$

The SORT constraint holds since:

- Values 1, 2, 5 and 9 have the same number of occurrences within both collections $\langle 1, 9, 1, 5, 2, 1\rangle$ and $\langle 1, 1, 1, 2, 5, 9\rangle$. Figure 5.757 illustrates this correspondence.
- The items of collection $\langle 1, 1, 1, 2, 5, 9\rangle$ are sorted in increasing order.

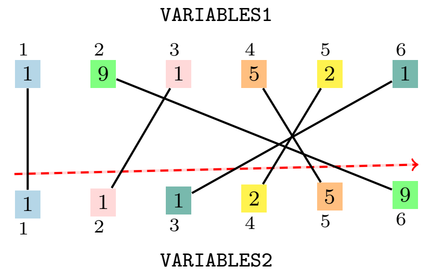


Figure 5.757: Illustration of the correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot (note that the items of the VARIABLES2 are sorted in increasing order)

All solutions Figure 5.758 gives all solutions to the following non ground instance of the SORT constraint: $V_1 \in [2,3]$, $V_2 \in [2,3]$, $V_3 \in [1,2]$, $V_4 \in [4,5]$, $V_5 \in [2,4]$, $S_1 \in [2,3]$, $S_2 \in [2,3]$, $S_3 \in [1,3]$, $S_4 \in [4,5]$, $S_5 \in [2,5]$, SORT$(\langle V_1, V_2, V_3, V_4, V_5\rangle, \langle S_1, S_2, S_3, S_4, S_5\rangle)$.

① $(\langle 2,2,2,4,4\rangle, \langle 2,2,2,4,4\rangle)$
② $(\langle 2,2,2,5,4\rangle, \langle 2,2,2,4,5\rangle)$
③ $(\langle 2,3,2,4,4\rangle, \langle 2,2,3,4,4\rangle)$
④ $(\langle 2,3,2,5,4\rangle, \langle 2,2,3,4,5\rangle)$
⑤ $(\langle 3,2,2,4,4\rangle, \langle 2,2,3,4,4\rangle)$
⑥ $(\langle 3,2,2,5,4\rangle, \langle 2,2,3,4,5\rangle)$
⑦ $(\langle 3,3,2,4,4\rangle, \langle 2,3,3,4,4\rangle)$
⑧ $(\langle 3,3,2,5,4\rangle, \langle 2,3,3,4,5\rangle)$

Figure 5.758: All solutions corresponding to the non ground example of the SORT constraint of the **All solutions** slot

Typical

$|$`VARIABLES1`$| > 1$
range(`VARIABLES1.var`) > 1

Symmetries

- Items of `VARIABLES1` are permutable.
- One and the same constant can be added to the `var` attributes of all items of `VARIABLES1` and `VARIABLES2`.

Arg. properties

Functional dependency: `VARIABLES2` determined by `VARIABLES1`.

Usage

The SORT constraint was initially introduced by Older *et al.* [308] for expressing disjunctive constraints in job-shop scheduling problems. It was also used by [22] for expressing disjunctive constraints in an air traffic control application. However the main usage of the SORT constraint, that was not foreseen when the SORT constraint was invented, is its use in many reformulations. Many constraints involving one or several collections of variables *become much simpler to express when the variables of these collections are sorted*. In addition these reformulations typically have a size that is linear in the number of variables of the original constraint. This justifies why the SORT constraint is considered to be a core constraint. As illustrative examples of these types of reformulations we successively consider the ALLDIFFERENT and the SAME constraints:

- The ALLDIFFERENT$(\langle v_1, v_2, \ldots, v_n\rangle)$ constraint can be reformulated as the conjunction SORT$(\langle v_1, v_2, \ldots, v_n\rangle, \langle w_1, w_2, \ldots, w_n\rangle)$ $\wedge$ STRICTLY_INCREASING$(\langle w_1, w_2, \ldots, w_n\rangle)$.
- The SAME$(\langle u_1, u_2, \ldots, u_n\rangle, \langle v_1, v_2, \ldots, v_n\rangle)$ constraint can be reformulated as the conjunction SORT$(\langle u_1, u_2, \ldots, u_n\rangle, \langle w_1, w_2, \ldots, w_n\rangle)$ $\wedge$ SORT$(\langle v_1, v_2, \ldots, v_n\rangle, \langle w_1, w_2, \ldots, w_n\rangle)$.

Remark

A variant of this constraint called SORT_PERMUTATION was introduced in [461]. In this variant an additional list of domain variables represents the permutation that allows one to go from `VARIABLES1` to `VARIABLES2`.

Algorithm

[85, 86, 292].

Systems SORTING in **Choco**, SORTED in **Gecode**, SORT in **MiniZinc**, SORTING in **SICStus**.

See also **generalisation:** SORT_PERMUTATION *(*PERMUTATION *parameter added)*.

implies: LEX_GREATEREQ, SAME.

uses in its reformulation: ALLDIFFERENT, SAME.

Keywords **application area:** air traffic management.

characteristic of a constraint: core, sort.

combinatorial object: permutation.

constraint arguments: constraint between two collections of variables, pure functional dependency.

filtering: bound-consistency.

modelling: functional dependency.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	• for all connected components: **NSOURCE**=**NSINK** • **NSOURCE**= |VARIABLES1| • **NSINK**= |VARIABLES2|

Arc input(s)	VARIABLES2
Arc generator	$PATH \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var $\leq$ variables2.var
Graph property(ies)	**NARC**= |VARIABLES2| $-$ 1

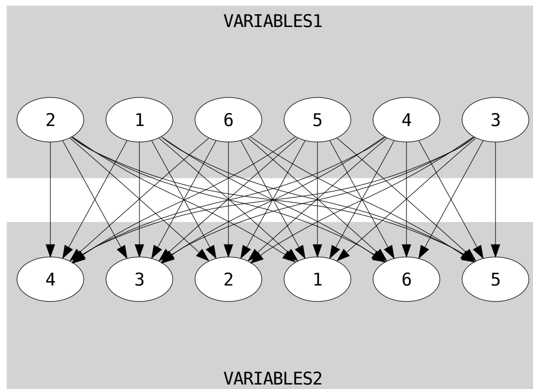
Graph model Parts (A) and (B) of Figure 5.759 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. Since it uses the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of this final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. The SORT constraint holds since:

- Each connected component of the final graph of the first graph constraint has the same number of sources and of sinks.
- The number of sources of the final graph of the first graph constraint is equal to |VARIABLES1|.
- The number of sinks of the final graph of the first graph constraint is equal to |VARIABLES2|.
- Finally the second graph constraint holds also since its corresponding final graph contains exactly |VARIABLES1 $-$ 1| arcs: all the inequalities constraints between consecutive variables of VARIABLES2 holds.

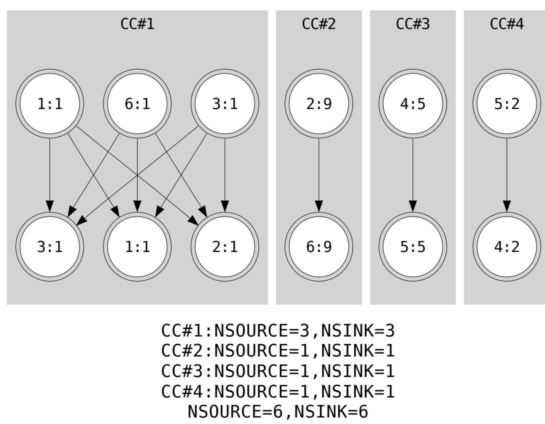
Signature Consider the first graph constraint. Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- Sources of the initial graph cannot become sinks of the final graph,
- Sinks of the initial graph cannot become sources of the final graph.

From the previous observations and since we use the $PRODUCT$ arc generator on the collections VARIABLES1 and VARIABLES2, we have that the maximum number of sources and sinks of the final graph is respectively equal to |VARIABLES1| and |VARIABLES2|. Therefore we can rewrite **NSOURCE** = |VARIABLES1| to **NSOURCE** $\geq$ |VARIABLES1| and simplify $\underline{\overline{\mathbf{NSOURCE}}}$ to $\overline{\mathbf{NSOURCE}}$. In a similar way, we can rewrite **NSINK** = |VARIABLES2| to **NSINK** $\geq$ |VARIABLES2| and simplify $\underline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.



(A)



(B)

Figure 5.759: Initial and final graph of the SORT constraint

Consider now the second graph constraint. Since we use the $PATH$ arc generator with an arity of 2 on the VARIABLES2 collection, the maximum number of arcs of the final graph is equal to $|\mathtt{VARIABLES2}| - 1$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\mathtt{VARIABLES2}| - 1$ to $\mathbf{NARC} \geq |\mathtt{VARIABLES2}| - 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Quiz

EXERCISE 1 (checking whether a ground instance holds or not)[a]

A. Does the constraint SORT$(\langle 1, 0, 0, 1\rangle, \langle 0, 0, 1\rangle)$ hold?

B. Does the constraint SORT$(\langle 3, 5, 3, 1\rangle, \langle 1, 3, 5\rangle)$ hold?

C. Does the constraint SORT$(\langle 2, 4, 2, 2, 4\rangle, \langle 2, 2, 2, 4, 4\rangle)$ hold?

D. Does the constraint SORT$(\langle 2, 4, 2, 2, 4\rangle, \langle 4, 4, 2, 2, 2\rangle)$ hold?

[a]Hint: go back to the definition of SORT.

EXERCISE 2 (finding all solutions)[a]

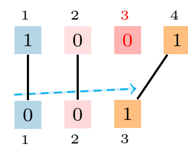
Give all the solutions to the constraint:

$$\left\{\begin{array}{lllll} X_1 \in [2,4], & X_2 \in [2,3], & X_3 \in [0,5], & X_4 \in [6,8], & X_5 \in [3,6], \\ Y_1 \in [3,4], & Y_2 \in [2,3], & Y_3 \in [0,5], & Y_4 \in [6,8], & Y_5 \in [3,6], \\ & & & & \\ \multicolumn{5}{l}{\textsc{sort}\left(\begin{array}{lllll} \langle X_1, & X_2, & X_3, & X_4, & X_5\rangle, \\ \langle Y_1, & Y_2, & Y_3, & Y_4, & Y_5\rangle \end{array}\right).} \end{array}\right.$$

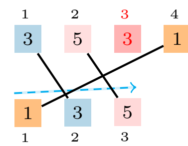
[a]Hint: first filter the bounds of the variables of the second argument with respect to the chain of precedences; second, since the second argument can be computed from the first one, focus on the variables of the first argument and enumerate solutions in lexicographic order.

SOLUTION TO EXERCISE 1

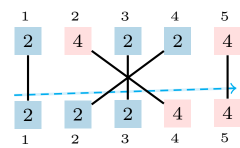
A. *No, since* $\langle 1, 0, 0, 1\rangle$ *and* $\langle 0, 0, 1\rangle$ *do not have the same number of elements.*



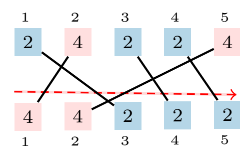
B. *No, since* $\langle 3, 5, 3, 1\rangle$ *and* $\langle 1, 3, 5\rangle$ *do not have the same number of elements.*



C. *Yes, since* $\langle 2, 2, 2, 4, 4\rangle$ *is a permutation of* $\langle 2, 4, 2, 2, 4\rangle$ *and since the elements* $2, 2, 2, 4, 4$ *are sorted in non-decreasing order.*



D. *No, since the elements of* $\langle 4, 4, 2, 2, 2\rangle$ *are not sorted in non-decreasing order.*



SOLUTION TO EXERCISE 2

the four solutions

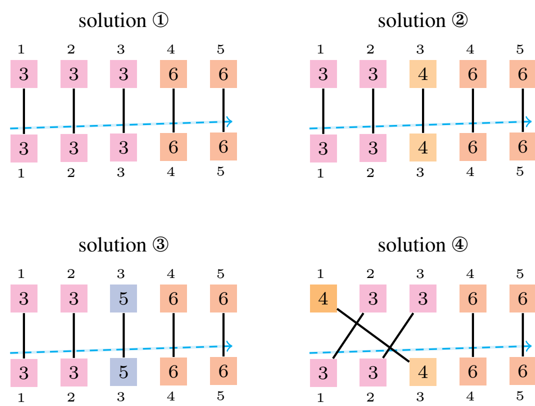
$\langle X_1, X_2, X_3, X_4, X_5 \rangle, \langle Y_1, Y_2, Y_3, Y_4, Y_5 \rangle$

① $(\langle 3, 3, 3, 6, 6 \rangle, \ \langle 3, 3, 3, 6, 6 \rangle)$

② $(\langle 3, 3, 4, 6, 6 \rangle, \ \langle 3, 3, 4, 6, 6 \rangle)$

③ $(\langle 3, 3, 5, 6, 6 \rangle, \ \langle 3, 3, 5, 6, 6 \rangle)$

④ $(\langle 4, 3, 3, 6, 6 \rangle, \ \langle 3, 3, 4, 6, 6 \rangle)$



5.379 SORT_PERMUTATION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [460]

Constraint SORT_PERMUTATION(FROM, PERMUTATION, TO)

Usual name SORT

Synonyms EXTENDED_SORTEDNESS, SORTEDNESS, SORTED, SORTING.

Arguments

FROM	:	collection(var−dvar)
PERMUTATION	:	collection(var−dvar)
TO	:	collection(var−dvar)

Restrictions

$|\text{PERMUTATION}| = |\text{FROM}|$
$|\text{PERMUTATION}| = |\text{TO}|$
PERMUTATION.var ≥ 1
PERMUTATION.var $\leq |\text{PERMUTATION}|$
ALLDIFFERENT(PERMUTATION)
required(FROM, var)
required(PERMUTATION, var)
required(TO, var)

Purpose The variables of collection FROM correspond to the variables of collection TO according to the permutation PERMUTATION (i.e., FROM$[i]$.var $=$ TO[PERMUTATION$[i]$.var].var). The variables of collection TO are also sorted in increasing order.

Example $(\langle 1,9,1,5,2,1\rangle, \langle 1,6,3,5,4,2\rangle, \langle 1,1,1,2,5,9\rangle)$

The SORT_PERMUTATION constraint holds since:

- – The first item FROM[1].var $=$ 1 of collection FROM corresponds to the PERMUTATION[1].var $= 1^{th}$ item of collection TO.
  – The second item FROM[2].var $=$ 9 of collection FROM corresponds to the PERMUTATION[2].var $= 6^{th}$ item of collection TO.
  – The third item FROM[3].var $=$ 1 of collection FROM corresponds to the PERMUTATION[3].var $= 3^{th}$ item of collection TO.
  – The fourth item FROM[4].var $=$ 5 of collection FROM corresponds to the PERMUTATION[4].var $= 5^{th}$ item of collection TO.
  – The fifth item FROM[5].var $=$ 2 of collection FROM corresponds to the PERMUTATION[5].var $= 4^{th}$ item of collection TO.
  – The sixth item FROM[6].var $=$ 1 of collection FROM corresponds to the PERMUTATION[6].var $= 2^{th}$ item of collection TO.
- The items of collection TO $= \langle 1,1,1,2,5,9\rangle$ are sorted in increasing order.

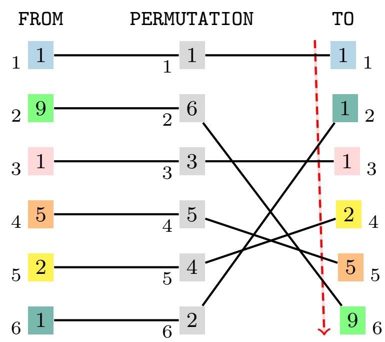


Figure 5.760: Illustration of the correspondence between the items of the FROM and the TO collections according to the permutation defined by the items of the PERMUTATION collection of the **Example** slot (note that the items of the TO collection are sorted in increasing order)

Typical

$|\texttt{FROM}| > 1$
$\texttt{range}(\texttt{FROM.var}) > 1$
LEX_DIFFERENT(FROM, TO)

Symmetry

One and the same constant can be added to the var attributes of all items of FROM and TO.

Arg. properties

Functional dependency: TO determined by FROM.

Remark

Observe that the argument PERMUTATION of the SORT_PERMUTATION constraint is not completely determined by FROM and TO when the items of the collection FROM are not all distinct. In other words this means that even though all items of the FROM and TO are completely fixed, *the permutation will not be completely determined* when some items of FROM are assigned the same value.

The SORT_PERMUTATION constraint is referenced under the name SORTING in **SICStus Prolog**.

Algorithm

[461].

Reformulation

Let n denote the number of variables in the collection FROM. The SORT_PERMUTATION constraint can be reformulated as a conjunction of the form:

ELEMENT(PERMUTATION[1], FROM, TO[1]),
ELEMENT(PERMUTATION[2], FROM, TO[2]),
...
ELEMENT(PERMUTATION[n], FROM, TO[n]),
ALLDIFFERENT(PERMUTATION),
INCREASING(TO).

To enhance the previous model, the following necessary condition was proposed by P. Schaus. $\forall i \in [1, n] : \sum_{j=1}^{j=n} (\texttt{FROM}[j] < \texttt{TO}[i]) \leq i-1$ (i.e., at most $i-1$ variables of the collection FROM are assigned a value strictly less than TO[i]). Similarly, we have that $\forall i \in [1, n] : \sum_{j=1}^{j=n} (\mathtt{FROM}[j] > \mathtt{TO}[i]) \geq n - i$ (i.e., at most $n - i$ variables of the collection FROM are assigned a value are strictly greater than TO$[i]$).

Systems SORTED in **Gecode**, SORTING in **SICStus**.

See also **common keyword:** ORDER *(sort, permutation)*.

implies: CORRESPONDENCE.

specialisation: SORT *(*PERMUTATION *parameter removed)*.

used in reformulation: ALLDIFFERENT, ELEMENT, INCREASING.

Keywords **characteristic of a constraint:** sort, derived collection.

combinatorial object: permutation.

constraint arguments: constraint between three collections of variables.

modelling: functional dependency.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{FROM\_PERMUTATION}-\texttt{collection}(\texttt{var}-\texttt{dvar},\texttt{ind}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{FROM.var},\texttt{ind}-\texttt{PERMUTATION.var})]\end{array}\right)$$

Arc input(s) FROM_PERMUTATION TO

Arc generator $PRODUCT \mapsto \texttt{collection}(\texttt{from\_permutation},\texttt{to})$

Arc arity 2

Arc constraint(s)
- $\texttt{from\_permutation.var} = \texttt{to.var}$
- $\texttt{from\_permutation.ind} = \texttt{to.key}$

Graph property(ies) $\mathbf{NARC} = |\texttt{PERMUTATION}|$

Arc input(s) TO

Arc generator $PATH \mapsto \texttt{collection}(\texttt{to1},\texttt{to2})$

Arc arity 2

Arc constraint(s) $\texttt{to1.var} \leq \texttt{to2.var}$

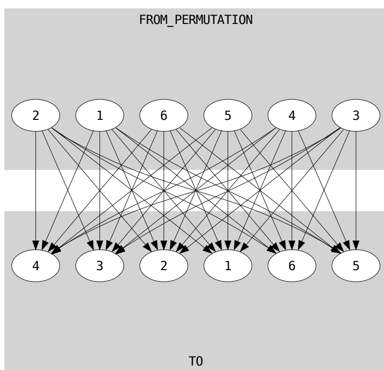
Graph property(ies) $\mathbf{NARC} = |\texttt{TO}| - 1$

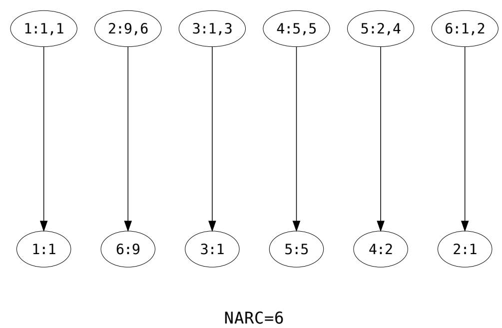
Graph model Parts (A) and (B) of Figure 5.761 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. In both graphs the source vertices correspond to the items of the derived collection FROM_PERMUTATION, while the sink vertices correspond to the items of the TO collection. Since the first graph constraint uses the $\mathbf{NARC}$ graph property, the arcs of its final graph are stressed in bold. The SORT_PERMUTATION constraint holds since:

- The first graph constraint holds since its final graph contains exactly PERMUTATION arcs.
- Finally the second graph constraint holds also since its corresponding final graph contains exactly $|\texttt{PERMUTATION} - 1|$ arcs: all the inequalities constraints between consecutive variables of TO holds.

Signature Consider the first graph constraint where we use the $PRODUCT$ arc generator. Since all the key attributes of the TO collection are distinct, and because of the second condition $\texttt{from\_permutation.ind} = \texttt{to.key}$ of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to $|\texttt{PERMUTATION}|$. So we can rewrite the graph property $\mathbf{NARC} = |\texttt{PERMUTATION}|$ to $\mathbf{NARC} \geq |\texttt{PERMUTATION}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Consider now the second graph constraint. Since we use the $PATH$ arc generator with an arity of 2 on the TO collection, the maximum number of arcs of the corresponding final graph is equal to $|\texttt{TO}| - 1$. Therefore we can rewrite $\mathbf{NARC} = |\texttt{TO}| - 1$ to $\mathbf{NARC} \geq |\texttt{TO}| - 1$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.





(A) **(B)**

Figure 5.761: Initial and final graph of the SORT_PERMUTATION constraint

5.380 STABLE_COMPATIBILITY

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin P. Flener, [49]

Constraint STABLE_COMPATIBILITY(NODES)

Argument

$$\texttt{NODES} : \texttt{collection}\begin{pmatrix}\texttt{index}-\texttt{int},\\ \texttt{father}-\texttt{dvar},\\ \texttt{prec}-\texttt{sint},\\ \texttt{inc}-\texttt{sint}\end{pmatrix}$$

Restrictions

required(NODES, [index, father, prec, inc])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.father ≥ 1
NODES.father $\leq$ |NODES|
NODES.prec ≥ 1
NODES.prec $\leq$ |NODES|
NODES.inc ≥ 1
NODES.inc $\leq$ |NODES|
NODES.inc $>$ NODES.index

Purpose

Enforce the construction of a *stably compatible* supertree that is compatible with several given trees. The notion of stable compatibility and its context are detailed in the **Usage** slot.

Example

$$\left\langle\begin{pmatrix}
\texttt{ind}-1 & \texttt{f}-4 & \texttt{p}-\{11,12\} & \texttt{inc}-\emptyset,\\
\texttt{ind}-2 & \texttt{f}-3 & \texttt{p}-\{8,9\} & \texttt{inc}-\emptyset,\\
\texttt{ind}-3 & \texttt{f}-4 & \texttt{p}-\{2,10\} & \texttt{inc}-\emptyset,\\
\texttt{ind}-4 & \texttt{f}-5 & \texttt{p}-\{1,3\} & \texttt{inc}-\emptyset,\\
\texttt{ind}-5 & \texttt{f}-7 & \texttt{p}-\{4,13\} & \texttt{inc}-\emptyset,\\
\texttt{ind}-6 & \texttt{f}-2 & \texttt{p}-\{8,14\} & \texttt{inc}-\emptyset,\\
\texttt{ind}-7 & \texttt{f}-7 & \texttt{p}-\{6,13\} & \texttt{inc}-\emptyset,\\
\texttt{ind}-8 & \texttt{f}-6 & \texttt{p}-\emptyset & \texttt{inc}-\{9,10,11,12,13,14\},\\
\texttt{ind}-9 & \texttt{f}-2 & \texttt{p}-\emptyset & \texttt{inc}-\{10,11,12,13\},\\
\texttt{ind}-10 & \texttt{f}-3 & \texttt{p}-\emptyset & \texttt{inc}-\{11,12,13\},\\
\texttt{ind}-11 & \texttt{f}-1 & \texttt{p}-\emptyset & \texttt{inc}-\{12,13\},\\
\texttt{ind}-12 & \texttt{f}-1 & \texttt{p}-\emptyset & \texttt{inc}-\{13\},\\
\texttt{ind}-13 & \texttt{f}-5 & \texttt{p}-\emptyset & \texttt{inc}-\{14\},\\
\texttt{ind}-14 & \texttt{f}-6 & \texttt{p}-\emptyset & \texttt{inc}-\emptyset
\end{pmatrix}\right\rangle$$

(ind for index, f for father, p for prec)

Figure 5.762 shows the two trees we want to merge. Note that the leaves a and f occur in both trees.

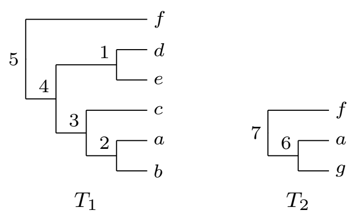


Figure 5.762: The two trees to merge

The left part of Figure 5.763 gives one way to merge the two previous trees. This solution corresponds to the ground instance provided by the example. Note that there exist 7 other ways to merge these two trees. They are respectively depicted by Figures 5.763 to 5.766.

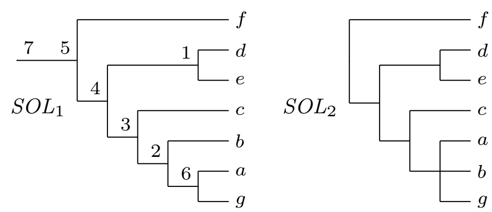


Figure 5.763: First solution (corresponding to the ground instance of the example) and second solution on how to merge the two trees T_1 and T_2 of Figure 5.762

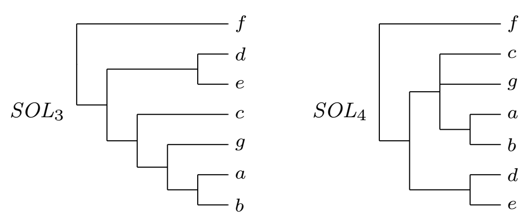


Figure 5.764: Third and fourth solutions on how to merge the two trees T_1 and T_2 of Figure 5.762

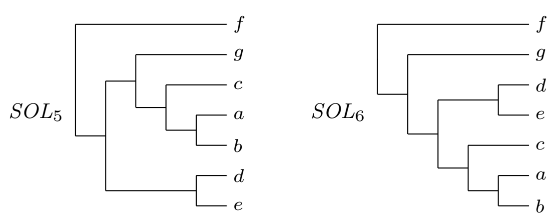


Figure 5.765: Fifth and sixth solutions on how to merge the two trees T_1 and T_2 of Figure 5.762

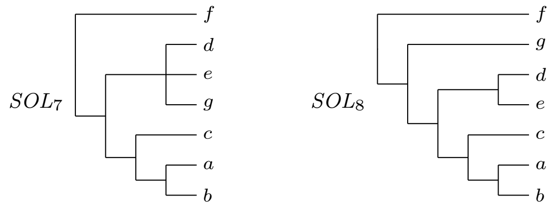


Figure 5.766: Seventh and eight solutions on how to merge the two trees T_1 and T_2 of Figure 5.762

Typical

$|$`NODES`$| > 2$
`range`(`NODES.father`) > 1

Symmetry

Items of `NODES` are permutable.

Usage

One objective of phylogeny is to construct the genealogy of the species, called the *tree of life*, whose leaves represent the contemporary species and whose internal nodes represent extinct species that are not necessarily named. An important problem in phylogeny is the construction of a supertree [83] that is compatible with several given trees. There are several definitions of tree compatibility in the literature:

- A tree $\mathcal{T}$ is *strongly compatible* with a tree $\mathcal{T}'$ if $\mathcal{T}'$ is topologically equivalent to a subtree $\mathcal{T}$ that respects the node labelling. [305]
- A tree $\mathcal{T}$ is *weakly compatible* with a tree $\mathcal{T}'$ if $\mathcal{T}'$ can be obtained from $\mathcal{T}$ by a series of arc contractions. [409]
- A tree $\mathcal{T}$ is *stably compatible* with a set $\mathcal{S}$ of trees if $\mathcal{T}$ is weakly compatible with each tree in $\mathcal{S}$ and each internal node of $\mathcal{T}$ can be labelled by at least one corresponding internal node of some tree in $\mathcal{S}$.

For the supertree problem, strong and weak compatibility coincide if and only if all the given trees are binary [305]. The existence of solutions is not lost when restricting weak compatibility to stable compatibility.

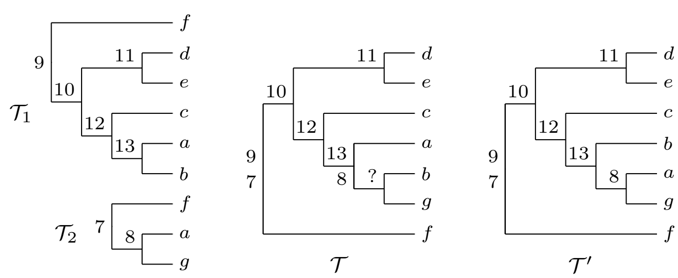


Figure 5.767: Supertree problem instance and two of its solutions

For example, the trees $\mathcal{T}_1$ and $\mathcal{T}_2$ of Figure 5.767 have $\mathcal{T}$ and $\mathcal{T}'$ as supertrees under both weak and strong compatibility. As shown, all the internal nodes of $\mathcal{T}'$ can be labelled by

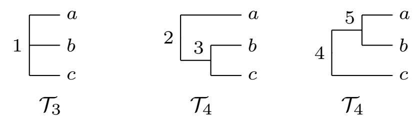


Figure 5.768: Three small phylogenetic trees

corresponding internal nodes of the two given trees, but this is not the case for the father of b and g in $\mathcal{T}$. Hence $\mathcal{T}$ and four other such supertrees are debatable because they speculate about the existence of extinct species that were not in any of the given trees. Consider also the three small trees in Figure 5.768: $\mathcal{T}_3$ and $\mathcal{T}_4$ have $\mathcal{T}_4$ as a supertree under weak compatibility, as it suffices to contract the arc $(3,2)$ to get $\mathcal{T}_3$ from $\mathcal{T}_4$. However, $\mathcal{T}_3$ and $\mathcal{T}_4$ have no supertree under strong compatibility, as the most recent common ancestor of b and c, denoted by $mrca(b,c)$, is the same as $mrca(a,b)$ in $\mathcal{T}_3$, namely 1, but not the same in $\mathcal{T}_4$, as $mrca(b,c) = 3$ is an evolutionary descendant of $mrca(a,b) = 2$. Also, $\mathcal{T}_4$ and $\mathcal{T}_5$ have neither weakly nor strongly compatible supertrees.

Under strong compatibility, a first supertree algorithm was given in [4], with an application for database management systems; it takes $O(l^2)$ time, where l is the number of leaves in the given trees. Derived algorithms have emerged from phylogeny, for example, *One-Tree* [305]. The first constraint program was proposed in [202], using standard, non-global constraints. Under weak compatibility, a phylogenetic supertree algorithm can be found in [409], for example. Under stable compatibility, the algorithm from computational linguistics of [87] has supertree construction as special case.

See also **root concept:** TREE.

Keywords **application area:** bioinformatics, phylogeny.

constraint type: graph constraint.

final graph structure: tree.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.father = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NCC** $= 1$ • **MAX_ID** ≤ 2 • **PATH_FROM_TO**(index, index, prec) $= 1$ • **PATH_FROM_TO**(index, index, inc) $= 0$ • **PATH_FROM_TO**(index, inc, index) $= 0$

Graph model To each distinct leave (i.e., each species) of the trees to merge corresponds a vertex of the initial graph. To each internal vertex of the trees to merge corresponds also a vertex of the initial graph. Each vertex of the initial graph has the following attributes:

- An *index* corresponding to a unique identifier.
- A *father* corresponding to the father of the vertex in the final tree. Since the leaves of the trees to merge must remain leaves we remove the index value of all the leaves from all the father variables.
- A *set of precedence constraints* corresponding to all the arcs of the trees to merge.
- A *set of incomparability constraints* corresponding to the incomparable vertices of each tree to merge.

The arc constraint describes the fact that we link a vertex to its father variable. Finally we use the following six graph properties on our final graph:

- The first graph property **MAX_NSCC** ≤ 1 enforces the fact that the size of the largest strongly connected component does not exceed one. This avoid having circuits containing more than one vertex. In fact the root of the merged tree is a strongly connected component with a single vertex.
- The second graph property **NCC** $= 1$ imposes having only a single tree.
- The third graph property **PATH_FROM_TO**(index, index, prec) $= 1$ enforces for each vertex i a set of precedence constraints; for each vertex j of the precedence set there is a path from i to j in the final graph.
- The fourth graph property **MAX_ID** ≤ 2 enforces that the number of predecessors (i.e., arcs from a vertex to itself are not counted) of each vertex does not exceed 2 (i.e., the final graph is a binary tree).
- The fifth and sixth graph properties **PATH_FROM_TO**(index, index, inc) $=$ 0 and **PATH_FROM_TO**(index, inc, index) $= 0$ enforces for each vertex i a set of incomparability constraints; for each vertex j of the incomparability set there is neither a path from i to j, nor a path from j to i.

Figures 5.769 and 5.770 respectively show the precedence and the incomparability graphs associated with the **Example** slot. As it contains too many arcs the initial graph is not shown. Figures 5.763 shows the first solution satisfying all the precedence and incomparability constraints.

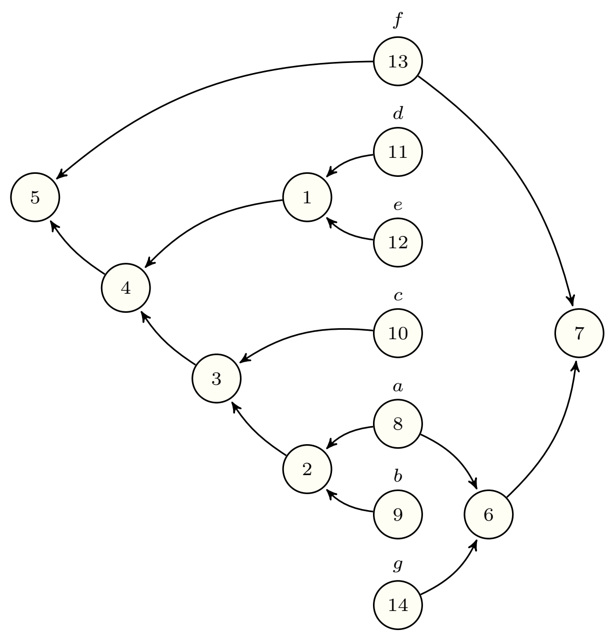


Figure 5.769: Precedence graph associated with the two trees to merge described by Figure 5.762

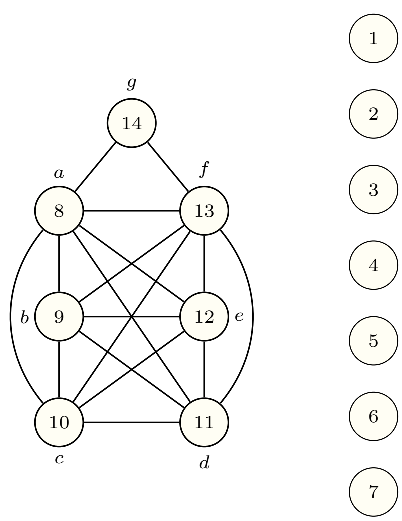


Figure 5.770: Incomparability graph associated with the two trees to merge described by Figure 5.762; the two cliques respectively correspond to the leaves of the two trees to merge.

5.381 STAGE_ELEMENT

DESCRIPTION LINKS GRAPH AUTOMATON

Origin Choco, derived from ELEMENT.

Constraint STAGE_ELEMENT(ITEM, TABLE)

Usual name STAGE_ELT

Synonym STAGE_ELEM.

Arguments

ITEM : collection(index−dvar, value−dvar)
TABLE : collection(low−int, up−int, value−int)

Restrictions

required(ITEM, [index, value])
$|\text{ITEM}| = 1$
$|\text{TABLE}| > 0$
required(TABLE, [low, up, value])
TABLE.low $\leq$ TABLE.up
increasing_seq(TABLE, [low])

Purpose

Let low_i, up_i and value_i respectively denote the values of the low, up and value attributes of the i^{th} item of the TABLE collection. First we have that: $\text{low}_i \leq \text{up}_i$ and $\text{up}_i + 1 = \text{low}_{i+1}$.
Second, the STAGE_ELEMENT constraint forces the following equivalence:
$\text{low}_i \leq \text{ITEM.index} \wedge \text{ITEM.index} \leq \text{up}_i \Leftrightarrow \text{ITEM.value} = \text{value}_i$.

Example

$$\left(\begin{array}{l}\langle \text{index}-5\ \text{value}-6\rangle,\\ \left\langle\begin{array}{lll}\text{low}-3 & \text{up}-7 & \text{value}-6,\\ \text{low}-8 & \text{up}-8 & \text{value}-8,\\ \text{low}-9 & \text{up}-14 & \text{value}-2,\\ \text{low}-15 & \text{up}-19 & \text{value}-9\end{array}\right\rangle\end{array}\right)$$

Figure 5.771 depicts the function associated with the items of the TABLE collection. The STAGE_ELEMENT constraint holds since:

- The value of ITEM[1].index is located between the values of the low and up attributes of the first item of the TABLE collection (i.e., $5 \in [3, 7]$).
- The value of ITEM[1].value corresponds to the value attribute of the first item of the TABLE collection (i.e., 6).

Typical

$|\text{TABLE}| > 1$
range(TABLE.value) > 1
TABLE.low $<$ TABLE.up

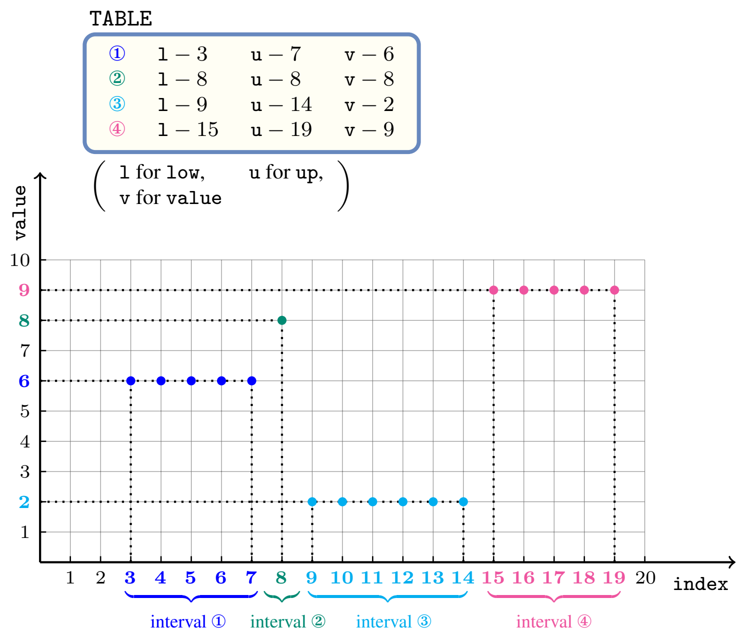


Figure 5.771: Function defined on four intervals ①, ②, ③ and ④ associated with the TABLE collection of the **Example** slot for linking the index and value attributes of the ITEM collection

Symmetry

All occurrences of two distinct values in ITEM.value or TABLE.value can be swapped; all occurrences of a value in ITEM.value or TABLE.value can be renamed to any unused value.

Arg. properties

- Functional dependency: ITEM.value determined by ITEM.index and TABLE.
- Suffix-extensible wrt. TABLE.

See also

common keyword: ELEM, ELEMENT *(data constraint)*.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint arguments: binary constraint, pure functional dependency.

constraint network structure: centered cyclic(2) constraint network(1).

constraint type: data constraint.

filtering: arc-consistency.

modelling: table, functional dependency.

Arc input(s)	TABLE
Arc generator	$PATH \mapsto$ collection(table1, table2)
Arc arity	2
Arc constraint(s)	• table1.low $\leq$ table1.up • table1.up $+ 1 =$ table2.low • table2.low $\leq$ table2.up
Graph property(ies)	**NARC** $= \vert$TABLE$\vert - 1$

Arc input(s)	ITEM TABLE
Arc generator	$PRODUCT \mapsto$ collection(item, table)
Arc arity	2
Arc constraint(s)	• item.index $\geq$ table.low • item.index $\leq$ table.up • item.value $=$ table.value
Graph property(ies)	**NARC** $= 1$

Graph model The first graph constraint models the restrictions on the low and up attributes of the TABLE collection, while the second graph constraint is similar to the one used for defining the ELEMENT constraint.

Parts (A) and (B) of Figure 5.772 respectively show the initial and final graph associated with the second graph constraint of the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold.

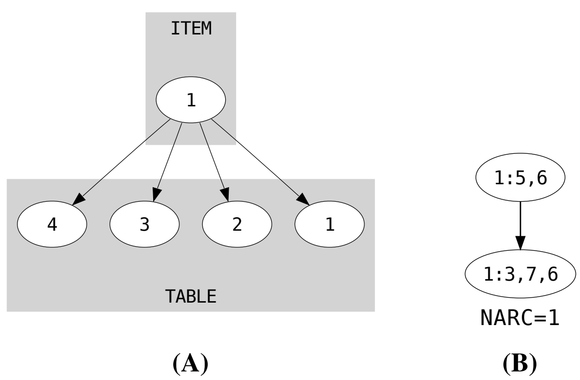


Figure 5.772: Initial and final graph of the STAGE_ELEMENT constraint

Automaton Figure 5.773 depicts the automaton associated with the STAGE_ELEMENT constraint. Let `INDEX` and `VALUE` respectively be the `index` and the `value` attributes of the unique item of the `ITEM` collection. Let $\texttt{LOW}_i$, $\texttt{UP}_i$ and $\texttt{VALUE}_i$ respectively be the `low`, the `up` and the `value` attributes of the i^{th} item of the `TABLE` collection. To each quintuple $(\texttt{INDEX}, \texttt{VALUE}, \texttt{LOW}_i, \texttt{UP}_i, \texttt{VALUE}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((\texttt{LOW}_i \leq \texttt{INDEX}) \wedge (\texttt{INDEX} \leq \texttt{UP}_i) \wedge (\texttt{VALUE} = \texttt{VALUE}_i)) \Leftrightarrow S_i$.

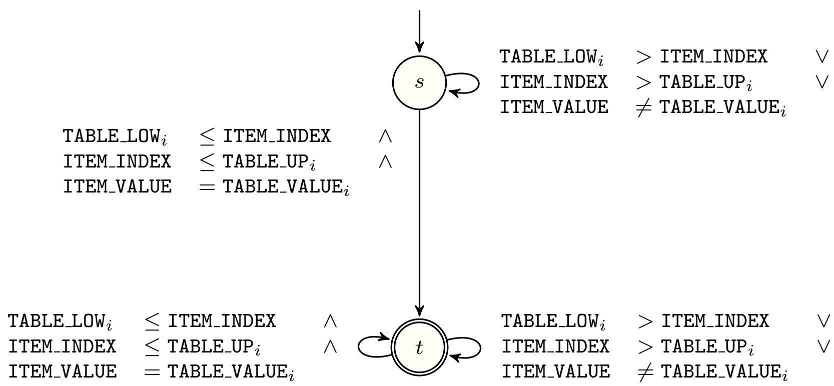


Figure 5.773: Automaton of the STAGE_ELEMENT constraint

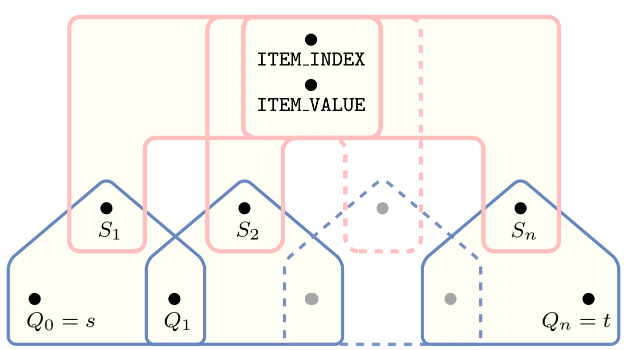


Figure 5.774: Hypergraph of the reformulation corresponding to the automaton of the STAGE_ELEMENT constraint

5.382 STRETCH_CIRCUIT

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [316]

Constraint STRETCH_CIRCUIT(VARIABLES, VALUES)

Usual name STRETCH

Arguments

VARIABLES : collection(var−dvar)
VALUES : collection(val−int, lmin−int, lmax−int)

Restrictions

|VARIABLES| > 0
required(VARIABLES, var)
|VALUES| > 0
required(VALUES, [val, lmin, lmax])
distinct(VALUES, val)
VALUES.lmin ≤ VALUES.lmax
VALUES.lmin ≤ |VARIABLES|
sum(VALUES.lmin) ≤ |VARIABLES|

Purpose

In order to define the meaning of the STRETCH_PATH constraint, we first introduce the notions of *stretch* and *span*. Let n be the number of variables of the collection VARIABLES and let i, j $(0 \leq i < n, 0 \leq j < n)$ be two positions within the collection of variables VARIABLES such that the following conditions apply:

- If $i \leq j$ then all variables $X_i, \ldots, X_j$ take a same value from the set of values of the val attribute.

  If $i > j$ then all variables $X_i, \ldots, X_{n-1}, X_0, \ldots, X_j$ take a same value from the set of values of the val attribute.
- $X_{(i-1) \bmod n}$ is different from X_i.
- $X_{(j+1) \bmod n}$ is different from X_j.

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $1 + (j - i) \bmod n$, while the *value* of the stretch is X_i. We now define the condition enforced by the STRETCH_CIRCUIT constraint.

Each item $(\mathtt{val} - v, \mathtt{lmin} - s, \mathtt{lmax} - t)$ of the VALUES collection enforces the minimum value s as well as the maximum value t for the span of a stretch of value v.

Note that:

1. Having an item $(\mathtt{val} - v, \mathtt{lmin} - s, \mathtt{lmax} - t)$ with s strictly greater than 0 does not mean that value v should be assigned to one of the variables of collection VARIABLES. It rather means that, when value v is used, all stretches of value v must have a span that belong to interval $[s, t]$.
2. A variable of the collection VARIABLES may be assigned a value that is not defined in the VALUES collection.

Example

$$\left(\begin{array}{l}\langle 6,6,3,1,1,1,6,6\rangle, \\ \left\langle\begin{array}{lll}\texttt{val}-1 & \texttt{lmin}-2 & \texttt{lmax}-4, \\ \texttt{val}-2 & \texttt{lmin}-2 & \texttt{lmax}-3, \\ \texttt{val}-3 & \texttt{lmin}-1 & \texttt{lmax}-6, \\ \texttt{val}-6 & \texttt{lmin}-2 & \texttt{lmax}-4\end{array}\right\rangle\end{array}\right)$$

The STRETCH_CIRCUIT constraint holds since the sequence $6\ 6\ 3\ 1\ 1\ 1\ 6\ 6$ contains three stretches $6\ 6\ 6\ 6$, 3, and $1\ 1\ 1$ respectively verifying the following conditions:

- The span of the first stretch $6\ 6\ 6\ 6$ is located within interval $[2,4]$ (i.e., the limit associated with value 6).
- The span of the second stretch 3 is located within interval $[1,6]$ (i.e., the limit associated with value 3).
- The span of the third stretch $1\ 1\ 1$ is located within interval $[2,4]$ (i.e., the limit associated with value 1).

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$
$|\texttt{VALUES}| > 1$
$\texttt{VALUES.lmax} \leq |\texttt{VARIABLES}|$

Symmetries

- Items of VARIABLES can be shifted.
- Items of VALUES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Usage

The article [316], which originally introduced the STRETCH constraint, quotes rostering problems as typical examples of use of this constraint.

Remark

We split the origin STRETCH constraint into the STRETCH_CIRCUIT and the STRETCH_PATH constraints that respectively use the *PATH LOOP* and *CIRCUIT LOOP* arc generators. We also reorganise the parameters: the VALUES collection describes the attributes of each value that can be assigned to the variables of the STRETCH_CIRCUIT constraint. Finally we skipped the pattern constraint that tells what values can follow a given value.

Algorithm

A first filtering algorithm was described in the original article of G. Pesant [316]. An algorithm that also generates explanations is given in [371]. The first filtering algorithm achieving arc-consistency is depicted in [219, 220]. This algorithm is based on dynamic programming and handles the fact that some values can be followed by only a given subset of values.

Reformulation

The STRETCH_CIRCUIT constraint can be reformulated in term of a STRETCH_PATH constraint. Let $LMAX$ denote the maximum value taken by the lmax attribute within the items of the collection VALUES, let n be the number of variables of the collection VARIABLES, and let $\delta = \min(LMAX, n)$. The first and second arguments of the STRETCH_PATH constraint are created in the following way:

- We pass to the STRETCH_PATH the variables of the collection VARIABLES to which we add the δ first variables of the collection VARIABLES.
- We pass to the STRETCH_PATH the values of the collection VALUES with the following modification: to each value v for which the corresponding lmax attribute is greater than or equal to n we reset its value to $n + \delta$.

Even if STRETCH_PATH can achieve arc-consistency this reformulation may not achieve arc-consistency since it duplicates variables.

Using this reformulation, the example

$$\text{STRETCH\_CIRCUIT}(\langle 6,6,3,1,1,1,6,6\rangle, \langle \texttt{val}-1\ \texttt{lmin}-2\ \texttt{lmax}-4,\ \texttt{val}-2\ \texttt{lmin}-2\ \texttt{lmax}-3,\ \texttt{val}-3\ \texttt{lmin}-1\ \texttt{lmax}-6,\ \texttt{val}-6\ \texttt{lmin}-2\ \texttt{lmax}-4\rangle)$$

of the **Example** slot is reformulated as:

$$\text{STRETCH\_PATH}(\langle 6,6,3,1,1,1,6,6,6,6,3,1,1,1\rangle, \langle \texttt{val}-1\ \texttt{lmin}-2\ \texttt{lmax}-4,\ \texttt{val}-2\ \texttt{lmin}-2\ \texttt{lmax}-3,\ \texttt{val}-3\ \texttt{lmin}-1\ \texttt{lmax}-6,\ \texttt{val}-6\ \texttt{lmin}-2\ \texttt{lmax}-4\rangle)$$

In the reformulation δ was equal to 6, and the VALUES collection was left unchanged since no lmax attribute was equal to the number of variables of the VARIABLES collection (i.e., 8).

See also

common keyword: GROUP *(timetabling constraint)*, PATTERN *(sliding sequence constraint,timetabling constraint)*, SLIDING_DISTRIBUTION *(sliding sequence constraint)*, STRETCH_PATH *(sliding sequence constraint,timetabling constraint)*.

used in reformulation: STRETCH_PATH.

Keywords

characteristic of a constraint: cyclic.

constraint type: timetabling constraint, sliding sequence constraint.

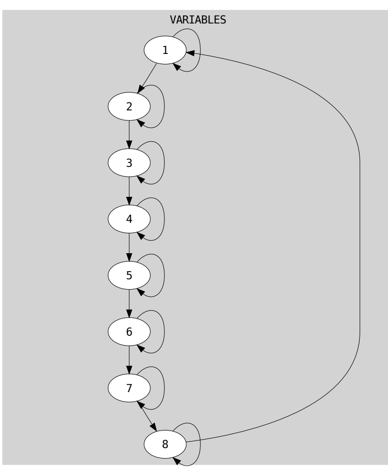
filtering: dynamic programming, arc-consistency, duplicated variables.

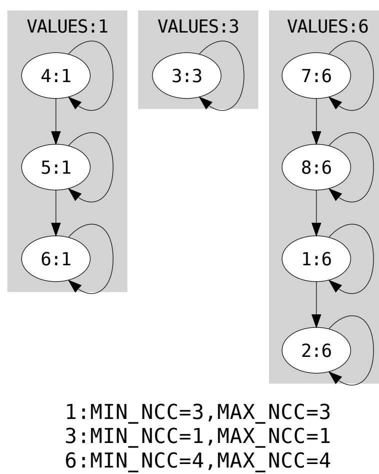
For all items of VALUES:

Arc input(s)	VARIABLES
Arc generator	*CIRCUIT*$\mapsto$collection(variables1, variables2) *LOOP*$\mapsto$collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	• variables1.var = VALUES.val • variables2.var = VALUES.val
Graph property(ies)	• NOT_IN(**MIN_NCC**, 1, VALUES.lmin − 1) • **MAX_NCC**$\leq$ VALUES.lmax

Graph model Part (A) of Figure 5.775 shows the initial graphs associated with values 1, 2, 3 and 6 of the **Example** slot. Part (B) of Figure 5.775 shows the corresponding final graphs associated with values 1, 3 and 6. Since value 2 is not assigned to any variable of the VARIABLES collection the final graph associated with value 2 is empty. The STRETCH_CIRCUIT constraint holds since:

- For value 1 we have one connected component for which the number of vertices is greater than or equal to 2 and less than or equal to 4,
- For value 2 we do not have any connected component,
- For value 3 we have one connected component for which the number of vertices is greater than or equal to 1 and less than or equal to 6,
- For value 6 we have one connected component for which the number of vertices is greater than or equal to 2 and less than or equal to 4.





(A) **(B)**

Figure 5.775: Initial and final graph of the STRETCH_CIRCUIT constraint

5.383 STRETCH_PATH

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin [316]

Constraint STRETCH_PATH(VARIABLES, VALUES)

Usual name STRETCH

Arguments

VARIABLES : collection(var−dvar)
VALUES : collection(val−int, lmin−int, lmax−int)

Restrictions

|VARIABLES| > 0
required(VARIABLES, var)
|VALUES| > 0
required(VALUES, [val, lmin, lmax])
distinct(VALUES, val)
VALUES.lmin ≥ 0
VALUES.lmin ≤ VALUES.lmax
VALUES.lmin ≤ |VARIABLES|

Purpose

In order to define the meaning of the STRETCH_PATH constraint, we first introduce the notions of *stretch* and *span*. Let n be the number of variables of the collection VARIABLES. Let $X_i, \ldots, X_j$ $(1 \leq i \leq j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ take a same value from the set of values of the val attribute,
- $i = 1$ or X_{i-1} is different from X_i,
- $j = n$ or X_{j+1} is different from X_j.

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $j - i + 1$, while the *value* of the stretch is X_i. We now define the condition enforced by the STRETCH_PATH constraint.

Each item (val $- v$, lmin $- s$, lmax $- t$) of the VALUES collection enforces the minimum value s as well as the maximum value t for the span of a stretch of value v over consecutive variables of the VARIABLES collection.

Note that:

1. Having an item (val $- v$, lmin $- s$, lmax $- t$) with s strictly greater than 0 does not mean that value v should be assigned to one of the variables of collection VARIABLES. It rather means that, when value v is used, all stretches of value v must have a span that belong to interval $[s, t]$.
2. A variable of the collection VARIABLES may be assigned a value that is not defined in the VALUES collection.

Example

$$\left(\begin{array}{l}\langle 6,6,3,1,1,1,6,6\rangle, \\ \left\langle\begin{array}{lll}\texttt{val}-1 & \texttt{lmin}-2 & \texttt{lmax}-4, \\ \texttt{val}-2 & \texttt{lmin}-2 & \texttt{lmax}-3, \\ \texttt{val}-3 & \texttt{lmin}-1 & \texttt{lmax}-6, \\ \texttt{val}-6 & \texttt{lmin}-2 & \texttt{lmax}-2\end{array}\right\rangle\end{array}\right)$$

The STRETCH_PATH constraint holds since the sequence 6 6 3 1 1 1 6 6 contains four stretches 6 6, 3, 1 1 1, and 6 6 respectively verifying the following conditions:

- The span of the first stretch 6 6 is located within interval $[2, 2]$ (i.e., the limit associated with value 6).
- The span of the second stretch 3 is located within interval $[1, 6]$ (i.e., the limit associated with value 3).
- The span of the third stretch 1 1 1 is located within interval $[2, 4]$ (i.e., the limit associated with value 1).
- The span of the fourth stretch 6 6 is located within interval $[2, 2]$ (i.e., the limit associated with value 6).

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VARIABLES}| > |\texttt{VALUES}|$
$|\texttt{VALUES}| > 1$
$\texttt{sum}(\texttt{VALUES.lmin}) \leq |\texttt{VARIABLES}|$
$\texttt{VALUES.lmax} \leq |\texttt{VARIABLES}|$

Symmetries

- Items of VARIABLES can be reversed.
- Items of VALUES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Usage

The article [316], which originally introduced the STRETCH constraint, quotes rostering problems as typical examples of use of this constraint.

Remark

We split the original STRETCH constraint into the STRETCH_PATH and the STRETCH_CIRCUIT constraints that respectively use the *PATH LOOP* and the *CIRCUIT LOOP* arc generators. We also reorganise the parameters: the VALUES collection describes the attributes of each value that can be assigned to the variables of the STRETCH_PATH constraint. Finally we skipped the pattern constraint that tells what values can follow a given value. A extension of this constraint (i.e., stretch plus pattern), called FORCED_SHIFT_STRETCH, where one can specify for each value v with a 0-1 variable, whether it should occur at least once or not at all, was proposed in [220]. By reduction to Hamiltonian path it was shown that enforcing arc-consistency for FORCED_SHIFT_STRETCH is NP-hard [220].

Algorithm

A first filtering algorithm was described in the original article of G. Pesant [316]. A second filtering algorithm, based on dynamic programming, achieving arc-consistency is depicted in [219, 220]. It also handles the fact that some values can be followed by only a given subset of values. An other alternative achieving arc-consistency is to use the automaton described in the **Automaton** slot.

Systems STRETCHPATH in **Choco**, STRETCH in **JaCoP**.

See also **common keyword:** CHANGE_CONTINUITY, GROUP *(timetabling constraint)*, GROUP_SKIP_ISOLATED_ITEM *(timetabling constraint,sequence)*, MIN_SIZE_FULL_ZERO_STRETCH *(sequence)*, PATTERN *(sliding sequence constraint,timetabling constraint)*, SLIDING_DISTRIBUTION *(sliding sequence constraint)*, STRETCH_CIRCUIT *(sliding sequence constraint,timetabling constraint)*.

generalisation: STRETCH_PATH_PARTITION *(*`variable` *replaced by* `variable` $\in$ `partition`*)*.

uses in its reformulation: STRETCH_CIRCUIT.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

combinatorial object: sequence.

constraint network structure: Berge-acyclic constraint network.

constraint type: timetabling constraint, sliding sequence constraint.

filtering: dynamic programming, arc-consistency.

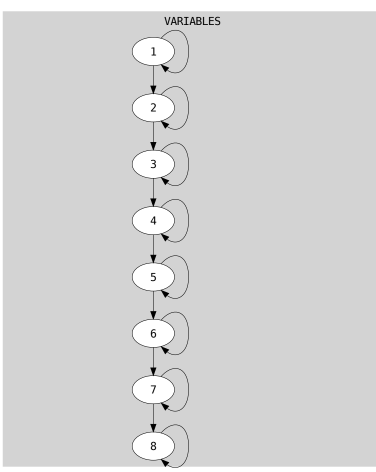
final graph structure: consecutive loops are connected.

For all items of `VALUES`:

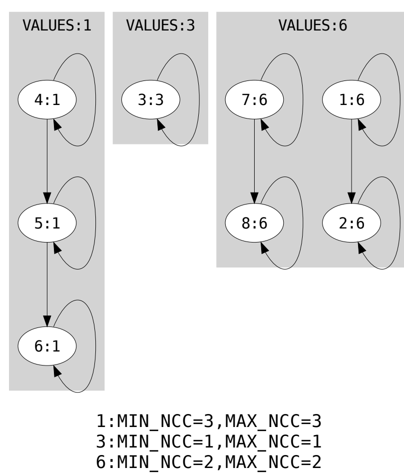
Arc input(s)	`VARIABLES`
Arc generator	*PATH* $\mapsto$ `collection(variables1, variables2)` *LOOP* $\mapsto$ `collection(variables1, variables2)`
Arc arity	2
Arc constraint(s)	• `variables1.var` = `VALUES.val` • `variables2.var` = `VALUES.val`
Graph property(ies)	• NOT_IN(**MIN_NCC**, 1, `VALUES.lmin` − 1) • **MAX_NCC** ≤ `VALUES.lmax`

Graph model Part (A) of Figure 5.776 shows the initial graphs associated with values 1, 2, 3 and 6 of the **Example** slot. Part (B) of Figure 5.776 shows the corresponding final graphs associated with values 1, 3 and 6. Since value 2 is not assigned to any variable of the `VARIABLES` collection the final graph associated with value 2 is empty. The STRETCH_PATH constraint holds since:

- For value 1 we have one connected component for which the number of vertices 3 is greater than or equal to 2 and less than or equal to 4,
- For value 2 we do not have any connected component,
- For value 3 we have one connected component for which the number of vertices 1 is greater than or equal to 1 and less than or equal to 6,
- For value 6 we have two connected components that both contain two vertices: this is greater than or equal to 2 and less than or equal to 2.



(A)



(B)

Figure 5.776: Initial and final graph of the STRETCH_PATH constraint

During the presentation of this constraint at CP'2001 the following point was mentioned: it could be useful to allow domain variables for the minimum and the maximum values of a stretch. This could be achieved in the following way: the `lmin` (respectively `lmax`) attribute would now be a domain variable that gives the size of the shortest (respectively longest) stretch. Finally within the **Graph property(ies)** slot we would replace $\geq$ (and $\leq$) by $=$.

Automaton Let n and m respectively denote the quantities $|\texttt{VARIABLES}|$ and $|\texttt{VALUES}|$. Furthermore, let $\texttt{val}_i$, $\texttt{lmin}_i$ and $\texttt{lmax}_i$, $(i \in [1, m])$, respectively be shortcuts for the expressions $\texttt{VALUES}[i]$.val, $\texttt{VALUES}[i]$.lmin and $\texttt{VALUES}[i]$.lmax. Without loss of generality, we assume that all the lmin attributes of the items of the VALUES collection are at least equal to 1. The following automaton $\mathcal{A}$ involving $1 + \texttt{lmax}_1 + \texttt{lmax}_2 + \cdots + \texttt{lmax}_m$ states only accepts solutions to the STRETCH_PATH constraint. Automaton $\mathcal{A}$ has the following states:

- an initial state s that is also an accepting state,
- $\forall i \in [1, m], \forall j \in [1, \texttt{lmin}_i - 1]$, a non-accepting state $s_{i,j}$,
- $\forall i \in [1, m], \forall j \in [\texttt{lmin}_i, \texttt{lmax}_i]$, an accepting state $s_{i,j}$.

Transitions of $\mathcal{A}$ are defined in the following way:

- $\forall i \in [1, m]$, a transition from s to $s_{i,1}$ labelled by condition $X_l = \texttt{val}_i$,
- a transition from s to s labelled by condition $X_l \neq \texttt{val}_1 \wedge X_l \neq \texttt{val}_2 \wedge \cdots \wedge X_l \neq \texttt{val}_m$,
- $\forall i \in [1, m], \forall j \in [\texttt{lmin}_i, \texttt{lmax}_i]$, a transition from $s_{i,j}$ to s labelled by condition $X_l \neq \texttt{val}_1 \wedge X_l \neq \texttt{val}_2 \wedge \cdots \wedge X_l \neq \texttt{val}_m$,
- $\forall i \in [1, m], \forall j \in [1, \texttt{lmax}_i - 1]$, a transition from $s_{i,j}$ to $s_{i,j+1}$ labelled by condition $X_l = \texttt{val}_i$,
- $\forall i \in [1, m], \forall j \in [\texttt{lmin}_i, \texttt{lmax}_i], \forall k \neq i \in [1, m]$, a transition from $s_{i,j}$ to $s_{k,1}$ labelled by condition $X_l = \texttt{val}_k$.

Figure 5.777 depicts the automaton associated with the STRETCH_PATH constraint of the **Example** slot. Transitions labels 0, 1, 2, 3 and 4 respectively correspond to the conditions $X_l \neq 1 \wedge X_l \neq 2 \wedge X_l \neq 3 \wedge X_l \neq 6$, $X_l = 1$, $X_l = 2$, $X_l = 3$, $X_l = 6$ (since values 1, 2, 3 and 6 respectively correspond to the values of the first, second, third and fourth item of the VALUES collection). The STRETCH_PATH constraint holds since the corresponding sequence of visited states, s s_{41} s_{42} s_{31} s_{11} s_{12} s_{13} s_{41} s_{42}, ends up in an accepting state (i.e., accepting states are denoted graphically by a double circle in the figure).

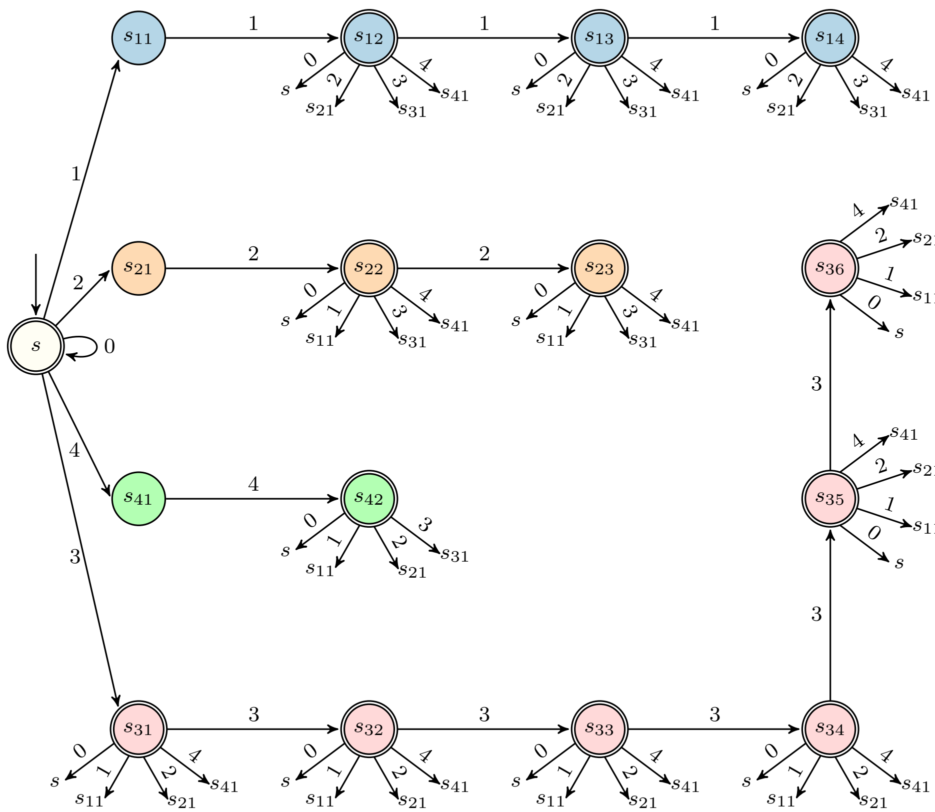


Figure 5.777: Automaton of the STRETCH_PATH constraint of the **Example** slot (states related to a same stretch have the same colour)

5.384 STRETCH_PATH_PARTITION

▶ ▷ ◁ ◀

DESCRIPTION LINKS

Origin Derived from STRETCH_PATH.

Constraint STRETCH_PATH_PARTITION(VARIABLES, PARTLIMITS)

Synonym STRETCH.

Type

VALUES : collection(val−int)

Arguments

VARIABLES : collection(var−dvar)
PARTLIMITS : collection(p − VALUES, lmin−int, lmax−int)

Restrictions

$|\texttt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)
$|\texttt{VARIABLES}| > 0$
required(VARIABLES, var)
$|\texttt{PARTLIMITS}| > 0$
required(PARTLIMITS, [p, lmin, lmax])
$\texttt{PARTLIMITS.lmin} \geq 0$
$\texttt{PARTLIMITS.lmin} \leq \texttt{PARTLIMITS.lmax}$
$\texttt{PARTLIMITS.lmin} \leq |\texttt{VARIABLES}|$

Purpose

In order to define the meaning of the STRETCH_PATH_PARTITION constraint, we first introduce the notions of *stretch* and *span*. Let n be the number of variables of the collection VARIABLES. Let $X_i, \ldots, X_j$ $(1 \leq i \leq j \leq n)$ be consecutive variables of the collection of variables VARIABLES such that the following conditions apply:

- All variables $X_i, \ldots, X_j$ take their values in the same partition of the PARTLIMITS collection (i.e., $\exists l \in [1, |\texttt{PARTLIMITS}|]$ such that $\forall k \in [i, j] : X_k \in \texttt{PARTLIMITS}[l].p$),
- $i = 1$ or X_{i-1} is different from X_i,
- $j = n$ or X_{j+1} is different from X_j.

We call such a set of variables a *stretch*. The *span* of the stretch is equal to $j - i + 1$, while the *value* of the stretch is l. We now define the condition enforced by the STRETCH_PATH_PARTITION constraint.

Each item $\texttt{PARTLIMITS}[l] = (\texttt{p} - values, \texttt{lmin} - s, \texttt{lmax} - t)$ of the PARTLIMITS collection enforces the minimum value s as well as the maximum value t for the span of a stretch of value l over consecutive variables of the VARIABLES collection.

Note that:

1. Having an item $\texttt{PARTLIMITS}[l] = (\texttt{p} - values, \texttt{lmin} - s, \texttt{lmax} - t)$ with s strictly greater than 0 does not mean that values of $values$ should be assigned to one of the variables of collection VARIABLES. It rather means that, when a value of $values$ is used, all stretches of value l must have a span that belong to interval $[s, t]$.
2. A variable of the collection VARIABLES may be assigned a value that is not defined in the attribute p of the PARTLIMITS collection.

Example

$$\left(\begin{array}{l} \langle 1, 2, 0, 0, 2, 2, 2, 0 \rangle, \\ \left\langle \begin{array}{lll} \texttt{p} - \langle 1, 2 \rangle & \texttt{lmin} - 2 & \texttt{lmax} - 4, \\ \texttt{p} - \langle 3 \rangle & \texttt{lmin} - 0 & \texttt{lmax} - 2 \end{array} \right\rangle \end{array} \right)$$

The STRETCH_PATH_PARTITION constraint holds since the sequence 1 2 0 0 2 2 2 0 contains two stretches 1 2, and 2 2 2 respectively verifying the following conditions:

- The span of the first stretch 1 2 is located within interval $[2, 4]$ (i.e., the limit associated with item $\texttt{PARTLIMITS}[1]$).
- The span of the second stretch 2 2 2 is located within interval $[2, 4]$ (i.e., the limit associated with item $\texttt{PARTLIMITS}[1]$).

Typical

$|\texttt{VARIABLES}| > 1$
$\texttt{range}(\texttt{VARIABLES.var}) > 1$
$|\texttt{VARIABLES}| > |\texttt{PARTLIMITS}|$
$|\texttt{PARTLIMITS}| > 1$
$\texttt{sum}(\texttt{PARTLIMITS.lmin}) \leq |\texttt{VARIABLES}|$
$\texttt{PARTLIMITS.lmax} \leq |\texttt{VARIABLES}|$

Symmetries

- Items of VARIABLES can be reversed.
- Items of PARTLIMITS are permutable.
- Items of PARTLIMITS.p are permutable.
- All occurrences of two distinct tuples of values in VARIABLES.var or PARLIMITS.p.val can be swapped; all occurrences of a tuple of values in VARIABLES.var or PARLIMITS.p.val can be renamed to any unused tuple of values.

See also

common keyword: PATTERN *(sliding sequence constraint)*.

specialisation: STRETCH_PATH *(*variable $\in$ partition *replaced by* variable*)*.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint, partition.

combinatorial object: sequence.

constraint network structure: Berge-acyclic constraint network.

constraint type: timetabling constraint, sliding sequence constraint.

filtering: arc-consistency.

final graph structure: consecutive loops are connected.

5.385 STRICT_LEX2

► ▷ ◁ ◀ DESCRIPTION LINKS

Origin [179]

Constraint STRICT_LEX2(MATRIX)

Type VECTOR : collection(var−dvar)

Argument MATRIX : collection(vec − VECTOR)

Restrictions
$|\texttt{VECTOR}| \geq 1$
required(VECTOR, var)
required(MATRIX, vec)
same_size(MATRIX, vec)

Purpose Given a matrix of domain variables, enforces that both adjacent rows, and adjacent columns are lexicographically ordered (adjacent rows and adjacent columns cannot be equal).

Example $(\langle \texttt{vec} - \langle 2, 2, 3\rangle, \texttt{vec} - \langle 2, 3, 1\rangle\rangle)$

The STRICT_LEX2 constraint holds since:

- The first row $\langle 2, 2, 3\rangle$ is lexicographically strictly less than the second row $\langle 2, 3, 1\rangle$.
- The first column $\langle 2, 2\rangle$ is lexicographically strictly less than the second column $\langle 2, 3\rangle$.
- The second column $\langle 2, 3\rangle$ is lexicographically strictly less than the third column $\langle 3, 1\rangle$.

Typical
$|\texttt{VECTOR}| > 1$
$|\texttt{MATRIX}| > 1$

Symmetry One and the same constant can be added to the var attribute of all items of MATRIX.vec.

Usage A *symmetry-breaking* constraint.

Reformulation The STRICT_LEX2 constraint can be expressed as a conjunction of two LEX_CHAIN_LESS constraints: A first LEX_CHAIN_LESS constraint on the MATRIX argument and a second LEX_CHAIN_LESS constraint on the transpose of the MATRIX argument.

Systems STRICT_LEX2 in **MiniZinc**.

See also

common keyword: ALLPERM, LEX_LESSEQ *(lexicographic order)*.

implies: LEX2, LEX_CHAIN_LESS.

part of system of constraints: LEX_CHAIN_LESS.

Keywords

constraint type: predefined constraint, system of constraints, order constraint.

modelling: matrix, matrix model.

symmetry: symmetry, matrix symmetry, lexicographic order.

5.386 STRICTLY_DECREASING

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin Derived from STRICTLY_INCREASING.

Constraint STRICTLY_DECREASING(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restriction required(VARIABLES, var)

Purpose The variables of the collection VARIABLES are strictly decreasing.

Example $(\langle 8, 4, 3, 1\rangle)$

The STRICTLY_DECREASING constraint holds since $8 > 4 > 3 > 1$.

Typical $|\texttt{VARIABLES}| > 2$

Typical model nval(VARIABLES.var) > 2

Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

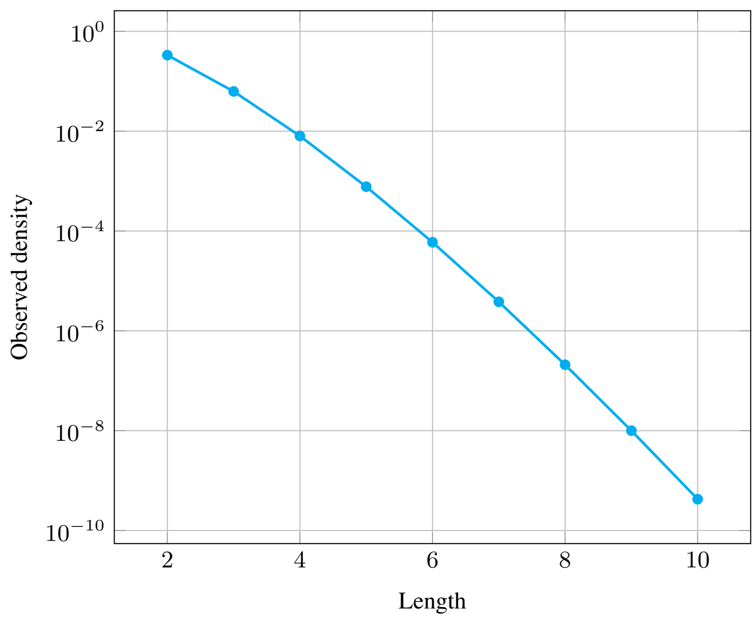
Arg. properties Contractible wrt. VARIABLES.

Counting

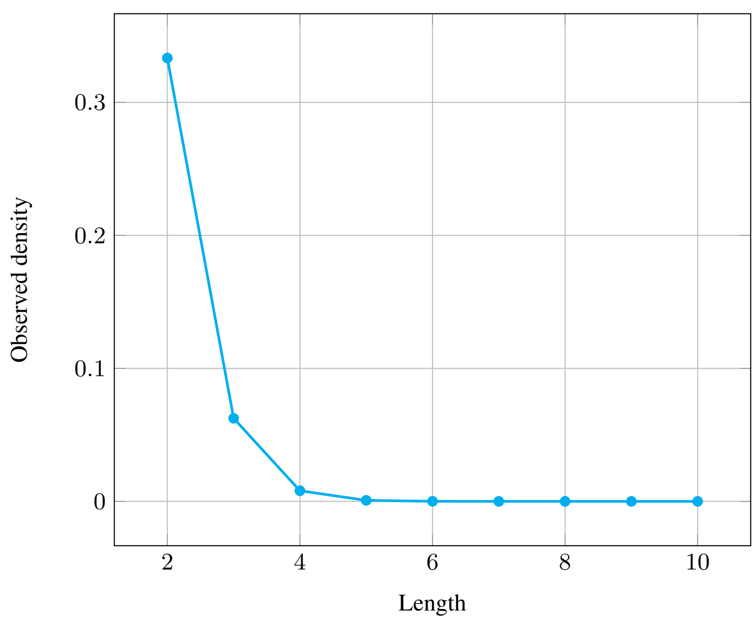
Length (n)	2	3	4	5	6	7	8	9	10
Solutions	3	4	5	6	7	8	9	10	11

Number of solutions for STRICTLY_DECREASING: domains $0..n$

Solution density for STRICTLY_DECREASING



Solution density for STRICTLY_DECREASING



Systems INCREASINGNVALUE in **Choco**, REL in **Gecode**.

See also **common keyword:** INCREASING *(order constraint)*.

comparison swapped: STRICTLY_INCREASING.

implies: ALLDIFFERENT, DECREASING.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

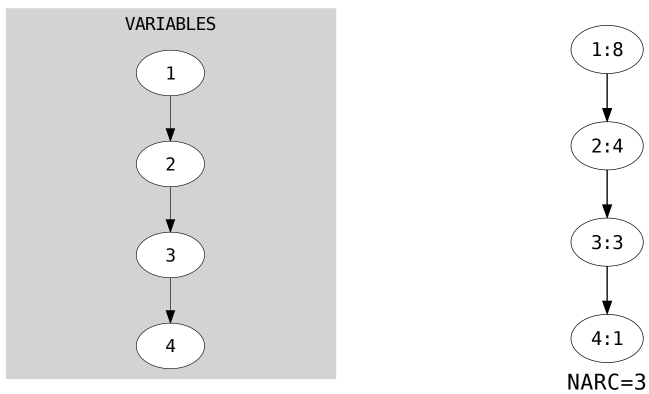
constraint network structure: sliding cyclic(1) constraint network(1).

constraint type: decomposition, order constraint.

filtering: arc-consistency.

Arc input(s)	`VARIABLES`
Arc generator	$PATH \mapsto$ `collection`(`variables1`, `variables2`)
Arc arity	2
Arc constraint(s)	`variables1.var` $>$ `variables2.var`
Graph property(ies)	**NARC** $= \vert$`VARIABLES`$\vert - 1$

Graph model Parts (A) and (B) of Figure 5.778 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.



(A) (B)

Figure 5.778: Initial and final graph of the STRICTLY_DECREASING constraint

Automaton Figure 5.779 depicts the automaton associated with the STRICTLY_DECREASING constraint. To each pair of consecutive variables $(\mathtt{VAR}_i, \mathtt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\mathtt{VAR}_i$, $\mathtt{VAR}_{i+1}$ and S_i: $\mathtt{VAR}_i \leq \mathtt{VAR}_{i+1} \Leftrightarrow S_i$.

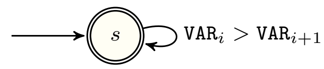


Figure 5.779: Automaton of the STRICTLY_DECREASING constraint

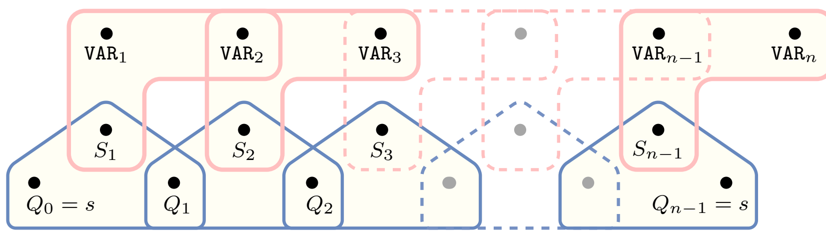


Figure 5.780: Hypergraph of the reformulation corresponding to the automaton of the STRICTLY_DECREASING constraint

5.387 STRICTLY_INCREASING

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin KOALOG

Constraint STRICTLY_INCREASING(VARIABLES)

Argument VARIABLES : collection(var−dvar)

Restriction required(VARIABLES, var)

Purpose The variables of the collection VARIABLES are strictly increasing.

Example $(\langle 1, 3, 6, 8\rangle)$

The STRICTLY_INCREASING constraint holds since $1 < 3 < 6 < 8$.

Typical $|\texttt{VARIABLES}| > 2$

Typical model $\texttt{nval}(\texttt{VARIABLES.var}) > 2$

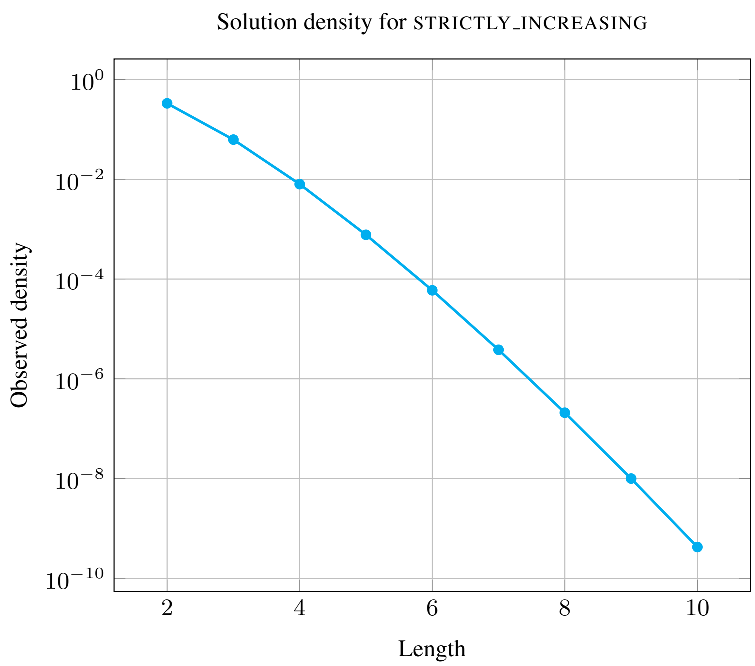
Symmetry One and the same constant can be added to the var attribute of all items of VARIABLES.

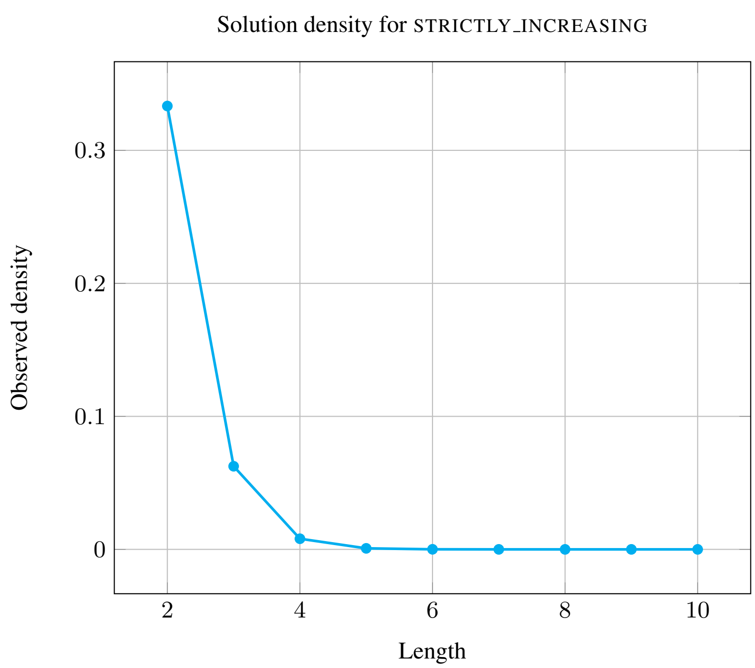
Arg. properties Contractible wrt. VARIABLES.

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	3	4	5	6	7	8	9	10	11

Number of solutions for STRICTLY_INCREASING: domains $0..n$





Systems INCREASINGNVALUE in **Choco**, REL in **Gecode**.

Used in GOLOMB, INT_VALUE_PRECEDE_CHAIN, MAX_OCC_OF_TUPLES_OF_VALUES.

See also

common keyword: DECREASING *(order constraint)*.

comparison swapped: STRICTLY_DECREASING.

implied by: GOLOMB.

implies: ALLDIFFERENT, INCREASING.

uses in its reformulation: ALLDIFFERENT.

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: sliding cyclic(1) constraint network(1).

constraint type: decomposition, order constraint.

filtering: arc-consistency.

Arc input(s)	VARIABLES
Arc generator	*PATH* ↦ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var < variables2.var
Graph property(ies)	**NARC** = |VARIABLES| − 1

Graph model Parts (A) and (B) of Figure 5.781 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

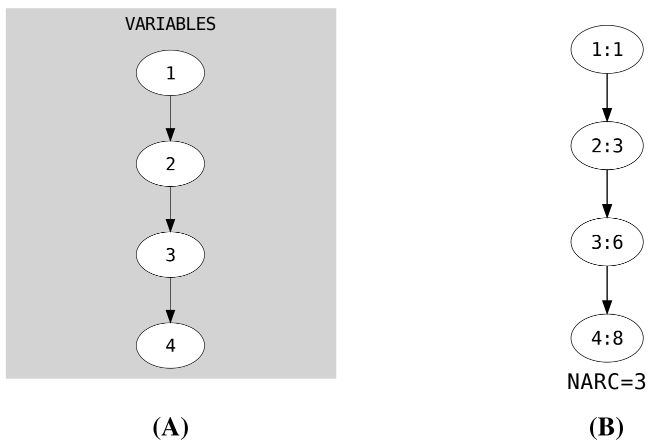


Figure 5.781: Initial and final graph of the STRICTLY_INCREASING constraint

Automaton Figure 5.782 depicts the automaton associated with the STRICTLY_INCREASING constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a 0-1 signature variable S_i. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and S_i: $\texttt{VAR}_i \geq \texttt{VAR}_{i+1} \Leftrightarrow S_i$.

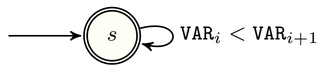


Figure 5.782: Automaton of the STRICTLY_INCREASING constraint

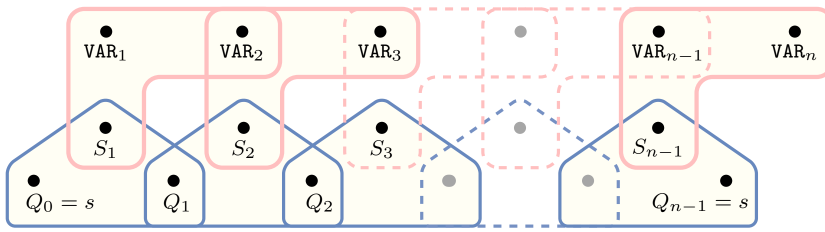


Figure 5.783: Hypergraph of the reformulation corresponding to the automaton of the STRICTLY_INCREASING constraint

5.388 STRONGLY_CONNECTED

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [6]

Constraint STRONGLY_CONNECTED(NODES)

Argument NODES : collection(index−int, succ−svar)

Restrictions
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)

Purpose Consider a digraph G described by the NODES collection. Select a subset of arcs of G so that we have a single strongly connected component involving all vertices of G.

Example

$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\{2\},\\ \texttt{index}-2 & \texttt{succ}-\{3\},\\ \texttt{index}-3 & \texttt{succ}-\{2,5\},\\ \texttt{index}-4 & \texttt{succ}-\{1\},\\ \texttt{index}-5 & \texttt{succ}-\{4\}\end{array}\right\rangle\right)$$

The STRONGLY_CONNECTED constraint holds since the NODES collection depicts a graph involving a single strongly connected component (i.e., since we have a circuit visiting successively the vertices 1, 2, 3, 5, and 4).

Typical |NODES| > 2

Symmetry Items of NODES are permutable.

Algorithm The sketch of a filtering algorithm for the STRONGLY_CONNECTED constraint is given in [151, page 89].

See also
common keyword: LINK_SET_TO_BOOLEANS *(constraint involving set variables)*.
implied by: CONNECTED.
related: CIRCUIT *(one single strongly connected component in the final solution)*.

Keywords
constraint arguments: constraint involving set variables.
constraint type: graph constraint.
filtering: linear programming.
final graph structure: strongly connected component.

Arc input(s) NODES

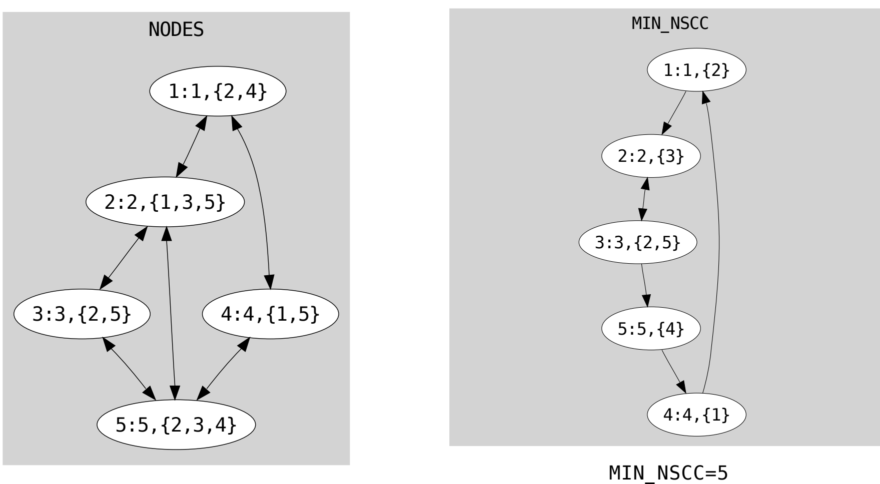
Arc generator *CLIQUE*$\mapsto$collection(nodes1, nodes2)

Arc arity 2

Arc constraint(s) IN_SET(nodes2.index, nodes1.succ)

Graph property(ies) **MIN_NSCC**$=$ |NODES|

Graph model Part (A) of Figure 5.784 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.784 gives the final graph associated with the **Example** slot. The STRONGLY_CONNECTED constraint holds since the final graph contains a single strongly connected component mentioning every vertex of the initial graph.



(A) **(B)**

Figure 5.784: Initial and final graph of the STRONGLY_CONNECTED set constraint

Signature Since the maximum number of vertices of the final graph is equal to |NODES| we can rewrite the graph property **MIN_NSCC** $=$ |NODES| to **MIN_NSCC** $\geq$ |NODES| and simplify $\underline{\textbf{MIN\_NSCC}}$ to $\overline{\textbf{MIN\_NSCC}}$.

5.389 SUBGRAPH_ISOMORPHISM

DESCRIPTION LINKS

Origin [288]

Constraint SUBGRAPH_ISOMORPHISM(NODES_PATTERN, NODES_TARGET, FUNCTION)

Arguments

```
NODES_PATTERN : collection(index-int, succ-sint)
NODES_TARGET  : collection(index-int, succ-svar)
FUNCTION      : collection(image-dvar)
```

Restrictions

```
required(NODES_PATTERN, [index, succ])
NODES_PATTERN.index ≥ 1
NODES_PATTERN.index ≤ |NODES_PATTERN|
distinct(NODES_PATTERN, index)
NODES_PATTERN.succ ≥ 1
NODES_PATTERN.succ ≤ |NODES_PATTERN|
required(NODES_TARGET, [index, succ])
NODES_TARGET.index ≥ 1
NODES_TARGET.index ≤ |NODES_TARGET|
distinct(NODES_TARGET, index)
NODES_TARGET.succ ≥ 1
NODES_TARGET.succ ≤ |NODES_TARGET|
required(FUNCTION, [image])
FUNCTION.image ≥ 1
FUNCTION.image ≤ |NODES_TARGET|
distinct(FUNCTION, image)
|FUNCTION| = |NODES_PATTERN|
```

Purpose

Given two directed graphs PATTERN and TARGET enforce a one to one correspondence, defined by the function FUNCTION, between the vertices of the graph PATTERN and the vertices of an induced subgraph of TARGET so that, if there is an arc from u to v in the graph PATTERN, then there is also an arc from the image of u to the image of v in the induced subgraph of TARGET. The vertices of both graphs are respectively defined by the two collections of vertices NODES_PATTERN and NODES_TARGET. Within collection NODES_PATTERN the set of successors of each node is fixed, while this is not the case for the collection NODES_TARGET. This stems from the fact that the TARGET graph is not fixed (i.e., the lower and upper bounds of the target graph are specified when we post the SUBGRAPH_ISOMORPHISM constraint, while the induced subgraph of a solution to the SUBGRAPH_ISOMORPHISM constraint corresponds to a graph for which the upper and lower bounds are identical).

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\{2,4\},\\ \texttt{index}-2 & \texttt{succ}-\{1,3,4\},\\ \texttt{index}-3 & \texttt{succ}-\emptyset,\\ \texttt{index}-4 & \texttt{succ}-\emptyset\end{array}\right\rangle,\\ \left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\emptyset,\\ \texttt{index}-2 & \texttt{succ}-\{3,4,5\},\\ \texttt{index}-3 & \texttt{succ}-\emptyset,\\ \texttt{index}-4 & \texttt{succ}-\{2,5\},\\ \texttt{index}-5 & \texttt{succ}-\emptyset\end{array}\right\rangle,\\ \langle 4,2,3,5\rangle\end{array}\right)$$

Figure 5.785 gives the pattern (see Part (A)) and target graph (see Part (B)) of the **Example** slot as well as the one to one correspondence (see Part (C)) between the pattern graph and the induced subgraph of the target graph. The SUBGRAPH_ISOMORPHISM constraint since:

- To the arc from vertex 1 to vertex 4 in the pattern graph corresponds the arc from vertex 4 to 5 in the induced subgraph of the target graph.
- To the arc from vertex 1 to vertex 2 in the pattern graph corresponds the arc from vertex 4 to 2 in the induced subgraph of the target graph.
- To the arc from vertex 2 to vertex 1 in the pattern graph corresponds the arc from vertex 2 to 4 in the induced subgraph of the target graph.
- To the arc from vertex 2 to vertex 4 in the pattern graph corresponds the arc from vertex 2 to 5 in the induced subgraph of the target graph.
- To the arc from vertex 2 to vertex 3 in the pattern graph corresponds the arc from vertex 2 to 3 in the induced subgraph of the target graph.

Typical

$|\texttt{NODES\_PATTERN}| > 1$
$|\texttt{NODES\_TARGET}| > 1$

Symmetries

- Items of NODES_PATTERN are permutable.
- Items of NODES_TARGET are permutable.

Usage

Within the context of constraint programming the constraint was used for finding symmetries [336, 338, 337].

Algorithm

[423, 352, 265, 456].

See also

related: GRAPH_ISOMORPHISM.

Keywords

constraint arguments: constraint involving set variables.

constraint type: predefined constraint, graph constraint.

symmetry: symmetry.

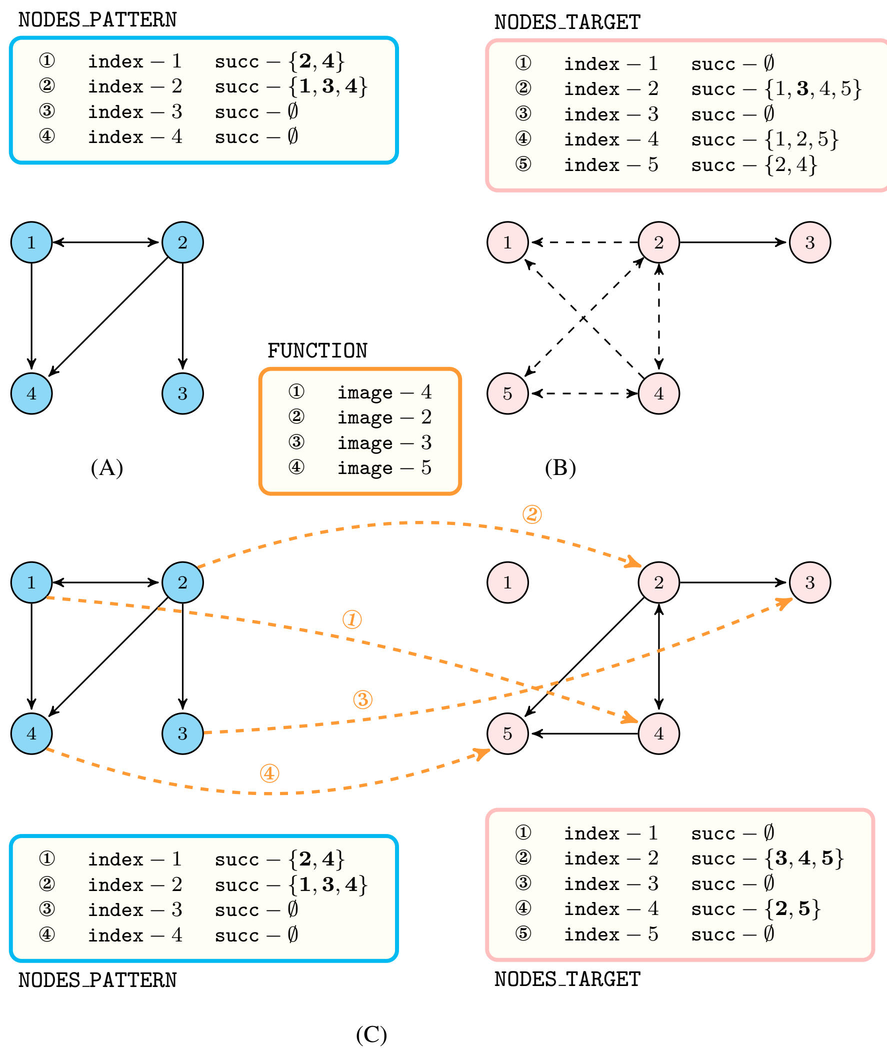


Figure 5.785: Illustration of the **Example** slot: (A) The pattern graph, (B) a possible initial target graph – plain arcs must belong to the induced subgraph, while dotted arcs may or may not belong to the induced subgraph – and (C) the correspondence, denoted by thick dashed arcs, between the vertices of the pattern graph and the vertices of the induced subgraph of the target graph. Within a set variable a bold value (respectively a plain value) represents a value that for sure belong (respectively that may belong) to the set.

5.390 SUM

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [455].

Constraint sum(INDEX, SETS, CONSTANTS, S)

Synonym SUM_PRED.

Arguments

INDEX	:	dvar
SETS	:	collection(ind−int, set−sint)
CONSTANTS	:	collection(cst−int)
S	:	dvar

Restrictions

$|\texttt{SETS}| \geq 1$
required(SETS, [ind, set])
distinct(SETS, ind)
$|\texttt{CONSTANTS}| \geq 1$
required(CONSTANTS, cst)

Purpose S is equal to the sum of the constants of CONSTANTS corresponding to the $\texttt{INDEX}^{th}$ set of the SETS collection.

Example

$$\left(\begin{array}{l} 8, \left\langle \begin{array}{ll} \texttt{ind}-8 & \texttt{set}-\{2,3\}, \\ \texttt{ind}-1 & \texttt{set}-\{3\}, \\ \texttt{ind}-3 & \texttt{set}-\{1,4,5\}, \\ \texttt{ind}-6 & \texttt{set}-\{2,4\} \end{array} \right\rangle, \\ \langle 4,9,1,3,1\rangle, 10 \end{array}\right)$$

The SUM constraint holds since its last argument S = 10 is equal to the sum of the 2^{th} and 3^{th} items of the collection $\langle 4,9,1,3,1\rangle$. As illustrated by Figure 5.786, this stems from the fact that its first argument INDEX = 8 corresponds to the value of the ind attribute of the first item of the SETS collection. Consequently the corresponding set $\{2,3\}$ is used for summing the 2^{th} and 3^{th} items of the CONSTANTS collection.

Typical

$|\texttt{SETS}| > 1$
$|\texttt{CONSTANTS}| > |\texttt{SETS}|$
range(CONSTANTS.cst) > 1

Symmetry Items of SETS are permutable.

Arg. properties Functional dependency: S determined by INDEX, SETS and CONSTANTS.

Usage In his article introducing the SUM constraint, Tallys H. Yunes mentions the *Sequence Dependent Cumulative Cost Problem* as the subproblem that originally motivates this constraint.

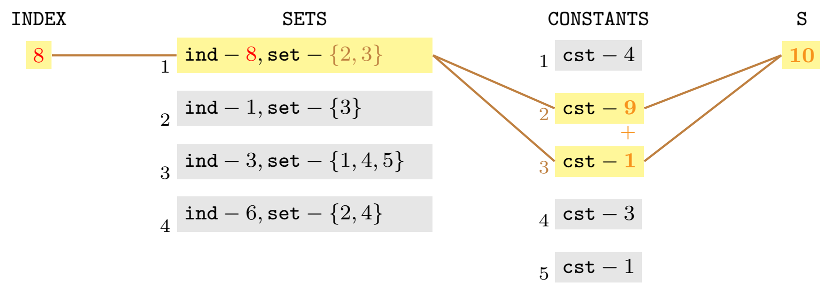


Figure 5.786: Illustration of the correspondence between the arguments of the SUM(INDEX, SETS, CONSTANTS, S) constraint in the context of the **Example** slot (from right to left, $\mathtt{S} = 10$ is equal to the sum of the constants 9 and 1 corresponding to the indices 2 and 3 of the set for which the ind attribute is equal to $\mathtt{INDEX} = 8$)

Remark The SUM constraint is called SUM_PRED in **MiniZinc** (http://www.minizinc.org/).

Algorithm The article [455] gives the convex hull relaxation of the SUM constraint.

Systems SUM_PRED in **MiniZinc**.

See also **common keyword:** ELEMENT *(data constraint)*, SUM_CTR, SUM_SET *(sum)*.

used in graph description: IN_SET.

Keywords **characteristic of a constraint:** convex hull relaxation, sum.

constraint type: data constraint.

filtering: linear programming.

modelling: functional dependency.

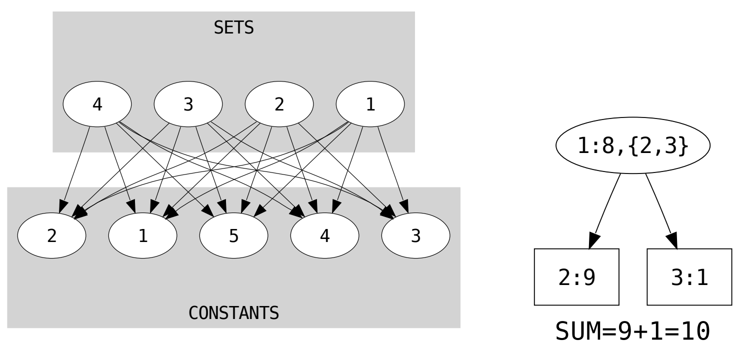
Arc input(s)	SETS CONSTANTS
Arc generator	*PRODUCT* $\mapsto$ collection(sets, constants)
Arc arity	2
Arc constraint(s)	• INDEX = sets.ind • IN_SET(constants.key, sets.set)
Graph property(ies)	**SUM**(CONSTANTS, cst) = S

Graph model According to the value assigned to INDEX the arc constraint selects for the final graph:

- The INDEXth item of the SETS collection,
- The items of the CONSTANTS collection for which the key correspond to the indices of the INDEXth set of the SETS collection.

Finally, since we use the **SUM** graph property on the cst attribute of the CONSTANTS collection, the last argument S of the SUM constraint is equal to the sum of the constants associated with the vertices of the final graph.

Parts (A) and (B) of Figure 5.787 respectively show the initial and final graph associated with the **Example** slot. Since we use the **SUM** graph property we show the vertices from which we compute S in a box.



(A) **(B)**

Figure 5.787: Initial and final graph of the SUM constraint

5.391 SUM_CTR

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Arithmetic constraint.

Constraint SUM_CTR(VARIABLES, CTR, VAR)

Synonyms CONSTANT_SUM, SUM, LINEAR, SCALAR_PRODUCT.

Arguments

VARIABLES	:	collection(var−dvar)
CTR	:	atom
VAR	:	dvar

Restrictions

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose Constraint the sum of a set of domain variables. More precisely, let S denote the sum of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

Example $(\langle 1, 1, 4\rangle, =, 6)$

The SUM_CTR constraint holds since the condition $1 + 1 + 4 = 6$ is satisfied.

Typical

$|\text{VARIABLES}| > 1$
range(VARIABLES.var) > 1
CTR $\in [=, <, \geq, >, \leq]$

Symmetry Items of VARIABLES are permutable.

Arg. properties

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) ≥ 0.
- Contractible wrt. VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) ≤ 0.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) $\geq$ 0.
- Extensible wrt. VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) $\leq$ 0.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

Remark J.-C. Régin *et al.* show in [365, 366] how to handle a SUM_CTR constraint subject of a set of difference constraints.

When CTR corresponds to $=$ this constraint is referenced under the names CONSTANT_SUM in KOALOG (http://www.koalog.com/php/index.php) and SUM in **JaCoP** (http://www.jacop.eu/).

Systems EQUATION in **Choco**, LINEAR in **Gecode**, SCALAR_PRODUCT in **SICStus**.

Used in BIN_PACKING, CUMULATIVE, CUMULATIVE_CONVEX, CUMULATIVE_WITH_LEVEL_OF_PRIORITY, CUMULATIVES, INDEXED_SUM, INTERVAL_AND_SUM, RELAXED_SLIDING_SUM, SLIDING_SUM, SLIDING_TIME_WINDOW_SUM.

See also **assignment dimension added:** INTERVAL_AND_SUM *(assignment dimension corresponding to intervals is added)*.

common keyword: ARITH_SLIDING *(arithmetic constraint)*, INCREASING_SUM *(sum)*, PRODUCT_CTR, RANGE_CTR *(arithmetic constraint)*, SUM, SUM_CUBES_CTR, SUM_POWERS4_CTR, SUM_POWERS5_CTR, SUM_POWERS6_CTR *(sum)*, SUM_SET *(arithmetic constraint)*, SUM_SQUARES_CTR *(sum)*.

generalisation: SCALAR_PRODUCT *(arithmetic constraint where all coefficients are not necessarly equal to* 1*)*.

implied by: ARITH_SLIDING.

system of constraints: SLIDING_SUM.

Keywords **characteristic of a constraint:** sum.

constraint type: arithmetic constraint.

heuristics: regret based heuristics, regret based heuristics in matrix problems.

Cond. implications

- SUM_CTR(VARIABLES, CTR, VAR)
  with VARIABLES.var ≥ 0
  and VARIABLES.var ≤ 1
  implies SUM_SQUARES_CTR(VARIABLES, CTR, VAR)
  when VARIABLES.var ≥ 0
  and VARIABLES.var ≤ 1.

- SUM_CTR(VARIABLES, CTR, VAR)
  with VARIABLES.var ≥ -1
  and VARIABLES.var ≤ 1
  implies SUM_CUBES_CTR(VARIABLES, CTR, VAR)
  when VARIABLES.var ≥ -1
  and VARIABLES.var ≤ 1.

- SUM_CTR(VARIABLES, CTR, VAR)
  with VARIABLES.var ≥ -1
  and VARIABLES.var ≤ 1
  implies SUM_POWERS5_CTR(VARIABLES, CTR, VAR)
  when VARIABLES.var ≥ -1
  and VARIABLES.var ≤ 1.

- SUM_CTR(VARIABLES, CTR, VAR)
  with CTR $\in [=]$
  and INCREASING(VARIABLES)
  implies INCREASING_SUM(VARIABLES, VAR).

Arc input(s)	VARIABLES
Arc generator	*SELF* $\mapsto$ collection(variables)
Arc arity	1
Arc constraint(s)	TRUE
Graph property(ies)	**SUM**(VARIABLES, var) CTR VAR

Graph model Since we want to keep all the vertices of the initial graph we use the *SELF* arc generator together with the TRUE arc constraint. This predefined arc constraint always holds.

Parts (A) and (B) of Figure 5.788 respectively show the initial and final graph associated with the **Example** slot. Since we use the TRUE arc constraint both graphs are identical.

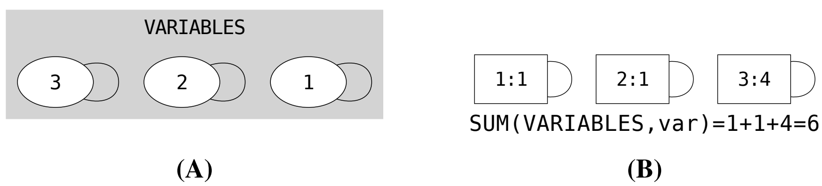


Figure 5.788: Initial and final graph of the SUM_CTR constraint

5.392 SUM_CUBES_CTR

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic constraint.

Constraint SUM_CUBES_CTR(VARIABLES, CTR, VAR)

Synonyms SUM_CUBES, SUM_OF_CUBES, SUM_OF_CUBES_CTR.

Arguments

VARIABLES	:	collection(var−dvar)
CTR	:	atom
VAR	:	dvar

Restrictions

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose Constraint the sum of the cubes of a set of domain variables. More precisely, let S denote the sum of the cubes of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

Example $(\langle 1, 2, 2\rangle, =, 17)$

The SUM_CUBES_CTR constraint holds since the condition $1^3 + 2^3 + 2^3 = 17$ is satisfied.

Typical

$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1
CTR $\in [=, <, \geq, >, \leq]$

Symmetry Items of VARIABLES are permutable.

Arg. properties

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) ≥ 0.
- Contractible wrt. VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) ≤ 0.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) ≥ 0.
- Extensible wrt. VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) ≤ 0.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

See also **common keyword:** SUM_CTR, SUM_POWERS4_CTR, SUM_POWERS5_CTR, SUM_POWERS6_CTR, SUM_SQUARES_CTR *(sum)*.

Keywords **characteristic of a constraint:** sum.

constraint type: predefined constraint, arithmetic constraint.

5.393 SUM_FREE

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin [439]

Constraint SUM_FREE(S)

Argument S : svar

Purpose Impose for all pairs of values (not necessarily distinct) i, j of the set S the fact that the sum $i + j$ is not an element of S.

Example $(\{1, 3, 5, 9\})$

The SUM_FREE$(\{1, 3, 5, 9\})$ constraint holds since:

- $1 + 1 = 2 \notin$ S, $1 + 3 = 4 \notin$ S, $1 + 5 = 6 \notin$ S, $1 + 9 = 10 \notin$ S.
- $3 + 3 = 6 \notin$ S, $3 + 5 = 8 \notin$ S, $3 + 9 = 12 \notin$ S.
- $5 + 5 = 10 \notin$ S, $5 + 9 = 14 \notin$ S.

Usage The SUM_FREE constraint was introduced by W.-J. van Hoeve and A. Sabharwal in order to model in a concise way Schur problems.

- On one hand, the first model has n domain variables x_i $(1 \leq i \leq n)$, where x_i corresponds to the subset in which element i occurs. The constraints $x_i = s \wedge x_j = s \Rightarrow x_{i+j} \neq s$ $(s \in [1, k], i, j \in [1, n], i \leq j, i + j \leq n)$ enforce that the k subsets are sum-free. We have $O(k \cdot n^2)$ such constraints.
- On the other hand, the model proposed by W.-J. van Hoeve and A. Sabharwal represents in an explicit way with a set variable S_i $(1 \leq i \leq n)$ each subset of the partition we are looking for. Now, to express the fact that these k subsets are sum-free they simply use k SUM_FREE constraints of the form SUM_FREE(S_i).

While the two models have the same behaviour when we focus on the number of backtracks the second model is much more efficient from a memory point of view.

Algorithm W.-J. van Hoeve and A. Sabharwal have proposed an algorithm that enforces bound-consistency for the SUM_FREE constraint in [439].

Keywords **constraint arguments:** unary constraint, constraint involving set variables.

constraint type: predefined constraint.

filtering: bound-consistency.

problems: Schur number.

5.394 SUM_OF_INCREMENTS

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin [95]

Constraint SUM_OF_INCREMENTS(VARIABLES, LIMIT)

Synonyms INCREMENTS_SUM, INCR_SUM, SUM_INCR, SUM_INCREMENTS.

Arguments

VARIABLES	:	collection(var−dvar)
LIMIT	:	dvar

Restrictions

required(VARIABLES, var)
VARIABLES.var ≥ 0
LIMIT ≥ 0

Purpose

Given a collection of variables VARIABLES which can only be assigned non negative values, and a variable LIMIT, enforce the condition $\texttt{VARIABLES}[1].\texttt{var} + \sum_{i=2}^{|\texttt{VARIABLES}|} \max(\texttt{VARIABLES}[i].\texttt{var} - \texttt{VARIABLES}[i-1].\texttt{var}, 0) \leq \texttt{LIMIT}$. $\texttt{VARIABLES}[1].\texttt{var}$ stands from the fact that we assume an additional implicit 0 before the first variable (i.e., $\texttt{VARIABLES}[1].\texttt{var} = \max(\texttt{VARIABLES}[1].\texttt{var} - 0, 0)$).

Example

$(\langle 4, 4, 3, 4, 6\rangle, 7)$

The SUM_OF_INCREMENTS constraint holds since we have that $4 + \max(4-4, 0) + \max(3-4, 0) + \max(4-3, 0) + \max(6-4, 0) \leq 7$.

Typical

$|\texttt{VARIABLES}| > 2$
range(VARIABLES.var) > 1
maxval(VARIABLES.var) > 0
LIMIT > 0
LIMIT $\leq$ |VARIABLES|*range(VARIABLES.var)/2

Symmetries

- One and the same constant can be added to VARIABLES.var and to LIMIT.
- Items of VARIABLES can be reversed.
- LIMIT can be increased.

Arg. properties

- Prefix-contractible wrt. VARIABLES.
- Suffix-contractible wrt. VARIABLES.

Usage The SUM_OF_INCREMENTS was initially motivated by the problem of decomposing a matrix of non-negative integers into a positive linear combination of matrices consisting of only zeros and ones, where the ones occur consecutively in each row.

Algorithm A $O(|\texttt{VARIABLES}|)$ bound-consistency filtering algorithm for the SUM_OF_INCREMENTS constraint is described in [95].

Reformulation The following reformulations are provided in [95]. Assuming $\texttt{VARIABLES}[0].\texttt{var}$ is defined as 0 (i.e., a zero is added before the first variable of the VARIABLES collection) we have:

- $\sum_{i=1}^{|\texttt{VARIABLES}|} S_i \leq \texttt{LIMIT}$ with $D_i = \texttt{VARIABLES}[i].\texttt{var} - \texttt{VARIABLES}[i-1].\texttt{var}$ and $S_i = \max(D_i, 0)$ $(1 \leq i \leq |\texttt{VARIABLES}|)$.

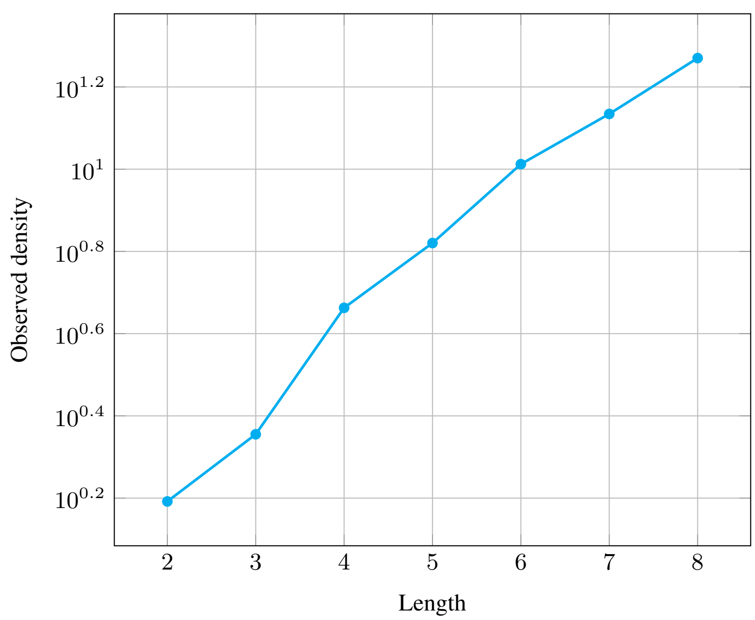
- $\sum_{i=1}^{|\texttt{VARIABLES}|} S_i \leq \texttt{LIMIT}$ with $\texttt{VARIABLES}[i].\texttt{var} - \texttt{VARIABLES}[i-1].\texttt{var} \leq S_i$ and $S_i \in [0, \overline{\texttt{LIMIT}}]$ $(1 \leq i \leq |\texttt{VARIABLES}|)$.

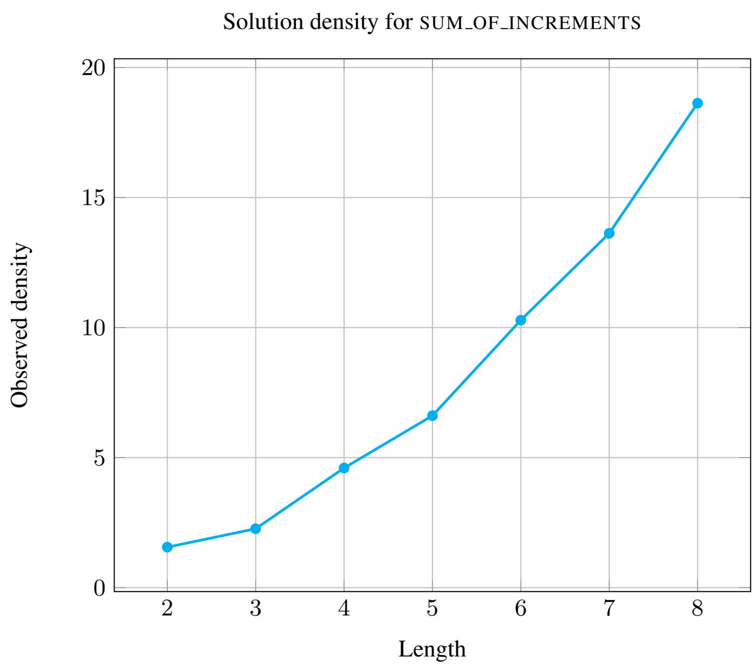
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	14	145	2875	51415	1210104	28573741	801944469

Number of solutions for SUM_OF_INCREMENTS: domains $0..n$

Solution density for SUM_OF_INCREMENTS

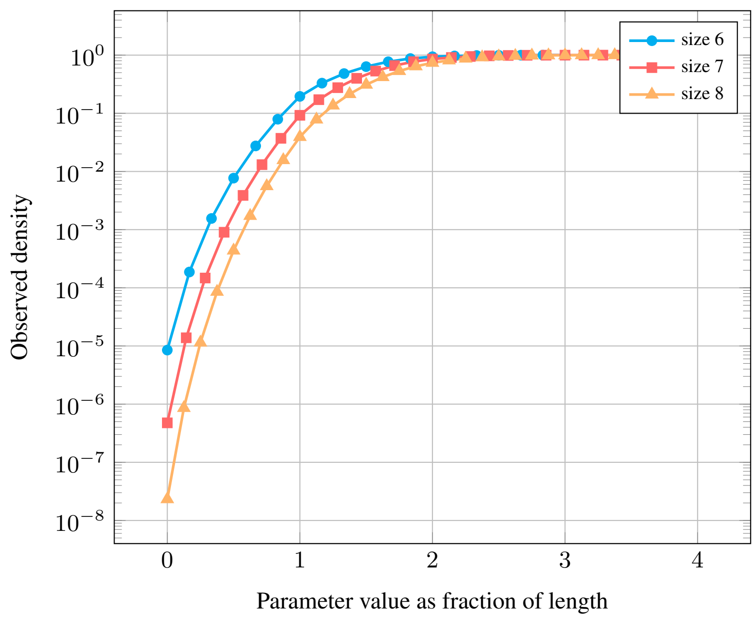



Solution density for SUM_OF_INCREMENTS
Observed density
0
5
10
15
20
2
3
4
5
6
7
8
Length

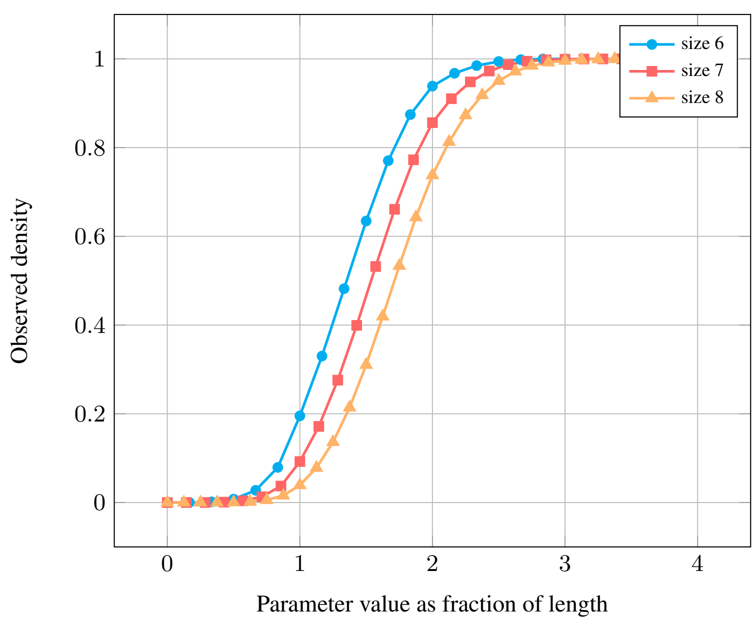
Length (n)		2	3	4	5	6	7	8
Total		14	145	2875	51415	1210104	28573741	801944469
Parameter value	0	1	1	1	1	1	1	1
	1	4	7	11	16	22	29	37
	2	9	23	51	101	183	309	493
	3	-	54	156	396	904	1891	3679
	4	-	60	375	1167	3235	8135	18835
	5	-	-	485	2848	9318	27483	74143
	6	-	-	563	4263	22981	77947	240751
	7	-	-	608	5568	38836	193742	675244
	8	-	-	625	6616	56703	359880	1688427
	9	-	-	-	7314	74658	578511	3369015
	10	-	-	-	7650	90639	837441	5865915
	11	-	-	-	7720	102875	1115687	9220695
	12	-	-	-	7755	110425	1386029	13354545
	13	-	-	-	-	113827	1619993	18051195
	14	-	-	-	-	115857	1795694	22965651
	15	-	-	-	-	116942	1908968	27670800
	16	-	-	-	-	117437	1988222	31755573
	17	-	-	-	-	117612	2039616	34989993
	18	-	-	-	-	117649	2069933	37574073
	19	-	-	-	-	-	2085763	39526569
	20	-	-	-	-	-	2092817	40912205
	21	-	-	-	-	-	2095436	41827847
	22	-	-	-	-	-	2096360	42386387
	23	-	-	-	-	-	2096822	42700112
	24	-	-	-	-	-	2097032	42865683
	25	-	-	-	-	-	-	42953199
	26	-	-	-	-	-	-	43002171
	27	-	-	-	-	-	-	43027581
	28	-	-	-	-	-	-	43039551
	29	-	-	-	-	-	-	43044507
	30	-	-	-	-	-	-	43046215
	31	-	-	-	-	-	-	43046656
	32	-	-	-	-	-	-	43046721

Solution count for SUM_OF_INCREMENTS: domains $0..n$

Solution density for SUM_OF_INCREMENTS



Solution density for SUM_OF_INCREMENTS



Keywords **characteristic of a constraint:** difference, sum.
constraint type: predefined constraint.
filtering: bound-consistency.

5.395 SUM_OF_WEIGHTS_OF_DISTINCT_VALUES

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [46]

Constraint SUM_OF_WEIGHTS_OF_DISTINCT_VALUES(VARIABLES, VALUES, COST)

Synonym SWDV.

Arguments

VARIABLES	:	collection(var−dvar)
VALUES	:	collection(val−int, weight−int)
COST	:	dvar

Restrictions

required(VARIABLES, var)
|VALUES| > 0
required(VALUES, [val, weight])
VALUES.weight ≥ 0
distinct(VALUES, val)
in_attr(VARIABLES, var, VALUES, val)
COST ≥ 0

Purpose

All variables of the VARIABLES collection take a value in the VALUES collection. In addition COST is the sum of the weight attributes associated with the distinct values taken by the variables of VARIABLES.

Example

$$\left(\begin{array}{l} \langle 1,6,1 \rangle, \\ \left\langle \begin{array}{ll} \mathtt{val}-1 & \mathtt{weight}-5, \\ \mathtt{val}-2 & \mathtt{weight}-3, \\ \mathtt{val}-6 & \mathtt{weight}-7 \end{array} \right\rangle, 12 \end{array} \right)$$

The SUM_OF_WEIGHTS_OF_DISTINCT_VALUES constraint holds since its last argument COST $= 12$ is equal to the sum $5 + 7$ of the weights of the values 1 and 6 that occur within the $\langle 1, 6, 1\rangle$ collection.

Typical

|VARIABLES| > 1
range(VARIABLES.var) > 1
|VALUES| > 1
range(VALUES.weight) > 1
maxval(VALUES.weight) > 0

Symmetries

- Items of VARIABLES are permutable.
- All occurrences of two distinct values of VARIABLES.var can be swapped.
- Items of VALUES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val can be renamed to any unused value.

Arg. properties Functional dependency: COST determined by VARIABLES and VALUES.

See also

attached to cost variant: NVALUE *(all values have a weight of* 1*)*.

common keyword: GLOBAL_CARDINALITY_WITH_COSTS, MINIMUM_WEIGHT_ALLDIFFERENT, WEIGHTED_PARTIAL_ALLDIFF *(weighted assignment)*.

Keywords

application area: assignment.

constraint arguments: pure functional dependency.

constraint type: relaxation.

filtering: cost filtering constraint.

modelling: functional dependency.

problems: domination, weighted assignment, facilities location problem.

Arc input(s)	VARIABLES VALUES
Arc generator	$PRODUCT \mapsto$ collection(variables, values)
Arc arity	2
Arc constraint(s)	variables.var = values.val
Graph property(ies)	• **NSOURCE** = |VARIABLES| • **SUM**(VALUES, weight) = COST

Signature Since we use the $PRODUCT$ arc generator, the number of sources of the final graph cannot exceed the number of sources of the initial graph. Since the initial graph contains |VARIABLES| sources, this number is an upper bound of the number of sources of the final graph. Therefore we can rewrite **NSOURCE** = |VARIABLES| to **NSOURCE** $\geq$ |VARIABLES| and simplify $\underline{\overline{\textbf{NSOURCE}}}$ to $\overline{\textbf{NSOURCE}}$.

Parts (A) and (B) of Figure 5.789 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** graph property, the source vertices of the final graph are shown in a double circle. Since we also use the **SUM** graph property we show the vertices from which we compute the total cost in a box.

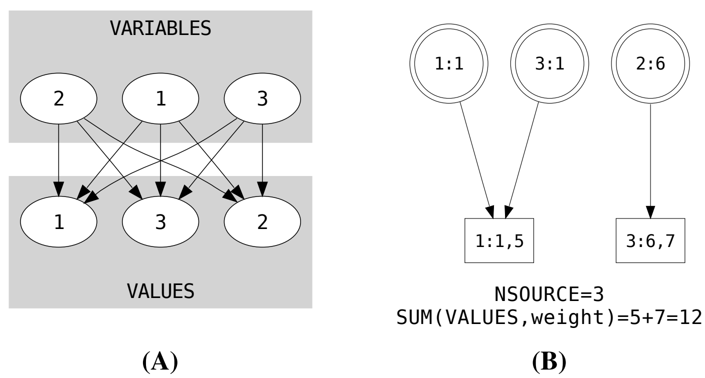


Figure 5.789: Initial and final graph of the SUM_OF_WEIGHTS_OF_DISTINCT_VALUES constraint

5.396 SUM_POWERS4_CTR

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin

Arithmetic constraint.

Constraint

SUM_POWERS4_CTR(VARIABLES, CTR, VAR)

Synonyms

SUM_POWERS4, SUM_OF_POWERS4, SUM_OF_POWERS4_CTR.

Arguments

VARIABLES	:	collection(var−dvar)
CTR	:	atom
VAR	:	dvar

Restrictions

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Constraint the sum of the power of four of a set of domain variables. More precisely, let S denote the sum of the power of four of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

Example

$(\langle 1, 1, 2\rangle, =, 18)$

The SUM_POWERS4_CTR constraint holds since the condition $1^4 + 1^4 + 2^4 = 18$ is satisfied.

Typical

$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1
CTR $\in [=, <, \geq, >, \leq]$

Symmetry

Items of VARIABLES are permutable.

Arg. properties

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

See also

common keyword: SUM_CTR, SUM_CUBES_CTR, SUM_POWERS5_CTR, SUM_POWERS6_CTR, SUM_SQUARES_CTR *(sum)*.

Keywords

characteristic of a constraint: sum.

constraint type: predefined constraint, arithmetic constraint.

5.397 SUM_POWERS5_CTR

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic constraint.

Constraint SUM_POWERS5_CTR(VARIABLES, CTR, VAR)

Synonyms SUM_POWERS5, SUM_OF_POWERS5, SUM_OF_POWERS5_CTR.

Arguments

VARIABLES	:	collection(var−dvar)
CTR	:	atom
VAR	:	dvar

Restrictions

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose Constraint the sum of the power of five of a set of domain variables. More precisely, let S denote the sum of the power of five of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

Example $(\langle 1, 1, 2\rangle, =, 34)$

The SUM_POWERS5_CTR constraint holds since the condition $1^5 + 1^5 + 2^5 = 34$ is satisfied.

Typical

|VARIABLES| > 1
range(VARIABLES.var) > 1
CTR $\in [=, <, \geq, >, \leq]$

Symmetry Items of VARIABLES are permutable.

Arg. properties

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$ and minval(VARIABLES.var) ≥ 0.
- Contractible wrt. VARIABLES when CTR $\in [\geq, >]$ and maxval(VARIABLES.var) ≤ 0.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$ and minval(VARIABLES.var) ≥ 0.
- Extensible wrt. VARIABLES when CTR $\in [<, \leq]$ and maxval(VARIABLES.var) ≤ 0.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

See also **common keyword:** SUM_CTR, SUM_CUBES_CTR, SUM_POWERS4_CTR, SUM_POWERS6_CTR, SUM_SQUARES_CTR *(sum)*.

Keywords

characteristic of a constraint: sum.

constraint type: predefined constraint, arithmetic constraint.

5.398 SUM_POWERS6_CTR

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin Arithmetic constraint.

Constraint SUM_POWERS6_CTR(VARIABLES, CTR, VAR)

Synonyms SUM_POWERS6, SUM_OF_POWERS6, SUM_OF_POWERS6_CTR.

Arguments

VARIABLES	:	collection(var−dvar)
CTR	:	atom
VAR	:	dvar

Restrictions

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose Constraint the sum of the power of six of a set of domain variables. More precisely, let S denote the sum of the power of six of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

Example $(\langle 1, 1, 2\rangle, =, 66)$

The SUM_POWERS6_CTR constraint holds since the condition $1^6 + 1^6 + 2^6 = 66$ is satisfied.

Typical

$|$VARIABLES$| > 1$
range(VARIABLES.var) > 1
CTR $\in [=, <, \geq, >, \leq]$

Symmetry Items of VARIABLES are permutable.

Arg. properties

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

See also **common keyword:** SUM_CTR, SUM_CUBES_CTR, SUM_POWERS4_CTR, SUM_POWERS5_CTR, SUM_SQUARES_CTR *(sum)*.

Keywords **characteristic of a constraint:** sum.

constraint type: predefined constraint, arithmetic constraint.

5.399 SUM_SET

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin

H. Cambazard

Constraint

SUM_SET(SV, VALUES, CTR, VAR)

Arguments

SV	:	svar
VALUES	:	collection(val−int, coef−int)
CTR	:	atom
VAR	:	dvar

Restrictions

required(VALUES, [val, coef])
distinct(VALUES, val)
VALUES.coef ≥ 0
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose

Let SUM denote the sum of the coef attributes of the VALUES collection for which the corresponding values val occur in the set SV. Enforce the following constraint to hold: SUM CTR VAR.

Example

$$\left(\begin{array}{l}\{2,3,6\}, \\ \left\langle\begin{array}{ll}\texttt{val}-2 & \texttt{coef}-7, \\ \texttt{val}-9 & \texttt{coef}-1, \\ \texttt{val}-5 & \texttt{coef}-7, \\ \texttt{val}-6 & \texttt{coef}-2\end{array}\right\rangle, =, 9\end{array}\right)$$

The SUM_SET constraint holds since the sum of the coef attributes $7 + 2$ for which the corresponding val attribute belongs to the first argument SV $= \{2, 3, 6\}$ is equal (i.e., since CTR is set to $=$) to its last argument VAR $= 9$.

Typical

$|\texttt{VALUES}| > 1$
VALUES.coef > 0
CTR $\in [=, <, \geq, >, \leq]$

Symmetry

Items of VALUES are permutable.

Systems

WEIGHTS in **Gecode**.

See also

common keyword: SUM, SUM_CTR (*sum*).

Keywords

characteristic of a constraint: sum.

constraint arguments: binary constraint, constraint involving set variables.

constraint type: arithmetic constraint.

Arc input(s)	VALUES
Arc generator	$SELF \mapsto$ collection(values)
Arc arity	1
Arc constraint(s)	IN_SET(values.val, SV)
Graph property(ies)	**SUM**(VALUES, coef) CTR VAR

Graph model Parts (A) and (B) of Figure 5.790 respectively show the initial and final graph associated with the **Example** slot.

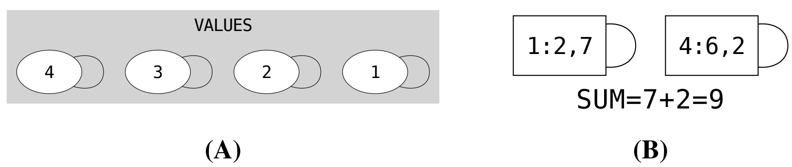


Figure 5.790: Initial and final graph of the SUM_SET constraint

5.400 SUM_SQUARES_CTR

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic constraint.

Constraint SUM_SQUARES_CTR(VARIABLES, CTR, VAR)

Synonyms SUM_SQUARES, SUM_OF_SQUARES, SUM_OF_SQUARES_CTR.

Arguments

VARIABLES	:	collection(var−dvar)
CTR	:	atom
VAR	:	dvar

Restrictions

required(VARIABLES, var)
CTR $\in [=, \neq, <, \geq, >, \leq]$

Purpose Constraint the sum of the squares of a set of domain variables. More precisely, let S denote the sum of the squares of the variables of the VARIABLES collection (when the collection is empty the corresponding sum is equal to 0). Enforce the following constraint to hold: S CTR VAR.

Example $(\langle 1, 1, 4\rangle, =, 18)$

The SUM_SQUARES_CTR constraint holds since the condition $1^2 + 1^2 + 4^2 = 18$ is satisfied.

Typical

$|\texttt{VARIABLES}| > 1$
range(VARIABLES.var) > 1
CTR $\in [=, <, \geq, >, \leq]$

Symmetry Items of VARIABLES are permutable.

Arg. properties

- Contractible wrt. VARIABLES when CTR $\in [<, \leq]$.
- Extensible wrt. VARIABLES when CTR $\in [\geq, >]$.
- Aggregate: VARIABLES(union), CTR(id), VAR(+).

See also **common keyword:** SUM_CTR, SUM_CUBES_CTR, SUM_POWERS4_CTR, SUM_POWERS5_CTR, SUM_POWERS6_CTR *(sum)*.

Keywords **characteristic of a constraint:** sum.

constraint type: predefined constraint, arithmetic constraint.

Cond. implications

- SUM_SQUARES_CTR(VARIABLES, CTR, VAR)
  with VARIABLES.var ≥ -1
  and VARIABLES.var ≤ 1
  implies SUM_POWERS4_CTR(VARIABLES, CTR, VAR)
  when VARIABLES.var ≥ -1
  and VARIABLES.var ≤ 1.

- SUM_SQUARES_CTR(VARIABLES, CTR, VAR)
  with VARIABLES.var ≥ -1
  and VARIABLES.var ≤ 1
  implies SUM_POWERS6_CTR(VARIABLES, CTR, VAR)
  when VARIABLES.var ≥ -1
  and VARIABLES.var ≤ 1.

5.401 SYMMETRIC

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [151]

Constraint SYMMETRIC(NODES)

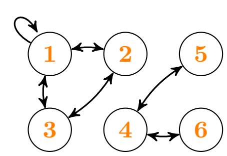
Argument NODES : collection(index−int, succ−svar)

Restrictions
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)

Purpose Consider a digraph G described by the NODES collection. Select a subset of arcs of G so that the corresponding graph is symmetric (i.e., if there is an arc from i to j, there is also an arc from j to i).

Example

$$\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-\{1,2,3\}, \\ \texttt{index}-2 & \texttt{succ}-\{1,3\}, \\ \texttt{index}-3 & \texttt{succ}-\{1,2\}, \\ \texttt{index}-4 & \texttt{succ}-\{5,6\}, \\ \texttt{index}-5 & \texttt{succ}-\{4\}, \\ \texttt{index}-6 & \texttt{succ}-\{4\} \end{array}\right\rangle\right)$$



The SYMMETRIC constraint holds since the NODES collection depicts a symmetric graph.

Typical |NODES| > 2

Symmetry Items of NODES are permutable.

Algorithm The filtering algorithm for the SYMMETRIC constraint is given in [151, page 87]. It removes (respectively imposes) the arcs (i, j) for which the arc (j, i) is not present (respectively is present). It has an overall complexity of $O(n+m)$ where n and m respectively denote the number of vertices and the number of arcs of the initial graph.

See also **common keyword:** CONNECTED *(symmetric)*.

used in graph description: IN_SET.

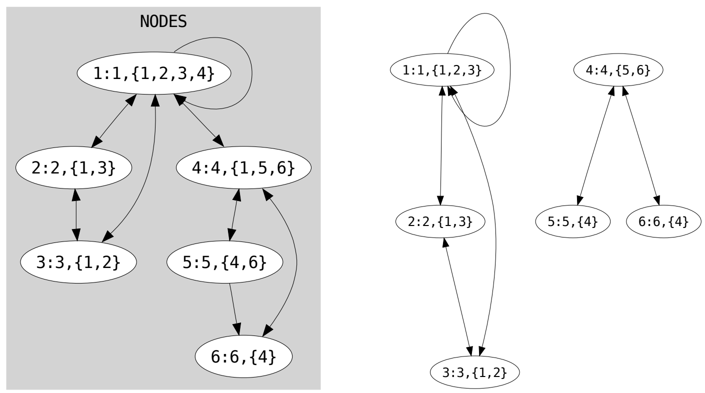
Keywords **constraint arguments:** constraint involving set variables.

constraint type: graph constraint.

final graph structure: symmetric.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.succ)
Graph class	SYMMETRIC

Graph model Part (A) of Figure 5.791 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.791 gives the final graph associated with the **Example** slot.



(A) **(B)**

Figure 5.791: Initial and final graph of the SYMMETRIC set constraint

5.402 SYMMETRIC_ALLDIFFERENT

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [356]

Constraint SYMMETRIC_ALLDIFFERENT(NODES)

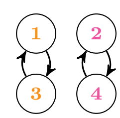
Synonyms SYMMETRIC_ALLDIFF, SYMMETRIC_ALLDISTINCT, SYMM_ALLDIFFERENT, SYMM_ALLDIFF, SYMM_ALLDISTINCT, ONE_FACTOR, TWO_CYCLE.

Argument NODES : collection(index−int, succ−dvar)

Restrictions
$|\texttt{NODES}| \bmod 2 = 0$
required(NODES, [index, succ])
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
distinct(NODES, index)
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$

Purpose All variables associated with the succ attribute of the NODES collection should be pairwise distinct. In addition enforce the following condition: if variable NODES$[i]$.succ takes value j with $j \neq i$ then variable NODES$[j]$.succ takes value i. This can be interpreted as a graph-covering problem where one has to cover a digraph G with circuits of length two in such a way that each vertex of G belongs to a single circuit.

Example
$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-3,\\ \texttt{index}-2 & \texttt{succ}-4,\\ \texttt{index}-3 & \texttt{succ}-1,\\ \texttt{index}-4 & \texttt{succ}-2\end{array}\right\rangle\right)$$



The SYMMETRIC_ALLDIFFERENT constraint holds since:

- NODES[1].succ $= 3 \Leftrightarrow$ NODES[3].succ $= 1$,
- NODES[2].succ $= 4 \Leftrightarrow$ NODES[4].succ $= 2$.

All solutions Figure 5.792 gives all solutions to the following non ground instance of the SYMMETRIC_ALLDIFFERENT constraint: $\texttt{S}_1 \in [1,4]$, $\texttt{S}_2 \in [1,3]$, $\texttt{S}_3 \in [1,4]$, $\texttt{S}_4 \in [1,3]$, SYMMETRIC_ALLDIFFERENT($\langle 1\,\texttt{S}_1, 2\,\texttt{S}_2, 3\,\texttt{S}_3, 4\,\texttt{S}_4\rangle$).

Typical $|\texttt{NODES}| \geq 4$

Symmetry Items of NODES are permutable.

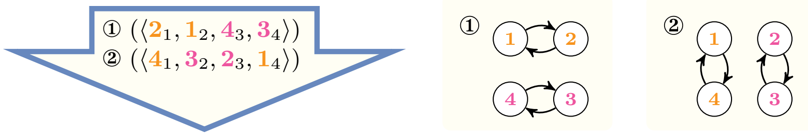


Figure 5.792: All solutions corresponding to the non ground example of the SYMMETRIC_ALLDIFFERENT constraint of the **All solutions** slot; in the left-hand side the `index` attributes are displayed as indices of the `succ` attribute, while in the right-hand side they are directly displayed within each node.

Usage As it was reported in [356, page 420], this constraint is useful to express matches between persons or between teams. The SYMMETRIC_ALLDIFFERENTconstraint also appears implicitly in the *cycle cover problem* and corresponds to the four conditions given in section 1 *Modeling the Cycle Cover Problem* of [319].

Remark This constraint is referenced under the name ONE_FACTOR in [222] as well as in [420]. From a modelling point of view this constraint can be expressed with the CYCLE constraint [47] where one imposes the additional condition that each cycle has only two nodes.

Algorithm A filtering algorithm for the SYMMETRIC_ALLDIFFERENT constraint was proposed by J.-C. Régin in [356]. It achieves arc-consistency and its running time is dominated by the complexity of finding all edges that do not belong to any maximum cardinality matching in an undirected n-vertex, m-edge graph, i.e., $O(m \cdot n)$.

For the soft case of the SYMMETRIC_ALLDIFFERENT constraint where the cost is the minimum number of variables to assign differently in order to get back to a solution, a filtering algorithm achieving arc-consistency is described in [140, 139]. It has a complexity of $O(p \cdot m)$, where p is the number of maximal extreme sets in the value graph associated with the constraint and m is the number of edges. It iterates over extreme sets and not over vertices as in the algorithm due to J.-C. Régin.

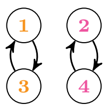
Reformulation The SYMMETRIC_ALLDIFFERENT(NODES) constraint can be expressed in term of a conjunction of $|\texttt{NODES}|^2$ reified constraints of the form $\texttt{NODES}[i].\texttt{succ} = j \Leftrightarrow \texttt{NODES}[j].\texttt{succ} = i$ $(1 \leq i, j \leq |\texttt{NODES}|)$. The SYMMETRIC_ALLDIFFERENT constraint can also be reformulated as an INVERSE constraint as shown below:

$$\text{SYMMETRIC\_ALLDIFFERENT}\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-s_1,\\ \texttt{index}-2 & \texttt{succ}-s_2,\\ \vdots & \vdots\\ \texttt{index}-n & \texttt{succ}-s_n\end{array}\right\rangle\right)$$

$$\text{INVERSE}\left(\left\langle\begin{array}{lll}\texttt{index}-1 & \texttt{succ}-s_1 & \texttt{pred}-s_1,\\ \texttt{index}-2 & \texttt{succ}-s_2 & \texttt{pred}-s_2,\\ \vdots & \vdots & \vdots\\ \texttt{index}-n & \texttt{succ}-s_n & \texttt{pred}-s_n\end{array}\right\rangle\right)$$

A third reformulation using one ALLDIFFERENT constraint and n ELEMENT constraints is as follows. ALLDIFFERENT$(\langle s_1, s_2, \ldots, s_n\rangle)$, $\forall i \in [1, n] : s_i \neq i$, ELEMENT$(s_i, \langle s_1, s_2, \ldots, s_n\rangle, i)$. Figure 5.793 illustrates this third reformulation of the SYMMETRIC_ALLDIFFERENT constraint.

(A)



(B)

ALLDIFFERENT$(\langle 3, 4, 1, 2\rangle)$	*two vertices do not point to the same vertex*
$3 \neq 1, 4 \neq 2, 1 \neq 3, 2 \neq 4$	*no self-loops*
ELEMENT$(3, \langle 3, 4, 1, 2\rangle, 1)$	*successor of vertex 1, vertex 3, goes back to 1*
ELEMENT$(4, \langle 3, 4, 1, 2\rangle, 2)$	*successor of vertex 2, vertex 4, goes back to 2*
ELEMENT$(1, \langle 3, 4, 1, 2\rangle, 3)$	*successor of vertex 3, vertex 1, goes back to 3*
ELEMENT$(2, \langle 3, 4, 1, 2\rangle, 4)$	*successor of vertex 4, vertex 2, goes back to 4*

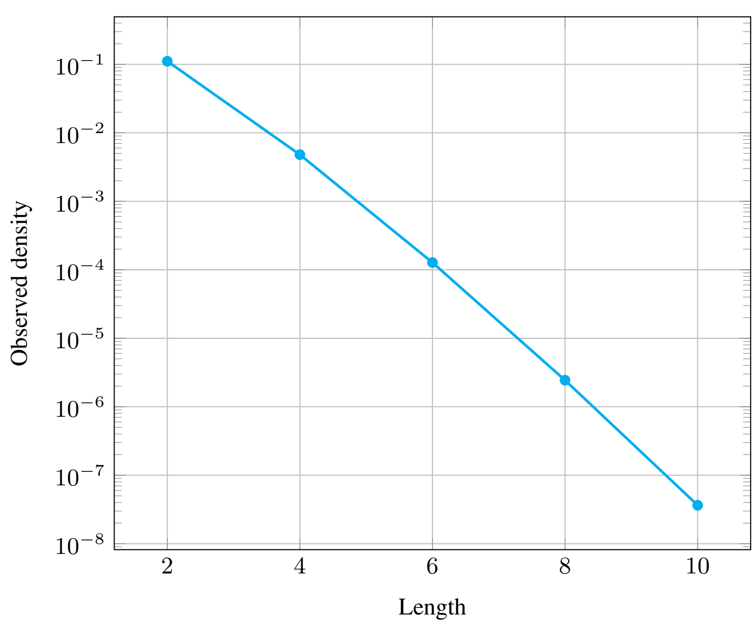
Figure 5.793: Illustrating the reformulation of the SYMMETRIC_ALLDIFFERENT$(\langle 1\ 3, 2\ 4, 3\ 1, 4\ 2\rangle)$ constraint with a set of ELEMENT constraints: (A) the figure of the **Example** slot, and (B) the corresponding constraints

Counting

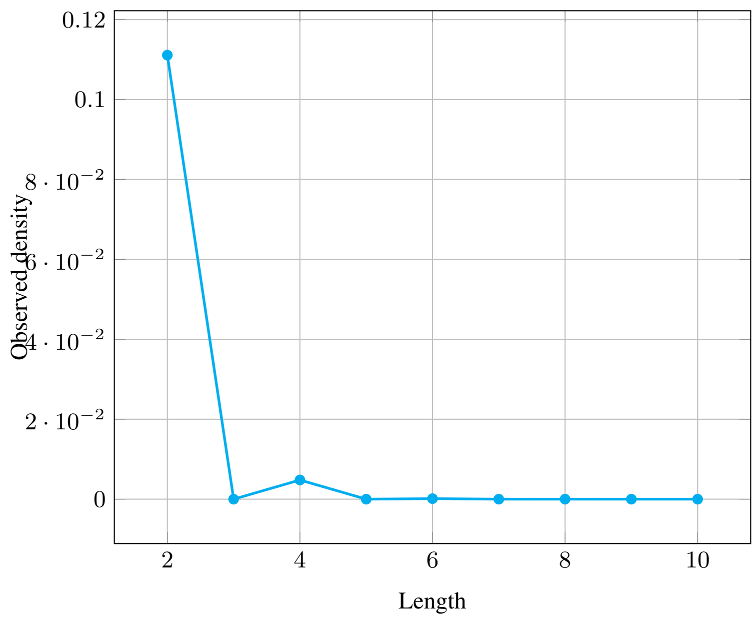
Length (n)	2	3	4	5	6	7	8	9	10
Solutions	1	0	3	0	15	0	105	0	945

Number of solutions for SYMMETRIC_ALLDIFFERENT: domains $0..n$

Solution density for SYMMETRIC_ALLDIFFERENT



Solution density for SYMMETRIC_ALLDIFFERENT



See also

common keyword: ALLDIFFERENT, CYCLE, INVERSE *(permutation)*.

implies: DERANGEMENT, SYMMETRIC_ALLDIFFERENT_EXCEPT_0, SYMMETRIC_ALLDIFFERENT_LOOP.

implies (items to collection): K_ALLDIFFERENT, LEX_ALLDIFFERENT.

related: ROOTS.

Keywords

application area: sport timetabling.

characteristic of a constraint: all different, disequality.

combinatorial object: permutation, involution, matching.

constraint type: graph constraint, timetabling constraint, graph partitioning constraint.

filtering: arc-consistency.

final graph structure: circuit.

modelling: cycle.

Cond. implications

- SYMMETRIC_ALLDIFFERENT(NODES)
  implies BALANCE_CYCLE(BALANCE, NODES)
  when BALANCE $= 0$.
- SYMMETRIC_ALLDIFFERENT(NODES)
  implies CYCLE(NCYCLE, NODES)
  when $2 *$ NCYCLE $= |$NODES$|$.
- SYMMETRIC_ALLDIFFERENT(NODES)
  implies PERMUTATION(VARIABLES : NODES).

Arc input(s)	NODES
Arc generator	$CLIQUE(\neq) \mapsto$collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	• nodes1.succ = nodes2.index • nodes2.succ = nodes1.index
Graph property(ies)	**NARC**= |NODES|

Graph model In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices.

Parts (A) and (B) of Figure 5.794 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

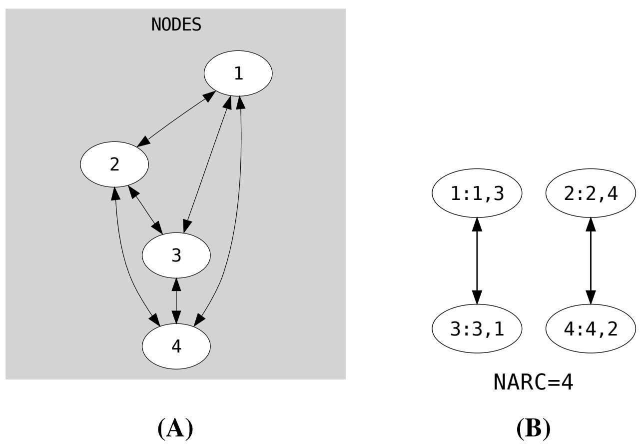


Figure 5.794: Initial and final graph of the SYMMETRIC_ALLDIFFERENT constraint

Signature Since all the index attributes of the NODES collection are distinct, and because of the first condition nodes1.succ = nodes2.index of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to the maximum number of vertices |NODES| of the final graph. So we can rewrite $\mathbf{NARC}$ = |NODES| to $\mathbf{NARC} \geq$ |NODES| and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.403 SYMMETRIC_ALLDIFFERENT_EXCEPT_0

► ▷ ◁ ◄ DESCRIPTION LINKS

Origin Derived from SYMMETRIC_ALLDIFFERENT

Constraint SYMMETRIC_ALLDIFFERENT_EXCEPT_0(NODES)

Synonyms SYMMETRIC_ALLDIFF_EXCEPT_0, SYMMETRIC_ALLDISTINCT_EXCEPT_0, SYMM_ALLDIFFERENT_EXCEPT_0, SYMM_ALLDIFF_EXCEPT_0, SYMM_ALLDISTINCT_EXCEPT_0.

Argument NODES : collection(index−int, succ−dvar)

Restrictions

required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 0
NODES.succ $\leq$ |NODES|

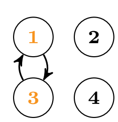
Purpose

Enforce the following three conditions:

1. $\forall i \in [1, |\texttt{NODES}|]$, $\forall j \in [1, |\texttt{NODES}|]$, $(j \neq i)$: $\texttt{NODES}[i].\texttt{succ} = 0 \vee \texttt{NODES}[j].\texttt{succ} = 0 \vee \texttt{NODES}[i].\texttt{succ} \neq \texttt{NODES}[j].\texttt{succ}$.
2. $\forall i \in [1, |\texttt{NODES}|] : \texttt{NODES}[i].\texttt{succ} \neq i$.
3. $\texttt{NODES}[i].\texttt{succ} = j \wedge j \neq i \wedge j \neq 0 \Leftrightarrow \texttt{NODES}[j].\texttt{succ} = i \wedge i \neq j \wedge i \neq 0$.

Example

$$\left(\left\langle\begin{array}{ll} \texttt{index}-1 & \texttt{succ}-3, \\ \texttt{index}-2 & \texttt{succ}-0, \\ \texttt{index}-3 & \texttt{succ}-1, \\ \texttt{index}-4 & \texttt{succ}-0 \end{array}\right\rangle\right)$$



The SYMMETRIC_ALLDIFFERENT_EXCEPT_0 constraint holds since:

- $\texttt{NODES}[1].\texttt{succ} = 3 \Leftrightarrow \texttt{NODES}[3].\texttt{succ} = 1$,
- $\texttt{NODES}[2].\texttt{succ} = 0$ and value 2 is not assigned to any variable.
- $\texttt{NODES}[4].\texttt{succ} = 0$ and value 4 is not assigned to any variable.

Given 3 successor variables that have to be assigned a value in interval $[0, 3]$, the solutions to the SYMMETRIC_ALLDIFFERENT_EXCEPT_0 $(\langle \texttt{index}-1\ \texttt{succ}-s_1, \texttt{index}-2\ \texttt{succ}-s_2, \texttt{index}-3\ \texttt{succ}-s_3\rangle)$ constraint are $\langle 1\ 0, 2\ 0, 3\ 0\rangle$, $\langle 1\ 0, 2\ 3, 3\ 2\rangle$, $\langle 1\ 2, 2\ 1, 3\ 0\rangle$, and $\langle 1\ 3, 2\ 0, 3\ 1\rangle$.

Given 4 successor variables that have to be assigned a value in interval $[0, 3]$, the solutions to the SYMMETRIC_ALLDIFFERENT_EXCEPT_0 $(\langle \texttt{index}-1\ \texttt{succ}-s_1, \texttt{index}-2\ \texttt{succ}-$

s_2, index $-$ 3 succ $-$ s_3, index $-$ 4 succ $-$ s_4⟩) constraint are $\langle 1\ 0, 2\ 0, 3\ 0, 4\ 0\rangle$, $\langle 1\ 0, 2\ 0, 3\ 4, 4\ 3\rangle$, $\langle 1\ 0, 2\ 3, 3\ 2, 4\ 0\rangle$, $\langle 1\ 0, 2\ 4, 3\ 0, 4\ 2\rangle$, $\langle 1\ 2, 2\ 1, 3\ 0, 4\ 0\rangle$, $\langle 1\ 2, 2\ 1, 3\ 4, 4\ 3\rangle$, $\langle 1\ 3, 2\ 0, 3\ 1, 4\ 0\rangle$, $\langle 1\ 3, 2\ 4, 3\ 1, 4\ 2\rangle$, $\langle 1\ 4, 2\ 0, 3\ 0, 4\ 1\rangle$, $\langle 1\ 4, 2\ 3, 3\ 2, 4\ 1\rangle$.

All solutions

Figure 5.795 gives all solutions to the following non ground instance of the SYMMETRIC_ALLDIFFERENT_EXCEPT_0 constraint: $\mathtt{S}_1 \in [0,5]$, $\mathtt{S}_2 \in [1,3]$, $\mathtt{S}_3 \in [1,4]$, $\mathtt{S}_4 \in [0,3]$, $\mathtt{S}_5 \in [0,2]$, SYMMETRIC_ALLDIFFERENT_EXCEPT_0($\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4, 5\ \mathtt{S}_5\rangle$).

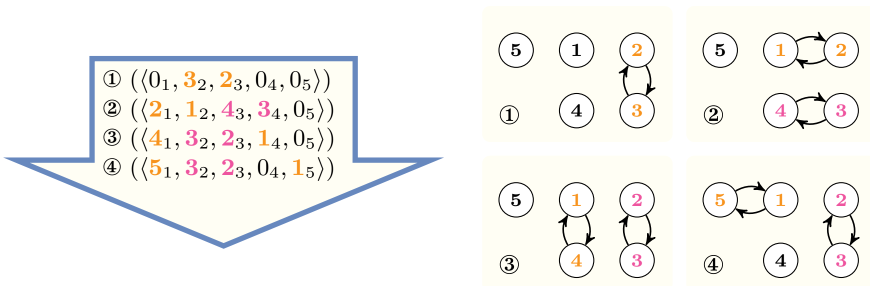


Figure 5.795: All solutions corresponding to the non ground example of the SYMMETRIC_ALLDIFFERENT_EXCEPT_0 constraint of the **All solutions** slot; in the left-hand side the index attributes are displayed as indices of the succ attribute, while in the right-hand side they are directly displayed within each node.

Typical

$|\mathtt{NODES}| \geq 4$
minval(NODES.succ) $= 0$
maxval(NODES.succ) > 0

Symmetry

Items of NODES are permutable.

Usage

Within the context of sport scheduling, NODES$[i]$.succ $= j$ (with $i, j \in [1, |\mathtt{NODES}|]$, $i \neq j$) is interpreted as the fact that team i plays against team j, while NODES$[i]$.succ $= 0$ (with $i \in [1, |\mathtt{NODES}|]$) is interpreted as the fact that team i does not play at all.

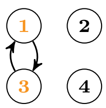
Algorithm

An arc-consistency filtering algorithm for the SYMMETRIC_ALLDIFFERENT_EXCEPT_0 constraint is described in [140, 139]. The algorithm is based on the following facts:

- First, one can map solutions to the SYMMETRIC_ALLDIFFERENT_EXCEPT_0 constraint to perfect (g, f)-matchings in a non-bipartite graph derived from the domain of the variables of the constraint where $g(x) = 0$, $f(x) = 1$ for vertices x which have 0 in their domains, and $g(x) = f(x) = 1$ for all the remaining vertices. A *perfect* (g, f)*-matching* $\mathcal{M}$ of a graph is a subset of edges such that every vertex x is incident with the number of edges in $\mathcal{M}$ between $g(x)$ and $f(x)$.
- Second, Gallai-Edmonds decomposition [190, 159] allows us to find out all edges that do not belong to any perfect (g, f)-matchings, and therefore prune the corresponding variables.

Reformulation Let n denotes the number nodes of the NODES collection, and let s_i (with $i \in [1, n]$) denotes NODES$[i]$.succ. The SYMMETRIC_ALLDIFFERENT_EXCEPT_0(NODES) constraint can be expressed in term of a conjunction of one ALLDIFFERENT_EXCEPT_0 constraint and n ELEMENT constraints as follows. ALLDIFFERENT_EXCEPT_0$(\langle s_1, s_2, \ldots, s_n\rangle)$, $\forall i \in [1, n] : s_i \neq i$, $t_i = s_i + 1$, ELEMENT$(t_i, \langle i, s_1, s_2, \ldots, s_n\rangle, i)$. Note the introduction of the intermediate variables t_i (with $i \in [1, n]$) motivated by the need to have an index starting at 1 rather than 0 in the ELEMENT constraint. Figure 5.796 illustrates this reformulation of the SYMMETRIC_ALLDIFFERENT_EXCEPT_0 constraint.

(A)



(B)

ALLDIFFERENT_EXCEPT_0$(\langle \mathbf{3}, \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle)$	*two vertices do not point to the same vertex*
$\mathbf{3} \neq \mathbf{1}, \mathbf{0} \neq \mathbf{2}, \mathbf{1} \neq \mathbf{3}, \mathbf{0} \neq \mathbf{4}$	*no self-loops*
ELEMENT$(\mathbf{3} + \mathbf{1}, \langle \mathbf{1}, \mathbf{3}, \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle, \mathbf{1})$	*successor of vertex* **1***, vertex* **3***, goes back to* **1**
ELEMENT$(\mathbf{0} + \mathbf{1}, \langle \mathbf{2}, \mathbf{3}, \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle, \mathbf{2})$	*successor of vertex* **2** *is undefined*
ELEMENT$(\mathbf{1} + \mathbf{1}, \langle \mathbf{3}, \mathbf{3}, \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle, \mathbf{3})$	*successor of vertex* **3***, vertex* **1***, goes back to* **3**
ELEMENT$(\mathbf{0} + \mathbf{1}, \langle \mathbf{4}, \mathbf{3}, \mathbf{0}, \mathbf{1}, \mathbf{0}\rangle, \mathbf{4})$	*successor of vertex* **4** *is undefined*

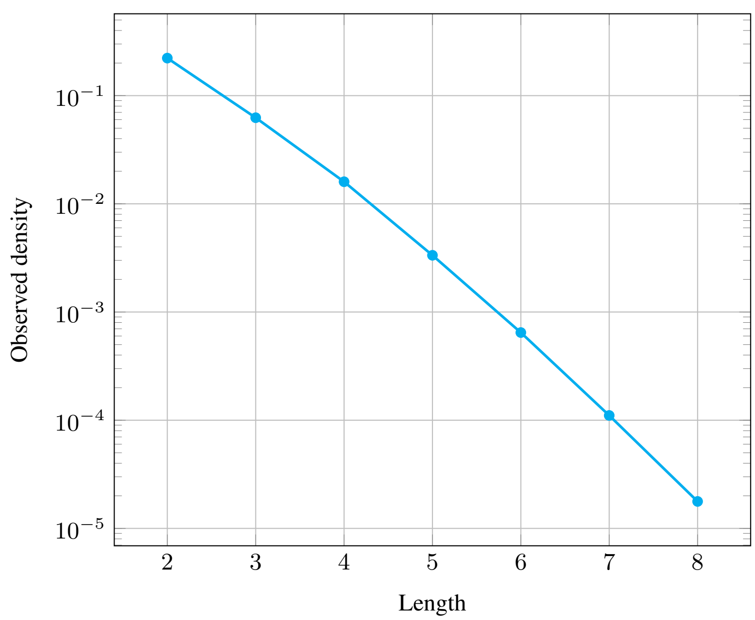
Figure 5.796: Illustrating the reformulation of the SYMMETRIC_ALLDIFFERENT_EXCEPT_0$(\langle 1\ \ 3, 2\ \ 0, 3\ \ 1, 4\ \ 0\rangle)$ constraint with a set of ELEMENT constraints: (A) the figure of the **Example** slot, and (B) the corresponding constraints

Counting

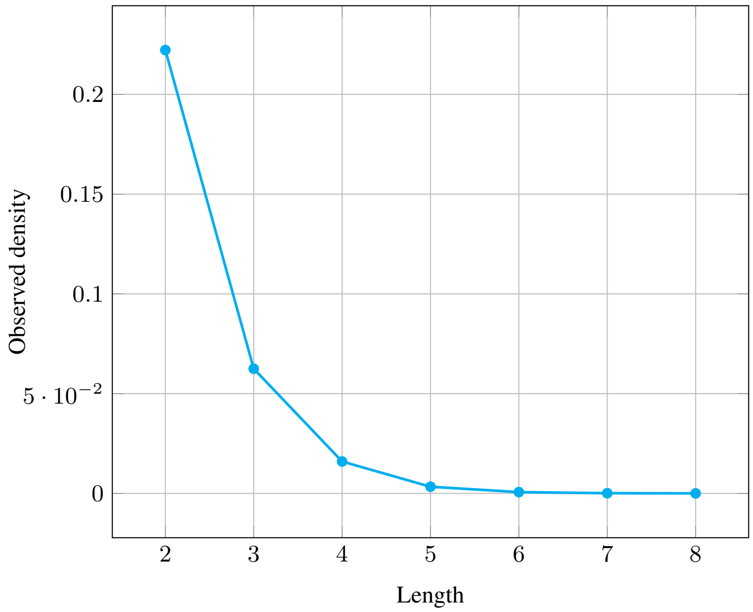
Length (n)	2	3	4	5	6	7	8
Solutions	2	4	10	26	76	232	764

Number of solutions for SYMMETRIC_ALLDIFFERENT_EXCEPT_0: domains $0..n$

Solution density for SYMMETRIC_ALLDIFFERENT_EXCEPT_0



Solution density for SYMMETRIC_ALLDIFFERENT_EXCEPT_0



See also **implied by:** SYMMETRIC_ALLDIFFERENT.

implies (items to collection): K_ALLDIFFERENT, LEX_ALLDIFFERENT.

Keywords **application area:** sport timetabling.

characteristic of a constraint: joker value.

combinatorial object: matching.

constraint type: predefined constraint, timetabling constraint.

Cond. implications SYMMETRIC_ALLDIFFERENT_EXCEPT_0(NODES)
implies ALLDIFFERENT_EXCEPT_0(VARIABLES : NODES).

5.404 SYMMETRIC_ALLDIFFERENT_LOOP

DESCRIPTION LINKS GRAPH

Origin

Derived from SYMMETRIC_ALLDIFFERENT

Constraint

SYMMETRIC_ALLDIFFERENT_LOOP(NODES)

Synonyms

SYMMETRIC_ALLDIFF_LOOP, SYMMETRIC_ALLDISTINCT_LOOP, SYMM_ALLDIFFERENT_LOOP, SYMM_ALLDIFF_LOOP, SYMM_ALLDISTINCT_LOOP.

Argument

NODES : collection(index−int, succ−dvar)

Restrictions

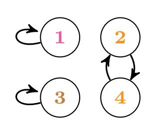
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

Purpose

All variables associated with the succ attribute of the NODES collection should be pairwise distinct. In addition enforce the following condition: if variable NODES$[i]$.succ is assigned value j then variable NODES$[j]$.succ is assigned value i. Note that i and j are not necessarily distinct. This can be interpreted as a graph-covering problem where one has to cover a digraph G with circuits of length two or one in such a way that each vertex of G belongs to a single circuit.

Example

$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{succ}-4,\\ \texttt{index}-3 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{succ}-2\end{array}\right\rangle\right)$$



The SYMMETRIC_ALLDIFFERENT_LOOP constraint holds since:

- We have two loops respectively corresponding to NODES[1].succ $=$ 1 and NODES[3].succ $= 3$.
- We have one circuit of length 2 corresponding to NODES[2].succ $= 4 \Leftrightarrow$ NODES[4].succ $= 2$.

Figure 5.797 provides a second example involving a SYMMETRIC_ALLDIFFERENT_LOOP constraint.

All solutions

Figure 5.798 gives all solutions to the following non ground instance of the SYMMETRIC_ALLDIFFERENT_LOOP constraint: $\mathtt{S}_1 \in [2,5]$, $\mathtt{S}_2 \in [1,3]$, $\mathtt{S}_3 \in [1,4]$, $\mathtt{S}_4 \in [2,4]$, $\mathtt{S}_5 \in [1,5]$, SYMMETRIC_ALLDIFFERENT_LOOP($\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4, 5\ \mathtt{S}_5\rangle$).

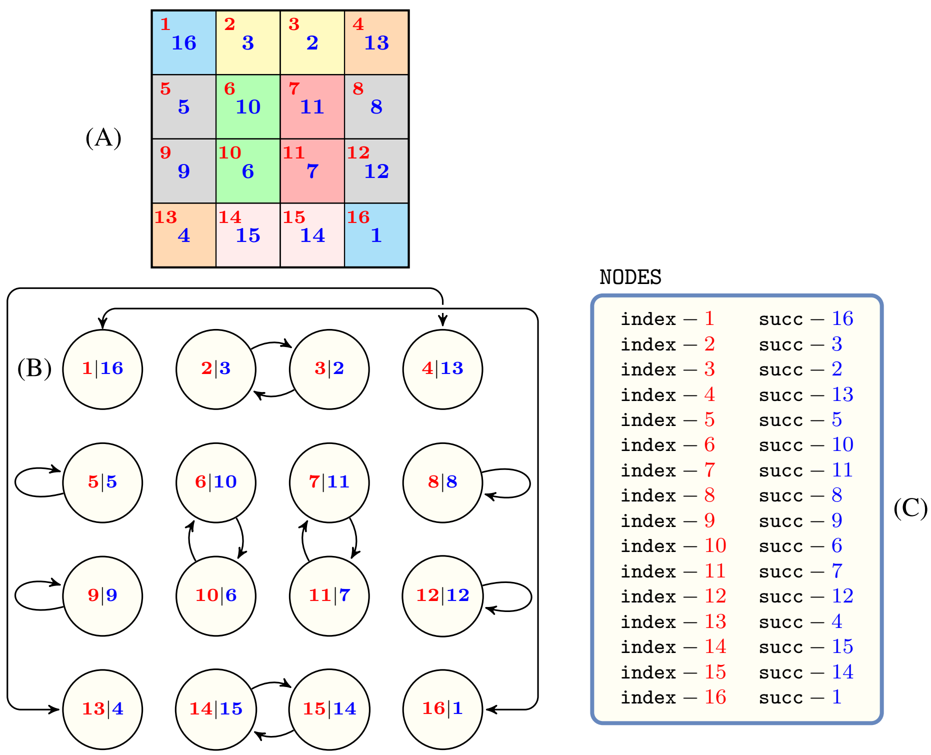


Figure 5.797: (A) Magic square Duerer where cells that belong to a same cycle are coloured identically by a colour different from grey; each cell has an *index* in its upper left corner (in red) and a *value* (in blue). (B) Corresponding graph where there is an arc from node i to node j if and only if the value of cell i is equal to the index of cell j. (C) Collection of nodes passed to the SYMMETRIC_ALLDIFFERENT_LOOP constraint: the four self-loops of the graph correspond to the four grey cells of the magic square such that the value of the cell (in blue) is equal to the index of the cell (in red).

Typical $|\mathtt{NODES}| \geq 4$

Symmetry Items of NODES are permutable.

Algorithm An arc-consistency filtering algorithm for the SYMMETRIC_ALLDIFFERENT_LOOP constraint is described in [140, 139]. The algorithm is based on the following ideas:

- First, one can map solutions to the SYMMETRIC_ALLDIFFERENT_LOOP constraint to perfect (g, f)-matchings in a non-bipartite graph derived from the domain of the variables of the constraint where $g(x) = 0$, $f(x) = 1$ for vertices x which have a self-loop, and $g(x) = f(x) = 1$ for all the remaining vertices. A *perfect* (g, f)-*matching* $\mathcal{M}$ of a graph is a subset of edges such that every vertex x is incident with the number of edges in $\mathcal{M}$ between $g(x)$ and $f(x)$.

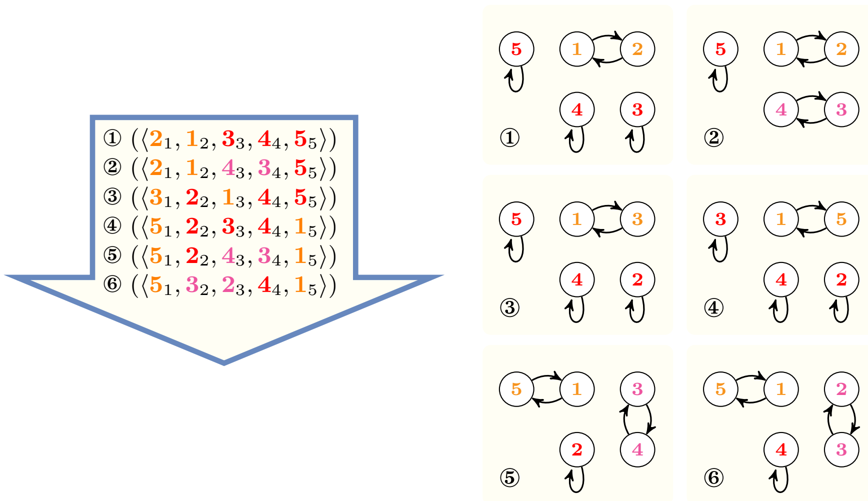


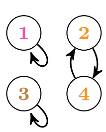
Figure 5.798: All solutions corresponding to the non ground example of the SYMMETRIC_ALLDIFFERENT_LOOP constraint of the **All solutions** slot; in the left-hand side the `index` attributes are displayed as indices of the `succ` attribute and self loops are coloured in red, while in the right-hand side the `index` attributes are directly displayed within each node.

- Second, Gallai-Edmonds decomposition [190, 159] allows us to find out all edges that do not belong any perfect (g, f)-matchings, and therefore prune the corresponding variables.

Reformulation

Let n denotes the number nodes of the `NODES` collection, and let s_i (with $i \in [1, n]$) denotes `NODES`$[i]$`.succ`. The SYMMETRIC_ALLDIFFERENT_LOOP(`NODES`) constraint can be expressed in term of a conjunction of one ALLDIFFERENT constraint and n ELEMENT constraints. ALLDIFFERENT$(\langle s_1, s_2, \ldots, s_n\rangle)$, $\forall i \in [1, n]$: ELEMENT$(s_i, \langle s_1, s_2, \ldots, s_n\rangle, i)$. Figure 5.799 illustrates this reformulation of the SYMMETRIC_ALLDIFFERENT_LOOP constraint.

(A)



(B)

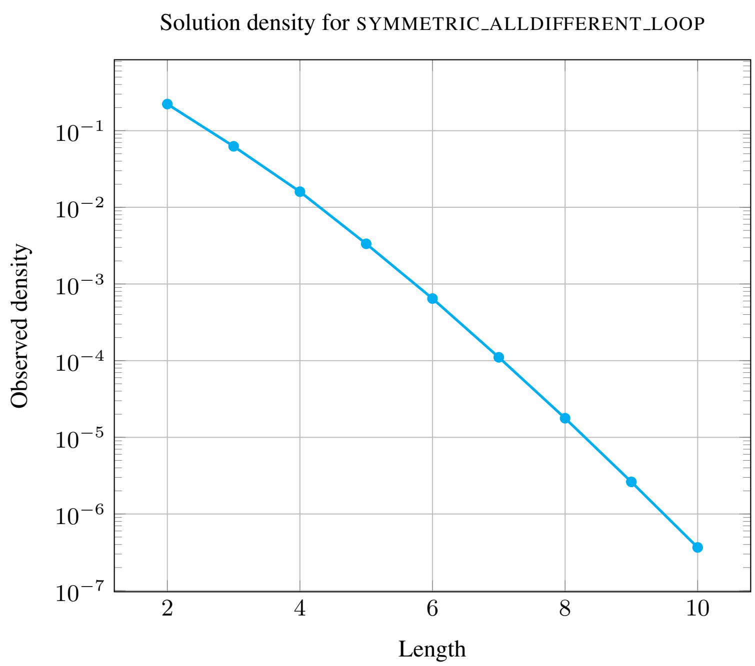
ALLDIFFERENT$(\langle 1, 4, 3, 2\rangle)$	*two vertices do not point to the same vertex*
ELEMENT$(1, \langle 1, 4, 3, 2\rangle, 1)$	*successor of vertex 1 is vertex 1, a self-loop*
ELEMENT$(4, \langle 1, 4, 3, 2\rangle, 2)$	*successor of vertex 2, vertex 4, goes back to 2*
ELEMENT$(3, \langle 1, 4, 3, 2\rangle, 3)$	*successor of vertex 3 is vertex 3, a self-loop*
ELEMENT$(2, \langle 1, 4, 3, 2\rangle, 4)$	*successor of vertex 4, vertex 2, goes back to 4*

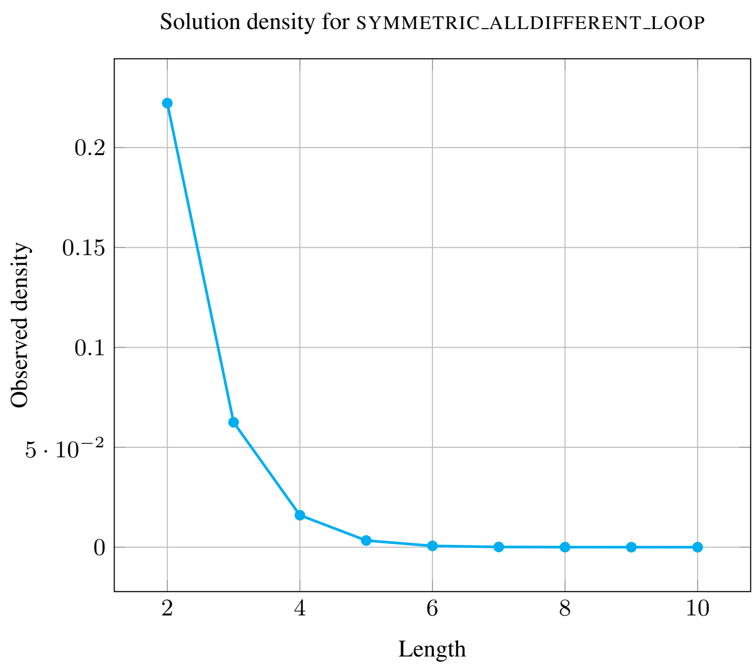
Figure 5.799: Illustrating the reformulation of the SYMMETRIC_ALLDIFFERENT_LOOP$(\langle 1\ 1, 2\ 4, 3\ 3, 4\ 2\rangle)$ constraint with a set of ELEMENT constraints: (A) the figure of the **Example** slot, and (B) the corresponding constraints

Counting

Length (n)	2	3	4	5	6	7	8	9	10
Solutions	2	4	10	26	76	232	764	2620	9496

Number of solutions for SYMMETRIC_ALLDIFFERENT_LOOP: domains $0..n$





See also

implied by: SYMMETRIC_ALLDIFFERENT.

implies: TWIN.

implies (items to collection): LEX_ALLDIFFERENT.

Keywords

characteristic of a constraint: all different, disequality.

combinatorial object: permutation, involution, matching.

constraint type: graph constraint, graph partitioning constraint.

final graph structure: circuit.

modelling: cycle.

Cond. implications

SYMMETRIC_ALLDIFFERENT_LOOP(NODES)
implies PERMUTATION(VARIABLES : NODES).

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	• nodes1.succ = nodes2.index • nodes2.succ = nodes1.index
Graph property(ies)	$\mathbf{NARC} = \vert\mathtt{NODES}\vert$

Graph model In order to express the binary constraint that links two vertices one has to make explicit the identifier of the vertices.

Parts (A) and (B) of Figure 5.800 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

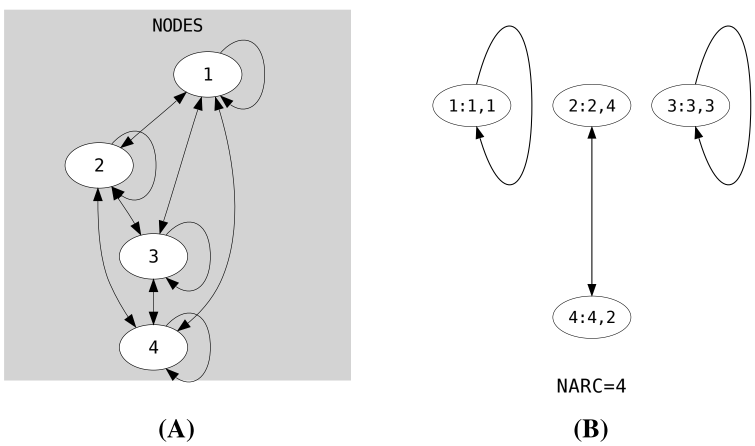


Figure 5.800: Initial and final graph of the SYMMETRIC_ALLDIFFERENT_LOOP constraint

Signature Since all the index attributes of the NODES collection are distinct, and because of the first condition nodes1.succ = nodes2.index of the arc constraint, each vertex of the final graph has at most one successor. Therefore the maximum number of arcs of the final graph is equal to the maximum number of vertices |NODES| of the final graph. So we can rewrite $\mathbf{NARC} = |\mathtt{NODES}|$ to $\mathbf{NARC} \geq |\mathtt{NODES}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.405 SYMMETRIC_CARDINALITY

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from GLOBAL_CARDINALITY by W. Kocjan.

Constraint SYMMETRIC_CARDINALITY(VARS, VALS)

Arguments

$$\begin{array}{lll}\texttt{VARS} & : & \texttt{collection}(\texttt{idvar}-\texttt{int}, \texttt{var}-\texttt{svar}, \texttt{l}-\texttt{int}, \texttt{u}-\texttt{int}) \\ \texttt{VALS} & : & \texttt{collection}(\texttt{idval}-\texttt{int}, \texttt{val}-\texttt{svar}, \texttt{l}-\texttt{int}, \texttt{u}-\texttt{int})\end{array}$$

Restrictions

$\texttt{required}(\texttt{VARS}, [\texttt{idvar}, \texttt{var}, \texttt{l}, \texttt{u}])$
$|\texttt{VARS}| \geq 1$
$\texttt{VARS.idvar} \geq 1$
$\texttt{VARS.idvar} \leq |\texttt{VARS}|$
$\texttt{distinct}(\texttt{VARS}, \texttt{idvar})$
$\texttt{VARS.l} \geq 0$
$\texttt{VARS.l} \leq \texttt{VARS.u}$
$\texttt{VARS.u} \leq |\texttt{VALS}|$
$\texttt{required}(\texttt{VALS}, [\texttt{idval}, \texttt{val}, \texttt{l}, \texttt{u}])$
$|\texttt{VALS}| \geq 1$
$\texttt{VALS.idval} \geq 1$
$\texttt{VALS.idval} \leq |\texttt{VALS}|$
$\texttt{distinct}(\texttt{VALS}, \texttt{idval})$
$\texttt{VALS.l} \geq 0$
$\texttt{VALS.l} \leq \texttt{VALS.u}$
$\texttt{VALS.u} \leq |\texttt{VARS}|$

Purpose Put in relation two sets: for each element of one set gives the corresponding elements of the other set to which it is associated. In addition, it constraints the number of elements associated with each element to be in a given interval.

Example

$$\left(\begin{array}{l}\left\langle\begin{array}{llll}\texttt{idvar}-1 & \texttt{var}-\{3\} & \texttt{l}-0 & \texttt{u}-1, \\ \texttt{idvar}-2 & \texttt{var}-\{1\} & \texttt{l}-1 & \texttt{u}-2, \\ \texttt{idvar}-3 & \texttt{var}-\{1,2\} & \texttt{l}-1 & \texttt{u}-2, \\ \texttt{idvar}-4 & \texttt{var}-\{1,3\} & \texttt{l}-2 & \texttt{u}-3\end{array}\right\rangle, \\ \left\langle\begin{array}{llll}\texttt{idval}-1 & \texttt{val}-\{2,3,4\} & \texttt{l}-3 & \texttt{u}-4, \\ \texttt{idval}-2 & \texttt{val}-\{3\} & \texttt{l}-1 & \texttt{u}-1, \\ \texttt{idval}-3 & \texttt{val}-\{1,4\} & \texttt{l}-1 & \texttt{u}-2, \\ \texttt{idval}-4 & \texttt{val}-\emptyset & \texttt{l}-0 & \texttt{u}-1\end{array}\right\rangle\end{array}\right)$$

The SYMMETRIC_CARDINALITY constraint holds since:

- $3 \in \texttt{VARS}[1].\texttt{var} \Leftrightarrow 1 \in \texttt{VALS}[3].\texttt{val}$,
- $1 \in \texttt{VARS}[2].\texttt{var} \Leftrightarrow 2 \in \texttt{VALS}[1].\texttt{val}$,
- $1 \in \texttt{VARS}[3].\texttt{var} \Leftrightarrow 3 \in \texttt{VALS}[1].\texttt{val}$,

- $2 \in$ VARS[3].var $\Leftrightarrow 3 \in$ VALS[2].val,
- $1 \in$ VARS[4].var $\Leftrightarrow 4 \in$ VALS[1].val,
- $3 \in$ VARS[4].var $\Leftrightarrow 4 \in$ VALS[3].val,
- The number of elements of VARS[1].var $= \{3\}$ belongs to interval $[0, 1]$,
- The number of elements of VARS[2].var $= \{1\}$ belongs to interval $[1, 2]$,
- The number of elements of VARS[3].var $= \{1, 2\}$ belongs to interval $[1, 2]$,
- The number of elements of VARS[4].var $= \{1, 3\}$ belongs to interval $[2, 3]$,
- The number of elements of VALS[1].val $= \{2, 3, 4\}$ belongs to interval $[3, 4]$,
- The number of elements of VALS[2].val $= \{3\}$ belongs to interval $[1, 1]$,
- The number of elements of VALS[3].val $= \{1, 4\}$ belongs to interval $[1, 2]$,
- The number of elements of VALS[4].val $= \emptyset$ belongs to interval $[0, 1]$.

Typical

|VARS| > 1
|VALS| > 1

Symmetries

- Items of VARS are permutable.
- Items of VALS are permutable.

Usage

The most simple example of applying SYMMETRIC_GCC is a variant of personnel assignment problem, where one person can be assigned to perform between n and m $(n \leq m)$ jobs, and every job requires between p and q $(p \leq q)$ persons. In addition every job requires different kind of skills. The previous problem can be modelled as follows:

- For each person we create an item of the VARS collection,
- For each job we create an item of the VALS collection,
- There is an arc between a person and the particular job if this person is qualified to perform it.

Remark

The SYMMETRIC_GCC constraint generalises the GLOBAL_CARDINALITY constraint by allowing a variable to take more than one value.

Algorithm

A first flow-based arc-consistency algorithm for the SYMMETRIC_CARDINALITY constraint is described in [252]. A second arc-consistency filtering algorithm exploiting matching theory [157] is described in [138, 139].

See also

common keyword: LINK_SET_TO_BOOLEANS *(constraint involving set variables)*.

generalisation: SYMMETRIC_GCC (fixed interval *replaced by* variable).

root concept: GLOBAL_CARDINALITY.

used in graph description: IN_SET.

Keywords

application area: assignment.

combinatorial object: relation.

constraint arguments: constraint involving set variables.

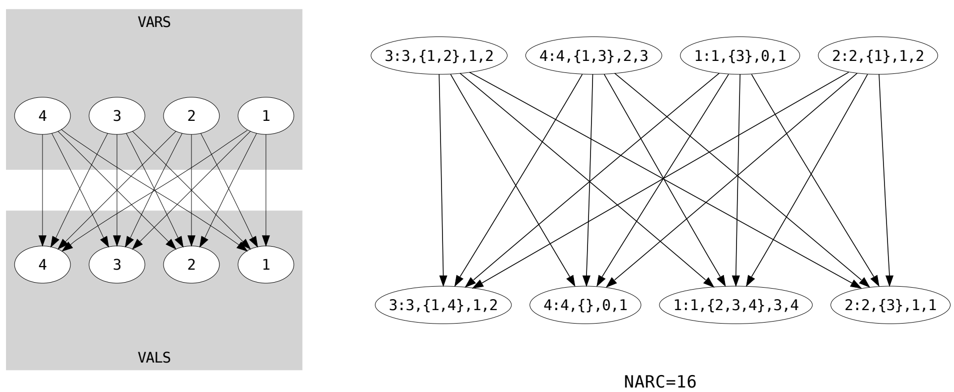
constraint type: decomposition, timetabling constraint.

filtering: flow, bipartite matching.

Arc input(s)	VARS VALS
Arc generator	$PRODUCT \mapsto$ collection(vars, vals)
Arc arity	2
Arc constraint(s)	• IN_SET(vars.idvar, vals.val) $\Leftrightarrow$ IN_SET(vals.idval, vars.var) • vars.l $\leq$ card_set(vars.var) • vars.u $\geq$ card_set(vars.var) • vals.l $\leq$ card_set(vals.val) • vals.u $\geq$ card_set(vals.val)
Graph property(ies)	**NARC** $= \|$VARS$\| * \|$VALS$\|$

Graph model The graph model used for the SYMMETRIC_CARDINALITY is similar to the one used in the DOMAIN_CONSTRAINT or in the LINK_SET_TO_BOOLEANS constraints: we use an equivalence in the arc constraint and ask all arc constraints to hold.

Parts (A) and (B) of Figure 5.801 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, all the arcs of the final graph are stressed in bold.



(A) **(B)**

Figure 5.801: Initial and final graph of the SYMMETRIC_CARDINALITY constraint

Signature Since we use the $PRODUCT$ arc generator on the collections VARS and VALS, the number of arcs of the initial graph is equal to $|$VARS$| \cdot |$VALS$|$. Therefore the maximum number of arcs of the final graph is also equal to $|$VARS$| \cdot |$VALS$|$ and we can rewrite $\mathbf{NARC} = |$VARS$| \cdot |$VALS$|$ to $\mathbf{NARC} \geq |$VARS$| \cdot |$VALS$|$. So we can simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.406 SYMMETRIC_GCC

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from GLOBAL_CARDINALITY by W. Kocjan.

Constraint SYMMETRIC_GCC(VARS, VALS)

Synonym SGCC.

Arguments

$$\begin{array}{lll} \mathtt{VARS} & : & \mathtt{collection}(\mathtt{idvar}-\mathtt{int}, \mathtt{var}-\mathtt{svar}, \mathtt{nocc}-\mathtt{dvar}) \\ \mathtt{VALS} & : & \mathtt{collection}(\mathtt{idval}-\mathtt{int}, \mathtt{val}-\mathtt{svar}, \mathtt{nocc}-\mathtt{dvar}) \end{array}$$

Restrictions

$\mathtt{required}(\mathtt{VARS}, [\mathtt{idvar}, \mathtt{var}, \mathtt{nocc}])$
$|\mathtt{VARS}| \geq 1$
$\mathtt{VARS.idvar} \geq 1$
$\mathtt{VARS.idvar} \leq |\mathtt{VARS}|$
$\mathtt{distinct}(\mathtt{VARS}, \mathtt{idvar})$
$\mathtt{VARS.nocc} \geq 0$
$\mathtt{VARS.nocc} \leq |\mathtt{VALS}|$
$\mathtt{required}(\mathtt{VALS}, [\mathtt{idval}, \mathtt{val}, \mathtt{nocc}])$
$|\mathtt{VALS}| \geq 1$
$\mathtt{VALS.idval} \geq 1$
$\mathtt{VALS.idval} \leq |\mathtt{VALS}|$
$\mathtt{distinct}(\mathtt{VALS}, \mathtt{idval})$
$\mathtt{VALS.nocc} \geq 0$
$\mathtt{VALS.nocc} \leq |\mathtt{VARS}|$

Purpose Put in relation two sets: for each element of one set gives the corresponding elements of the other set to which it is associated. In addition, enforce a cardinality constraint on the number of occurrences of each value.

Example

$$\left(\begin{array}{l} \left\langle\begin{array}{lll} \mathtt{idvar}-1 & \mathtt{var}-\{3\} & \mathtt{nocc}-1, \\ \mathtt{idvar}-2 & \mathtt{var}-\{1\} & \mathtt{nocc}-1, \\ \mathtt{idvar}-3 & \mathtt{var}-\{1,2\} & \mathtt{nocc}-2, \\ \mathtt{idvar}-4 & \mathtt{var}-\{1,3\} & \mathtt{nocc}-2 \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \mathtt{idval}-1 & \mathtt{val}-\{2,3,4\} & \mathtt{nocc}-3, \\ \mathtt{idval}-2 & \mathtt{val}-\{3\} & \mathtt{nocc}-1, \\ \mathtt{idval}-3 & \mathtt{val}-\{1,4\} & \mathtt{nocc}-2, \\ \mathtt{idval}-4 & \mathtt{val}-\emptyset & \mathtt{nocc}-0 \end{array}\right\rangle \end{array}\right)$$

The SYMMETRIC_GCC constraint holds since:

- $3 \in \mathtt{VARS}[1].\mathtt{var} \Leftrightarrow 1 \in \mathtt{VALS}[3].\mathtt{val}$,
- $1 \in \mathtt{VARS}[2].\mathtt{var} \Leftrightarrow 2 \in \mathtt{VALS}[1].\mathtt{val}$,
- $1 \in \mathtt{VARS}[3].\mathtt{var} \Leftrightarrow 3 \in \mathtt{VALS}[1].\mathtt{val}$,

- $2 \in$ VARS[3].var $\Leftrightarrow 3 \in$ VALS[2].val,
- $1 \in$ VARS[4].var $\Leftrightarrow 4 \in$ VALS[1].val,
- $3 \in$ VARS[4].var $\Leftrightarrow 4 \in$ VALS[3].val,
- The number of elements of VARS[1].var $= \{3\}$ is equal to 1,
- The number of elements of VARS[2].var $= \{1\}$ is equal to 1,
- The number of elements of VARS[3].var $= \{1, 2\}$ is equal to 2,
- The number of elements of VARS[4].var $= \{1, 3\}$ is equal to 2,
- The number of elements of VALS[1].val $= \{2, 3, 4\}$ is equal to 3,
- The number of elements of VALS[2].val $= \{3\}$ is equal to 1,
- The number of elements of VALS[3].val $= \{1, 4\}$ is equal to 2,
- The number of elements of VALS[4].val $= \emptyset$ is equal to 0.

Typical

|VARS| > 1
|VALS| > 1

Symmetries

- Items of VARS are permutable.
- Items of VALS are permutable.

Usage

The most simple example of applying SYMMETRIC_GCC is a variant of personnel assignment problem, where one person can be assigned to perform between n and m $(n \leq m)$ jobs, and every job requires between p and q $(p \leq q)$ persons. In addition every job requires different kind of skills. The previous problem can be modelled as follows:

- For each person we create an item of the VARS collection,
- For each job we create an item of the VALS collection,
- There is an arc between a person and the particular job if this person is qualified to perform it.

Remark

The SYMMETRIC_GCC constraint generalises the GLOBAL_CARDINALITY constraint by allowing a variable to take more than one value. It corresponds to a variant of the SYMMETRIC_CARDINALITY constraint described in [252] where the occurrence variables of the VARS and VALS collections are replaced by fixed intervals.

See also

common keyword: LINK_SET_TO_BOOLEANS *(constraint involving set variables)*.

root concept: GLOBAL_CARDINALITY.

specialisation: SYMMETRIC_CARDINALITY (`variable` *replaced by* `fixed interval`).

used in graph description: IN_SET.

Keywords

application area: assignment.

combinatorial object: relation.

constraint arguments: constraint involving set variables.

constraint type: decomposition, timetabling constraint.

filtering: flow.

Arc input(s)	VARS VALS
Arc generator	$PRODUCT$ $\mapsto$ collection(vars, vals)
Arc arity	2
Arc constraint(s)	• IN_SET(vars.idvar, vals.val) $\Leftrightarrow$ IN_SET(vals.idval, vars.var) • vars.nocc = card_set(vars.var) • vals.nocc = card_set(vals.val)
Graph property(ies)	**NARC** = |VARS| $*$ |VALS|

Graph model The graph model used for the SYMMETRIC_GCC is similar to the one used in the DOMAIN_CONSTRAINT or in the LINK_SET_TO_BOOLEANS constraints: we use an equivalence in the arc constraint and ask all arc constraints to hold.

Parts (A) and (B) of Figure 5.802 respectively show the initial and final graph. Since we use the **NARC** graph property, all the arcs of the final graph are stressed in bold.

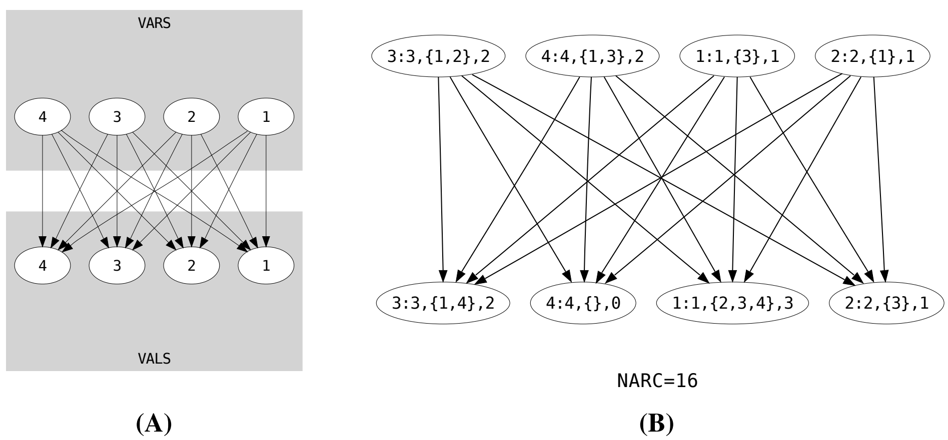


Figure 5.802: Initial and final graph of the SYMMETRIC_GCC constraint

Signature Since we use the $PRODUCT$ arc generator on the collections VARS and VALS, the number of arcs of the initial graph is equal to $|\mathtt{VARS}| \cdot |\mathtt{VALS}|$. Therefore the maximum number of arcs of the final graph is also equal to $|\mathtt{VARS}| \cdot |\mathtt{VALS}|$ and we can rewrite $\mathbf{NARC} = |\mathtt{VARS}| \cdot |\mathtt{VALS}|$ to $\mathbf{NARC} \geq |\mathtt{VARS}| \cdot |\mathtt{VALS}|$. So we can simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

5.407 TASKS_INTERSECTION

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin

Inspired by video summarization.

Constraint

TASKS_INTERSECTION(INTERSECTION, TASKS1, TASKS2)

Synonyms

INTERSECTION_BETWEEN_SEQUENCES_OF_TASKS, INTERSECTION_BETWEEN_INTERVALS, INTERSECTION_BETWEEN_TASKS_CHAINS.

Arguments

INTERSECTION	:	dvar
TASKS1	:	collection(origin−dvar, duration−dvar, end−dvar)
TASKS2	:	collection(origin−dvar, duration−dvar, end−dvar)

Restrictions

INTERSECTION ≥ 0
require_at_least(2, TASKS1, [origin, duration, end])
require_at_least(2, TASKS2, [origin, duration, end])
TASKS1.duration ≥ 0
TASKS2.duration ≥ 0
TASKS1.origin $\leq$ TASKS1.end
TASKS2.origin $\leq$ TASKS2.end
INTERSECTION $\leq$sum(TASKS1.duration)
INTERSECTION $\leq$sum(TASKS2.duration)

Purpose

INTERSECTION is the intersection between two collections of ordered tasks TASKS1 and TASKS2:

1. $\forall s \in [1, |\texttt{TASKS1}|] : \texttt{TASKS1}[s].\texttt{end} = \texttt{TASKS1}[s].\texttt{origin} + \texttt{TASKS1}[s].\texttt{duration}$,
2. $\forall t \in [1, |\texttt{TASKS2}|] : \texttt{TASKS2}[t].\texttt{end} = \texttt{TASKS2}[t].\texttt{origin} + \texttt{TASKS2}[t].\texttt{duration}$,
3. $\forall s \in [1, |\texttt{TASKS1}| - 1] : \texttt{TASKS1}[s].\texttt{end} \leq \texttt{TASKS1}[s+1].\texttt{origin}$,
4. $\forall t \in [1, |\texttt{TASKS2}| - 1] : \texttt{TASKS2}[t].\texttt{end} \leq \texttt{TASKS2}[t+1].\texttt{origin}$,
5. $\texttt{INTERSECTION} = \sum_{\substack{s \in [1,|\texttt{TASKS1}|] \\ t \in [1,|\texttt{TASKS2}|]}} \max(\beta_{s,t} - \alpha_{s,t}, 0)$ with

$$\alpha_{s,t} = \max\begin{pmatrix} \texttt{TASKS1}[s].\texttt{origin}, \\ \texttt{TASKS2}[t].\texttt{origin} \end{pmatrix}, \quad \beta_{s,t} = \min\begin{pmatrix} \texttt{TASKS1}[s].\texttt{end}, \\ \texttt{TASKS2}[t].\texttt{end} \end{pmatrix}.$$

Example

$$\left(\begin{array}{l} 3, \left\langle \begin{array}{lll} \texttt{origin}-2 & \texttt{duration}-2 & \texttt{end}-4, \\ \texttt{origin}-7 & \texttt{duration}-2 & \texttt{end}-9, \\ \texttt{origin}-9 & \texttt{duration}-0 & \texttt{end}-9 \end{array} \right\rangle, \\ \left\langle \begin{array}{lll} \texttt{origin}-1 & \texttt{duration}-3 & \texttt{end}-4, \\ \texttt{origin}-5 & \texttt{duration}-1 & \texttt{end}-6, \\ \texttt{origin}-8 & \texttt{duration}-2 & \texttt{end}-10 \end{array} \right\rangle \end{array}\right)$$

As illustrated by Figure 5.803, the constraint holds since:

- The first task of TASKS1 is included within the first task of TASKS2 and therefore contributes from its total duration 2 to the overall intersection.
- While the second task of TASKS1 does not intersect the first and second tasks of TASKS2, it has a non empty intersection of 1 with the third task of TASKS2.
- The third task of TASKS1 does not contribute to the overall intersection since its duration is equal to zero.

The overall intersection 3 is equal to $2 + 1 + 0$.

TASKS1

①	o − 2	d − 2	e − 4
②	o − 7	d − 2	e − 9
③	o − 9	d − 0	e − 9

(o for origin, d for duration, e for end)

TASKS2

(a)	o − 1	d − 3	e − 4
(b)	o − 5	d − 1	e − 6
(c)	o − 8	d − 2	e − 10

(o for origin, d for duration, e for end)

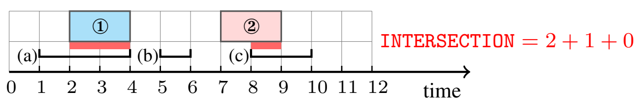


Figure 5.803: The TASKS_INTERSECTION solution to the **Example** slot

All solutions Figure 5.804 gives all solutions to the following non ground instance of the TASKS_INTERSECTION constraint: $\mathtt{O}_1 \in [0,1]$, $\mathtt{D}_1 \in [0,6]$, $\mathtt{E}_1 \in [3,5]$, $\mathtt{O}_2 \in [0,6]$, $\mathtt{D}_2 \in [1,3]$, $\mathtt{E}_2 \in [0,9]$, TASKS_INTERSECTION(**2**, $\langle \mathtt{O}_1\ \mathtt{D}_1\ \mathtt{E}_1,\ \mathtt{O}_2\ \mathtt{D}_2\ \mathtt{E}_2\rangle$, $\langle$**1 3 4**, **5 1 6**, **8 2 10**$\rangle$).

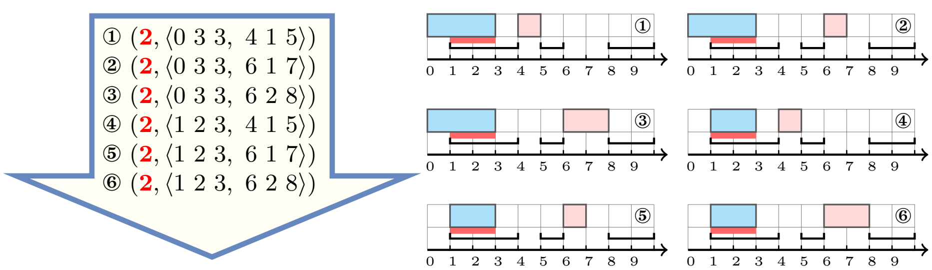


Figure 5.804: All solutions corresponding to the non ground example of the TASKS_INTERSECTION constraint of the **All solutions** slot

Typical

$\mathtt{INTERSECTION} > 0$
$|\mathtt{TASKS1}| > 1$
$|\mathtt{TASKS2}| > 1$
range(TASKS1.duration) > 1
range(TASKS2.duration) > 1

Arg. properties Functional dependency: INTERSECTION determined by TASKS1 and TASKS2.

Keywords

constraint type: predefined constraint, scheduling constraint.

filtering: minimum task duration.

modelling: zero-duration task.

5.408 TEMPORAL_PATH

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin ILOG

Constraint TEMPORAL_PATH(NPATH, NODES)

Arguments

$$\begin{array}{lll} \texttt{NPATH} & : & \texttt{dvar} \\ \texttt{NODES} & : & \texttt{collection}\left(\begin{array}{l} \texttt{index}-\texttt{int}, \\ \texttt{succ}-\texttt{dvar}, \\ \texttt{start}-\texttt{dvar}, \\ \texttt{end}-\texttt{dvar} \end{array}\right) \end{array}$$

Restrictions

$\texttt{NPATH} \geq 1$
$\texttt{NPATH} \leq |\texttt{NODES}|$
$\texttt{required}(\texttt{NODES}, [\texttt{index}, \texttt{succ}, \texttt{start}, \texttt{end}])$
$|\texttt{NODES}| > 0$
$\texttt{NODES.index} \geq 1$
$\texttt{NODES.index} \leq |\texttt{NODES}|$
$\texttt{distinct}(\texttt{NODES}, \texttt{index})$
$\texttt{NODES.succ} \geq 1$
$\texttt{NODES.succ} \leq |\texttt{NODES}|$
$\texttt{NODES.start} \leq \texttt{NODES.end}$

Purpose

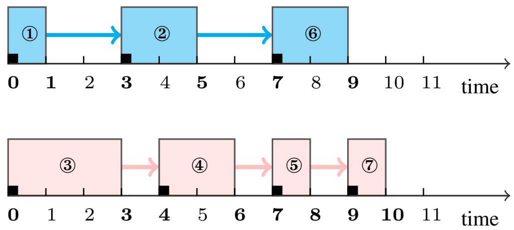
Let G be the digraph described by the NODES collection. Partition G with a set of disjoint paths such that each vertex of the graph belongs to a single path. In addition, for all pairs of consecutive vertices of a path we have a precedence constraint that enforces the end associated with the first vertex to be less than or equal to the start related to the second vertex.

Example

$$\left(2, \left\langle \begin{array}{llll} \texttt{index}-1 & \texttt{succ}-2 & \texttt{start}-0 & \texttt{end}-1, \\ \texttt{index}-2 & \texttt{succ}-6 & \texttt{start}-3 & \texttt{end}-5, \\ \texttt{index}-3 & \texttt{succ}-4 & \texttt{start}-0 & \texttt{end}-3, \\ \texttt{index}-4 & \texttt{succ}-5 & \texttt{start}-4 & \texttt{end}-6, \\ \texttt{index}-5 & \texttt{succ}-7 & \texttt{start}-7 & \texttt{end}-8, \\ \texttt{index}-6 & \texttt{succ}-6 & \texttt{start}-7 & \texttt{end}-9, \\ \texttt{index}-7 & \texttt{succ}-7 & \texttt{start}-9 & \texttt{end}-10 \end{array} \right\rangle \right)$$

The TEMPORAL_PATH constraint holds since:

- The items of the NODES collection represent the two (NPATH $= 2$) paths $1 \rightarrow 2 \rightarrow 6$ and $3 \rightarrow 4 \rightarrow 5 \rightarrow 7$.
- As illustrated by Figure 5.805, all precedences between adjacent vertices of a same path hold: each item i ($1 \leq i \leq 7$) of the NODES collection is represented by a rectangle starting and ending at instants NODES$[i]$.start and NODES$[i]$.end; the number within each rectangle designates the index of the corresponding item within the NODES collection.



①	index − 1	succ − 2	start − 0	end − 1
②	index − 2	succ − 6	start − 3	end − 5
③	index − 3	succ − 4	start − 0	end − 3
④	index − 4	succ − 5	start − 4	end − 6
⑤	index − 5	succ − 7	start − 7	end − 8
⑥	index − 6	succ − 6	start − 7	end − 9
⑦	index − 7	succ − 7	start − 9	end − 10

NODES

Figure 5.805: The two paths of the **Example** slot represented as two sequences of tasks

Typical

NPATH $<$ |NODES|
|NODES| >1
NODES.start $<$ NODES.end

Symmetries

- Items of NODES are permutable.
- One and the same constant can be added to the start and end attributes of all items of NODES.

Arg. properties

Functional dependency: NPATH determined by NODES.

Remark

This constraint is related to the PATH constraint of **Ilog Solver**. It can also be directly expressed with the CYCLE [47] constraint of **CHIP** by using the *diff nodes* and the origin parameters. A generic model based on linear programming that handles paths, trees and cycles is presented in [255].

Reformulation

The TEMPORAL_PATH(NPATH, NODES) constraint can be expressed in term of a conjunction of one PATH constraint, |NODES| ELEMENT constraints, and |NODES| inequalities constraints:

- We pass to the PATH constraint the number of path variable NPATH as well as the items of the NODES collection form which we remove the start and end attributes.
- To the i-th ($1 \leq i \leq$ |NODES|) item of the NODES collection, we create a variable $Start_{succ_i}$ and an ELEMENT(NODES[i].succ, $\langle T_{i,1}, T_{i,2}, \ldots, T_{i,\text{NODES}}\rangle$, $Start_{succ_i}$) constraint, where $T_{i,j}$ = NODES[i].start if $i \neq j$ and $T_{i,i}$ = NODES[i].end otherwise.

- Finaly to the i-th $(1 \leq i \leq |\texttt{NODES}|)$ item of the NODES collection, we also create an inequality constraint $\texttt{NODES}[i].\texttt{end} \leq Start_{succ_i}$. Note that, since $T_{i,i}$ was initialised to $\texttt{NODES}[i].\texttt{end}$, the inequality $\texttt{NODES}[i].\texttt{end} \leq T_{i,j}$ holds when $i = j$.

With respect to the **Example** slot we get the following conjunction of constraints:

PATH$(2, \langle \texttt{index}-1\ \texttt{succ}-2, \texttt{index}-2\ \texttt{succ}-6, \texttt{index}-3\ \texttt{succ}-4,$
$\texttt{index}-4\ \texttt{succ}-5, \texttt{index}-5\ \texttt{succ}-7, \texttt{index}-6\ \texttt{succ}-6,$
$\texttt{index}-7\ \texttt{succ}-7\rangle),$
ELEMENT$(2, \langle 1,3,0,4,7,7,9\rangle, 3),$
ELEMENT$(6, \langle 1,5,0,4,7,7,9\rangle, 7),$
ELEMENT$(4, \langle 1,5,3,4,7,7,9\rangle, 4),$
ELEMENT$(5, \langle 1,5,3,6,7,7,9\rangle, 7),$
ELEMENT$(7, \langle 1,5,3,6,8,7,9\rangle, 9),$
ELEMENT$(6, \langle 1,5,3,6,8,9,9\rangle, 9),$
ELEMENT$(7, \langle 1,5,3,6,8,9,10\rangle, 10),$
$1 \leq 3, 5 \leq 7, 3 \leq 4, 6 \leq 7, 8 \leq 9, 9 \leq 9, 10 \leq 10.$

See also

common keyword: PATH_FROM_TO *(path)*.

implies (items to collection): ATLEAST_NVECTOR.

specialisation: PATH *(time dimension removed)*.

Keywords

combinatorial object: path.

constraint type: graph constraint, graph partitioning constraint.

final graph structure: connected component.

modelling: sequence dependent set-up, functional dependency.

modelling exercises: sequence dependent set-up.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	• nodes1.succ $=$ nodes2.index • nodes1.succ $=$ nodes1.index $\vee$ nodes1.end $\leq$ nodes2.start • nodes1.start $\leq$ nodes1.end • nodes2.start $\leq$ nodes2.end
Graph property(ies)	• **MAX_ID** ≤ 1 • **NCC** $=$ NPATH • **NVERTEX** $=$ |NODES|

Graph model The arc constraint is a conjunction of four conditions that respectively correspond to:

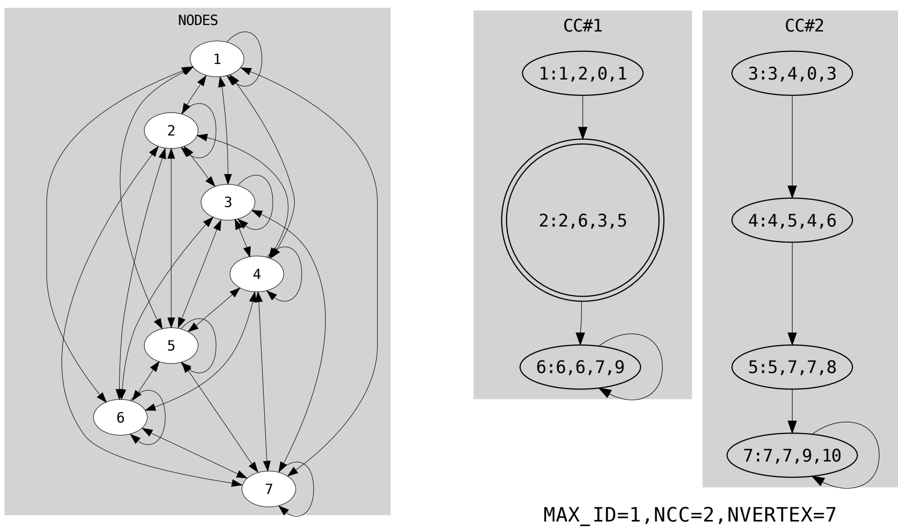
- A constraint that links the successor variable of a first vertex to the index attribute of a second vertex,
- A precedence constraint that applies on one vertex and its distinct successor,
- One precedence constraint between the start and the end of the vertex that corresponds to the departure of an arc,
- One precedence constraint between the start and the end of the vertex that corresponds to the arrival of an arc.

We use the following three graph properties in order to enforce the partitioning of the graph in distinct paths:

- The first property **MAX_ID** ≤ 1 enforces that each vertex has no more than one predecessor (**MAX_ID** does not consider loops),
- The second property **NCC** $=$ NPATH ensures that we have the required number of paths,
- The third property **NVERTEX** $=$ |NODES| enforces that, for each vertex, the start is not located after the end.

Parts (A) and (B) of Figure 5.806 respectively show the initial and final graph associated with the **Example** slot. Since we use the **MAX_ID**, the **NCC** and the **NVERTEX** graph properties we display the following information on the final graph:

- We show with a double circle a vertex that has the maximum number of predecessors.
- We show the two connected components corresponding to the two paths.
- We put in bold the vertices.



(A) **(B)**

Figure 5.806: Initial and final graph of the TEMPORAL_PATH constraint

5.409 TOUR

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

[6]

Constraint

TOUR(NODES)

Synonyms

ATOUR, CYCLE.

Argument

NODES : collection(index−int, succ−svar)

Restrictions

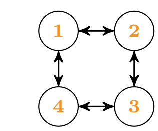
|NODES| ≥ 3
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)

Purpose

Enforce to cover an undirected graph G described by the NODES collection with a Hamiltonian cycle.

Example

$$\left(\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-\{2,4\},\\ \texttt{index}-2 & \texttt{succ}-\{1,3\},\\ \texttt{index}-3 & \texttt{succ}-\{2,4\},\\ \texttt{index}-4 & \texttt{succ}-\{1,3\}\end{array}\right\rangle\right)$$



The TOUR constraint holds since its NODES argument depicts the following Hamiltonian cycle visiting successively the vertices 1, 2, 3 and 4.

Symmetry

Items of NODES are permutable.

Algorithm

When the number of vertices is odd (i.e., |NODES| is odd) a necessary condition is that the graph is not bipartite. Other necessary conditions for filtering the TOUR constraint are given in [140, 139].

See also

common keyword: CIRCUIT *(graph partitioning constraint,Hamiltonian)*, CYCLE *(graph constraint)*, LINK_SET_TO_BOOLEANS *(constraint involving set variables)*.

used in graph description: IN_SET.

Keywords

characteristic of a constraint: undirected graph.

combinatorial object: matching.

constraint arguments: constraint involving set variables.

constraint type: graph constraint.

filtering: DFS-bottleneck, linear programming.

problems: Hamiltonian.

Arc input(s)	NODES
Arc generator	$CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.succ) $\Leftrightarrow$ IN_SET(nodes1.index, nodes2.succ)
Graph property(ies)	**NARC**$= \|$NODES$\| * \|$NODES$\| - \|$NODES$\|$

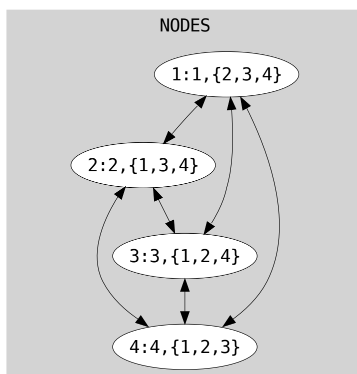
Arc input(s)	NODES
Arc generator	$CLIQUE(\neq) \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	IN_SET(nodes2.index, nodes1.succ)
Graph property(ies)	• **MIN_NSCC**$= \|$NODES$\|$ • **MIN_ID**$= 2$ • **MAX_ID**$= 2$ • **MIN_OD**$= 2$ • **MAX_OD**$= 2$

Graph model The first graph property enforces the subsequent condition: If we have an arc from the i^{th} vertex to the j^{th} vertex then we have also an arc from the j^{th} vertex to the i^{th} vertex. The second graph property enforces the following constraints:

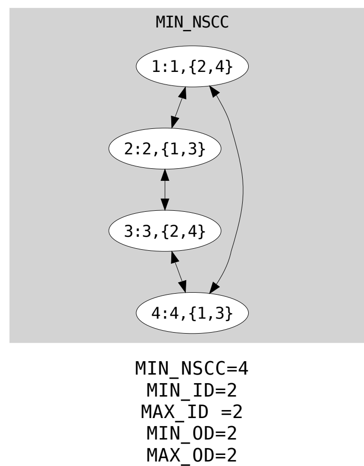
- We have one strongly connected component containing |NODES| vertices,
- Each vertex has exactly two predecessors and two successors.

Part (A) of Figure 5.807 shows the initial graph from which we start. It is derived from the set associated with each vertex. Each set describes the potential values of the succ attribute of a given vertex. Part (B) of Figure 5.807 gives the final graph associated with the **Example** slot. The TOUR constraint holds since the final graph corresponds to a Hamiltonian cycle.

Signature Since the maximum number of vertices of the final graph is equal to |NODES|, we can rewrite the graph property **MIN_NSCC** $=$ |NODES| to **MIN_NSCC** $\geq$ |NODES| and simplify $\underline{\textbf{MIN\_NSCC}}$ to $\overline{\textbf{MIN\_NSCC}}$.



(A)



(B)

Figure 5.807: Initial and final graph of the TOUR set constraint

5.410 TRACK

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin [285]

Constraint TRACK(NTRAIL, TASKS)

Arguments

NTRAIL : int
TASKS : collection(trail−int, origin−dvar, end−dvar)

Restrictions

$\mathtt{NTRAIL} > 0$
$\mathtt{NTRAIL} \leq |\mathtt{TASKS}|$
$|\mathtt{TASKS}| > 0$
required(TASKS, [trail, origin, end])
$\mathtt{TASKS.origin} \leq \mathtt{TASKS.end}$

Purpose

The TRACK constraint forces that, at each point in time overlapped by at least one task, the number of distinct values of the trail attribute of the set of tasks that overlap that point, is equal to NTRAIL.

Example

$$\left(2, \left\langle \begin{array}{lll} \mathtt{trail}-1 & \mathtt{origin}-1 & \mathtt{end}-2, \\ \mathtt{trail}-2 & \mathtt{origin}-1 & \mathtt{end}-2, \\ \mathtt{trail}-1 & \mathtt{origin}-2 & \mathtt{end}-4, \\ \mathtt{trail}-2 & \mathtt{origin}-2 & \mathtt{end}-3, \\ \mathtt{trail}-2 & \mathtt{origin}-3 & \mathtt{end}-4 \end{array} \right\rangle \right)$$

Figure 5.808 represents the tasks of the example: to the i^{th} task of the TASKS collection corresponds a rectangle labelled by i. The TRACK constraint holds since:

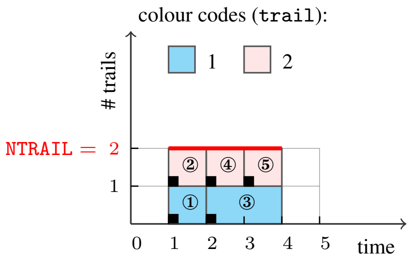
- The first and second tasks both overlap instant 1 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 1.
- The third and fourth tasks both overlap instant 2 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 2.
- The third and fifth tasks both overlap instant 3 and have a respective trail of 1 and 2. This makes two distinct values for the trail attribute at instant 3.

Typical

$\mathtt{NTRAIL} < |\mathtt{TASKS}|$
$|\mathtt{TASKS}| > 1$
$\mathrm{range}(\mathtt{TASKS.trail}) > 1$
$\mathtt{TASKS.origin} < \mathtt{TASKS.end}$

Symmetries

- Items of TASKS are permutable.
- All occurrences of two distinct values of TASKS.trail can be swapped; all occurrences of a value of TASKS.trail can be renamed to any unused value.
- One and the same constant can be added to the origin and end attributes of all items of TASKS.



TASKS

①	trail − 1	origin − 1	end − 2
②	trail − 2	origin − 1	end − 2
③	trail − 1	origin − 2	end − 4
④	trail − 2	origin − 2	end − 3
⑤	trail − 2	origin − 3	end − 4

Figure 5.808: The tasks associated with the example of the **Example** slot, at each instant we have two distinct values for the trail attribute (NTRAIL $= 2$)

Reformulation The TRACK constraint can be expressed in term of a set of reified constraints and of $2 \cdot |\texttt{TASKS}|$ NVALUE constraints:

1. For each pair of tasks $\texttt{TASKS}[i], \texttt{TASKS}[j]$ $(i, j \in [1, |\texttt{TASKS}|])$ of the TASKS collection we create a variable $T_{ij}^{\texttt{origin}}$ which is set to the trail attribute of task $\texttt{TASKS}[j]$ if task $\texttt{TASKS}[j]$ overlaps the origin attribute of task $\texttt{TASKS}[i]$, and to the trail attribute of task $\texttt{TASKS}[i]$ otherwise:
   - If $i = j$:
     - $T_{ij}^{\texttt{origin}} = \texttt{TASKS}[i].\texttt{trail}$.
   - If $i \neq j$:
     - $T_{ij}^{\texttt{origin}} = \texttt{TASKS}[i].\texttt{trail} \vee T_{ij}^{\texttt{origin}} = \texttt{TASKS}[j].\texttt{trail}$.
     - $((\texttt{TASKS}[j].\texttt{origin} \leq \texttt{TASKS}[i].\texttt{origin} \wedge$
       $\texttt{TASKS}[j].\texttt{end} > \texttt{TASKS}[i].\texttt{origin}) \wedge (T_{ij}^{\texttt{origin}} = \texttt{TASKS}[j].\texttt{trail})) \vee$
       $((\texttt{TASKS}[j].\texttt{origin} > \texttt{TASKS}[i].\texttt{origin} \vee$
       $\texttt{TASKS}[j].\texttt{end} \leq \texttt{TASKS}[i].\texttt{origin}) \wedge (T_{ij}^{\texttt{origin}} = \texttt{TASKS}[i].\texttt{trail}))$
2. For each task $\texttt{TASKS}[i]$ $(i \in [1, |\texttt{TASKS}|])$ we impose the number of distinct trails associated with the tasks that overlap the origin of task $\texttt{TASKS}[i]$ ($\texttt{TASKS}[i]$ overlaps its own origin) to be equal to NTRAIL:
   NVALUE$(\texttt{NTRAIL}, \langle T_{i1}^{\texttt{origin}}, T_{i2}^{\texttt{origin}}, \ldots, T_{i|\texttt{TASKS}|}^{\texttt{origin}} \rangle)$.
3. For each pair of tasks $\texttt{TASKS}[i], \texttt{TASKS}[j]$ $(i, j \in [1, |\texttt{TASKS}|])$ of the TASKS collection we create a variable $T_{ij}^{\texttt{end}}$ which is set to the trail attribute of task $\texttt{TASKS}[j]$ if task $\texttt{TASKS}[j]$ overlaps the end attribute of task $\texttt{TASKS}[i]$, and to the trail attribute of task $\texttt{TASKS}[i]$ otherwise:
   - If $i = j$:
     - $T_{ij}^{\texttt{end}} = \texttt{TASKS}[i].\texttt{trail}$.
   - If $i \neq j$:
     - $T_{ij}^{\texttt{end}} = \texttt{TASKS}[i].\texttt{trail} \vee T_{ij}^{\texttt{end}} = \texttt{TASKS}[j].\texttt{trail}$.
     - $((\texttt{TASKS}[j].\texttt{origin} \leq \texttt{TASKS}[i].\texttt{end} - 1 \wedge$
       $\texttt{TASKS}[j].\texttt{end} > \texttt{TASKS}[i].\texttt{end} - 1) \wedge (T_{ij}^{\texttt{end}} = \texttt{TASKS}[j].\texttt{trail})) \vee$
       $((\texttt{TASKS}[j].\texttt{origin} > \texttt{TASKS}[i].\texttt{end} - 1 \vee$
       $\texttt{TASKS}[j].\texttt{end} \leq \texttt{TASKS}[i].\texttt{end} - 1) \wedge (T_{ij}^{\texttt{end}} = \texttt{TASKS}[i].\texttt{trail}))$
4. For each task $\texttt{TASKS}[i]$ $(i \in [1, |\texttt{TASKS}|])$ we impose the number of distinct trails associated with the tasks that overlap the end of task $\texttt{TASKS}[i]$ ($\texttt{TASKS}[i]$ overlaps its

own end) to be equal to NTRAIL:
NVALUE(NTRAIL, $\langle T_{i1}^{\texttt{end}}, T_{i2}^{\texttt{end}}, \ldots, T_{i|\texttt{TASKS}|}^{\texttt{end}}\rangle$).

With respect to the **Example** slot we get the following conjunction of NVALUE constraints:

- The NVALUE$(2, \langle 1, 2, 1, 1, 1\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the origin of the first task (i.e., instant 1) that has a trail of 1.
- The NVALUE$(2, \langle 1, 2, 2, 2, 2\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the origin of the second task (i.e., instant 1) that has a trail of 2.
- The NVALUE$(2, \langle 1, 1, 1, 2, 1\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the origin of the third task (i.e., instant 2) that has a trail of 1.
- The NVALUE$(2, \langle 2, 2, 1, 2, 2\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the origin of the fourth task (i.e., instant 2) that has a trail of 2.
- The NVALUE$(2, \langle 2, 2, 1, 2, 2\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the origin of the fifth task (i.e., instant 3) that has a trail of 2.
- The NVALUE$(2, \langle 1, 2, 1, 1, 1\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the last instant of the first task (i.e., instant 1) that has a trail of 1.
- The NVALUE$(2, \langle 1, 2, 2, 2, 2\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the last instant of the second task (i.e., instant 1) that has a trail of 2.
- The NVALUE$(2, \langle 1, 1, 1, 1, 2\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the last instant of the third task (i.e., instant 3) that has a trail of 1.
- The NVALUE$(2, \langle 2, 2, 1, 2, 2\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the last instant of the fourth task (i.e., instant 2) that has a trail of 2.
- The NVALUE$(2, \langle 2, 2, 1, 2, 2\rangle)$ constraint corresponding to the `trail` attributes of the tasks that overlap the last instant of the fifth task (i.e., instant 3) that has a trail of 2.

See also

common keyword: COLOURED_CUMULATIVE *(resource constraint)*.

implies (items to collection): ATLEAST_NVECTOR.

used in graph description: NVALUE.

Keywords

characteristic of a constraint: derived collection.

constraint type: timetabling constraint, resource constraint, temporal constraint.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{TIME\_POINTS}-\texttt{collection}\left(\begin{array}{l}\texttt{origin}-\texttt{dvar},\\ \texttt{end}-\texttt{dvar},\\ \texttt{point}-\texttt{dvar}\end{array}\right),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{origin}-\texttt{TASKS.origin},\\ \texttt{end}-\texttt{TASKS.end},\\ \texttt{point}-\texttt{TASKS.origin}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{origin}-\texttt{TASKS.origin},\\ \texttt{end}-\texttt{TASKS.end},\\ \texttt{point}-\texttt{TASKS.end}-1\end{array}\right)\end{array}\right]\end{array}\right)$$

Arc input(s) TASKS

Arc generator $SELF \mapsto$collection(tasks)

Arc arity 1

Arc constraint(s) tasks.origin $\leq$ tasks.end

Graph property(ies) **NARC**$=$ |TASKS|

Arc input(s) TIME_POINTS TASKS

Arc generator $PRODUCT \mapsto$collection(time_points, tasks)

Arc arity 2

Arc constraint(s)
- time_points.end $>$ time_points.origin
- tasks.origin $\leq$ time_points.point
- time_points.point $<$ tasks.end

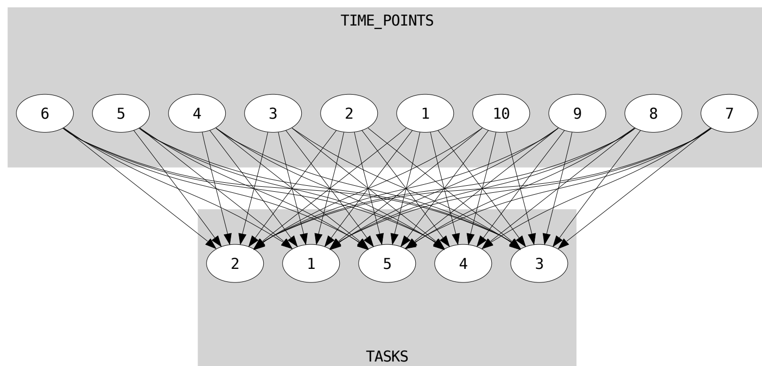
Sets

$$\text{SUCC} \mapsto \left[\begin{array}{l}\texttt{source},\\ \texttt{variables}-\texttt{col}\left(\begin{array}{l}\texttt{VARIABLES}-\texttt{collection}(\texttt{var}-\texttt{dvar}),\\ [\texttt{item}(\texttt{var}-\texttt{TASKS.trail})]\end{array}\right)\end{array}\right]$$

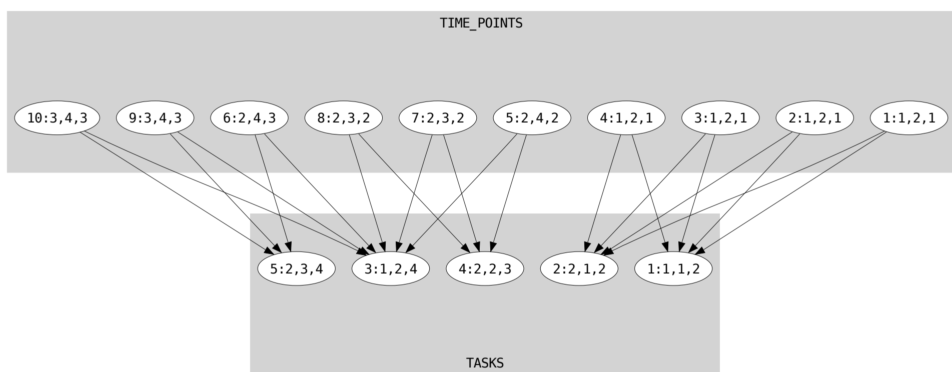
Constraint(s) on sets NVALUE(NTRAIL, variables)

Graph model Parts (A) and (B) of Figure 5.809 respectively show the initial and final graph of the second graph constraint of the **Example** slot.

Signature Consider the first graph constraint. Since we use the $SELF$ arc generator on the TASKS collection, the maximum number of arcs of the final graph is equal to |TASKS|. Therefore we can rewrite **NARC** $=$ |TASKS| to **NARC** $\geq$ |TASKS| and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.



(A)



(B)

Figure 5.809: Initial and final graph of the TRACK constraint

5.411 TREE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin N. Beldiceanu

Constraint TREE(NTREES, NODES)

Arguments

NTREES : dvar
NODES : collection(index−int, succ−dvar)

Restrictions

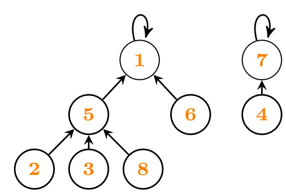
NTREES ≥ 1
NTREES $\leq$ |NODES|
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

Purpose

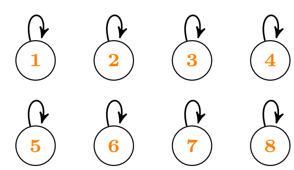
Given a digraph G described by the NODES collection, cover G by a set of NTREES trees in such a way that each vertex of G belongs to one distinct tree. The edges of the trees are directed from their leaves to their respective roots.

Example

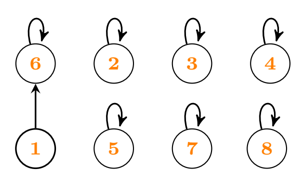
$$\left(2,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{succ}-5,\\ \texttt{index}-3 & \texttt{succ}-5,\\ \texttt{index}-4 & \texttt{succ}-7,\\ \texttt{index}-5 & \texttt{succ}-1,\\ \texttt{index}-6 & \texttt{succ}-1,\\ \texttt{index}-7 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{succ}-5\end{array}\right\rangle\right)$$



$$\left(8,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-1,\\ \texttt{index}-2 & \texttt{succ}-2,\\ \texttt{index}-3 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{succ}-4,\\ \texttt{index}-5 & \texttt{succ}-5,\\ \texttt{index}-6 & \texttt{succ}-6,\\ \texttt{index}-7 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{succ}-8\end{array}\right\rangle\right)$$



$$\left(7,\left\langle\begin{array}{ll}\texttt{index}-1 & \texttt{succ}-6,\\ \texttt{index}-2 & \texttt{succ}-2,\\ \texttt{index}-3 & \texttt{succ}-3,\\ \texttt{index}-4 & \texttt{succ}-4,\\ \texttt{index}-5 & \texttt{succ}-5,\\ \texttt{index}-6 & \texttt{succ}-6,\\ \texttt{index}-7 & \texttt{succ}-7,\\ \texttt{index}-8 & \texttt{succ}-8\end{array}\right\rangle\right)$$



The first TREE constraint holds since the graph associated with the items of the NODES collection corresponds to two trees (i.e., NTREES $= 2$): each tree respectively involves the vertices $\{1, 2, 3, 5, 6, 8\}$ and $\{4, 7\}$. They are depicted by Figure 5.810.

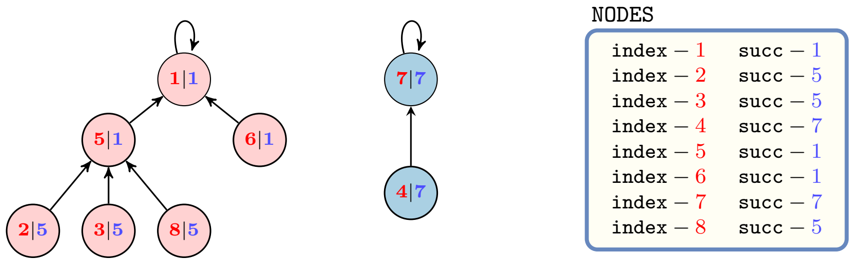


Figure 5.810: The two trees corresponding to the first example of the **Example** slot; each vertex contains the information index|succ where succ is the index of its father in the tree (by convention the father of the root is the root itself).

All solutions

Figure 5.811 gives all solutions to the following non ground instance of the TREE constraint: NTREES $\in [3, 4]$, $\mathtt{S}_1 \in [1, 2]$, $\mathtt{S}_2 \in [1, 3]$, $\mathtt{S}_3 \in [1, 4]$, $\mathtt{S}_4 \in [2, 4]$, TREE(NTREES, $\langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4\rangle$).

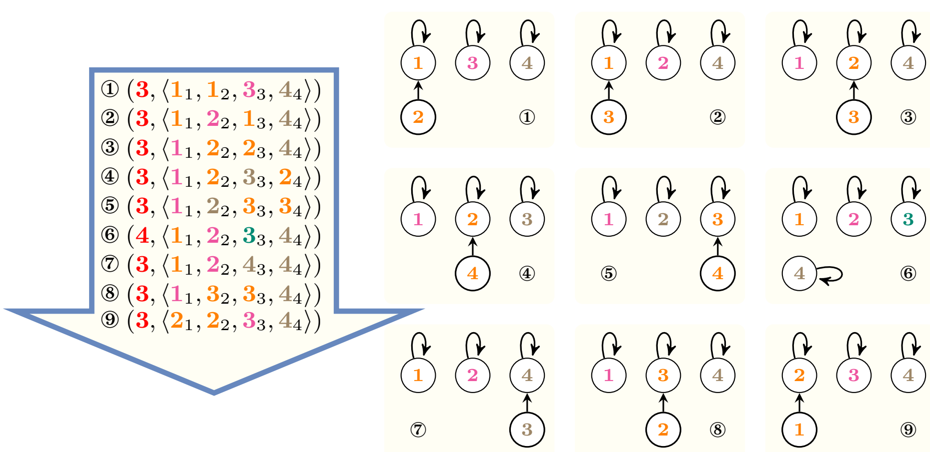


Figure 5.811: All solutions corresponding to the non ground example of the TREE constraint of the **All solutions** slot, where all vertices of a same tree are coloured by the same colour; in the left-hand side the index attributes are displayed as indices of the succ attribute, while in the right-hand side they are directly displayed within each node.

Typical

NTREES $<$ |NODES|
|NODES| > 2

Symmetry Items of NODES are permutable.

Arg. properties Functional dependency: NTREES determined by NODES.

Remark Given a complete digraph of n vertices as well as an unrestricted number of trees NTREES, the total number of solutions to the corresponding TREE constraint corresponds to the sequence A000272 of the On-Line Encyclopaedia of Integer Sequences [403].

In the context of an undirected weighted graph, extension of the TREE constraint to the *minimum spanning tree* constraint is described in [152, 360, 363].

Algorithm An arc-consistency filtering algorithm for the TREE constraint is described in [48]. This algorithm is based on a necessary and sufficient condition that we now depict.

To any TREE constraint we associate the digraph $G = (V, E)$, where:

- To each item NODES$[i]$ of the NODES collection corresponds a vertex v_i of G.
- For every pair of items (NODES$[i]$, NODES$[j]$) of the NODES collection, where i and j are not necessarily distinct, there is an arc from v_i to v_j in E if and only if j is a potential value of NODES$[i]$.succ.

A strongly connected component C of G is called a *sink component* if all the successors of all vertices of C belong to C. Let MINTREES and MAXTREES respectively denote the number of sink components of G and the number of vertices of G with a loop.

The TREE constraint has a solution if and only if:

- Each sink component of G contains at least one vertex with a loop,
- The domain of NTREES has at least one value within interval [MINTREES, MAXTREES].

Inspired by the idea of using dominators used in [234] for getting a linear time algorithm for computing strong articulation points of a digraph G, the worst case complexity of the algorithm proposed in [48] was also enhanced in a similar way by J.-G. Fages and X. Lorca [167].

Reformulation The TREE constraint can be expressed in term of (1) a set of $|\text{NODES}|^2$ reified constraints for avoiding circuit between more than one node and of (2) $|\text{NODES}|$ reified constraints and of one sum constraint for counting the trees:

1. For each vertex NODES$[i]$ ($i \in [1, |\text{NODES}|]$) of the NODES collection we create a variable r_i that takes its value within interval $[1, |\text{NODES}|]$. This variable represents the *rank* of vertex NODES$[i]$ within a solution. It is used to prevent the creation of circuit involving more than one vertex as explained now. For each pair of vertices NODES$[i]$, NODES$[j]$ ($i, j \in [1, |\text{NODES}|]$) of the NODES collection we create a reified constraint of the form NODES$[i]$.succ $=$ NODES$[j]$.index $\wedge\ i \neq j \Rightarrow r_i > r_j$. The purpose of this constraint is to express the fact that, if there is an arc from vertex NODES$[i]$ to another vertex NODES$[j]$, then r_i should be strictly greater than r_j.
2. For each vertex NODES$[i]$ ($i \in [1, |\text{NODES}|]$) of the NODES collection we create a 0-1 variable b_i and state the following reified constraint NODES$[i]$.succ $=$ NODES$[i]$.index $\Leftrightarrow b_i$ in order to force variable b_i to be set to value 1 if and only if there is a loop on vertex NODES$[i]$. Finally we create a constraint NTREES $= b_1 + b_2 + \cdots + b_{|\text{NODES}|}$ for stating the fact that the number of trees is equal to the number of loops of the graph.

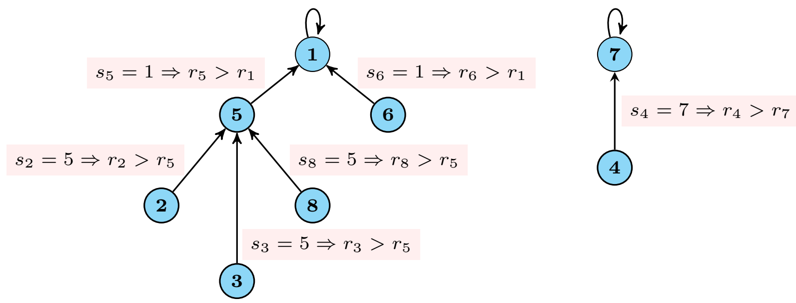


Figure 5.812: Illustrating the reformulation of the TREE constraint with rank variables for preventing the creation of circuits involving more than one vertex: the two trees corresponding to the first example of the **Example** slot with the link between the successor variables $s_1, s_2, \ldots, s_8$ and the rank variables $r_1, r_2, \ldots, r_8$

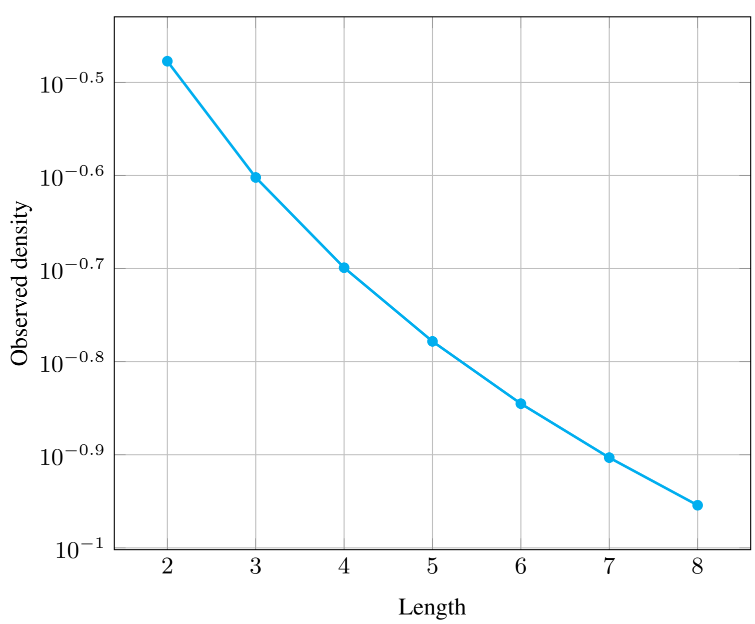
Figure 5.812 illustrates this reformulation on the first example of the **Example** slot.

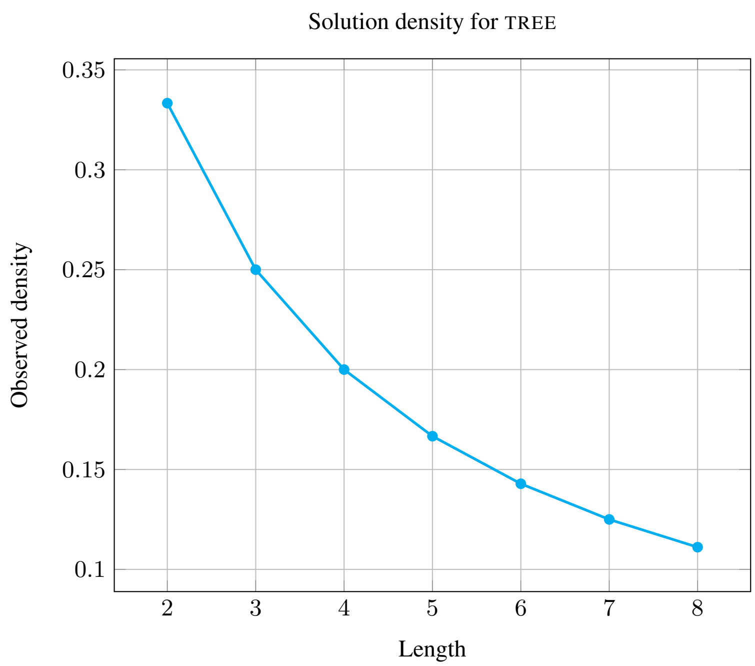
Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	16	125	1296	16807	262144	4782969

Number of solutions for TREE: domains $0..n$

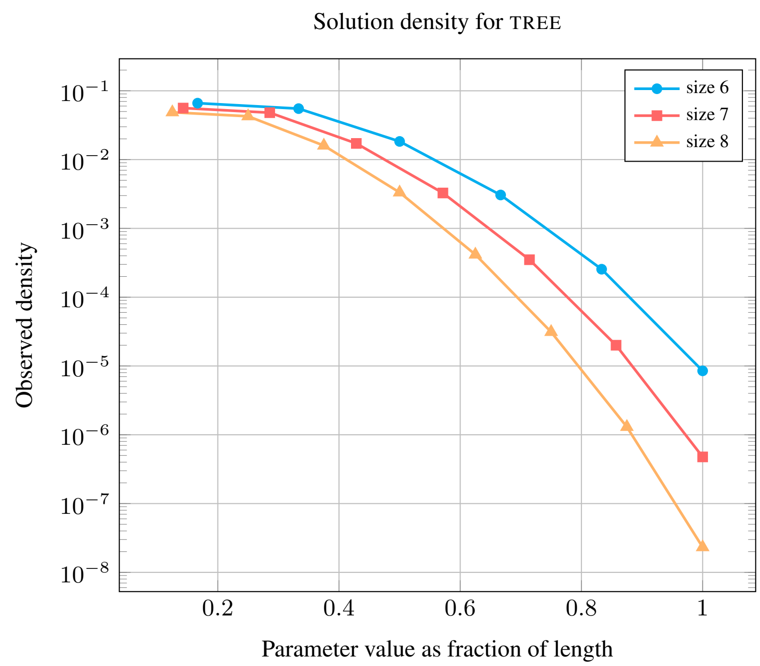
Solution density for TREE

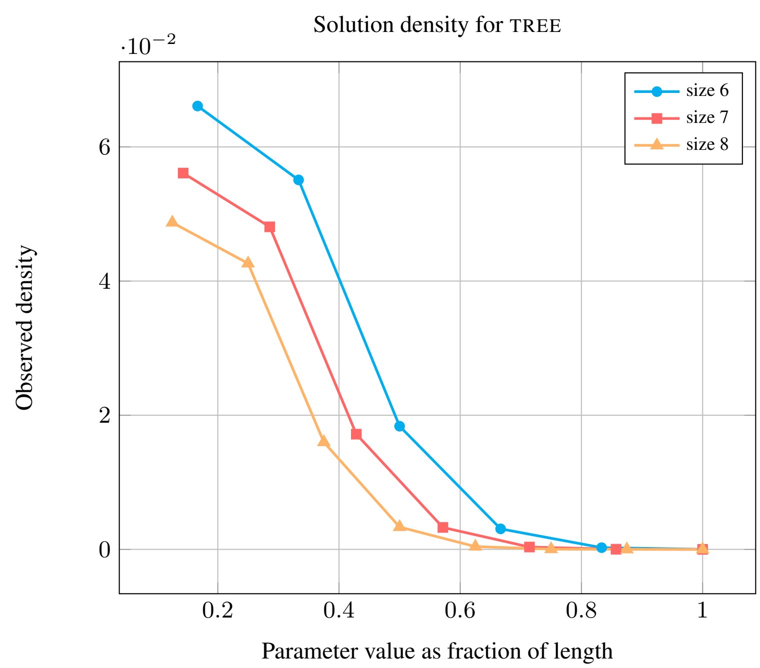




Length (n)		2	3	4	5	6	7	8
Total		3	16	125	1296	16807	262144	4782969
Parameter value	1	2	9	64	625	7776	117649	2097152
	2	1	6	48	500	6480	100842	1835008
	3	-	1	12	150	2160	36015	688128
	4	-	-	1	20	360	6860	143360
	5	-	-	-	1	30	735	17920
	6	-	-	-	-	1	42	1344
	7	-	-	-	-	-	1	56
	8	-	-	-	-	-	-	1

Solution count for TREE: domains $0..n$





Systems TREE in **Choco**.

See also **common keyword:** CYCLE, GRAPH_CROSSING, MAP *(graph partitioning constraint)*, PROPER_FOREST *(connected component,tree)*.

implied by: BINARY_TREE.

implies (items to collection): ATLEAST_NVECTOR.

related: BALANCE_TREE *(counting number of trees versus controlling how balanced the trees are)*, GLOBAL_CARDINALITY_LOW_UP_NO_LOOP, GLOBAL_CARDINALITY_NO_LOOP *(can be used for restricting number of children since discard loops associated with tree roots)*.

shift of concept: STABLE_COMPATIBILITY, TREE_RANGE, TREE_RESOURCE.

specialisation: BINARY_TREE *(no limit on the number of children replaced by at most two children)*, PATH *(no limit on the number of children replaced by at most one child)*.

uses in its reformulation: TREE_RANGE, TREE_RESOURCE.

Keywords

constraint type: graph constraint, graph partitioning constraint.

filtering: DFS-bottleneck, strong articulation point, arc-consistency.

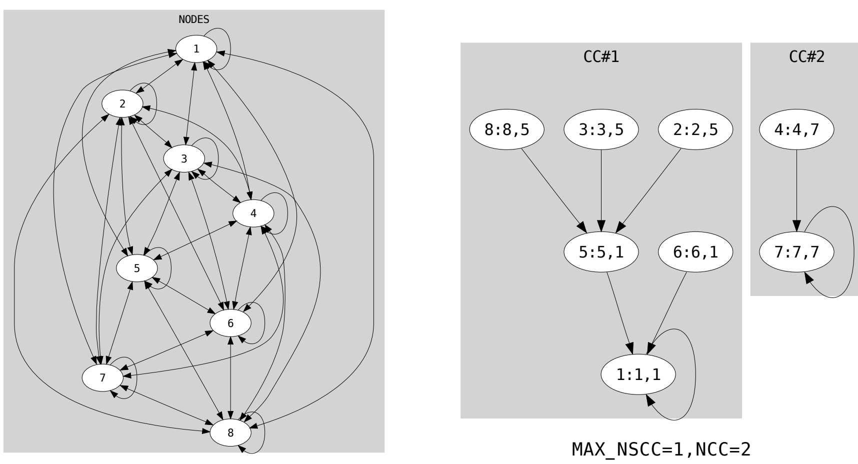
final graph structure: connected component, tree, one_succ.

modelling: functional dependency.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NCC** = NTREES

Graph model We use the graph property **MAX_NSCC** ≤ 1 in order to specify the fact that the size of the largest strongly connected component should not exceed one. In fact each root of a tree is a strongly connected component with a single vertex. The second graph property **NCC** = NTREES enforces the number of trees to be equal to the number of connected components.

Parts (A) and (B) of Figure 5.813 respectively show the initial and final graph associated with the first example of the **Example** slot. Since we use the **NCC** graph property, we display the two connected components of the final graph. Each of them corresponds to a tree. The TREE constraint holds since all strongly connected components of the final graph have no more than one vertex and since NTREES = **NCC** = 2.



(A) **(B)**

Figure 5.813: Initial and final graph of the TREE constraint

5.412 TREE_RANGE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Derived from TREE.

Constraint TREE_RANGE(NTREES, R, NODES)

Arguments

NTREES	:	dvar
R	:	dvar
NODES	:	collection(index−int, succ−dvar)

Restrictions

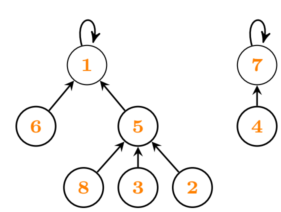
NTREES ≥ 0
R ≥ 0
R $<$ |NODES|
|NODES| > 0
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index $\leq$ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ $\leq$ |NODES|

Purpose

Cover the digraph G described by the NODES collection with NTREES trees in such a way that each vertex of G belongs to one distinct tree. R is the difference between the longest and the shortest paths (from a leaf to a root) of the final graph.

Example

$$\left(2, 1, \left\langle \begin{array}{ll} \texttt{index}-1 & \texttt{succ}-1, \\ \texttt{index}-2 & \texttt{succ}-5, \\ \texttt{index}-3 & \texttt{succ}-5, \\ \texttt{index}-4 & \texttt{succ}-7, \\ \texttt{index}-5 & \texttt{succ}-1, \\ \texttt{index}-6 & \texttt{succ}-1, \\ \texttt{index}-7 & \texttt{succ}-7, \\ \texttt{index}-8 & \texttt{succ}-5 \end{array} \right\rangle \right)$$



The TREE_RANGE constraint holds since the graph associated with the items of the NODES collection corresponds to two trees (i.e., NTREES $= 2$): each tree respectively involves the vertices $\{1, 2, 3, 5, 6, 8\}$ and $\{4, 7\}$. Furthermore R $= 1$ is set to the difference between the longest path (for example, $2 \rightarrow 5 \rightarrow 1$) and the shortest path (for example, $4 \rightarrow 7$) from a leaf to a root. Figure 5.814 provides the two trees associated with the example.

All solutions

Figure 5.815 gives all solutions to the following non ground instance of the TREE_RANGE constraint: NTREES $\in [3, 4]$, R $= 2$, $\texttt{S}_1 \in [1, 2]$, $\texttt{S}_2 \in [1, 3]$, $\texttt{S}_3 \in [1, 4]$, $\texttt{S}_4 \in [2, 3]$, $\texttt{S}_5 \in [3, 4]$, $\texttt{S}_6 \in [4, 5]$, TREE_RANGE(NTREES, R, $\langle \texttt{S}_1, \texttt{S}_2, \texttt{S}_3, \texttt{S}_4, \texttt{S}_5, \texttt{S}_6 \rangle$).

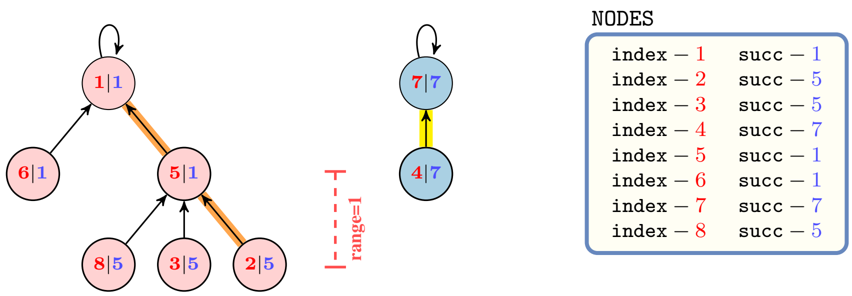


Figure 5.814: The two trees corresponding to the **Example** slot; each vertex contains the information index|succ where succ is the index of its father in the tree (by convention the father of the root is the root itself); the longest and shortest paths from a leaf to a root are respectively shown by thick orange and yellow line segments and have a length of 2 and 1; consequently the range is equal to 1.

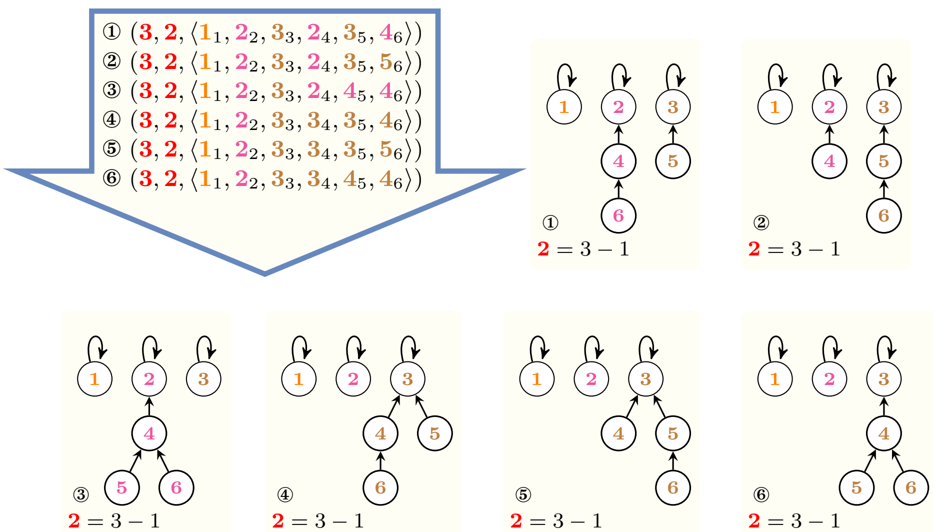


Figure 5.815: All solutions corresponding to the non ground example of the TREE_RANGE constraint of the **All solutions** slot, where all vertices of a same tree are coloured by the same colour; in the left-hand side the index attributes are displayed as indices of the succ attribute, while in the right-hand side they are directly displayed within each node; the bottom left part of each subfigure shows how the R argument (in red) is related to the longest and to the smallest paths from any leaf to the corresponding root.

Typical

NTREES $<$ |NODES|
|NODES| > 2

Symmetry Items of NODES are permutable.

Arg. properties

- Functional dependency: NTREES determined by NODES.
- Functional dependency: R determined by NODES.

Reformulation By introducing a *distance variable* D_i, an *occurrence variable* O_i and a *leave variable* L_i $(1 \leq i \leq |\texttt{NODES}|)$ for each item i of the NODES collection, where:

- D_i represents the number of vertices from i to the root of the corresponding tree,
- O_i gives the number of occurrences of value i within variables $\texttt{NODES}[1].\texttt{succ}, \texttt{NODES}[2].\texttt{succ}, \ldots, \texttt{NODES}[n].\texttt{succ}$,
- L_i is set to 1 if item i corresponds to a leave (i.e., $O_i > 0$) and 0 otherwise,

the TREE_RANGE(NTREES, R, NODES) constraint can be expressed in term of a conjunction of one TREE constraint, |NODES| ELEMENT constraints, |NODES| linear constraints, one GLOBAL_CARDINALITY constraint, |NODES| reified constraints, one OPEN_MINIMUM, one MAXIMUM and one linear constraint, where:

- The TREE constraint models the fact that we have a forest of NTREES trees.
- Each ELEMENT constraint provides the link between the attribute succ of the i-th item and the distance variable $D_{\texttt{NODES}[i].\texttt{succ}}$ associated with item $\texttt{NODES}[i].\texttt{succ}$.
- Each linear constraint associated with the i-th item states that the difference between the distance variable D_i and the distance variable $D_{\texttt{NODES}[i].\texttt{succ}}$ is equal to 1.
- The GLOBAL_CARDINALITY constraint provides the number of occurrences O_i of value i $(1 \leq i \leq |\texttt{NODES}|)$ within variables $\texttt{NODES}[1].\texttt{succ}, \texttt{NODES}[2].\texttt{succ}, \ldots, \texttt{NODES}[|\texttt{NODES}|].\texttt{succ}$. Note that, when O_i is equal to 0, the corresponding i-th item is a leave of one of the NTREES trees.
- Each reified constraint of the form $L_i \Leftrightarrow O_i > 0$ makes the link between the i-th occurrence variable O_i and the i-th leave variable L_i.
- The OPEN_MINIMUM constraint computes the minimum distance MIN from the leaves to the corresponding roots. The leave variable L_i is used in order to select only the distance variables corresponding to leaves.
- The MAXIMUM constraint computes the maximum distance MAX from the vertices to the roots. Since the maximum is achieved by a leave we do not need to focus just on the leaves as it was the case for the minimum distance MIN.
- The linear constraint $\texttt{MAX} - \texttt{MIN} = \texttt{R}$ links together argument R to the minimum and maximum distances.

With respect to the **Example** slot we get the following conjunction of constraints:

$$\begin{array}{l}
\text{TREE}(2, \langle \texttt{index}-1\ \texttt{succ}-1,\ \texttt{index}-2\ \texttt{succ}-5, \\
\quad \texttt{index}-3\ \texttt{succ}-5,\ \texttt{index}-4\ \texttt{succ}-7, \\
\quad \texttt{index}-5\ \texttt{succ}-1,\ \texttt{index}-6\ \texttt{succ}-1, \\
\quad \texttt{index}-7\ \texttt{succ}-7,\ \texttt{index}-8\ \texttt{succ}-5\rangle), \\
\text{DOMAIN}(\langle D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8\rangle, 0, 8), \\
DS_1 \in [0,8],\ \text{ELEMENT}(1, \langle 0, D_2, D_3, D_4, D_5, D_6, D_7, D_8\rangle, DS_1),\ D_1 - 0 = 1, \\
DS_2 \in [0,8],\ \text{ELEMENT}(5, \langle 1, 0, D_3, D_4, D_5, D_6, D_7, D_8\rangle, DS_2),\ D_2 - D_5 = 1, \\
DS_3 \in [0,8],\ \text{ELEMENT}(5, \langle 1, D_2, 0, D_4, D_5, D_6, D_7, D_8\rangle, DS_3),\ D_3 - D_5 = 1,
\end{array}$$

$DS_4 \in [0,8]$, ELEMENT$(7, \langle 1, D_2, D_3, 0, D_5, D_6, D_7, D_8\rangle, DS_4)$, $D_4 - D_7 = 1$,
$DS_5 \in [0,8]$, ELEMENT$(1, \langle 1, D_2, D_3, D_4, 0, D_6, D_7, D_8\rangle, DS_5)$, $D_5 - 1 = 1$,
$DS_6 \in [0,8]$, ELEMENT$(1, \langle 1, 3, 3, D_4, 2, 0, D_7, D_8\rangle, DS_6)$, $D_6 - 1 = 1$,
$DS_7 \in [0,8]$, ELEMENT$(7, \langle 1, 3, 3, D_4, 2, 2, 0, D_8\rangle, DS_7)$, $D_7 - 0 = 1$,
$DS_8 \in [0,8]$, ELEMENT$(5, \langle 1, 3, 3, 2, 2, 2, 1, 0\rangle, DS_8)$, $D_8 - 2 = 1$,
GLOBAL_CARDINALITY$(\langle 1, 5, 5, 7, 1, 1, 7, 5\rangle$, $\langle \texttt{val} - 1\ \texttt{noccurrence} - 3$,
$\texttt{val} - 2\ \texttt{noccurrence} - 0$,
$\texttt{val} - 3\ \texttt{noccurrence} - 0$,
$\texttt{val} - 4\ \texttt{noccurrence} - 0$,
$\texttt{val} - 5\ \texttt{noccurrence} - 3$,
$\texttt{val} - 6\ \texttt{noccurrence} - 0$,
$\texttt{val} - 7\ \texttt{noccurrence} - 2$,
$\texttt{val} - 8\ \texttt{noccurrence} - 0\rangle)$,
$1 \Leftrightarrow 3 > 0, 0 \Leftrightarrow 0 > 0, 0 \Leftrightarrow 0 > 0, 0 \Leftrightarrow 0 > 0$,
$1 \Leftrightarrow 3 > 0, 0 \Leftrightarrow 0 > 0, 1 \Leftrightarrow 2 > 0, 0 \Leftrightarrow 0 > 0$,
OPEN_MINIMUM$(\texttt{MIN}, \langle \texttt{var} - 3\ \texttt{bool} - 1, \texttt{var} - 0\ \texttt{bool} - 0$,
$\texttt{var} - 0\ \texttt{bool} - 0, \texttt{var} - 0\ \texttt{bool} - 0$,
$\texttt{var} - 3\ \texttt{bool} - 1, \texttt{var} - 0\ \texttt{bool} - 0$,
$\texttt{var} - 2\ \texttt{bool} - 1, \texttt{var} - 0\ \texttt{bool} - 0\rangle)$,
MAXIMUM$(\texttt{MAX}, \langle 1, 3, 3, 2, 2, 2, 1, 3\rangle)$,
$\texttt{MAX} - \texttt{MIN} = \texttt{R} = 1$.

See also

related: BALANCE *(balanced tree versus balanced assignment)*.

root concept: TREE.

used in reformulation: DOMAIN, ELEMENT, GLOBAL_CARDINALITY, MAXIMUM, OPEN_MINIMUM, TREE.

Keywords

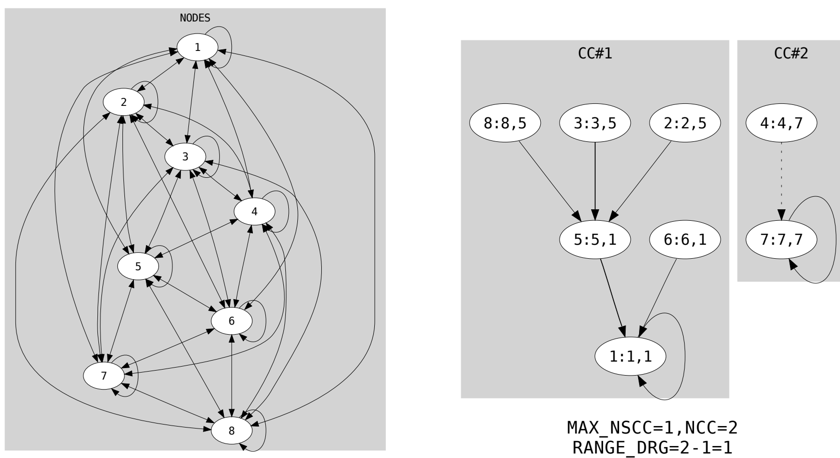
constraint type: graph constraint, graph partitioning constraint.

final graph structure: connected component, tree.

modelling: balanced tree, functional dependency.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NCC** = NTREES • **RANGE_DRG** = R

Graph model Parts (A) and (B) of Figure 5.816 respectively show the initial and final graph associated with the **Example** slot. Since we use the RANGE_DRG graph property, we respectively display the longest and shortest paths of the final graph with a bold and a dash line.



(A) **(B)**

Figure 5.816: Initial and final graph of the TREE_RANGE constraint

5.413 TREE_RESOURCE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from TREE.

Constraint TREE_RESOURCE(RESOURCE, TASK)

Arguments

$$\begin{array}{lll} \text{RESOURCE} & : & \text{collection}(\text{id}-\text{int}, \text{nb\_task}-\text{dvar}) \\ \text{TASK} & : & \text{collection}(\text{id}-\text{int}, \text{father}-\text{dvar}, \text{resource}-\text{dvar}) \end{array}$$

Restrictions

$|\text{RESOURCE}| > 0$
$\text{required}(\text{RESOURCE}, [\text{id}, \text{nb\_task}])$
$\text{RESOURCE.id} \geq 1$
$\text{RESOURCE.id} \leq |\text{RESOURCE}|$
$\text{distinct}(\text{RESOURCE}, \text{id})$
$\text{RESOURCE.nb\_task} \geq 0$
$\text{RESOURCE.nb\_task} \leq |\text{TASK}|$
$\text{required}(\text{TASK}, [\text{id}, \text{father}, \text{resource}])$
$\text{TASK.id} > |\text{RESOURCE}|$
$\text{TASK.id} \leq |\text{RESOURCE}| + |\text{TASK}|$
$\text{distinct}(\text{TASK}, \text{id})$
$\text{TASK.father} \geq 1$
$\text{TASK.father} \leq |\text{RESOURCE}| + |\text{TASK}|$
$\text{TASK.resource} \geq 1$
$\text{TASK.resource} \leq |\text{RESOURCE}|$

Purpose

Cover a digraph G in such a way that each vertex belongs to one distinct tree. Each tree is made up from one *resource* vertex and several *task* vertices. The resource vertices correspond to the roots of the different trees. For each resource a domain variable nb_task indicates how many task-vertices belong to the corresponding tree. For each task a domain variable resource gives the identifier of the resource that can handle the task.

Example

$$\left(\begin{array}{l} \left\langle\begin{array}{ll} \text{id}-1 & \text{nb\_task}-4, \\ \text{id}-2 & \text{nb\_task}-0, \\ \text{id}-3 & \text{nb\_task}-1 \end{array}\right\rangle, \\ \left\langle\begin{array}{lll} \text{id}-4 & \text{father}-8 & \text{resource}-1, \\ \text{id}-5 & \text{father}-3 & \text{resource}-3, \\ \text{id}-6 & \text{father}-8 & \text{resource}-1, \\ \text{id}-7 & \text{father}-1 & \text{resource}-1, \\ \text{id}-8 & \text{father}-1 & \text{resource}-1 \end{array}\right\rangle \end{array}\right)$$

The TREE_RESOURCE constraint holds since the graph associated with the items of the RESOURCE and the TASK collections corresponds to 3 trees (i.e., $|\text{RESOURCE}| = 3$): each tree respectively involves the vertices $\{1, 4, 6, 7, 8\}$, $\{2\}$ and $\{3, 5\}$. They are depicted by Figure 5.817, where *resource* and *task* vertices are respectively coloured in blue and pink.

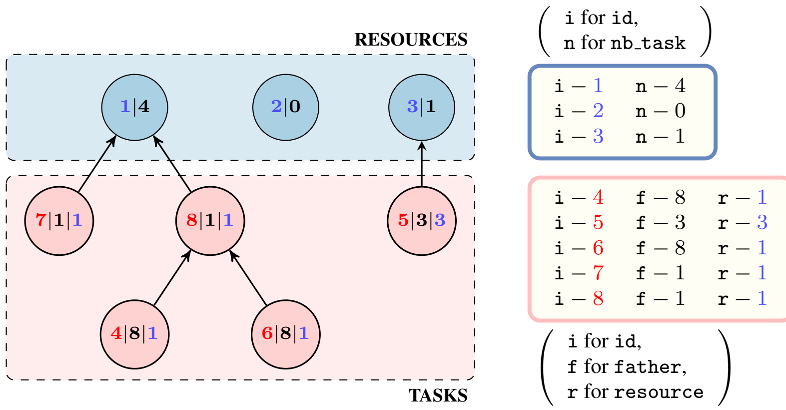


Figure 5.817: The three trees corresponding to the **Example** slot; each resource vertex (in blue) contains the information id|nb_task where nb_task is the number of tasks in the tree, while each task vertex (in pink) contains the information id|father|resource where father is the index of its father in the tree and resource is the index of the corresponding root task in the tree.

Typical

$|\mathtt{RESOURCE}| > 0$
$|\mathtt{TASK}| > |\mathtt{RESOURCE}|$

Symmetries

- Items of RESOURCE are permutable.
- Items of TASK are permutable.

Reformulation

The TREE_RESOURCE(RESOURCE, TASK) constraint can be expressed in term of a conjunction of one TREE constraint, |TASK| ELEMENT constraints and one GLOBAL_CARDINALITY constraint:

- The TREE constraint expresses the fact that we have a well formed tree.
- The ELEMENT constraint is used for expressing the link between the father attribute of an item of the TASK collection and its corresponding resource attribute.
- The GLOBAL_CARDINALITY constraint is used to link the resource attribute of the items of the TASK collection with the nb_task attribute of the items of the RESOURCE collection.

With respect to the **Example** slot we get the following conjunction of constraints:

TREE$(3, \langle$index $-\ 1$ succ $-\ 1,$
index $-\ 2$ succ $-\ 2,$
index $-\ 3$ succ $-\ 3,$
index $-\ 4$ succ $-\ 8,$
index $-\ 5$ succ $-\ 3,$
index $-\ 6$ succ $-\ 8,$
index $-\ 7$ succ $-\ 1,$
index $-\ 8$ succ $-\ 1\rangle),$
ELEMENT$(8, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 1),$

$\text{ELEMENT}(3, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 3)$,
$\text{ELEMENT}(8, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 1)$,
$\text{ELEMENT}(1, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 1)$,
$\text{ELEMENT}(1, \langle 1, 2, 3, 1, 3, 1, 1, 1\rangle, 1)$,
$\text{GLOBAL\_CARDINALITY}(\langle 1, 3, 1, 1, 1\rangle,$
$\langle \texttt{val} - 1\ \texttt{noccurrence} - 4,$
$\texttt{val} - 2\ \texttt{noccurrence} - 0,$
$\texttt{val} - 3\ \texttt{noccurrence} - 1\rangle)$.

See also **root concept:** TREE.

used in reformulation: ELEMENT, GLOBAL_CARDINALITY, TREE.

Keywords **characteristic of a constraint:** derived collection.

constraint type: graph constraint, resource constraint, graph partitioning constraint.

final graph structure: tree, connected component.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{RESOURCE\_TASK}-\texttt{collection}\left(\begin{array}{l}\texttt{index}-\texttt{int},\\ \texttt{succ}-\texttt{dvar},\\ \texttt{name}-\texttt{dvar}\end{array}\right),\\ \left[\begin{array}{l}\texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{RESOURCE.id},\\ \texttt{succ}-\texttt{RESOURCE.id},\\ \texttt{name}-\texttt{RESOURCE.id}\end{array}\right),\\ \texttt{item}\left(\begin{array}{l}\texttt{index}-\texttt{TASK.id},\\ \texttt{succ}-\texttt{TASK.father},\\ \texttt{name}-\texttt{TASK.resource}\end{array}\right)\end{array}\right]\end{array}\right)$$

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

Arc arity 2

Arc constraint(s)
- $\texttt{resource\_task1.succ} = \texttt{resource\_task2.index}$
- $\texttt{resource\_task1.name} = \texttt{resource\_task2.name}$

Graph property(ies)
- $\mathbf{MAX\_NSCC} \leq 1$
- $\mathbf{NCC} = |\texttt{RESOURCE}|$
- $\mathbf{NVERTEX} = |\texttt{RESOURCE}| + |\texttt{TASK}|$

For all items of RESOURCE:

Arc input(s) RESOURCE_TASK

Arc generator $CLIQUE \mapsto \texttt{collection}(\texttt{resource\_task1}, \texttt{resource\_task2})$

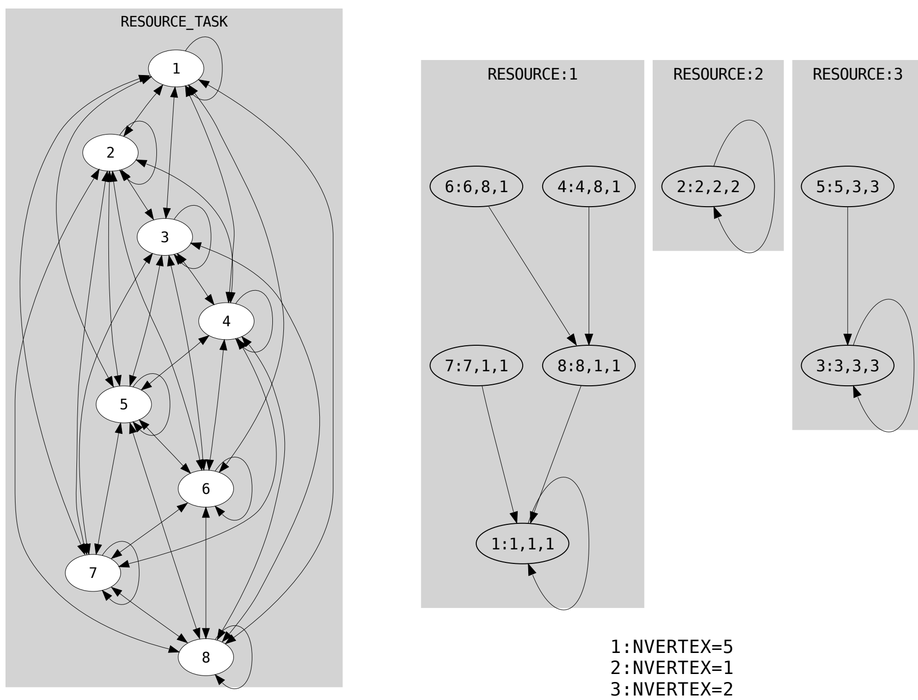
Arc arity 2

Arc constraint(s)
- $\texttt{resource\_task1.succ} = \texttt{resource\_task2.index}$
- $\texttt{resource\_task1.name} = \texttt{resource\_task2.name}$
- $\texttt{resource\_task1.name} = \texttt{RESOURCE.id}$

Graph property(ies) $\mathbf{NVERTEX} = \texttt{RESOURCE.nb\_task} + 1$

Graph model For the second graph constraint, part (A) of Figure 5.818 shows the initial graphs associated with resources 1, 2 and 3 of the **Example** slot. For the second graph constraint, part (B) of Figure 5.818 shows the corresponding final graphs associated with resources 1, 2 and 3. Since we use the **NVERTEX** graph property, the vertices of the final graphs are stressed in bold. To each resource corresponds a tree of respectively 4, 0 and 1 task-vertices.

Signature Since the initial graph of the first graph constraint contains $|\texttt{RESOURCE}| + |\texttt{TASK}|$ vertices, the corresponding final graph cannot have more than $|\texttt{RESOURCE}| + |\texttt{TASK}|$ vertices. Therefore we can rewrite the graph property $\mathbf{NVERTEX} = |\texttt{RESOURCE}| + |\texttt{TASK}|$ to $\mathbf{NVERTEX} \geq |\texttt{RESOURCE}| + |\texttt{TASK}|$ and simplify $\underline{\mathbf{NVERTEX}}$ to $\overline{\mathbf{NVERTEX}}$.



(A) **(B)**

Figure 5.818: Initial and final graph of the TREE_RESOURCE constraint

5.414 TWIN

▶ ▷ ◁ ◀ DESCRIPTION LINKS

Origin

Pairs of variables related by hiden ELEMENT constraints sharing the same table.

Constraint

TWIN(PAIRS)

Argument

PAIRS : collection(x−dvar, y−dvar)

Restrictions

required(PAIRS, x)
required(PAIRS, y)
$|\text{PAIRS}| > 0$

Purpose

Enforce the condition $\text{PAIRS}[i].\text{x} = u \wedge \text{PAIRS}[i].\text{y} = v \ (i \in [1, |\text{PAIRS}|]) \Rightarrow \forall j \in [1, |\text{PAIRS}|] : \text{PAIRS}[j].\text{x} = u \Leftrightarrow \text{PAIRS}[j].\text{y} = v$.

Example

$$\left(\left\langle\begin{array}{ll} \text{x}-1 & \text{y}-8, \\ \text{x}-9 & \text{y}-6, \\ \text{x}-1 & \text{y}-8, \\ \text{x}-5 & \text{y}-0, \\ \text{x}-6 & \text{y}-7, \\ \text{x}-9 & \text{y}-6 \end{array}\right\rangle\right)$$

The TWIN constraint holds since 1 is paired with 8, 9 is paired with 6, 5 is paired with 0, 6 is paired with 7.

Typical

$|\text{PAIRS}| > 1$
$|\text{PAIRS}| >$ nval(PAIRS.x)
$|\text{PAIRS}| >$ nval(PAIRS.y)
nval(PAIRS.x) > 1
nval(PAIRS.y) > 1
nval(PAIRS.x) = nval(PAIRS.y)
nval(PAIRS.x) $< |\text{PAIRS}|$
nval(PAIRS.y) $< |\text{PAIRS}|$

Arg. properties

Contractible wrt. PAIRS.

See also

implied by: CIRCUIT, DERANGEMENT, PROPER_CIRCUIT, SYMMETRIC_ALLDIFFERENT_LOOP.

related: ELEMENT *(pairs linked by an element with the same table)*.

Keywords

characteristic of a constraint: pair.

constraint type: predefined constraint.

5.415 TWO_LAYER_EDGE_CROSSING

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Inspired by [212].

Constraint

$$\text{TWO\_LAYER\_EDGE\_CROSSING}\begin{pmatrix}\texttt{NCROSS},\\ \texttt{VERTICES\_LAYER1},\\ \texttt{VERTICES\_LAYER2},\\ \texttt{EDGES}\end{pmatrix}$$

Arguments

NCROSS	:	dvar
VERTICES_LAYER1	:	collection(id−int, pos−dvar)
VERTICES_LAYER2	:	collection(id−int, pos−dvar)
EDGES	:	collection(id−int, vertex1−int, vertex2−int)

Restrictions

NCROSS ≥ 0
required(VERTICES_LAYER1, [id, pos])
VERTICES_LAYER1.id ≥ 1
VERTICES_LAYER1.id $\leq$ |VERTICES_LAYER1|
distinct(VERTICES_LAYER1, id)
distinct(VERTICES_LAYER1, pos)
required(VERTICES_LAYER2, [id, pos])
VERTICES_LAYER2.id ≥ 1
VERTICES_LAYER2.id $\leq$ |VERTICES_LAYER2|
distinct(VERTICES_LAYER2, id)
distinct(VERTICES_LAYER2, pos)
required(EDGES, [id, vertex1, vertex2])
EDGES.id ≥ 1
EDGES.id $\leq$ |EDGES|
distinct(EDGES, id)
EDGES.vertex1 ≥ 1
EDGES.vertex1 $\leq$ |VERTICES_LAYER1|
EDGES.vertex2 ≥ 1
EDGES.vertex2 $\leq$ |VERTICES_LAYER2|

Purpose NCROSS is the number of line segments intersections.

Example

$$\left(\begin{array}{l} 2, \langle \texttt{id}-1\ \texttt{pos}-1, \texttt{id}-2\ \texttt{pos}-2\rangle, \\ \langle \texttt{id}-1\ \texttt{pos}-3, \texttt{id}-2\ \texttt{pos}-1, \texttt{id}-3\ \texttt{pos}-2\rangle, \\ \left\langle\begin{array}{lll} \texttt{id}-1 & \texttt{vertex1}-2 & \texttt{vertex2}-2, \\ \texttt{id}-2 & \texttt{vertex1}-2 & \texttt{vertex2}-3, \\ \texttt{id}-3 & \texttt{vertex1}-1 & \texttt{vertex2}-1 \end{array}\right\rangle \end{array}\right)$$

Figure 5.819 provides a picture of the example, where one can see the two line segments intersections. Each line segment of Figure 5.819 is labelled with its identifier and corresponds to an item of the EDGES collection. The two vertices on top of Figure 5.819 correspond to the items of the VERTICES_LAYER1 collection, while the three other vertices are associated with the items of VERTICES_LAYER2.

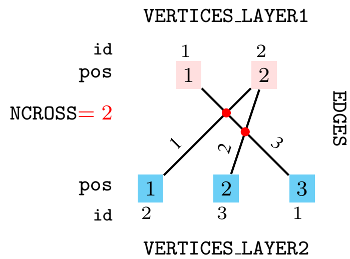


Figure 5.819: Intersection between line segments joining two layers of the **Example** slot for the constraint TWO_LAYER_EDGE_CROSSING(NCROSS, VERTICES_LAYER1, VERTICES_LAYER2, EDGES)

Typical

$|\mathtt{VERTICES\_LAYER1}| > 1$
$|\mathtt{VERTICES\_LAYER2}| > 1$
$|\mathtt{EDGES}| \geq |\mathtt{VERTICES\_LAYER1}|$
$|\mathtt{EDGES}| \geq |\mathtt{VERTICES\_LAYER2}|$

Symmetries

- Arguments are permutable w.r.t. permutation (NCROSS) (VERTICES_LAYER1, VERTICES_LAYER2) (EDGES).
- Items of VERTICES_LAYER1 are permutable.
- Items of VERTICES_LAYER2 are permutable.

Arg. properties

Functional dependency: NCROSS determined by VERTICES_LAYER1, VERTICES_LAYER2 and EDGES.

Remark

The two-layer edge crossing minimisation problem was proved to be NP-hard in [195].

See also

common keyword: CROSSING, GRAPH_CROSSING *(line segments intersection)*.

Keywords

characteristic of a constraint: derived collection.

constraint arguments: pure functional dependency.

geometry: geometrical constraint, line segments intersection.

miscellaneous: obscure.

modelling: functional dependency.

Derived Collection

$$\texttt{col}\left(\begin{array}{l}\texttt{EDGES\_EXTREMITIES}-\texttt{collection}(\texttt{layer1}-\texttt{dvar},\texttt{layer2}-\texttt{dvar}),\\ \left[\texttt{item}\left(\begin{array}{l}\texttt{layer1}-\texttt{EDGES.vertex1}(\texttt{VERTICES\_LAYER1},\texttt{pos},\texttt{id}),\\ \texttt{layer2}-\texttt{EDGES.vertex2}(\texttt{VERTICES\_LAYER2},\texttt{pos},\texttt{id})\end{array}\right)\right]\end{array}\right)$$

Arc input(s) EDGES_EXTREMITIES

Arc generator $CLIQUE(<)\mapsto\texttt{collection}(\texttt{edges\_extremities1},\texttt{edges\_extremities2})$

Arc arity 2

Arc constraint(s)

$$\bigvee\left(\begin{array}{l}\bigwedge\left(\begin{array}{l}\texttt{edges\_extremities1.layer1}<\texttt{edges\_extremities2.layer1},\\ \texttt{edges\_extremities1.layer2}>\texttt{edges\_extremities2.layer2}\end{array}\right),\\ \bigwedge\left(\begin{array}{l}\texttt{edges\_extremities1.layer1}>\texttt{edges\_extremities2.layer1},\\ \texttt{edges\_extremities1.layer2}<\texttt{edges\_extremities2.layer2}\end{array}\right)\end{array}\right)$$

Graph property(ies) $\mathbf{NARC}=\texttt{NCROSS}$

Graph model As usual for the two-layer edge crossing problem [212], [24], positions of the vertices on each layer are represented as a permutation of the vertices. We generate a derived collection that, for each edge, contains the position of its extremities on both layers. In the arc generator we use the restriction $<$ in order to generate a single arc for each pair of segments. This is required, since otherwise we would count more than once a line segments intersection.

Parts (A) and (B) of Figure 5.820 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

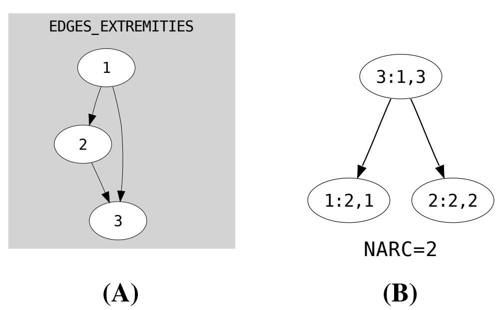


Figure 5.820: Initial and final graph of the TWO_LAYER_EDGE_CROSSING constraint

5.416 TWO_ORTH_ARE_IN_CONTACT

DESCRIPTION LINKS GRAPH AUTOMATON

Origin [369], used for defining ORTHS_ARE_CONNECTED.

Constraint TWO_ORTH_ARE_IN_CONTACT(ORTHOTOPE1, ORTHOTOPE2)

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments

ORTHOTOPE1 : ORTHOTOPE
ORTHOTOPE2 : ORTHOTOPE

Restrictions

|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz > 0
ORTHOTOPE.ori ≤ ORTHOTOPE.end
|ORTHOTOPE1| = |ORTHOTOPE2|
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE1)
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE2)

Purpose

Enforce the following conditions on two orthotopes O_1 and O_2:

- For all dimensions i, except one dimension, the projections of O_1 and O_2 onto i have a non-empty intersection.
- For all dimensions i, the distance between the projections of O_1 and O_2 onto i is equal to 0.

Example

$$\left(\begin{array}{l}\langle \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-5\ \texttt{siz}-2\ \texttt{end}-7\rangle,\\ \langle \texttt{ori}-3\ \texttt{siz}-2\ \texttt{end}-5, \texttt{ori}-2\ \texttt{siz}-3\ \texttt{end}-5\rangle\end{array}\right)$$

Figure 5.821 shows the two rectangles of the example. The TWO_ORTH_ARE_IN_CONTACT constraint holds since the two rectangles are in contact: the contact is depicted by a pink line-segment.

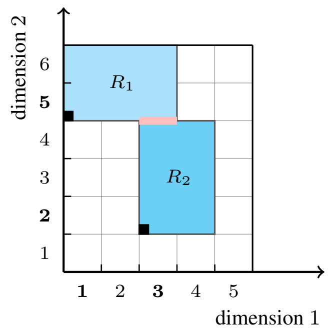
Typical |ORTHOTOPE| > 1

Symmetries

- Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2).
- Items of ORTHOTOPE1 and ORTHOTOPE2 are permutable (*same permutation used*).

Used in ORTHS_ARE_CONNECTED.

See also **implies:** TWO_ORTH_DO_NOT_OVERLAP.



R_1:	$\langle \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-5\ \texttt{siz}-2\ \texttt{end}-7\rangle$
R_2:	$\langle \texttt{ori}-3\ \texttt{siz}-2\ \texttt{end}-5, \texttt{ori}-2\ \texttt{siz}-3\ \texttt{end}-5\rangle$

`ORTHOTOPES` (rectangles)

Figure 5.821: The two rectangles that are in contact of the **Example** slot where the contact is shown in pink

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: logic.

filtering: arc-consistency.

geometry: geometrical constraint, touch, contact, non-overlapping, orthotope.

Arc input(s) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)

Arc arity 2

Arc constraint(s)
- orthotope1.end > orthotope2.ori
- orthotope2.end > orthotope1.ori

Graph property(ies) $\mathbf{NARC} = |\mathtt{ORTHOTOPE1}| - 1$

Arc input(s) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)

Arc arity 2

Arc constraint(s) $\mathtt{max}\left(0, \begin{array}{l}\mathtt{max}(\mathtt{orthotope1.ori}, \mathtt{orthotope2.ori}) - \\ \mathtt{min}(\mathtt{orthotope1.end}, \mathtt{orthotope2.end})\end{array}\right) = 0$

Graph property(ies) $\mathbf{NARC} = |\mathtt{ORTHOTOPE1}|$

Graph model Parts (A) and (B) of Figure 5.822 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold. It corresponds to the fact that the projection onto dimension 1 of the two rectangles of the example overlap.

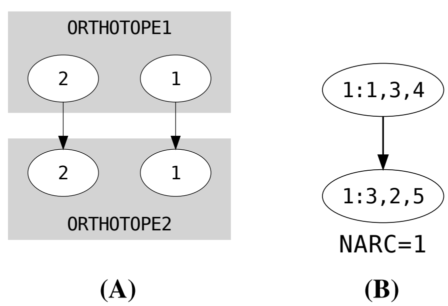


Figure 5.822: Initial and final graph of the TWO_ORTH_ARE_IN_CONTACT constraint

Signature Consider the second graph constraint. Since we use the arc generator $PRODUCT(=)$ on the collections ORTHOTOPE1 and ORTHOTOPE2, and because of the restriction $|\mathtt{ORTHOTOPE1}| = |\mathtt{ORTHOTOPE2}|$, the maximum number of arcs of the corresponding final graph is equal to $|\mathtt{ORTHOTOPE1}|$. Therefore we can rewrite the graph property $\mathbf{NARC} = |\mathtt{ORTHOTOPE1}|$ to $\mathbf{NARC} \geq |\mathtt{ORTHOTOPE1}|$ and simplify $\underline{\overline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton Figure 5.823 depicts the automaton associated with the TWO_ORTH_ARE_IN_CONTACT constraint. Let $\mathtt{ORI1}_i$, $\mathtt{SIZ1}_i$ and $\mathtt{END1}_i$ respectively be the `ori`, the `siz` and the `end` attributes of the i^{th} item of the `ORTHOTOPE1` collection. Let $\mathtt{ORI2}_i$, $\mathtt{SIZ2}_i$ and $\mathtt{END2}_i$ respectively be the `ori`, the `siz` and the `end` attributes of the i^{th} item of the `ORTHOTOPE2` collection. To each sextuple $(\mathtt{ORI1}_i, \mathtt{SIZ1}_i, \mathtt{END1}_i, \mathtt{ORI2}_i, \mathtt{SIZ2}_i, \mathtt{END2}_i)$ corresponds a signature variable S_i, which takes its value in $\{0, 1, 2\}$, as well as the following signature constraint:

$$((\mathtt{SIZ1}_i > 0) \wedge (\mathtt{SIZ2}_i > 0) \wedge (\mathtt{END1}_i > \mathtt{ORI2}_i) \wedge (\mathtt{END2}_i > \mathtt{ORI1}_i)) \Leftrightarrow S_i = 0$$

$$((\mathtt{SIZ1}_i > 0) \wedge (\mathtt{SIZ2}_i > 0) \wedge (\mathtt{END1}_i = \mathtt{ORI2}_i \vee \mathtt{END2}_i = \mathtt{ORI1}_i)) \Leftrightarrow S_i = 1.$$

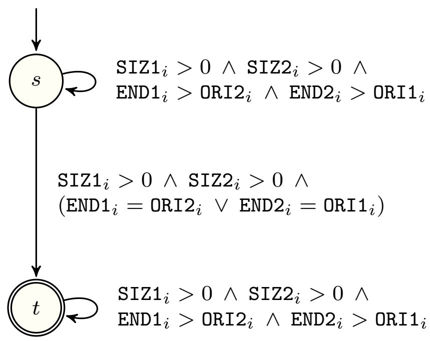


Figure 5.823: Automaton of the TWO_ORTH_ARE_IN_CONTACT constraint

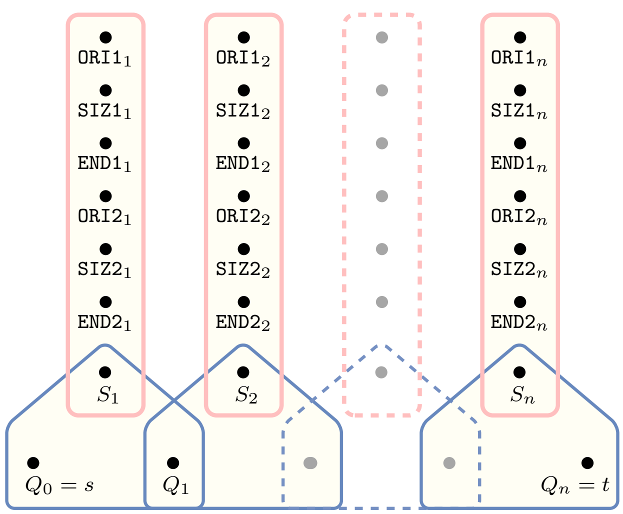


Figure 5.824: Hypergraph of the reformulation corresponding to the automaton of the TWO_ORTH_ARE_IN_CONTACT constraint

5.417 TWO_ORTH_COLUMN

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Used for defining DIFFN_COLUMN.

Constraint TWO_ORTH_COLUMN(ORTHOTOPE1, ORTHOTOPE2, DIM)

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments

ORTHOTOPE1	:	ORTHOTOPE
ORTHOTOPE2	:	ORTHOTOPE
DIM	:	int

Restrictions

|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori ≤ ORTHOTOPE.end
|ORTHOTOPE1| = |ORTHOTOPE2|
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE1)
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE2)
DIM > 0
DIM ≤ |ORTHOTOPE1|

Purpose Let P_1 and P_2 respectively denote the projections of ORTHOTOPE1 and ORTHOTOPE2 onto dimension DIM. If P_1 and P_2 overlap then the size of their intersection is equal to the size of ORTHOTOPE1 in dimension DIM, as well as to the size of ORTHOTOPE2 in dimension DIM.

Example

$$\left(\begin{array}{l}\langle \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-1\ \texttt{end}-2\rangle,\\ \langle \texttt{ori}-4\ \texttt{siz}-2\ \texttt{end}-6, \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4\rangle, 1\end{array}\right)$$

Typical |ORTHOTOPE| > 1

Symmetry Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2) (DIM).

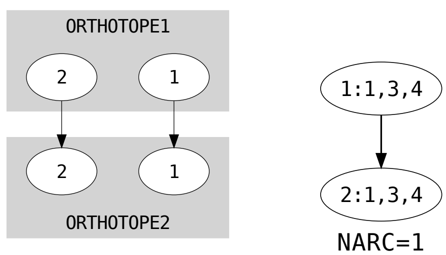
Used in DIFFN_COLUMN.

See also **implies:** TWO_ORTH_INCLUDE.

related: DIFFN *(an extension of the* DIFFN *constraint).*

Keywords **constraint type:** logic.

geometry: geometrical constraint, positioning constraint, orthotope, guillotine cut.



(A) **(B)**

Figure 5.825: Initial and final graph of the TWO_ORTH_COLUMN constraint

Arc input(s) ORTHOTOPE1 ORTHOTOPE2

Arc generator $PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)

Arc arity 2

Arc constraint(s)

$$\bigwedge\left(\begin{array}{l}\texttt{orthotope1.key} = \texttt{DIM},\\ \texttt{orthotope1.ori} < \texttt{orthotope2.end},\\ \texttt{orthotope2.ori} < \texttt{orthotope1.end},\\ \texttt{orthotope1.siz} > 0,\\ \texttt{orthotope2.siz} > 0\end{array}\right) \Rightarrow \bigwedge\left(\begin{array}{l}\dfrac{\begin{array}{l}\texttt{min}(\texttt{orthotope1.end}, \texttt{orthotope2.end}) -\\ \texttt{max}(\texttt{orthotope1.ori}, \texttt{orthotope2.ori})\end{array}}{\texttt{orthotope1.siz}} = ,\\ \texttt{orthotope1.siz} = \texttt{orthotope2.siz}\end{array}\right)$$

Graph property(ies) **NARC** $= 1$

5.418 TWO_ORTH_DO_NOT_OVERLAP

DESCRIPTION LINKS GRAPH AUTOMATON

Origin Used for defining DIFFN.

Constraint TWO_ORTH_DO_NOT_OVERLAP(ORTHOTOPE1, ORTHOTOPE2)

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments

ORTHOTOPE1 : ORTHOTOPE
ORTHOTOPE2 : ORTHOTOPE

Restrictions

|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori ≤ ORTHOTOPE.end
|ORTHOTOPE1| = |ORTHOTOPE2|
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE1)
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE2)

Purpose For two orthotopes O_1 and O_2 enforce that there exists at least one dimension i such that the projections on i of O_1 and O_2 do not overlap.

Example

$$\left(\begin{array}{l}\langle \texttt{ori}-2\ \texttt{siz}-2\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4\rangle, \\ \langle \texttt{ori}-4\ \texttt{siz}-4\ \texttt{end}-8, \texttt{ori}-3\ \texttt{siz}-3\ \texttt{end}-6\rangle\end{array}\right)$$

Figure 5.826 represents the respective position of the two rectangles of the example. The coordinates of the leftmost lowest corner of each rectangle are stressed in bold. The TWO_ORTH_DO_NOT_OVERLAP constraint holds since the two rectangles do not overlap.

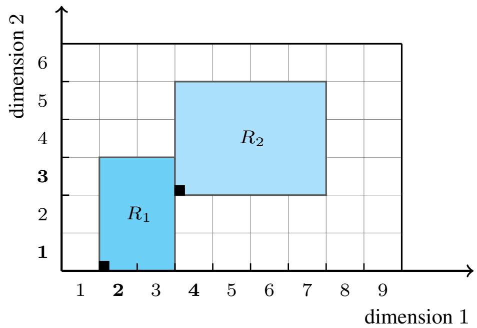
Typical |ORTHOTOPE| > 1

Symmetries

- Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2).
- Items of ORTHOTOPE1 and ORTHOTOPE2 are permutable (*same permutation used*).
- ORTHOTOPE1.siz can be decreased to any value ≥ 0.
- ORTHOTOPE2.siz can be decreased to any value ≥ 0.

Used in DIFFN.

See also **implied by:** TWO_ORTH_ARE_IN_CONTACT.



R_1: $\langle \texttt{ori}-2\ \texttt{siz}-2\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4\rangle$
R_2: $\langle \texttt{ori}-4\ \texttt{siz}-4\ \texttt{end}-8, \texttt{ori}-3\ \texttt{siz}-3\ \texttt{end}-6\rangle$

`ORTHOTOPES` (rectangles)

Figure 5.826: The two non overlapping rectangles of the **Example** slot

Keywords

characteristic of a constraint: automaton, automaton without counters, reified automaton constraint.

constraint network structure: Berge-acyclic constraint network.

constraint type: logic.

filtering: arc-consistency, constructive disjunction.

final graph structure: bipartite, no loop.

geometry: geometrical constraint, non-overlapping, orthotope.

Arc input(s)	ORTHOTOPE1 ORTHOTOPE2
Arc generator	$SYMMETRIC\_PRODUCT(=) \mapsto$ collection(orthotope1, orthotope2)
Arc arity	2
Arc constraint(s)	orthotope1.end $\leq$ orthotope2.ori $\vee$ orthotope1.siz $= 0$
Graph property(ies)	**NARC** ≥ 1
Graph class	• BIPARTITE • NO_LOOP

Graph model We build an initial graph where each arc corresponds to the fact that, either the projection of an orthotope on a given dimension is empty, either it is located before the projection in the same dimension of the other orthotope. Finally we ask that at least one arc constraint remains in the final graph.

Parts (A) and (B) of Figure 5.827 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the unique arc of the final graph is stressed in bold. It corresponds to the fact that the projection in dimension 1 of the first orthotope is located before the projection in dimension 1 of the second orthotope. Therefore the two orthotopes do not overlap.

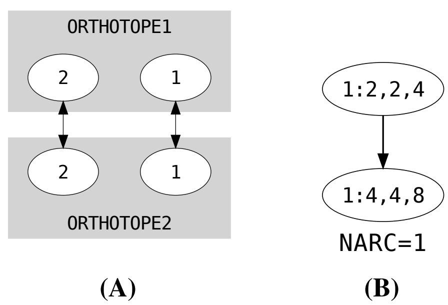


Figure 5.827: Initial and final graph of the TWO_ORTH_DO_NOT_OVERLAP constraint

Automaton Figure 5.828 depicts the automaton associated with the TWO_ORTH_DO_NOT_OVERLAP constraint. Let $\mathtt{ORI1}_i$, $\mathtt{SIZ1}_i$ and $\mathtt{END1}_i$ respectively be the ori, the siz and the end attributes of the i^{th} item of the ORTHOTOPE1 collection. Let $\mathtt{ORI2}_i$, $\mathtt{SIZ2}_i$ and $\mathtt{END2}_i$ respectively be the ori, the siz and the end attributes of the i^{th} item of the ORTHOTOPE2 collection. To each sextuple $(\mathtt{ORI1}_i, \mathtt{SIZ1}_i, \mathtt{END1}_i, \mathtt{ORI2}_i, \mathtt{SIZ2}_i, \mathtt{END2}_i)$ corresponds a 0-1 signature variable S_i as well as the following signature constraint: $((\mathtt{SIZ1}_i > 0) \wedge (\mathtt{SIZ2}_i > 0) \wedge (\mathtt{END1}_i > \mathtt{ORI2}_i) \wedge (\mathtt{END2}_i > \mathtt{ORI1}_i)) \Leftrightarrow S_i$.

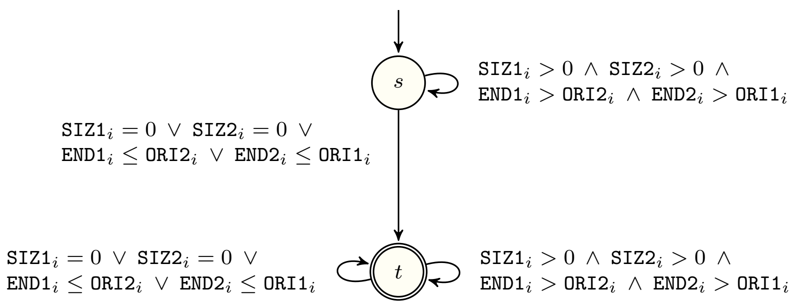


Figure 5.828: Automaton of the TWO_ORTH_DO_NOT_OVERLAP constraint

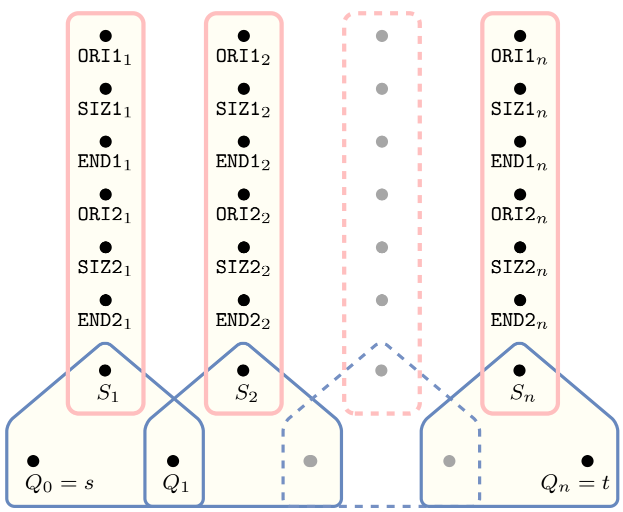


Figure 5.829: Hypergraph of the reformulation corresponding to the automaton of the TWO_ORTH_DO_NOT_OVERLAP constraint

5.419 TWO_ORTH_INCLUDE

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Used for defining DIFFN_INCLUDE.

Constraint TWO_ORTH_INCLUDE(ORTHOTOPE1, ORTHOTOPE2, DIM)

Type ORTHOTOPE : collection(ori−dvar, siz−dvar, end−dvar)

Arguments

ORTHOTOPE1	:	ORTHOTOPE
ORTHOTOPE2	:	ORTHOTOPE
DIM	:	int

Restrictions

|ORTHOTOPE| > 0
require_at_least(2, ORTHOTOPE, [ori, siz, end])
ORTHOTOPE.siz ≥ 0
ORTHOTOPE.ori ≤ ORTHOTOPE.end
|ORTHOTOPE1| = |ORTHOTOPE2|
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE1)
ORTH_LINK_ORI_SIZ_END(ORTHOTOPE2)
DIM > 0
DIM ≤ |ORTHOTOPE1|

Purpose Let P_1 and P_2 respectively denote the projections of ORTHOTOPE1 and ORTHOTOPE2 onto dimension DIM. If P_1 and P_2 overlap then, either P_1 is included in P_2, either P_2 is included in P_1.

Example

$$\left(\begin{array}{l}\langle \texttt{ori}-1\ \texttt{siz}-3\ \texttt{end}-4, \texttt{ori}-1\ \texttt{siz}-1\ \texttt{end}-2\rangle, \\ \langle \texttt{ori}-1\ \texttt{siz}-2\ \texttt{end}-3, \texttt{ori}-2\ \texttt{siz}-3\ \texttt{end}-5\rangle, 1\end{array}\right)$$

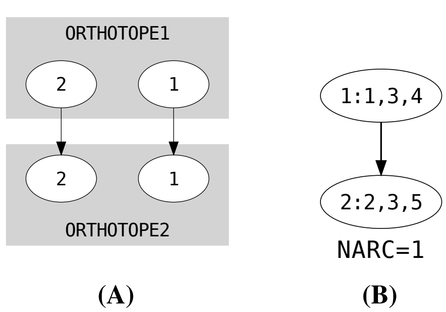


Figure 5.830: Initial and final graph of the TWO_ORTH_INCLUDE constraint

Typical $|\texttt{ORTHOTOPE}| > 1$

Symmetry Arguments are permutable w.r.t. permutation (ORTHOTOPE1, ORTHOTOPE2) (DIM).

Used in DIFFN_INCLUDE.

See also **implied by:** TWO_ORTH_COLUMN.

related: DIFFN *(an extension of the* DIFFN *constraint)*.

Keywords **constraint type:** logic.

geometry: geometrical constraint, positioning constraint, orthotope.

Arc input(s)	ORTHOTOPE1 ORTHOTOPE2
Arc generator	$PRODUCT(=) \mapsto \texttt{collection}(\texttt{orthotope1}, \texttt{orthotope2})$
Arc arity	2
Arc constraint(s)	$\bigwedge \left(\begin{array}{l} \texttt{orthotope1.key} = \texttt{DIM}, \\ \texttt{orthotope1.ori} < \texttt{orthotope2.end}, \\ \texttt{orthotope2.ori} < \texttt{orthotope1.end}, \\ \texttt{orthotope1.siz} > 0, \\ \texttt{orthotope2.siz} > 0 \end{array} \right) \Rightarrow$ $\begin{array}{l} \texttt{min}(\texttt{orthotope1.end}, \texttt{orthotope2.end}) - \\ \texttt{max}(\texttt{orthotope1.ori}, \texttt{orthotope2.ori}) \end{array} = \texttt{min}(\texttt{orthotope1.siz}, \texttt{orthotope2.siz})$
Graph property(ies)	**NARC** $= 1$

5.420 USED_BY

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH AUTOMATON

Origin N. Beldiceanu

Constraint USED_BY(VARIABLES1, VARIABLES2)

Arguments

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

Restrictions

$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose All the values of the variables of collection VARIABLES2 are used by the variables of collection VARIABLES1.

Example $(\langle 1,9,1,5,2,1\rangle, \langle 1,1,2,5\rangle)$

The USED_BY constraint holds since, for each value occurring within the collection VARIABLES2 $= \langle 1,1,2,5\rangle$, its number of occurrences within VARIABLES1 $= \langle 1,9,1,5,2,1\rangle$ is greater than or equal to its number of occurrences within VARIABLES2:

- Value 1 occurs 3 times within $\langle 1,9,1,5,2,1\rangle$ and 2 times within $\langle 1,1,2,5\rangle$.
- Value 2 occurs 1 times within $\langle 1,9,1,5,2,1\rangle$ and 1 times within $\langle 1,1,2,5\rangle$.
- Value 5 occurs 1 times within $\langle 1,9,1,5,2,1\rangle$ and 1 times within $\langle 1,1,2,5\rangle$.

All solutions Figure 5.831 gives all solutions to the following non ground instance of the USED_BY constraint: $\texttt{U}_1 \in \{1,5\}$, $\texttt{U}_2 \in [1,2]$, $\texttt{U}_3 \in [1,2]$, $\texttt{V}_1 \in [0,2]$, $\texttt{V}_2 \in [2,4]$, USED_BY$(\langle \texttt{U}_1,\texttt{U}_2,\texttt{U}_3\rangle, \langle \texttt{V}_1,\texttt{V}_2\rangle)$.

Typical

$|\texttt{VARIABLES1}| > 1$
range(VARIABLES1.var) > 1
$|\texttt{VARIABLES2}| > 1$
range(VARIABLES2.var) > 1

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

① $(\langle 1, 1, 2\rangle, \langle 1, 2\rangle)$
② $(\langle 1, 2, 1\rangle, \langle 1, 2\rangle)$
③ $(\langle 1, 2, 2\rangle, \langle 1, 2\rangle)$
④ $(\langle 1, 2, 2\rangle, \langle 2, 2\rangle)$
⑤ $(\langle 5, 1, 2\rangle, \langle 1, 2\rangle)$
⑥ $(\langle 5, 2, 1\rangle, \langle 1, 2\rangle)$
⑦ $(\langle 5, 2, 2\rangle, \langle 2, 2\rangle)$

Figure 5.831: All solutions corresponding to the non ground example of the USED_BY constraint of the **All solutions** slot where identical values are coloured in the same way in both collections

Arg. properties

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union).

Algorithm

As described in [53] we can pad VARIABLES2 with dummy variables such that its cardinality will be equal to that cardinality of VARIABLES1. The domain of a dummy variable contains all of the values. Then, we have a SAME constraint between the two sets. Direct arc-consistency and bound-consistency algorithms based on a flow model are also proposed in [53, 55, 242]. The leftmost part of Figure 3.31 illustrates this flow model.

More recently [138, 139] presents a second filtering algorithm also achieving arc-consistency based on a mapping of the solutions to the USED_BY constraint to var-perfect matchings[17] in a bipartite intersection graph derived from the domain of the variables of the constraint in the following way. To each variable of the VARIABLES1 and VARIABLES2 collection corresponds a vertex of the intersection graph. There is an edge between a vertex associated with a variable of the VARIABLES1 collection and a vertex associated with a variable of the VARIABLES2 collection if and only if the corresponding variables have at least one value in common in their domains.

Reformulation

The USED_BY$(\langle \texttt{var}-U_1\ \texttt{var}-U_2, \ldots, \texttt{var}-U_{|\texttt{VARIABLES1}|}\rangle, \langle \texttt{var}-V_1\ \texttt{var}-V_2, \ldots, \texttt{var}-V_{|\texttt{VARIABLES2}|}\rangle)$ constraint can be expressed in term of a conjunction of $|\texttt{VARIABLES2}|$ reified constraints of the form:

$$\sum_{1\leq j\leq|\texttt{VARIABLES1}|}(V_i = U_j) \geq \sum_{1\leq j\leq|\texttt{VARIABLES2}|}(V_i = V_j)\ (i \in [1, |\texttt{VARIABLES2}|]).$$

Used in

INT_VALUE_PRECEDE_CHAIN, K_USED_BY.

See also

generalisation: USED_BY_INTERVAL *(*variable *replaced by* variable/constant*)*, USED_BY_MODULO *(*variable *replaced by* variable mod constant*)*, USED_BY_PARTITION *(*variable *replaced by* variable $\in$ partition*)*.

implied by: SAME.

implies: USES.

[17] A *var-perfect matching* is a maximum matching covering all vertices corresponding to the variables of VARIABLES2.

soft variant: SOFT_USED_BY_VAR *(variable-based violation measure)*.

system of constraints: K_USED_BY.

Keywords

characteristic of a constraint: sort based reformulation, automaton, automaton with array of counters.

combinatorial object: multiset.

constraint arguments: constraint between two collections of variables.

filtering: flow, bipartite matching, arc-consistency, bound-consistency, DFS-bottleneck.

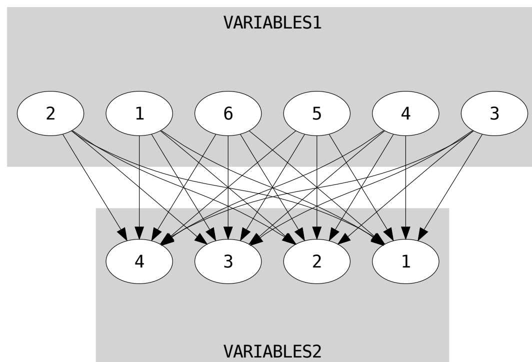
modelling: inclusion.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	*PRODUCT* $\mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var = variables2.var
Graph property(ies)	• for all connected components: **NSOURCE** $\geq$ **NSINK** • **NSINK** $=$ |VARIABLES2|

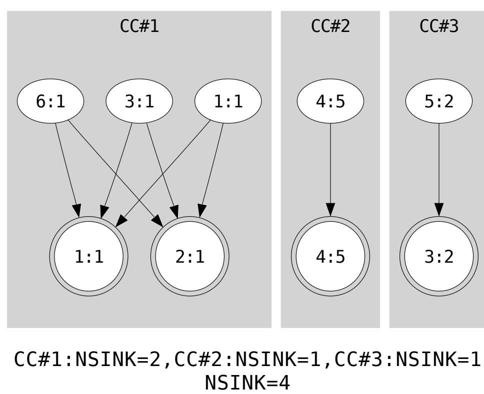
Graph model Parts (A) and (B) of Figure 5.832 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable assigned to value 9 was removed from the final graph since there is no arc for which the associated equality constraint holds. The USED_BY constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to |VARIABLES2|.

Signature Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to |VARIABLES2|. Therefore we can rewrite **NSINK** $=$ |VARIABLES2| to **NSINK** $\geq$ |VARIABLES2| and simplify $\underline{\overline{\mathbf{NSINK}}}$ to $\overline{\mathbf{NSINK}}$.



(A)



(B)

Figure 5.832: Initial and final graph of the USED_BY constraint

Automaton Figure 5.833 depicts the automaton associated with the USED_BY constraint. To each item of the collection VARIABLES1 corresponds a signature variable S_i that is equal to 0. To each item of the collection VARIABLES2 corresponds a signature variable $S_{i+|\texttt{VARIABLES1}|}$ that is equal to 1.

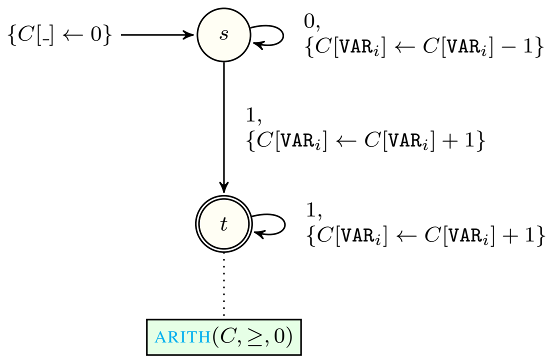


Figure 5.833: Automaton of the USED_BY constraint

5.421 USED_BY_INTERVAL

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin

Derived from USED_BY.

Constraint

USED_BY_INTERVAL(VARIABLES1, VARIABLES2, SIZE_INTERVAL)

Arguments

VARIABLES1	:	collection(var−dvar)
VARIABLES2	:	collection(var−dvar)
SIZE_INTERVAL	:	int

Restrictions

$|\mathtt{VARIABLES1}| \geq |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\mathtt{SIZE\_INTERVAL} > 0$

Purpose

Let N_i (respectively M_i) denote the number of variables of the collection VARIABLES1 (respectively VARIABLES2) that take a value in the interval $[\mathtt{SIZE\_INTERVAL} \cdot i, \mathtt{SIZE\_INTERVAL} \cdot i + \mathtt{SIZE\_INTERVAL} - 1]$. For all integer i we have $M_i > 0 \Rightarrow N_i \geq M_i$.

Example

$(\langle 1, 9, 1, 8, 6, 2\rangle, \langle 1, 0, 7, 7\rangle, 3)$

In the example, the third argument $\mathtt{SIZE\_INTERVAL} = 3$ defines the following family of intervals $[3 \cdot k, 3 \cdot k + 2]$, where k is an integer. Consequently the values of the collection $\mathtt{VARIABLES2} = \langle 1, 0, 7, 7\rangle$ are respectively located within intervals $[0, 2]$, $[0, 2]$, $[6, 8]$, $[6, 8]$. Therefore intervals $[0, 2]$ and $[6, 8]$ are respectively used 2 and 2 times.

Similarly, the values of the collection $\mathtt{VARIABLES1} = \langle 1, 9, 1, 8, 6, 2\rangle$ are respectively located within intervals $[0, 2]$, $[9, 11]$, $[0, 2]$, $[6, 8]$, $[6, 8]$, $[0, 2]$. Therefore intervals $[0, 2]$, $[6, 8]$ and $[9, 11]$ are respectively used 3, 2 and 1 times.

Consequently, the USED_BY_INTERVAL constraint holds since, for each interval associated with the collection $\mathtt{VARIABLES2} = \langle 1, 0, 7, 7\rangle$, its number of occurrences within $\mathtt{VARIABLES1} = \langle 1, 9, 1, 8, 6, 2\rangle$ is greater than or equal to its number of occurrences within VARIABLES2:

- Interval $[0, 2]$ occurs 3 times within $\langle 1, 9, 1, 8, 6, 2\rangle$ and 2 times within $\langle 1, 0, 7, 7\rangle$.
- Interval $[6, 8]$ occurs 2 times within $\langle 1, 9, 1, 8, 6, 2\rangle$ and 2 times within $\langle 1, 0, 7, 7\rangle$.

Typical

$|\mathtt{VARIABLES1}| > 1$
range(VARIABLES1.var) > 1
$|\mathtt{VARIABLES2}| > 1$
range(VARIABLES2.var) > 1
$\mathtt{SIZE\_INTERVAL} > 1$
SIZE_INTERVAL < range(VARIABLES1.var)
SIZE_INTERVAL < range(VARIABLES2.var)

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value of VARIABLES1.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.
- An occurrence of a value of VARIABLES2.var that belongs to the k-th interval, of size SIZE_INTERVAL, can be replaced by any other value of the same interval.

Arg. properties

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union), SIZE_INTERVAL(id).

Reformulation

The USED_BY_INTERVAL($\langle \texttt{var}-U_1\ \texttt{var}-U_2, \dots, \texttt{var}-U_{|\texttt{VARIABLES1}|}\rangle$, $\langle \texttt{var}-V_1\ \texttt{var}-V_2, \dots, \texttt{var}-V_{|\texttt{VARIABLES2}|}\rangle$, SIZE_INTERVAL) constraint can be expressed by introducing $|\texttt{VARIABLES1}| + |\texttt{VARIABLES2}|$ *quotient* variables
$U_i = \texttt{SIZE\_INTERVAL} \cdot P_i + R_i$, $R_i \in [0, \texttt{SIZE\_INTERVAL}-1]$ $(i \in [1, |\texttt{VARIABLES1}|])$,
$V_i = \texttt{SIZE\_INTERVAL} \cdot Q_i + S_i$, $S_i \in [0, \texttt{SIZE\_INTERVAL}-1]$ $(i \in [1, |\texttt{VARIABLES2}|])$,
in term of a conjunction of $|\texttt{VARIABLES2}|$ reified constraints of the form:
$\sum_{1 \le j \le |\texttt{VARIABLES1}|} (Q_i = P_j) \ge \sum_{1 \le j \le |\texttt{VARIABLES2}|} (Q_i = Q_j)$ $(i \in [1, |\texttt{VARIABLES2}|])$.

Used in

K_USED_BY_INTERVAL.

See also

implied by: SAME_INTERVAL.

soft variant: SOFT_USED_BY_INTERVAL_VAR *(variable-based violation measure)*.

specialisation: USED_BY *(*variable/constant *replaced by* variable*)*.

system of constraints: K_USED_BY_INTERVAL.

Keywords

characteristic of a constraint: sort based reformulation.

constraint arguments: constraint between two collections of variables.

modelling: inclusion, interval.

Arc input(s)	`VARIABLES1` `VARIABLES2`
Arc generator	$PRODUCT \mapsto$ `collection`(`variables1`, `variables2`)
Arc arity	2
Arc constraint(s)	`variables1.var`/`SIZE_INTERVAL` = `variables2.var`/`SIZE_INTERVAL`
Graph property(ies)	• for all connected components: **NSOURCE**≥**NSINK** • **NSINK**= |`VARIABLES2`|

Graph model Parts (A) and (B) of Figure 5.834 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The USED_BY_INTERVAL constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to |`VARIABLES2`|.

Signature Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to |`VARIABLES2`|. Therefore we can rewrite **NSINK** = |`VARIABLES2`| to **NSINK** ≥ |`VARIABLES2`| and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.

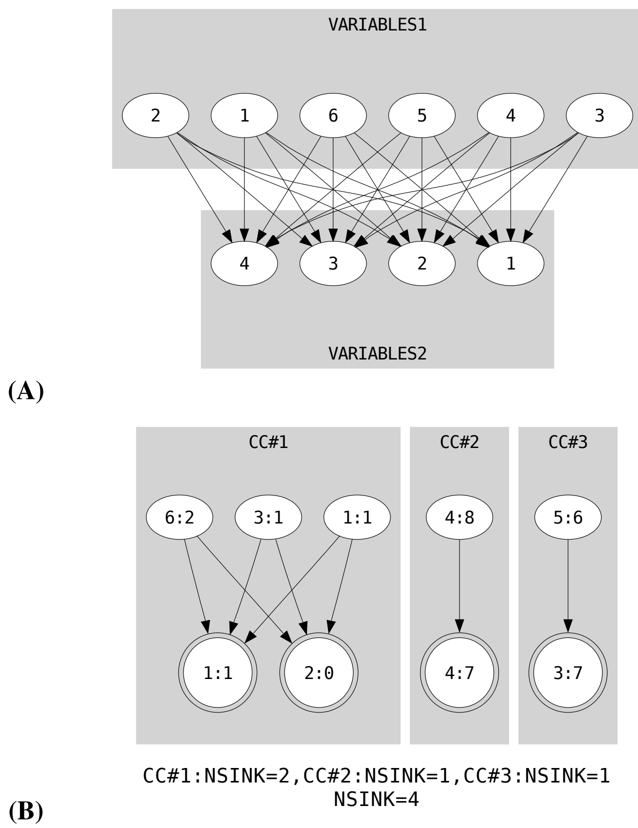


Figure 5.834: Initial and final graph of the USED_BY_INTERVAL constraint

5.422 USED_BY_MODULO

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY.

Constraint USED_BY_MODULO(VARIABLES1, VARIABLES2, M)

Arguments

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
M : int

Restrictions

$|\mathtt{VARIABLES1}| \geq |\mathtt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
$\mathtt{M} > 0$

Purpose

For each integer R in $[0, \mathtt{M}-1]$, let $N1_R$ (respectively $N2_R$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that have R as a rest when divided by M. For all R in $[0, \mathtt{M}-1]$ we have $N2_R > 0 \Rightarrow N1_R \geq N2_R$.

Example

$(\langle 1, 9, 4, 5, 2, 1\rangle, \langle 7, 1, 2, 5\rangle, 3)$

The values of the collection VARIABLES2 $= \langle 7, 1, 2, 5\rangle$ are respectively associated with the equivalence classes $7 \bmod 3 = 1$, $1 \bmod 3 = 1$, $2 \bmod 3 = 2$, $5 \bmod 3 = 2$. Therefore the equivalence classes 1 and 2 are respectively used 2 and 2 times.

Similarly, the values of the collection VARIABLES1 $= \langle 1, 9, 4, 5, 2, 1\rangle$ associated with the equivalence classes $1 \bmod 3 = 1$, $9 \bmod 3 = 0$, $4 \bmod 3 = 1$, $5 \bmod 3 = 2$, $2 \bmod 3 = 2$, $1 \bmod 3 = 1$. Therefore the equivalence classes 0, 1 and 2 are respectively used 1, 3 and 2 times.

Consequently, the USED_BY_MODULO constraint holds since, for each equivalence class associated with the collection VARIABLES2 $= \langle 7, 1, 2, 5\rangle$, its number of occurrences within VARIABLES1 $= \langle 1, 9, 4, 5, 2, 1\rangle$ is greater than or equal to its number of occurrences within VARIABLES2:

- The equivalence class 1 occurs 3 times within $\langle 1, 9, 4, 5, 2, 1\rangle$ and 2 times within $\langle 7, 1, 2, 5\rangle$.
- The equivalence class 2 occurs 2 times within $\langle 1, 9, 4, 5, 2, 1\rangle$ and 2 times within $\langle 7, 1, 2, 5\rangle$.

Typical

$|\mathtt{VARIABLES1}| > 1$
range(VARIABLES1.var) > 1
$|\mathtt{VARIABLES2}| > 1$
range(VARIABLES2.var) > 1
$\mathtt{M} > 1$
M $<$ maxval(VARIABLES1.var)
M $<$ maxval(VARIABLES2.var)

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- An occurrence of a value u of VARIABLES1.var can be replaced by any other value v such that v is congruent to u modulo M.
- An occurrence of a value u of VARIABLES2.var can be replaced by any other value v such that v is congruent to u modulo M.

Arg. properties

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union), M(id).

Used in K_USED_BY_MODULO.

See also

implied by: SAME_MODULO.

soft variant: SOFT_USED_BY_MODULO_VAR *(variable-based violation measure)*.

specialisation: USED_BY *(*variable mod constant *replaced by* variable*)*.

system of constraints: K_USED_BY_MODULO.

Keywords

characteristic of a constraint: modulo, sort based reformulation.

constraint arguments: constraint between two collections of variables.

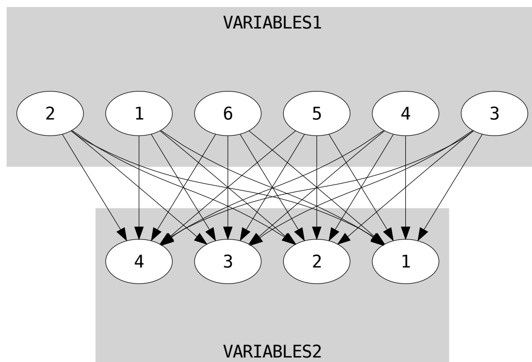
modelling: inclusion.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	variables1.var mod M = variables2.var mod M
Graph property(ies)	• for all connected components: **NSOURCE**$\geq$**NSINK** • **NSINK**$=$ |VARIABLES2|

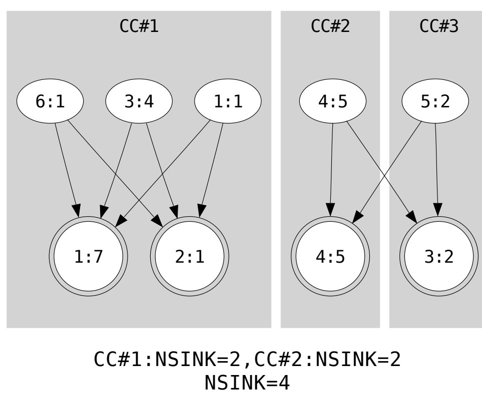
Graph model Parts (A) and (B) of Figure 5.835 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The USED_BY_MODULO constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to |VARIABLES2|.

Signature Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to |VARIABLES2|. Therefore we can rewrite **NSINK** $=$ |VARIABLES2| to **NSINK** $\geq$ |VARIABLES2| and simplify $\underline{\mathbf{NSINK}}$ to $\overline{\mathbf{NSINK}}$.



(A)



(B)

Figure 5.835: Initial and final graph of the USED_BY_MODULO constraint

5.423 USED_BY_PARTITION

▶ ▷ ◁ ◀ DESCRIPTION LINKS GRAPH

Origin Derived from USED_BY.

Constraint USED_BY_PARTITION(VARIABLES1, VARIABLES2, PARTITIONS)

Type

VALUES : collection(val−int)

Arguments

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)
PARTITIONS : collection(p − VALUES)

Restrictions

$|\texttt{VALUES}| \geq 1$
required(VALUES, val)
distinct(VALUES, val)
$|\texttt{VARIABLES1}| \geq |\texttt{VARIABLES2}|$
required(VARIABLES1, var)
required(VARIABLES2, var)
required(PARTITIONS, p)
$|\texttt{PARTITIONS}| \geq 2$

Purpose

For each integer i in $[1, |\texttt{PARTITIONS}|]$, let $N1_i$ (respectively $N2_i$) denote the number of variables of VARIABLES1 (respectively VARIABLES2) that take their values in the i^{th} partition of the collection PARTITIONS. For all i in $[1, |\texttt{PARTITIONS}|]$ we have $N2_i > 0 \Rightarrow N1_i \geq N2_i$.

Example

$$\left(\begin{array}{l} \langle 1,9,1,6,2,3\rangle, \\ \langle 1,3,6,6\rangle, \\ \langle \texttt{p}-\langle 1,3\rangle, \texttt{p}-\langle 4\rangle, \texttt{p}-\langle 2,6\rangle\rangle \end{array}\right)$$

The different values of the collection VARIABLES2 $= \langle 1,3,6,6\rangle$ are respectively associated with the partitions $\texttt{p}-\langle 1,3\rangle$, $\texttt{p}-\langle 1,3\rangle$, $\texttt{p}-\langle 2,6\rangle$, and $\texttt{p}-\langle 2,6\rangle$. Therefore partitions $\texttt{p}-\langle 1,3\rangle$ and $\texttt{p}-\langle 2,6\rangle$ are respectively used 2 and 2 times.

Similarly, the different values of the collection VARIABLES1 $= \langle 1,9,1,6,2,3\rangle$ (except value 9, which does not occur in any partition) are respectively associated with the partitions $\texttt{p}-\langle 1,3\rangle$, $\texttt{p}-\langle 1,3\rangle$, $\texttt{p}-\langle 2,6\rangle$, $\texttt{p}-\langle 2,6\rangle$, and $\texttt{p}-\langle 1,3\rangle$. Therefore partitions $\texttt{p}-\langle 1,3\rangle$ and $\texttt{p}-\langle 2,6\rangle$ are respectively used 3 and 2 times.

Consequently, the USED_BY_PARTITION constraint holds since, for each partition associated with the collection VARIABLES2 $= \langle 1,3,6,6\rangle$, its number of occurrences within VARIABLES1 $= \langle 1,9,1,6,2,3\rangle$ is greater than or equal to its number of occurrences within VARIABLES2:

- Partition $\texttt{p}-\langle 1,3\rangle$ occurs 3 times within $\langle 1,9,1,6,2,3\rangle$ and 2 times within $\langle 1,3,6,6\rangle$.

- Partition p $-$ $\langle 2, 6\rangle$ occurs 2 times within $\langle 1, 9, 1, 6, 2, 3\rangle$ and 2 times within $\langle 1, 3, 6, 6\rangle$.

Typical

$|$VARIABLES1$| > 1$
range(VARIABLES1.var) > 1
$|$VARIABLES2$| > 1$
range(VARIABLES2.var) > 1
$|$VARIABLES1$| > |$PARTITIONS$|$
$|$VARIABLES2$| > |$PARTITIONS$|$

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- Items of PARTITIONS are permutable.
- Items of PARTITIONS.p are permutable.
- An occurrence of a value of VARIABLES1.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.
- An occurrence of a value of VARIABLES2.var can be replaced by any other value that also belongs to the same partition of PARTITIONS.

Arg. properties

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union), PARTITIONS(id).

Used in K_USED_BY_PARTITION.

See also

implied by: SAME_PARTITION.

soft variant: SOFT_USED_BY_PARTITION_VAR *(variable-based violation measure)*.

specialisation: USED_BY *(*variable $\in$ partition *replaced by* variable*)*.

system of constraints: K_USED_BY_PARTITION.

used in graph description: IN_SAME_PARTITION.

Keywords

characteristic of a constraint: partition, sort based reformulation.

constraint arguments: constraint between two collections of variables.

modelling: inclusion.

Arc input(s)	VARIABLES1 VARIABLES2
Arc generator	$PRODUCT \mapsto$ collection(variables1, variables2)
Arc arity	2
Arc constraint(s)	IN_SAME_PARTITION(variables1.var, variables2.var, PARTITIONS)
Graph property(ies)	• for all connected components: **NSOURCE**$\geq$**NSINK** • **NSINK**$=$ |VARIABLES2|

Graph model

Parts (A) and (B) of Figure 5.836 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint on each connected component of the final graph we also show the different connected components. Each of them corresponds to an equivalence class according to the arc constraint. Note that the vertex corresponding to the variable that takes value 9 was removed from the final graph since there is no arc for which the associated equivalence constraint holds. The USED_BY_PARTITION constraint holds since:

- For each connected component of the final graph the number of sources is greater than or equal to the number of sinks.
- The number of sinks of the final graph is equal to |VARIABLES2|.

Signature

Since the initial graph contains only sources and sinks, and since sources of the initial graph cannot become sinks of the final graph, we have that the maximum number of sinks of the final graph is equal to |VARIABLES2|. Therefore we can rewrite **NSINK** $=$ |VARIABLES2| to **NSINK** $\geq$ |VARIABLES2| and simplify $\underline{\textbf{NSINK}}$ to $\overline{\textbf{NSINK}}$.

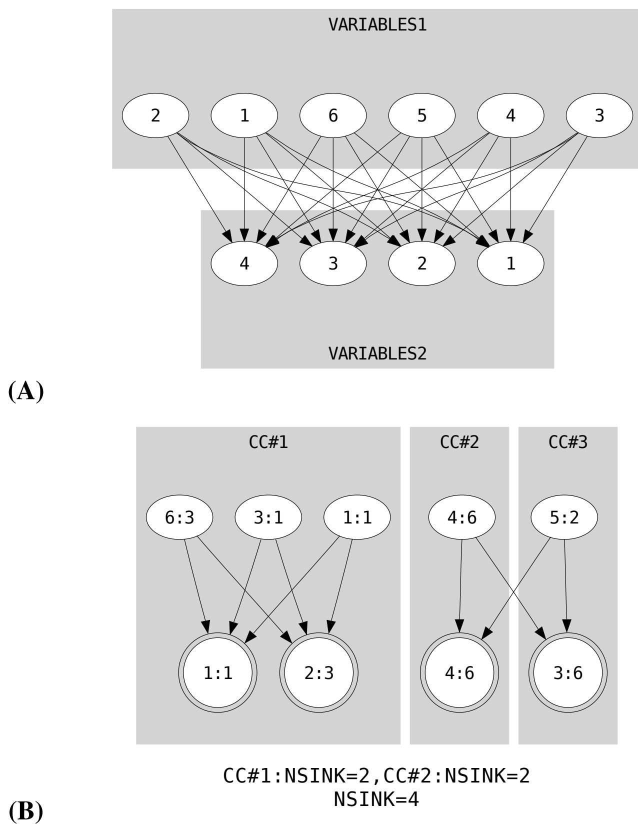


Figure 5.836: Initial and final graph of the USED_BY_PARTITION constraint

5.424 USES

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin [69]

Constraint USES(VARIABLES1, VARIABLES2)

Arguments

VARIABLES1 : collection(var−dvar)
VARIABLES2 : collection(var−dvar)

Restrictions

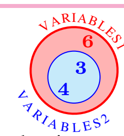
$\texttt{min}(1, |\texttt{VARIABLES1}|) \geq \texttt{min}(1, |\texttt{VARIABLES2}|)$
required(VARIABLES1, var)
required(VARIABLES2, var)

Purpose

The set of values assigned to the variables of the collection of variables VARIABLES2 is included within the set of values assigned to the variables of the collection of variables VARIABLES1.

Example

$(\langle 3, 3, 4, 6\rangle, \langle 3, 4, 4, 4, 4\rangle)$



The USES constraint holds since the set of values $\{3, 4\}$ assigned to the items of collection $\langle 3, 4, 4, 4, 4\rangle$ is included within the set of values $\{3, 4, 6\}$ occurring within $\langle 3, 3, 4, 6\rangle$.

All solutions

Figure 5.837 gives all solutions to the following non ground instance of the USES constraint: $\texttt{U}_1 \in [0, 1]$, $\texttt{U}_2 \in [1, 2]$, $\texttt{V}_1 \in [0, 2]$, $\texttt{V}_2 \in [2, 4]$, $\texttt{V}_3 \in [2, 4]$, $\text{USES}(\langle \texttt{U}_1, \texttt{U}_2\rangle, \langle \texttt{V}_1, \texttt{V}_2, \texttt{V}_3\rangle)$.

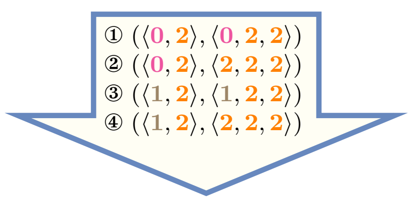


Figure 5.837: All solutions corresponding to the non ground example of the USES constraint of the **All solutions** slot where identical values are coloured in the same way in both collections

Typical

$|\texttt{VARIABLES1}| > 1$
$\texttt{range}(\texttt{VARIABLES1.var}) > 1$
$|\texttt{VARIABLES2}| > 1$
$\texttt{range}(\texttt{VARIABLES2.var}) > 1$
$|\texttt{VARIABLES1}| \leq |\texttt{VARIABLES2}|$

Symmetries

- Items of VARIABLES1 are permutable.
- Items of VARIABLES2 are permutable.
- All occurrences of two distinct values in VARIABLES1.var or VARIABLES2.var can be swapped; all occurrences of a value in VARIABLES1.var or VARIABLES2.var can be renamed to any unused value.

Arg. properties

- Contractible wrt. VARIABLES2.
- Extensible wrt. VARIABLES1.
- Aggregate: VARIABLES1(union), VARIABLES2(union).

Remark

It was shown in [69] that, finding out whether a USES constraint has a solution or not is NP-hard. This was achieved by reduction from 3-SAT.

See also

generalisation: COMMON.

implied by: USED_BY.

related: ROOTS.

Keywords

complexity: 3-SAT.

constraint arguments: constraint between two collections of variables.

final graph structure: acyclic, bipartite, no loop.

modelling: inclusion.

Arc input(s) VARIABLES1 VARIABLES2

Arc generator *PRODUCT*$\mapsto$collection(variables1, variables2)

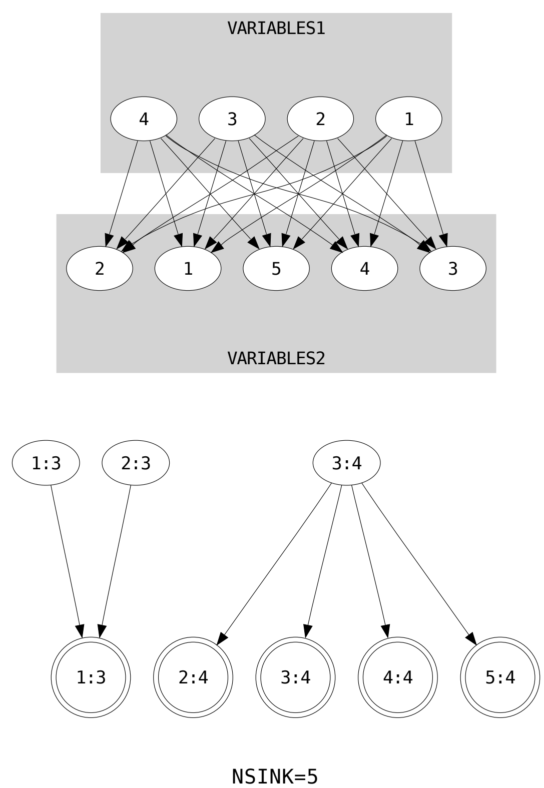
Arc arity 2

Arc constraint(s) variables1.var $=$ variables2.var

Graph property(ies) **NSINK**$=$ |VARIABLES2|

Graph class
- ACYCLIC
- BIPARTITE
- NO_LOOP

Graph model Parts (A) and (B) of Figure 5.838 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSINK** graph property, the sink vertices of the final graph are stressed with a double circle. Note that all the vertices corresponding to the variables that take values 9 or 2 were removed from the final graph since there is no arc for which the associated equality constraint holds.



(A)
(B)

Figure 5.838: Initial and final graph of the USES constraint

5.425 VALLEY

► ▷ ◁ ◄ DESCRIPTION LINKS AUTOMATON

Origin

Derived from INFLEXION.

Constraint

VALLEY(N, VARIABLES)

Arguments

N : dvar
VARIABLES : collection(var−dvar)

Restrictions

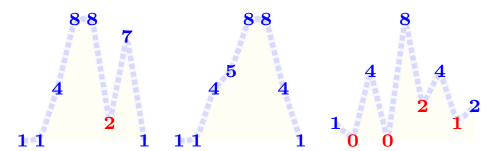
$\mathtt{N} \geq 0$
$2 * \mathtt{N} \leq \mathtt{max}(|\mathtt{VARIABLES}| - 1, 0)$
required(VARIABLES, var)

Purpose

A variable V_v $(1 < v < m)$ of the sequence of variables $\mathtt{VARIABLES} = V_1, \ldots, V_m$ is a *valley* if and only if there exists an i (with $1 < i \leq v$) such that $V_{i-1} > V_i$ and $V_i = V_{i+1} = \cdots = V_v$ and $V_v < V_{v+1}$. N is the total number of valleys of the sequence of variables VARIABLES.

Example

$(1, \langle 1, 1, 4, 8, 8, 2, 7, 1\rangle)$
$(0, \langle 1, 1, 4, 5, 8, 8, 4, 1\rangle)$
$(4, \langle 1, 0, 4, 0, 8, 2, 4, 1, 2\rangle)$



The first VALLEY constraint holds since the sequence 1 1 4 8 8 2 7 1 contains one valley that corresponds to the variable that is assigned to value 2.

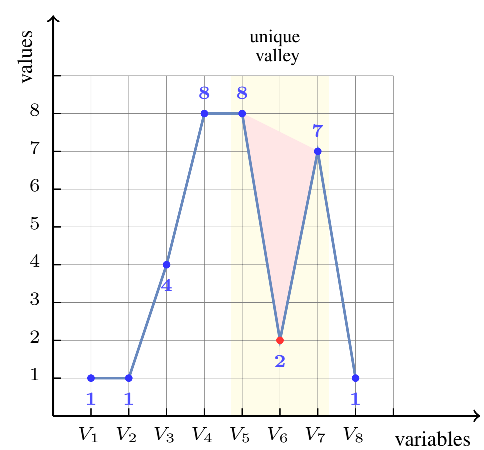


Figure 5.839: Illustration of the first example of the **Example** slot: a sequence of eight variables V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8 respectively fixed to values 1, 1, 4, 8, 8, 2, 7, 1 and its corresponding unique valley ($\mathtt{N} = 1$)

All solutions

Figure 5.840 gives all solutions to the following non ground instance of the VALLEY constraint: $\mathtt{N} \in [1,2]$, $\mathtt{V}_1 \in [0,1]$, $\mathtt{V}_2 \in [0,2]$, $\mathtt{V}_3 \in [0,2]$, $\mathtt{V}_4 \in [0,1]$, VALLEY$(\mathtt{N}, \langle \mathtt{V}_1, \mathtt{V}_2, \mathtt{V}_3, \mathtt{V}_4 \rangle)$.

① $(\mathbf{1}, \langle 0, 1, 0, 1 \rangle)$
② $(\mathbf{1}, \langle 0, 2, 0, 1 \rangle)$
③ $(\mathbf{1}, \langle 1, 0, 0, 1 \rangle)$
④ $(\mathbf{1}, \langle 1, 0, 1, 0 \rangle)$
⑤ $(\mathbf{1}, \langle 1, 0, 1, 1 \rangle)$
⑥ $(\mathbf{1}, \langle 1, 0, 2, 0 \rangle)$
⑦ $(\mathbf{1}, \langle 1, 0, 2, 1 \rangle)$
⑧ $(\mathbf{1}, \langle 1, 1, 0, 1 \rangle)$
⑨ $(\mathbf{1}, \langle 1, 2, 0, 1 \rangle)$

Figure 5.840: All solutions corresponding to the non ground example of the VALLEY constraint of the **All solutions** slot where each valley is coloured in orange

Typical

$|\mathtt{VARIABLES}| > 2$
$\mathtt{range}(\mathtt{VARIABLES.var}) > 1$

Typical model

$\mathtt{nval}(\mathtt{VARIABLES.var}) > 2$

Symmetries

- Items of VARIABLES can be reversed.
- One and the same constant can be added to the var attribute of all items of VARIABLES.

Arg. properties

- Functional dependency: N determined by VARIABLES.
- Contractible wrt. VARIABLES when $\mathtt{N} = 0$.

Usage

Useful for constraining the number of *valleys* of a sequence of domain variables.

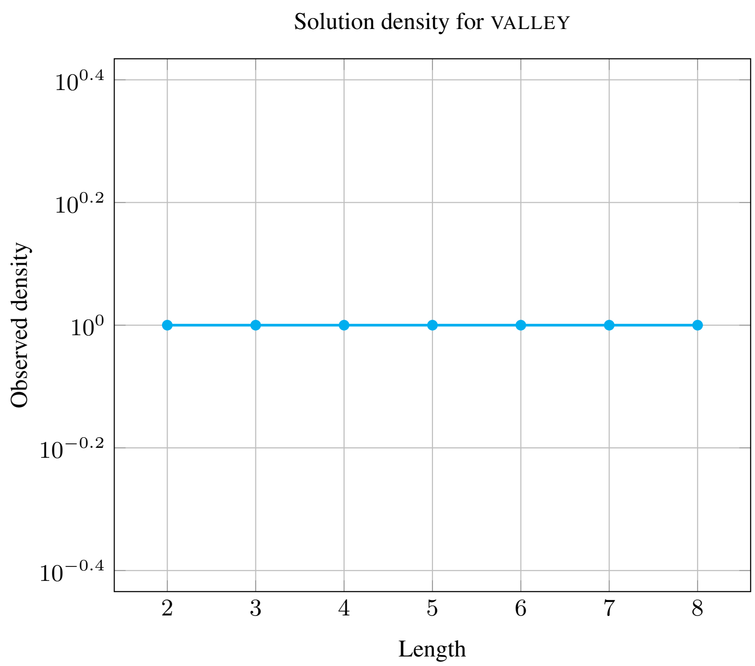
Remark

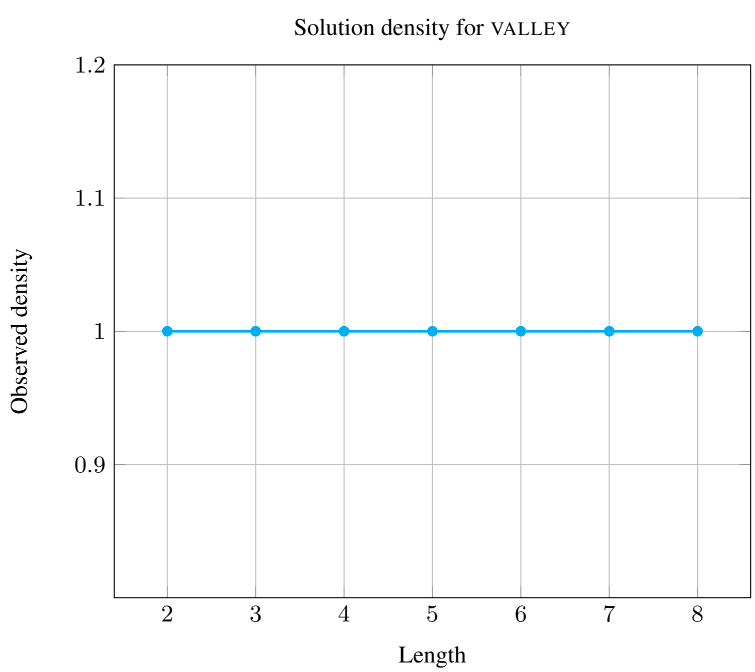
Since the arity of the arc constraint is not fixed, the VALLEY constraint cannot be currently described with the graph-based representation. However, this would not hold anymore if we were introducing a slot that specifies how to merge adjacent vertices of the final graph.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	9	64	625	7776	117649	2097152	43046721

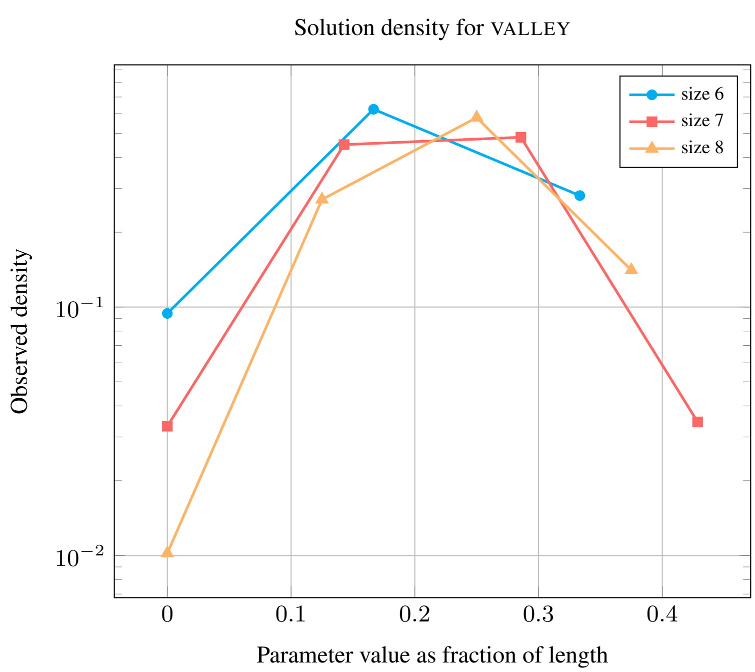
Number of solutions for VALLEY: domains $0..n$

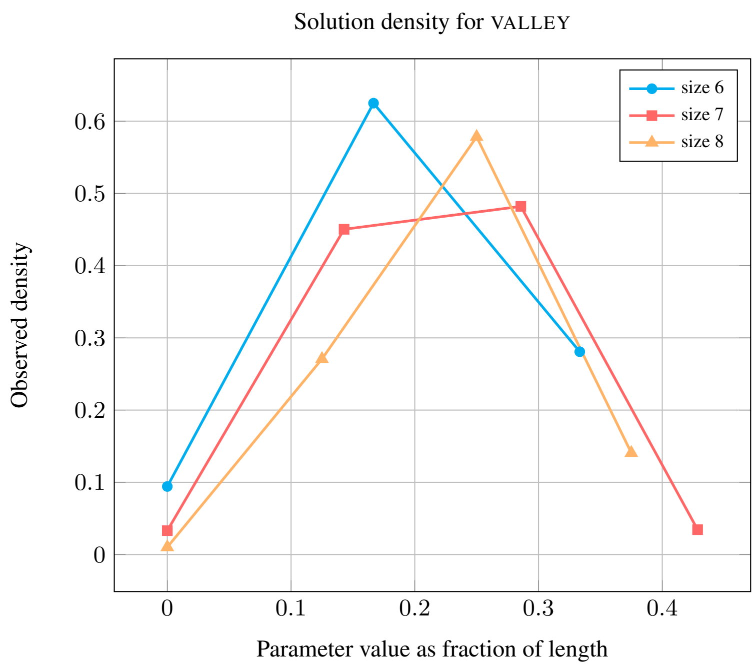

Solution density for VALLEY
Observed density
$10^{0.4}$
$10^{0.2}$
10^{0}
$10^{-0.2}$
$10^{-0.4}$
2 3 4 5 6 7 8
Length


Solution density for VALLEY
Observed density
1.2
1.1
1
0.9
2 3 4 5 6 7 8
Length

Length (n)		2	3	4	5	6	7	8
Total		9	64	625	7776	117649	2097152	43046721
Parameter value	0	9	50	295	1792	11088	69498	439791
	1	-	14	330	5313	73528	944430	11654622
	2	-	-	-	671	33033	1010922	24895038
	3	-	-	-	-	-	72302	6057270

Solution count for VALLEY: domains $0..n$





See also

common keyword: DEEPEST_VALLEY, INFLEXION, MIN_DIST_BETWEEN_INFLEXION, MIN_WIDTH_VALLEY *(sequence)*.

comparison swapped: PEAK.

generalisation: BIG_VALLEY *(a tolerance parameter is added for counting only big valleys)*.

related: ALL_EQUAL_VALLEY, ALL_EQUAL_VALLEY_MIN, DECREASING_VALLEY, INCREASING_VALLEY, NO_PEAK.

specialisation: NO_VALLEY *(the variable counting the number of valleys is set to* 0 *and removed)*.

Keywords

characteristic of a constraint: automaton, automaton with counters, automaton with same input symbol.

combinatorial object: sequence.

constraint arguments: reverse of a constraint, pure functional dependency.

constraint network structure: sliding cyclic(1) constraint network(2).

filtering: glue matrix.

modelling: functional dependency.

Cond. implications

- VALLEY($\texttt{N}, \texttt{VARIABLES}$)
  with $\texttt{N} > 0$
  implies ATLEAST_NVALUE($\texttt{NVAL}, \texttt{VARIABLES}$)
  when $\texttt{NVAL} = 2$.

- VALLEY(N, VARIABLES)
  implies INFLEXION(N, VARIABLES)
  when N = PEAK(VARIABLES.var) + VALLEY(VARIABLES.var).

Automaton Figure 5.841 depicts the automaton associated with the VALLEY constraint. To each pair of consecutive variables $(\texttt{VAR}_i, \texttt{VAR}_{i+1})$ of the collection VARIABLES corresponds a signature variable S_i. The following signature constraint links $\texttt{VAR}_i$, $\texttt{VAR}_{i+1}$ and S_i: $(\texttt{VAR}_i < \texttt{VAR}_{i+1} \Leftrightarrow S_i = 0) \wedge (\texttt{VAR}_i = \texttt{VAR}_{i+1} \Leftrightarrow S_i = 1) \wedge (\texttt{VAR}_i > \texttt{VAR}_{i+1} \Leftrightarrow S_i = 2)$.

STATE SEMANTICS

s	: stationary/increasing mode	$(\{<\mid=\}^*)$
u	: decreasing mode	$(>\{>\mid=\}^*)$

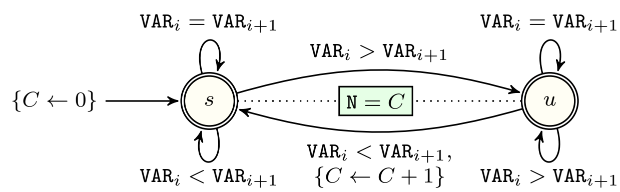


Figure 5.841: Automaton of the VALLEY constraint

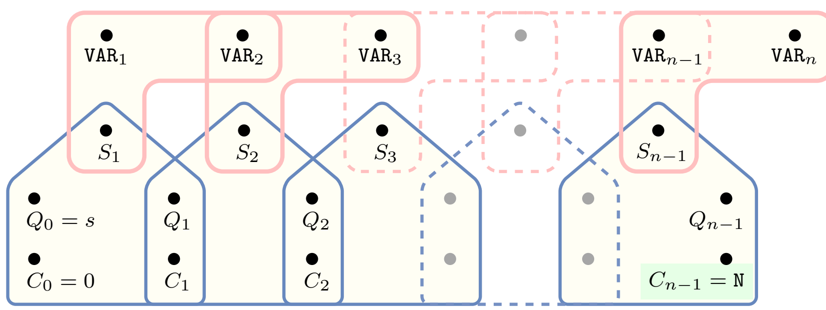


Figure 5.842: Hypergraph of the reformulation corresponding to the automaton of the VALLEY constraint (since all states of the automaton are accepting there is no restriction on the last variable Q_{n-1})

Glue matrix where $\overrightarrow{C}$ and $\overleftarrow{C}$ resp. represent the counter value C at the end of a prefix and at the end of the corresponding reverse suffix that partitions the sequence VARIABLES.

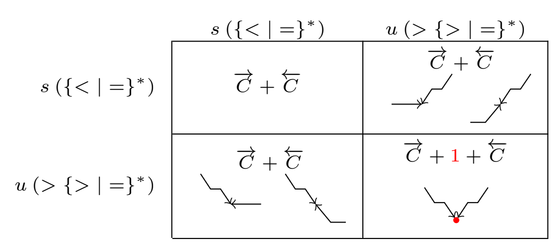


Figure 5.843: Glue matrix of the VALLEY constraint

VALLEY($\mathtt{N} = 1, \langle 1, 1, 4, 8, 8, 2, 7, 1 \rangle$)

	1	1	4	8	8	2		2	7	1	
		$=$	$<$	$<$	$=$	$>$			$>$	$<$	
i	0	1	2	3	4	5		2	1	0	i
$\overrightarrow{Q_i}$	s	s	s	s	s	u		u	s	s	$\overleftarrow{Q_i}$
$\overrightarrow{C_i}$	0	0	0	0	0	0		0	0	0	$\overleftarrow{C_i}$
$\overrightarrow{\mathtt{N}_i}$	0	0	0	0	0	0		0	0	0	$\overleftarrow{\mathtt{N}_i}$

VALLEY$\left(\begin{array}{c}\overrightarrow{\mathtt{N}_5} = 0,\\ \langle 1, 1, 4, 8, 8, 2 \rangle\end{array}\right)$ VALLEY$\left(\begin{array}{c}\overleftarrow{\mathtt{N}_2} = 0,\\ \langle 1, 7, 2 \rangle\end{array}\right)$

glue matrix entry associated with the state pair (u, u):

$\mathtt{N} = \overrightarrow{\mathtt{C}_5} + 1 + \overleftarrow{\mathtt{C}_2} = 0 + 1 + 0 = 1$

Figure 5.844: Illustrating the use of the state pair (u, u) of the glue matrix for linking $\mathtt{N}$ with the counters variables obtained after reading the prefix $1, 1, 4, 8, 8, 2$ and corresponding suffix $2, 7, 1$ of the sequence $1, 1, 4, 8, 8, 2, 7, 1$; note that the suffix $2, 7, 1$ (in pink) is proceed in reverse order; the left (resp. right) table shows the initialisation (for $i = 0$) and the evolution (for $i > 0$) of the state of the automaton and its counter C upon reading the prefix $1, 1, 4, 8, 8, 2$ (resp. the reverse suffix $1, 7, 2$).

5.426 VEC_EQ_TUPLE

► ▷ ◁ ◄ DESCRIPTION LINKS GRAPH

Origin Used for defining IN_RELATION.

Constraint VEC_EQ_TUPLE(VARIABLES, TUPLE)

Arguments

VARIABLES	:	collection(var−dvar)
TUPLE	:	collection(val−int)

Restrictions

required(VARIABLES, var)
required(TUPLE, val)
$|\text{VARIABLES}| = |\text{TUPLE}|$

Purpose Enforce a vector of domain variables to be equal to a tuple of values.

Example $(\langle 5, 3, 3\rangle, \langle 5, 3, 3\rangle)$

The VEC_EQ_TUPLE constraint holds since the first, the second and the third items of VARIABLES $= \langle 5, 3, 3\rangle$ are respectively equal to the first, the second and the third items of TUPLE $= \langle 5, 3, 3\rangle$.

Typical

$|\text{VARIABLES}| > 1$
range(VARIABLES.var) > 1
range(TUPLE.val) > 1

Symmetries

- Arguments are permutable w.r.t. permutation (VARIABLES, TUPLE).
- Items of VARIABLES and TUPLE are permutable (*same permutation used*).

Arg. properties Contractible wrt. VARIABLES and TUPLE (*remove items from same position*).

Used in IN_RELATION.

See also **generalisation:** LEX_EQUAL *(*integer *replaced by* variable *in second argument)*.

implies: LEX_EQUAL.

Keywords **characteristic of a constraint:** tuple.

constraint type: value constraint.

filtering: arc-consistency.

Arc input(s)	VARIABLES TUPLE
Arc generator	$PRODUCT(=) \mapsto$ collection(variables, tuple)
Arc arity	2
Arc constraint(s)	variables.var = tuple.val
Graph property(ies)	**NARC**= |VARIABLES|

Graph model Parts (A) and (B) of Figure 5.845 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NARC** graph property, the arcs of the final graph are stressed in bold.

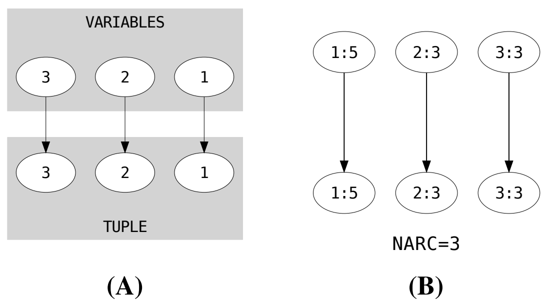


Figure 5.845: Initial and final graph of the VEC_EQ_TUPLE constraint

Signature Since we use the arc generator $PRODUCT(=)$ on the collections VARIABLES and TUPLE, and because of the restriction |VARIABLES| = |TUPLE|, the maximum number of arcs of the final graph is equal to |VARIABLES|. Therefore we can rewrite the graph property **NARC** = |VARIABLES| to **NARC** ≥ |VARIABLES| and simplify $\underline{\mathbf{NARC}}$ to $\overline{\mathbf{NARC}}$.

5.427 VISIBLE

▶ ▷ ◁ ◀

DESCRIPTION LINKS

Origin Extension of *accessibility* parameter of DIFFN.

Constraint VISIBLE(K, DIMS, FROM, OBJECTS, SBOXES)

Types

$$\begin{array}{lll} \text{VARIABLES} & : & \text{collection}(\text{v}-\text{dvar}) \\ \text{INTEGERS} & : & \text{collection}(\text{v}-\text{int}) \\ \text{POSITIVES} & : & \text{collection}(\text{v}-\text{int}) \\ \text{DIMDIR} & : & \text{collection}(\text{dim}-\text{int}, \text{dir}-\text{int}) \end{array}$$

Arguments

$$\begin{array}{lll} \text{K} & : & \text{int} \\ \text{DIMS} & : & \text{sint} \\ \text{FROM} & : & \text{DIMDIR} \\ \text{OBJECTS} & : & \text{collection}\begin{pmatrix} \text{oid}-\text{int}, \\ \text{sid}-\text{dvar}, \\ \text{x}-\text{VARIABLES}, \\ \text{start}-\text{dvar}, \\ \text{duration}-\text{dvar}, \\ \text{end}-\text{dvar} \end{pmatrix} \\ \text{SBOXES} & : & \text{collection}\begin{pmatrix} \text{sid}-\text{int}, \\ \text{t}-\text{INTEGERS}, \\ \text{l}-\text{POSITIVES}, \\ \text{f}-\text{DIMDIR} \end{pmatrix} \end{array}$$

Restrictions

$|\texttt{VARIABLES}| \geq 1$
$|\texttt{INTEGERS}| \geq 1$
$|\texttt{POSITIVES}| \geq 1$
$\texttt{required}(\texttt{VARIABLES}, \texttt{v})$
$|\texttt{VARIABLES}| = \texttt{K}$
$\texttt{required}(\texttt{INTEGERS}, \texttt{v})$
$|\texttt{INTEGERS}| = \texttt{K}$
$\texttt{required}(\texttt{POSITIVES}, \texttt{v})$
$|\texttt{POSITIVES}| = \texttt{K}$
$\texttt{POSITIVES.v} > 0$
$\texttt{required}(\texttt{DIMDIR}, [\texttt{dim}, \texttt{dir}])$
$|\texttt{DIMDIR}| > 0$
$|\texttt{DIMDIR}| \leq \texttt{K} + \texttt{K}$
$\texttt{distinct}(\texttt{DIMDIR}, [])$
$\texttt{DIMDIR.dim} \geq 0$
$\texttt{DIMDIR.dim} < \texttt{K}$
$\texttt{DIMDIR.dir} \geq 0$
$\texttt{DIMDIR.dir} \leq 1$
$\texttt{K} \geq 0$
$\texttt{DIMS} \geq 0$
$\texttt{DIMS} < \texttt{K}$
$\texttt{distinct}(\texttt{OBJECTS}, \texttt{oid})$
$\texttt{required}(\texttt{OBJECTS}, [\texttt{oid}, \texttt{sid}, \texttt{x}])$
$\texttt{require\_at\_least}(2, \texttt{OBJECTS}, [\texttt{start}, \texttt{duration}, \texttt{end}])$
$\texttt{OBJECTS.oid} \geq 1$
$\texttt{OBJECTS.oid} \leq |\texttt{OBJECTS}|$
$\texttt{OBJECTS.sid} \geq 1$
$\texttt{OBJECTS.sid} \leq |\texttt{SBOXES}|$
$\texttt{OBJECTS.duration} \geq 0$
$|\texttt{SBOXES}| \geq 1$
$\texttt{required}(\texttt{SBOXES}, [\texttt{sid}, \texttt{t}, \texttt{l}])$
$\texttt{SBOXES.sid} \geq 1$
$\texttt{SBOXES.sid} \leq |\texttt{SBOXES}|$
$\texttt{do\_not\_overlap}(\texttt{SBOXES})$

Purpose

Holds if and only if:

1. The difference between the end in time and the start in time of each object is equal to its duration in time.
2. Given a collection of potential observations places FROM, where each observation place is specified by a *dimension* (i.e., an integer between 0 and $k-1$) and by a *direction* (i.e., an integer between 0 and 1), and given for each shifted box of SBOXES a set of visible faces, enforce that *at least one visible face of each shifted box associated with an object* $o \in$ OBJECTS *should be entirely visible from at least one observation place of* FROM *at time* o.start *as well as at time* o.end -1. This notion is defined in a more formal way in the **Remark** slot.

Example

$$\left(\begin{array}{l} 2,\{0,1\}, \\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{s}-8 & \texttt{d}-8 & \texttt{e}-16, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,2\rangle & \texttt{s}-1 & \texttt{d}-15 & \texttt{e}-16 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

$$\left(\begin{array}{l} 2,\{0,1\}, \\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{s}-1 & \texttt{d}-8 & \texttt{e}-9, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,2\rangle & \texttt{s}-1 & \texttt{d}-15 & \texttt{e}-16 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

$$\left(\begin{array}{l} 2,\{0,1\}, \\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle & \texttt{s}-1 & \texttt{d}-15 & \texttt{e}-16, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 2,2\rangle & \texttt{s}-6 & \texttt{d}-6 & \texttt{e}-12 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

$$\left(\begin{array}{l} 2,\{0,1\}, \\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 4,1\rangle & \texttt{s}-1 & \texttt{d}-8 & \texttt{e}-9, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 1,2\rangle & \texttt{s}-1 & \texttt{d}-15 & \texttt{e}-16 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

$$\left(\begin{array}{l} 2,\{0\}, \\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-1 & \texttt{sid}-1 & \texttt{x}-\langle 2,1\rangle & \texttt{s}-1 & \texttt{d}-8 & \texttt{e}-9, \\ \texttt{oid}-2 & \texttt{sid}-2 & \texttt{x}-\langle 4,3\rangle & \texttt{s}-1 & \texttt{d}-15 & \texttt{e}-16 \end{array}\right\rangle, \\ \left\langle \begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

(`s` for `start`, `d` for `duration`, `e` for `end`)

The five previous examples correspond respectively to parts (I), (II) of Figure 5.847, to parts (III) and (IV) of Figure 5.848, and to Figure 5.849. Before introducing these five examples Figure 5.846 first illustrates the notion of *observations places* and of *visible faces*.

We first need to introduce a number of definitions in order to illustrate the notion of *visibility*.

Definition 1. *Consider two distinct objects o and o' of the* VISIBLE *constraint (i.e., $o, o' \in$ iobjects) as well as an observation place defined by the pair $\langle \texttt{dim}, \texttt{dir}\rangle \in$ `FROM`. The object o* is masked by *the object o' according to the observation place $\langle \texttt{dim}, \texttt{dir}\rangle$ if there exist two shifted boxes s and s' respectively associated with o and o' such that conditions* **A**, **B**, **C**, **D** *and* **E** *all hold:*

- *(**A**) $o.\texttt{duration} > 0 \wedge o'.\texttt{duration} > 0 \wedge o.\texttt{end} > o'.\texttt{start} \wedge o'.\texttt{end} > o.\texttt{start}$ (i.e., the time intervals associated with o and o' intersect).*
- *(**B**) Discarding dimension `dim`, s and s' intersect in all dimensions specified by `DIMS` (i.e., objects o and o' are in vis-à-vis).*

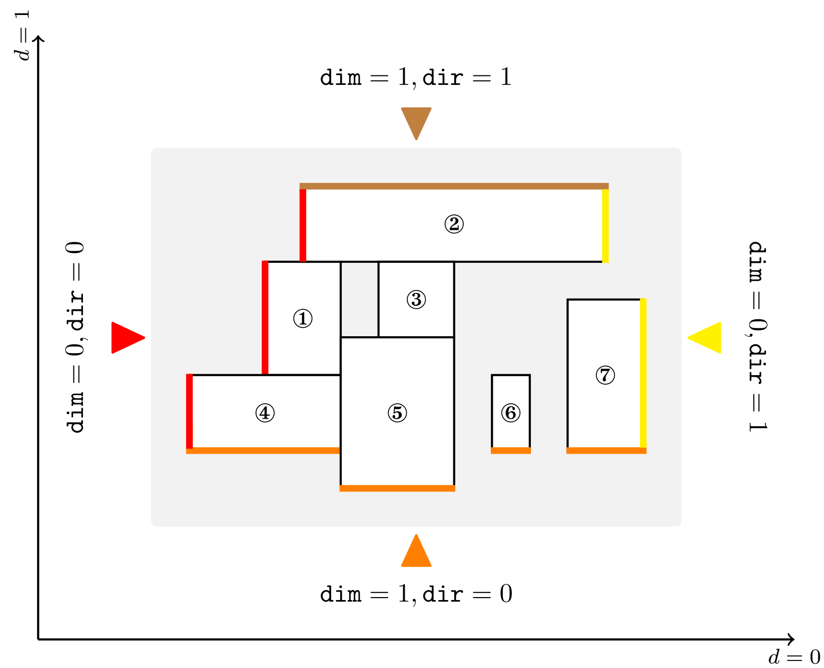


Figure 5.846: Entirely visible faces (depicted by a thick line) of rectangles ①, ②, ③, ④, ⑤, ⑥ and ⑦ from the four observation places $\langle \mathtt{dim}=0, \mathtt{dir}=1\rangle$, $\langle \mathtt{dim}=0, \mathtt{dir}=0\rangle$, $\langle \mathtt{dim}=1, \mathtt{dir}=1\rangle$ and $\langle \mathtt{dim}=1, \mathtt{dir}=0\rangle$ (depicted by a small triangle)

- *(**C**) If* $\mathtt{dir}=0$
  then $o.x[\mathtt{dim}] + s.t[\mathtt{dim}] \geq o'.x[\mathtt{dim}] + s'.t[\mathtt{dim}] + s'.l[\mathtt{dim}]$
  else $o'.x[\mathtt{dim}] + s'.t[\mathtt{dim}] \geq o.x[\mathtt{dim}] + s.t[\mathtt{dim}] + s.l[\mathtt{dim}]$ *(i.e., in dimension* dim*, o and o' are ordered in the wrong way according to direction* dir*).*
- *(**D**)* $o.\mathtt{start} > o'.\mathtt{start} \vee o.\mathtt{end} < o'.\mathtt{end}$ *(i.e., instants* $o.\mathtt{start}$ *or* $o.\mathtt{end}$ *are located within interval* $[o'.\mathtt{start}, o'.\mathtt{end}]$*; we consider also condition **A**.).*
- *(**E**) The observation place* $\langle \mathtt{dim}, \mathtt{dir}\rangle$ *occurs within the list of visible faces associated with the face attribute* f *of the shifted box s (i.e., the pair* $\langle \mathtt{dim}, \mathtt{dir}\rangle$ *is a potentially visible face of o).*

Definition 2. *Consider an object o of the collection* OBJECTS *as well as a possible observation place defined by the pair* $\langle \mathtt{dim}, \mathtt{dir}\rangle$*. The object o* is masked according to the observation place $\langle \mathtt{dim}, \mathtt{dir}\rangle$ *if and only if at least one of the following conditions holds:*

- *No shifted box associated with o has the pair* $\langle \mathtt{dim}, \mathtt{dir}\rangle$ *as one of its potentially visible face.*
- *The object o is masked according to the possible observation place* $\langle \mathtt{dim}, \mathtt{dir}\rangle$ *by another object o'.*

Figures 5.847, 5.848, and 5.849 respectively illustrate Definition 1 in the context of an observation place (depicted by a triangle) that is equal to the pair $\langle \mathtt{dim}=0, \mathtt{dir}=1\rangle$. Note that, in the context of Figure 5.849, as the DIMS parameter of the VISIBLE constraint only mentions dimension 0 (and not dimension 1), one object may be masked by another object even though the two objects do not intersect in any dimension: i.e., only their respective ordering in the dimension $\mathtt{dim} = 0$ as well as their positions in time matter.

Definition 3. *Consider an object o of the collection* OBJECTS *as well as a possible observation place defined by the pair $\langle \mathtt{dim}, \mathtt{dir} \rangle$. The object o* is masked according to the observation place $\langle \mathtt{dim}, \mathtt{dir} \rangle$ *if and only if at least one of the following conditions holds:*

- *No shifted box associated with o has the pair $\langle \mathtt{dim}, \mathtt{dir} \rangle$ as one of its potentially visible face.*
- *The object o is masked according to the possible observation place $\langle \mathtt{dim}, \mathtt{dir} \rangle$ by another object o'.*

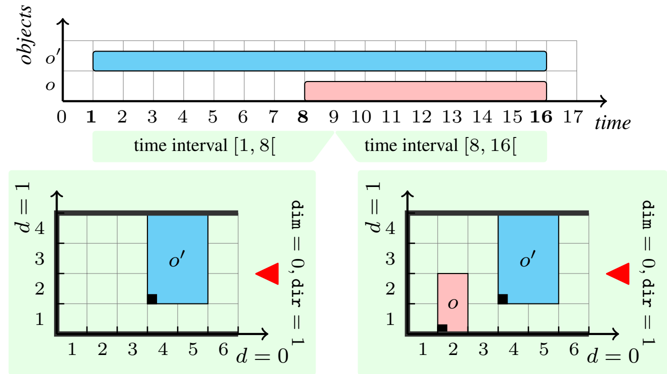
Definition 4. *An object of the collection* OBJECTS *constraint is* masked according to a set of possible observation places FROM *if it is masked according to each observation place of* FROM.

We are now in position to define the VISIBLE constraint.

Definition 5. *Given a* VISIBLE(K, DIMS, FROM, OBJECTS, SBOXES) *constraint, the* VISIBLE *constraint holds if none of the objects of* OBJECTS *is masked according to the dimensions of* DIMS *and to the set of possible observation places defined by* FROM.

$$\texttt{VISIBLE}\left(\begin{array}{c} 2,\ \{0,1\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle\begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-8 & \texttt{duration}-8 & \texttt{end}-16, \\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 4,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16 \end{array}\right\rangle, \\ \left\langle\begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

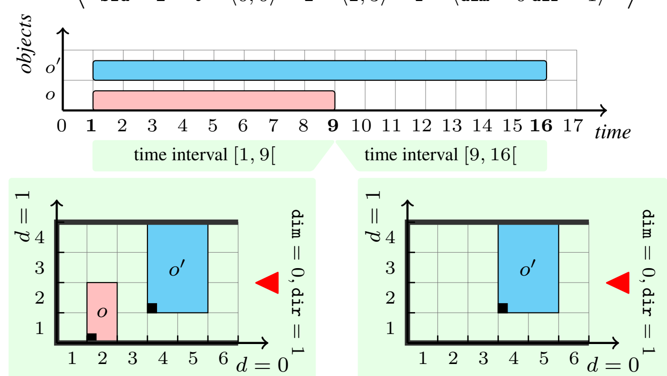
(I)



o is masked by o' according to $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ since:
(A) o and o' intersect in time,
(B) o and o' intersect in dimension 1,
(C) in dimension 0, o' starts after the end of o,
(D) the start in time of o is located after the start in time of o',
(E) $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of o.

$$\texttt{VISIBLE}\left(\begin{array}{c} 2,\ \{0,1\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle\begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9, \\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 4,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16 \end{array}\right\rangle, \\ \left\langle\begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

(II)

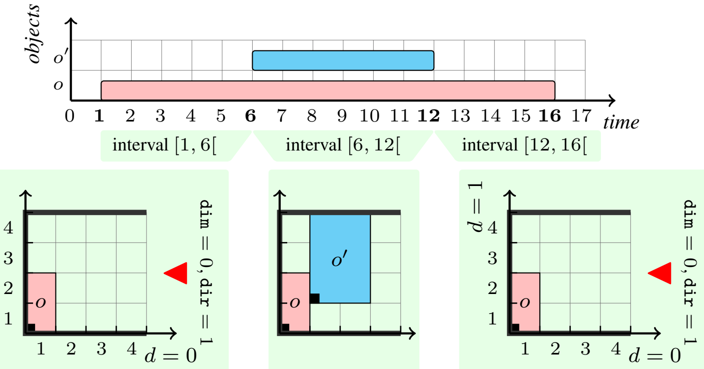


o is masked by o' according to $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ since:
(A) o and o' intersect in time,
(B) o and o' intersect in dimension 1,
(C) in dimension 0, o' starts after the end of o,
(D) the end in time of o is located before the end in time of o',
(E) $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of o.

Figure 5.847: Illustration of Definition 1: two examples (I) and (II) where an object o is masked by an object o' according to dimensions $\{0,1\}$ and to the observation place $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ because **(A)** o and o' intersect in time, **(B)** o and o' intersect in dimension 1, **(C)** o and o' are not well ordered according to the observation place, **(D)** there exists an instant where o' if present (but not o) and **(E)** $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of o.

$$\text{VISIBLE}\left(\begin{array}{c} 2,\ \{0,1\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\ \left\langle\begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 1,1\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16,\\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 2,2\rangle & \texttt{start}-6 & \texttt{duration}-6 & \texttt{end}-12\end{array}\right\rangle,\\ \left\langle\begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle\end{array}\right\rangle\end{array}\right)$$

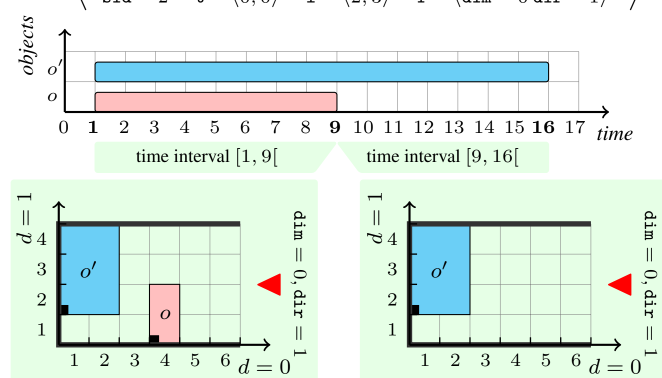
(III)



o is not masked by o' according to $\langle\texttt{dim}=0,\texttt{dir}=1\rangle$ since:
(A) Even though o and o' intersect in time,
(B) and even though o and o' intersect in dimension 1,
(C) and even though, in dimension 0, o' starts after the end of o,
(E) and even though $\langle\texttt{dim}=0,\texttt{dir}=1\rangle$ is a potentially visible face of o, condition **(D)** does not hold.

$$\text{VISIBLE}\left(\begin{array}{c} 2,\ \{0,1\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\ \left\langle\begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 4,1\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9,\\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 1,2\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16\end{array}\right\rangle,\\ \left\langle\begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle,\\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,3\rangle & \texttt{f}-\langle \texttt{dim}-0\ \texttt{dir}-1\rangle\end{array}\right\rangle\end{array}\right)$$

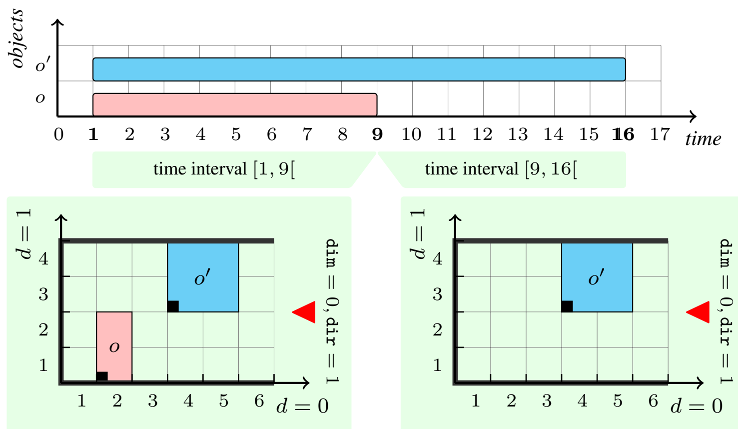
(IV)



o is not masked by o' according to $\langle\texttt{dim}=0,\texttt{dir}=1\rangle$ since:
(A) Even though o and o' intersect in time,
(B) and even though o and o' intersect in dimension 1, and even
(D) though the end in time of o is located before the end in time of o',
(E) and even though $\langle\texttt{dim}=0,\texttt{dir}=1\rangle$ is a potentially visible face of o, condition **(C)** does not hold.

Figure 5.848: Illustration of Definition 1: two examples (III) and (IV) where an object o is not masked by an object o' according to the observation place $\langle\texttt{dim}=0,\texttt{dir}=1\rangle$.

$$\text{VISIBLE}\left(\begin{array}{c} 2,\ \{0\},\ \langle \texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \left\langle\begin{array}{llllll} \texttt{oid}-o & \texttt{sid}-1 & \texttt{x}-\langle 2,1\rangle & \texttt{start}-1 & \texttt{duration}-8 & \texttt{end}-9, \\ \texttt{oid}-o' & \texttt{sid}-2 & \texttt{x}-\langle 4,3\rangle & \texttt{start}-1 & \texttt{duration}-15 & \texttt{end}-16 \end{array}\right\rangle, \\ \left\langle\begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 1,2\rangle & \texttt{f}-\langle\texttt{dim}-0\ \texttt{dir}-1\rangle, \\ \texttt{sid}-2 & \texttt{t}-\langle 0,0\rangle & \texttt{l}-\langle 2,2\rangle & \texttt{f}-\langle\texttt{dim}-0\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$



o is masked by o' according to $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ since:

A. o and o' intersect in time,

B. in dimension 0, o' starts after the end of o,

C. the end in time of o is located before the end in time of o',

D. $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of o.

Figure 5.849: Illustration of Definition 1: the case where an object o is masked by an object o' according to dimension 0 and to the observation place $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ because: **(A)** o and o' intersect in time, **(C)** o and o' are not well ordered according to the observation place and **(D)** there exists an instant where o' if present (but not o) and **(E)** $\langle \texttt{dim}=0, \texttt{dir}=1\rangle$ is a potentially visible face of o.

Typical $|\texttt{OBJECTS}| > 1$

Symmetries

- Items of `OBJECTS` are permutable.
- Items of `SBOXES` are permutable.

Usage We now give several typical concrete uses of the VISIBLE constraint, which all mention the DIFFST as well as the VISIBLE constraints:

- Figure 5.850 corresponds to a *ship loading problem* where containers are piled within a ship by a crane each time the ship visits a given harbour. In this context we have first to express the fact that *a container can only be placed on top of an already placed container* and second, that *a container can only be taken away if no container is placed on top of it*. These two conditions are expressed by a single VISIBLE constraint for which the `DIMS` parameter mentions all three dimensions of the placement space and the `FROM` parameter mentions the pair $\langle \texttt{dim} = 2, \texttt{dir} = 1\rangle$ as its unique observation place. In addition we also use a DIFFST constraint for expressing non-overlapping.

$$\text{VISIBLE}\left(\begin{array}{c} 3,\ \{0,1,2\},\ \langle \texttt{dim}-2\ \texttt{dir}-1\rangle, \\ \left\langle \begin{array}{llllll} \texttt{oid}-\mathbf{1} & \texttt{sid}-1 & \texttt{x}-\langle 1,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17, \\ \texttt{oid}-\mathbf{2} & \texttt{sid}-1 & \texttt{x}-\langle 1,1,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8, \\ \texttt{oid}-\mathbf{3} & \texttt{sid}-1 & \texttt{x}-\langle 4,1,1\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8, \\ \texttt{oid}-\mathbf{4} & \texttt{sid}-1 & \texttt{x}-\langle 1,1,3\rangle & \texttt{start}-8 & \texttt{duration}-9 & \texttt{end}-17, \\ \texttt{oid}-\mathbf{5} & \texttt{sid}-1 & \texttt{x}-\langle 4,1,1\rangle & \texttt{start}-8 & \texttt{duration}-16 & \texttt{end}-24, \\ \texttt{oid}-\mathbf{6} & \texttt{sid}-1 & \texttt{x}-\langle 1,1,1\rangle & \texttt{start}-17 & \texttt{duration}-7 & \texttt{end}-24 \end{array}\right\rangle, \\ \langle\ \texttt{sid}-1\quad \texttt{t}-\langle 0,0,0\rangle\quad \texttt{l}-\langle 2,4,2\rangle\quad \texttt{f}-\langle \texttt{dim}-2\ \texttt{dir}-1\rangle\ \rangle \end{array}\right)$$

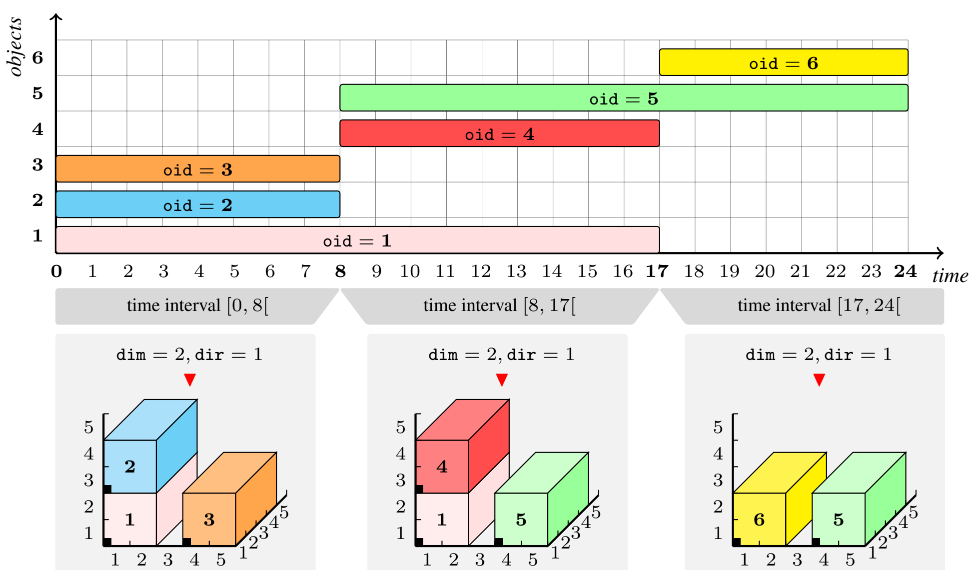


Figure 5.850: Illustration of the ship loading problem

- Figure 5.851 corresponds to a *container loading/unloading problem* in the context of a pick-up delivery problem where the loading/unloading takes place with respect to the front door of the container. Beside the DIFFST constraint used for expressing non-overlapping, we use two distinct VISIBLE constraints:

  - The first VISIBLE constraint takes care of the location of the front door of the container (each object o has to be loaded/unloaded without moving around any other object, i.e., objects that are in the vis-à-vis of o according to the front door of the container). This is expressed by a single VISIBLE constraint for which the DIMS parameter mentions all three dimensions of the placement space and the FROM parameter mentions the pair $\langle \texttt{dim} = 1, \texttt{dir} = 0 \rangle$ as its unique observation place.
  - The second VISIBLE constraint takes care of the *gravity dimension* (i.e., each object that has to be loaded should not be put under another object, and reciprocally each object that has to be unloaded should not be located under another object). This is expressed by the same VISIBLE constraint that was used for the ship loading problem, i.e., a VISIBLE constraint for which the DIMS parameter mentions all three dimensions of the placement space and the FROM parameter mentions the pair $\langle \texttt{dim} = 2, \texttt{dir} = 1 \rangle$ as its unique observation place.

- Figure 5.852 corresponds to a *pallet loading problem* where one has to place six objects on a pallet. Each object corresponds to a parallelepiped that has a bar code on one of its four sides (i.e., the sides that are different from the top and the bottom of the parallelepiped). If, for some reason, an object has no bar code then we simply remove it from the objects that will be passed to the VISIBLE constraint: this is the case, for example, for the sixth object. In this context the constraint to enforce (beside the non-overlapping constraint between the parallelepipeds that are assigned to a same pallet) is the fact that the bar code of each object should be visible (i.e., visible from one of the four sides of the pallet). This is expressed by the VISIBLE constraint given in Part (F) of Figure 5.852.

Remark The VISIBLE constraint is a generalisation of the ACCESSIBILITY constraint initially introduced in the context of the DIFFN constraint.

See also **common keyword:** DIFFN *(geometrical constraint)*, GEOST, GEOST_TIME *(geometrical constraint,sweep)*, NON_OVERLAP_SBOXES *(geometrical constraint)*.

Keywords **constraint type:** decomposition, predefined constraint.

filtering: sweep.

geometry: geometrical constraint.

$$\text{VISIBLE}\left(\begin{array}{c} 3,\ \{0,1,2\},\ \langle \texttt{dim}-1\ \texttt{dir}-0\rangle,\\ \left\langle\begin{array}{llllll} \texttt{oid}-\mathbf{1} & \texttt{sid}-1 & \texttt{x}-\langle 1,2,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8,\\ \texttt{oid}-\mathbf{2} & \texttt{sid}-2 & \texttt{x}-\langle 1,3,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8,\\ \texttt{oid}-\mathbf{3} & \texttt{sid}-3 & \texttt{x}-\langle 1,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17,\\ \texttt{oid}-\mathbf{4} & \texttt{sid}-4 & \texttt{x}-\langle 4,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17,\\ \texttt{oid}-\mathbf{5} & \texttt{sid}-5 & \texttt{x}-\langle 1,2,3\rangle & \texttt{start}-8 & \texttt{duration}-9 & \texttt{end}-17,\\ \texttt{oid}-\mathbf{6} & \texttt{sid}-6 & \texttt{x}-\langle 3,1,1\rangle & \texttt{start}-8 & \texttt{duration}-12 & \texttt{end}-24,\\ \texttt{oid}-\mathbf{7} & \texttt{sid}-3 & \texttt{x}-\langle 1,1,1\rangle & \texttt{start}-17 & \texttt{duration}-7 & \texttt{end}-24 \end{array}\right\rangle,\\ \left\langle\begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,1,1\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-2 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,2,2\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-3 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,4,2\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-4 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,4,1\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-5 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,3,1\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-6 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 1,2,2\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

$$\text{VISIBLE}\left(\begin{array}{c} 3,\ \{0,1,2\},\ \langle \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \left\langle\begin{array}{llllll} \texttt{oid}-\mathbf{1} & \texttt{sid}-1 & \texttt{x}-\langle 1,2,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8,\\ \texttt{oid}-\mathbf{2} & \texttt{sid}-2 & \texttt{x}-\langle 1,3,3\rangle & \texttt{start}-0 & \texttt{duration}-8 & \texttt{end}-8,\\ \texttt{oid}-\mathbf{3} & \texttt{sid}-3 & \texttt{x}-\langle 1,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17,\\ \texttt{oid}-\mathbf{4} & \texttt{sid}-4 & \texttt{x}-\langle 4,1,1\rangle & \texttt{start}-0 & \texttt{duration}-17 & \texttt{end}-17,\\ \texttt{oid}-\mathbf{5} & \texttt{sid}-5 & \texttt{x}-\langle 1,2,3\rangle & \texttt{start}-8 & \texttt{duration}-9 & \texttt{end}-17,\\ \texttt{oid}-\mathbf{6} & \texttt{sid}-6 & \texttt{x}-\langle 3,1,1\rangle & \texttt{start}-8 & \texttt{duration}-12 & \texttt{end}-24,\\ \texttt{oid}-\mathbf{7} & \texttt{sid}-3 & \texttt{x}-\langle 1,1,1\rangle & \texttt{start}-17 & \texttt{duration}-7 & \texttt{end}-24 \end{array}\right\rangle,\\ \left\langle\begin{array}{llll} \texttt{sid}-1 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,1,1\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-2 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,2,2\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-3 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,4,2\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-4 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,4,1\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-5 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 2,3,1\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle,\\ \texttt{sid}-6 & \texttt{t}-\langle 0,0,0\rangle & \texttt{l}-\langle 1,2,2\rangle & \texttt{f}-\langle \texttt{dim}-1\ \texttt{dir}-0,\ \texttt{dim}-2\ \texttt{dir}-1\rangle \end{array}\right\rangle \end{array}\right)$$

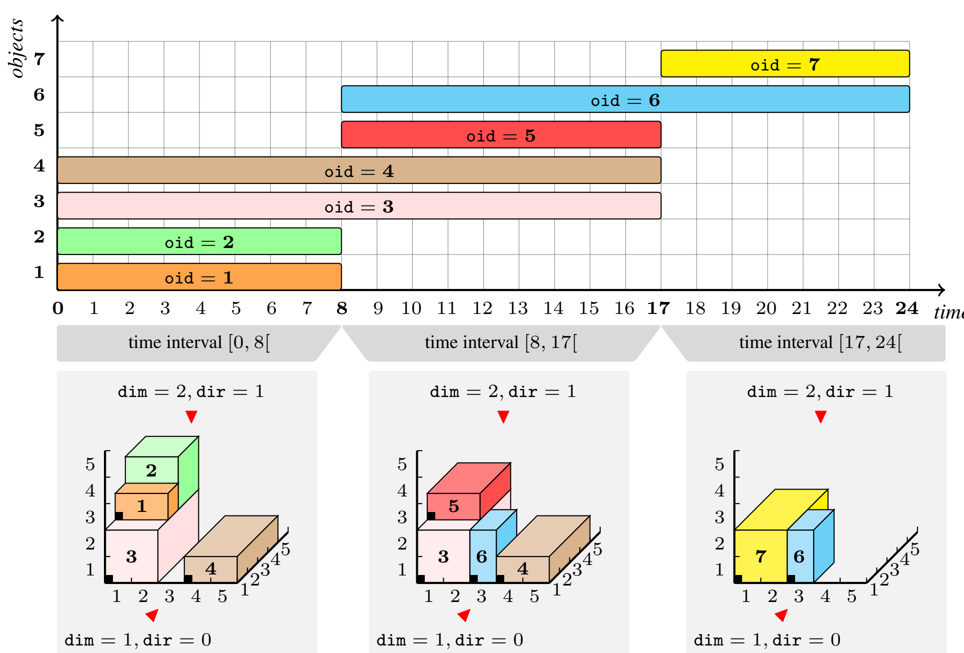
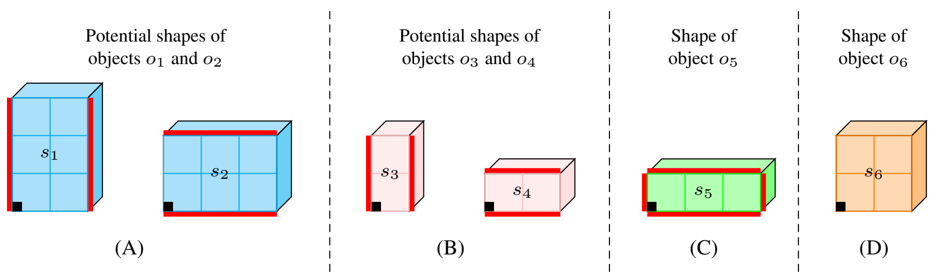
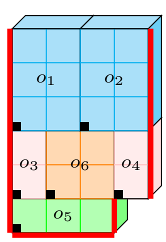


Figure 5.851: Illustration of the pick-up delivery problem



VISIBLE



(E)

$$
\left(\begin{array}{l}
3,\ \{0,1,2\},\ \langle \mathtt{dim}-0\,\mathtt{dir}-0, \mathtt{dim}-0\,\mathtt{dir}-1, \mathtt{dim}-1\,\mathtt{dir}-0, \mathtt{dim}-1\,\mathtt{dir}-1\rangle, \\
\left\langle \begin{array}{llllll}
\mathtt{oid}-o_1 & \mathtt{sid}-s_1 & \mathtt{x}-\langle 1,4,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\
\mathtt{oid}-o_2 & \mathtt{sid}-s_1 & \mathtt{x}-\langle 3,4,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\
\mathtt{oid}-o_3 & \mathtt{sid}-s_2 & \mathtt{x}-\langle 1,2,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\
\mathtt{oid}-o_4 & \mathtt{sid}-s_2 & \mathtt{x}-\langle 4,1,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\
\mathtt{oid}-o_5 & \mathtt{sid}-s_3 & \mathtt{x}-\langle 1,1,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1, \\
\mathtt{oid}-o_6 & \mathtt{sid}-s_4 & \mathtt{x}-\langle 2,2,1\rangle & \mathtt{start}-0 & \mathtt{duration}-1 & \mathtt{end}-1
\end{array}\right\rangle, \\
\left\langle \begin{array}{llll}
\mathtt{sid}-s_1 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 2,3,1\rangle & \mathtt{f}-\langle \mathtt{dim}-0\,\mathtt{dir}-0, \mathtt{dim}-0\,\mathtt{dir}-1\rangle, \\
\mathtt{sid}-s_2 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 3,2,1\rangle & \mathtt{f}-\angle \mathtt{dim}-1\,\mathtt{dir}-0, \mathtt{dim}-1\,\mathtt{dir}-1\rangle, \\
\mathtt{sid}-s_3 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 1,2,1\rangle & \mathtt{f}-\langle \mathtt{dim}-0\,\mathtt{dir}-0, \mathtt{dim}-0\,\mathtt{dir}-1\rangle, \\
\mathtt{sid}-s_4 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 2,1,1\rangle & \mathtt{f}-\langle \mathtt{dim}-1\,\mathtt{dir}-0, \mathtt{dim}-1\,\mathtt{dir}-1\rangle, \\
\mathtt{sid}-s_5 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 3,1,1\rangle & \mathtt{f}-\left\langle \begin{array}{l} \mathtt{dim}-0\,\mathtt{dir}-0, \mathtt{dim}-0\,\mathtt{dir}-1, \\ \mathtt{dim}-1\,\mathtt{dir}-0, \mathtt{dim}-1\,\mathtt{dir}-1 \end{array}\right\rangle, \\
\mathtt{sid}-s_6 & \mathtt{t}-\langle 0,0,0\rangle & \mathtt{l}-\langle 2,2,1\rangle & \mathtt{f}-\langle\rangle,
\end{array}\right\rangle
\end{array}\right)
$$

(F)

Figure 5.852: Illustration of the pallet loading problem

5.428 WEIGHTED_PARTIAL_ALLDIFF

DESCRIPTION LINKS GRAPH

Origin [417, page 71]

Constraint WEIGHTED_PARTIAL_ALLDIFF(VARIABLES, UNDEFINED, VALUES, COST)

Synonyms WEIGHTED_PARTIAL_ALLDIFFERENT, WEIGHTED_PARTIAL_ALLDISTINCT, WPA.

Arguments

VARIABLES	:	collection(var−dvar)
UNDEFINED	:	int
VALUES	:	collection(val−int, weight−int)
COST	:	dvar

Restrictions

required(VARIABLES, var)
|VALUES| > 0
required(VALUES, [val, weight])
in_attr(VARIABLES, var, VALUES, val)
distinct(VALUES, val)

Purpose All variables of the VARIABLES collection that are not assigned to value UNDEFINED must have pairwise distinct values from the val attribute of the VALUES collection. In addition COST is the sum of the weight attributes associated with the values assigned to the variables of VARIABLES. Within the VALUES collection, value UNDEFINED must be explicitly defined with a weight of 0.

Example

$$\left(\begin{array}{l}\langle 4,0,1,2,0,0\rangle, 0, \\ \left\langle\begin{array}{ll}\texttt{val}-0 & \texttt{weight}-0, \\ \texttt{val}-1 & \texttt{weight}-2, \\ \texttt{val}-2 & \texttt{weight}-{-1}, \\ \texttt{val}-4 & \texttt{weight}-7, \\ \texttt{val}-5 & \texttt{weight}-{-8}, \\ \texttt{val}-6 & \texttt{weight}-2\end{array}\right\rangle, 8\end{array}\right)$$

The WEIGHTED_PARTIAL_ALLDIFF constraint holds since:

- No value, except value UNDEFINED $= 0$, is used more than once.
- COST $= 8$ is equal to the sum of the weights 2, -1 and 7 of the values 1, 2 and 4 assigned to the variables of VARIABLES $= \langle 4, 0, 1, 2, 0, 0\rangle$.

Typical

|VARIABLES| > 0
ATLEAST(1, VARIABLES, UNDEFINED)
|VARIABLES| ≤ |VALUES| + 2

Symmetries

- Items of VARIABLES are permutable.
- Items of VALUES are permutable.
- All occurrences of two distinct values in VARIABLES.var or VALUES.val that are both different from UNDEFINED can be swapped; all occurrences of a value in VARIABLES.var or VALUES.val that is different from UNDEFINED can be renamed to any unused value that is also different from UNDEFINED.

Arg. properties

Functional dependency: COST determined by VARIABLES and VALUES.

Usage

In his PhD thesis [417, pages 71–72], Sven Thiel describes the following three potential scenarios of the WEIGHTED_PARTIAL_ALLDIFF constraint:

- Given a set of tasks (i.e., the items of the VARIABLES collection), assign to each task a resource (i.e., an item of the VALUES collection). Except for the resource associated with value UNDEFINED, every resource can be used at most once. The cost of a resource is independent from the task to which the resource is assigned. The cost of value UNDEFINED is equal to 0. The total cost COST of an assignment corresponds to the sum of the costs of the resources effectively assigned to the tasks. Finally we impose an upper bound on the total cost.
- Given a set of persons (i.e., the items of the VARIABLES collection), select for each person an offer (i.e., an item of the VALUES collection). Except for the offer associated with value UNDEFINED, every offer should be selected at most once. The profit associated with an offer is independent from the person that selects the offer. The profit of value UNDEFINED is equal to 0. The total benefit COST is equal to the sum of the profits of the offers effectively selected. In addition we impose a lower bound on the total benefit.
- The last scenario deals with an application to an over-constraint problem involving the ALLDIFFERENT constraint. Allowing some variables to take an ”undefined” value is done by setting all weights of all the values different from UNDEFINED to 1. As a consequence all variables assigned to a value different from UNDEFINED will have to take distinct values. The COST variable allows one to control the number of such variables.

Remark

It was shown in [417, page 104] that, finding out whether the WEIGHTED_PARTIAL_ALLDIFF constraint has a solution or not is NP-hard. This was achieved by reduction from subset sum.

Algorithm

A filtering algorithm is given in [417, pages 73–104]. After showing that, deciding whether the WEIGHTED_PARTIAL_ALLDIFF has a solution is NP-complete, [417, pages 105–106] gives the following results of his filtering algorithm with respect to consistency under the 3 scenarios previously described:

- For scenario 1, if there is no restriction of the lower bound of the COST variable, the filtering algorithm achieves arc-consistency for all variables of the VARIABLES collection (but not for the COST variable itself).
- For scenario 2, if there is no restriction of the upper bound of the COST variable, the filtering algorithm achieves arc-consistency for all variables of the VARIABLES collection (but not for the COST variable itself).

- Finally, for scenario 3, the filtering algorithm achieves arc-consistency for all variables of the VARIABLES collection as well as for the COST variable.

See also

attached to cost variant: ALLDIFFERENT, ALLDIFFERENT_EXCEPT_0.

common keyword: GLOBAL_CARDINALITY_WITH_COSTS *(weighted assignment)*, MINIMUM_WEIGHT_ALLDIFFERENT *(cost filtering constraint,weighted assignment)*, SOFT_ALLDIFFERENT_VAR *(soft constraint)*, SUM_OF_WEIGHTS_OF_DISTINCT_VALUES *(weighted assignment)*.

Keywords

application area: assignment.

characteristic of a constraint: all different, joker value.

complexity: subset sum.

constraint type: soft constraint, relaxation.

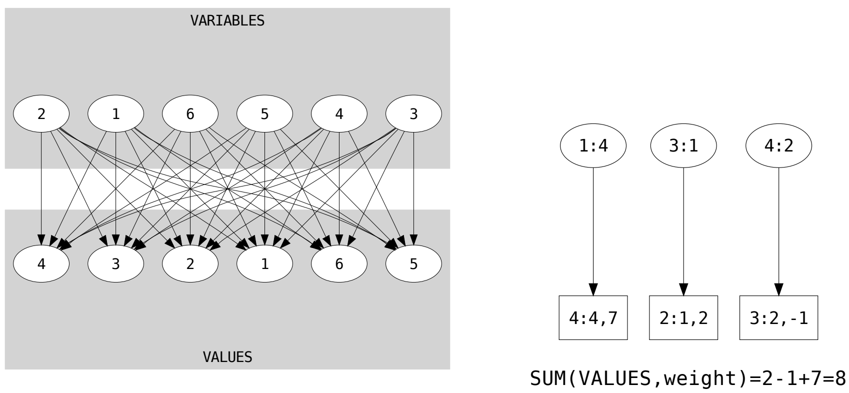
filtering: cost filtering constraint.

modelling: functional dependency.

problems: weighted assignment.

Arc input(s)	VARIABLES VALUES
Arc generator	*PRODUCT* $\mapsto$ collection(variables, values)
Arc arity	2
Arc constraint(s)	• variables.var $\neq$ UNDEFINED • variables.var $=$ values.val
Graph property(ies)	• **MAX_ID** ≤ 1 • **SUM**(VALUES, weight) $=$ COST

Graph model Parts (A) and (B) of Figure 5.853 respectively show the initial and final graph associated with the **Example** slot. Since we also use the **SUM** graph property we show the vertices of the final graph from which we compute the total cost in a box.



(A) **(B)**

Figure 5.853: Initial and final graph of the WEIGHTED_PARTIAL_ALLDIFF constraint

5.429 XOR

▶ ▷ ◁ ◀ DESCRIPTION LINKS AUTOMATON

Origin Logic

Constraint XOR(VAR, VARIABLES)

Synonyms ODD, REL.

Arguments

VAR : dvar
VARIABLES : collection(var−dvar)

Restrictions

VAR ≥ 0
VAR ≤ 1
$|$VARIABLES$| = 2$
required(VARIABLES, var)
VARIABLES.var ≥ 0
VARIABLES.var ≤ 1

Purpose

Let VARIABLES be a collection of 0-1 variables $\text{VAR}_1, \text{VAR}_2$. Enforce $\text{VAR} = (\text{VAR}_1 \neq \text{VAR}_2)$.

Example

$(0, \langle 0, 0\rangle)$
$(1, \langle 0, 1\rangle)$
$(1, \langle 1, 0\rangle)$
$(0, \langle 1, 1\rangle)$

Symmetry Items of VARIABLES are permutable.

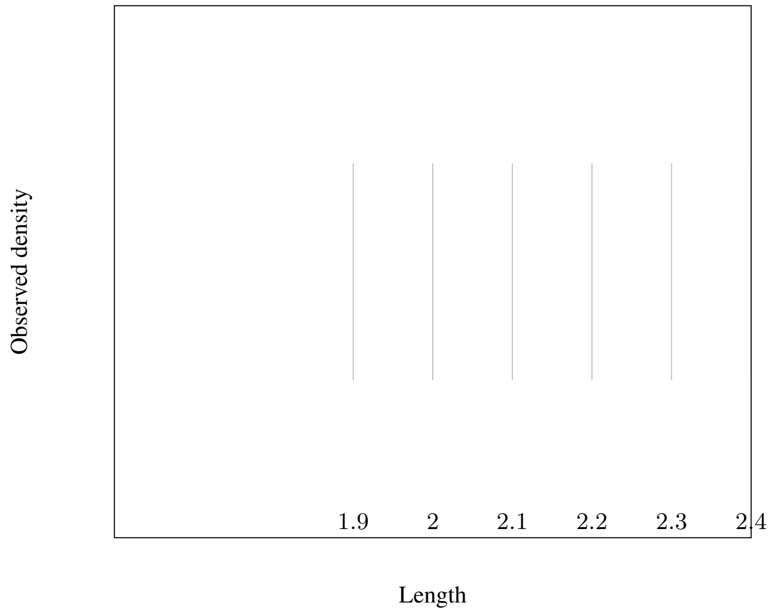
Arg. properties Functional dependency: VAR determined by VARIABLES.

Counting

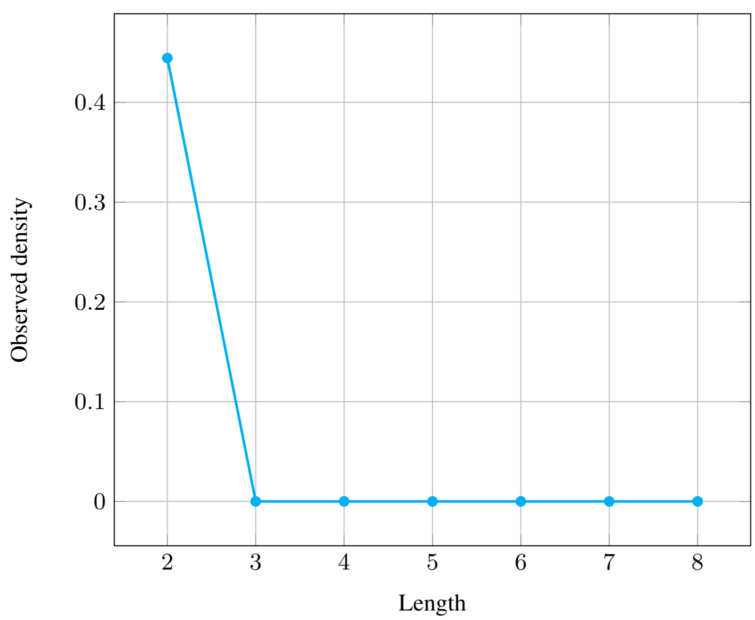
Length (n)	2	3	4	5	6	7	8
Solutions	4	0	0	0	0	0	0

Number of solutions for XOR: domains $0..n$

Solution density for XOR

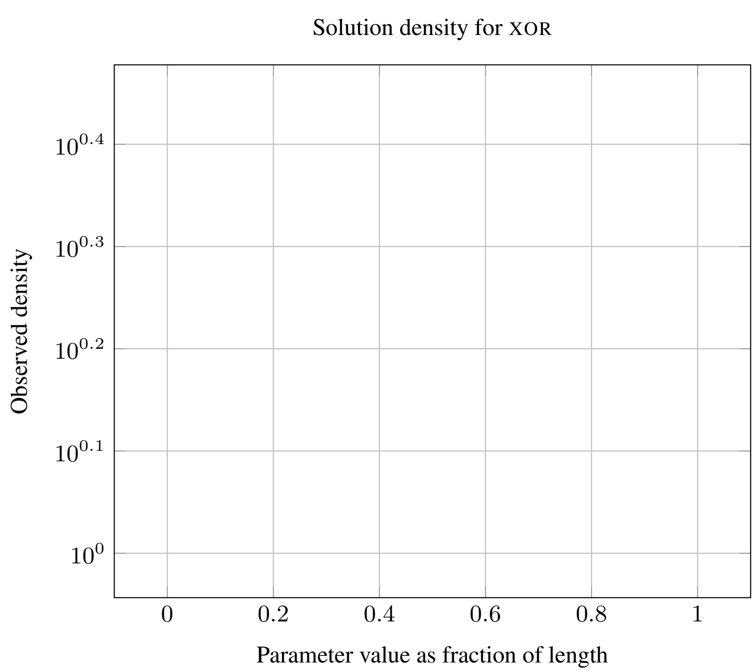


Solution density for XOR

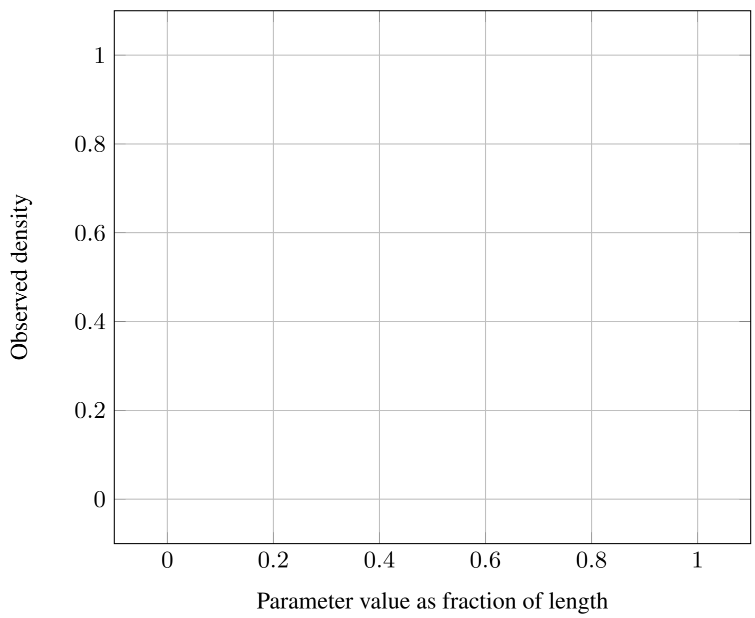


Length (n)		2
Total		4
Parameter value	0	2
	1	2

Solution count for XOR: domains $0..n$



Solution density for XOR



Systems REIFIEDXOR in **Choco**, REL in **Gecode**, XORBOOL in **JaCoP**, #\ in **SICStus**.

See also **common keyword:** AND, EQUIVALENT, IMPLY, NAND, NOR, OR *(Boolean constraint)*.

implies: ATLEAST_NVALUE, SOFT_ALL_EQUAL_MAX_VAR, SOFT_ALL_EQUAL_MIN_VAR.

Keywords **characteristic of a constraint:** automaton, automaton without counters, reified automaton constraint.

constraint arguments: pure functional dependency.

constraint network structure: Berge-acyclic constraint network.

constraint type: Boolean constraint.

filtering: arc-consistency.

modelling: functional dependency.

Automaton Figure 5.854 depicts the automaton associated with the XOR constraint. To the first argument VAR of the XOR constraint corresponds the first signature variable. To each variable $\mathtt{VAR}_i$ of the second argument VARIABLES of the XOR constraint corresponds the next signature variable. There is no signature constraint.

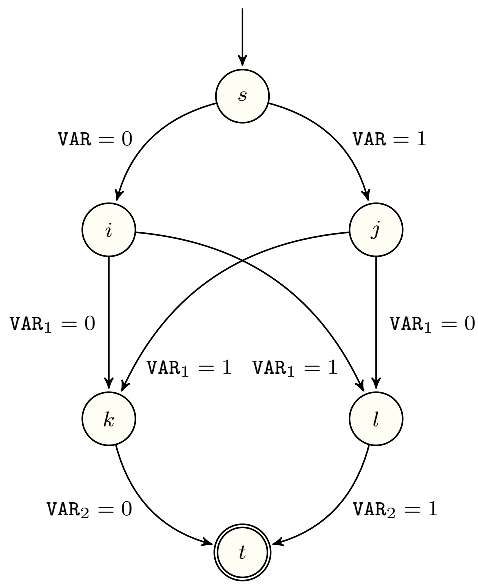


Figure 5.854: Automaton of the XOR constraint

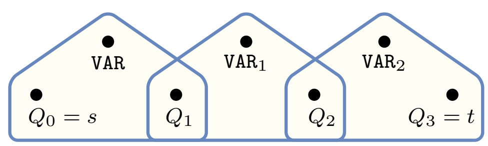


Figure 5.855: Hypergraph of the reformulation corresponding to the automaton of the XOR constraint

5.430 ZERO_OR_NOT_ZERO

► ◁ ◄ DESCRIPTION LINKS

Origin Arithmetic.

Constraint ZERO_OR_NOT_ZERO(VAR1, VAR2)

Synonyms ZEROS_OR_NOT_ZEROS, NOT_ZERO_OR_ZERO, NOT_ZEROS_OR_ZEROS.

Arguments

VAR1	:	dvar
VAR2	:	dvar

Purpose Enforce the fact that either both variables are equal to 0, or both variables are not equal to 0.

Example $(1, 8)$

The ZERO_OR_NOT_ZERO constraint holds since values 1 and 8 are both not equal to zero.

Symmetry Arguments are permutable w.r.t. permutation (VAR1, VAR2).

See also **implied by:** ABS_VALUE, DIVISIBLE_OR, EQ, SIGN_OF.

implies (if swap arguments): ABS_VALUE.

Keywords **constraint arguments:** binary constraint.

constraint type: predefined constraint, arithmetic constraint.

5.431 ZERO_OR_NOT_ZERO_VECTORS

◁ ◀ DESCRIPTION LINKS

Origin Tournament scheduling

Constraint ZERO_OR_NOT_ZERO_VECTORS(VECTORS)

Synonyms ZEROS_OR_NOT_ZEROS_VECTORS, NOT_ZERO_OR_ZERO_VECTORS, NOT_ZEROS_OR_ZEROS_VECTORS.

Type VECTOR : collection(var−dvar)

Argument VECTORS : collection(vec − VECTOR)

Restrictions
|VECTOR| ≥ 1
required(VECTOR, var)
|VECTORS| ≥ 1
required(VECTORS, vec)
same_size(VECTORS, vec)

Purpose Given a collection of vectors enforces for each vector that either all its components are equal to 0, or all its components are different from 0. In addition imposes that at least one 0 is used.

Example

$$\left(\left\langle\begin{array}{l}\texttt{vec} - \langle 5,6\rangle,\\ \texttt{vec} - \langle 5,6\rangle,\\ \texttt{vec} - \langle 0,0\rangle,\\ \texttt{vec} - \langle 9,3\rangle,\\ \texttt{vec} - \langle 0,0\rangle\end{array}\right\rangle\right)$$

The ZERO_OR_NOT_ZERO_VECTORS constraint holds since:

- Both components of the first vector $\langle 5,6\rangle$ are different from 0.
- Both components of the second vector $\langle 5,6\rangle$ are different from 0.
- Both components of the third vector $\langle 0,0\rangle$ are equal to 0.
- Both components of the fourth vector $\langle 9,3\rangle$ are different from 0.
- Both components of the fifth vector $\langle 0,0\rangle$ are equal to 0.

Typical
|VECTOR| > 1
|VECTORS| > 1

Arg. properties Contractible wrt. VECTORS.

Keywords **characteristic of a constraint:** vector.
constraint type: predefined constraint, arithmetic constraint.

Appendix A

Legend for the Description

This section provides the list of *restrictions*, of *arc generators*, of *graph parameters* and of *set generators* sorted in alphabetic order with the page where they are defined.

Restrictions :

Arc generators :

Graph parameters :

Set generators :

Appendix B

Electronic Constraint Catalogue

Contents

B.1 abs_value

◊ META-DATA:

```
ctr_predefined(abs_value).

ctr_date(abs_value,['20100821']).

ctr_origin(abs_value,'Arithmetic.',[]).

ctr_usual_name(abs_value,abs).

ctr_synonyms(abs_value,[absolute_value]).

ctr_arguments(abs_value,['Y'-dvar,'X'-dvar]).

ctr_restrictions(abs_value,['Y'>=0]).

ctr_example(abs_value,abs_value(8,-8)).

ctr_eval(abs_value,[checker(abs_value_c),builtin(abs_value_b)]).

ctr_pure_functional_dependency(abs_value,[]).

ctr_functional_dependency(abs_value,1,[2]).

abs_value_c(Y,X) :-
        check_type(int,Y),
        check_type(int,X),
        Y is abs(X).

abs_value_b(Y,X) :-
        check_type(dvar,Y),
        check_type(dvar,X),
        X#>=0#/\Y#=X#\/X#<0#/\X+Y#=0.
```

B.2 all_balance

♢ META-DATA:

```
ctr_predefined(all_balance).

ctr_date(all_balance,['20141014']).

ctr_origin(
    all_balance,
    derived from %c in \cite{BessiereHebrardKatsirelosKiziltanPicardCantinQuimperWals
    [balance]).

ctr_arguments(
    all_balance,
    ['BALANCE'-dvar,'VARIABLES'-collection(var-dvar),'M'-int]).

ctr_restrictions(
    all_balance,
    ['BALANCE'>=0,
     'BALANCE'=<size('VARIABLES'),
     required('VARIABLES',var),
     'VARIABLES'^var>=1,
     'VARIABLES'^var=<'M',
     'M'>=1,
     'M'=<size('VARIABLES')]).

ctr_example(
    all_balance,
    [all_balance(
         2,
         [[var-3],[var-1],[var-2],[var-1],[var-1]],
         3),
     all_balance(
         3,
         [[var-3],[var-1],[var-2],[var-1],[var-1]],
         4)]).

ctr_typical(
    all_balance,
    ['BALANCE'=<2+size('VARIABLES')/10,size('VARIABLES')>2]).

ctr_typical_model(all_balance,[nval('VARIABLES'^var)>2]).

ctr_eval(
    all_balance,
```

```
    [checker(all_balance_c),reformulation(all_balance_r)]).

ctr_pure_functional_dependency(all_balance,[]).

ctr_functional_dependency(all_balance,1,[2,3]).

all_balance_c(BALANCE,VARIABLES,M) :-
        check_type(dvar,BALANCE),
        collection(VARIABLES,[int(1,M)]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        N>0,
        BALANCE#>=0,
        BALANCE#=<N,
        check_type(int(1,N),M),
        samsort(VARS,SVARS),
        SVARS=[V|R],
        min_max_seq_size0(R,M,1,V,N,1,MIN,MAX),
        BALANCE#=MAX-MIN.

min_max_seq_size0([],M,C,_28658,BestMin,BestMax,ResMin,ResMax) :-
        !,
        (   M=1 ->
            ResMin is min(C,BestMin)
        ;   ResMin is 0
        ),
        ResMax is max(C,BestMax).

min_max_seq_size0([V|R],M,C,V,BestMin,BestMax,ResMin,ResMax) :-
        !,
        C1 is C+1,
        min_max_seq_size0(
            R,
            M,
            C1,
            V,
            BestMin,
            BestMax,
            ResMin,
            ResMax).

min_max_seq_size0([V|R],M,C,Prev,BestMin,BestMax,ResMin,ResMax) :-
        C>0,
        V=\=Prev,
        NewBestMin is min(C,BestMin),
        NewBestMax is max(C,BestMax),
```

```
        NewM is M-1,
        min_max_seq_size0(
            R,
            NewM,
            1,
            V,
            NewBestMin,
            NewBestMax,
            ResMin,
            ResMax).

all_balance_r(BALANCE,VARIABLES,M) :-
        check_type(dvar,BALANCE),
        collection(VARIABLES,[dvar(1,M)]),
        length(VARIABLES,N),
        N>0,
        BALANCE#>=0,
        BALANCE#=<N,
        check_type(int(1,N),M),
        create_nocc_vars(1,M,N,VALS,OCCS),
        eval(global_cardinality(VARIABLES,VALS)),
        Vmin is(N+M-1)//M,
        MIN in 0..Vmin,
        eval(minimum(MIN,OCCS)),
        Vmax is N//M,
        MAX in Vmax..N,
        eval(maximum(MAX,OCCS)),
        BALANCE+MIN#=MAX,
        M1 is M-1,
        M*MAX-M1*BALANCE#=<N,
        M*MIN+M1*BALANCE#>=N,
        Diff is 1+Vmax-Vmin,
        BALANCE#\=Diff.
```

B.3 all_differ_from_at_least_k_pos

◇ META-DATA:

```
ctr_date(
    all_differ_from_at_least_k_pos,
    ['20030820','20040530','20060803']).

ctr_origin(
    all_differ_from_at_least_k_pos,
    Inspired by \cite{Frutos97}.,
    []).

ctr_types(
    all_differ_from_at_least_k_pos,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    all_differ_from_at_least_k_pos,
    ['K'-int,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    all_differ_from_at_least_k_pos,
    [required('VECTOR',var),
     size('VECTOR')>=1,
     size('VECTOR')>='K',
     'K'>=0,
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    all_differ_from_at_least_k_pos,
    all_differ_from_at_least_k_pos(
        2,
        [[vec-[[var-2],[var-5],[var-2],[var-0]]],
         [vec-[[var-3],[var-6],[var-2],[var-1]]],
         [vec-[[var-3],[var-6],[var-1],[var-0]]]])).

ctr_typical(
    all_differ_from_at_least_k_pos,
    ['K'>0,size('VECTORS')>1]).

ctr_exchangeable(
    all_differ_from_at_least_k_pos,
    [items('VECTORS',all),items_sync('VECTORS'^vec,all)]).
```

```
ctr_graph(
    all_differ_from_at_least_k_pos,
    ['VECTORS'],
    2,
    ['CLIQUE'(=\=)>>collection(vectors1,vectors2)],
    [differ_from_at_least_k_pos(
         K,
         vectors1^vec,
         vectors2^vec)],
    ['NARC'=size('VECTORS')*size('VECTORS')-size('VECTORS')],
    ['NO_LOOP','SYMMETRIC']).

ctr_eval(
    all_differ_from_at_least_k_pos,
    [reformulation(all_differ_from_at_least_k_pos_r),
     checker(all_differ_from_at_least_k_pos_c),
     density(all_differ_from_at_least_k_pos_d)]).

ctr_contractible(
    all_differ_from_at_least_k_pos,
    [],
    VECTORS,
    any).

ctr_extensible(
    all_differ_from_at_least_k_pos,
    [],
    'VECTORS'^vec,
    any).

ctr_cond_imply(
    all_differ_from_at_least_k_pos,
    atleast_nvector,
    ['K'=<size('VECTORS')],
    [],
    id).

all_differ_from_at_least_k_pos_r(K,VECTORS) :-
        integer(K),
        K>=0,
        all_differ_from_at_least_k_pos_rr(VECTORS,K).

all_differ_from_at_least_k_pos_c(K,[]) :-
        !,
        integer(K),
        K>=0.
```

```
all_differ_from_at_least_k_pos_c(0,VECTORS) :-
        !,
        collection(VECTORS,[col([int])]),
        VECTORS=[[_41188-VECTOR]|_41184],
        length(VECTOR,N),
        N>=1,
        same_size(VECTORS).

all_differ_from_at_least_k_pos_c(1,VECTORS) :-
        !,
        collection(VECTORS,[col([int])]),
        VECTORS=[[_41188-VECTOR]|_41184],
        length(VECTOR,N),
        N>=1,
        same_size(VECTORS),
        length(VECTORS,L),
        sort(VECTORS,SVECTORS),
        length(SVECTORS,L).

all_differ_from_at_least_k_pos_c(K,VECTORS) :-
        integer(K),
        VECTORS=[[_41180-VECTOR]|_41176],
        length(VECTOR,N),
        K=N,
        !,
        N>=1,
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        transpose(VECTS,TVECTS),
        length(VECTORS,M),
        all_differ_from_at_least_k_pos_distinct_comp(TVECTS,M).

all_differ_from_at_least_k_pos_c(K,VECTORS) :-
        integer(K),
        collection(VECTORS,[col([int])]),
        VECTORS=[[_41192-VECTOR]|_41188],
        length(VECTOR,N),
        N>=1,
        N>=K,
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        all_differ_from_at_least_k_pos_cc(VECTS,N,K).

all_differ_from_at_least_k_pos_distinct_comp([],_41159) :-
```

```
        !.

all_differ_from_at_least_k_pos_distinct_comp([L|R],N) :-
        sort(L,S),
        length(S,N),
        all_differ_from_at_least_k_pos_distinct_comp(R,N).

all_differ_from_at_least_k_pos_check_pairs([],_41159,_41160) :-
        !.

all_differ_from_at_least_k_pos_check_pairs(
    [_41161],
    _41527,
    _41574) :-
        !.

all_differ_from_at_least_k_pos_check_pairs(
    [t(I,J)-_41165,t(I,J)-K|R],
    CUR,
    LIMIT) :-
        !,
        NEXT is CUR+1,
        NEXT=<LIMIT,
        all_differ_from_at_least_k_pos_check_pairs(
            [t(I,J)-K|R],
            NEXT,
            LIMIT).

all_differ_from_at_least_k_pos_check_pairs(
    [_41161,t(I,J)-K|R],
    _CUR,
    LIMIT) :-
        all_differ_from_at_least_k_pos_check_pairs(
            [t(I,J)-K|R],
            1,
            LIMIT).

all_differ_from_at_least_k_pos_gen_pairs([],_41159,[],_41161) :-
        !.

all_differ_from_at_least_k_pos_gen_pairs([L|R],Id,CONT,RES) :-
        all_differ_from_at_least_k_pos_gen_pair(
            L,
            Id,
            Id1,
            CONT,
```

```
            NEW_CONT),
        all_differ_from_at_least_k_pos_gen_pairs(
            R,
            Id1,
            NEW_CONT,
            RES).

all_differ_from_at_least_k_pos_gen_pair([],Id,Id,CONT,CONT) :-
        !.

all_differ_from_at_least_k_pos_gen_pair(
    [X|Y],
    Id,
    NewId,
    CONT,
    NEW_CONT) :-
        all_differ_from_at_least_k_pos_gen_pair(
            Y,
            X,
            Id,
            Id1,
            CONT,
            CCONT),
        all_differ_from_at_least_k_pos_gen_pair(
            Y,
            Id1,
            NewId,
            CCONT,
            NEW_CONT).

all_differ_from_at_least_k_pos_gen_pair(
    [],
    _41461,
    Id,
    Id,
    CONT,
    CONT) :-
        !.

all_differ_from_at_least_k_pos_gen_pair(
    [X|Y],
    Z,
    Id,
    NewId,
    CONT,
    CCONT) :-
```

```
        MIN is min(X,Z),
        MAX is max(X,Z),
        CONT=[t(MIN,MAX)-Id|NEW_CONT],
        Id1 is Id+1,
        all_differ_from_at_least_k_pos_gen_pair(
            Y,
            Z,
            Id1,
            NewId,
            NEW_CONT,
            CCONT).

all_differ_from_at_least_k_pos_length([],LEN,LEN) :-
        !.

all_differ_from_at_least_k_pos_length([L|R],CUR,RES) :-
        length(L,N),
        NEXT is CUR+N*(N-1)//2,
        all_differ_from_at_least_k_pos_length(R,NEXT,RES).

all_differ_from_at_least_k_pos_regroup([],[]) :-
        !.

all_differ_from_at_least_k_pos_regroup([t(Val,Pos,Id)|Y],RES) :-
        all_differ_from_at_least_k_pos_prefix(
            [t(Val,Pos,Id)|Y],
            t(Val,Pos,Id),
            P,
            Rest),
        (   length(P,1) ->
            RES=R
        ;   RES=[P|R]
        ),
        all_differ_from_at_least_k_pos_regroup(Rest,R).

all_differ_from_at_least_k_pos_prefix(
    [t(Val,Pos,Id)|Y],
    t(Val,Pos,Jd),
    [Id|S],
    R) :-
        !,
        all_differ_from_at_least_k_pos_prefix(
            Y,
            t(Val,Pos,Jd),
            S,
            R).
```

```
all_differ_from_at_least_k_pos_prefix(L,_41156,[],L).

all_differ_from_at_least_k_pos_gen_triples([],_41159,[],_41161) :-
        !.

all_differ_from_at_least_k_pos_gen_triples(
    [VECTOR|R],
    ID,
    CONTINUATION,
    TRIPLES) :-
        all_differ_from_at_least_k_pos_gen_triples1(
            VECTOR,
            1,
            ID,
            CONTINUATION,
            NEW_CONTINUATION,
            TRIPLES),
        ID1 is ID+1,
        all_differ_from_at_least_k_pos_gen_triples(
            R,
            ID1,
            NEW_CONTINUATION,
            TRIPLES).

all_differ_from_at_least_k_pos_gen_triples1(
    [],
    _41451,
    _41498,
    CONTINUATION,
    CONTINUATION,
    _41635) :-
        !.

all_differ_from_at_least_k_pos_gen_triples1(
    [V|R],
    I,
    ID,
    CONTINUATION,
    NEW_CONTINUATION,
    TRIPLES) :-
        CONTINUATION=[t(V,I,ID)|CONT],
        I1 is I+1,
        all_differ_from_at_least_k_pos_gen_triples1(
            R,
            I1,
```

```
            ID,
            CONT,
            NEW_CONTINUATION,
            TRIPLES).

all_differ_from_at_least_k_pos_rr([],_41159) :-
        !.

all_differ_from_at_least_k_pos_rr([[_41165-VECTOR1]|R],K) :-
        length(VECTOR1,N),
        N>=1,
        N>=K,
        all_differ_from_at_least_k_pos_rr(R,VECTOR1,K),
        all_differ_from_at_least_k_pos_rr(R,K).

all_differ_from_at_least_k_pos_rr([],_41156,_41157).

all_differ_from_at_least_k_pos_rr(
    [[_41166-VECTOR2]|R],
    VECTOR1,
    K) :-
        eval(differ_from_at_least_k_pos(K,VECTOR1,VECTOR2)),
        all_differ_from_at_least_k_pos_rr(R,VECTOR1,K).

all_differ_from_at_least_k_pos_cc([],_41159,_41160) :-
        !.

all_differ_from_at_least_k_pos_cc([VECTOR1|R],N,K) :-
        all_differ_from_at_least_k_pos_cc(R,VECTOR1,N,K),
        all_differ_from_at_least_k_pos_cc(R,N,K).

all_differ_from_at_least_k_pos_cc([],_41156,_41157,_41158).

all_differ_from_at_least_k_pos_cc([VECTOR2|R],VECTOR1,N,K) :-
        all_differ_from_at_least_k_pos_check(
            VECTOR1,
            VECTOR2,
            N,
            K),
        all_differ_from_at_least_k_pos_cc(R,VECTOR1,N,K).

all_differ_from_at_least_k_pos_check([],[],_41160,0) :-
        !.

all_differ_from_at_least_k_pos_check([U|R],[V|S],N,K) :-
        (   U=V ->
```

```
            NewK is K
        ;   NewK is K-1
        ),
        (   NewK=<0 ->
            true
        ;   NewN is N-1,
            NewK=<NewN,
            all_differ_from_at_least_k_pos_check(R,S,NewN,NewK)
        ).

all_differ_from_at_least_k_pos_d(0,_41159,[]) :-
        !.

all_differ_from_at_least_k_pos_d(
    Density,
    K,
    [[_41170-VECTOR]|_41166]) :-
        length(VECTOR,Available),
        Density is K/Available.
```

B.4 all_differ_from_at_most_k_pos

◇ **META-DATA:**

```
ctr_date(all_differ_from_at_most_k_pos,['20120228']).

ctr_origin(
    all_differ_from_at_most_k_pos,
    Inspired by %c.,
    [all_differ_from_at_least_k_pos]).

ctr_types(
    all_differ_from_at_most_k_pos,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    all_differ_from_at_most_k_pos,
    ['K'-int,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    all_differ_from_at_most_k_pos,
    [required('VECTOR',var),
     size('VECTOR')>=1,
     size('VECTOR')>='K',
     'K'>=0,
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    all_differ_from_at_most_k_pos,
    all_differ_from_at_most_k_pos(
        2,
        [[vec-[[var-0],[var-3],[var-0],[var-6]]],
         [vec-[[var-0],[var-3],[var-4],[var-1]]],
         [vec-[[var-0],[var-3],[var-4],[var-6]]]])).

ctr_typical(
    all_differ_from_at_most_k_pos,
    ['K'>0,'K'<size('VECTOR'),size('VECTORS')>1]).

ctr_exchangeable(
    all_differ_from_at_most_k_pos,
    [items('VECTORS',all),items_sync('VECTORS'^vec,all)]).

ctr_graph(
    all_differ_from_at_most_k_pos,
```

```
    ['VECTORS'],
    2,
    ['CLIQUE'(=\=)>>collection(vectors1,vectors2)],
    [differ_from_at_most_k_pos('K',vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')*size('VECTORS')-size('VECTORS')],
    ['NO_LOOP','SYMMETRIC']).

ctr_eval(
    all_differ_from_at_most_k_pos,
    [reformulation(all_differ_from_at_most_k_pos_r),
     checker(all_differ_from_at_most_k_pos_c)]).

ctr_contractible(
    all_differ_from_at_most_k_pos,
    [],
    VECTORS,
    any).

ctr_contractible(
    all_differ_from_at_most_k_pos,
    [],
    'VECTORS'^vec,
    any).

all_differ_from_at_most_k_pos_r(K,VECTORS) :-
        integer(K),
        K>=0,
        all_differ_from_at_most_k_pos_rr(VECTORS,K).

all_differ_from_at_most_k_pos_c(K,[]) :-
        !,
        integer(K),
        K>=0.

all_differ_from_at_most_k_pos_c(K,VECTORS) :-
        integer(K),
        K>=0,
        VECTORS=[[_39498-VECTOR]|_39494],
        length(VECTOR,N),
        N>=1,
        N>=K,
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        all_differ_from_at_most_k_pos_cc(VECTS,N,K).

all_differ_from_at_most_k_pos_rr([],_39471) :-
```

```
        !.

all_differ_from_at_most_k_pos_rr([[_39477-VECTOR1]|R],K) :-
        length(VECTOR1,N),
        N>=1,
        N>=K,
        all_differ_from_at_most_k_pos_rr(R,VECTOR1,K),
        all_differ_from_at_most_k_pos_rr(R,K).

all_differ_from_at_most_k_pos_rr([],_39468,_39469).

all_differ_from_at_most_k_pos_rr(
    [[_39478-VECTOR2]|R],
    VECTOR1,
    K) :-
        eval(differ_from_at_most_k_pos(K,VECTOR1,VECTOR2)),
        all_differ_from_at_most_k_pos_rr(R,VECTOR1,K).

all_differ_from_at_most_k_pos_cc([],_39471,_39472) :-
        !.

all_differ_from_at_most_k_pos_cc([VECTOR1|R],N,K) :-
        all_differ_from_at_most_k_pos_cc(R,VECTOR1,N,K),
        all_differ_from_at_most_k_pos_cc(R,N,K).

all_differ_from_at_most_k_pos_cc([],_39468,_39469,_39470).

all_differ_from_at_most_k_pos_cc([VECTOR2|R],VECTOR1,N,K) :-
        all_differ_from_at_most_k_pos_check(
            VECTOR1,
            VECTOR2,
            N,
            K),
        all_differ_from_at_most_k_pos_cc(R,VECTOR1,N,K).

all_differ_from_at_most_k_pos_check([],[],_39472,0) :-
        !.

all_differ_from_at_most_k_pos_check([U|R],[V|S],N,K) :-
        (   U=V ->
            NewK is K
        ;   NewK is K-1,
            NewK>=0
        ),
        NewN is N-1,
        (   NewN=<NewK ->
```

```
        true
    ;   all_differ_from_at_most_k_pos_check(R,S,NewN,NewK)
    ).
```

B.5 all_differ_from_exactly_k_pos

◇ **META-DATA:**

```
ctr_date(all_differ_from_exactly_k_pos,['20120227']).

ctr_origin(
    all_differ_from_exactly_k_pos,
    Inspired by %c.,
    [all_differ_from_at_least_k_pos]).

ctr_types(
    all_differ_from_exactly_k_pos,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    all_differ_from_exactly_k_pos,
    ['K'-int,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    all_differ_from_exactly_k_pos,
    [required('VECTOR',var),
     size('VECTOR')>=1,
     size('VECTOR')>='K',
     'K'>=0,
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    all_differ_from_exactly_k_pos,
    all_differ_from_exactly_k_pos(
        2,
        [[vec-[[var-0],[var-3],[var-0],[var-6]]],
         [vec-[[var-0],[var-3],[var-4],[var-1]]],
         [vec-[[var-9],[var-3],[var-4],[var-6]]]])).

ctr_typical(
    all_differ_from_exactly_k_pos,
    ['K'>0,'K'<size('VECTOR'),size('VECTORS')>1]).

ctr_exchangeable(
    all_differ_from_exactly_k_pos,
    [items('VECTORS',all),items_sync('VECTORS'^vec,all)]).

ctr_graph(
    all_differ_from_exactly_k_pos,
```

```
    ['VECTORS'],
    2,
    ['CLIQUE'(=\=)>>collection(vectors1,vectors2)],
    [differ_from_exactly_k_pos('K',vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')*size('VECTORS')-size('VECTORS')],
    ['NO_LOOP','SYMMETRIC']).

ctr_eval(
    all_differ_from_exactly_k_pos,
    [reformulation(all_differ_from_exactly_k_pos_r),
     checker(all_differ_from_exactly_k_pos_c)]).

ctr_contractible(
    all_differ_from_exactly_k_pos,
    [],
    VECTORS,
    any).

ctr_cond_imply(
    all_differ_from_exactly_k_pos,
    atleast_nvector,
    ['K'=<size('VECTORS')],
    [],
    id).

all_differ_from_exactly_k_pos_r(K,[]) :-
        !,
        integer(K),
        K=0.

all_differ_from_exactly_k_pos_r(K,[_VECTOR]) :-
        !,
        integer(K),
        K=0.

all_differ_from_exactly_k_pos_r(K,VECTORS) :-
        integer(K),
        K>=0,
        all_differ_from_exactly_k_pos_rr(VECTORS,K).

all_differ_from_exactly_k_pos_c(K,[]) :-
        !,
        check_type(dvar,K),
        K=0.

all_differ_from_exactly_k_pos_c(K,VECTORS) :-
```

```
        check_type(dvar,K),
        K#>=0,
        length(VECTORS,L),
        (   integer(K) ->
            true
        ;   L>1 ->
            true
        ;   K=0
        ),
        VECTORS=[[_41351-VECTOR]|_41347],
        length(VECTOR,N),
        N>=1,
        N#>=K,
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        all_differ_from_exactly_k_pos_cc(VECTS,N,K).

all_differ_from_exactly_k_pos_rr([],_41292) :-
        !.

all_differ_from_exactly_k_pos_rr([[_41298-VECTOR1]|R],K) :-
        length(VECTOR1,N),
        N>=1,
        N#>=K,
        all_differ_from_exactly_k_pos_rr(R,VECTOR1,K),
        all_differ_from_exactly_k_pos_rr(R,K).

all_differ_from_exactly_k_pos_rr([],_41289,_41290).

all_differ_from_exactly_k_pos_rr(
    [[_41299-VECTOR2]|R],
    VECTOR1,
    K) :-
        eval(differ_from_exactly_k_pos(K,VECTOR1,VECTOR2)),
        all_differ_from_exactly_k_pos_rr(R,VECTOR1,K).

all_differ_from_exactly_k_pos_cc([],_41292,_41293) :-
        !.

all_differ_from_exactly_k_pos_cc([VECTOR1|R],N,K) :-
        all_differ_from_exactly_k_pos_cc(R,VECTOR1,N,K),
        all_differ_from_exactly_k_pos_cc(R,N,K).

all_differ_from_exactly_k_pos_cc([],_41289,_41290,_41291).

all_differ_from_exactly_k_pos_cc([VECTOR2|R],VECTOR1,N,K) :-
```

```
        (   integer(K) ->
            all_differ_from_exactly_k_pos_check(
                VECTOR1,
                VECTOR2,
                N,
                K)
        ;   count_differ_k_pos(VECTOR1,VECTOR2,0,K)
        ),
        all_differ_from_exactly_k_pos_cc(R,VECTOR1,N,K).

all_differ_from_exactly_k_pos_check([],[],_41293,0) :-
        !.

all_differ_from_exactly_k_pos_check([U|R],[V|S],N,K) :-
        (   U=V ->
            NewK is K
        ;   NewK is K-1,
            NewK>=0
        ),
        NewN is N-1,
        NewK=<NewN,
        all_differ_from_exactly_k_pos_check(R,S,NewN,NewK).

count_differ_k_pos([],[],C,C) :-
        !.

count_differ_k_pos([U|R],[V|S],C,K) :-
        (   U=V ->
            NewC is C
        ;   NewC is C+1
        ),
        count_differ_k_pos(R,S,NewC,K).
```

B.6 all_equal

◊ **META-DATA:**

```
ctr_date(all_equal,['20081005','20100418']).

ctr_origin(
    all_equal,
    Derived from %c,
    [soft_all_equal_min_ctr]).

ctr_synonyms(all_equal,[rel]).

ctr_arguments(all_equal,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    all_equal,
    [required('VARIABLES',var),size('VARIABLES')>0]).

ctr_example(
    all_equal,
    all_equal([[var-5],[var-5],[var-5],[var-5]])).

ctr_typical(
    all_equal,
    [size('VARIABLES')>2,minval('VARIABLES'^var)=\=0]).

ctr_typical_model(all_equal,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    all_equal,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    all_equal,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NARC'=size('VARIABLES')-1],
    []).

ctr_eval(
    all_equal,
    [checker(all_equal_c),reformulation(all_equal_r)]).
```

```
ctr_contractible(all_equal,[],'VARIABLES',any).

ctr_cond_imply(
    all_equal,
    some_equal,
    [size('VARIABLES')>1],
    [],
    id).

ctr_sol(all_equal,2,0,2,3,-).

ctr_sol(all_equal,3,0,3,4,-).

ctr_sol(all_equal,4,0,4,5,-).

ctr_sol(all_equal,5,0,5,6,-).

ctr_sol(all_equal,6,0,6,7,-).

ctr_sol(all_equal,7,0,7,8,-).

ctr_sol(all_equal,8,0,8,9,-).

all_equal_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        all_equal2(VARS).

all_equal_r(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        all_equal1(VARS).

all_equal1([]).

all_equal1([_45435]) :-
        !.

all_equal1([V1,V2|R]) :-
        V1#=V2,
        all_equal1([V2|R]).

all_equal2([V,V|R]) :-
        !,
        all_equal2([V|R]).
```

```
all_equal2([_45435]) :-
        !.

all_equal2([]).
```

B.7 all_equal_except_0

◊ **META-DATA:**

```
ctr_date(all_equal_except_0,['20141009']).

ctr_origin(all_equal_except_0,'Derived from %c',[all_equal]).

ctr_arguments(
    all_equal_except_0,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    all_equal_except_0,
    [required('VARIABLES',var),size('VARIABLES')>0]).

ctr_example(
    all_equal_except_0,
    all_equal_except_0([[var-5],[var-0],[var-5],[var-5]])).

ctr_typical(all_equal_except_0,[size('VARIABLES')>1]).

ctr_typical_model(
    all_equal_except_0,
    [atleast(2,'VARIABLES',0),nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    all_equal_except_0,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int(=\=(0)),=\=,all,dontcare)]).

ctr_eval(
    all_equal_except_0,
    [checker(all_equal_except_0_c),
     automata(all_equal_except_a)]).

ctr_contractible(all_equal_except_0,[],'VARIABLES',any).

all_equal_except_0_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        all_equal_except_01(VARS,0).

all_equal_except_01([],_24106) :-
        !.
```

```
all_equal_except_01([0|R],U) :-
        !,
        all_equal_except_01(R,U).

all_equal_except_01([V|R],0) :-
        !,
        all_equal_except_01(R,V).

all_equal_except_01([V|R],V) :-
        all_equal_except_01(R,V).

all_equal_except_a([]) :-
        !.

all_equal_except_a(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        VARIABLES=[_24122|_24123],
        all_equal_except_signature(VARIABLES,SIGNATURE,VARS),
        automaton(
            VARS,
            VARi,
            SIGNATURE,
            [source(s),source(t),sink(t)],
            [arc(s,0,s),
             arc(s,1,t,[VARi]),
             arc(t,0,t),
             arc(t,1,t,(C#=VARi->[C]))],
            [C],
            [0],
            [_24200]).

all_equal_except_signature([],[],[]).

all_equal_except_signature([[var-VAR]|VARs],[S|Ss],[VAR|Ts]) :-
        VAR#\=0#<=>S,
        all_equal_except_signature(VARs,Ss,Ts).
```

B.8 all_equal_peak

◇ **META-DATA:**

```
ctr_date(all_equal_peak,['20130107']).

ctr_origin(
    all_equal_peak,
    Derived from %c and %c.,
    [peak,all_equal]).

ctr_arguments(
    all_equal_peak,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    all_equal_peak,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    all_equal_peak,
    all_equal_peak(
        [[var-1],
         [var-5],
         [var-5],
         [var-4],
         [var-3],
         [var-5],
         [var-2],
         [var-7]])).

ctr_typical(
    all_equal_peak,
    [size('VARIABLES')>=5,
     range('VARIABLES'^var)>1,
     peak('VARIABLES'^var)>=2]).

ctr_typical_model(all_equal_peak,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    all_equal_peak,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    all_equal_peak,
    [checker(all_equal_peak_c),
```

```
     automaton(all_equal_peak_a),
     automaton_with_signature(all_equal_peak_a_s)]).

ctr_contractible(all_equal_peak,[],'VARIABLES',prefix).

ctr_contractible(all_equal_peak,[],'VARIABLES',suffix).

ctr_cond_imply(
    all_equal_peak,
    some_equal,
    [peak('VARIABLES'^var)>1],
    [],
    id).

ctr_cond_imply(
    all_equal_peak,
    not_all_equal,
    [peak('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(all_equal_peak,2,0,2,9,-).

ctr_sol(all_equal_peak,3,0,3,64,-).

ctr_sol(all_equal_peak,4,0,4,625,-).

ctr_sol(all_equal_peak,5,0,5,7330,-).

ctr_sol(all_equal_peak,6,0,6,93947,-).

ctr_sol(all_equal_peak,7,0,7,1267790,-).

ctr_sol(all_equal_peak,8,0,8,17908059,-).

ctr_sol(all_equal_peak,9,0,9,266201992,-).

all_equal_peak_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        all_equal_peak_c(s,VARS,0).

all_equal_peak_c(s,[V1,V2|R],A) :-
        V1>=V2,
```

```
        !,
        all_equal_peak_c(s,[V2|R],A).

all_equal_peak_c(s,[_34075,V2|R],A) :-
        !,
        all_equal_peak_c(i,[V2|R],A).

all_equal_peak_c(i,[V1,V2|R],A) :-
        V1=<V2,
        !,
        all_equal_peak_c(i,[V2|R],A).

all_equal_peak_c(i,[V1,V2|R],_34074) :-
        !,
        all_equal_peak_c(j,[V2|R],V1).

all_equal_peak_c(j,[V1,V2|R],A) :-
        V1>=V2,
        !,
        all_equal_peak_c(j,[V2|R],A).

all_equal_peak_c(j,[_34075,V2|R],A) :-
        !,
        all_equal_peak_c(k,[V2|R],A).

all_equal_peak_c(k,[V1,V2|R],A) :-
        V1=<V2,
        !,
        all_equal_peak_c(k,[V2|R],A).

all_equal_peak_c(k,[V1,V2|R],V1) :-
        !,
        all_equal_peak_c(j,[V2|R],V1).

all_equal_peak_c(_34069,[_34072],_34071).

ctr_automaton_signature(
    all_equal_peak,
    all_equal_peak_a,
    pair_signature(1,signature)).

all_equal_peak_a(FLAG,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        all_equal_peak_a_s(FLAG,VARIABLES,SIGNATURE).

all_equal_peak_a_s(FLAG,VARIABLES,SIGNATURE) :-
```

```
collection(VARIABLES,[dvar]),
length(VARIABLES,N),
N>0,
pair_first_signature(VARIABLES,VARS),
automaton(
    VARS,
    VARi,
    SIGNATURE,
    [source(s),sink(i),sink(j),sink(k),sink(s)],
    [arc(s,1,s),
     arc(s,2,s),
     arc(s,0,i),
     arc(i,0,i),
     arc(i,1,i),
     arc(i,2,j,[VARi,F]),
     arc(j,1,j),
     arc(j,2,j),
     arc(j,0,k),
     arc(k,0,k),
     arc(k,1,k),
     arc(k,2,j,(Altitude#=VARi->[Altitude,F])),
     arc(k,2,j,(Altitude#\=VARi->[Altitude,0]))],
    [Altitude,F],
    [0,1],
    [_34259,FLAG]).
```

B.9 all_equal_peak_max

◇ Meta-Data:

```
ctr_date(all_equal_peak_max,['20130107']).

ctr_origin(
    all_equal_peak_max,
    Derived from %c and %c.,
    [peak,all_equal]).

ctr_arguments(
    all_equal_peak_max,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    all_equal_peak_max,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    all_equal_peak_max,
    all_equal_peak_max(
        [[var-1],
         [var-5],
         [var-5],
         [var-4],
         [var-3],
         [var-5],
         [var-2],
         [var-5]])).

ctr_typical(
    all_equal_peak_max,
    [size('VARIABLES')>=5,
     range('VARIABLES'^var)>1,
     peak('VARIABLES'^var)>=2]).

ctr_typical_model(all_equal_peak_max,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    all_equal_peak_max,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    all_equal_peak_max,
    [checker(all_equal_peak_max_c),
```

```
     automaton(all_equal_peak_max_a),
     automaton_with_signature(all_equal_peak_max_a_s)]).

ctr_contractible(all_equal_peak_max,[],'VARIABLES',prefix).

ctr_contractible(all_equal_peak_max,[],'VARIABLES',suffix).

ctr_cond_imply(
    all_equal_peak_max,
    some_equal,
    [peak('VARIABLES'^var)>1],
    [],
    id).

ctr_cond_imply(
    all_equal_peak_max,
    not_all_equal,
    [peak('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(all_equal_peak_max,2,0,2,9,-).

ctr_sol(all_equal_peak_max,3,0,3,64,-).

ctr_sol(all_equal_peak_max,4,0,4,605,-).

ctr_sol(all_equal_peak_max,5,0,5,6707,-).

ctr_sol(all_equal_peak_max,6,0,6,81648,-).

ctr_sol(all_equal_peak_max,7,0,7,1065542,-).

ctr_sol(all_equal_peak_max,8,0,8,14829903,-).

all_equal_peak_max_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        last(VARS,LastVAR),
        all_equal_peak_max_c(s,VARS,LastVAR).

all_equal_peak_max_c(s,[V1,V2|R],A) :-
        V1>=V2,
        !,
```

```
        all_equal_peak_max_c(s,[V2|R],A).

all_equal_peak_max_c(s,[_32663,V2|R],A) :-
        !,
        all_equal_peak_max_c(i,[V2|R],A).

all_equal_peak_max_c(i,[V1,V2|R],A) :-
        V1=<V2,
        !,
        all_equal_peak_max_c(i,[V2|R],A).

all_equal_peak_max_c(i,[V1,V2|R],_32662) :-
        !,
        all_equal_peak_max_c(j,[V2|R],V1).

all_equal_peak_max_c(j,[V1,V2|R],A) :-
        V1>=V2,
        !,
        all_equal_peak_max_c(j,[V2|R],A).

all_equal_peak_max_c(j,[V1,V2|R],A) :-
        !,
        A>=V1,
        all_equal_peak_max_c(k,[V2|R],A).

all_equal_peak_max_c(k,[V1,V2|R],A) :-
        V1=<V2,
        !,
        A>=V1,
        all_equal_peak_max_c(k,[V2|R],A).

all_equal_peak_max_c(k,[A,V2|R],A) :-
        !,
        all_equal_peak_max_c(j,[V2|R],A).

all_equal_peak_max_c(_32660,[LastVAR],A) :-
        A>=LastVAR.

ctr_automaton_signature(
    all_equal_peak_max,
    all_equal_peak_max_a,
    pair_signature(1,signature)).

all_equal_peak_max_a(FLAG,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        all_equal_peak_max_a_s(FLAG,VARIABLES,SIGNATURE).
```

```
all_equal_peak_max_a_s(FLAG,VARIABLES,SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        pair_first_last_signature(VARIABLES,VARS,LastVAR),
        automaton(
            VARS,
            VARi,
            SIGNATURE,
            [source(s),sink(i),sink(j),sink(k),sink(s)],
            [arc(s,1,s),
             arc(s,2,s),
             arc(s,0,i),
             arc(i,0,i),
             arc(i,1,i),
             arc(i,2,j,[VARi,F]),
             arc(j,1,j),
             arc(j,2,j),
             arc(j,0,k,(Altitude#>=VARi->[Altitude,F])),
             arc(j,0,k,(Altitude#<VARi->[Altitude,0])),
             arc(k,0,k,(Altitude#>=VARi->[Altitude,F])),
             arc(k,0,k,(Altitude#<VARi->[Altitude,0])),
             arc(k,1,k,(Altitude#>=VARi->[Altitude,F])),
             arc(k,1,k,(Altitude#<VARi->[Altitude,0])),
             arc(k,2,j,(Altitude#=VARi->[Altitude,F])),
             arc(k,2,j,(Altitude#\=VARi->[Altitude,0]))],
            [Altitude,F],
            [LastVAR,1],
            [A,FL]),
        FLAG#<=>FL#/\A#>=LastVAR.
```

B.10 all_equal_valley

◇ **META-DATA:**

```
ctr_date(all_equal_valley,['20130108']).

ctr_origin(
    all_equal_valley,
    Derived from %c and %c.,
    [valley,all_equal]).

ctr_arguments(
    all_equal_valley,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    all_equal_valley,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    all_equal_valley,
    all_equal_valley(
        [[var-1],
         [var-5],
         [var-5],
         [var-4],
         [var-2],
         [var-2],
         [var-6],
         [var-2],
         [var-7]])).

ctr_typical(
    all_equal_valley,
    [size('VARIABLES')>=5,
     range('VARIABLES'^var)>1,
     valley('VARIABLES'^var)>=2]).

ctr_typical_model(all_equal_valley,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    all_equal_valley,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    all_equal_valley,
```

```
    [checker(all_equal_valley_c),
     automaton(all_equal_valley_a),
     automaton_with_signature(all_equal_valley_a_s)]).

ctr_contractible(all_equal_valley,[],'VARIABLES',prefix).

ctr_contractible(all_equal_valley,[],'VARIABLES',suffix).

ctr_cond_imply(
    all_equal_valley,
    some_equal,
    [valley('VARIABLES'^var)>1],
    [],
    id).

ctr_cond_imply(
    all_equal_valley,
    not_all_equal,
    [valley('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(all_equal_valley,2,0,2,9,-).

ctr_sol(all_equal_valley,3,0,3,64,-).

ctr_sol(all_equal_valley,4,0,4,625,-).

ctr_sol(all_equal_valley,5,0,5,7330,-).

ctr_sol(all_equal_valley,6,0,6,93947,-).

ctr_sol(all_equal_valley,7,0,7,1267790,-).

ctr_sol(all_equal_valley,8,0,8,17908059,-).

all_equal_valley_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        all_equal_valley_c(s,VARS,0).

all_equal_valley_c(s,[V1,V2|R],A) :-
        V1=<V2,
        !,
```

```
        all_equal_valley_c(s,[V2|R],A).

all_equal_valley_c(s,[_33896,V2|R],A) :-
        !,
        all_equal_valley_c(i,[V2|R],A).

all_equal_valley_c(i,[V1,V2|R],A) :-
        V1>=V2,
        !,
        all_equal_valley_c(i,[V2|R],A).

all_equal_valley_c(i,[V1,V2|R],_33895) :-
        !,
        all_equal_valley_c(j,[V2|R],V1).

all_equal_valley_c(j,[V1,V2|R],A) :-
        V1=<V2,
        !,
        all_equal_valley_c(j,[V2|R],A).

all_equal_valley_c(j,[_33896,V2|R],A) :-
        !,
        all_equal_valley_c(k,[V2|R],A).

all_equal_valley_c(k,[V1,V2|R],A) :-
        V1>=V2,
        !,
        all_equal_valley_c(k,[V2|R],A).

all_equal_valley_c(k,[V1,V2|R],V1) :-
        !,
        all_equal_valley_c(j,[V2|R],V1).

all_equal_valley_c(_33893,[_33896],_33895) :-
        !.

all_equal_valley_c(_33890,[],_33892).

ctr_automaton_signature(
    all_equal_valley,
    all_equal_valley_a,
    pair_signature(1,signature)).

all_equal_valley_a(FLAG,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        all_equal_valley_a_s(FLAG,VARIABLES,SIGNATURE).
```

```
all_equal_valley_a_s(FLAG,VARIABLES,SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        pair_first_signature(VARIABLES,VARS),
        automaton(
            VARS,
            VARi,
            SIGNATURE,
            [source(s),sink(i),sink(j),sink(k),sink(s)],
            [arc(s,1,s),
             arc(s,0,s),
             arc(s,2,i),
             arc(i,2,i),
             arc(i,1,i),
             arc(i,0,j,[VARi,F]),
             arc(j,1,j),
             arc(j,0,j),
             arc(j,2,k),
             arc(k,2,k),
             arc(k,1,k),
             arc(k,0,j,(Altitude#=VARi->[Altitude,F])),
             arc(k,0,j,(Altitude#\=VARi->[Altitude,0]))],
            [Altitude,F],
            [0,1],
            [_34080,FLAG]).
```

B.11 all_equal_valley_min

◇ **META-DATA:**

```
ctr_date(all_equal_valley_min,['20130108']).

ctr_origin(
    all_equal_valley_min,
    Derived from %c and %c.,
    [valley,all_equal]).

ctr_arguments(
    all_equal_valley_min,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    all_equal_valley_min,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    all_equal_valley_min,
    all_equal_valley_min(
        [[var-2],
         [var-5],
         [var-5],
         [var-4],
         [var-2],
         [var-2],
         [var-6],
         [var-2],
         [var-7]])).

ctr_typical(
    all_equal_valley_min,
    [size('VARIABLES')>=5,
     range('VARIABLES'^var)>1,
     valley('VARIABLES'^var)>=2]).

ctr_typical_model(
    all_equal_valley_min,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    all_equal_valley_min,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).
```

```
ctr_eval(
    all_equal_valley_min,
    [checker(all_equal_valley_min_c),
     automaton(all_equal_valley_min_a),
     automaton_with_signature(all_equal_valley_min_a_s)]).

ctr_contractible(all_equal_valley_min,[],'VARIABLES',prefix).

ctr_contractible(all_equal_valley_min,[],'VARIABLES',suffix).

ctr_cond_imply(
    all_equal_valley_min,
    some_equal,
    [valley('VARIABLES'^var)>1],
    [],
    id).

ctr_cond_imply(
    all_equal_valley_min,
    not_all_equal,
    [valley('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(all_equal_valley_min,2,0,2,9,-).

ctr_sol(all_equal_valley_min,3,0,3,64,-).

ctr_sol(all_equal_valley_min,4,0,4,605,-).

ctr_sol(all_equal_valley_min,5,0,5,6707,-).

ctr_sol(all_equal_valley_min,6,0,6,81648,-).

ctr_sol(all_equal_valley_min,7,0,7,1065542,-).

ctr_sol(all_equal_valley_min,8,0,8,14829903,-).

all_equal_valley_min_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        last(VARS,LastVAR),
        all_equal_valley_min_c(s,VARS,LastVAR).
```

```
all_equal_valley_min_c(s,[V1,V2|R],A) :-
        V1=<V2,
        !,
        all_equal_valley_min_c(s,[V2|R],A).

all_equal_valley_min_c(s,[_32725,V2|R],A) :-
        !,
        all_equal_valley_min_c(i,[V2|R],A).

all_equal_valley_min_c(i,[V1,V2|R],A) :-
        V1>=V2,
        !,
        all_equal_valley_min_c(i,[V2|R],A).

all_equal_valley_min_c(i,[V1,V2|R],_32724) :-
        !,
        all_equal_valley_min_c(j,[V2|R],V1).

all_equal_valley_min_c(j,[V1,V2|R],A) :-
        V1=<V2,
        !,
        all_equal_valley_min_c(j,[V2|R],A).

all_equal_valley_min_c(j,[V1,V2|R],A) :-
        !,
        A=<V1,
        all_equal_valley_min_c(k,[V2|R],A).

all_equal_valley_min_c(k,[V1,V2|R],A) :-
        V1>=V2,
        !,
        A=<V1,
        all_equal_valley_min_c(k,[V2|R],A).

all_equal_valley_min_c(k,[A,V2|R],A) :-
        !,
        all_equal_valley_min_c(j,[V2|R],A).

all_equal_valley_min_c(_32722,[LastVAR],A) :-
        A=<LastVAR.

ctr_automaton_signature(
    all_equal_valley_min,
    all_equal_valley_min_a,
    pair_signature(1,signature)).
```

```
all_equal_valley_min_a(FLAG,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        all_equal_valley_min_a_s(FLAG,VARIABLES,SIGNATURE).

all_equal_valley_min_a_s(FLAG,VARIABLES,SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        pair_first_last_signature(VARIABLES,VARS,LastVAR),
        automaton(
            VARS,
            VARi,
            SIGNATURE,
            [source(s),sink(i),sink(j),sink(k),sink(s)],
            [arc(s,1,s),
             arc(s,0,s),
             arc(s,2,i),
             arc(i,2,i),
             arc(i,1,i),
             arc(i,0,j,[VARi,F]),
             arc(j,1,j),
             arc(j,0,j),
             arc(j,2,k,(Altitude#=<VARi->[Altitude,F])),
             arc(j,2,k,(Altitude#>VARi->[Altitude,0])),
             arc(k,2,k,(Altitude#=<VARi->[Altitude,F])),
             arc(k,2,k,(Altitude#>VARi->[Altitude,0])),
             arc(k,1,k,(Altitude#=<VARi->[Altitude,F])),
             arc(k,1,k,(Altitude#>VARi->[Altitude,0])),
             arc(k,0,j,(Altitude#=VARi->[Altitude,F])),
             arc(k,0,j,(Altitude#\=VARi->[Altitude,0]))],
            [Altitude,F],
            [LastVAR,1],
            [A,FL]),
        FLAG#<=>FL#/\A#=<LastVAR.
```

B.12 all_incomparable

♢ META-DATA:

```
ctr_date(all_incomparable,['20120202']).

ctr_origin(
    all_incomparable,
    Inspired by incomparable rectangles.,
    []).

ctr_synonyms(all_incomparable,[all_incomparables]).

ctr_types(all_incomparable,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    all_incomparable,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    all_incomparable,
    [required('VECTOR',var),
     size('VECTOR')>=1,
     required('VECTORS',vec),
     size('VECTORS')>=1,
     same_size('VECTORS',vec)]).

ctr_example(
    all_incomparable,
    all_incomparable(
        [[vec-[[var-1],[var-18]]],
         [vec-[[var-2],[var-16]]],
         [vec-[[var-3],[var-13]]],
         [vec-[[var-4],[var-11]]],
         [vec-[[var-5],[var-10]]],
         [vec-[[var-6],[var-9]]],
         [vec-[[var-7],[var-7]]]])).

ctr_typical(
    all_incomparable,
    [size('VECTOR')>1,
     size('VECTORS')>1,
     size('VECTORS')>size('VECTOR')]).

ctr_exchangeable(all_incomparable,[items('VECTORS',all)]).
```

```
ctr_graph(
    all_incomparable,
    ['VECTORS'],
    2,
    ['CLIQUE'(=\=)>>collection(vectors1,vectors2)],
    [incomparable(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')*size('VECTORS')-size('VECTORS')],
    ['NO_LOOP','SYMMETRIC']).

ctr_eval(
    all_incomparable,
    [reformulation(all_incomparable_r),
     checker(all_incomparable_c)]).

ctr_contractible(all_incomparable,[],'VECTORS',any).

ctr_cond_imply(
    all_incomparable,
    k_disjoint,
    [size('VECTOR')=2],
    [],
    same).

ctr_cond_imply(
    all_incomparable,
    twin,
    [size('VECTOR')=2],
    [],
    same).

all_incomparable_r(VECTORS) :-
        collection(VECTORS,[col([dvar])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        VECTS=[VEC|_53928],
        length(VEC,N),
        N>=1,
        all_incomparable(VECTS,N).

all_incomparable([_53895],_53894) :-
        !.

all_incomparable(_53893,1) :-
        !,
        fail.
```

```
all_incomparable(VECTS,_53894) :-
        all_incomparable1(VECTS,NEW_PVARS),
        flattern(VECTS,VARS),
        when(ground(VARS),once(labeling([],NEW_PVARS))).

all_incomparable1([],[]).

all_incomparable1([_53892],[]).

all_incomparable1([V,W|R],P) :-
        all_incomparable2([W|R],V,P1),
        all_incomparable1([W|R],P2),
        append(P1,P2,P).

all_incomparable2([],_53891,[]).

all_incomparable2([V|R],U,P) :-
        all_incomparable3(U,V,PUV),
        all_incomparable2(R,U,PR),
        append(PUV,PR,P).

all_incomparable3(U,V,PUV) :-
        length(U,N),
        length(V,N),
        N>1,
        length(PU,N),
        length(PV,N),
        domain(PU,1,N),
        domain(PV,1,N),
        get_minimum(U,MinU),
        get_maximum(U,MaxU),
        get_minimum(V,MinV),
        get_maximum(V,MaxV),
        length(SU,N),
        length(SV,N),
        domain(SU,MinU,MaxU),
        domain(SV,MinV,MaxV),
        sorting(U,PU,SU),
        sorting(V,PV,SV),
        all_incomparable4(SU,SV,Cond1),
        all_incomparable4(SV,SU,Cond2),
        call(Cond1),
        call(Cond2),
        append(PU,PV,PUV).

all_incomparable4([],[],0).
```

```
all_incomparable4([U|R],[V|S],U#>V#\/T) :-
        all_incomparable4(R,S,T).

all_incomparable_c(VECTORS) :-
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        VECTS=[VEC|_53928],
        length(VEC,N),
        N>=1,
        all_incomparable_check(VECTS,N).

all_incomparable_check([_53895],_53894) :-
        !.

all_incomparable_check(_53893,1) :-
        !,
        fail.

all_incomparable_check(VECTS,_53894) :-
        all_incomparablec1(VECTS).

all_incomparablec1([]) :-
        !.

all_incomparablec1([_53894]) :-
        !.

all_incomparablec1([V|R]) :-
        all_incomparablec2(R,V),
        all_incomparablec1(R).

all_incomparablec2([],_53894) :-
        !.

all_incomparablec2([V|R],U) :-
        incomparablec(U,V),
        all_incomparablec2(R,U).
```

B.13 all_min_dist

◇ **META-DATA:**

```
ctr_date(all_min_dist,['20050508','20060803']).

ctr_origin(all_min_dist,'\\cite{Regin97}',[]).

ctr_synonyms(all_min_dist,[minimum_distance,inter_distance]).

ctr_arguments(
    all_min_dist,
    ['MINDIST'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    all_min_dist,
    ['MINDIST'>0,
     size('VARIABLES')<2#\/'MINDIST'<range('VARIABLES'^var),
     required('VARIABLES',var)]).

ctr_example(
    all_min_dist,
    all_min_dist(2,[[var-5],[var-1],[var-9],[var-3]])).

ctr_typical(all_min_dist,['MINDIST'>1,size('VARIABLES')>1]).

ctr_exchangeable(
    all_min_dist,
    [vals(['MINDIST'],int(>=(1)),>,dontcare,dontcare),
     items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     translate(['VARIABLES'^var])]).

ctr_graph(
    all_min_dist,
    ['VARIABLES'],
    2,
    ['CLIQUE'(<)>>collection(variables1,variables2)],
    [abs(variables1^var-variables2^var)>='MINDIST'],
    ['NARC'=size('VARIABLES')*(size('VARIABLES')-1)/2],
    ['ACYCLIC','NO_LOOP']).

ctr_eval(
    all_min_dist,
    [checker(all_min_dist_c),reformulation(all_min_dist_r)]).
```

```
ctr_contractible(all_min_dist,[],'VARIABLES',any).

ctr_cond_imply(
    all_min_dist,
    soft_all_equal_max_var,
    [],
    ['N'>=size('VARIABLES')-1],
    [none,'VARIABLES']).

ctr_sol(all_min_dist,2,0,2,8,[1-6,2-2]).

ctr_sol(all_min_dist,3,0,3,24,[1-24]).

ctr_sol(all_min_dist,4,0,4,120,[1-120]).

ctr_sol(all_min_dist,5,0,5,720,[1-720]).

ctr_sol(all_min_dist,6,0,6,5040,[1-5040]).

ctr_sol(all_min_dist,7,0,7,40320,[1-40320]).

ctr_sol(all_min_dist,8,0,8,362880,[1-362880]).

all_min_dist_c(MINDIST,[]) :-
        !,
        integer(MINDIST),
        MINDIST>0.

all_min_dist_c(MINDIST,VARIABLES) :-
        integer(MINDIST),
        MINDIST>0,
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        (   VARS=[_60493,_60495|_60496] ->
            samsort(VARS,SVARS),
            all_dist_geq_mindist(SVARS,MINDIST)
        ;   true
        ).

all_dist_geq_mindist([V1,V2|R],MINDIST) :-
        !,
        Dist is V2-V1,
        Dist>=MINDIST,
        all_dist_geq_mindist([V2|R],MINDIST).

all_dist_geq_mindist(_60448,_60449).
```

```
all_min_dist_r(MINDIST,[]) :-
        !,
        integer(MINDIST),
        MINDIST>0.

all_min_dist_r(MINDIST,VARIABLES) :-
        integer(MINDIST),
        MINDIST>0,
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        (   N>1 ->
            list_dvar_range(VARS,RANGE),
            MINDIST#<RANGE,
            all_min_dist1(VARIABLES,MINDIST)
        ;   true
        ).

all_min_dist1([],_60449).

all_min_dist1([[_60458-VAR1]|R],MINDIST) :-
        all_min_dist2(R,VAR1,MINDIST),
        all_min_dist1(R,MINDIST).

all_min_dist2([],_60449,_60450).

all_min_dist2([[_60459-VAR2]|R],VAR1,MINDIST) :-
        abs(VAR1-VAR2)#>=MINDIST,
        all_min_dist2(R,VAR1,MINDIST).
```

B.14 alldifferent

◊ **META-DATA:**

```
ctr_date(
    alldifferent,
    [20000128,
     20030820,
     20040530,
     20060803,
     20081227,
     20090521]).

ctr_origin(alldifferent,'\\cite{Lauriere78}',[]).

ctr_synonyms(
    alldifferent,
    [alldiff,
     alldistinct,
     distinct,
     bound_alldifferent,
     bound_alldiff,
     bound_distinct,
     rel]).

ctr_arguments(alldifferent,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(alldifferent,[required('VARIABLES',var)]).

ctr_example(
    alldifferent,
    alldifferent([[var-5],[var-1],[var-9],[var-3]])).

ctr_typical(alldifferent,[size('VARIABLES')>2]).

ctr_exchangeable(
    alldifferent,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    alldifferent,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
```

```
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).

ctr_eval(
    alldifferent,
    [checker(alldifferent_c),
     builtin(alldifferent_b),
     reformulation(alldifferent_r1),
     reformulation(alldifferent_r2)]).

ctr_contractible(alldifferent,[],'VARIABLES',any).

ctr_cond_imply(alldifferent,lex_alldifferent,[],[],int_to_col).

ctr_cond_imply(
    alldifferent,
    soft_alldifferent_ctr,
    [],
    [],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent,
    balance,
    [],
    ['BALANCE'=0],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent,
    soft_all_equal_max_var,
    [],
    ['N'<size('VARIABLES')],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent,
    soft_all_equal_min_var,
    [],
    ['N'>size('VARIABLES')],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent,
    change,
    [],
```

```
    ['NCHANGE'=size('VARIABLES')-1,in_list('CTR',[=\=])],
    [none,'VARIABLES',none]).

ctr_cond_imply(
    alldifferent,
    circular_change,
    [],
    ['NCHANGE'=size('VARIABLES'),in_list('CTR',[=\=])],
    [none,'VARIABLES',none]).

ctr_cond_imply(
    alldifferent,
    longest_change,
    [],
    ['SIZE'=size('VARIABLES'),in_list('CTR',[=\=])],
    [none,'VARIABLES',none]).

ctr_cond_imply(
    alldifferent,
    length_first_sequence,
    [size('VARIABLES')>0],
    ['LEN'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent,
    length_last_sequence,
    [size('VARIABLES')>0],
    ['LEN'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent,
    min_nvalue,
    [size('VARIABLES')>0],
    ['MIN'=1],
    [none,'VARIABLES']).

ctr_sol(alldifferent,2,0,2,6,-).

ctr_sol(alldifferent,3,0,3,24,-).

ctr_sol(alldifferent,4,0,4,120,-).

ctr_sol(alldifferent,5,0,5,720,-).
```

```
ctr_sol(alldifferent,6,0,6,5040,-).

ctr_sol(alldifferent,7,0,7,40320,-).

ctr_sol(alldifferent,8,0,8,362880,-).

ctr_sol(alldifferent,9,0,9,3628800,-).

ctr_sol(alldifferent,10,0,10,39916800,-).

alldifferent_c([V,V|_95468]) :-
        !,
        fail.

alldifferent_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        sort(VARS,SVARS),
        length(VARS,N),
        length(SVARS,N).

alldifferent_b(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        all_distinct(VARS).

alldifferent_r1(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        get_minimum(VARS,MIN),
        get_maximum(VARS,MAX),
        length(VARS,N),
        length(L,N),
        domain(L,MIN,MAX),
        gen_collection(L,var,SORTED_VARIABLES),
        eval(sort(VARIABLES,SORTED_VARIABLES)),
        eval(strictly_increasing(SORTED_VARIABLES)).

alldifferent_r2(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        get_minimum(VARS,MIN),
        get_maximum(VARS,MAX),
        alldifferent_r20(MIN,MAX,VARS).

alldifferent_r20(L,MAX,_95464) :-
```

```
        L>MAX,
        !.

alldifferent_r20(L,MAX,VARS) :-
        alldifferent_r21(L,MAX,VARS),
        L1 is L+1,
        alldifferent_r20(L1,MAX,VARS).

alldifferent_r21(L,U,_95464) :-
        L>U,
        !.

alldifferent_r21(L,U,VARS) :-
        alldifferent_r22(VARS,L,U,T),
        S is U-L+1,
        call(T#=<S),
        U1 is U-1,
        alldifferent_r21(L,U1,VARS).

alldifferent_r22([],_95463,_95464,0) :-
        !.

alldifferent_r22([Vi|R],L,U,Bilu+S) :-
        Vi in L..U#<=>Bilu,
        alldifferent_r22(R,L,U,S).
```

B.15 alldifferent_between_sets

♢ META-DATA:

```
ctr_date(
    alldifferent_between_sets,
    ['20030820','20051008','20060803']).

ctr_origin(alldifferent_between_sets,'ILOG',[]).

ctr_synonyms(
    alldifferent_between_sets,
    [all_null_intersect,
     alldiff_between_sets,
     alldistinct_between_sets,
     alldiff_on_sets,
     alldistinct_on_sets,
     alldifferent_on_sets]).

ctr_arguments(
    alldifferent_between_sets,
    ['VARIABLES'-collection(var-svar)]).

ctr_restrictions(
    alldifferent_between_sets,
    [required('VARIABLES',var)]).

ctr_example(
    alldifferent_between_sets,
    alldifferent_between_sets(
        [[var-{3,5}],[var-{}],[var-{3}],[var-{3,5,7}]])).

ctr_typical(alldifferent_between_sets,[size('VARIABLES')>2]).

ctr_exchangeable(
    alldifferent_between_sets,
    [items('VARIABLES',all)]).

ctr_graph(
    alldifferent_between_sets,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [eq_set(variables1^var,variables2^var)],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).
```

```
ctr_contractible(alldifferent_between_sets,[],'VARIABLES',any).
```

B.16 alldifferent_consecutive_values

♢ META-DATA:

```
ctr_date(alldifferent_consecutive_values,['20080618']).

ctr_origin(
    alldifferent_consecutive_values,
    Derived from %c.,
    [alldifferent]).

ctr_arguments(
    alldifferent_consecutive_values,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_consecutive_values,
    [required('VARIABLES',var),alldifferent('VARIABLES')]).

ctr_example(
    alldifferent_consecutive_values,
    alldifferent_consecutive_values(
        [[var-5],[var-4],[var-3],[var-6]])).

ctr_typical(
    alldifferent_consecutive_values,
    [size('VARIABLES')>2]).

ctr_exchangeable(
    alldifferent_consecutive_values,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     translate(['VARIABLES'^var])]).

ctr_graph(
    alldifferent_consecutive_values,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['TRUE'],
    ['RANGE'('VARIABLES',var)=size('VARIABLES')-1],
    []).

ctr_eval(
    alldifferent_consecutive_values,
    [checker(alldifferent_consecutive_values_c),
```

```
     reformulation(alldifferent_consecutive_values_r)]).

ctr_cond_imply(
    alldifferent_consecutive_values,
    among_diff_0,
    [minval('VARIABLES'^var)=<0,maxval('VARIABLES'^var)>=0],
    ['NVAR'=size('VARIABLES')-1],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    among_diff_0,
    [minval('VARIABLES'^var)>0],
    ['NVAR'=size('VARIABLES')],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    among_diff_0,
    [maxval('VARIABLES'^var)<0],
    ['NVAR'=size('VARIABLES')],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    balance,
    [],
    ['BALANCE'=0],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    length_first_sequence,
    [size('VARIABLES')>0],
    ['LEN'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    length_last_sequence,
    [size('VARIABLES')>0],
    ['LEN'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
```

```
    max_n,
    [],
    ['MAX'=maxval('VARIABLES'^var)-'RANK'],
    [none,none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    min_n,
    [],
    ['MIN'=minval('VARIABLES'^var)+'RANK'],
    [none,none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    min_nvalue,
    [size('VARIABLES')>0],
    ['MIN'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    ninterval,
    [minval('VARIABLES'^var)=0],
    [NVAL=
     (size('VARIABLES')+'SIZE_INTERVAL'-1)/'SIZE_INTERVAL'],
    [none,'VARIABLES',none]).

ctr_cond_imply(
    alldifferent_consecutive_values,
    range_ctr,
    [],
    [in_list('CTR',[=<]),'R'=size('VARIABLES')],
    [none,none,'VARIABLES']).

ctr_cond_imply(
    alldifferent_consecutive_values,
    soft_alldifferent_ctr,
    [],
    [],
    [none,'VARIABLES']).

ctr_sol(alldifferent_consecutive_values,2,0,2,4,-).

ctr_sol(alldifferent_consecutive_values,3,0,3,12,-).

ctr_sol(alldifferent_consecutive_values,4,0,4,48,-).
```

```
ctr_sol(alldifferent_consecutive_values,5,0,5,240,-).

ctr_sol(alldifferent_consecutive_values,6,0,6,1440,-).

ctr_sol(alldifferent_consecutive_values,7,0,7,10080,-).

ctr_sol(alldifferent_consecutive_values,8,0,8,80640,-).

ctr_sol(alldifferent_consecutive_values,9,0,9,725760,-).

ctr_sol(alldifferent_consecutive_values,10,0,10,7257600,-).

alldifferent_consecutive_values_c([V,V|_41441]) :-
        !,
        fail.

alldifferent_consecutive_values_c([]) :-
        !.

alldifferent_consecutive_values_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        sort(VARS,SVARS),
        length(VARS,N),
        length(SVARS,N),
        min_member(MIN,VARS),
        max_member(MAX,VARS),
        N is MAX-MIN+1.

alldifferent_consecutive_values_r([]) :-
        !.

alldifferent_consecutive_values_r(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        all_different(VARS),
        minimum(MIN,VARS),
        maximum(MAX,VARS),
        length(VARIABLES,N),
        N#=MAX-MIN+1.
```

B.17 alldifferent_cst

◊ META-DATA:

```
ctr_date(alldifferent_cst,['20051104','20060803']).

ctr_origin(alldifferent_cst,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(alldifferent_cst,[alldiff_cst,alldistinct_cst]).

ctr_arguments(
    alldifferent_cst,
    ['VARIABLES'-collection(var-dvar,cst-int)]).

ctr_restrictions(
    alldifferent_cst,
    [required('VARIABLES',[var,cst])]).

ctr_example(
    alldifferent_cst,
    alldifferent_cst(
        [[var-5,cst-0],
         [var-1,cst-1],
         [var-9,cst-0],
         [var-3,cst-4]])).

ctr_typical(
    alldifferent_cst,
    [size('VARIABLES')>2,
     range('VARIABLES'^var)>1,
     2*range('VARIABLES'^var)<3*size('VARIABLES'),
     range('VARIABLES'^cst)>1]).

ctr_exchangeable(
    alldifferent_cst,
    [items('VARIABLES',all),
     attrs('VARIABLES',[[var,cst]]),
     translate(['VARIABLES'^var]),
     translate(['VARIABLES'^cst])]).

ctr_graph(
    alldifferent_cst,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var+variables1^cst=
```

```
     variables2^var+variables2^cst],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).

ctr_eval(
    alldifferent_cst,
    [checker(alldifferent_cst_c),
     reformulation(alldifferent_cst_r)]).

ctr_contractible(alldifferent_cst,[],'VARIABLES',any).

alldifferent_cst_r(VARIABLES) :-
        collection(VARIABLES,[dvar,int]),
        get_attr1(VARIABLES,VARS),
        get_attr2(VARIABLES,CSTS),
        gen_varcst(VARS,CSTS,VARCSTS),
        all_different(VARCSTS).

alldifferent_cst_c(VARIABLES) :-
        collection(VARIABLES,[int,int]),
        get_attr12_sum(VARIABLES,SUMS),
        sort(SUMS,SORTED),
        length(SUMS,N),
        length(SORTED,N).
```

B.18 alldifferent_except_0

◇ META-DATA:

```
ctr_date(
    alldifferent_except_0,
    ['20000128','20030820','20040530','20060803']).

ctr_origin(
    alldifferent_except_0,
    Derived from %c.,
    [alldifferent]).

ctr_synonyms(
    alldifferent_except_0,
    [alldiff_except_0,alldistinct_except_0]).

ctr_arguments(
    alldifferent_except_0,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_except_0,
    [required('VARIABLES',var)]).

ctr_example(
    alldifferent_except_0,
    alldifferent_except_0(
        [[var-5],[var-0],[var-1],[var-9],[var-0],[var-3]])).

ctr_typical(
    alldifferent_except_0,
    [size('VARIABLES')>2,
     atleast(2,'VARIABLES',0),
     range('VARIABLES'^var)>1]).

ctr_typical_model(
    alldifferent_except_0,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    alldifferent_except_0,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int(=\=(0)),=\=,all,dontcare)]).

ctr_graph(
```

```
    alldifferent_except_0,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=\=0,variables1^var=variables2^var],
    ['MAX_NSCC'=<1],
    []).

ctr_eval(
    alldifferent_except_0,
    [reformulation(alldifferent_except_0_r),
     checker(alldifferent_except_0_c),
     density(alldifferent_except_0_d)]).

ctr_contractible(alldifferent_except_0,[],'VARIABLES',any).

ctr_sol(alldifferent_except_0,2,0,2,7,-).

ctr_sol(alldifferent_except_0,3,0,3,34,-).

ctr_sol(alldifferent_except_0,4,0,4,209,-).

ctr_sol(alldifferent_except_0,5,0,5,1546,-).

ctr_sol(alldifferent_except_0,6,0,6,13327,-).

ctr_sol(alldifferent_except_0,7,0,7,130922,-).

ctr_sol(alldifferent_except_0,8,0,8,1441729,-).

alldifferent_except_0_c([[var-V],[var-V]|_53043]) :-
        V=\=0,
        !,
        fail.

alldifferent_except_0_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        filter_zeros(VARS,L),
        sort(L,SL),
        length(L,N),
        length(SL,N).

filter_zeros([],[]) :-
        !.
```

```
filter_zeros([0|R],S) :-
        !,
        filter_zeros(R,S).

filter_zeros([X|R],[X|S]) :-
        filter_zeros(R,S).

alldifferent_except_0_r(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        alldifferent_except_01(VARS).

alldifferent_except_01([]).

alldifferent_except_01([_53033]) :-
        !.

alldifferent_except_01([V1|R]) :-
        alldifferent_except_01(R,V1),
        alldifferent_except_01(R).

alldifferent_except_01([],_53030).

alldifferent_except_01([V2|R],V1) :-
        V1#=0#\/V2#=0#\/V1#\=V2,
        alldifferent_except_01(R,V1).

alldifferent_except_0_d(Density,VARIABLES) :-
        get_attr1(VARIABLES,VARS),
        sort(VARS,SVARS),
        length(VARS,N),
        length(SVARS,S),
        Density is S/N.
```

B.19 alldifferent_interval

◇ **META-DATA:**

```
ctr_date(alldifferent_interval,['20030820','20060803']).

ctr_origin(
    alldifferent_interval,
    Derived from %c.,
    [alldifferent]).

ctr_synonyms(
    alldifferent_interval,
    [alldiff_interval,alldistinct_interval]).

ctr_arguments(
    alldifferent_interval,
    ['VARIABLES'-collection(var-dvar),'SIZE_INTERVAL'-int]).

ctr_restrictions(
    alldifferent_interval,
    [required('VARIABLES',var),'SIZE_INTERVAL'>0]).

ctr_example(
    alldifferent_interval,
    alldifferent_interval([[var-2],[var-4],[var-10]],3)).

ctr_typical(
    alldifferent_interval,
    [size('VARIABLES')>1,
     'SIZE_INTERVAL'>1,
     'SIZE_INTERVAL'<max(3,range('VARIABLES'^var))]).

ctr_exchangeable(
    alldifferent_interval,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         intervals('SIZE_INTERVAL'),
         =,
         all,
         dontcare),
     vals(
         ['VARIABLES'^var],
         intervals('SIZE_INTERVAL'),
         =\=,
```

```
         all,
         in)]).

ctr_graph(
    alldifferent_interval,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var/'SIZE_INTERVAL'=
     variables2^var/'SIZE_INTERVAL'],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).

ctr_eval(
    alldifferent_interval,
    [checker(alldifferent_interval_c),
     reformulation(alldifferent_interval_r),
     density(alldifferent_interval_d)]).

ctr_contractible(alldifferent_interval,[],'VARIABLES',any).

ctr_sol(alldifferent_interval,2,0,2,10,[1-6,2-4]).

ctr_sol(alldifferent_interval,3,0,3,24,[1-24]).

ctr_sol(alldifferent_interval,4,0,4,120,[1-120]).

ctr_sol(alldifferent_interval,5,0,5,720,[1-720]).

ctr_sol(alldifferent_interval,6,0,6,5040,[1-5040]).

ctr_sol(alldifferent_interval,7,0,7,40320,[1-40320]).

ctr_sol(alldifferent_interval,8,0,8,362880,[1-362880]).

alldifferent_interval_c(VARIABLES,SIZE_INTERVAL) :-
        collection(VARIABLES,[int]),
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
        get_attr1(VARIABLES,VARS),
        gen_quotient_fix(VARS,SIZE_INTERVAL,QUOTIENT),
        (   QUOTIENT=[V,V|_53822] ->
            fail
        ;   sort(QUOTIENT,SORTED),
            length(QUOTIENT,N),
            length(SORTED,N)
```

```
        ).

alldifferent_interval_r(VARIABLES,SIZE_INTERVAL) :-
        collection(VARIABLES,[dvar]),
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
        get_attr1(VARIABLES,VARS),
        gen_quotient(VARS,SIZE_INTERVAL,QUOTVARS),
        all_different(QUOTVARS).

alldifferent_interval_d(Density,VARIABLES,SIZE_INTERVAL) :-
        get_attr1(VARIABLES,VARS),
        min_member(Min,VARS),
        max_member(Max,VARS),
        NormalizedMin is Min//SIZE_INTERVAL,
        NormalizedMax is Max//SIZE_INTERVAL,
        Available is NormalizedMax-NormalizedMin+1,
        length(VARS,Needed),
        Density is Needed/Available.
```

B.20 alldifferent_modulo

◇ META-DATA:

```
ctr_date(alldifferent_modulo,['20030820','20060803']).

ctr_origin(
    alldifferent_modulo,
    Derived from %c.,
    [alldifferent]).

ctr_synonyms(
    alldifferent_modulo,
    [alldiff_modulo,alldistinct_modulo]).

ctr_arguments(
    alldifferent_modulo,
    ['VARIABLES'-collection(var-dvar),'M'-int]).

ctr_restrictions(
    alldifferent_modulo,
    [required('VARIABLES',var),'M'>0,'M'>=size('VARIABLES')]).

ctr_example(
    alldifferent_modulo,
    alldifferent_modulo([[var-25],[var-1],[var-14],[var-3]],5)).

ctr_typical(alldifferent_modulo,[size('VARIABLES')>2,'M'>1]).

ctr_exchangeable(
    alldifferent_modulo,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],mod('M'),=,all,dontcare),
     vals(['VARIABLES'^var],mod('M'),=\=,all,in)]).

ctr_graph(
    alldifferent_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).

ctr_eval(
    alldifferent_modulo,
```

```
    [reformulation(alldifferent_modulo_r)]).

ctr_contractible(alldifferent_modulo,[],'VARIABLES',any).

ctr_sol(alldifferent_modulo,2,0,2,4,[2-4]).

ctr_sol(alldifferent_modulo,3,0,3,12,[3-12]).

ctr_sol(alldifferent_modulo,4,0,4,48,[4-48]).

ctr_sol(alldifferent_modulo,5,0,5,240,[5-240]).

ctr_sol(alldifferent_modulo,6,0,6,1440,[6-1440]).

ctr_sol(alldifferent_modulo,7,0,7,10080,[7-10080]).

ctr_sol(alldifferent_modulo,8,0,8,80640,[8-80640]).

alldifferent_modulo_r(VARIABLES,M) :-
        collection(VARIABLES,[dvar]),
        integer(M),
        M>0,
        length(VARIABLES,N),
        M>=N,
        get_attr1(VARIABLES,VARS),
        gen_remainder(VARS,M,REMVARS),
        all_different(REMVARS).
```

B.21 alldifferent_on_intersection

◇ META-DATA:

```
ctr_date(alldifferent_on_intersection,['20040530','20060803']).

ctr_origin(
    alldifferent_on_intersection,
    Derived from %c and %c.,
    [common,alldifferent]).

ctr_synonyms(
    alldifferent_on_intersection,
    [alldiff_on_intersection,alldistinct_on_intersection]).

ctr_arguments(
    alldifferent_on_intersection,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_on_intersection,
    [required('VARIABLES1',var),required('VARIABLES2',var)]).

ctr_example(
    alldifferent_on_intersection,
    alldifferent_on_intersection(
        [[var-5],[var-9],[var-1],[var-5]],
        [[var-2],[var-1],[var-6],[var-9],[var-6],[var-2]])).

ctr_typical(
    alldifferent_on_intersection,
    [size('VARIABLES1')>1,size('VARIABLES2')>1]).

ctr_exchangeable(
    alldifferent_on_intersection,
    [args([['VARIABLES1','VARIABLES2']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
         =\=,
         all,
         dontcare)]).
```

```
ctr_graph(
    alldifferent_on_intersection,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['MAX_NCC'=<2],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    alldifferent_on_intersection,
    [reformulation(alldifferent_on_intersection_r)]).

ctr_contractible(
    alldifferent_on_intersection,
    [],
    VARIABLES1,
    any).

ctr_contractible(
    alldifferent_on_intersection,
    [],
    VARIABLES2,
    any).

alldifferent_on_intersection_r(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        alldifferent_on_intersection(VARS1,1,VARS1,VARS2).

alldifferent_on_intersection([],_47379,_47380,_47381).

alldifferent_on_intersection([VAR1|R1],I,VARS1,VARS2) :-
        alldifferent_on_intersection(
            VARS2,
            1,
            VAR1,
            I,
            VARS1,
            VARS2),
        I1 is I+1,
        alldifferent_on_intersection(R1,I1,VARS1,VARS2).

alldifferent_on_intersection(
```

```
    [],
    _47661,
    _47708,
    _47755,
    _47802,
    _47849).

alldifferent_on_intersection([VAR2|R2],J,VAR1,I,VARS1,VARS2) :-
        alldifferent_on_intersection1(VARS1,1,VAR1,I,VAR2,J),
        alldifferent_on_intersection1(VARS2,1,VAR2,J,VAR1,I),
        J1 is J+1,
        alldifferent_on_intersection(R2,J1,VAR1,I,VARS1,VARS2).

alldifferent_on_intersection1(
    [],
    _47661,
    _47708,
    _47755,
    _47802,
    _47849).

alldifferent_on_intersection1([VAR|R],K,VAR1,I,VAR2,J) :-
        K=\=I,
        !,
        VAR1#=VAR2#=>VAR#\=VAR1,
        K1 is K+1,
        alldifferent_on_intersection1(R,K1,VAR1,I,VAR2,J).

alldifferent_on_intersection1([_47387|R],K,VAR1,I,VAR2,J) :-
        K=:=I,
        K1 is K+1,
        alldifferent_on_intersection1(R,K1,VAR1,I,VAR2,J).
```

B.22 alldifferent_partition

◇ **META-DATA:**

```
ctr_date(alldifferent_partition,['20030820','20060803']).

ctr_origin(
    alldifferent_partition,
    Derived from %c.,
    [alldifferent]).

ctr_synonyms(
    alldifferent_partition,
    [alldiff_partition,alldistinct_partition]).

ctr_types(
    alldifferent_partition,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    alldifferent_partition,
    ['VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    alldifferent_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     size('VARIABLES')=<size('PARTITIONS'),
     required('VARIABLES',var),
     size('PARTITIONS')>=2,
     required('PARTITIONS',p)]).

ctr_example(
    alldifferent_partition,
    alldifferent_partition(
        [[var-6],[var-3],[var-4]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(alldifferent_partition,[size('VARIABLES')>2]).

ctr_exchangeable(
    alldifferent_partition,
```

```
    [items('VARIABLES',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['VARIABLES'^var],
         part('PARTITIONS'),
         =,
         all,
         dontcare),
     vals(['VARIABLES'^var],part('PARTITIONS'),=\=,all,in)]).

ctr_graph(
    alldifferent_partition,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).

ctr_eval(
    alldifferent_partition,
    [reformulation(alldifferent_partition_r)]).

ctr_contractible(alldifferent_partition,[],'VARIABLES',any).

alldifferent_partition_r(VARIABLES,PARTITIONS) :-
        collection(VARIABLES,[dvar]),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        get_attr1(VARIABLES,VARS),
        get_col_attr1(PARTITIONS,1,PVALS),
        flattern(PVALS,VALS),
        all_different(VALS),
        length(VARIABLES,N),
        length(PARTITIONS,M),
        N=<M,
        M>1,
        length(PVALS,LPVALS),
        get_partition_var(VARS,PVALS,PVARS,LPVALS,0),
        all_different(PVARS).
```

B.23 alldifferent_same_value

◇ **META-DATA:**

```
ctr_date(
    alldifferent_same_value,
    ['20000128','20030820','20060803']).

ctr_origin(
    alldifferent_same_value,
    Derived from %c.,
    [alldifferent]).

ctr_synonyms(
    alldifferent_same_value,
    [alldiff_same_value,alldistinct_same_value]).

ctr_arguments(
    alldifferent_same_value,
    ['NSAME'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    alldifferent_same_value,
    ['NSAME'>=0,
     'NSAME'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    alldifferent_same_value,
    alldifferent_same_value(
        2,
        [[var-7],[var-3],[var-1],[var-5]],
        [[var-1],[var-3],[var-1],[var-7]])).

ctr_typical(
    alldifferent_same_value,
    ['NSAME'<size('VARIABLES1'),size('VARIABLES1')>2]).

ctr_exchangeable(
    alldifferent_same_value,
    [items_sync('VARIABLES1','VARIABLES2',all),
     vals(
```

```
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    alldifferent_same_value,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'('CLIQUE','LOOP',=)>>
     collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['MAX_NSCC'=<1,'NARC_NO_LOOP'='NSAME'],
    []).

ctr_eval(
    alldifferent_same_value,
    [reformulation(alldifferent_same_value_r)]).

ctr_functional_dependency(alldifferent_same_value,1,[2,3]).

ctr_cond_imply(
    alldifferent_same_value,
    differ_from_exactly_k_pos,
    [2*'NSAME'=size('VARIABLES1')],
    [],
    id).

alldifferent_same_value_r(NSAME,VARIABLES1,VARIABLES2) :-
        check_type(dvar,NSAME),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        NSAME#>=0,
        NSAME#=<N1,
        N1=N2,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        all_different(VARS1),
        alldifferent_same_value1(VARS1,VARS2,SUMBOOLS),
        call(NSAME#=SUMBOOLS).

alldifferent_same_value1([],[],0).
```

```
alldifferent_same_value1([V1|R1],[V2|R2],B+R) :-
        V1#=V2#<=>B,
        alldifferent_same_value1(R1,R2,R).
```

B.24 allperm

◊ META-DATA:

```
ctr_date(allperm,['20031008','20070916']).

ctr_origin(
    allperm,
    \cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02},
    []).

ctr_synonyms(allperm,[all_perm,all_permutations]).

ctr_types(allperm,['VECTOR'-collection(var-dvar)]).

ctr_arguments(allperm,['MATRIX'-collection(vec-'VECTOR')]).

ctr_restrictions(
    allperm,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('MATRIX',vec),
     same_size('MATRIX',vec)]).

ctr_example(
    allperm,
    allperm(
        [[vec-[[var-1],[var-2],[var-3]]],
         [vec-[[var-3],[var-1],[var-2]]]])).

ctr_typical(allperm,[size('VECTOR')>1,size('MATRIX')>1]).

ctr_exchangeable(allperm,[translate(['MATRIX'^vec^var])]).

ctr_graph(
    allperm,
    ['MATRIX'],
    2,
    ['CLIQUE'(<)>>collection(matrix1,matrix2)],
    [matrix1^key=1,
     matrix2^key>1,
     lex_lesseq_allperm(matrix1^vec,matrix2^vec)],
    ['NARC'=size('MATRIX')-1],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(allperm,[checker(allperm_c),reformulation(allperm_r)]).
```

```
ctr_contractible(allperm,[],'MATRIX'^vec,suffix).

allperm_c(MATRIX) :-
        collection(MATRIX,[col([int])]),
        same_size(MATRIX),
        get_attr11(MATRIX,MAT),
        MAT=[FIRST_ROW|R],
        allperm_c1(R,FIRST_ROW).

allperm_c1([],_42120) :-
        !.

allperm_c1([CUR_ROW|R],FIRST_ROW) :-
        create_pairs(CUR_ROW,PAIRS),
        keysort(PAIRS,SORTED_ROW),
        remove_key_from_collection(SORTED_ROW,SORTED),
        lex_lesseq_c1(FIRST_ROW,SORTED),
        allperm_c1(R,FIRST_ROW).

allperm_r(MATRIX) :-
        collection(MATRIX,[col([dvar])]),
        same_size(MATRIX),
        MATRIX=[[vec-F]|R],
        allperm_sorted(R,S),
        allperm_order(S,F).

allperm_sorted([],[]).

allperm_sorted([[vec-X]|R],[S|T]) :-
        get_attr1(X,L),
        get_minimum(L,MIN),
        get_maximum(L,MAX),
        length(X,LX),
        length(Y,LX),
        domain(Y,MIN,MAX),
        gen_collection(Y,var,S),
        eval(sort(X,S)),
        allperm_sorted(R,T).

allperm_order([],_42117).

allperm_order([X|R],F) :-
        eval(lex_lesseq(F,X)),
        allperm_order(R,F).
```

B.25 among

◇ **META-DATA:**

```
ctr_date(among,['20000128','20030820','20040807','20060804']).

ctr_origin(among,'\\cite{BeldiceanuContejean94}',[]).

ctr_synonyms(among,[between,count]).

ctr_arguments(
    among,
    ['NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    among,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    among,
    among(
        3,
        [[var-4],[var-5],[var-5],[var-4],[var-1]],
        [[val-1],[val-5],[val-8]])).

ctr_typical(
    among,
    ['NVAR'>0,
     'NVAR'<size('VARIABLES'),
     size('VARIABLES')>1,
     size('VALUES')>1,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    among,
    [items('VARIABLES',all),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var],
         comp('VALUES'^val),
```

```
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    among,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var in 'VALUES'],
    ['NARC'='NVAR'],
    []).

ctr_eval(
    among,
    [checker(among_c),
     reformulation(among_r),
     automaton(among_a)]).

ctr_pure_functional_dependency(among,[]).

ctr_functional_dependency(among,1,[2,3]).

ctr_contractible(among,['NVAR'=0],'VARIABLES',any).

ctr_contractible(
    among,
    ['NVAR'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_aggregate(among,[],[+,union,sunion]).

among_c(N,VARIABLES,VALUES) :-
        integer(N),
        N>=0,
        collection(VALUES,[int]),
        get_attr1(VALUES,VALS),
        all_different(VALS),
        among_c1(VARIABLES,N,VALS).

among_c1([[var-V]|R],N,VALS) :-
        !,
        integer(V),
        (   memberchk(V,VALS) ->
            N1 is N-1,
```

```
            N1>=0
        ;   N1 is N
        ),
        among_c1(R,N1,VALS).

among_c1([],0,_57522).

among_r(NVAR,VARIABLES,VALUES) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int]),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        length(VARIABLES,N),
        NVAR#>=0,
        NVAR#=<N,
        all_different(VALS),
        among1(VARS,VALS,SUM_BVARS),
        call(NVAR#=SUM_BVARS).

among1([],_57521,0).

among1([V|R],VALS,B+S) :-
        build_or_var_in_values(VALS,V,OR),
        call(OR#<=>B),
        among1(R,VALS,S).

among_a(FLAG,NVAR,VARIABLES,VALUES) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int]),
        get_attr1(VALUES,LIST_VALUES),
        length(VARIABLES,N),
        NVAR#>=0,
        NVAR#=<N,
        all_different(LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        among_signature(VARIABLES,SIGNATURE,SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _59826,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
```

```
            [COUNT]),
        COUNT#=NVAR#<=>FLAG.

among_signature([],[],_57522).

among_signature([[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
        VAR in_set SET_OF_VALUES#<=>S,
        among_signature(VARs,Ss,SET_OF_VALUES).
```

B.26 among_diff_0

◇ **META-DATA:**

```
ctr_date(among_diff_0,['20040807','20060804']).

ctr_origin(
    among_diff_0,
    Used in the automaton of %c.,
    [nvalue]).

ctr_arguments(
    among_diff_0,
    ['NVAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    among_diff_0,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    among_diff_0,
    [among_diff_0(
         3,
         [[var-0],[var-5],[var-5],[var-0],[var-1]]),
     among_diff_0(
         0,
         [[var-0],[var-0],[var-0],[var-0],[var-0]]),
     among_diff_0(
         1,
         [[var-0],[var-0],[var-0],[var-6],[var-0]])]).

ctr_typical(
    among_diff_0,
    ['NVAR'>0,
     'NVAR'<size('VARIABLES'),
     size('VARIABLES')>1,
     atleast(1,'VARIABLES',0),
     2*among_diff_0('VARIABLES'^var)>size('VARIABLES')]).

ctr_typical_model(among_diff_0,[atleast(2,'VARIABLES',0)]).

ctr_exchangeable(
    among_diff_0,
    [items('VARIABLES',all),
```

```
     vals(
         ['VARIABLES'^var],
         int(=\=(0)),
         =\=,
         dontcare,
         dontcare)]).

ctr_graph(
    among_diff_0,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var=\=0],
    ['NARC'='NVAR'],
    []).

ctr_eval(
    among_diff_0,
    [checker(among_diff_0_c),
     reformulation(among_diff_0_r),
     automaton(among_diff_0_a)]).

ctr_pure_functional_dependency(among_diff_0,[]).

ctr_functional_dependency(among_diff_0,1,[2]).

ctr_contractible(among_diff_0,['NVAR'=0],'VARIABLES',any).

ctr_contractible(
    among_diff_0,
    ['NVAR'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_aggregate(among_diff_0,[],[+,union]).

ctr_sol(among_diff_0,2,0,2,9,[0-1,1-4,2-4]).

ctr_sol(among_diff_0,3,0,3,64,[0-1,1-9,2-27,3-27]).

ctr_sol(among_diff_0,4,0,4,625,[0-1,1-16,2-96,3-256,4-256]).

ctr_sol(
    among_diff_0,
    5,
    0,
```

```
    5,
    7776,
    [0-1,1-25,2-250,3-1250,4-3125,5-3125]).

ctr_sol(
    among_diff_0,
    6,
    0,
    6,
    117649,
    [0-1,1-36,2-540,3-4320,4-19440,5-46656,6-46656]).

ctr_sol(
    among_diff_0,
    7,
    0,
    7,
    2097152,
    [0-1,
     1-49,
     2-1029,
     3-12005,
     4-84035,
     5-352947,
     6-823543,
     7-823543]).

ctr_sol(
    among_diff_0,
    8,
    0,
    8,
    43046721,
    [0-1,
     1-64,
     2-1792,
     3-28672,
     4-286720,
     5-1835008,
     6-7340032,
     7-16777216,
     8-16777216]).

among_diff_0_c(NVAR,VARIABLES) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[int]),
```

```
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        NVAR#>=0,
        NVAR#=<N,
        among_diff_0_c(VARS,0,NVAR).

among_diff_0_c([V|R],C,NVAR) :-
        !,
        (   V=0 ->
            among_diff_0_c(R,C,NVAR)
        ;   C1 is C+1,
            among_diff_0_c(R,C1,NVAR)
        ).

among_diff_0_c([],NVAR,NVAR).

among_diff_0_counters_check([V|R],C,[D|S]) :-
        !,
        (   V=0 ->
            D=C
        ;   D is C+1
        ),
        among_diff_0_counters_check(R,D,S).

among_diff_0_counters_check([],_61462,[]).

among_diff_0_r(NVAR,VARIABLES) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        NVAR#>=0,
        NVAR#=<N,
        among_diff_01(VARS,SUM_BVARS),
        call(NVAR#=SUM_BVARS).

among_diff_01([],0).

among_diff_01([V|R],B+S) :-
        V#\=0#<=>B,
        among_diff_01(R,S).

among_diff_0_a(FLAG,NVAR,VARIABLES) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
```

```
        NVAR#>=0,
        NVAR#=<N,
        among_diff_0_signature(VARIABLES,SIGNATURE),
        automaton(
            SIGNATURE,
            _63099,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NVAR#<=>FLAG.

among_diff_0_signature([],[]).

among_diff_0_signature([[var-VAR]|VARs],[S|Ss]) :-
        VAR#\=0#<=>S,
        among_diff_0_signature(VARs,Ss).
```

B.27 among_interval

◇ **META-DATA:**

```
ctr_date(among_interval,['20030820','20040530','20060804']).

ctr_origin(among_interval,'Derived from %c.',[among]).

ctr_arguments(
    among_interval,
    ['NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'LOW'-int,
     'UP'-int]).

ctr_restrictions(
    among_interval,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var),
     'LOW'=<'UP']).

ctr_example(
    among_interval,
    among_interval(
        3,
        [[var-4],[var-5],[var-8],[var-4],[var-1]],
        3,
        5)).

ctr_typical(
    among_interval,
    ['NVAR'>0,
     'NVAR'<size('VARIABLES'),
     size('VARIABLES')>1,
     'LOW'<'UP',
     'LOW'=<maxval('VARIABLES'^var),
     'UP'>=minval('VARIABLES'^var)]).

ctr_exchangeable(
    among_interval,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         comp('LOW' in 'UP'),
         =,
```

```
         dontcare,
         dontcare)]).

ctr_graph(
    among_interval,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['LOW'=<variables^var,variables^var=<'UP'],
    ['NARC'='NVAR'],
    []).

ctr_eval(
    among_interval,
    [reformulation(among_interval_r),
     automaton(among_interval_a)]).

ctr_pure_functional_dependency(among_interval,[]).

ctr_functional_dependency(among_interval,1,[2,3,4]).

ctr_contractible(among_interval,['NVAR'=0],'VARIABLES',any).

ctr_contractible(
    among_interval,
    ['NVAR'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_aggregate(among_interval,[],[+,union,id,id]).

among_interval_r(NVAR,VARIABLES,LOW,UP) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        integer(LOW),
        integer(UP),
        length(VARIABLES,N),
        NVAR#>=0,
        NVAR#=<N,
        LOW=<UP,
        among_interval1(VARS,SUM_BVARS,LOW,UP),
        call(NVAR#=SUM_BVARS).

among_interval1([],0,_42060,_42061).
```

```
among_interval1([V|R],B+S,LOW,UP) :-
        V#>=LOW#/\V#=<UP#<=>B,
        among_interval1(R,S,LOW,UP).

among_interval_a(FLAG,NVAR,VARIABLES,LOW,UP) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        integer(LOW),
        integer(UP),
        length(VARIABLES,N),
        NVAR#>=0,
        NVAR#=<N,
        LOW=<UP,
        among_interval_signature(VARIABLES,SIGNATURE,LOW,UP),
        automaton(
            SIGNATURE,
            _44223,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NVAR#<=>FLAG.

among_interval_signature([],[],_42060,_42061).

among_interval_signature([[var-VAR]|VARs],[S|Ss],LOW,UP) :-
        LOW#=<VAR#/\VAR#=<UP#<=>S,
        among_interval_signature(VARs,Ss,LOW,UP).
```

B.28 among_low_up

◇ **META-DATA:**

```
ctr_date(among_low_up,['20030820','20040530','20060804']).

ctr_origin(among_low_up,'\\cite{BeldiceanuContejean94}',[]).

ctr_arguments(
    among_low_up,
    ['LOW'-int,
     'UP'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    among_low_up,
    ['LOW'>=0,
     'LOW'=<size('VARIABLES'),
     'UP'>=0,
     'UP'=<size('VARIABLES'),
     'UP'>='LOW',
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    among_low_up,
    among_low_up(
        1,
        2,
        [[var-9],[var-2],[var-4],[var-5]],
        [[val-0],[val-2],[val-4],[val-6],[val-8]])).

ctr_typical(
    among_low_up,
    ['LOW'<size('VARIABLES'),
     'UP'>0,
     'LOW'<'UP',
     size('VARIABLES')>1,
     size('VALUES')>1,
     size('VARIABLES')>size('VALUES'),
     'LOW'>0#\/'UP'<size('VARIABLES')]).

ctr_exchangeable(
    among_low_up,
```

```
    [items('VARIABLES',all),
     items('VALUES',all),
     vals(['LOW'],int(>=(0)),>,dontcare,dontcare),
     vals(
         ['UP'],
         int(=<(size('VARIABLES'))),
         <,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES'^var],
         comp('VALUES'^val),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    among_low_up,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['NARC'>='LOW','NARC'=<'UP'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    among_low_up,
    [reformulation(among_low_up_r),automaton(among_low_up_a)]).

ctr_contractible(among_low_up,['UP'=0],'VARIABLES',any).

ctr_contractible(
    among_low_up,
    ['UP'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_aggregate(among_low_up,[],[+,+,union,sunion]).

ctr_cond_imply(
    among_low_up,
    among_low_up,
    [distinct('VARIABLES',var)],
    [],
    ['LOW','UP','VALUES','VARIABLES']).
```

```
among_low_up_r(LOW,UP,VARIABLES,VALUES) :-
        integer(LOW),
        integer(UP),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int]),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        length(VARIABLES,N),
        LOW>=0,
        LOW=<N,
        UP>=0,
        UP=<N,
        UP>=LOW,
        all_different(VALS),
        among_low_up1(VARS,VALS,SUM_BVARS),
        call(LOW#=<SUM_BVARS),
        call(UP#>=SUM_BVARS).

among_low_up1([],_49694,0).

among_low_up1([V|R],VALS,B+S) :-
        build_or_var_in_values(VALS,V,OR),
        call(OR#<=>B),
        among_low_up1(R,VALS,S).

among_low_up_a(FLAG,LOW,UP,VARIABLES,VALUES) :-
        integer(LOW),
        integer(UP),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int]),
        get_attr1(VALUES,LIST_VALUES),
        length(VARIABLES,N),
        LOW>=0,
        LOW=<N,
        UP>=0,
        UP=<N,
        UP>=LOW,
        all_different(LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        among_low_up_signature(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        NVAR in LOW..UP,
        automaton(
            SIGNATURE,
```

```
            _53178,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NVAR#<=>FLAG.

among_low_up_signature([],[],_49695).

among_low_up_signature([[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
        VAR in_set SET_OF_VALUES#<=>S,
        among_low_up_signature(VARs,Ss,SET_OF_VALUES).
```

B.29 among_modulo

◇ META-DATA:

```
ctr_date(among_modulo,['20030820','20040530','20060804']).

ctr_origin(among_modulo,'Derived from %c.',[among]).

ctr_arguments(
    among_modulo,
    ['NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'REMAINDER'-int,
     'QUOTIENT'-int]).

ctr_restrictions(
    among_modulo,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var),
     'REMAINDER'>=0,
     'REMAINDER'<'QUOTIENT',
     'QUOTIENT'>0]).

ctr_example(
    among_modulo,
    among_modulo(
        3,
        [[var-4],[var-5],[var-8],[var-4],[var-1]],
        0,
        2)).

ctr_typical(
    among_modulo,
    ['NVAR'>0,
     'NVAR'<size('VARIABLES'),
     size('VARIABLES')>1,
     'QUOTIENT'>1,
     'QUOTIENT'<maxval('VARIABLES'^var)]).

ctr_exchangeable(
    among_modulo,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         comp('QUOTIENT' mod 'REMAINDER'),
```

```
        =,
        dontcare,
        dontcare)]).

ctr_graph(
    among_modulo,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var mod 'QUOTIENT'='REMAINDER'],
    ['NARC'='NVAR'],
    []).

ctr_eval(
    among_modulo,
    [reformulation(among_modulo_r),automaton(among_modulo_a)]).

ctr_pure_functional_dependency(among_modulo,[]).

ctr_functional_dependency(among_modulo,1,[2,3,4]).

ctr_contractible(among_modulo,['NVAR'=0],'VARIABLES',any).

ctr_contractible(
    among_modulo,
    ['NVAR'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_aggregate(among_modulo,[],[+,union,id,id]).

among_modulo_r(NVAR,VARIABLES,REMAINDER,QUOTIENT) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        integer(REMAINDER),
        integer(QUOTIENT),
        NVAR#>=0,
        NVAR#=<N,
        REMAINDER>=0,
        REMAINDER<QUOTIENT,
        QUOTIENT>0,
        gen_remainder(VARS,QUOTIENT,REMVARS),
        among_modulo1(REMVARS,REMAINDER,SUM_BVARS),
        call(NVAR#=SUM_BVARS).
```

```
among_modulo1([],_41767,0).

among_modulo1([V|R],REMAINDER,B+S) :-
        V#=REMAINDER#<=>B,
        among_modulo1(R,REMAINDER,S).

among_modulo_a(FLAG,NVAR,VARIABLES,REMAINDER,QUOTIENT) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        integer(REMAINDER),
        integer(QUOTIENT),
        length(VARIABLES,N),
        NVAR#>=0,
        NVAR#=<N,
        REMAINDER>=0,
        REMAINDER<QUOTIENT,
        QUOTIENT>0,
        among_modulo_signature(
            VARIABLES,
            SIGNATURE,
            REMAINDER,
            QUOTIENT),
        automaton(
            SIGNATURE,
            _44609,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NVAR#<=>FLAG.

among_modulo_signature([],[],_41768,_41769).

among_modulo_signature(
    [[var-VAR]|VARs],
    [S|Ss],
    REMAINDER,
    QUOTIENT) :-
        VAR mod QUOTIENT#=REMAINDER#<=>S,
        among_modulo_signature(VARs,Ss,REMAINDER,QUOTIENT).
```

B.30 among_seq

◇ **META-DATA:**

```
ctr_date(among_seq,['20000128','20030820']).

ctr_origin(among_seq,'\\cite{BeldiceanuContejean94}',[]).

ctr_synonyms(among_seq,[sequence]).

ctr_arguments(
    among_seq,
    ['LOW'-int,
     'UP'-int,
     'SEQ'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    among_seq,
    ['LOW'>=0,
     'LOW'=<size('VARIABLES'),
     'UP'>='LOW',
     'SEQ'>0,
     'SEQ'>='LOW',
     'SEQ'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    among_seq,
    among_seq(
        1,
        2,
        4,
        [[var-9],
         [var-2],
         [var-4],
         [var-5],
         [var-5],
         [var-7],
         [var-2]],
        [[val-0],[val-2],[val-4],[val-6],[val-8]])).

ctr_typical(
```

```
    among_seq,
    ['LOW'<'SEQ',
     'UP'>0,
     'SEQ'>1,
     'SEQ'<size('VARIABLES'),
     size('VARIABLES')>1,
     size('VALUES')>0,
     size('VARIABLES')>size('VALUES'),
     'LOW'>0#\/'UP'<'SEQ']).

ctr_exchangeable(
    among_seq,
    [items('VARIABLES',reverse),
     items('VALUES',all),
     vals(['LOW'],int(>=(0)),>,dontcare,dontcare),
     vals(['UP'],int(=<('SEQ')),<,dontcare,dontcare),
     vals(
         ['VARIABLES'^var],
         comp('VALUES'^val),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    among_seq,
    ['VARIABLES'],
    SEQ,
    ['PATH'>>collection],
    [among_low_up('LOW','UP',collection,'VALUES')],
    ['NARC'=size('VARIABLES')-'SEQ'+1],
    []).

ctr_eval(
    among_seq,
    [checker(among_seq_c),reformulation(among_seq_r)]).

ctr_contractible(among_seq,['UP'=0],'VARIABLES',any).

ctr_contractible(among_seq,['SEQ'=1],'VARIABLES',any).

ctr_contractible(among_seq,[],'VARIABLES',prefix).

ctr_contractible(among_seq,[],'VARIABLES',suffix).

among_seq_r(LOW,UP,SEQ,VARIABLES,VALUES) :-
        integer(LOW),
```

```
        integer(UP),
        integer(SEQ),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int]),
        get_attr1(VALUES,VALS),
        length(VARIABLES,N),
        LOW>=0,
        LOW=<N,
        SEQ>0,
        SEQ>=LOW,
        SEQ=<N,
        all_different(VALS),
        among_seq1(LOW,UP,SEQ,VARIABLES,VALUES).

among_seq1(_LOW,_UP,SEQ,VARIABLES,_VALUES) :-
        length(VARIABLES,N),
        N<SEQ,
        !.

among_seq1(LOW,UP,SEQ,VARIABLES,VALUES) :-
        length(VARIABLES,N),
        N>=SEQ,
        among_seq2(VARIABLES,SEQ,SEQVARIABLES),
        eval(among_low_up(LOW,UP,SEQVARIABLES,VALUES)),
        VARIABLES=[_38047|RVARIABLES],
        among_seq1(LOW,UP,SEQ,RVARIABLES,VALUES).

among_seq2(_37993,0,[]) :-
        !.

among_seq2([VAR|VARS],SEQ,[VAR|RVARS]) :-
        SEQ>0,
        SEQ1 is SEQ-1,
        among_seq2(VARS,SEQ1,RVARS).

among_seq_c(LOW,UP,SEQ,VARIABLES,VALUES) :-
        integer(LOW),
        integer(UP),
        integer(SEQ),
        collection(VARIABLES,[int]),
        collection(VALUES,[int]),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        length(VARIABLES,N),
        LOW>=0,
        LOW=<N,
```

```
        SEQ>0,
        SEQ>=LOW,
        SEQ=<N,
        sort(VALS,SVALS),
        length(VALS,M),
        length(SVALS,M),
        among_seq_check(
            VARS,
            SEQ,
            CONTINUATION,
            CONTINUATION,
            0,
            LOW,
            UP,
            VALS).

among_seq_check(
    [],
    _38282,
    _38329,
    _38376,
    _38423,
    _38470,
    _38517,
    _38564) :-
        !.

among_seq_check([V|R],I,BUFFER,CONT,SUM,LOW,UP,VALS) :-
        (   memberchk(V,VALS) ->
            IN=1,
            SUM1 is SUM+1
        ;   IN=0,
            SUM1 is SUM
        ),
        CONT=[IN|CONT1],
        I1 is I-1,
        (   I1<0 ->
            BUFFER=[OUT|RBUFFER],
            SUM2 is SUM1-OUT,
            SUM2>=LOW,
            SUM2=<UP,
            among_seq_check(
                R,
                I1,
                RBUFFER,
                CONT1,
```

```
        SUM2,
        LOW,
        UP,
        VALS)
;   (   I1=0 ->
        SUM1>=LOW,
        SUM1=<UP
    ;   true
    ),
    among_seq_check(R,I1,BUFFER,CONT1,SUM1,LOW,UP,VALS)
).
```

B.31 among_var

◊ META-DATA:

```
ctr_date(among_var,['20090418']).

ctr_origin(among_var,'Generalisation of %c',[among]).

ctr_arguments(
    among_var,
    ['NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-dvar)]).

ctr_restrictions(
    among_var,
    ['NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val)]).

ctr_example(
    among_var,
    among_var(
        3,
        [[var-4],[var-5],[var-5],[var-4],[var-1]],
        [[val-1],[val-5],[val-8],[val-1]])).

ctr_typical(
    among_var,
    [size('VARIABLES')>1,
     size('VALUES')>1,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    among_var,
    [items('VARIABLES',all),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare),
     vals(
         ['VARIABLES'^var],
```

```
          comp('VALUES'^val),
          =,
          dontcare,
          dontcare)]).

ctr_graph(
    among_var,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['NSOURCE'='NVAR'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(among_var,[reformulation(among_var_r)]).

ctr_pure_functional_dependency(among_var,[]).

ctr_functional_dependency(among_var,1,[2,3]).

ctr_contractible(among_var,['NVAR'=0],'VARIABLES',any).

ctr_contractible(
    among_var,
    ['NVAR'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_aggregate(among_var,[],[+,union,union]).

among_var_r(NVAR,VARIABLES,[]) :-
        !,
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        NVAR=0.

among_var_r(NVAR,VARIABLES,VALUES) :-
        check_type(dvar,NVAR),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[dvar]),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        length(VARIABLES,N),
        NVAR#>=0,
        NVAR#=<N,
        among_var1(VARS,VALS,SUM_BVARS),
```

```
        call(NVAR#=SUM_BVARS).

among_var1([],_45749,0).

among_var1([V|R],VALS,B+S) :-
        build_or_var_in_values(VALS,V,OR),
        call(OR#<=>B),
        among_var1(R,VALS,S).
```

B.32 and

◇ **META-DATA:**

```
ctr_date(and,['20051226']).

ctr_origin(and,'Logic',[]).

ctr_synonyms(and,[rel]).

ctr_arguments(
    and,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    and,
    ['VAR'>=0,
     'VAR'=<1,
     size('VARIABLES')>=2,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    and,
    [and(0,[[var-0],[var-0]]),
     and(0,[[var-0],[var-1]]),
     and(0,[[var-1],[var-0]]),
     and(1,[[var-1],[var-1]]),
     and(0,[[var-1],[var-0],[var-1]])]).

ctr_exchangeable(and,[items('VARIABLES',all)]).

ctr_eval(
    and,
    [checker(and_c),reformulation(and_r),automaton(and_a)]).

ctr_pure_functional_dependency(and,[]).

ctr_functional_dependency(and,1,[2]).

ctr_extensible(and,['VAR'=0],'VARIABLES',any).

ctr_aggregate(and,[],[#/\,union]).

ctr_cond_imply(
```

```
    and,
    some_equal,
    [size('VARIABLES')>2],
    [],
    ['VARIABLES']).

ctr_cond_imply(and,nand,['VAR'=0],['VAR'=1],[none,'VARIABLES']).

ctr_cond_imply(and,nand,['VAR'=1],['VAR'=0],[none,'VARIABLES']).

ctr_sol(and,2,0,2,4,[0-3,1-1]).

ctr_sol(and,3,0,3,8,[0-7,1-1]).

ctr_sol(and,4,0,4,16,[0-15,1-1]).

ctr_sol(and,5,0,5,32,[0-31,1-1]).

ctr_sol(and,6,0,6,64,[0-63,1-1]).

ctr_sol(and,7,0,7,128,[0-127,1-1]).

ctr_sol(and,8,0,8,256,[0-255,1-1]).

and_c(VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[int(0,1)]),
        length(VARIABLES,N),
        N>=2,
        get_attr1(VARIABLES,VARS),
        and_c1(VARS,VAR).

and_c1([],1) :-
        !.

and_c1([0|_47649],0) :-
        !.

and_c1([_47648|R],VAR) :-
        and_c1(R,VAR).

and_counters_check([V|R],init,[-|S]) :-
        !,
        and_counters_check(R,V,S).

and_counters_check([0|R],_47647,[0|S]) :-
```

```
        !,
        and_counters_check(R,0,S).

and_counters_check([1|R],C,[C|S]) :-
        !,
        and_counters_check(R,C,S).

and_counters_check([],_47644,[]).

and_r(VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,N),
        N>=2,
        get_attr1(VARIABLES,VARS),
        and1(VARS,ANDVARS),
        call(ANDVARS#<=>VAR).

and1([VAR],VAR) :-
        !.

and1([VAR|VARS],VAR#/\S) :-
        and1(VARS,S).

and_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,N),
        N>=2,
        get_attr1(VARIABLES,LIST),
        append([VAR],LIST,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _49670,
            LIST_VARIABLES,
            [source(s),sink(k),sink(j)],
            [arc(s,0,i),
             arc(s,1,j),
             arc(i,0,k),
             arc(i,1,i),
             arc(k,0,k),
             arc(k,1,k),
             arc(j,1,j)],
            [],
            [],
```

```
     []),
automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.33 arith

◊ META-DATA:

```
ctr_date(arith,['20040814','20060804']).

ctr_origin(
    arith,
    Used in the definition of several automata,
    []).

ctr_synonyms(arith,[rel]).

ctr_arguments(
    arith,
    ['VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'VALUE'-int]).

ctr_restrictions(
    arith,
    [required('VARIABLES',var),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    arith,
    arith([[var-4],[var-5],[var-7],[var-4],[var-5]],<,9)).

ctr_typical(arith,[size('VARIABLES')>1,in_list('RELOP',[=])]).

ctr_exchangeable(
    arith,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,dontcare,in)]).

ctr_graph(
    arith,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['RELOP'(variables^var,'VALUE')],
    ['NARC'=size('VARIABLES')],
    []).

ctr_eval(arith,[reformulation(arith_r),automaton(arith_a)]).
```

```
ctr_contractible(arith,[],'VARIABLES',any).

ctr_cond_imply(
    arith,
    range_ctr,
    [in_list('RELOP',[<]),minval('VARIABLES'^var)>=0],
    [in_list('CTR',[<])],
    id).

arith_r(VARIABLES,RELOP,VALUE) :-
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        integer(VALUE),
        get_attr1(VARIABLES,VARS),
        arith1(VARS,RELOP,VALUE).

arith1([],_48072,_48073).

arith1([VAR|RVARS],RELOP,VALUE) :-
        call_term_relop_value(VAR,RELOP,VALUE),
        arith1(RVARS,RELOP,VALUE).

arith_a(FLAG,VARIABLES,RELOP,VALUE) :-
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        integer(VALUE),
        arith_signature(VARIABLES,SIGNATURE,RELOP,VALUE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _49585,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

arith_signature([],[],_48073,_48074).

arith_signature([[var-VAR]|VARs],[S|Ss],=,VALUE) :-
        !,
        VAR#=VALUE#<=>S,
        arith_signature(VARs,Ss,=,VALUE).
```

```
arith_signature([[var-VAR]|VARs],[S|Ss],=\=,VALUE) :-
        !,
        VAR#\=VALUE#<=>S,
        arith_signature(VARs,Ss,=\=,VALUE).

arith_signature([[var-VAR]|VARs],[S|Ss],<,VALUE) :-
        !,
        VAR#<VALUE#<=>S,
        arith_signature(VARs,Ss,<,VALUE).

arith_signature([[var-VAR]|VARs],[S|Ss],>=,VALUE) :-
        !,
        VAR#>=VALUE#<=>S,
        arith_signature(VARs,Ss,>=,VALUE).

arith_signature([[var-VAR]|VARs],[S|Ss],>,VALUE) :-
        !,
        VAR#>VALUE#<=>S,
        arith_signature(VARs,Ss,>,VALUE).

arith_signature([[var-VAR]|VARs],[S|Ss],=<,VALUE) :-
        VAR#=<VALUE#<=>S,
        arith_signature(VARs,Ss,=<,VALUE).
```

B.34 arith_or

♢ META-DATA:

```
ctr_date(arith_or,['20040814','20060804']).

ctr_origin(
    arith_or,
    Used in the definition of several automata,
    []).

ctr_arguments(
    arith_or,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'RELOP'-atom,
     'VALUE'-int]).

ctr_restrictions(
    arith_or,
    [required('VARIABLES1',var),
     required('VARIABLES2',var),
     size('VARIABLES1')=size('VARIABLES2'),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    arith_or,
    arith_or(
        [[var-0],[var-1],[var-0],[var-0],[var-1]],
        [[var-0],[var-0],[var-0],[var-1],[var-0]],
        =,
        0)).

ctr_typical(
    arith_or,
    [size('VARIABLES1')>0,in_list('RELOP',[=])]).

ctr_exchangeable(
    arith_or,
    [args([['VARIABLES1','VARIABLES2'],['RELOP'],['VALUE']]),
     items_sync('VARIABLES1','VARIABLES2',all)]).

ctr_graph(
    arith_or,
    ['VARIABLES1','VARIABLES2'],
    2,
```

```
    ['PRODUCT'(=)>>collection(variables1,variables2)],
    ['RELOP'(variables1^var,'VALUE')#\/
     'RELOP'(variables2^var,'VALUE')],
    ['NARC'=size('VARIABLES1')],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    arith_or,
    [reformulation(arith_or_r),automaton(arith_or_a)]).

ctr_contractible(arith_or,[],['VARIABLES1','VARIABLES2'],any).

arith_or_r(VARIABLES1,VARIABLES2,RELOP,VALUE) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        integer(VALUE),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1=N2,
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        arith_or1(VARS1,VARS2,RELOP,VALUE).

arith_or1([],[],_45716,_45717).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],=,VALUE) :-
        !,
        VAR1#=VALUE#\/VAR2#=VALUE,
        arith_or1(RVAR1,RVAR2,=,VALUE).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],=\=,VALUE) :-
        !,
        VAR1#\=VALUE#\/VAR2#\=VALUE,
        arith_or1(RVAR1,RVAR2,=\=,VALUE).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],<,VALUE) :-
        !,
        VAR1#<VALUE#\/VAR2#<VALUE,
        arith_or1(RVAR1,RVAR2,[VAR2|RVAR2],<,VALUE).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],>=,VALUE) :-
        !,
        VAR1#>=VALUE#\/VAR2#>=VALUE,
        arith_or1(RVAR1,RVAR2,[VAR2|RVAR2],>=,VALUE).
```

```
arith_or1([VAR1|RVAR1],[VAR2|RVAR2],>,VALUE) :-
        !,
        VAR1#>VALUE#\/VAR2#>VALUE,
        arith_or1(RVAR1,RVAR2,[VAR2|RVAR2],>,VALUE).

arith_or1([VAR1|RVAR1],[VAR2|RVAR2],=<,VALUE) :-
        VAR1#=<VALUE#\/VAR2#=<VALUE,
        arith_or1(RVAR1,RVAR2,=<,VALUE).

arith_or_a(FLAG,VARIABLES1,VARIABLES2,RELOP,VALUE) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        integer(VALUE),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1=N2,
        arith_or_signature(
            VARIABLES1,
            VARIABLES2,
            SIGNATURE,
            RELOP,
            VALUE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _48305,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

arith_or_signature([],[],[],_45717,_45718).

arith_or_signature(
    [[var-VAR1]|VAR1s],
    [[var-VAR2]|VAR2s],
    [S|Ss],
    =,
    VALUE) :-
        !,
        VAR1#=VALUE#\/VAR2#=VALUE#<=>S,
```

```
        arith_or_signature(VAR1s,VAR2s,Ss,=,VALUE).

arith_or_signature(
    [[var-VAR1]|VAR1s],
    [[var-VAR2]|VAR2s],
    [S|Ss],
    =\=,
    VALUE) :-
        !,
        VAR1#\=VALUE#\/VAR2#\=VALUE#<=>S,
        arith_or_signature(VAR1s,VAR2s,Ss,=\=,VALUE).

arith_or_signature(
    [[var-VAR1]|VAR1s],
    [[var-VAR2]|VAR2s],
    [S|Ss],
    <,
    VALUE) :-
        !,
        VAR1#<VALUE#\/VAR2#<VALUE#<=>S,
        arith_or_signature(VAR1s,VAR2s,Ss,<,VALUE).

arith_or_signature(
    [[var-VAR1]|VAR1s],
    [[var-VAR2]|VAR2s],
    [S|Ss],
    >=,
    VALUE) :-
        !,
        VAR1#>=VALUE#\/VAR2#>=VALUE#<=>S,
        arith_or_signature(VAR1s,VAR2s,Ss,>=,VALUE).

arith_or_signature(
    [[var-VAR1]|VAR1s],
    [[var-VAR2]|VAR2s],
    [S|Ss],
    >,
    VALUE) :-
        !,
        VAR1#>VALUE#\/VAR2#>VALUE#<=>S,
        arith_or_signature(VAR1s,VAR2s,Ss,>,VALUE).

arith_or_signature(
    [[var-VAR1]|VAR1s],
    [[var-VAR2]|VAR2s],
    [S|Ss],
```

```
=<,
VALUE) :-
    VAR1#=<VALUE#\/VAR2#=<VALUE#<=>S,
    arith_or_signature(VAR1s,VAR2s,Ss,=<,VALUE).
```

B.35 arith_sliding

◇ META-DATA:

```
ctr_date(arith_sliding,['20040814']).

ctr_origin(
    arith_sliding,
    Used in the definition of some automaton,
    []).

ctr_arguments(
    arith_sliding,
    ['VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'VALUE'-int]).

ctr_restrictions(
    arith_sliding,
    [required('VARIABLES',var),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    arith_sliding,
    arith_sliding(
        [[var-0],
         [var-0],
         [var-1],
         [var-2],
         [var-0],
         [var-0],
         [var- -3]],
        <,
        4)).

ctr_typical(
    arith_sliding,
    [size('VARIABLES')>1,in_list('RELOP',[<,>=,>,=<])]).

ctr_graph(
    arith_sliding,
    ['VARIABLES'],
    *,
    ['PATH_1'>>collection],
    [arith(collection,'RELOP','VALUE')],
    ['NARC'=size('VARIABLES')],
```

```
    []).

ctr_eval(
    arith_sliding,
    [reformulation(arith_sliding_r),
     automaton(arith_sliding_a)]).

ctr_contractible(
    arith_sliding,
    [in_list('RELOP',[<,=<]),minval('VARIABLES'^var)>=0],
    VARIABLES,
    any).

ctr_contractible(arith_sliding,[],'VARIABLES',suffix).

arith_sliding_r(VARIABLES,RELOP,VALUE) :-
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        integer(VALUE),
        get_attr1(VARIABLES,VARS),
        reverse(VARS,RVARS),
        arith_sliding1(RVARS,RELOP,VALUE).

arith_sliding1([],_29732,_29733).

arith_sliding1([VAR|RVARS],RELOP,VALUE) :-
        arith_sliding2([VAR|RVARS],SUM),
        call_term_relop_value(SUM,RELOP,VALUE),
        arith_sliding1(RVARS,RELOP,VALUE).

arith_sliding2([],0).

arith_sliding2([VAR|RVARS],VAR+R) :-
        arith_sliding2(RVARS,R).

arith_sliding_a(FLAG,VARIABLES,=,VALUE) :-
        !,
        collection(VARIABLES,[dvar]),
        integer(VALUE),
        length(VARIABLES,N),
        length(SIGNATURE,N),
        domain(SIGNATURE,0,0),
        arith_sliding_signature(VARIABLES,VARS,SIGNATURE),
        automaton(
            VARS,
            VAR,
```

```
            SIGNATURE,
            [source(s),sink(s),sink(t)],
            [arc(s,0,t,[T,C+VAR]),
             arc(t,0,t,(C#=VALUE->[T,C+VAR])),
             arc(t,0,t,(C#\=VALUE->[0,C+VAR]))],
            [T,C],
            [1,0],
            [T1,C1]),
        T1#=1#/\C1#=VALUE#<=>FLAG.

arith_sliding_a(FLAG,VARIABLES,=\=,VALUE) :-
        !,
        collection(VARIABLES,[dvar]),
        integer(VALUE),
        length(VARIABLES,N),
        length(SIGNATURE,N),
        domain(SIGNATURE,0,0),
        arith_sliding_signature(VARIABLES,VARS,SIGNATURE),
        automaton(
            VARS,
            VAR,
            SIGNATURE,
            [source(s),sink(s),sink(t)],
            [arc(s,0,t,[T,C+VAR]),
             arc(t,0,t,(C#\=VALUE->[T,C+VAR])),
             arc(t,0,t,(C#=VALUE->[0,C+VAR]))],
            [T,C],
            [1,0],
            [T1,C1]),
        T1#=1#/\C1#\=VALUE#<=>FLAG.

arith_sliding_a(FLAG,VARIABLES,<,VALUE) :-
        !,
        collection(VARIABLES,[dvar]),
        integer(VALUE),
        length(VARIABLES,N),
        length(SIGNATURE,N),
        domain(SIGNATURE,0,0),
        arith_sliding_signature(VARIABLES,VARS,SIGNATURE),
        automaton(
            VARS,
            VAR,
            SIGNATURE,
            [source(s),sink(s),sink(t)],
            [arc(s,0,t,[T,C+VAR]),
             arc(t,0,t,(C#<VALUE->[T,C+VAR])),
```

```
          arc(t,0,t,(C#>=VALUE->[0,C+VAR]))],
         [T,C],
         [1,0],
         [T1,C1]),
    T1#=1#/\C1#<VALUE#<=>FLAG.

arith_sliding_a(FLAG,VARIABLES,>=,VALUE) :-
    !,
    collection(VARIABLES,[dvar]),
    integer(VALUE),
    length(VARIABLES,N),
    length(SIGNATURE,N),
    domain(SIGNATURE,0,0),
    arith_sliding_signature(VARIABLES,VARS,SIGNATURE),
    automaton(
        VARS,
        VAR,
        SIGNATURE,
        [source(s),sink(s),sink(t)],
        [arc(s,0,t,[T,C+VAR]),
         arc(t,0,t,(C#>=VALUE->[T,C+VAR])),
         arc(t,0,t,(C#<VALUE->[0,C+VAR]))],
        [T,C],
        [1,0],
        [T1,C1]),
    T1#=1#/\C1#>=VALUE#<=>FLAG.

arith_sliding_a(FLAG,VARIABLES,>,VALUE) :-
    !,
    collection(VARIABLES,[dvar]),
    integer(VALUE),
    length(VARIABLES,N),
    length(SIGNATURE,N),
    domain(SIGNATURE,0,0),
    arith_sliding_signature(VARIABLES,VARS,SIGNATURE),
    automaton(
        VARS,
        VAR,
        SIGNATURE,
        [source(s),sink(s),sink(t)],
        [arc(s,0,t,[T,C+VAR]),
         arc(t,0,t,(C#>VALUE->[T,C+VAR])),
         arc(t,0,t,(C#=<VALUE->[0,C+VAR]))],
        [T,C],
        [1,0],
        [T1,C1]),
```

```
        T1#=1#/\C1#>VALUE#<=>FLAG.

arith_sliding_a(FLAG,VARIABLES,=<,VALUE) :-
        collection(VARIABLES,[dvar]),
        integer(VALUE),
        length(VARIABLES,N),
        length(SIGNATURE,N),
        domain(SIGNATURE,0,0),
        arith_sliding_signature(VARIABLES,VARS,SIGNATURE),
        automaton(
            VARS,
            VAR,
            SIGNATURE,
            [source(s),sink(s),sink(t)],
            [arc(s,0,t,[T,C+VAR]),
             arc(t,0,t,(C#=<VALUE->[T,C+VAR])),
             arc(t,0,t,(C#>VALUE->[0,C+VAR]))],
            [T,C],
            [1,0],
            [T1,C1]),
        T1#=1#/\C1#=<VALUE#<=>FLAG.

arith_sliding_signature([],[],[]).

arith_sliding_signature([[var-V]|VARs],[V|Vs],[0|Ss]) :-
        arith_sliding_signature(VARs,Vs,Ss).
```

B.36 assign_and_counts

◇ META-DATA:

```
ctr_date(assign_and_counts,['20000128','20030820','20060804']).

ctr_origin(assign_and_counts,'N.~Beldiceanu',[]).

ctr_arguments(
    assign_and_counts,
    ['COLOURS'-collection(val-int),
     'ITEMS'-collection(bin-dvar,colour-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    assign_and_counts,
    [required('COLOURS',val),
     distinct('COLOURS',val),
     required('ITEMS',[bin,colour]),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    assign_and_counts,
    assign_and_counts(
        [[val-4]],
        [[bin-1,colour-4],
         [bin-3,colour-4],
         [bin-1,colour-4],
         [bin-1,colour-5]],
        =<,
        2)).

ctr_typical(
    assign_and_counts,
    [size('COLOURS')>0,
     size('ITEMS')>1,
     range('ITEMS'^bin)>1,
     in_list('RELOP',[<,=<]),
     'LIMIT'>0,
     'LIMIT'<size('ITEMS')]).

ctr_exchangeable(
    assign_and_counts,
    [items('COLOURS',all),
     items('ITEMS',all),
```

```
     vals(['ITEMS'^bin],int,=\=,all,dontcare)]).

ctr_derived_collections(
    assign_and_counts,
    [col('VALUES'-collection(val-int),
         [item(val-'COLOURS'^val)])]).

ctr_graph(
    assign_and_counts,
    ['ITEMS','ITEMS'],
    2,
    ['PRODUCT'>>collection(items1,items2)],
    [items1^bin=items2^bin],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'ITEMS'^colour)])]],
    [counts('VALUES',variables,'RELOP','LIMIT')]).

ctr_eval(
    assign_and_counts,
    [reformulation(assign_and_counts_r)]).

ctr_contractible(
    assign_and_counts,
    [in_list('RELOP',[<,=<])],
    ITEMS,
    any).

ctr_extensible(
    assign_and_counts,
    [in_list('RELOP',[>=,>])],
    ITEMS,
    any).

ctr_application(assign_and_counts,[2]).

assign_and_counts_r(COLOURS,ITEMS,RELOP,LIMIT) :-
        collection(COLOURS,[int]),
        collection(ITEMS,[dvar,dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        get_attr1(COLOURS,COLS),
```

```
        all_different(COLS),
        get_attr1(ITEMS,BINS),
        get_attr2(ITEMS,ITEMSCOLOURS),
        get_minimum(BINS,MINBINS),
        get_maximum(BINS,MAXBINS),
        gen_matrix_bool(MINBINS,MAXBINS,BINS,BMATRIX),
        assign_and_counts1(ITEMSCOLOURS,COLS,CLINE),
        assign_and_counts2(BMATRIX,CLINE,RELOP,LIMIT).

assign_and_counts1([],_49718,[]).

assign_and_counts1([ITEMCOLOUR|RITEMCOLOURS],COLS,[B|R]) :-
        build_or_var_in_values(COLS,ITEMCOLOUR,OR),
        call(OR#<=>B),
        assign_and_counts1(RITEMCOLOURS,COLS,R).

assign_and_counts2([],_49718,_49719,_49720).

assign_and_counts2([BLINE|RBMATRIX],CLINE,RELOP,LIMIT) :-
        assign_and_counts3(BLINE,CLINE,TERM,OR_B),
        call(A#=OR_B),
        call_term_relop_value(TERM,RELOP,A*LIMIT),
        assign_and_counts2(RBMATRIX,CLINE,RELOP,LIMIT).

assign_and_counts3([],[],0,0).

assign_and_counts3([B|RBLINE],[C|RCLINE],B*C+R,BC#\/S) :-
        BC#<=>B#/\C,
        assign_and_counts3(RBLINE,RCLINE,R,S).
```

B.37 assign_and_nvalues

◇ **META-DATA:**

```
ctr_date(
    assign_and_nvalues,
    ['20000128','20030820','20040530','20050321','20060804']).

ctr_origin(
    assign_and_nvalues,
    Derived from %c and %c.,
    [assign_and_counts,nvalues]).

ctr_arguments(
    assign_and_nvalues,
    ['ITEMS'-collection(bin-dvar,value-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    assign_and_nvalues,
    [required('ITEMS',[bin,value]),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    assign_and_nvalues,
    assign_and_nvalues(
        [[bin-2,value-3],
         [bin-1,value-5],
         [bin-2,value-3],
         [bin-2,value-3],
         [bin-2,value-4]],
        =<,
        2)).

ctr_typical(
    assign_and_nvalues,
    [size('ITEMS')>1,
     range('ITEMS'^bin)>1,
     range('ITEMS'^value)>1,
     in_list('RELOP',[<,=<]),
     'LIMIT'>1,
     'LIMIT'<size('ITEMS')]).

ctr_exchangeable(
    assign_and_nvalues,
```

```
    [items('ITEMS',all),
     vals(['ITEMS'^bin],int,=\=,all,dontcare)]).

ctr_graph(
    assign_and_nvalues,
    ['ITEMS','ITEMS'],
    2,
    ['PRODUCT'>>collection(items1,items2)],
    [items1^bin=items2^bin],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'ITEMS'^value)])]],
    [nvalues(variables,'RELOP','LIMIT')]).

ctr_eval(
    assign_and_nvalues,
    [reformulation(assign_and_nvalues_r)]).

ctr_contractible(
    assign_and_nvalues,
    [in_list('RELOP',[<,=<])],
    ITEMS,
    any).

ctr_extensible(
    assign_and_nvalues,
    [in_list('RELOP',[>=,>])],
    ITEMS,
    any).

ctr_application(assign_and_nvalues,[1]).

assign_and_nvalues_r(ITEMS,RELOP,LIMIT) :-
        collection(ITEMS,[dvar,dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        get_attr1(ITEMS,BINS),
        get_attr2(ITEMS,VALUES),
        get_minimum(BINS,MINBINS),
        get_maximum(BINS,MAXBINS),
        gen_matrix_bool(MINBINS,MAXBINS,BINS,BMATRIX),
        get_minimum(VALUES,MINVALUES),
```

```
        JOKER is MINVALUES-1,
        LIM is LIMIT+1,
        assign_and_nvalues1(BMATRIX,VALUES,JOKER,RELOP,LIM).

assign_and_nvalues1([],_49683,_49684,_49685,_49686).

assign_and_nvalues1([BLINE|RBMATRIX],VALUES,JOKER,RELOP,LIM) :-
        assign_and_nvalues2(BLINE,VALUES,JOKER,VALS),
        length(VALS,M),
        N in 0..M,
        nvalue(N,VALS),
        call_term_relop_value(N,RELOP,LIM),
        assign_and_nvalues1(RBMATRIX,VALUES,JOKER,RELOP,LIM).

assign_and_nvalues2([],[],JOKER,[JOKER]).

assign_and_nvalues2([VAR|RVAR],[VAL|RVAL],JOKER,[V|R]) :-
        fd_max(VAL,MAX),
        V in JOKER..MAX,
        VAR#=0#/\V#=JOKER#\/VAR#=1#/\V#=VAL,
        assign_and_nvalues2(RVAR,RVAL,JOKER,R).
```

B.38 atleast

◇ META-DATA:

```
ctr_date(atleast,['20030820','20040807','20060804']).

ctr_origin(atleast,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(atleast,[count]).

ctr_arguments(
    atleast,
    ['N'-int,'VARIABLES'-collection(var-dvar),'VALUE'-int]).

ctr_restrictions(
    atleast,
    ['N'>=0,'N'=<size('VARIABLES'),required('VARIABLES',var)]).

ctr_example(
    atleast,
    atleast(2,[[var-4],[var-2],[var-4],[var-5]],4)).

ctr_typical(
    atleast,
    ['N'>0,'N'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_exchangeable(
    atleast,
    [items('VARIABLES',all),
     vals(['N'],int(>=(0)),>,dontcare,dontcare),
     vals(
         ['VARIABLES'^var],
         comp('VALUE'),
         >=,
         dontcare,
         dontcare)]).

ctr_graph(
    atleast,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE'],
    ['NARC'>='N'],
    []).
```

```
ctr_eval(
    atleast,
    [checker(atleast_c),
     reformulation(atleast_r),
     automaton(atleast_a)]).

ctr_extensible(atleast,[],'VARIABLES',any).

ctr_total_relation(atleast).

atleast_c(N,VARIABLES,VALUE) :-
        integer(N),
        integer(VALUE),
        atleast_c1(VARIABLES,N,VALUE).

atleast_c1([[var-V]|R],N,VALUE) :-
        !,
        integer(V),
        (   V=VALUE ->
            N1 is N-1
        ;   N1 is N
        ),
        atleast_c1(R,N1,VALUE).

atleast_c1([],N,_44879) :-
        N=<0.

atleast_r(N,VARIABLES,VALUE) :-
        integer(N),
        collection(VARIABLES,[dvar]),
        integer(VALUE),
        length(VARIABLES,NVARIABLES),
        N>=0,
        N=<NVARIABLES,
        get_attr1(VARIABLES,VARS),
        atleast1(VARS,VALUE,SUM_BVARS),
        call(SUM_BVARS#>=N).

atleast1([],_44875,0).

atleast1([V|R],VALUE,B+S) :-
        V#=VALUE#<=>B,
        atleast1(R,VALUE,S).

atleast_a(FLAG,N,VARIABLES,VALUE) :-
        integer(N),
```

```
        collection(VARIABLES,[dvar]),
        integer(VALUE),
        length(VARIABLES,M),
        N>=0,
        N=<M,
        atleast_signature(VARIABLES,SIGNATURE,VALUE),
        NVAR in N..M,
        automaton(
            SIGNATURE,
            _46896,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NVAR#<=>FLAG.

atleast_signature([],[],_44876).

atleast_signature([[var-VAR]|VARs],[S|Ss],VALUE) :-
        VAR#=VALUE#<=>S,
        atleast_signature(VARs,Ss,VALUE).
```

B.39 atleast_nvalue

◇ **META-DATA:**

```
ctr_date(atleast_nvalue,['20050618','20060804']).

ctr_origin(atleast_nvalue,'\\cite{Regin95}',[]).

ctr_synonyms(atleast_nvalue,[k_diff]).

ctr_arguments(
    atleast_nvalue,
    ['NVAL'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    atleast_nvalue,
    [required('VARIABLES',var),
     'NVAL'>=0,
     'NVAL'=<size('VARIABLES'),
     'NVAL'=<range('VARIABLES'^var)]).

ctr_example(
    atleast_nvalue,
    [atleast_nvalue(
         2,
         [[var-3],[var-1],[var-7],[var-1],[var-6]]),
     atleast_nvalue(
         4,
         [[var-3],[var-1],[var-7],[var-1],[var-6]]),
     atleast_nvalue(
         5,
         [[var-3],[var-1],[var-7],[var-0],[var-6]])]).

ctr_typical(
    atleast_nvalue,
    ['NVAL'>0,
     'NVAL'<size('VARIABLES'),
     'NVAL'<range('VARIABLES'^var),
     size('VARIABLES')>1]).

ctr_typical_model(atleast_nvalue,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    atleast_nvalue,
    [vals(['NVAL'],int(>=(0)),>,dontcare,dontcare),
     items('VARIABLES',all),
```

```
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    atleast_nvalue,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSCC'>='NVAL'],
    ['EQUIVALENCE']).

ctr_eval(
    atleast_nvalue,
    [checker(atleast_nvalue_c),
     reformulation(atleast_nvalue_r)]).

ctr_extensible(atleast_nvalue,[],'VARIABLES',any).

ctr_total_relation(atleast_nvalue).

ctr_sol(atleast_nvalue,2,0,2,24,[0-9,1-9,2-6]).

ctr_sol(atleast_nvalue,3,0,3,212,[0-64,1-64,2-60,3-24]).

ctr_sol(
    atleast_nvalue,
    4,
    0,
    4,
    2470,
    [0-625,1-625,2-620,3-480,4-120]).

ctr_sol(
    atleast_nvalue,
    5,
    0,
    5,
    35682,
    [0-7776,1-7776,2-7770,3-7320,4-4320,5-720]).

ctr_sol(
    atleast_nvalue,
    6,
    0,
    6,
    614600,
```

```
    [0-117649,
     1-117649,
     2-117642,
     3-116340,
     4-97440,
     5-42840,
     6-5040]).

ctr_sol(
    atleast_nvalue,
    7,
    0,
    7,
    12286024,
    [0-2097152,
     1-2097152,
     2-2097144,
     3-2093616,
     4-1992480,
     5-1404480,
     6-463680,
     7-40320]).

ctr_sol(
    atleast_nvalue,
    8,
    0,
    8,
    279472266,
    [0-43046721,
     1-43046721,
     2-43046712,
     3-43037568,
     4-42550704,
     5-37406880,
     6-21530880,
     7-5443200,
     8-362880]).

atleast_nvalue_r(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        NVAL#>=0,
        NVAL#=<N,
```

```
        list_dvar_range(VARS,R),
        NVAL#=<R,
        V in 0..N,
        V#>=NVAL,
        nvalue(V,VARS).

atleast_nvalue_c(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        (   integer(NVAL) ->
            NVAL>=0,
            NVAL=<N,
            sort(VARS,SVARS),
            length(SVARS,M),
            M>=NVAL
        ;   NVAL#>=0,
            NVAL#=<N,
            sort(VARS,SVARS),
            length(SVARS,M),
            M#>=NVAL
        ).
```

B.40 atleast_nvector

◇ **META-DATA:**

```
ctr_date(atleast_nvector,['20081226']).

ctr_origin(atleast_nvector,'Derived from %c',[nvector]).

ctr_types(atleast_nvector,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    atleast_nvector,
    ['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    atleast_nvector,
    [size('VECTOR')>=1,
     'NVEC'>=0,
     'NVEC'=<size('VECTORS'),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    atleast_nvector,
    atleast_nvector(
        2,
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-4]]]])).

ctr_typical(
    atleast_nvector,
    [size('VECTOR')>1,
     'NVEC'>1,
     'NVEC'<size('VECTORS'),
     size('VECTORS')>1]).

ctr_exchangeable(
    atleast_nvector,
    [vals(['NVEC'],int(>=(0)),>,dontcare,dontcare),
     items('VECTORS',all),
     items_sync('VECTORS'^vec,all),
     vals(['VECTORS'^vec],int,=\=,all,dontcare)]).
```

```
ctr_graph(
    atleast_nvector,
    ['VECTORS'],
    2,
    ['CLIQUE'>>collection(vectors1,vectors2)],
    [lex_equal(vectors1^vec,vectors2^vec)],
    ['NSCC'>='NVEC'],
    ['EQUIVALENCE']).

ctr_eval(atleast_nvector,[reformulation(atleast_nvector_r)]).

ctr_extensible(atleast_nvector,[],'VECTORS',any).

atleast_nvector_r(NVEC,[]) :-
        !,
        check_type(dvar,NVEC),
        NVEC#=0.

atleast_nvector_r(NVEC,VECTORS) :-
        check_type(dvar,NVEC),
        length(VECTORS,N),
        NVEC#>=0,
        NVEC#=<N,
        NV in 0..N,
        nvector_common(NV,VECTORS),
        NV#>=NVEC.
```

B.41 atmost

◇ **META-DATA:**

```
ctr_date(atmost,['20030820','20040807','20060804']).

ctr_origin(atmost,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(atmost,[count]).

ctr_arguments(
    atmost,
    ['N'-int,'VARIABLES'-collection(var-dvar),'VALUE'-int]).

ctr_restrictions(atmost,['N'>=0,required('VARIABLES',var)]).

ctr_example(
    atmost,
    atmost(1,[[var-4],[var-2],[var-4],[var-5]],2)).

ctr_typical(
    atmost,
    ['N'>0,
     'N'<size('VARIABLES'),
     size('VARIABLES')>1,
     atleast(1,'VARIABLES','VALUE')]).

ctr_exchangeable(
    atmost,
    [items('VARIABLES',all),
     vals(['N'],int,<,dontcare,dontcare),
     vals(
         ['VARIABLES'^var],
         comp('VALUE'),
         =<,
         dontcare,
         dontcare)]).

ctr_graph(
    atmost,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE'],
    ['NARC'=<'N'],
    []).
```

```
ctr_eval(
    atmost,
    [checker(atmost_c),
     reformulation(atmost_r),
     automaton(atmost_a)]).

ctr_contractible(atmost,[],'VARIABLES',any).

ctr_total_relation(atmost).

atmost_c(N,VARIABLES,VALUE) :-
        integer(N),
        integer(VALUE),
        N>=0,
        atmost_c1(VARIABLES,N,VALUE).

atmost_c1([[var-V]|R],N,VALUE) :-
        !,
        integer(V),
        (   V=VALUE ->
            N1 is N-1,
            N1>=0
        ;   N1 is N
        ),
        atmost_c1(R,N1,VALUE).

atmost_c1([],_42304,_42305).

atmost_r(N,VARIABLES,VALUE) :-
        integer(N),
        collection(VARIABLES,[dvar]),
        integer(VALUE),
        N>=0,
        get_attr1(VARIABLES,VARS),
        atmost1_(VARS,VALUE,SUM_BVARS),
        call(SUM_BVARS#=<N).

atmost1_([],_42304,0).

atmost1_([V|R],VALUE,B+S) :-
        V#=VALUE#<=>B,
        atmost1_(R,VALUE,S).

atmost_a(FLAG,N,VARIABLES,VALUE) :-
        integer(N),
```

```
        collection(VARIABLES,[dvar]),
        integer(VALUE),
        N>=0,
        atmost_signature(VARIABLES,SIGNATURE,VALUE),
        length(VARIABLES,M),
        MN is min(M,N),
        NVAR in 0..MN,
        automaton(
            SIGNATURE,
            _44337,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NVAR#<=>FLAG.

atmost_signature([],[],_42305).

atmost_signature([[var-VAR]|VARs],[S|Ss],VALUE) :-
        VAR#=VALUE#<=>S,
        atmost_signature(VARs,Ss,VALUE).
```

B.42 atmost1

◇ META-DATA:

```
ctr_predefined(atmost1).

ctr_date(atmost1,['20061003']).

ctr_origin(atmost1,'\\cite{SadlerGervet01}',[]).

ctr_synonyms(atmost1,[pair_atmost1]).

ctr_arguments(atmost1,['SETS'-collection(s-svar,c-int)]).

ctr_restrictions(atmost1,[required('SETS',[s,c]),'SETS'^c>=1]).

ctr_example(
    atmost1,
    atmost1(
        [[s-{5,8},c-2],
         [s-{5},c-1],
         [s-{5,6,7},c-3],
         [s-{1,4},c-2]])).

ctr_typical(atmost1,[size('SETS')>1]).

ctr_exchangeable(
    atmost1,
    [items('SETS',all),vals(['SETS'^s],int,=\=,all,dontcare)]).

ctr_contractible(atmost1,[],'SETS',any).
```

B.43 atmost_nvalue

◇ **META-DATA:**

```
ctr_date(atmost_nvalue,['20050618','20060804','20090926']).

ctr_origin(
    atmost_nvalue,
    \cite{BessiereHebrardHnichKiziltanWalsh05},
    []).

ctr_synonyms(
    atmost_nvalue,
    [soft_alldiff_max_var,
     soft_alldifferent_max_var,
     soft_alldistinct_max_var]).

ctr_arguments(
    atmost_nvalue,
    ['NVAL'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    atmost_nvalue,
    ['NVAL'>=min(1,size('VARIABLES')),
     required('VARIABLES',var)]).

ctr_example(
    atmost_nvalue,
    [atmost_nvalue(
         4,
         [[var-3],[var-1],[var-3],[var-1],[var-6]]),
     atmost_nvalue(
         3,
         [[var-3],[var-1],[var-3],[var-1],[var-6]]),
     atmost_nvalue(
         1,
         [[var-3],[var-3],[var-3],[var-3],[var-3]])]).

ctr_typical(
    atmost_nvalue,
    ['NVAL'>1,'NVAL'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_exchangeable(
    atmost_nvalue,
    [vals(['NVAL'],int,<,dontcare,dontcare),
     items('VARIABLES',all),
```

```
     vals(['VARIABLES'^var],int,=\=,all,dontcare),
     vals(['VARIABLES'^var],int,=\=,dontcare,in)]).

ctr_graph(
    atmost_nvalue,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSCC'=<'NVAL'],
    ['EQUIVALENCE']).

ctr_eval(
    atmost_nvalue,
    [reformulation(atmost_nvalue_r),checker(atmost_nvalue_c)]).

ctr_contractible(atmost_nvalue,[],'VARIABLES',any).

ctr_total_relation(atmost_nvalue).

ctr_sol(atmost_nvalue,2,0,2,12,[1-3,2-9]).

ctr_sol(atmost_nvalue,3,0,3,108,[1-4,2-40,3-64]).

ctr_sol(atmost_nvalue,4,0,4,1280,[1-5,2-145,3-505,4-625]).

ctr_sol(
    atmost_nvalue,
    5,
    0,
    5,
    18750,
    [1-6,2-456,3-3456,4-7056,5-7776]).

ctr_sol(
    atmost_nvalue,
    6,
    0,
    6,
    326592,
    [1-7,2-1309,3-20209,4-74809,5-112609,6-117649]).

ctr_sol(
    atmost_nvalue,
    7,
    0,
```

```
    7,
    6588344,
    [1-8,
     2-3536,
     3-104672,
     4-692672,
     5-1633472,
     6-2056832,
     7-2097152]).

ctr_sol(
    atmost_nvalue,
    8,
    0,
    8,
    150994944,
    [1-9,
     2-9153,
     3-496017,
     4-5639841,
     5-21515841,
     6-37603521,
     7-42683841,
     8-43046721]).

atmost_nvalue_r(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        NVAL#=<N,
        V in 0..N,
        V#=<NVAL,
        nvalue(V,VARS).

atmost_nvalue_c(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        (   integer(NVAL) ->
            MIN is min(1,N),
            NVAL>=MIN,
            sort(VARS,SVARS),
            length(SVARS,M),
            M=<NVAL
```

```
;    NVAL#>=min(1,N),
     sort(VARS,SVARS),
     length(SVARS,M),
     M#=<NVAL
).
```

B.44 atmost_nvector

◊ **META-DATA:**

```
ctr_date(atmost_nvector,['20081226']).

ctr_origin(atmost_nvector,'Derived from %c',[nvector]).

ctr_types(atmost_nvector,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    atmost_nvector,
    ['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    atmost_nvector,
    [size('VECTOR')>=1,
     'NVEC'>=min(1,size('VECTORS')),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    atmost_nvector,
    atmost_nvector(
        3,
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]]])).

ctr_typical(
    atmost_nvector,
    [size('VECTOR')>1,
     'NVEC'>1,
     'NVEC'<size('VECTORS'),
     size('VECTORS')>1]).

ctr_exchangeable(
    atmost_nvector,
    [vals(['NVEC'],int,<,dontcare,dontcare),
     items('VECTORS',all),
     items_sync('VECTORS'^vec,all),
     vals(['VECTORS'^vec],int,=\=,all,dontcare)]).

ctr_graph(
```

```
    atmost_nvector,
    ['VECTORS'],
    2,
    ['CLIQUE'>>collection(vectors1,vectors2)],
    [lex_equal(vectors1^vec,vectors2^vec)],
    ['NSCC'=<'NVEC'],
    ['EQUIVALENCE']).

ctr_eval(atmost_nvector,[reformulation(atmost_nvector_r)]).

ctr_contractible(atmost_nvector,[],'VECTORS',any).

atmost_nvector_r(NVEC,[]) :-
        !,
        check_type(dvar,NVEC),
        0#=<NVEC.

atmost_nvector_r(NVEC,VECTORS) :-
        check_type(dvar,NVEC),
        length(VECTORS,N),
        NV in 0..N,
        nvector_common(NV,VECTORS),
        NV#=<NVEC.
```

B.45 balance

◇ **META-DATA:**

```
ctr_date(balance,['20000128','20030820','20060804','20110713']).

ctr_origin(balance,'N.~Beldiceanu',[]).

ctr_arguments(
    balance,
    ['BALANCE'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    balance,
    ['BALANCE'>=0,
     'BALANCE'=<max(0,size('VARIABLES')-2),
     required('VARIABLES',var)]).

ctr_example(
    balance,
    [balance(2,[[var-3],[var-1],[var-7],[var-1],[var-1]]),
     balance(
         0,
         [[var-3],[var-3],[var-1],[var-1],[var-1],[var-3]]),
     balance(
         4,
         [[var-3],[var-1],[var-1],[var-1],[var-1],[var-1]])]).

ctr_typical(
    balance,
    ['BALANCE'=<2+size('VARIABLES')/10,size('VARIABLES')>2]).

ctr_typical_model(balance,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    balance,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    balance,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['RANGE_NSCC'='BALANCE'],
```

```
    ['EQUIVALENCE']).

ctr_eval(balance,[checker(balance_c),reformulation(balance_r)]).

ctr_pure_functional_dependency(balance,[]).

ctr_functional_dependency(balance,1,[2]).

ctr_sol(balance,2,0,2,9,[0-9]).

ctr_sol(balance,3,0,3,64,[0-28,1-36]).

ctr_sol(balance,4,0,4,625,[0-185,1-360,2-80]).

ctr_sol(balance,5,0,5,7776,[0-726,1-5700,2-1200,3-150]).

ctr_sol(
    balance,
    6,
    0,
    6,
    117649,
    [0-8617,1-75600,2-30030,3-3150,4-252]).

ctr_sol(
    balance,
    7,
    0,
    7,
    2097152,
    [0-40328,1-1342600,2-611520,3-95256,4-7056,5-392]).

ctr_sol(
    balance,
    8,
    0,
    8,
    43046721,
    [0-682929,
     1-24272640,
     2-15350832,
     3-2469600,
     4-256032,
     5-14112,
     6-576]).
```

```
balance_c(0,[]) :-
        !.

balance_c(BALANCE,VARIABLES) :-
        check_type(dvar,BALANCE),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        N2 is max(N-2,0),
        BALANCE#>=0,
        BALANCE#=<N2,
        samsort(VARS,SVARS),
        SVARS=[V|R],
        min_max_seq_size(R,1,V,N,1,MIN,MAX),
        BALANCE#=MAX-MIN.

min_max_seq_size([],C,_70287,BestMin,BestMax,ResMin,ResMax) :-
        !,
        ResMin is min(C,BestMin),
        ResMax is max(C,BestMax).

min_max_seq_size([V|R],C,V,BestMin,BestMax,ResMin,ResMax) :-
        !,
        C1 is C+1,
        min_max_seq_size(R,C1,V,BestMin,BestMax,ResMin,ResMax).

min_max_seq_size([V|R],C,Prev,BestMin,BestMax,ResMin,ResMax) :-
        C>0,
        V=\=Prev,
        NewBestMin is min(C,BestMin),
        NewBestMax is max(C,BestMax),
        min_max_seq_size(
            R,
            1,
            V,
            NewBestMin,
            NewBestMax,
            ResMin,
            ResMax).

balance_r(0,[]) :-
        !.

balance_r(BALANCE,VARIABLES) :-
        check_type(dvar,BALANCE),
        collection(VARIABLES,[dvar]),
```

```
get_attr1(VARIABLES,VARS),
length(VARIABLES,N),
N2 is max(N-2,0),
BALANCE#>=0,
BALANCE#=<N2,
union_dom_list_int(VARS,UnionDomainsVARS),
NSquare is N*N,
length(UnionDomainsVARS,SizeUnion),
(   SizeUnion=<NSquare ->
    balance1(UnionDomainsVARS,N,VALS,OCCS,OCCS1),
    eval(global_cardinality(VARIABLES,VALS))
;   balance2(VARS,N,VARS,OCCS),
    OCCS1=OCCS
),
MIN in 1..N,
MAX in 1..N,
eval(minimum(MIN,OCCS1)),
eval(maximum(MAX,OCCS)),
BALANCE+MIN#=MAX.
```

B.46 balance_cycle

◊ **META-DATA:**

```
ctr_date(balance_cycle,['20111218']).

ctr_origin(
    balance_cycle,
    derived from %c and %c,
    [balance,cycle]).

ctr_arguments(
    balance_cycle,
    ['BALANCE'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    balance_cycle,
    ['BALANCE'>=0,
     'BALANCE'=<max(0,size('NODES')-2),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    balance_cycle,
    [balance_cycle(
         1,
         [[index-1,succ-2],
          [index-2,succ-1],
          [index-3,succ-5],
          [index-4,succ-3],
          [index-5,succ-4]]),
     balance_cycle(
         0,
         [[index-1,succ-2],
          [index-2,succ-3],
          [index-3,succ-1],
          [index-4,succ-5],
          [index-5,succ-6],
          [index-6,succ-4]]),
     balance_cycle(
         4,
         [[index-1,succ-2],
```

```
          [index-2,succ-3],
          [index-3,succ-4],
          [index-4,succ-5],
          [index-5,succ-1],
          [index-6,succ-6]])]).

ctr_typical(balance_cycle,[size('NODES')>2]).

ctr_exchangeable(balance_cycle,[items('NODES',all)]).

ctr_graph(
    balance_cycle,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['NTREE'=0,'RANGE_NCC'='BALANCE'],
    ['ONE_SUCC']).

ctr_eval(balance_cycle,[checker(balance_cycle_c)]).

ctr_functional_dependency(balance_cycle,1,[2]).

ctr_cond_imply(
    balance_cycle,
    all_differ_from_at_least_k_pos,
    ['BALANCE'>0,'BALANCE'=<2],
    [],
    [same('BALANCE'),same('NODES')]).

ctr_cond_imply(
    balance_cycle,
    permutation,
    [],
    [],
    [index_to_col('NODES')]).

ctr_application(balance_cycle,[2]).

ctr_sol(balance_cycle,2,0,2,2,[0-2]).

ctr_sol(balance_cycle,3,0,3,6,[0-3,1-3]).

ctr_sol(balance_cycle,4,0,4,24,[0-10,1-6,2-8]).

ctr_sol(balance_cycle,5,0,5,120,[0-25,1-45,2-20,3-30]).
```

```
ctr_sol(balance_cycle,6,0,6,720,[0-176,1-60,2-250,3-90,4-144]).

ctr_sol(
    balance_cycle,
    7,
    0,
    7,
    5040,
    [0-721,1-861,2-770,3-1344,4-504,5-840]).

ctr_sol(
    balance_cycle,
    8,
    0,
    8,
    40320,
    [0-6406,1-1778,2-7980,3-6300,4-8736,5-3360,6-5760]).

ctr_sol(
    balance_cycle,
    9,
    0,
    9,
    362880,
    [0-42561,
     1-23283,
     2-38808,
     3-75348,
     4-45360,
     5-66240,
     6-25920,
     7-45360]).

ctr_sol(
    balance_cycle,
    10,
    0,
    10,
    3628800,
    [0-436402,
     1-84150,
     2-363680,
     3-456120,
     4-708048,
     5-378000,
```

```
     6-572400,
     7-226800,
     8-403200]).

balance_cycle_c(BALANCE,NODES) :-
        length(NODES,N),
        N2 is max(N-2,0),
        check_type(dvar(0,N2),BALANCE),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        sort(INDEXES,Js),
        sort(SUCCS,Js),
        length(Js,N),
        (for(J,1,N),
         foreach(X,SUCCS),
         foreach(Free,Term),
         foreach(Free-1,KeyTerm),foreach(J,Js),param(Term,N)do
         nth1(X,Term,Free)),
        keysort(KeyTerm,KeySorted),
        keyclumped(KeySorted,KeyClumped),
        (foreach(_72583-Ones,KeyClumped),
         foreach(Count,Counts)do
         length(Ones,Count)),
        min_member(Min,Counts),
        max_member(Max,Counts),
        BALANCE is Max-Min.
```

B.47 balance_interval

◇ **META-DATA:**

```
ctr_date(balance_interval,['20030820','20060804']).

ctr_origin(balance_interval,'Derived from %c.',[balance]).

ctr_arguments(
    balance_interval,
    ['BALANCE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    balance_interval,
    ['BALANCE'>=0,
     'BALANCE'=<max(0,size('VARIABLES')-2),
     required('VARIABLES',var),
     'SIZE_INTERVAL'>0]).

ctr_example(
    balance_interval,
    balance_interval(
        3,
        [[var-6],[var-4],[var-3],[var-3],[var-4]],
        3)).

ctr_typical(
    balance_interval,
    [size('VARIABLES')>2,
     'SIZE_INTERVAL'>1,
     'SIZE_INTERVAL'<range('VARIABLES'^var)]).

ctr_exchangeable(
    balance_interval,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    balance_interval,
```

```
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var/'SIZE_INTERVAL'=
     variables2^var/'SIZE_INTERVAL'],
    ['RANGE_NSCC'='BALANCE'],
    ['EQUIVALENCE']).

ctr_pure_functional_dependency(balance_interval,[]).

ctr_functional_dependency(balance_interval,1,[2,3]).
```

B.48 balance_modulo

◇ **META-DATA:**

```
ctr_date(balance_modulo,['20030820','20060804']).

ctr_origin(balance_modulo,'Derived from %c.',[balance]).

ctr_arguments(
    balance_modulo,
    ['BALANCE'-dvar,'VARIABLES'-collection(var-dvar),'M'-int]).

ctr_restrictions(
    balance_modulo,
    ['BALANCE'>=0,
     'BALANCE'=<max(0,size('VARIABLES')-2),
     required('VARIABLES',var),
     'M'>0]).

ctr_example(
    balance_modulo,
    balance_modulo(
        2,
        [[var-6],[var-1],[var-7],[var-1],[var-5]],
        3)).

ctr_typical(
    balance_modulo,
    [size('VARIABLES')>2,'M'>1,'M'<maxval('VARIABLES'^var)]).

ctr_exchangeable(
    balance_modulo,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
    balance_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['RANGE_NSCC'='BALANCE'],
    ['EQUIVALENCE']).

ctr_pure_functional_dependency(balance_modulo,[]).
```

```
ctr_functional_dependency(balance_modulo,1,[2,3]).
```

B.49 balance_partition

◊ **META-DATA:**

```
ctr_date(balance_partition,['20030820','20060804']).

ctr_origin(balance_partition,'Derived from %c.',[balance]).

ctr_types(balance_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    balance_partition,
    ['BALANCE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    balance_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     'BALANCE'>=0,
     'BALANCE'=<max(0,size('VARIABLES')-2),
     required('VARIABLES',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    balance_partition,
    balance_partition(
        1,
        [[var-6],[var-2],[var-6],[var-4],[var-4]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(
    balance_partition,
    [size('VARIABLES')>2,size('VARIABLES')>size('PARTITIONS')]).

ctr_exchangeable(
    balance_partition,
    [items('VARIABLES',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
```

```
          ['VARIABLES'^var],
          part('PARTITIONS'),
          =,
          dontcare,
          dontcare)]).

ctr_graph(
    balance_partition,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    ['RANGE_NSCC'='BALANCE'],
    ['EQUIVALENCE']).

ctr_pure_functional_dependency(balance_partition,[]).

ctr_functional_dependency(balance_partition,1,[2,3]).
```

B.50 balance_path

◊ **META-DATA:**

```
ctr_date(balance_path,['20111226']).

ctr_origin(
    balance_path,
    derived from %c and %c,
    [balance,path]).

ctr_arguments(
    balance_path,
    ['BALANCE'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    balance_path,
    ['BALANCE'>=0,
     'BALANCE'=<max(0,size('NODES')-2),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    balance_path,
    [balance_path(
         3,
         [[index-1,succ-1],
          [index-2,succ-3],
          [index-3,succ-5],
          [index-4,succ-4],
          [index-5,succ-1],
          [index-6,succ-6],
          [index-7,succ-7],
          [index-8,succ-6]]),
     balance_path(
         0,
         [[index-1,succ-2],
          [index-2,succ-3],
          [index-3,succ-4],
          [index-4,succ-4],
          [index-5,succ-6],
          [index-6,succ-7],
```

```
          [index-7,succ-8],
          [index-8,succ-8]]),
     balance_path(
         6,
         [[index-1,succ-2],
          [index-2,succ-3],
          [index-3,succ-4],
          [index-4,succ-5],
          [index-5,succ-6],
          [index-6,succ-7],
          [index-7,succ-7],
          [index-8,succ-8]])]).

ctr_typical(balance_path,[size('NODES')>2]).

ctr_exchangeable(balance_path,[items('NODES',all)]).

ctr_graph(
    balance_path,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1,'MAX_ID'=<1,'RANGE_NCC'='BALANCE'],
    ['ONE_SUCC']).

ctr_eval(balance_path,[checker(balance_path_c)]).

ctr_functional_dependency(balance_path,1,[2]).

ctr_application(balance_path,[2]).

ctr_sol(balance_path,2,0,2,3,[0-3]).

ctr_sol(balance_path,3,0,3,13,[0-7,1-6]).

ctr_sol(balance_path,4,0,4,73,[0-37,1-12,2-24]).

ctr_sol(balance_path,5,0,5,501,[0-121,1-200,2-60,3-120]).

ctr_sol(
    balance_path,
    6,
    0,
    6,
    4051,
```

```
    [0-1201,1-210,2-1560,3-360,4-720]).

ctr_sol(
    balance_path,
    7,
    0,
    7,
    37633,
    [0-5041,1-8862,2-5250,3-10920,4-2520,5-5040]).

ctr_sol(
    balance_path,
    8,
    0,
    8,
    394353,
    [0-62161,1-24416,2-97776,3-62160,4-87360,5-20160,6-40320]).

balance_path_c(BALANCE,NODES) :-
        length(NODES,N),
        N2 is max(N-2,0),
        check_type(dvar(0,N2),BALANCE),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        sort(INDEXES,SIND),
        length(SIND,N),
        length(RANKS,N),
        domain(RANKS,1,N),
        balance_path1(INDEXES,SUCCS,RANKS,SUCC_WITHOUT_LOOPS),
        sort(SUCC_WITHOUT_LOOPS,SSUCC_WITHOUT_LOOPS),
        length(SUCC_WITHOUT_LOOPS,NSL),
        length(SSUCC_WITHOUT_LOOPS,NSL),
        (foreach(X,SUCCS),
         foreach(Free,Term),
         foreach(Free-1,KeyTerm),param(Term,N)do
         nth1(X,Term,Free)),
        keysort(KeyTerm,KeySorted),
        keyclumped(KeySorted,KeyClumped),
        (foreach(_72191-Ones,KeyClumped),
         foreach(Count,Counts)do
         length(Ones,Count)),
        min_member(Min,Counts),
        max_member(Max,Counts),
        BALANCE is Max-Min.
```

```
balance_path1([],[],_71992,[]) :-
        !.

balance_path1([I|RI],[I|RS],RANKS,SUCC) :-
        !,
        balance_path1(RI,RS,RANKS,SUCC).

balance_path1([I|RI],[S|RS],RANKS,[S|SUCC]) :-
        nth1(I,RANKS,Ri),
        nth1(S,RANKS,Rs),
        Ri#<Rs,
        balance_path1(RI,RS,RANKS,SUCC).
```

B.51 balance_tree

◇ **META-DATA:**

```
ctr_date(balance_tree,['20111226']).

ctr_origin(
    balance_tree,
    derived from %c and %c,
    [balance,tree]).

ctr_arguments(
    balance_tree,
    ['BALANCE'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    balance_tree,
    ['BALANCE'>=0,
     'BALANCE'=<max(0,size('NODES')-2),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    balance_tree,
    [balance_tree(
         4,
         [[index-1,succ-1],
          [index-2,succ-5],
          [index-3,succ-5],
          [index-4,succ-7],
          [index-5,succ-1],
          [index-6,succ-1],
          [index-7,succ-7],
          [index-8,succ-5]]),
     balance_tree(
         2,
         [[index-1,succ-1],
          [index-2,succ-1],
          [index-3,succ-1],
          [index-4,succ-2],
          [index-5,succ-6],
          [index-6,succ-6]])]).
```

```
ctr_typical(balance_tree,[size('NODES')>2]).

ctr_exchangeable(balance_tree,[items('NODES',all)]).

ctr_graph(
    balance_tree,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1,'RANGE_NCC'='BALANCE'],
    []).

ctr_eval(balance_tree,[checker(balance_tree_c)]).

ctr_functional_dependency(balance_tree,1,[2]).

ctr_cond_imply(
    balance_tree,
    ordered_atleast_nvector,
    ['BALANCE'>0,'BALANCE'=<size('NODES')],
    [],
    [same('BALANCE'),same('NODES')]).

ctr_application(balance_tree,[2]).

ctr_sol(balance_tree,2,0,2,3,[0-3]).

ctr_sol(balance_tree,3,0,3,16,[0-10,1-6]).

ctr_sol(balance_tree,4,0,4,125,[0-77,1-12,2-36]).

ctr_sol(balance_tree,5,0,5,1296,[0-626,1-260,2-90,3-320]).

ctr_sol(
    balance_tree,
    6,
    0,
    6,
    16807,
    [0-8707,1-210,2-3180,3-960,4-3750]).

ctr_sol(
    balance_tree,
    7,
```

```
    0,
    7,
    262144,
    [0-117650,1-25242,2-9765,3-41930,4-13125,5-54432]).

ctr_sol(
    balance_tree,
    8,
    0,
    8,
    4782969,
    [0-2242193,
     1-49616,
     2-432264,
     3-219520,
     4-680456,
     5-217728,
     6-941192]).

balance_tree_c(BALANCE,NODES) :-
        length(NODES,N),
        N2 is max(N-2,0),
        check_type(dvar(0,N2),BALANCE),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        sort(INDEXES,SIND),
        length(SIND,N),
        length(RANKS,N),
        domain(RANKS,1,N),
        balance_tree1(INDEXES,SUCCS,RANKS),
        (foreach(X,SUCCS),
         foreach(Free,Term),
         foreach(Free-1,KeyTerm),param(Term,N)do
         nth1(X,Term,Free)),
        keysort(KeyTerm,KeySorted),
        keyclumped(KeySorted,KeyClumped),
        (foreach(_70436-Ones,KeyClumped),
         foreach(Count,Counts)do
         length(Ones,Count)),
        min_member(Min,Counts),
        max_member(Max,Counts),
        BALANCE is Max-Min.

balance_tree1([],[],_70262) :-
        !.
```

```
balance_tree1([I|RI],[I|RS],RANKS) :-
        !,
        balance_tree1(RI,RS,RANKS).

balance_tree1([I|RI],[S|RS],RANKS) :-
        nth1(I,RANKS,Ri),
        nth1(S,RANKS,Rs),
        Ri#<Rs,
        balance_tree1(RI,RS,RANKS).
```

B.52 between_min_max

◇ **META-DATA:**

```
ctr_date(between_min_max,['20050824','20060804']).

ctr_origin(
    between_min_max,
    Used for defining %c.,
    [cumulative_convex]).

ctr_arguments(
    between_min_max,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    between_min_max,
    [required('VARIABLES',var),size('VARIABLES')>0]).

ctr_example(
    between_min_max,
    [between_min_max(3,[[var-1],[var-1],[var-4],[var-8]]),
     between_min_max(1,[[var-1],[var-1],[var-4],[var-8]]),
     between_min_max(8,[[var-1],[var-1],[var-4],[var-8]])]).

ctr_typical(
    between_min_max,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_exchangeable(
    between_min_max,
    [items('VARIABLES',all),
     vals(
         ['VAR'],
         int(['VAR','VARIABLES'^var]),
         =\=,
         all,
         dontcare)]).

ctr_derived_collections(
    between_min_max,
    [col('ITEM'-collection(var-dvar),[item(var-'VAR')])]).

ctr_graph(
    between_min_max,
    ['ITEM','VARIABLES'],
```

```
    2,
    ['PRODUCT'>>collection(item,variables)],
    [item^var>=variables^var],
    ['NARC'>=1],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_graph(
    between_min_max,
    ['ITEM','VARIABLES'],
    2,
    ['PRODUCT'>>collection(item,variables)],
    [item^var=<variables^var],
    ['NARC'>=1],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    between_min_max,
    [checker(between_min_max_c),
     reformulation(between_min_max_r),
     automaton(between_min_max_a)]).

ctr_extensible(between_min_max,[],'VARIABLES',any).

ctr_sol(between_min_max,2,0,2,17,[0-5,1-7,2-5]).

ctr_sol(between_min_max,3,0,3,184,[0-37,1-55,2-55,3-37]).

ctr_sol(
    between_min_max,
    4,
    0,
    4,
    2417,
    [0-369,1-543,2-593,3-543,4-369]).

ctr_sol(
    between_min_max,
    5,
    0,
    5,
    37806,
    [0-4651,1-6751,2-7501,3-7501,4-6751,5-4651]).

ctr_sol(
    between_min_max,
    6,
```

```
    0,
    6,
    689201,
    [0-70993,
     1-102023,
     2-113489,
     3-116191,
     4-113489,
     5-102023,
     6-70993]).

ctr_sol(
    between_min_max,
    7,
    0,
    7,
    14376608,
    [0-1273609,
     1-1817215,
     2-2018899,
     3-2078581,
     4-2078581,
     5-2018899,
     6-1817215,
     7-1273609]).

ctr_sol(
    between_min_max,
    8,
    0,
    8,
    338051265,
    [0-26269505,
     1-37281919,
     2-41366849,
     3-42649535,
     4-42915649,
     5-42649535,
     6-41366849,
     7-37281919,
     8-26269505]).

between_min_max_c(VAR,VARIABLES) :-
        integer(VAR),
        between_min_max_c(VARIABLES,VAR,0,0).
```

```
between_min_max_c([],_64292,1,1) :-
        !.

between_min_max_c([[var-V]|R],VAR,Min,Max) :-
        integer(V),
        (   VAR=V ->
            between_min_max_c(R,VAR,1,1)
        ;   VAR>V ->
            between_min_max_c(R,VAR,1,Max)
        ;   between_min_max_c(R,VAR,Min,1)
        ).

between_min_max_r(VAR,VARIABLES) :-
        check_type(dvar,VAR),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        get_minimum(VARS,MINIMUM),
        get_maximum(VARS,MAXIMUM),
        MIN in MINIMUM..MAXIMUM,
        MAX in MINIMUM..MAXIMUM,
        minimum(MIN,VARS),
        maximum(MAX,VARS),
        VAR#>=MIN,
        VAR#=<MAX.

between_min_max_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar,VAR),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        between_min_max_signature(VARIABLES,VAR,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _66242,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,i),
             arc(s,1,t),
             arc(s,2,j),
             arc(i,0,i),
             arc(i,1,t),
             arc(i,2,t),
             arc(j,0,t),
```

```
            arc(j,1,t),
            arc(j,2,j),
            arc(t,0,t),
            arc(t,1,t),
            arc(t,2,t)],
           [],
           [],
           []),
        automaton_bool(FLAG,[0,1,2],AUTOMATON).

between_min_max_signature([],_64289,[]).

between_min_max_signature([[var-VARi]|VARs],VAR,[S|Ss]) :-
        S in 0..2,
        VAR#<VARi#<=>S#=0,
        VAR#=VARi#<=>S#=1,
        VAR#>VARi#<=>S#=2,
        between_min_max_signature(VARs,VAR,Ss).
```

B.53 big_peak

◊ **META-DATA:**

```
ctr_date(big_peak,['20130125']).

ctr_origin(big_peak,'Derived from %c.',[peak]).

ctr_arguments(
    big_peak,
    ['N'-dvar,
     'VARIABLES'-collection(var-dvar),
     'TOLERANCE'-int]).

ctr_restrictions(
    big_peak,
    ['N'>=0,
     2*'N'=<max(size('VARIABLES')-1,0),
     required('VARIABLES',var),
     'TOLERANCE'>=0]).

ctr_example(
    big_peak,
    [big_peak(
         7,
         [[var-4],
          [var-2],
          [var-2],
          [var-4],
          [var-3],
          [var-8],
          [var-6],
          [var-7],
          [var-7],
          [var-9],
          [var-5],
          [var-6],
          [var-3],
          [var-12],
          [var-12],
          [var-6],
          [var-6],
          [var-8],
          [var-4],
          [var-5],
          [var-1]],
```

```
        0),
    big_peak(
        4,
        [[var-4],
         [var-2],
         [var-2],
         [var-4],
         [var-3],
         [var-8],
         [var-6],
         [var-7],
         [var-7],
         [var-9],
         [var-5],
         [var-6],
         [var-3],
         [var-12],
         [var-12],
         [var-6],
         [var-6],
         [var-8],
         [var-4],
         [var-5],
         [var-1]],
        1)]).

ctr_typical(
    big_peak,
    ['N'>=1,
     size('VARIABLES')>6,
     range('VARIABLES'^var)>1,
     'TOLERANCE'>1]).

ctr_typical_model(
    big_peak,
    [nval('VARIABLES'^var)>2,range('VARIABLES'^var)>2]).

ctr_exchangeable(
    big_peak,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    big_peak,
    [checker(big_peak_c),
     automaton(big_peak_a),
     automaton_with_signature(big_peak_a_s)]).
```

```
ctr_pure_functional_dependency(big_peak,[]).

ctr_functional_dependency(big_peak,1,[2,3]).

ctr_contractible(
    big_peak,
    ['N'=0,'TOLERANCE'=0],
    VARIABLES,
    any).

big_peak_c(N,VARIABLES,TOLERANCE) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        MAX is max(L-1,0),
        2*N#=<MAX,
        integer(TOLERANCE),
        TOLERANCE>=0,
        (   L<3 ->
            N=0
        ;   get_attr1(VARIABLES,VARS),
            last(VARS,Last),
            big_peak_c(s,VARS,0,0,-2000000,TOLERANCE,Last,N)
        ).

big_peak_c(s,[V1,V2|R],C,S,P,T,L,N) :-
        V1>=V2,
        !,
        big_peak_c(s,[V2|R],C,S,P,T,L,N).

big_peak_c(s,[V1,V2|R],C,_28327,P,T,L,N) :-
        !,
        big_peak_c(u,[V2|R],C,V1,P,T,L,N).

big_peak_c(u,[V1,V2|R],C,S,P,T,L,N) :-
        V1=<V2,
        !,
        big_peak_c(u,[V2|R],C,S,P,T,L,N).

big_peak_c(u,[V1,V2|R],C,S,P,T,L,N) :-
        D is V1-S,
        D=<T,
        !,
        big_peak_c(u,[V2|R],C,S,P,T,L,N).
```

```
big_peak_c(u,[V1,V2|R],C,S,_28328,T,L,N) :-
        !,
        big_peak_c(v,[V2|R],C,S,V1,T,L,N).

big_peak_c(v,[V1,V2|R],C,S,P,T,L,N) :-
        V1>=V2,
        !,
        big_peak_c(v,[V2|R],C,S,P,T,L,N).

big_peak_c(v,[V1,V2|R],C,_28327,P,T,L,N) :-
        P= -2000000,
        !,
        big_peak_c(w,[V2|R],C,V1,P,T,L,N).

big_peak_c(v,[V1,V2|R],C,S,P,T,L,N) :-
        P> -2000000,
        D is P-V1,
        D=<T,
        !,
        big_peak_c(w,[V2|R],C,S,P,T,L,N).

big_peak_c(v,[V1,V2|R],C,_28327,_28328,T,L,N) :-
        !,
        C1 is C+1,
        big_peak_c(w,[V2|R],C1,V1,-2000000,T,L,N).

big_peak_c(w,[V1,V2|R],C,S,P,T,L,N) :-
        V1=<V2,
        !,
        big_peak_c(w,[V2|R],C,S,P,T,L,N).

big_peak_c(w,[V1,V2|R],C,S,P,T,L,N) :-
        D is V1-S,
        D=<T,
        !,
        big_peak_c(v,[V2|R],C,S,P,T,L,N).

big_peak_c(w,[V1,V2|R],C,S,P,T,L,N) :-
        !,
        PP is max(P,V1),
        big_peak_c(v,[V2|R],C,S,PP,T,L,N).

big_peak_c(_28324,[_28332],C,_28327,P,T,L,N) :-
        !,
        D is P-L,
        (   D>T ->
```

```
            C1 is C+1,
            N=C1
        ;   N=C
        ).

big_peak_c(_28321,[],_28323,_28324,_28325,_28326,_28327,0).

ctr_automaton_signature(
    big_peak,
    big_peak_a,
    pair_signature(2,signature)).

big_peak_a(FLAG,N,VARIABLES,TOLERANCE) :-
        length(VARIABLES,L),
        (   L<3 ->
            true
        ;   pair_signature(VARIABLES,SIGNATURE)
        ),
        big_peak_a_s(FLAG,N,VARIABLES,TOLERANCE,SIGNATURE).

big_peak_a_s(FLAG,N,VARIABLES,TOLERANCE,SIGNATURE) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        MAX is max(L-1,0),
        2*N#=<MAX,
        integer(TOLERANCE),
        TOLERANCE>=0,
        (   L<3 ->
            N#=0#<=>FLAG
        ;   pair_first_last_signature(VARIABLES,VARS,Last),
            automaton(
                VARS,
                VARi,
                SIGNATURE,
                [source(s),sink(s),sink(u),sink(v),sink(w)],
                [arc(s,2,s),
                 arc(s,1,s),
                 arc(s,0,u,[C,VARi,P]),
                 arc(u,2,u,(VARi-S#=<TOLERANCE->[C,S,P])),
                 arc(u,2,v,(VARi-S#>TOLERANCE->[C,S,VARi])),
                 arc(u,1,u),
                 arc(u,0,u),
                 arc(v,2,v),
                 arc(v,1,v),
                 arc(v,0,w,(P#= -2000000->[C,VARi,P])),
```

```
          arc(v,
              0,
              w,
              (P#> -2000000#/\P-VARi#=<TOLERANCE->
               [C,S,P])),
          arc(v,
              0,
              w,
              (P-VARi#>TOLERANCE->
               [C+1,VARi,-2000000])),
          arc(w,2,v,(VARi-S#=<TOLERANCE->[C,S,P])),
          arc(w,
              2,
              v,
              (VARi-S#>TOLERANCE->[C,S,max(P,VARi)])),
          arc(w,1,w),
          arc(w,0,w)],
         [C,S,P],
         [0,0,-2000000],
         [CC,_28703,PP]),
    Inc in 0..1,
    PP#>Last+TOLERANCE#<=>Inc,
    CC+Inc#=N#<=>FLAG
).
```

B.54 big_valley

◊ **META-DATA:**

```
ctr_date(big_valley,['20130127']).

ctr_origin(big_valley,'Derived from %c.',[valley]).

ctr_arguments(
    big_valley,
    ['N'-dvar,
     'VARIABLES'-collection(var-dvar),
     'TOLERANCE'-int]).

ctr_restrictions(
    big_valley,
    ['N'>=0,
     2*'N'=<max(size('VARIABLES')-1,0),
     required('VARIABLES',var),
     'TOLERANCE'>=0]).

ctr_example(
    big_valley,
    [big_valley(
         7,
         [[var-9],
          [var-11],
          [var-11],
          [var-9],
          [var-10],
          [var-5],
          [var-7],
          [var-6],
          [var-6],
          [var-4],
          [var-8],
          [var-7],
          [var-10],
          [var-1],
          [var-1],
          [var-7],
          [var-7],
          [var-5],
          [var-9],
          [var-8],
          [var-12]],
```

```
        0),
     big_valley(
        4,
        [[var-9],
         [var-11],
         [var-11],
         [var-9],
         [var-10],
         [var-5],
         [var-7],
         [var-6],
         [var-6],
         [var-4],
         [var-8],
         [var-7],
         [var-10],
         [var-1],
         [var-1],
         [var-7],
         [var-7],
         [var-5],
         [var-9],
         [var-8],
         [var-12]],
        1)]).

ctr_typical(
    big_valley,
    ['N'>=1,
     size('VARIABLES')>6,
     range('VARIABLES'^var)>1,
     'TOLERANCE'>1]).

ctr_typical_model(
    big_valley,
    [nval('VARIABLES'^var)>2,range('VARIABLES'^var)>2]).

ctr_exchangeable(
    big_valley,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    big_valley,
    [checker(big_valley_c),
     automaton(big_valley_a),
     automaton_with_signature(big_valley_a_s)]).
```

```
ctr_pure_functional_dependency(big_valley,[]).

ctr_functional_dependency(big_valley,1,[2,3]).

ctr_contractible(
    big_valley,
    ['N'=0,'TOLERANCE'=0],
    VARIABLES,
    any).

big_valley_c(N,VARIABLES,TOLERANCE) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        MAX is max(L-1,0),
        2*N#=<MAX,
        integer(TOLERANCE),
        TOLERANCE>=0,
        (   L<3 ->
            N=0
        ;   get_attr1(VARIABLES,VARS),
            last(VARS,Last),
            big_valley_c(s,VARS,0,0,2000000,TOLERANCE,Last,N)
        ).

big_valley_c(s,[V1,V2|R],C,S,V,T,L,N) :-
        V1=<V2,
        !,
        big_valley_c(s,[V2|R],C,S,V,T,L,N).

big_valley_c(s,[V1,V2|R],C,_28347,V,T,L,N) :-
        !,
        big_valley_c(u,[V2|R],C,V1,V,T,L,N).

big_valley_c(u,[V1,V2|R],C,S,V,T,L,N) :-
        V1>=V2,
        !,
        big_valley_c(u,[V2|R],C,S,V,T,L,N).

big_valley_c(u,[V1,V2|R],C,S,V,T,L,N) :-
        D is S-V1,
        D=<T,
        !,
        big_valley_c(u,[V2|R],C,S,V,T,L,N).
```

```
big_valley_c(u,[V1,V2|R],C,S,_28348,T,L,N) :-
        !,
        big_valley_c(v,[V2|R],C,S,V1,T,L,N).

big_valley_c(v,[V1,V2|R],C,S,V,T,L,N) :-
        V1=<V2,
        !,
        big_valley_c(v,[V2|R],C,S,V,T,L,N).

big_valley_c(v,[V1,V2|R],C,_28347,V,T,L,N) :-
        V=2000000,
        !,
        big_valley_c(w,[V2|R],C,V1,V,T,L,N).

big_valley_c(v,[V1,V2|R],C,S,V,T,L,N) :-
        V<2000000,
        D is V1-V,
        D=<T,
        !,
        big_valley_c(w,[V2|R],C,S,V,T,L,N).

big_valley_c(v,[V1,V2|R],C,_28347,_28348,T,L,N) :-
        !,
        C1 is C+1,
        big_valley_c(w,[V2|R],C1,V1,2000000,T,L,N).

big_valley_c(w,[V1,V2|R],C,S,V,T,L,N) :-
        V1>=V2,
        !,
        big_valley_c(w,[V2|R],C,S,V,T,L,N).

big_valley_c(w,[V1,V2|R],C,S,V,T,L,N) :-
        D is S-V1,
        D=<T,
        !,
        big_valley_c(v,[V2|R],C,S,V,T,L,N).

big_valley_c(w,[V1,V2|R],C,S,V,T,L,N) :-
        !,
        VV is min(V,V1),
        big_valley_c(v,[V2|R],C,S,VV,T,L,N).

big_valley_c(_28344,[_28352],C,_28347,V,T,L,N) :-
        !,
        D is L-V,
        (   D>T ->
```

```
            C1 is C+1,
            N=C1
        ;   N=C
        ).

big_valley_c(_28341,[],_28343,_28344,_28345,_28346,_28347,0).

ctr_automaton_signature(
    big_valley,
    big_valley_a,
    pair_signature(2,signature)).

big_valley_a(FLAG,N,VARIABLES,TOLERANCE) :-
        length(VARIABLES,L),
        (   L<3 ->
            true
        ;   pair_signature(VARIABLES,SIGNATURE)
        ),
        big_valley_a_s(FLAG,N,VARIABLES,TOLERANCE,SIGNATURE).

big_valley_a_s(FLAG,N,VARIABLES,TOLERANCE,SIGNATURE) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        MAX is max(L-1,0),
        2*N#=<MAX,
        integer(TOLERANCE),
        TOLERANCE>=0,
        (   L<3 ->
            N#=0#<=>FLAG
        ;   pair_first_last_signature(VARIABLES,VARS,Last),
            automaton(
                VARS,
                VARi,
                SIGNATURE,
                [source(s),sink(s),sink(u),sink(v),sink(w)],
                [arc(s,0,s),
                 arc(s,1,s),
                 arc(s,2,u,[C,VARi,V]),
                 arc(u,0,u,(S-VARi#=<TOLERANCE->[C,S,V])),
                 arc(u,0,v,(S-VARi#>TOLERANCE->[C,S,VARi])),
                 arc(u,1,u),
                 arc(u,2,u),
                 arc(v,0,v),
                 arc(v,1,v),
                 arc(v,2,w,(V#=2000000->[C,VARi,V])),
```

```
                arc(v,
                    2,
                    w,
                    (V#<2000000#/\VARi-V#=<TOLERANCE->
                     [C,S,V])),
                arc(v,
                    2,
                    w,
                    (VARi-V#>TOLERANCE->[C+1,VARi,2000000])),
                arc(w,0,v,(S-VARi#=<TOLERANCE->[C,S,V])),
                arc(w,
                    0,
                    v,
                    (S-VARi#>TOLERANCE->[C,S,min(V,VARi)])),
                arc(w,1,w),
                arc(w,2,w)],
               [C,S,V],
               [0,0,2000000],
               [CC,_28723,VV]),
          Inc in 0..1,
          Last#>VV+TOLERANCE#<=>Inc,
          CC+Inc#=N#<=>FLAG
        ).

big_valley_signature([[var-Last]],[],[],Last) :-
        !.

big_valley_signature(
     [[var-VAR1],[var-VAR2]|VARs],
     [S|Ss],
     [VAR1|Rs],
     Last) :-
        S in 0..2,
        VAR1#<VAR2#<=>S#=0,
        VAR1#=VAR2#<=>S#=1,
        VAR1#>VAR2#<=>S#=2,
        big_valley_signature([[var-VAR2]|VARs],Ss,Rs,Last).
```

B.55 bin_packing

◇ **META-DATA:**

```
ctr_date(
    bin_packing,
    ['20000128','20030820','20040530','20060804']).

ctr_origin(bin_packing,'Derived from %c.',[cumulative]).

ctr_arguments(
    bin_packing,
    ['CAPACITY'-int,'ITEMS'-collection(bin-dvar,weight-int)]).

ctr_restrictions(
    bin_packing,
    ['CAPACITY'>=0,
     required('ITEMS',[bin,weight]),
     'ITEMS'^weight>=0,
     'ITEMS'^weight=<'CAPACITY']).

ctr_example(
    bin_packing,
    bin_packing(
        5,
        [[bin-3,weight-4],[bin-1,weight-3],[bin-3,weight-1]])).

ctr_typical(
    bin_packing,
    ['CAPACITY'>maxval('ITEMS'^weight),
     'CAPACITY'=<sum('ITEMS'^weight),
     size('ITEMS')>1,
     range('ITEMS'^bin)>1,
     range('ITEMS'^weight)>1,
     'ITEMS'^bin>=0,
     'ITEMS'^weight>0]).

ctr_exchangeable(
    bin_packing,
    [vals(['CAPACITY'],int,<,dontcare,dontcare),
     items('ITEMS',all),
     vals(['ITEMS'^weight],int(>=(0)),>,dontcare,dontcare),
     vals(['ITEMS'^bin],int,=\=,all,dontcare)]).

ctr_graph(
    bin_packing,
```

```
    ['ITEMS','ITEMS'],
    2,
    ['PRODUCT'>>collection(items1,items2)],
    [items1^bin=items2^bin],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'ITEMS'^weight)])]],
    [sum_ctr(variables,=<,'CAPACITY')]).

ctr_eval(bin_packing,[reformulation(bin_packing_r)]).

ctr_contractible(bin_packing,[],'ITEMS',any).

ctr_cond_imply(
    bin_packing,
    atmost_nvector,
    ['CAPACITY'>=size('ITEMS')],
    [],
    [same('CAPACITY'),same('ITEMS')]).

ctr_application(bin_packing,[2]).

bin_packing_r(CAPACITY,ITEMS) :-
        integer(CAPACITY),
        CAPACITY>=0,
        collection(ITEMS,[dvar,int(0,CAPACITY)]),
        bin_packing1(ITEMS,1,TASKS),
        cumulative(TASKS,[limit(CAPACITY)]).
```

B.56 bin_packing_capa

◇ META-DATA:

```
ctr_predefined(bin_packing_capa).

ctr_date(bin_packing_capa,['20091404']).

ctr_origin(bin_packing_capa,'Derived from %c.',[bin_packing]).

ctr_arguments(
    bin_packing_capa,
    ['BINS'-collection(id-int,capa-int),
     'ITEMS'-collection(bin-dvar,weight-int)]).

ctr_restrictions(
    bin_packing_capa,
    [size('BINS')>0,
     required('BINS',[id,capa]),
     distinct('BINS',id),
     'BINS'^id>=1,
     'BINS'^id=<size('BINS'),
     'BINS'^capa>=0,
     required('ITEMS',[bin,weight]),
     in_attr('ITEMS',bin,'BINS',id),
     'ITEMS'^weight>=0]).

ctr_example(
    bin_packing_capa,
    bin_packing_capa(
        [[id-1,capa-4],
         [id-2,capa-3],
         [id-3,capa-5],
         [id-4,capa-3],
         [id-5,capa-3]],
        [[bin-3,weight-4],[bin-1,weight-3],[bin-3,weight-1]])).

ctr_typical(
    bin_packing_capa,
    [size('BINS')>1,
     range('BINS'^capa)>1,
     'BINS'^capa>maxval('ITEMS'^weight),
     'BINS'^capa=<sum('ITEMS'^weight),
     size('ITEMS')>1,
     range('ITEMS'^bin)>1,
     range('ITEMS'^weight)>1,
```

```
     'ITEMS'^weight>0]).

ctr_exchangeable(
    bin_packing_capa,
    [items('BINS',all),
     items('ITEMS',all),
     vals(['BINS'^capa],int,<,dontcare,dontcare),
     vals(['ITEMS'^weight],int(>=(0)),>,dontcare,dontcare),
     vals(['BINS'^id,'ITEMS'^bin],int,=\=,all,dontcare)]).

ctr_eval(bin_packing_capa,[reformulation(bin_packing_capa_r)]).

ctr_contractible(bin_packing_capa,[],'ITEMS',any).

ctr_application(bin_packing_capa,[2]).

bin_packing_capa_r(BINS,ITEMS) :-
        length(BINS,N),
        collection(BINS,[int(1,N),int_gteq(0)]),
        collection(ITEMS,[dvar,int_gteq(0)]),
        get_attr1(BINS,IDS),
        get_attr2(BINS,CAPAS),
        get_maximum(CAPAS,MAX),
        MAX1 is MAX+1,
        all_different(IDS),
        bin_packing1(ITEMS,1,TASKS),
        length(ITEMS,M),
        M1 is M+1,
        bin_packing_capa1(BINS,M1,MAX,COMPLEMENTS),
        append(COMPLEMENTS,TASKS,COMPLEMENTS_TASKS),
        cumulative(COMPLEMENTS_TASKS,[limit(MAX1)]).

bin_packing_capa1([],_30259,_30260,[]).

bin_packing_capa1(
    [[_30270-I,_30277-W]|R],
    ID,
    MAX,
    [task(I,1,I1,H,ID)|S]) :-
        I1 is I+1,
        H is MAX-W+1,
        bin_packing_capa1(R,ID,MAX,S).
```

B.57 binary_tree

◇ META-DATA:

```
ctr_date(binary_tree,['20000128','20030820','20060804']).

ctr_origin(binary_tree,'Derived from %c.',[tree]).

ctr_arguments(
    binary_tree,
    ['NTREES'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    binary_tree,
    ['NTREES'>=0,
     'NTREES'=<size('NODES'),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    binary_tree,
    [binary_tree(
         2,
         [[index-1,succ-1],
          [index-2,succ-3],
          [index-3,succ-5],
          [index-4,succ-7],
          [index-5,succ-1],
          [index-6,succ-1],
          [index-7,succ-7],
          [index-8,succ-5]]),
     binary_tree(
         8,
         [[index-1,succ-1],
          [index-2,succ-2],
          [index-3,succ-3],
          [index-4,succ-4],
          [index-5,succ-5],
          [index-6,succ-6],
          [index-7,succ-7],
          [index-8,succ-8]]),
     binary_tree(
```

```
         7,
         [[index-1,succ-8],
          [index-2,succ-2],
          [index-3,succ-3],
          [index-4,succ-4],
          [index-5,succ-5],
          [index-6,succ-6],
          [index-7,succ-7],
          [index-8,succ-8]]])]).

ctr_typical(
    binary_tree,
    ['NTREES'>0,'NTREES'<size('NODES'),size('NODES')>2]).

ctr_exchangeable(binary_tree,[items('NODES',all)]).

ctr_graph(
    binary_tree,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1,'NCC'='NTREES','MAX_ID'=<2],
    ['ONE_SUCC']).

ctr_eval(binary_tree,[reformulation(binary_tree_r)]).

ctr_functional_dependency(binary_tree,1,[2]).

ctr_application(binary_tree,[2]).

ctr_sol(binary_tree,2,0,2,3,[1-2,2-1]).

ctr_sol(binary_tree,3,0,3,16,[1-9,2-6,3-1]).

ctr_sol(binary_tree,4,0,4,121,[1-60,2-48,3-12,4-1]).

ctr_sol(binary_tree,5,0,5,1191,[1-540,2-480,3-150,4-20,5-1]).

ctr_sol(
    binary_tree,
    6,
    0,
    6,
    14461,
    [1-6120,2-5850,3-2100,4-360,5-30,6-1]).
```

```
ctr_sol(
    binary_tree,
    7,
    0,
    7,
    209098,
    [1-83790,2-84420,3-33390,4-6720,5-735,6-42,7-1]).

ctr_sol(
    binary_tree,
    8,
    0,
    8,
    3510921,
    [1-1345680,
     2-1411200,
     3-599760,
     4-135240,
     5-17640,
     6-1344,
     7-56,
     8-1]).

binary_tree_r(NTREES,NODES) :-
        eval(tree(NTREES,NODES)),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        k_ary_tree(INDEXES,INDEXES,SUCCS,2).
```

B.58 bipartite

◇ **META-DATA:**

```
ctr_date(bipartite,['20061001']).

ctr_origin(bipartite,'\\cite{Dooms06}',[]).

ctr_arguments(
    bipartite,
    ['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    bipartite,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    bipartite,
    bipartite(
        [[index-1,succ-{2,3}],
         [index-2,succ-{1,4}],
         [index-3,succ-{1,4,5}],
         [index-4,succ-{2,3,6}],
         [index-5,succ-{3,6}],
         [index-6,succ-{4,5}]])).

ctr_typical(bipartite,[size('NODES')>2]).

ctr_exchangeable(bipartite,[items('NODES',all)]).

ctr_graph(
    bipartite,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ],
    [],
    ['SYMMETRIC','BIPARTITE']).

ctr_application(bipartite,[1]).
```

B.59 calendar

◇ **META-DATA:**

```
ctr_predefined(calendar).

ctr_date(calendar,['20061014']).

ctr_origin(calendar,'\\cite{BeldiceanuR00}',[]).

ctr_types(
    calendar,
    ['UNAVAILABILITIES'-collection(low-int,up-int)]).

ctr_arguments(
    calendar,
    [INSTANTS-
     collection(
         machine-dvar,
         virtual-dvar,
         ireal-dvar,
         flagend-int),
     'MACHINES'-collection(id-int,cal-'UNAVAILABILITIES')]).

ctr_restrictions(
    calendar,
    [required('UNAVAILABILITIES',[low,up]),
     'UNAVAILABILITIES'^low=<'UNAVAILABILITIES'^up,
     required('INSTANTS',[machine,virtual,ireal,flagend]),
     in_attr('INSTANTS',machine,'MACHINES',id),
     'INSTANTS'^flagend>=0,
     'INSTANTS'^flagend=<1,
     size('MACHINES')>0,
     required('MACHINES',[id,cal]),
     distinct('MACHINES',id)]).

ctr_example(
    calendar,
    calendar(
        [[machine-1,virtual-2,ireal-3,flagend-0],
         [machine-1,virtual-5,ireal-6,flagend-1],
         [machine-2,virtual-4,ireal-5,flagend-0],
         [machine-2,virtual-6,ireal-9,flagend-1],
         [machine-3,virtual-2,ireal-2,flagend-0],
         [machine-3,virtual-5,ireal-5,flagend-1],
         [machine-4,virtual-2,ireal-2,flagend-0],
```

```
          [machine-4,virtual-7,ireal-9,flagend-1]],
         [[id-1,cal-[[low-2,up-2],[low-6,up-7]]],
          [id-2,cal-[[low-2,up-2],[low-6,up-7]]],
          [id-3,cal-[]],
          [id-4,cal-[[low-3,up-4]]]]])).

ctr_typical(calendar,[size('INSTANTS')>1,size('MACHINES')>1]).

ctr_exchangeable(
    calendar,
    [items('INSTANTS',all),items('MACHINES',all)]).

ctr_eval(calendar,[reformulation(calendar_r)]).

ctr_contractible(calendar,[],'INSTANTS',any).

ctr_application(calendar,[1]).

calendar_r(INSTANTS,MACHINES) :-
        collection(INSTANTS,[dvar,dvar,dvar,int(0,1)]),
        collection(MACHINES,[int,col([int,int])]),
        length(MACHINES,M),
        M>0,
        get_attr1(MACHINES,IDS),
        all_different(IDS),
        calendar_low_up(MACHINES),
        (   INSTANTS=[] ->
            true
        ;   calendar_in_attr(INSTANTS,IDS),
            calendar_normalize(MACHINES,MACHINESN),
            calendar_gen(INSTANTS,MACHINESN)
        ).

calendar_in_attr([],_33114).

calendar_in_attr([[_33123-M|_33121]|R],IDS) :-
        build_or_var_in_values(IDS,M,TERM),
        call(TERM),
        calendar_in_attr(R,IDS).

calendar_low_up([]).

calendar_low_up([[_33119,_33124-CAL]|R]) :-
        calendar_low_up1(CAL),
        calendar_low_up(R).
```

```
calendar_low_up1([]).

calendar_low_up1([[_33122-L,_33129-U]|R]) :-
        L=<U,
        calendar_low_up1(R).

calendar_normalize([],[]).

calendar_normalize(
    [[id-ID,cal-CAL]|R],
    [[id-ID,cal-MERGED_CAL]|S]) :-
        calendar_merge_intervals(CAL,MERGED_CAL),
        calendar_normalize(R,S).

calendar_merge_intervals(List,NewList) :-
        (foreach([low-L,up-U],List),fromto([],S1,S3,Set)do
         fdset_interval(S2,L,U),fdset_union(S1,S2,S3)),
        (foreach([A|B],Set),foreach([low-A,up-B],NewList)do
         true).

calendar_gen([],_33114).

calendar_gen(
    [[machine-M,virtual-V,ireal-R,flagend-F]|T],
    CALENDARS) :-
        calendar_gen(CALENDARS,M,V,R,F),
        calendar_gen(T,CALENDARS).

calendar_gen([],_33114,_33115,_33116,_33117).

calendar_gen([[id-I,cal-C]|S],M,V,R,F) :-
        calendar_gen(C,1,0,I,M,V,R,F),
        calendar_gen(S,M,V,R,F).

calendar_gen([],1,0,I,M,V,R,_F) :-
        M#=I#<=>M#=I#/\R#=V.

calendar_gen([[low-L,up-U]|S],1,0,I,M,V,R,F) :-
        LF is L+F,
        M#=I#/\R#<LF#<=>M#=I#/\R#=V,
        calendar_gen([[low-L,up-U]|S],0,0,I,M,V,R,F).

calendar_gen([[low-K,up-U],[low-L,up-W]|S],0,Sum,I,M,V,R,F) :-
        NSum is Sum+U-K+1,
        KF is K+F,
        UF is U+F,
```

```
        LF is L+F,
        R in KF..UF#=>M#\=I,
        M#=I#/\R#>UF#/\R#<LF#<=>M#=I#/\R#=V+NSum,
        calendar_gen([[low-L,up-W]|S],0,NSum,I,M,V,R,F).

calendar_gen([[low-L,up-U]],0,Sum,I,M,V,R,F) :-
        NSum is Sum+U-L+1,
        LF is L+F,
        UF is U+F,
        R in LF..UF#=>M#\=I,
        M#=I#/\R#>UF#<=>M#=I#/\R#=V+NSum.
```

B.60 cardinality_atleast

◇ **META-DATA:**

```
ctr_date(
    cardinality_atleast,
    ['20030820','20040530','20060805']).

ctr_origin(
    cardinality_atleast,
    Derived from %c.,
    [global_cardinality]).

ctr_arguments(
    cardinality_atleast,
    ['ATLEAST'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    cardinality_atleast,
    ['ATLEAST'>=0,
     'ATLEAST'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    cardinality_atleast,
    cardinality_atleast(
        1,
        [[var-3],[var-3],[var-8]],
        [[val-3],[val-8]])).

ctr_typical(
    cardinality_atleast,
    ['ATLEAST'>0,
     'ATLEAST'<size('VARIABLES'),
     size('VARIABLES')>1,
     size('VALUES')>0,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    cardinality_atleast,
    [items('VARIABLES',all),
     items('VALUES',all),
```

```
     vals(
         ['VARIABLES'^var],
         all(notin('VALUES'^val)),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    cardinality_atleast,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=\=values^val],
    ['MAX_ID'=size('VARIABLES')-'ATLEAST'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    cardinality_atleast,
    [reformulation(cardinality_atleast_r)]).

ctr_pure_functional_dependency(cardinality_atleast,[]).

ctr_functional_dependency(cardinality_atleast,1,[2,3]).

cardinality_atleast_r(ATLEAST,VARIABLES,VALUES) :-
        check_type(dvar,ATLEAST),
        ATLEAST#>=0,
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        ATLEAST#=<N,
        (   VALUES=[] ->
            true
        ;   collection(VALUES,[int]),
            length(VALUES,M),
            get_attr1(VARIABLES,VARS),
            get_attr1(VALUES,VALS),
            all_different(VALS),
            length(NOCCS,M),
            fd_min(ATLEAST,MIN_ATLEAST),
            domain(NOCCS,MIN_ATLEAST,N),
```

```
        get_minimum(VARS,MINVARS),
        get_maximum(VARS,MAXVARS),
        get_minimum(VALS,MINVALS),
        get_maximum(VALS,MAXVALS),
        MIN is min(MINVARS,MINVALS),
        MAX is max(MAXVARS,MAXVALS),
        complete_card(MIN,MAX,N,VALS,NOCCS,VN),
        global_cardinality(VARS,VN)
    ).
```

B.61 cardinality_atmost

◇ **META-DATA:**

```
ctr_date(cardinality_atmost,['20030820','20040530','20060805']).

ctr_origin(
    cardinality_atmost,
    Derived from %c.,
    [global_cardinality]).

ctr_arguments(
    cardinality_atmost,
    ['ATMOST'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    cardinality_atmost,
    ['ATMOST'>=0,
     'ATMOST'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    cardinality_atmost,
    cardinality_atmost(
        2,
        [[var-2],[var-1],[var-7],[var-1],[var-2]],
        [[val-5],[val-7],[val-2],[val-9]])).

ctr_typical(
    cardinality_atmost,
    ['ATMOST'>0,
     'ATMOST'<size('VARIABLES'),
     size('VARIABLES')>1,
     size('VALUES')>0,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    cardinality_atmost,
    [items('VARIABLES',all),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var],
```

```
         all(notin('VALUES'^val)),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    cardinality_atmost,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['MAX_ID'='ATMOST'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    cardinality_atmost,
    [reformulation(cardinality_atmost_r)]).

ctr_pure_functional_dependency(cardinality_atmost,[]).

ctr_functional_dependency(cardinality_atmost,1,[2,3]).

cardinality_atmost_r(ATMOST,VARIABLES,VALUES) :-
        check_type(dvar,ATMOST),
        ATMOST#>=0,
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        ATMOST#=<N,
        (   VALUES=[] ->
            true
        ;   collection(VALUES,[int]),
            length(VALUES,M),
            get_attr1(VARIABLES,VARS),
            get_attr1(VALUES,VALS),
            all_different(VALS),
            length(NOCCS,M),
            fd_max(ATMOST,MAX_ATMOST),
            domain(NOCCS,0,MAX_ATMOST),
            get_minimum(VARS,MINVARS),
            get_maximum(VARS,MAXVARS),
```

```
    get_minimum(VALS,MINVALS),
    get_maximum(VALS,MAXVALS),
    MIN is min(MINVARS,MINVALS),
    MAX is max(MAXVARS,MAXVALS),
    complete_card(MIN,MAX,N,VALS,NOCCS,VN),
    global_cardinality(VARS,VN)
).
```

B.62 cardinality_atmost_partition

◇ META-DATA:

```
ctr_date(cardinality_atmost_partition,['20030820','20060805']).

ctr_origin(
    cardinality_atmost_partition,
    Derived from %c.,
    [global_cardinality]).

ctr_types(
    cardinality_atmost_partition,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    cardinality_atmost_partition,
    ['ATMOST'-dvar,
     'VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    cardinality_atmost_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     'ATMOST'>=0,
     'ATMOST'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    cardinality_atmost_partition,
    cardinality_atmost_partition(
        2,
        [[var-2],[var-3],[var-7],[var-1],[var-6],[var-0]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(
    cardinality_atmost_partition,
    ['ATMOST'>0,
     'ATMOST'<size('VARIABLES'),
     size('VARIABLES')>1,
```

```
     size('VARIABLES')>size('PARTITIONS')]).

ctr_exchangeable(
    cardinality_atmost_partition,
    [items('VARIABLES',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all)]).

ctr_graph(
    cardinality_atmost_partition,
    ['VARIABLES','PARTITIONS'],
    2,
    ['PRODUCT'>>collection(variables,partitions)],
    [variables^var in partitions^p],
    ['MAX_ID'='ATMOST'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    cardinality_atmost_partition,
    [reformulation(cardinality_atmost_partition_r)]).

ctr_pure_functional_dependency(cardinality_atmost_partition,[]).

ctr_functional_dependency(cardinality_atmost_partition,1,[2,3]).

cardinality_atmost_partition_r(ATMOST,VARIABLES,PARTITIONS) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        check_type(dvar(0,N),ATMOST),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        length(PARTITIONS,P),
        P>1,
        get_attr1(VARIABLES,VARS),
        get_col_attr1(PARTITIONS,1,PVALS),
        flattern(PVALS,VALS),
        all_different(VALS),
        length(PVALS,LPVALS),
        LPVALS1 is LPVALS+1,
        get_partition_var(VARS,PVALS,PVARS,LPVALS1,0),
        complete_card_consec(1,LPVALS1,ATMOST,N,VALUES),
        global_cardinality(PVARS,VALUES).
```

B.63 change

♢ META-DATA:

```
ctr_date(change,['20000128','20030820','20040530','20060805']).

ctr_origin(change,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(change,[nbchanges,similarity]).

ctr_arguments(
    change,
    ['NCHANGE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    change,
    ['NCHANGE'>=0,
     'NCHANGE'<size('VARIABLES'),
     required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    change,
    [change(3,[[var-4],[var-4],[var-3],[var-4],[var-1]],=\=),
     change(1,[[var-1],[var-2],[var-4],[var-3],[var-7]],>)]).

ctr_typical(
    change,
    ['NCHANGE'>0,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=\=])]).

ctr_exchangeable(change,[translate(['VARIABLES'^var])]).

ctr_graph(
    change,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'(variables1^var,variables2^var)],
    ['NARC'='NCHANGE'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).
```

```
ctr_eval(change,[checker(change_c),automaton(change_a)]).

ctr_pure_functional_dependency(change,[]).

ctr_functional_dependency(change,1,[2,3]).

ctr_contractible(
    change,
    [in_list('CTR',[=\=,<,>=,>,=<]),'NCHANGE'=0],
    VARIABLES,
    any).

ctr_contractible(
    change,
    [in_list('CTR',[=,<,>=,>,=<]),
     'NCHANGE'=size('VARIABLES')-1],
    VARIABLES,
    any).

change_c(NCHANGE,VARIABLES,=) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        (   N=<1 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            change_eq_c(VARS,0,NCHANGE)
        ).

change_c(NCHANGE,VARIABLES,=\=) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        (   N=<1 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            change_neq_c(VARS,0,NCHANGE)
        ).

change_c(NCHANGE,VARIABLES,<) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
```

```
        (   N=<1 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            change_lt_c(VARS,0,NCHANGE)
        ).

change_c(NCHANGE,VARIABLES,>=) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        (   N=<1 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            change_geq_c(VARS,0,NCHANGE)
        ).

change_c(NCHANGE,VARIABLES,>) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        (   N=<1 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            change_gt_c(VARS,0,NCHANGE)
        ).

change_c(NCHANGE,VARIABLES,=<) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        (   N=<1 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            change_leq_c(VARS,0,NCHANGE)
        ).

change_neq_counters_check([V,V|R],C,[C|S]) :-
        !,
        change_neq_counters_check([V|R],C,S).

change_neq_counters_check([_50925,V|R],C,[C1|S]) :-
        !,
        C1 is C+1,
```

```
        change_neq_counters_check([V|R],C1,S).

change_neq_counters_check(_50919,_50920,[0]).

change_a(FLAG,NCHANGE,VARIABLES,CTR) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        fd_min(NCHANGE,MIN_NCHANGE),
        fd_max(NCHANGE,MAX_NCHANGE),
        (   FLAG=1,
            MIN_NCHANGE=0,
            MAX_NCHANGE=0,
            memberchk(CTR,[=\=]) ->
            eval(all_equal(VARIABLES))
        ;   change_signature(VARIABLES,SIGNATURE,CTR),
            automaton(
                SIGNATURE,
                _53865,
                SIGNATURE,
                [source(s),sink(s)],
                [arc(s,0,s),arc(s,1,s,[C+1])],
                [C],
                [0],
                [COUNT]),
            NCHANGE#=COUNT#<=>FLAG
        ).

change_signature([],[],_50921).

change_signature([_50925],[],_50924) :-
        !.

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],=) :-
        !,
        VAR1#=VAR2#<=>S,
        change_signature([[var-VAR2]|VARs],Ss,=).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],=\=) :-
        !,
        VAR1#\=VAR2#<=>S,
        change_signature([[var-VAR2]|VARs],Ss,=\=).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],<) :-
```

```
        !,
        VAR1#<VAR2#<=>S,
        change_signature([[var-VAR2]|VARs],Ss,<).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],>=) :-
        !,
        VAR1#>=VAR2#<=>S,
        change_signature([[var-VAR2]|VARs],Ss,>=).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],>) :-
        !,
        VAR1#>VAR2#<=>S,
        change_signature([[var-VAR2]|VARs],Ss,>).

change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],=<) :-
        !,
        VAR1#=<VAR2#<=>S,
        change_signature([[var-VAR2]|VARs],Ss,=<).
```

B.64 change_continuity

◇ **META-DATA:**

```
ctr_date(
    change_continuity,
    ['20000128','20030820','20040530','20060805']).

ctr_origin(change_continuity,'N.~Beldiceanu',[]).

ctr_arguments(
    change_continuity,
    ['NB_PERIOD_CHANGE'-dvar,
     'NB_PERIOD_CONTINUITY'-dvar,
     'MIN_SIZE_CHANGE'-dvar,
     'MAX_SIZE_CHANGE'-dvar,
     'MIN_SIZE_CONTINUITY'-dvar,
     'MAX_SIZE_CONTINUITY'-dvar,
     'NB_CHANGE'-dvar,
     'NB_CONTINUITY'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    change_continuity,
    ['NB_PERIOD_CHANGE'>=0,
     'NB_PERIOD_CONTINUITY'>=0,
     'MIN_SIZE_CHANGE'>=0,
     'MAX_SIZE_CHANGE'>='MIN_SIZE_CHANGE',
     'MIN_SIZE_CONTINUITY'>=0,
     'MAX_SIZE_CONTINUITY'>='MIN_SIZE_CONTINUITY',
     'NB_CHANGE'>=0,
     'NB_CONTINUITY'>=0,
     required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    change_continuity,
    change_continuity(
        3,
        2,
        2,
        4,
        2,
        4,
        6,
```

```
        4,
        [[var-1],
         [var-3],
         [var-1],
         [var-8],
         [var-8],
         [var-4],
         [var-7],
         [var-7],
         [var-7],
         [var-7],
         [var-2]],
        =\=)).

ctr_typical(
    change_continuity,
    ['NB_PERIOD_CHANGE'>0,
     'NB_PERIOD_CONTINUITY'>0,
     'MIN_SIZE_CHANGE'>0,
     'MIN_SIZE_CONTINUITY'>0,
     'NB_CHANGE'>0,
     'NB_CONTINUITY'>0,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=\=])]).

ctr_exchangeable(
    change_continuity,
    [translate(['VARIABLES'^var])]).

ctr_graph(
    change_continuity,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'(variables1^var,variables2^var)],
    ['NCC'='NB_PERIOD_CHANGE',
     'MIN_NCC'='MIN_SIZE_CHANGE',
     'MAX_NCC'='MAX_SIZE_CHANGE',
     'NARC'='NB_CHANGE'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_graph(
    change_continuity,
    ['VARIABLES'],
    2,
```

```
    ['PATH'>>collection(variables1,variables2)],
    [#\'CTR'(variables1^var,variables2^var)],
    ['NCC'='NB_PERIOD_CONTINUITY',
     'MIN_NCC'='MIN_SIZE_CONTINUITY',
     'MAX_NCC'='MAX_SIZE_CONTINUITY',
     'NARC'='NB_CONTINUITY'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    change_continuity,
    [checker(change_continuity_c),
     automata(change_continuity_a)]).

ctr_functional_dependency(change_continuity,1,[9,10]).

ctr_functional_dependency(change_continuity,2,[9,10]).

ctr_functional_dependency(change_continuity,3,[9,10]).

ctr_functional_dependency(change_continuity,4,[9,10]).

ctr_functional_dependency(change_continuity,5,[9,10]).

ctr_functional_dependency(change_continuity,6,[9,10]).

ctr_functional_dependency(change_continuity,7,[9,10]).

ctr_functional_dependency(change_continuity,8,[9,10]).

change_continuity_a(
    NB_PERIOD_CHANGE,
    NB_PERIOD_CONTINUITY,
    MIN_SIZE_CHANGE,
    MAX_SIZE_CHANGE,
    MIN_SIZE_CONTINUITY,
    MAX_SIZE_CONTINUITY,
    NB_CHANGE,
    NB_CONTINUITY,
    VARIABLES,
    CTR) :-
        check_type(dvar,NB_PERIOD_CHANGE),
        check_type(dvar,NB_PERIOD_CONTINUITY),
        check_type(dvar,MIN_SIZE_CHANGE),
        check_type(dvar,MAX_SIZE_CHANGE),
        check_type(dvar,MIN_SIZE_CONTINUITY),
        check_type(dvar,MAX_SIZE_CONTINUITY),
```

```
check_type(dvar,NB_CHANGE),
check_type(dvar,NB_CONTINUITY),
memberchk(CTR,[=,=\=,<,>=,>,=<]),
length(VARIABLES,N),
(   N=0 ->
    NB_PERIOD_CHANGE#=0,
    NB_PERIOD_CONTINUITY#=0,
    MIN_SIZE_CHANGE#=0,
    MAX_SIZE_CHANGE#=0,
    MIN_SIZE_CONTINUITY#=0,
    MAX_SIZE_CONTINUITY#=0,
    NB_CHANGE#=0,
    NB_CONTINUITY#=0
;   collection(VARIABLES,[dvar]),
    NB_PERIOD_CHANGE#>=0,
    NB_PERIOD_CONTINUITY#>=0,
    MIN_SIZE_CHANGE#>=0,
    MAX_SIZE_CHANGE#>=MIN_SIZE_CHANGE,
    MIN_SIZE_CONTINUITY#>=0,
    MAX_SIZE_CONTINUITY#>=MIN_SIZE_CONTINUITY,
    NB_CHANGE#>=0,
    NB_CONTINUITY#>=0,
    change_continuity_signature(
        VARIABLES,
        SIGNATURE_CTR,
        1,
        CTR),
    change_continuity_signature(
        VARIABLES,
        SIGNATURE_NOT_CTR,
        0,
        CTR),
    change_continuity_nb_period(
        NB_PERIOD_CHANGE,
        SIGNATURE_CTR),
    change_continuity_nb_period(
        NB_PERIOD_CONTINUITY,
        SIGNATURE_NOT_CTR),
    change_continuity_min_size(
        MIN_SIZE_CHANGE,
        SIGNATURE_CTR),
    change_continuity_min_size(
        MIN_SIZE_CONTINUITY,
        SIGNATURE_NOT_CTR),
    change_continuity_max_size(
        MAX_SIZE_CHANGE,
```

```
                SIGNATURE_CTR),
            change_continuity_max_size(
                MAX_SIZE_CONTINUITY,
                SIGNATURE_NOT_CTR),
            change_continuity_nb(NB_CHANGE,SIGNATURE_CTR),
            change_continuity_nb(
                NB_CONTINUITY,
                SIGNATURE_NOT_CTR)
        ).

change_continuity_nb_period(NB_PERIOD,SIGNATURE) :-
        automaton(
            SIGNATURE,
            _59551,
            SIGNATURE,
            [source(s),sink(s),sink(i)],
            [arc(s,0,s),
             arc(s,1,i,[C+1]),
             arc(i,1,i),
             arc(i,0,s)],
            [C],
            [0],
            [NB_PERIOD]).

change_continuity_min_size(MIN_SIZE,SIGNATURE) :-
        MIN_SIZE#=min(C1,D1),
        length(SIGNATURE,N),
        N1 is N+1,
        automaton(
            SIGNATURE,
            _60200,
            SIGNATURE,
            [source(s),sink(s),sink(i)],
            [arc(s,0,s),
             arc(s,1,i,[C,2]),
             arc(i,0,s,[min(C,D),D]),
             arc(i,1,i,[C,D+1])],
            [C,D],
            [N1,0],
            [C1,D1]).

change_continuity_max_size(MAX_SIZE,SIGNATURE) :-
        MAX_SIZE#=max(C1,D1),
        automaton(
            SIGNATURE,
            _59833,
```

```
            SIGNATURE,
            [source(s),sink(i),sink(s)],
            [arc(s,0,s,[C,D]),
             arc(s,1,i,[C,2]),
             arc(i,0,i,[max(C,D),1]),
             arc(i,1,i,[C,D+1])],
            [C,D],
            [0,0],
            [C1,D1]).

change_continuity_nb(NB,SIGNATURE) :-
        automaton(
            SIGNATURE,
            _59509,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [NB]).

change_continuity_signature([],[],_58906,_58907).

change_continuity_signature([_58911],[],_58909,_58910) :-
        !.

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    1,
    =) :-
        !,
        VAR1#=VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,1,=).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    1,
    =\=) :-
        !,
        VAR1#\=VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,1,=\=).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
```

```
    [S|Ss],
    1,
    <) :-
        !,
        VAR1#<VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,1,<).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    1,
    >=) :-
        !,
        VAR1#>=VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,1,>=).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    1,
    >) :-
        !,
        VAR1#>VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,1,>).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    1,
    =<) :-
        !,
        VAR1#=<VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,1,=<).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    0,
    =) :-
        !,
        VAR1#\=VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,0,=).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
```

```
    0,
    =\=) :-
        !,
        VAR1#=VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,0,=\=).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    0,
    <) :-
        !,
        VAR1#>=VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,0,<).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    0,
    >=) :-
        !,
        VAR1#<VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,0,>=).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    0,
    >) :-
        !,
        VAR1#=<VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,0,>).

change_continuity_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    0,
    =<) :-
        !,
        VAR1#>VAR2#<=>S,
        change_continuity_signature([[var-VAR2]|VARs],Ss,0,=<).

change_continuity_c(
    NB_PERIOD_CHANGE,
    NB_PERIOD_CONTINUITY,
    MIN_SIZE_CHANGE,
```

```
MAX_SIZE_CHANGE,
MIN_SIZE_CONTINUITY,
MAX_SIZE_CONTINUITY,
NB_CHANGE,
NB_CONTINUITY,
VARIABLES,
CTR) :-
    check_type(dvar,NB_PERIOD_CHANGE),
    check_type(dvar,NB_PERIOD_CONTINUITY),
    check_type(dvar,MIN_SIZE_CHANGE),
    check_type(dvar,MAX_SIZE_CHANGE),
    check_type(dvar,MIN_SIZE_CONTINUITY),
    check_type(dvar,MAX_SIZE_CONTINUITY),
    check_type(dvar,NB_CHANGE),
    check_type(dvar,NB_CONTINUITY),
    memberchk(CTR,[=,=\=,<,>=,>,=<]),
    length(VARIABLES,N),
    (   N=0 ->
        NB_PERIOD_CHANGE#=0,
        NB_PERIOD_CONTINUITY#=0,
        MIN_SIZE_CHANGE#=0,
        MAX_SIZE_CHANGE#=0,
        MIN_SIZE_CONTINUITY#=0,
        MAX_SIZE_CONTINUITY#=0,
        NB_CHANGE#=0,
        NB_CONTINUITY#=0
    ;   collection(VARIABLES,[int]),
        NB_PERIOD_CHANGE#>=0,
        NB_PERIOD_CONTINUITY#>=0,
        MIN_SIZE_CHANGE#>=0,
        MAX_SIZE_CHANGE#>=MIN_SIZE_CHANGE,
        MIN_SIZE_CONTINUITY#>=0,
        MAX_SIZE_CONTINUITY#>=MIN_SIZE_CONTINUITY,
        NB_CHANGE#>=0,
        NB_CONTINUITY#>=0,
        change_continuity_signature_c(
            CTR,
            VARIABLES,
            SIGNATURE_CTR,
            SIGNATURE_NOT_CTR),
        (   SIGNATURE_CTR=[] ->
            NB_PERIOD_CONTINUITY is 0,
            NB_PERIOD_CHANGE is 0
        ;   change_continuity_nb_period_c(
                s,
                SIGNATURE_CTR,
```

```
                    0,
                    NB_PERIOD_CHANGE),
                SIGNATURE_CTR=[First|_59169],
                last(SIGNATURE_CTR,Last),
                (   First=0,
                    Last=0 ->
                    NB_PERIOD_CONTINUITY is NB_PERIOD_CHANGE+1
                ;   First=1,
                    Last=1 ->
                    NB_PERIOD_CONTINUITY is NB_PERIOD_CHANGE-1
                ;   NB_PERIOD_CONTINUITY is NB_PERIOD_CHANGE
                )
            ),
            change_continuity_min_size_c(
                s,
                SIGNATURE_CTR,
                N,
                0,
                MIN_SIZE_CHANGE),
            change_continuity_min_size_c(
                s,
                SIGNATURE_NOT_CTR,
                N,
                0,
                MIN_SIZE_CONTINUITY),
            change_continuity_max_size_c(
                s,
                SIGNATURE_CTR,
                0,
                0,
                MAX_SIZE_CHANGE),
            change_continuity_max_size_c(
                s,
                SIGNATURE_NOT_CTR,
                0,
                0,
                MAX_SIZE_CONTINUITY),
            change_continuity_nb_c(SIGNATURE_CTR,0,NB_CHANGE),
            NB_CONTINUITY is N-NB_CHANGE-1
        ).

change_continuity_nb_period_c(s,[0|R],C,NB_PERIOD_CHANGE) :-
        !,
        change_continuity_nb_period_c(s,R,C,NB_PERIOD_CHANGE).

change_continuity_nb_period_c(s,[1|R],C,NB_PERIOD_CHANGE) :-
```

```
        !,
        C1 is C+1,
        change_continuity_nb_period_c(i,R,C1,NB_PERIOD_CHANGE).

change_continuity_nb_period_c(i,[1|R],C,NB_PERIOD_CHANGE) :-
        !,
        change_continuity_nb_period_c(i,R,C,NB_PERIOD_CHANGE).

change_continuity_nb_period_c(i,[0|R],C,NB_PERIOD_CHANGE) :-
        !,
        change_continuity_nb_period_c(s,R,C,NB_PERIOD_CHANGE).

change_continuity_nb_period_c(_58904,[],C,C).

change_continuity_min_size_c(s,[0|R],C,D,MIN_SIZE_CHANGE) :-
        !,
        change_continuity_min_size_c(s,R,C,D,MIN_SIZE_CHANGE).

change_continuity_min_size_c(s,[1|R],C,_58910,MIN_SIZE_CHANGE) :-
        !,
        change_continuity_min_size_c(i,R,C,2,MIN_SIZE_CHANGE).

change_continuity_min_size_c(i,[0|R],C,D,MIN_SIZE_CHANGE) :-
        !,
        C1 is min(C,D),
        (   C1>1 ->
            change_continuity_min_size_c(
                s,
                R,
                C1,
                D,
                MIN_SIZE_CHANGE)
        ;   MIN_SIZE_CHANGE#=C1
        ).

change_continuity_min_size_c(i,[1|R],C,D,MIN_SIZE_CHANGE) :-
        !,
        D1 is D+1,
        change_continuity_min_size_c(i,R,C,D1,MIN_SIZE_CHANGE).

change_continuity_min_size_c(_58907,[],C,D,MIN_SIZE_CHANGE) :-
        MIN is min(C,D),
        MIN_SIZE_CHANGE#=MIN.

change_continuity_max_size_c(s,[0|R],C,D,MAX_SIZE_CHANGE) :-
        !,
```

```
        change_continuity_max_size_c(s,R,C,D,MAX_SIZE_CHANGE).

change_continuity_max_size_c(s,[1|R],C,_D,MAX_SIZE_CHANGE) :-
        !,
        D1 is 2,
        change_continuity_max_size_c(i,R,C,D1,MAX_SIZE_CHANGE).

change_continuity_max_size_c(i,[0|R],C,D,MAX_SIZE_CHANGE) :-
        !,
        C1 is max(C,D),
        change_continuity_max_size_c(i,R,C1,1,MAX_SIZE_CHANGE).

change_continuity_max_size_c(i,[1|R],C,D,MAX_SIZE_CHANGE) :-
        !,
        D1 is D+1,
        change_continuity_max_size_c(i,R,C,D1,MAX_SIZE_CHANGE).

change_continuity_max_size_c(_58907,[],C,D,MAX_SIZE_CHANGE) :-
        MAX is max(C,D),
        MAX_SIZE_CHANGE#=MAX.

change_continuity_nb_c([B|R],C,NB_CHANGE) :-
        !,
        C1 is C+B,
        change_continuity_nb_c(R,C1,NB_CHANGE).

change_continuity_nb_c([],C,C).

change_continuity_signature_c(
    =,
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    [R|Rs]) :-
        !,
        (   VAR1=VAR2 ->
            S=1,
            R=0
        ;   S=0,
            R=1
        ),
        change_continuity_signature_c(
            =,
            [[var-VAR2]|VARs],
            Ss,
            Rs).
```

```
change_continuity_signature_c(
    =\=,
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    [R|Rs]) :-
        !,
        (   VAR1=\=VAR2 ->
            S=1,
            R=0
        ;   S=0,
            R=1
        ),
        change_continuity_signature_c(
            =\=,
            [[var-VAR2]|VARs],
            Ss,
            Rs).

change_continuity_signature_c(
    <,
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    [R|Rs]) :-
        !,
        (   VAR1<VAR2 ->
            S=1,
            R=0
        ;   S=0,
            R=1
        ),
        change_continuity_signature_c(
            <,
            [[var-VAR2]|VARs],
            Ss,
            Rs).

change_continuity_signature_c(
    >=,
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    [R|Rs]) :-
        !,
        (   VAR1>=VAR2 ->
            S=1,
            R=0
        ;   S=0,
```

```
            R=1
        ),
        change_continuity_signature_c(
            >=,
            [[var-VAR2]|VARs],
            Ss,
            Rs).

change_continuity_signature_c(
    >,
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    [R|Rs]) :-
        !,
        (   VAR1>VAR2 ->
            S=1,
            R=0
        ;   S=0,
            R=1
        ),
        change_continuity_signature_c(
            >,
            [[var-VAR2]|VARs],
            Ss,
            Rs).

change_continuity_signature_c(
    =<,
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    [R|Rs]) :-
        !,
        (   VAR1=<VAR2 ->
            S=1,
            R=0
        ;   S=0,
            R=1
        ),
        change_continuity_signature_c(
            =<,
            [[var-VAR2]|VARs],
            Ss,
            Rs).

change_continuity_signature_c(_58907,[_58911],[],[]) :-
        !.
```

```
change_continuity_signature_c(_58904,[],[],[]).
```

B.65 change_pair

◇ **META-DATA:**

```
ctr_date(change_pair,['20030820','20040530','20060805']).

ctr_origin(change_pair,'Derived from %c.',[change]).

ctr_arguments(
    change_pair,
    ['NCHANGE'-dvar,
     'PAIRS'-collection(x-dvar,y-dvar),
     'CTRX'-atom,
     'CTRY'-atom]).

ctr_restrictions(
    change_pair,
    ['NCHANGE'>=0,
     'NCHANGE'<size('PAIRS'),
     required('PAIRS',[x,y]),
     in_list('CTRX',[=,=\=,<,>=,>,=<]),
     in_list('CTRY',[=,=\=,<,>=,>,=<])]).

ctr_example(
    change_pair,
    change_pair(
        3,
        [[x-3,y-5],
         [x-3,y-7],
         [x-3,y-7],
         [x-3,y-8],
         [x-3,y-4],
         [x-3,y-7],
         [x-1,y-3],
         [x-1,y-6],
         [x-1,y-6],
         [x-3,y-7]],
        =\=,
        >)).

ctr_typical(
    change_pair,
    ['NCHANGE'>0,
     size('PAIRS')>1,
     range('PAIRS'^x)>1,
     range('PAIRS'^y)>1]).
```

```
ctr_exchangeable(
    change_pair,
    [translate(['PAIRS'^x]),translate(['PAIRS'^y])]).

ctr_graph(
    change_pair,
    ['PAIRS'],
    2,
    ['PATH'>>collection(pairs1,pairs2)],
    ['CTRX'(pairs1^x,pairs2^x)#\/'CTRY'(pairs1^y,pairs2^y)],
    ['NARC'='NCHANGE'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(change_pair,[automaton(change_pair_a)]).

ctr_pure_functional_dependency(change_pair,[]).

ctr_functional_dependency(change_pair,1,[2,3,4]).

ctr_application(change_pair,[2]).

change_pair_a(FLAG,NCHANGE,PAIRS,CTRX,CTRY) :-
        collection(PAIRS,[dvar,dvar]),
        length(PAIRS,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        memberchk(CTRX,[=,=\=,<,>=,>,=<]),
        memberchk(CTRY,[=,=\=,<,>=,>,=<]),
        change_pair_signature(PAIRS,SIGNATURE,CTRX,CTRY),
        automaton(
            SIGNATURE,
            _47739,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NCHANGE#<=>FLAG.

change_pair_signature([],[],_45846,_45847).

change_pair_signature([_45851],[],_45849,_45850) :-
        !.
```

```
change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =,
    =) :-
        !,
        X1#=X2#\/Y1#=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =,
    =\=) :-
        !,
        X1#=X2#\/Y1#\=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,=\=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =,
    <) :-
        !,
        X1#=X2#\/Y1#<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =,
    >=) :-
        !,
        X1#=X2#\/Y1#>=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,>=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =,
    >) :-
        !,
        X1#=X2#\/Y1#>Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,>).

change_pair_signature(
```

```
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =,
    =<) :-
        !,
        X1#=X2#\/Y1#=<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=,=<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    =) :-
        !,
        X1#\=X2#\/Y1#=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=\=,=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    =\=) :-
        !,
        X1#\=X2#\/Y1#\=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=\=,=\=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    <) :-
        !,
        X1#\=X2#\/Y1#<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=\=,<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    >=) :-
        !,
        X1#\=X2#\/Y1#>=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=\=,>=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
```

```
    [S|Ss],
    =\=,
    >) :-
        !,
        X1#\=X2#\/Y1#>Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=\=,>).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =\=,
    =<) :-
        !,
        X1#\=X2#\/Y1#=<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=\=,=<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    <,
    =) :-
        !,
        X1#<X2#\/Y1#=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,<,=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    <,
    =\=) :-
        !,
        X1#<X2#\/Y1#\=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,<,=\=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    <,
    <) :-
        !,
        X1#<X2#\/Y1#<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,<,<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
```

```
    <,
    >=) :-
        !,
        X1#<X2#\/Y1#>=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,<,>=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    <,
    >) :-
        !,
        X1#<X2#\/Y1#>Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,<,>).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    <,
    =<) :-
        !,
        X1#<X2#\/Y1#=<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,<,=<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >=,
    =) :-
        !,
        X1#>=X2#\/Y1#=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>=,=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >=,
    =\=) :-
        !,
        X1#>=X2#\/Y1#\=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>=,=\=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >=,
```

```
    <) :-
        !,
        X1#>=X2#\/Y1#<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>=,<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >=,
    >=) :-
        !,
        X1#>=X2#\/Y1#>=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>=,>=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >=,
    >) :-
        !,
        X1#>=X2#\/Y1#>Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>=,>).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >=,
    =<) :-
        !,
        X1#>=X2#\/Y1#=<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>=,=<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >,
    =) :-
        !,
        X1#>X2#\/Y1#=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>,=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >,
    =\=) :-
```

```
        !,
        X1#>X2#\/Y1#\=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>,=\=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >,
    <) :-
        !,
        X1#>X2#\/Y1#<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>,<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >,
    >=) :-
        !,
        X1#>X2#\/Y1#>=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>,>=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >,
    >) :-
        !,
        X1#>X2#\/Y1#>Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>,>).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    >,
    =<) :-
        !,
        X1#>X2#\/Y1#=<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,>,=<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    =) :-
        !,
```

```
        X1#=<X2#\/Y1#=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=<,=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    =\=) :-
        !,
        X1#=<X2#\/Y1#\=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=<,=\=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    <) :-
        !,
        X1#=<X2#\/Y1#<Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=<,<).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    >=) :-
        !,
        X1#=<X2#\/Y1#>=Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=<,>=).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    >) :-
        !,
        X1#=<X2#\/Y1#>Y2#<=>S,
        change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=<,>).

change_pair_signature(
    [[x-X1,y-Y1],[x-X2,y-Y2]|PAIRs],
    [S|Ss],
    =<,
    =<) :-
        !,
        X1#=<X2#\/Y1#=<Y2#<=>S,
```

```
change_pair_signature([[x-X2,y-Y2]|PAIRs],Ss,=<,=<).
```

B.66 change_partition

◇ META-DATA:

```
ctr_date(
    change_partition,
    ['20000128','20030820','20040530','20060805']).

ctr_origin(change_partition,'Derived from %c.',[change]).

ctr_types(change_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    change_partition,
    ['NCHANGE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    change_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     'NCHANGE'>=0,
     'NCHANGE'<size('VARIABLES'),
     required('VARIABLES',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    change_partition,
    change_partition(
        2,
        [[var-6],
         [var-6],
         [var-2],
         [var-1],
         [var-3],
         [var-3],
         [var-1],
         [var-6],
         [var-2],
         [var-2],
         [var-2]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
```

```
        [p-[[val-2],[val-6]]]])).

ctr_typical(
    change_partition,
    ['NCHANGE'>0,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VARIABLES')>size('PARTITIONS')]).

ctr_exchangeable(
    change_partition,
    [items('VARIABLES',reverse),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['VARIABLES'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    change_partition,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    ['NARC'='NCHANGE'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_pure_functional_dependency(change_partition,[]).

ctr_functional_dependency(change_partition,1,[2,3]).
```

B.67 change_vectors

◊ **META-DATA:**

```
ctr_date(change_vectors,['20110616']).

ctr_origin(change_vectors,'Derived from %c',[change]).

ctr_types(
    change_vectors,
    ['VECTOR'-collection(var-dvar),'CTR'-atom]).

ctr_arguments(
    change_vectors,
    ['NCHANGE'-dvar,
     'VECTORS'-collection(vec-'VECTOR'),
     'CTRS'-collection(ctr-'CTR')]).

ctr_restrictions(
    change_vectors,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     in_list('CTR',[=,=\=,<,>=,>,=<]),
     'NCHANGE'>=0,
     'NCHANGE'<size('VECTORS'),
     required('VECTORS',vec),
     same_size('VECTORS',vec),
     required('CTRS',ctr),
     size('CTRS')=size('VECTOR')]).

ctr_example(
    change_vectors,
    change_vectors(
        3,
        [[vec-[[var-4],[var-0]]],
         [vec-[[var-4],[var-0]]],
         [vec-[[var-4],[var-5]]],
         [vec-[[var-3],[var-4]]],
         [vec-[[var-3],[var-4]]],
         [vec-[[var-3],[var-4]]],
         [vec-[[var-4],[var-0]]]],
        [[ctr- =\=],[ctr- =\=]])).

ctr_typical(
    change_vectors,
    [in_list('CTR',[=\=]),
```

```
    size('VECTOR')>1,
    'NCHANGE'>0,
    size('VECTORS')>1]).

ctr_eval(change_vectors,[automaton(change_vectors_a)]).

ctr_pure_functional_dependency(change_vectors,[]).

ctr_functional_dependency(change_vectors,1,[2,3]).

change_vectors_a(FLAG,NCHANGE,VECTORS,CTRS) :-
        collection(VECTORS,[col([dvar])]),
        length(VECTORS,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        collection(CTRS,[atom([=,=\=,<,>=,>,=<])]),
        same_size(VECTORS),
        length(CTRS,M),
        VECTORS=[[_27261-VECTOR1]|_27257],
        length(VECTOR1,M),
        M>=1,
        get_attr11(VECTORS,VECTS),
        get_attr1(CTRS,LCTRS),
        change_vectors_signature(VECTS,SIGNATURE,LCTRS),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _30416,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [NCHANGE]),
        automaton_bool(FLAG,[0,1],AUTOMATON).

change_vectors_signature([],[],_27165) :-
        !.

change_vectors_signature([_27166],[],_27165) :-
        !.

change_vectors_signature([VEC1,VEC2|VECs],[S|Ss],CTRS) :-
        !,
        build_vectors_compare_change(VEC1,VEC2,CTRS,Term),
        call(Term#<=>S),
```

```
change_vectors_signature([VEC2|VECs],Ss,CTRS).
```

B.68 circuit

◇ **META-DATA:**

```
ctr_date(circuit,['20030820','20040530','20060805']).

ctr_origin(circuit,'\\cite{Lauriere78}',[]).

ctr_synonyms(circuit,[atour,cycle]).

ctr_arguments(
    circuit,
    ['NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    circuit,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    circuit,
    circuit(
        [[index-1,succ-2],
         [index-2,succ-3],
         [index-3,succ-4],
         [index-4,succ-1]])).

ctr_typical(circuit,[size('NODES')>2]).

ctr_exchangeable(circuit,[items('NODES',all)]).

ctr_graph(
    circuit,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['MIN_NSCC'=size('NODES'),'MAX_ID'=<1],
    ['ONE_SUCC']).

ctr_eval(circuit,[builtin(circuit_b)]).
```

```
ctr_cond_imply(circuit,cycle,[],['NCYCLE'=1],[none,'NODES']).

ctr_cond_imply(circuit,derangement,[size('NODES')>1],[],id).

ctr_cond_imply(
    circuit,
    k_alldifferent,
    [size('NODES')>1],
    [],
    same).

ctr_cond_imply(circuit,permutation,[],[],index_to_col).

ctr_application(circuit,[1]).

ctr_sol(circuit,2,0,2,1,-).

ctr_sol(circuit,3,0,3,2,-).

ctr_sol(circuit,4,0,4,6,-).

ctr_sol(circuit,5,0,5,24,-).

ctr_sol(circuit,6,0,6,120,-).

ctr_sol(circuit,7,0,7,720,-).

ctr_sol(circuit,8,0,8,5040,-).

ctr_sol(circuit,9,0,9,40320,-).

ctr_sol(circuit,10,0,10,362880,-).

circuit_b(NODES) :-
        length(NODES,N),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,INDEX),
        all_different(INDEX),
        sort_collection(NODES,index,SORTED_NODES),
        get_attr2(SORTED_NODES,SUCC),
        circuit(SUCC).
```

B.69 circuit_cluster

◊ **META-DATA:**

```
ctr_date(circuit_cluster,['20000128','20030820','20060805']).

ctr_origin(
    circuit_cluster,
    Inspired by \cite{LaporteAsefVaziriSriskandarajah96}.,
    []).

ctr_arguments(
    circuit_cluster,
    ['NCIRCUIT'-dvar,
     'NODES'-collection(index-int,cluster-int,succ-dvar)]).

ctr_restrictions(
    circuit_cluster,
    ['NCIRCUIT'>=1,
     'NCIRCUIT'=<size('NODES'),
     required('NODES',[index,cluster,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    circuit_cluster,
    [circuit_cluster(
         1,
         [[index-1,cluster-1,succ-1],
          [index-2,cluster-1,succ-4],
          [index-3,cluster-2,succ-3],
          [index-4,cluster-2,succ-5],
          [index-5,cluster-3,succ-8],
          [index-6,cluster-3,succ-6],
          [index-7,cluster-3,succ-7],
          [index-8,cluster-4,succ-2],
          [index-9,cluster-4,succ-9]]),
     circuit_cluster(
         2,
         [[index-1,cluster-1,succ-1],
          [index-2,cluster-1,succ-4],
          [index-3,cluster-2,succ-3],
          [index-4,cluster-2,succ-2],
```

```
          [index-5,cluster-3,succ-5],
          [index-6,cluster-3,succ-9],
          [index-7,cluster-3,succ-7],
          [index-8,cluster-4,succ-8],
          [index-9,cluster-4,succ-6]])]).

ctr_typical(
    circuit_cluster,
    ['NCIRCUIT'<size('NODES'),
     size('NODES')>2,
     range('NODES'^cluster)>1]).

ctr_exchangeable(circuit_cluster,[items('NODES',all)]).

ctr_graph(
    circuit_cluster,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=\=nodes1^index,nodes1^succ=nodes2^index],
    ['NTREE'=0,'NSCC'='NCIRCUIT'],
    ['ONE_SUCC'],
    [ALL_VERTICES>>
     [variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'NODES'^cluster)])]],
    [alldifferent(variables),
     nvalues(variables,=,size('NODES',cluster))]).

ctr_application(circuit_cluster,[2]).
```

B.70 circular_change

◇ **META-DATA:**

```
ctr_date(circular_change,['20030820','20040530','20060805']).

ctr_origin(circular_change,'Derived from %c.',[change]).

ctr_arguments(
    circular_change,
    ['NCHANGE'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    circular_change,
    ['NCHANGE'>=0,
     'NCHANGE'=<size('VARIABLES'),
     required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    circular_change,
    circular_change(
        4,
        [[var-4],[var-4],[var-3],[var-4],[var-1]],
        =\=)).

ctr_typical(
    circular_change,
    ['NCHANGE'>0,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=\=])]).

ctr_exchangeable(
    circular_change,
    [items('VARIABLES',shift),translate(['VARIABLES'^var])]).

ctr_graph(
    circular_change,
    ['VARIABLES'],
    2,
    ['CIRCUIT'>>collection(variables1,variables2)],
    ['CTR'(variables1^var,variables2^var)],
    ['NARC'='NCHANGE'],
```

```
    []).

ctr_eval(
    circular_change,
    [checker(circular_change_c),automaton(circular_change_a)]).

ctr_pure_functional_dependency(circular_change,[]).

ctr_functional_dependency(circular_change,1,[2,3]).

circular_change_c(NCHANGE,VARIABLES,=) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),NCHANGE),
        (   N=0 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            VARS=[V|_39503],
            append(VARS,[V],NEWVARS),
            change_eq_c(NEWVARS,0,NCHANGE)
        ).

circular_change_c(NCHANGE,VARIABLES,=\=) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),NCHANGE),
        (   N=0 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            VARS=[V|_39503],
            append(VARS,[V],NEWVARS),
            change_neq_c(NEWVARS,0,NCHANGE)
        ).

circular_change_c(NCHANGE,VARIABLES,<) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),NCHANGE),
        (   N=0 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            VARS=[V|_39503],
            append(VARS,[V],NEWVARS),
            change_lt_c(NEWVARS,0,NCHANGE)
        ).
```

```
circular_change_c(NCHANGE,VARIABLES,>=) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),NCHANGE),
        (   N=0 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            VARS=[V|_39503],
            append(VARS,[V],NEWVARS),
            change_geq_c(NEWVARS,0,NCHANGE)
        ).

circular_change_c(NCHANGE,VARIABLES,>) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),NCHANGE),
        (   N=0 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            VARS=[V|_39503],
            append(VARS,[V],NEWVARS),
            change_gt_c(NEWVARS,0,NCHANGE)
        ).

circular_change_c(NCHANGE,VARIABLES,=<) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),NCHANGE),
        (   N=0 ->
            NCHANGE#=0
        ;   get_attr1(VARIABLES,VARS),
            VARS=[V|_39503],
            append(VARS,[V],NEWVARS),
            change_leq_c(NEWVARS,0,NCHANGE)
        ).

circular_change_a(FLAG,NCHANGE,VARIABLES,CTR) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        check_type(dvar(0,N),NCHANGE),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        VARIABLES=[V1|_39506],
        append(VARIABLES,[V1],CVARIABLES),
        circular_change_signature(CVARIABLES,SIGNATURE,CTR),
        automaton(
            SIGNATURE,
```

```
            _41291,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NCHANGE#<=>FLAG.

circular_change_signature([],[],_39443).

circular_change_signature([_39447],[],_39446) :-
        !.

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    =) :-
        !,
        VAR1#=VAR2#<=>S,
        circular_change_signature([[var-VAR2]|VARs],Ss,=).

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    =\=) :-
        !,
        VAR1#\=VAR2#<=>S,
        circular_change_signature([[var-VAR2]|VARs],Ss,=\=).

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    <) :-
        !,
        VAR1#<VAR2#<=>S,
        circular_change_signature([[var-VAR2]|VARs],Ss,<).

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    >=) :-
        !,
        VAR1#>=VAR2#<=>S,
        circular_change_signature([[var-VAR2]|VARs],Ss,>=).
```

```
circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    >) :-
        !,
        VAR1#>VAR2#<=>S,
        circular_change_signature([[var-VAR2]|VARs],Ss,>).

circular_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    =<) :-
        !,
        VAR1#=<VAR2#<=>S,
        circular_change_signature([[var-VAR2]|VARs],Ss,=<).
```

B.71 clause_and

◇ META-DATA:

```
ctr_date(clause_and,['20090416']).

ctr_origin(clause_and,'Logic',[]).

ctr_synonyms(clause_and,[clause]).

ctr_arguments(
    clause_and,
    ['POSVARS'-collection(var-dvar),
     'NEGVARS'-collection(var-dvar),
     'VAR'-dvar]).

ctr_restrictions(
    clause_and,
    [size('POSVARS')+size('NEGVARS')>0,
     required('POSVARS',var),
     'POSVARS'^var>=0,
     'POSVARS'^var=<1,
     required('NEGVARS',var),
     'NEGVARS'^var>=0,
     'NEGVARS'^var=<1,
     'VAR'>=0,
     'VAR'=<1]).

ctr_example(
    clause_and,
    clause_and([[var-1],[var-0]],[[var-0]],0)).

ctr_typical(clause_and,[size('POSVARS')+size('NEGVARS')>1]).

ctr_exchangeable(
    clause_and,
    [items('POSVARS',all),items('NEGVARS',all)]).

ctr_eval(clause_and,[automaton(clause_and_a)]).

ctr_extensible(clause_and,['VAR'=0],'POSVARS',any).

ctr_extensible(clause_and,['VAR'=0],'NEGVARS',any).

clause_and_a(FLAG,POSVARS,NEGVARS,VAR) :-
        collection(POSVARS,[dvar(0,1)]),
```

```
        collection(NEGVARS,[dvar(0,1)]),
        check_type(dvar(0,1),VAR),
        length(POSVARS,LP),
        length(NEGVARS,LN),
        L is LP+LN,
        L>0,
        get_attr1(POSVARS,LISTP),
        get_attr1(NEGVARS,LISTN),
        clause_and_negate(LISTN,LISTNN),
        append([VAR],LISTP,LIST),
        append(LIST,LISTNN,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _32124,
            LIST_VARIABLES,
            [source(s),sink(k),sink(j)],
            [arc(s,0,i),
             arc(s,1,j),
             arc(i,0,k),
             arc(i,1,i),
             arc(k,0,k),
             arc(k,1,k),
             arc(j,1,j)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

clause_and_negate([],[]).

clause_and_negate([V|R],[U|S]) :-
        V#<=> #\U,
        clause_and_negate(R,S).
```

B.72 clause_or

◇ **META-DATA:**

```
ctr_date(clause_or,['20090415']).

ctr_origin(clause_or,'Logic',[]).

ctr_synonyms(clause_or,[clause]).

ctr_arguments(
    clause_or,
    ['POSVARS'-collection(var-dvar),
     'NEGVARS'-collection(var-dvar),
     'VAR'-dvar]).

ctr_restrictions(
    clause_or,
    [size('POSVARS')+size('NEGVARS')>0,
     required('POSVARS',var),
     'POSVARS'^var>=0,
     'POSVARS'^var=<1,
     required('NEGVARS',var),
     'NEGVARS'^var>=0,
     'NEGVARS'^var=<1,
     'VAR'>=0,
     'VAR'=<1]).

ctr_example(clause_or,clause_or([[var-0],[var-0]],[[var-0]],1)).

ctr_typical(clause_or,[size('POSVARS')+size('NEGVARS')>1]).

ctr_exchangeable(
    clause_or,
    [items('POSVARS',all),items('NEGVARS',all)]).

ctr_eval(clause_or,[automaton(clause_or_a)]).

ctr_extensible(clause_or,['VAR'=1],'POSVARS',any).

ctr_extensible(clause_or,['VAR'=1],'NEGVARS',any).

clause_or_a(FLAG,POSVARS,NEGVARS,VAR) :-
        collection(POSVARS,[dvar(0,1)]),
        collection(NEGVARS,[dvar(0,1)]),
        check_type(dvar(0,1),VAR),
```

```
        length(POSVARS,LP),
        length(NEGVARS,LN),
        L is LP+LN,
        L>0,
        get_attr1(POSVARS,LISTP),
        get_attr1(NEGVARS,LISTN),
        clause_or_negate(LISTN,LISTNN),
        append([VAR],LISTP,LIST),
        append(LIST,LISTNN,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _32610,
            LIST_VARIABLES,
            [source(s),sink(i),sink(k)],
            [arc(s,0,i),
             arc(s,1,j),
             arc(i,0,i),
             arc(j,0,j),
             arc(j,1,k),
             arc(k,0,k),
             arc(k,1,k)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

clause_or_negate([],[]).

clause_or_negate([V|R],[U|S]) :-
        V#<=> #\U,
        clause_or_negate(R,S).
```

B.73 clique

◇ META-DATA:

```
ctr_date(clique,['20030820','20040530','20060805']).

ctr_origin(clique,'\\cite{Fahle02}',[]).

ctr_arguments(
    clique,
    ['SIZE_CLIQUE'-dvar,
     'NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    clique,
    ['SIZE_CLIQUE'>=0,
     'SIZE_CLIQUE'=<size('NODES'),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    clique,
    clique(
        3,
        [[index-1,succ-{}],
         [index-2,succ-{3,5}],
         [index-3,succ-{2,5}],
         [index-4,succ-{}],
         [index-5,succ-{2,3}]])).

ctr_typical(
    clique,
    ['SIZE_CLIQUE'>=2,
     'SIZE_CLIQUE'<size('NODES'),
     size('NODES')>2]).

ctr_exchangeable(clique,[items('NODES',all)]).

ctr_graph(
    clique,
    ['NODES'],
    2,
```

```
    ['CLIQUE'(=\=)>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ],
    ['NARC'='SIZE_CLIQUE'*'SIZE_CLIQUE'-'SIZE_CLIQUE',
     'NVERTEX'='SIZE_CLIQUE'],
    ['SYMMETRIC']).

ctr_functional_dependency(clique,1,[2]).

ctr_application(clique,[2]).
```

B.74 colored_matrix

◇ META-DATA:

```
ctr_predefined(colored_matrix).

ctr_date(colored_matrix,['20031017','20040530']).

ctr_origin(colored_matrix,'KOALOG',[]).

ctr_synonyms(
    colored_matrix,
    [coloured_matrix,cardinality_matrix,card_matrix]).

ctr_arguments(
    colored_matrix,
    ['C'-int,
     'L'-int,
     'K'-int,
     'MATRIX'-collection(column-int,line-int,var-dvar),
     'CPROJ'-collection(column-int,val-int,nocc-dvar),
     'LPROJ'-collection(line-int,val-int,nocc-dvar)]).

ctr_restrictions(
    colored_matrix,
    ['C'>=0,
     'L'>=0,
     'K'>=0,
     required('MATRIX',[column,line,var]),
     increasing_seq('MATRIX',[column,line]),
     size('MATRIX')='C'*'L'+'C'+'L'+1,
     'MATRIX'^column>=0,
     'MATRIX'^column=<'C',
     'MATRIX'^line>=0,
     'MATRIX'^line=<'L',
     'MATRIX'^var>=0,
     'MATRIX'^var=<'K',
     required('CPROJ',[column,val,nocc]),
     increasing_seq('CPROJ',[column,val]),
     size('CPROJ')='C'*'K'+'C'+'K'+1,
     'CPROJ'^column>=0,
     'CPROJ'^column=<'C',
     'CPROJ'^val>=0,
     'CPROJ'^val=<'K',
     required('LPROJ',[line,val,nocc]),
     increasing_seq('LPROJ',[line,val]),
```

```
     size('LPROJ')='L'*'K'+'L'+'K'+1,
     'LPROJ'^line>=0,
     'LPROJ'^line=<'L',
     'LPROJ'^val>=0,
     'LPROJ'^val=<'K']).

ctr_example(
    colored_matrix,
    colored_matrix(
        1,
        2,
        4,
        [[column-0,line-0,var-3],
         [column-0,line-1,var-1],
         [column-0,line-2,var-3],
         [column-1,line-0,var-4],
         [column-1,line-1,var-4],
         [column-1,line-2,var-3]],
        [[column-0,val-0,nocc-0],
         [column-0,val-1,nocc-1],
         [column-0,val-2,nocc-0],
         [column-0,val-3,nocc-2],
         [column-0,val-4,nocc-0],
         [column-1,val-0,nocc-0],
         [column-1,val-1,nocc-0],
         [column-1,val-2,nocc-0],
         [column-1,val-3,nocc-1],
         [column-1,val-4,nocc-2]],
        [[line-0,val-0,nocc-0],
         [line-0,val-1,nocc-0],
         [line-0,val-2,nocc-0],
         [line-0,val-3,nocc-1],
         [line-0,val-4,nocc-1],
         [line-1,val-0,nocc-0],
         [line-1,val-1,nocc-1],
         [line-1,val-2,nocc-0],
         [line-1,val-3,nocc-0],
         [line-1,val-4,nocc-1],
         [line-2,val-0,nocc-0],
         [line-2,val-1,nocc-0],
         [line-2,val-2,nocc-0],
         [line-2,val-3,nocc-2],
         [line-2,val-4,nocc-0]])).

ctr_typical(
    colored_matrix,
```

```
    ['C'>=1,'L'>=1,'K'>=1,range('MATRIX'^var)>1]).

ctr_pure_functional_dependency(colored_matrix,[]).

ctr_functional_dependency(colored_matrix,5-3,[1,2,3]).

ctr_functional_dependency(colored_matrix,6-3,[1,2,3]).
```

B.75 coloured_cumulative

◇ **META-DATA:**

```
ctr_date(
    coloured_cumulative,
    ['20000128','20030820','20060805']).

ctr_origin(
    coloured_cumulative,
    Derived from %c and %c.,
    [cumulative,nvalues]).

ctr_synonyms(coloured_cumulative,[colored_cumulative]).

ctr_arguments(
    coloured_cumulative,
    [TASKS-
     collection(
         origin-dvar,
         duration-dvar,
         end-dvar,
         colour-dvar),
     'LIMIT'-int]).

ctr_restrictions(
    coloured_cumulative,
    [require_at_least(2,'TASKS',[origin,duration,end]),
     required('TASKS',colour),
     'TASKS'^duration>=0,
     'TASKS'^origin=<'TASKS'^end,
     'LIMIT'>=0]).

ctr_example(
    coloured_cumulative,
    coloured_cumulative(
        [[origin-1,duration-2,end-3,colour-1],
         [origin-2,duration-9,end-11,colour-2],
         [origin-3,duration-10,end-13,colour-3],
         [origin-6,duration-6,end-12,colour-2],
         [origin-7,duration-2,end-9,colour-3]],
        2)).

ctr_typical(
    coloured_cumulative,
    [size('TASKS')>1,
```

```
     range('TASKS'^origin)>1,
     range('TASKS'^duration)>1,
     range('TASKS'^end)>1,
     range('TASKS'^colour)>1,
     'LIMIT'<nval('TASKS'^colour)]).

ctr_exchangeable(
    coloured_cumulative,
    [items('TASKS',all),
     translate(['TASKS'^origin,'TASKS'^end]),
     vals(['TASKS'^colour],int,=\=,all,dontcare),
     vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_graph(
    coloured_cumulative,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin+tasks^duration=tasks^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    coloured_cumulative,
    ['TASKS','TASKS'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1^duration>0,
     tasks2^origin=<tasks1^origin,
     tasks1^origin<tasks2^end],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^colour)])]],
    [nvalues(variables,=<,'LIMIT')]).

ctr_eval(
    coloured_cumulative,
    [reformulation(coloured_cumulative_r)]).

ctr_contractible(coloured_cumulative,[],'TASKS',any).

ctr_application(coloured_cumulative,[1]).
```

```
coloured_cumulative_r(TASKS,LIMIT) :-
        collection(TASKS,[dvar,dvar_gteq(0),dvar,dvar]),
        integer(LIMIT),
        LIMIT>=0,
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        get_attr3(TASKS,ENDS),
        get_attr4(TASKS,COLOURS),
        ori_dur_end(ORIGINS,DURATIONS,ENDS),
        coloured_cumulative1(
            ORIGINS,
            ENDS,
            COLOURS,
            1,
            ORIGINS,
            ENDS,
            COLOURS,
            LIMIT).

coloured_cumulative1(
    [],
    [],
    [],
    _58463,
    _58510,
    _58557,
    _58604,
    _58651).

coloured_cumulative1(
    [Oi|RO],
    [Ei|RE],
    [Ci|RC],
    I,
    ORIGINS,
    ENDS,
    COLOURS,
    LIMIT) :-
        coloured_cumulative2(
            ORIGINS,
            ENDS,
            COLOURS,
            1,
            I,
            Oi,
```

```
            Ei,
            Ci,
            COLi),
        Ni in 1..LIMIT,
        nvalue(Ni,COLi),
        I1 is I+1,
        coloured_cumulative1(
            RO,
            RE,
            RC,
            I1,
            ORIGINS,
            ENDS,
            COLOURS,
            LIMIT).

coloured_cumulative2(
    [],
    [],
    [],
    _58466,
    _58513,
    _58560,
    _58607,
    _58654,
    []).

coloured_cumulative2(
    [_58096|RO],
    [_58100|RE],
    [_58104|RC],
    J,
    I,
    Oi,
    Ei,
    Ci,
    [Ci|R]) :-
        I=J,
        !,
        J1 is J+1,
        coloured_cumulative2(RO,RE,RC,J1,I,Oi,Ei,Ci,R).

coloured_cumulative2(
    [Oj|RO],
    [Ej|RE],
    [Cj|RC],
```

```
J,
I,
Oi,
Ei,
Ci,
[Cij|R]) :-
    I=\=J,
    K in 1..2,
    fd_min(Ci,Ci_min),
    fd_max(Ci,Ci_max),
    fd_min(Cj,Cj_min),
    fd_max(Cj,Cj_max),
    Min is min(Ci_min,Cj_min),
    Max is max(Ci_max,Cj_max),
    Cij in Min..Max,
    element(K,[Ci,Cj],Cij),
    Oj#=<Oi#/\Ej#>Oi#/\Cij#=Cj#\/
    (Oj#>Oi#\/Ej#=<Oi)#/\Cij#=Ci,
    J1 is J+1,
    coloured_cumulative2(RO,RE,RC,J1,I,Oi,Ei,Ci,R).
```

B.76 coloured_cumulatives

◇ META-DATA:

```
ctr_date(
    coloured_cumulatives,
    ['20000128','20030820','20060805']).

ctr_origin(
    coloured_cumulatives,
    Derived from %c and %c.,
    [cumulatives,nvalues]).

ctr_synonyms(coloured_cumulatives,[colored_cumulatives]).

ctr_arguments(
    coloured_cumulatives,
    [TASKS-
     collection(
         machine-dvar,
         origin-dvar,
         duration-dvar,
         end-dvar,
         colour-dvar),
     'MACHINES'-collection(id-int,capacity-int)]).

ctr_restrictions(
    coloured_cumulatives,
    [required('TASKS',[machine,colour]),
     require_at_least(2,'TASKS',[origin,duration,end]),
     'TASKS'^duration>=0,
     'TASKS'^origin=<'TASKS'^end,
     required('MACHINES',[id,capacity]),
     distinct('MACHINES',id),
     'MACHINES'^capacity>=0]).

ctr_example(
    coloured_cumulatives,
    coloured_cumulatives(
        [[machine-1,origin-6,duration-6,end-12,colour-2],
         [machine-1,origin-2,duration-9,end-11,colour-3],
         [machine-2,origin-7,duration-3,end-10,colour-3],
         [machine-1,origin-1,duration-2,end-3,colour-1],
         [machine-2,origin-4,duration-5,end-9,colour-3],
         [machine-1,origin-3,duration-10,end-13,colour-2]],
        [[id-1,capacity-2],[id-2,capacity-1]])).
```

```
ctr_typical(
    coloured_cumulatives,
    [size('TASKS')>1,
     range('TASKS'^machine)>1,
     range('TASKS'^origin)>1,
     range('TASKS'^duration)>1,
     range('TASKS'^end)>1,
     range('TASKS'^colour)>1,
     'TASKS'^duration>0,
     size('MACHINES')>1,
     'MACHINES'^capacity>0,
     'MACHINES'^capacity<nval('TASKS'^colour),
     size('TASKS')>size('MACHINES')]).

ctr_exchangeable(
    coloured_cumulatives,
    [items('TASKS',all),
     items('MACHINES',all),
     vals(['MACHINES'^capacity],int,<,dontcare,dontcare),
     vals(
         ['TASKS'^machine,'MACHINES'^id],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    coloured_cumulatives,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin+tasks^duration=tasks^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    coloured_cumulatives,
    ['TASKS','TASKS'],
    2,
    foreach('MACHINES',['PRODUCT'>>collection(tasks1,tasks2)]),
    [tasks1^machine='MACHINES'^id,
     tasks1^machine=tasks2^machine,
     tasks1^duration>0,
     tasks2^origin=<tasks1^origin,
     tasks1^origin<tasks2^end],
```

```
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^colour)])]],
    [nvalues(variables,=<,'MACHINES'^capacity)]).

ctr_eval(
    coloured_cumulatives,
    [reformulation(coloured_cumulatives_r)]).

ctr_contractible(coloured_cumulatives,[],'TASKS',any).

ctr_application(coloured_cumulatives,[1]).

coloured_cumulatives_r(TASKS,MACHINES) :-
        collection(TASKS,[dvar,dvar,dvar_gteq(0),dvar,dvar]),
        get_attr1(TASKS,VMACHINES),
        get_attr2(TASKS,ORIGINS),
        get_attr3(TASKS,DURATIONS),
        get_attr4(TASKS,ENDS),
        get_attr5(TASKS,COLOURS),
        ori_dur_end(ORIGINS,DURATIONS,ENDS),
        collection(MACHINES,[int,int_gteq(0)]),
        get_attr1(MACHINES,IDS),
        get_attr2(MACHINES,CAPACITIES),
        all_different(IDS),
        get_maximum(CAPACITIES,CAPA_MAX),
        coloured_cumulatives1(
            VMACHINES,
            ORIGINS,
            ENDS,
            COLOURS,
            1,
            VMACHINES,
            ORIGINS,
            ENDS,
            COLOURS,
            IDS,
            CAPACITIES,
            CAPA_MAX).

coloured_cumulatives1(
    [],
```

```
    [],
    [],
    [],
    _65672,
    _65719,
    _65766,
    _65813,
    _65860,
    _65907,
    _65954,
    _66001).

coloured_cumulatives1(
    [Mi|RM],
    [Oi|RO],
    [Ei|RE],
    [Ci|RC],
    I,
    VMACHINES,
    ORIGINS,
    ENDS,
    COLOURS,
    IDS,
    CAPACITIES,
    CAPA_MAX) :-
        coloured_cumulatives2(
            VMACHINES,
            ORIGINS,
            ENDS,
            COLOURS,
            1,
            I,
            Mi,
            Oi,
            Ei,
            Ci,
            COLi),
        LIMIT in 0..CAPA_MAX,
        link_index_to_attribute(IDS,CAPACITIES,Mi,LIMIT),
        Ni in 0..CAPA_MAX,
        Ni#=<LIMIT,
        nvalue(Ni,COLi),
        I1 is I+1,
        coloured_cumulatives1(
            RM,
            RO,
```

```
            RE,
            RC,
            I1,
            VMACHINES,
            ORIGINS,
            ENDS,
            COLOURS,
            IDS,
            CAPACITIES,
            CAPA_MAX).

coloured_cumulatives2(
    [],
    [],
    [],
    [],
    _65669,
    _65716,
    _65763,
    _65810,
    _65857,
    _65904,
    []).

coloured_cumulatives2(
    [_65250|RM],
    [_65254|RO],
    [_65258|RE],
    [_65262|RC],
    J,
    I,
    Mi,
    Oi,
    Ei,
    Ci,
    [Ci|R]) :-
        I=J,
        !,
        J1 is J+1,
        coloured_cumulatives2(RM,RO,RE,RC,J1,I,Mi,Oi,Ei,Ci,R).

coloured_cumulatives2(
    [Mj|RM],
    [Oj|RO],
    [Ej|RE],
    [Cj|RC],
```

```
J,
I,
Mi,
Oi,
Ei,
Ci,
[Cij|R]) :-
    I=\=J,
    K in 1..2,
    fd_min(Ci,Ci_min),
    fd_max(Ci,Ci_max),
    fd_min(Cj,Cj_min),
    fd_max(Cj,Cj_max),
    Min is min(Ci_min,Cj_min),
    Max is max(Ci_max,Cj_max),
    Cij in Min..Max,
    element(K,[Ci,Cj],Cij),
    Mj#=Mi#/\Oj#=<Oi#/\Ej#>Oi#/\Cij#=Cj#\/
    (Mj#\=Mi#\/Oj#>Oi#\/Ej#=<Oi)#/\Cij#=Ci,
    J1 is J+1,
    coloured_cumulatives2(RM,RO,RE,RC,J1,I,Mi,Oi,Ei,Ci,R).
```

B.77 common

♢ META-DATA:

```
ctr_date(common,['20000128','20030820','20060805']).

ctr_origin(common,'N.~Beldiceanu',[]).

ctr_arguments(
    common,
    ['NCOMMON1'-dvar,
     'NCOMMON2'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    common,
    ['NCOMMON1'>=0,
     'NCOMMON1'=<size('VARIABLES1'),
     'NCOMMON2'>=0,
     'NCOMMON2'=<size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    common,
    common(
        3,
        4,
        [[var-1],[var-9],[var-1],[var-5]],
        [[var-2],[var-1],[var-9],[var-9],[var-6],[var-9]])).

ctr_typical(
    common,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1]).

ctr_exchangeable(
    common,
    [args(
         [['NCOMMON1','NCOMMON2'],
          ['VARIABLES1','VARIABLES2']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
```

```
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    common,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSOURCE'='NCOMMON1','NSINK'='NCOMMON2'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(common,[reformulation(common_r)]).

ctr_pure_functional_dependency(common,[]).

ctr_functional_dependency(common,1,[3,4]).

ctr_functional_dependency(common,2,[3,4]).

common_r(NCOMMON1,NCOMMON2,VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        check_type(dvar(0,N1),NCOMMON1),
        check_type(dvar(0,N2),NCOMMON2),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        common1(VARS1,VARS2,_MAT12,SUM1),
        call(NCOMMON1#=SUM1),
        common1(VARS2,VARS1,_MAT21,SUM2),
        call(NCOMMON2#=SUM2).
```

B.78 common_interval

◇ META-DATA:

```
ctr_date(common_interval,['20030820','20060805']).

ctr_origin(common_interval,'Derived from %c.',[common]).

ctr_arguments(
    common_interval,
    ['NCOMMON1'-dvar,
     'NCOMMON2'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    common_interval,
    ['NCOMMON1'>=0,
     'NCOMMON1'=<size('VARIABLES1'),
     'NCOMMON2'>=0,
     'NCOMMON2'=<size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'SIZE_INTERVAL'>0]).

ctr_example(
    common_interval,
    common_interval(
        3,
        2,
        [[var-8],[var-6],[var-6],[var-0]],
        [[var-7],[var-3],[var-3],[var-3],[var-3],[var-7]],
        3)).

ctr_typical(
    common_interval,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1,
     'SIZE_INTERVAL'>1,
     'SIZE_INTERVAL'<range('VARIABLES1'^var),
     'SIZE_INTERVAL'<range('VARIABLES2'^var)]).

ctr_exchangeable(
```

```
    common_interval,
    [args(
         [['NCOMMON1','NCOMMON2'],
          ['VARIABLES1','VARIABLES2'],
          ['SIZE_INTERVAL']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    common_interval,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var/'SIZE_INTERVAL'=
     variables2^var/'SIZE_INTERVAL'],
    ['NSOURCE'='NCOMMON1','NSINK'='NCOMMON2'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(common_interval,[reformulation(common_interval_r)]).

ctr_pure_functional_dependency(common_interval,[]).

ctr_functional_dependency(common_interval,1,[3,4,5]).

ctr_functional_dependency(common_interval,2,[3,4,5]).

common_interval_r(
    NCOMMON1,
    NCOMMON2,
    VARIABLES1,
    VARIABLES2,
    SIZE_INTERVAL) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
```

```
length(VARIABLES1,N1),
length(VARIABLES2,N2),
check_type(dvar(0,N1),NCOMMON1),
check_type(dvar(0,N2),NCOMMON2),
integer(SIZE_INTERVAL),
SIZE_INTERVAL>0,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
common1(QUOTVARS1,QUOTVARS2,_MAT12,SUM1),
call(NCOMMON1#=SUM1),
common1(QUOTVARS2,QUOTVARS1,_MAT21,SUM2),
call(NCOMMON2#=SUM2).
```

B.79 common_modulo

◇ **META-DATA:**

```
ctr_date(common_modulo,['20030820','20060806']).

ctr_origin(common_modulo,'Derived from %c.',[common]).

ctr_arguments(
    common_modulo,
    ['NCOMMON1'-dvar,
     'NCOMMON2'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    common_modulo,
    ['NCOMMON1'>=0,
     'NCOMMON1'=<size('VARIABLES1'),
     'NCOMMON2'>=0,
     'NCOMMON2'=<size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'M'>0]).

ctr_example(
    common_modulo,
    common_modulo(
        3,
        4,
        [[var-0],[var-4],[var-0],[var-8]],
        [[var-7],[var-5],[var-4],[var-9],[var-2],[var-4]],
        5)).

ctr_typical(
    common_modulo,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1,
     'M'>1,
     'M'<maxval('VARIABLES1'^var),
     'M'<maxval('VARIABLES2'^var)]).

ctr_exchangeable(
```

```
    common_modulo,
    [args(
         [['NCOMMON1','NCOMMON2'],
          ['VARIABLES1','VARIABLES2'],
          ['M']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(['VARIABLES1'^var],mod('M'),=,dontcare,dontcare),
     vals(['VARIABLES2'^var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
    common_modulo,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['NSOURCE'='NCOMMON1','NSINK'='NCOMMON2'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(common_modulo,[reformulation(common_modulo_r)]).

ctr_pure_functional_dependency(common_modulo,[]).

ctr_functional_dependency(common_modulo,1,[3,4,5]).

ctr_functional_dependency(common_modulo,2,[3,4,5]).

common_modulo_r(NCOMMON1,NCOMMON2,VARIABLES1,VARIABLES2,M) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        check_type(dvar(0,N1),NCOMMON1),
        check_type(dvar(0,N2),NCOMMON2),
        integer(M),
        M>0,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        gen_remainder(VARS1,M,REMVARS1),
        gen_remainder(VARS2,M,REMVARS2),
        common1(REMVARS1,REMVARS2,_MAT12,SUM1),
        call(NCOMMON1#=SUM1),
        common1(REMVARS2,REMVARS1,_MAT21,SUM2),
        call(NCOMMON2#=SUM2).
```

B.80 common_partition

◇ **META-DATA:**

```
ctr_date(common_partition,['20030820','20060806']).

ctr_origin(common_partition,'Derived from %c.',[common]).

ctr_types(common_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    common_partition,
    ['NCOMMON1'-dvar,
     'NCOMMON2'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    common_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     'NCOMMON1'>=0,
     'NCOMMON1'=<size('VARIABLES1'),
     'NCOMMON2'>=0,
     'NCOMMON2'=<size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    common_partition,
    common_partition(
        3,
        4,
        [[var-2],[var-3],[var-6],[var-0]],
        [[var-0],[var-6],[var-3],[var-3],[var-7],[var-1]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(
    common_partition,
    [size('VARIABLES1')>1,
```

```
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1,
     size('VARIABLES1')>size('PARTITIONS'),
     size('VARIABLES2')>size('PARTITIONS')]).

ctr_exchangeable(
    common_partition,
    [args(
         [['NCOMMON1','NCOMMON2'],
          ['VARIABLES1','VARIABLES2'],
          ['PARTITIONS']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['VARIABLES1'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    common_partition,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    ['NSOURCE'='NCOMMON1','NSINK'='NCOMMON2'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(common_partition,[reformulation(common_partition_r)]).

ctr_pure_functional_dependency(common_partition,[]).

ctr_functional_dependency(common_partition,1,[3,4,5]).
```

```
ctr_functional_dependency(common_partition,2,[3,4,5]).

common_partition_r(
    NCOMMON1,
    NCOMMON2,
    VARIABLES1,
    VARIABLES2,
    PARTITIONS) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        check_type(dvar(0,N1),NCOMMON1),
        check_type(dvar(0,N2),NCOMMON2),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        length(PARTITIONS,P),
        P>1,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        get_col_attr1(PARTITIONS,1,PVALS),
        flattern(PVALS,VALS),
        all_different(VALS),
        length(PVALS,LPVALS),
        LPVALS1 is LPVALS+1,
        get_partition_var(VARS1,PVALS,PVARS1,LPVALS1,0),
        LPVALS2 is LPVALS1+1,
        get_partition_var(VARS2,PVALS,PVARS2,LPVALS2,LPVALS1),
        common1(PVARS1,PVARS2,_MAT12,SUM1),
        call(NCOMMON1#=SUM1),
        common1(PVARS2,PVARS1,_MAT21,SUM2),
        call(NCOMMON2#=SUM2).
```

B.81 compare_and_count

◇ META-DATA:

```
ctr_predefined(compare_and_count).

ctr_date(compare_and_count,['20110628']).

ctr_origin(compare_and_count,'Generalise %c',[discrepancy]).

ctr_arguments(
    compare_and_count,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'COMPARE'-atom,
     'COUNT'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    compare_and_count,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     in_list('COMPARE',[=,=\=,<,>=,>,=<]),
     in_list('COUNT',[=,=\=,<,>=,>,=<]),
     'LIMIT'>=0]).

ctr_example(
    compare_and_count,
    compare_and_count(
        [[var-4],[var-5],[var-5],[var-4],[var-5]],
        [[var-4],[var-2],[var-5],[var-1],[var-5]],
        =,
        =<,
        3)).

ctr_typical(
    compare_and_count,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     in_list('COMPARE',[=]),
     in_list('COUNT',[=,<,>=,>,=<]),
     'LIMIT'>0,
     'LIMIT'<size('VARIABLES1')]).
```

```
ctr_eval(
    compare_and_count,
    [reformulation(compare_and_count_r)]).

ctr_pure_functional_dependency(
    compare_and_count,
    [in_list('COUNT',[=])]).

ctr_contractible(
    compare_and_count,
    [in_list('COUNT',[<,=<])],
    ['VARIABLES1','VARIABLES2'],
    any).

ctr_extensible(
    compare_and_count,
    [in_list('COUNT',[>=,>])],
    ['VARIABLES1','VARIABLES2'],
    any).

compare_and_count_r(VARIABLES1,VARIABLES2,COMPARE,COUNT,LIMIT) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1=N2,
        memberchk(COMPARE,[=,=\=,<,>=,>,=<]),
        memberchk(COUNT,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        LIMIT#>=0,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        compare_and_count_r1(VARS1,VARS2,COMPARE,TERM),
        compare_and_count_r2(COUNT,TERM,LIMIT).

compare_and_count_r1([],[],_23730,0).

compare_and_count_r1([V1|R1],[V2|R2],=,B+T) :-
        V1#=V2#<=>B,
        compare_and_count_r1(R1,R2,=,T).

compare_and_count_r1([V1|R1],[V2|R2],=\=,B+T) :-
        V1#\=V2#<=>B,
        compare_and_count_r1(R1,R2,=\=,T).

compare_and_count_r1([V1|R1],[V2|R2],<,B+T) :-
```

```
        V1#<V2#<=>B,
        compare_and_count_r1(R1,R2,<,T).

compare_and_count_r1([V1|R1],[V2|R2],>=,B+T) :-
        V1#>=V2#<=>B,
        compare_and_count_r1(R1,R2,>=,T).

compare_and_count_r1([V1|R1],[V2|R2],>,B+T) :-
        V1#>V2#<=>B,
        compare_and_count_r1(R1,R2,>,T).

compare_and_count_r1([V1|R1],[V2|R2],=<,B+T) :-
        V1#=<V2#<=>B,
        compare_and_count_r1(R1,R2,=<,T).

compare_and_count_r2(=,TERM,LIMIT) :-
        call(TERM#=LIMIT).

compare_and_count_r2(=\=,TERM,LIMIT) :-
        call(TERM#\=LIMIT).

compare_and_count_r2(<,TERM,LIMIT) :-
        call(TERM#<LIMIT).

compare_and_count_r2(>=,TERM,LIMIT) :-
        call(TERM#>=LIMIT).

compare_and_count_r2(>,TERM,LIMIT) :-
        call(TERM#>LIMIT).

compare_and_count_r2(=<,TERM,LIMIT) :-
        call(TERM#=<LIMIT).
```

B.82 cond_lex_cost

◇ **META-DATA:**

```
ctr_date(cond_lex_cost,['20060416']).

ctr_origin(
    cond_lex_cost,
    Inspired by \cite{WallaceWilson06}.,
    []).

ctr_types(cond_lex_cost,['TUPLE_OF_VALS'-collection(val-int)]).

ctr_arguments(
    cond_lex_cost,
    ['VECTOR'-collection(var-dvar),
     'PREFERENCE_TABLE'-collection(tuple-'TUPLE_OF_VALS'),
     'COST'-dvar]).

ctr_restrictions(
    cond_lex_cost,
    [size('TUPLE_OF_VALS')>=1,
     required('TUPLE_OF_VALS',val),
     required('VECTOR',var),
     size('VECTOR')=size('TUPLE_OF_VALS'),
     required('PREFERENCE_TABLE',tuple),
     same_size('PREFERENCE_TABLE',tuple),
     distinct('PREFERENCE_TABLE',[]),
     in_relation('VECTOR','PREFERENCE_TABLE'),
     'COST'>=1,
     'COST'=<size('PREFERENCE_TABLE')]).

ctr_example(
    cond_lex_cost,
    cond_lex_cost(
        [[var-0],[var-1]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]]],
        2)).

ctr_typical(
    cond_lex_cost,
    [size('TUPLE_OF_VALS')>1,
     size('VECTOR')>1,
```

```
     size('PREFERENCE_TABLE')>1]).

ctr_exchangeable(
    cond_lex_cost,
    [items_sync('VECTOR','PREFERENCE_TABLE'^tuple,all),
     vals(
         ['VECTOR','PREFERENCE_TABLE'^tuple],
         int,
         =\=,
         all,
         dontcare)]).

ctr_eval(cond_lex_cost,[automata(cond_lex_cost_a)]).

cond_lex_cost_a(VECTOR,PREFERENCE_TABLE,COST) :-
        collection(VECTOR,[dvar]),
        collection(PREFERENCE_TABLE,[col([dvar])]),
        same_size(PREFERENCE_TABLE),
        check_type(dvar,COST),
        length(PREFERENCE_TABLE,LP),
        COST#>=1,
        COST#=<LP,
        PREFERENCE_TABLE=[[_31682-L]|_31678],
        length(VECTOR,LV),
        length(L,N),
        N>=1,
        LV=N,
        create_collection(PREFERENCE_TABLE,vec,var,PREF),
        eval(lex_alldifferent(PREF)),
        eval(in_relation(VECTOR,PREFERENCE_TABLE)),
        cond_lex(VECTOR,PREFERENCE_TABLE,COST).
```

B.83 cond_lex_greater

◇ **META-DATA:**

```
ctr_date(cond_lex_greater,['20060430']).

ctr_origin(
    cond_lex_greater,
    Inspired by \cite{WallaceWilson06}.,
    []).

ctr_types(
    cond_lex_greater,
    ['TUPLE_OF_VALS'-collection(val-int)]).

ctr_arguments(
    cond_lex_greater,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar),
     'PREFERENCE_TABLE'-collection(tuple-'TUPLE_OF_VALS')]).

ctr_restrictions(
    cond_lex_greater,
    [size('TUPLE_OF_VALS')>=1,
     required('TUPLE_OF_VALS',val),
     required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2'),
     size('VECTOR1')=size('TUPLE_OF_VALS'),
     required('PREFERENCE_TABLE',tuple),
     same_size('PREFERENCE_TABLE',tuple),
     distinct('PREFERENCE_TABLE',[]),
     in_relation('VECTOR1','PREFERENCE_TABLE'),
     in_relation('VECTOR2','PREFERENCE_TABLE')]).

ctr_example(
    cond_lex_greater,
    cond_lex_greater(
        [[var-0],[var-0]],
        [[var-1],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]]])).

ctr_typical(
```

```
    cond_lex_greater,
    [size('TUPLE_OF_VALS')>1,
     size('VECTOR1')>1,
     size('VECTOR2')>1,
     size('PREFERENCE_TABLE')>1]).

ctr_exchangeable(
    cond_lex_greater,
    [items_sync(
         VECTOR1,
         VECTOR2,
         'PREFERENCE_TABLE'^tuple,
         all),
     vals(
         ['VECTOR1','VECTOR2','PREFERENCE_TABLE'^tuple],
         int,
         =\=,
         all,
         dontcare)]).

ctr_eval(cond_lex_greater,[automata(cond_lex_greater_a)]).

cond_lex_greater_a(VECTOR1,VECTOR2,PREFERENCE_TABLE) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        collection(PREFERENCE_TABLE,[col([dvar])]),
        same_size(PREFERENCE_TABLE),
        PREFERENCE_TABLE=[[_31187-L]|_R],
        length(VECTOR1,LV1),
        length(VECTOR2,LV2),
        length(L,N),
        N>=1,
        LV1=LV2,
        LV1=N,
        create_collection(PREFERENCE_TABLE,vec,var,PREF),
        eval(lex_alldifferent(PREF)),
        eval(in_relation(VECTOR1,PREFERENCE_TABLE)),
        eval(in_relation(VECTOR2,PREFERENCE_TABLE)),
        cond_lex(VECTOR1,VECTOR2,PREFERENCE_TABLE,I,J),
        I#>J.
```

B.84 cond_lex_greatereq

◇ **META-DATA:**

```
ctr_date(cond_lex_greatereq,['20060416']).

ctr_origin(
    cond_lex_greatereq,
    Inspired by \cite{WallaceWilson06}.,
    []).

ctr_types(
    cond_lex_greatereq,
    ['TUPLE_OF_VALS'-collection(val-int)]).

ctr_arguments(
    cond_lex_greatereq,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar),
     'PREFERENCE_TABLE'-collection(tuple-'TUPLE_OF_VALS')]).

ctr_restrictions(
    cond_lex_greatereq,
    [size('TUPLE_OF_VALS')>=1,
     required('TUPLE_OF_VALS',val),
     required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2'),
     size('VECTOR1')=size('TUPLE_OF_VALS'),
     required('PREFERENCE_TABLE',tuple),
     same_size('PREFERENCE_TABLE',tuple),
     distinct('PREFERENCE_TABLE',[]),
     in_relation('VECTOR1','PREFERENCE_TABLE'),
     in_relation('VECTOR2','PREFERENCE_TABLE')]).

ctr_example(
    cond_lex_greatereq,
    cond_lex_greatereq(
        [[var-0],[var-0]],
        [[var-1],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]]])).

ctr_typical(
```

```
    cond_lex_greatereq,
    [size('TUPLE_OF_VALS')>1,
     size('VECTOR1')>1,
     size('VECTOR2')>1,
     size('PREFERENCE_TABLE')>1]).

ctr_exchangeable(
    cond_lex_greatereq,
    [items_sync(
         VECTOR1,
         VECTOR2,
         'PREFERENCE_TABLE'^tuple,
         all),
     vals(
         ['VECTOR1','VECTOR2','PREFERENCE_TABLE'^tuple],
         int,
         =\=,
         all,
         dontcare)]).

ctr_eval(cond_lex_greatereq,[automata(cond_lex_greatereq_a)]).

cond_lex_greatereq_a(VECTOR1,VECTOR2,PREFERENCE_TABLE) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        collection(PREFERENCE_TABLE,[col([dvar])]),
        same_size(PREFERENCE_TABLE),
        PREFERENCE_TABLE=[[_31199-L]|_R],
        length(VECTOR1,LV1),
        length(VECTOR2,LV2),
        length(L,N),
        N>=1,
        LV1=LV2,
        LV1=N,
        create_collection(PREFERENCE_TABLE,vec,var,PREF),
        eval(lex_alldifferent(PREF)),
        eval(in_relation(VECTOR1,PREFERENCE_TABLE)),
        eval(in_relation(VECTOR2,PREFERENCE_TABLE)),
        cond_lex(VECTOR1,VECTOR2,PREFERENCE_TABLE,I,J),
        I#>=J.
```

B.85 cond_lex_less

◇ **META-DATA:**

```
ctr_date(cond_lex_less,['20060430']).

ctr_origin(
    cond_lex_less,
    Inspired by \cite{WallaceWilson06}.,
    []).

ctr_types(cond_lex_less,['TUPLE_OF_VALS'-collection(val-int)]).

ctr_arguments(
    cond_lex_less,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar),
     'PREFERENCE_TABLE'-collection(tuple-'TUPLE_OF_VALS')]).

ctr_restrictions(
    cond_lex_less,
    [size('TUPLE_OF_VALS')>=1,
     required('TUPLE_OF_VALS',val),
     required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2'),
     size('VECTOR1')=size('TUPLE_OF_VALS'),
     required('PREFERENCE_TABLE',tuple),
     same_size('PREFERENCE_TABLE',tuple),
     distinct('PREFERENCE_TABLE',[]),
     in_relation('VECTOR1','PREFERENCE_TABLE'),
     in_relation('VECTOR2','PREFERENCE_TABLE')]).

ctr_example(
    cond_lex_less,
    cond_lex_less(
        [[var-1],[var-0]],
        [[var-0],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]]])).

ctr_typical(
    cond_lex_less,
    [size('TUPLE_OF_VALS')>1,
```

```
     size('VECTOR1')>1,
     size('VECTOR2')>1,
     size('PREFERENCE_TABLE')>1]).

ctr_exchangeable(
    cond_lex_less,
    [items_sync(
         VECTOR1,
         VECTOR2,
         'PREFERENCE_TABLE'^tuple,
         all),
     vals(
         ['VECTOR1','VECTOR2','PREFERENCE_TABLE'^tuple],
         int,
         =\=,
         all,
         dontcare)]).

ctr_eval(cond_lex_less,[automata(cond_lex_less_a)]).

cond_lex_less_a(VECTOR1,VECTOR2,PREFERENCE_TABLE) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        collection(PREFERENCE_TABLE,[col([dvar])]),
        same_size(PREFERENCE_TABLE),
        PREFERENCE_TABLE=[[_31183-L]|_R],
        length(VECTOR1,LV1),
        length(VECTOR2,LV2),
        length(L,N),
        N>=1,
        LV1=LV2,
        LV1=N,
        create_collection(PREFERENCE_TABLE,vec,var,PREF),
        eval(lex_alldifferent(PREF)),
        eval(in_relation(VECTOR1,PREFERENCE_TABLE)),
        eval(in_relation(VECTOR2,PREFERENCE_TABLE)),
        cond_lex(VECTOR1,VECTOR2,PREFERENCE_TABLE,I,J),
        I#<J.
```

B.86 cond_lex_lesseq

◇ **META-DATA:**

```
ctr_date(cond_lex_lesseq,['20060416']).

ctr_origin(
    cond_lex_lesseq,
    Inspired by \cite{WallaceWilson06}.,
    []).

ctr_types(
    cond_lex_lesseq,
    ['TUPLE_OF_VALS'-collection(val-int)]).

ctr_arguments(
    cond_lex_lesseq,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar),
     'PREFERENCE_TABLE'-collection(tuple-'TUPLE_OF_VALS')]).

ctr_restrictions(
    cond_lex_lesseq,
    [size('TUPLE_OF_VALS')>=1,
     required('TUPLE_OF_VALS',val),
     required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2'),
     size('VECTOR1')=size('TUPLE_OF_VALS'),
     required('PREFERENCE_TABLE',tuple),
     same_size('PREFERENCE_TABLE',tuple),
     distinct('PREFERENCE_TABLE',[]),
     in_relation('VECTOR1','PREFERENCE_TABLE'),
     in_relation('VECTOR2','PREFERENCE_TABLE')]).

ctr_example(
    cond_lex_lesseq,
    cond_lex_lesseq(
        [[var-1],[var-0]],
        [[var-0],[var-0]],
        [[tuple-[[val-1],[val-0]]],
         [tuple-[[val-0],[val-1]]],
         [tuple-[[val-0],[val-0]]],
         [tuple-[[val-1],[val-1]]]])).

ctr_typical(
```

```
    cond_lex_lesseq,
    [size('TUPLE_OF_VALS')>1,
     size('VECTOR1')>1,
     size('VECTOR2')>1,
     size('PREFERENCE_TABLE')>1]).

ctr_exchangeable(
    cond_lex_lesseq,
    [items_sync(
         VECTOR1,
         VECTOR2,
         'PREFERENCE_TABLE'^tuple,
         all),
     vals(
         ['VECTOR1','VECTOR2','PREFERENCE_TABLE'^tuple],
         int,
         =\=,
         all,
         dontcare)]).

ctr_eval(cond_lex_lesseq,[automata(cond_lex_lesseq_a)]).

cond_lex_lesseq_a(VECTOR1,VECTOR2,PREFERENCE_TABLE) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        collection(PREFERENCE_TABLE,[col([dvar])]),
        same_size(PREFERENCE_TABLE),
        PREFERENCE_TABLE=[[_31195-L]|_R],
        length(VECTOR1,LV1),
        length(VECTOR2,LV2),
        length(L,N),
        N>=1,
        LV1=LV2,
        LV1=N,
        create_collection(PREFERENCE_TABLE,vec,var,PREF),
        eval(lex_alldifferent(PREF)),
        eval(in_relation(VECTOR1,PREFERENCE_TABLE)),
        eval(in_relation(VECTOR2,PREFERENCE_TABLE)),
        cond_lex(VECTOR1,VECTOR2,PREFERENCE_TABLE,I,J),
        I#=<J.
```

B.87 connect_points

◇ **META-DATA:**

```
ctr_date(
    connect_points,
    ['20000128','20030820','20040530','20060806']).

ctr_origin(connect_points,'N.~Beldiceanu',[]).

ctr_arguments(
    connect_points,
    ['SIZE1'-int,
     'SIZE2'-int,
     'SIZE3'-int,
     'NGROUP'-dvar,
     'POINTS'-collection(p-dvar)]).

ctr_restrictions(
    connect_points,
    ['SIZE1'>0,
     'SIZE2'>0,
     'SIZE3'>0,
     'NGROUP'>=0,
     'NGROUP'=<size('POINTS'),
     'SIZE1'*'SIZE2'*'SIZE3'=size('POINTS'),
     required('POINTS',p)]).

ctr_example(
    connect_points,
    connect_points(
        8,
        4,
        2,
        2,
        [[p-0],
         [p-0],
         [p-1],
         [p-1],
         [p-0],
         [p-2],
         [p-0],
         [p-0],
         [p-0],
         [p-0],
         [p-0],
```

```
[p-1],
[p-0],
[p-2],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-1],
[p-1],
[p-1],
[p-1],
[p-1],
[p-0],
[p-2],
[p-0],
[p-1],
[p-0],
[p-2],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-0],
[p-2],
[p-0],
[p-0],
[p-0],
[p-2],
[p-2],
[p-2],
[p-2],
[p-2],
[p-0],
[p-0],
[p-0],
```

```
          [p-2],
          [p-0],
          [p-0],
          [p-0],
          [p-2],
          [p-0],
          [p-0]])).

ctr_typical(
    connect_points,
    ['SIZE1'>1,
     'SIZE2'>1,
     'NGROUP'>0,
     'NGROUP'<size('POINTS'),
     size('POINTS')>3]).

ctr_exchangeable(
    connect_points,
    [vals(['POINTS'^p],int(=\=(0)),=\=,all,dontcare)]).

ctr_graph(
    connect_points,
    ['POINTS'],
    2,
    ['GRID'(['SIZE1','SIZE2','SIZE3'])>>
     collection(points1,points2)],
    [points1^p=\=0,points1^p=points2^p],
    ['NSCC'='NGROUP'],
    ['SYMMETRIC']).

ctr_functional_dependency(connect_points,4,[1,2,3,5]).
```

B.88 connected

◇ META-DATA:

```
ctr_date(connected,['20061001']).

ctr_origin(connected,'\\cite{Dooms06}',[]).

ctr_arguments(
    connected,
    ['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    connected,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    connected,
    connected(
        [[index-1,succ-{1,2,3}],
         [index-2,succ-{1,3}],
         [index-3,succ-{1,2,4}],
         [index-4,succ-{3,5,6}],
         [index-5,succ-{4}],
         [index-6,succ-{4}]])).

ctr_typical(connected,[size('NODES')>1]).

ctr_exchangeable(connected,[items('NODES',all)]).

ctr_graph(
    connected,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ],
    ['NCC'=1],
    ['SYMMETRIC']).

ctr_application(connected,[1]).
```

B.89 consecutive_groups_of_ones

◇ **META-DATA:**

```
ctr_date(consecutive_groups_of_ones,['20091227']).

ctr_origin(
    consecutive_groups_of_ones,
    Derived from %c,
    [group]).

ctr_arguments(
    consecutive_groups_of_ones,
    ['GROUP_SIZES'-collection(nb-int),
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    consecutive_groups_of_ones,
    [required('GROUP_SIZES',nb),
     size('GROUP_SIZES')>=1,
     'GROUP_SIZES'^nb>=1,
     'GROUP_SIZES'^nb=<size('VARIABLES'),
     required('VARIABLES',var),
     size('VARIABLES')>=2*size('GROUP_SIZES')-1,
     size('VARIABLES')>=
     sum('GROUP_SIZES'^nb)+size('GROUP_SIZES')-1,
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    consecutive_groups_of_ones,
    consecutive_groups_of_ones(
        [[nb-2],[nb-1]],
        [[var-1],
         [var-1],
         [var-0],
         [var-0],
         [var-0],
         [var-1],
         [var-0]])).

ctr_typical(
    consecutive_groups_of_ones,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_typical_model(
```

```
    consecutive_groups_of_ones,
    [sum('VARIABLES'^var)>2]).

ctr_exchangeable(
    consecutive_groups_of_ones,
    [items_sync('GROUP_SIZES','VARIABLES',reverse)]).

ctr_eval(
    consecutive_groups_of_ones,
    [automaton(consecutive_groups_of_ones_a)]).

consecutive_groups_of_ones_a(FLAG,GROUP_SIZES,VARIABLES) :-
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,N),
        collection(GROUP_SIZES,[int(1,N)]),
        length(GROUP_SIZES,M),
        M>=1,
        N>=M,
        N>=2*M-1,
        get_attr1(GROUP_SIZES,SIZES),
        get_attr1(VARIABLES,VARS),
        get_sum(SIZES,S),
        N>=S+M-1,
        consecutive_groups_of_ones_transitions(
            SIZES,
            -1,
            TRANSITIONS,
            LAST),
        AUTOMATON=
        automaton(
            VARS,
            _30480,
            VARS,
            [source(0),sink(LAST)],
            TRANSITIONS,
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

consecutive_groups_of_ones_transitions([],P,[arc(P,0,P)],P).

consecutive_groups_of_ones_transitions([N|R],P,L,Last) :-
        P1 is P+1,
        PN is N+P1,
        (   P>=0 ->
```

```
            L1=[arc(P,0,P1),arc(P1,0,P1)]
        ;   L1=[arc(P1,0,P1)]
        ),
        consecutive_groups_of_ones_trans(N,P1,L2),
        consecutive_groups_of_ones_transitions(R,PN,L3,Last),
        append(L1,L2,L12),
        append(L12,L3,L).

consecutive_groups_of_ones_trans(0,_27153,[]) :-
        !.

consecutive_groups_of_ones_trans(I,P,[arc(P,1,P1)|R]) :-
        I>0,
        P1 is P+1,
        I1 is I-1,
        consecutive_groups_of_ones_trans(I1,P1,R).
```

B.90 consecutive_values

♢ META-DATA:

```
ctr_predefined(consecutive_values).

ctr_date(consecutive_values,['20100106']).

ctr_origin(
    consecutive_values,
    Derived from %c.,
    [alldifferent_consecutive_values]).

ctr_arguments(
    consecutive_values,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    consecutive_values,
    [required('VARIABLES',var)]).

ctr_example(
    consecutive_values,
    consecutive_values([[var-5],[var-4],[var-3],[var-5]])).

ctr_typical(
    consecutive_values,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_typical_model(consecutive_values,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    consecutive_values,
    [items('VARIABLES',all),translate(['VARIABLES'^var])]).

ctr_eval(
    consecutive_values,
    [checker(consecutive_values_c),
     reformulation(consecutive_values_r)]).

ctr_cond_imply(
    consecutive_values,
    some_equal,
    [size('VARIABLES')>range('VARIABLES'^var)],
    [],
    id).
```

```
ctr_sol(consecutive_values,2,0,2,7,-).

ctr_sol(consecutive_values,3,0,3,34,-).

ctr_sol(consecutive_values,4,0,4,217,-).

ctr_sol(consecutive_values,5,0,5,1716,-).

ctr_sol(consecutive_values,6,0,6,16159,-).

ctr_sol(consecutive_values,7,0,7,176366,-).

ctr_sol(consecutive_values,8,0,8,2187637,-).

consecutive_values_c([]) :-
        !.

consecutive_values_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        min_member(MIN,VARS),
        max_member(MAX,VARS),
        RANGE is MAX-MIN+1,
        length(VARS,N),
        N>=RANGE,
        sort(VARS,S),
        length(S,RANGE).

consecutive_values_r([]) :-
        !.

consecutive_values_r(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        minimum(MIN,VARS),
        maximum(MAX,VARS),
        length(VARIABLES,N),
        NVAL in 1..N,
        nvalue(NVAL,VARS),
        NVAL#=MAX-MIN+1.
```

B.91 contains_sboxes

◇ META-DATA:

```
ctr_date(contains_sboxes,['20070622','20090725']).

ctr_origin(
    contains_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(contains_sboxes,[contains]).

ctr_types(
    contains_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    contains_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    contains_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    contains_sboxes,
    contains_sboxes(
        2,
        {0,1},
        [[oid-1,sid-1,x-[[v-1],[v-1]]],
         [oid-2,sid-2,x-[[v-2],[v-2]]],
         [oid-3,sid-3,x-[[v-3],[v-3]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-5],[v-5]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-3]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-1],[v-1]]]])).

ctr_typical(contains_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    contains_sboxes,
    [items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all)]).

ctr_eval(contains_sboxes,[logic(contains_sboxes_g)]).

ctr_logic(
    contains_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (contains_sboxes(Dims,O1,S1,O2,S2)--->
      forall(
          D,
          Dims,
          origin(O1,S1,D)#<origin(O2,S2,D)#/\
          end(O2,S2,D)#<end(O1,S1,D))),
     (contains_objects(Dims,O1,O2)--->
      forall(
          S1,
          sboxes([O1^sid]),
          exists(
              S2,
              sboxes([O2^sid]),
```

```
                 contains_sboxes(Dims,O1,S1,O2,S2)))),
      (all_contains(Dims,OIDS)--->
       forall(
           O1,
           objects(OIDS),
           forall(
               O2,
               objects(OIDS),
               O1^oid#<O2^oid#=>
               contains_objects(Dims,O1,O2)))),
      all_contains(DIMENSIONS,OIDS)]).

ctr_contractible(contains_sboxes,[],'OBJECTS',suffix).

ctr_application(contains_sboxes,[3]).

contains_sboxes_g(K,_38219,[],_38221) :-
        !,
        check_type(int_gteq(1),K).

contains_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        collection_increasing_seq(OBJECTS,[1]),
        geost1(OIDS,SIDS,XS,Objects),
        geost2(SIDES,TS,TL,Sboxes),
        geost_dims(1,K,DIMENSIONS),
        ctr_logic(contains_sboxes,[DIMENSIONS,OIDS],Rules),
        geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.92 correspondence

◊ **META-DATA:**

```
ctr_date(correspondence,['20030820','20060806']).

ctr_origin(
    correspondence,
    Derived from %c by removing the sorting condition.,
    [sort_permutation]).

ctr_arguments(
    correspondence,
    ['FROM'-collection(from-dvar),
     'PERMUTATION'-collection(var-dvar),
     'TO'-collection(tvar-dvar)]).

ctr_restrictions(
    correspondence,
    [size('PERMUTATION')=size('FROM'),
     size('PERMUTATION')=size('TO'),
     'PERMUTATION'^var>=1,
     'PERMUTATION'^var=<size('PERMUTATION'),
     alldifferent('PERMUTATION'),
     required('FROM',from),
     required('PERMUTATION',var),
     required('TO',tvar)]).

ctr_example(
    correspondence,
    correspondence(
        [[from-1],
         [from-9],
         [from-1],
         [from-5],
         [from-2],
         [from-1]],
        [[var-6],[var-1],[var-3],[var-5],[var-4],[var-2]],
        [[tvar-9],
         [tvar-1],
         [tvar-1],
         [tvar-2],
         [tvar-5],
         [tvar-1]])).

ctr_typical(
```

```
    correspondence,
    [size('FROM')>1,range('FROM'^from)>1]).

ctr_exchangeable(
    correspondence,
    [vals(['FROM'^from,'TO'^tvar],int,=\=,all,dontcare)]).

ctr_derived_collections(
    correspondence,
    [col('FROM_PERMUTATION'-collection(from-dvar,var-dvar),
         [item(from-'FROM'^from,var-'PERMUTATION'^var)])]).

ctr_graph(
    correspondence,
    ['FROM_PERMUTATION','TO'],
    2,
    ['PRODUCT'>>collection(from_permutation,to)],
    [from_permutation^from=to^tvar,
     from_permutation^var=to^key],
    ['NARC'=size('PERMUTATION')],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(correspondence,[reformulation(correspondence_r)]).

correspondence_r(FROM,PERMUTATION,TO) :-
        collection(FROM,[dvar]),
        length(FROM,NFROM),
        collection(PERMUTATION,[dvar(1,NFROM)]),
        length(PERMUTATION,NPERMUTATION),
        collection(TO,[dvar]),
        length(TO,NTO),
        NPERMUTATION=NFROM,
        NPERMUTATION=NTO,
        get_attr1(FROM,FROMS),
        get_attr1(PERMUTATION,PERMS),
        get_attr1(TO,TOS),
        all_different(PERMS),
        correspondence1(PERMS,FROMS,TOS).

correspondence1([],[],_51997).

correspondence1([Pi|R],[Fi|S],TOS) :-
        element(Pi,TOS,Fi),
        correspondence1(R,S,TOS).
```

B.93 count

♢ **META-DATA:**

```
ctr_date(
    count,
    ['20000128','20030820','20040530','20060806','20100204']).

ctr_origin(count,'\\cite{Sicstus95}',[]).

ctr_synonyms(count,[occurencemax,occurencemin,occurrence]).

ctr_arguments(
    count,
    ['VALUE'-int,
     'VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    count,
    [required('VARIABLES',var),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    count,
    count(5,[[var-4],[var-5],[var-5],[var-4],[var-5]],>=,2)).

ctr_typical(
    count,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('RELOP',[=,<,>=,>,=<]),
     'LIMIT'>0,
     'LIMIT'<size('VARIABLES')]).

ctr_exchangeable(
    count,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         int(=\=('VALUE')),
         =\=,
         dontcare,
         dontcare)]).
```

```
ctr_graph(
    count,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE'],
    ['RELOP'('NARC','LIMIT')],
    []).

ctr_eval(count,[reformulation(count_r),automaton(count_a)]).

ctr_pure_functional_dependency(count,[in_list('RELOP',[=])]).

ctr_contractible(
    count,
    [in_list('RELOP',[<,=<])],
    VARIABLES,
    any).

ctr_extensible(count,[in_list('RELOP',[>=,>])],'VARIABLES',any).

ctr_aggregate(
    count,
    [in_list('RELOP',[<,=<,>=,>])],
    [id,union,id,+]).

count_r(VALUE,VARIABLES,RELOP,LIMIT) :-
        check_type(int,VALUE),
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        length(VARIABLES,NVARIABLES),
        N in 0..NVARIABLES,
        eval(among(N,VARIABLES,[[val-VALUE]])),
        call_term_relop_value(N,RELOP,LIMIT).

count_a(FLAG,VALUE,VARIABLES,RELOP,LIMIT) :-
        check_type(int,VALUE),
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        count_signature(VARIABLES,SIGNATURE,VALUE),
        automaton(
            SIGNATURE,
            _54125,
            SIGNATURE,
```

```
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [NIN]),
        count_relop(RELOP,NIN,LIMIT,FLAG).

count_signature([],[],_52651).

count_signature([[var-VAR]|VARs],[S|Ss],VALUE) :-
        VAR#=VALUE#<=>S,
        count_signature(VARs,Ss,VALUE).
```

B.94 counts

◇ **META-DATA:**

```
ctr_date(counts,['20030820','20040530','20060806']).

ctr_origin(counts,'Derived from %c.',[count]).

ctr_arguments(
    counts,
    ['VALUES'-collection(val-int),
     'VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    counts,
    [required('VALUES',val),
     distinct('VALUES',val),
     required('VARIABLES',var),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    counts,
    counts(
        [[val-1],[val-3],[val-4],[val-9]],
        [[var-4],[var-5],[var-5],[var-4],[var-1],[var-5]],
        =,
        3)).

ctr_typical(
    counts,
    [size('VALUES')>1,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VARIABLES')>size('VALUES'),
     in_list('RELOP',[=,<,>=,>,=<]),
     'LIMIT'>0,
     'LIMIT'<size('VARIABLES')]).

ctr_exchangeable(
    counts,
    [items('VALUES',all),
     items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
```

```
          comp('VALUES'^val),
          =,
          dontcare,
          dontcare)]).

ctr_graph(
    counts,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['RELOP'('NARC','LIMIT')],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(counts,[reformulation(counts_r),automaton(counts_a)]).

ctr_pure_functional_dependency(counts,[in_list('RELOP',[=])]).

ctr_contractible(
    counts,
    [in_list('RELOP',[<,=<])],
    VARIABLES,
    any).

ctr_extensible(
    counts,
    [in_list('RELOP',[>=,>])],
    VARIABLES,
    any).

ctr_aggregate(
    counts,
    [in_list('RELOP',[<,=<,>=,>])],
    [sunion,union,id,+]).

counts_r(VALUES,VARIABLES,RELOP,LIMIT) :-
        collection(VALUES,[int]),
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        get_attr1(VALUES,VALS),
        all_different(VALS),
        length(VARIABLES,NVARIABLES),
        N in 0..NVARIABLES,
        eval(among(N,VARIABLES,VALUES)),
        call_term_relop_value(N,RELOP,LIMIT).
```

```
counts_a(FLAG,VALUES,VARIABLES,RELOP,LIMIT) :-
        collection(VALUES,[int]),
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        get_attr1(VALUES,LIST_VALUES),
        all_different(LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        counts_signature(VARIABLES,SIGNATURE,SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _51562,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [NIN]),
        count_relop(RELOP,NIN,LIMIT,FLAG).

counts_signature([],[],_49596).

counts_signature([[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
        VAR in_set SET_OF_VALUES#<=>S,
        counts_signature(VARs,Ss,SET_OF_VALUES).
```

B.95 coveredby_sboxes

◇ **META-DATA:**

```
ctr_date(coveredby_sboxes,['20070622','20090725']).

ctr_origin(
    coveredby_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(coveredby_sboxes,[coveredby]).

ctr_types(
    coveredby_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    coveredby_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    coveredby_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     required('SBOXES',[sid,t,l]),
     size('SBOXES')>=1,
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    coveredby_sboxes,
    coveredby_sboxes(
        2,
        {0,1},
        [[oid-1,sid-4,x-[[v-2],[v-3]]],
         [oid-2,sid-2,x-[[v-2],[v-2]]],
         [oid-3,sid-1,x-[[v-1],[v-1]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-3],[v-3]]],
         [sid-1,t-[[v-3],[v-0]],l-[[v-2],[v-2]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-2],[v-2]]],
         [sid-2,t-[[v-2],[v-0]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-2],[v-2]]],
         [sid-3,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-4,t-[[v-0],[v-0]],l-[[v-1],[v-1]]]])).

ctr_typical(coveredby_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    coveredby_sboxes,
    [items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all)]).

ctr_eval(coveredby_sboxes,[logic(coveredby_sboxes_g)]).

ctr_logic(
    coveredby_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (coveredby_sboxes(Dims,O1,S1,O2,S2)--->
      forall(
          D,
          Dims,
          origin(O2,S2,D)#=<origin(O1,S1,D)#/\
          end(O1,S1,D)#=<end(O2,S2,D))#/\
      exists(
          D,
          Dims,
```

```
          origin(O2,S2,D)#=origin(O1,S1,D)#\/
          end(O1,S1,D)#=end(O2,S2,D))),
    (coveredby_objects(Dims,O1,O2)--->
     forall(
          S1,
          sboxes([O1^sid]),
          exists(
              S2,
              sboxes([O2^sid]),
              coveredby_sboxes(Dims,O1,S1,O2,S2)))),
    (all_coveredby(Dims,OIDS)--->
     forall(
          O1,
          objects(OIDS),
          forall(
              O2,
              objects(OIDS),
              O1^oid#<O2^oid#=>
              coveredby_objects(Dims,O1,O2)))),
    all_coveredby(DIMENSIONS,OIDS)]).

ctr_application(coveredby_sboxes,[3]).

coveredby_sboxes_g(K,_40040,[],_40042) :-
        !,
        check_type(int_gteq(1),K).

coveredby_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        collection_increasing_seq(OBJECTS,[1]),
        geost1(OIDS,SIDS,XS,Objects),
        geost2(SIDES,TS,TL,Sboxes),
```

```
geost_dims(1,K,DIMENSIONS),
ctr_logic(coveredby_sboxes,[DIMENSIONS,OIDS],Rules),
geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.96 covers_sboxes

◇ **META-DATA:**

```
ctr_date(covers_sboxes,['20070622','20090725']).

ctr_origin(
    covers_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(covers_sboxes,[covers]).

ctr_types(
    covers_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    covers_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    covers_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    covers_sboxes,
    covers_sboxes(
        2,
        {0,1},
        [[oid-1,sid-1,x-[[v-1],[v-1]]],
         [oid-2,sid-2,x-[[v-2],[v-2]]],
         [oid-3,sid-4,x-[[v-2],[v-3]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-3],[v-3]]],
         [sid-1,t-[[v-3],[v-0]],l-[[v-2],[v-2]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-2],[v-2]]],
         [sid-2,t-[[v-2],[v-0]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-2],[v-2]]],
         [sid-3,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-4,t-[[v-0],[v-0]],l-[[v-1],[v-1]]]])).

ctr_typical(covers_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    covers_sboxes,
    [items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all)]).

ctr_eval(covers_sboxes,[logic(covers_sboxes_g)]).

ctr_logic(
    covers_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (covers_sboxes(Dims,O1,S1,O2,S2)--->
      forall(
          D,
          Dims,
          origin(O1,S1,D)#=<origin(O2,S2,D)#/\
          end(O2,S2,D)#=<end(O1,S1,D))#/\
      exists(
          D,
          Dims,
```

```
          origin(O1,S1,D)#=origin(O2,S2,D)#\/
          end(O1,S1,D)#=end(O2,S2,D))),
     (covers_objects(Dims,O1,O2)--->
      forall(
          S2,
          sboxes([O2^sid]),
          exists(
              S1,
              sboxes([O1^sid]),
              covers_sboxes(Dims,O1,S1,O2,S2)))),
     (all_covers(Dims,OIDS)--->
      forall(
          O1,
          objects(OIDS),
          forall(
              O2,
              objects(OIDS),
              O1^oid#<O2^oid#=>covers_objects(Dims,O1,O2)))),
     all_covers(DIMENSIONS,OIDS)]).

ctr_contractible(covers_sboxes,[],'OBJECTS',suffix).

ctr_application(covers_sboxes,[3]).

covers_sboxes_g(K,_40345,[],_40347) :-
        !,
        check_type(int_gteq(1),K).

covers_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        collection_increasing_seq(OBJECTS,[1]),
        geost1(OIDS,SIDS,XS,Objects),
```

```
geost2(SIDES,TS,TL,Sboxes),
geost_dims(1,K,DIMENSIONS),
ctr_logic(covers_sboxes,[DIMENSIONS,OIDS],Rules),
geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.97 crossing

◇ **META-DATA:**

```
ctr_date(crossing,['20000128','20030820','20060806']).

ctr_origin(
    crossing,
    Inspired by \cite{CormenLeisersonRivest90}.,
    []).

ctr_arguments(
    crossing,
    ['NCROSS'-dvar,
     'SEGMENTS'-collection(ox-dvar,oy-dvar,ex-dvar,ey-dvar)]).

ctr_restrictions(
    crossing,
    ['NCROSS'>=0,
     NCROSS=<
     (size('SEGMENTS')*size('SEGMENTS')-size('SEGMENTS'))/2,
     required('SEGMENTS',[ox,oy,ex,ey])]).

ctr_example(
    crossing,
    crossing(
        3,
        [[ox-1,oy-4,ex-9,ey-2],
         [ox-1,oy-1,ex-3,ey-5],
         [ox-3,oy-2,ex-7,ey-4],
         [ox-9,oy-1,ex-9,ey-4]])).

ctr_typical(crossing,[size('SEGMENTS')>1]).

ctr_exchangeable(
    crossing,
    [items('SEGMENTS',all),
     attrs_sync('SEGMENTS',[[ox,oy],[ex,ey]]),
     translate(['SEGMENTS'^ox,'SEGMENTS'^ex]),
     translate(['SEGMENTS'^oy,'SEGMENTS'^ey])]).

ctr_graph(
    crossing,
    ['SEGMENTS'],
    2,
    ['CLIQUE'(<)>>collection(s1,s2)],
```

```
    [max(s1^ox,s1^ex)>=min(s2^ox,s2^ex),
     max(s2^ox,s2^ex)>=min(s1^ox,s1^ex),
     max(s1^oy,s1^ey)>=min(s2^oy,s2^ey),
     max(s2^oy,s2^ey)>=min(s1^oy,s1^ey),
     (s2^ox-s1^ex)*(s1^ey-s1^oy)-
     (s1^ex-s1^ox)*(s2^oy-s1^ey)=
     0#\/
     (s2^ex-s1^ex)*(s2^oy-s1^oy)-
     (s2^ox-s1^ox)*(s2^ey-s1^ey)=
     0#\/
     sign(
          (s2^ox-s1^ex)*(s1^ey-s1^oy)-
          (s1^ex-s1^ox)*(s2^oy-s1^ey))=\=
     sign(
          (s2^ex-s1^ex)*(s2^oy-s1^oy)-
          (s2^ox-s1^ox)*(s2^ey-s1^ey))],
    ['NARC'='NCROSS'],
    ['ACYCLIC','NO_LOOP']).

ctr_pure_functional_dependency(crossing,[]).

ctr_functional_dependency(crossing,1,[2]).

ctr_application(crossing,[2]).
```

B.98 cumulative

◊ **META-DATA:**

```
ctr_date(
    cumulative,
    ['20000128','20030820','20040530','20060806','20090923']).

ctr_origin(cumulative,'\\cite{AggounBeldiceanu93}',[]).

ctr_synonyms(cumulative,[cumulative_max]).

ctr_arguments(
    cumulative,
    [TASKS-
     collection(
         origin-dvar,
         duration-dvar,
         end-dvar,
         height-dvar),
     'LIMIT'-int]).

ctr_restrictions(
    cumulative,
    [require_at_least(2,'TASKS',[origin,duration,end]),
     required('TASKS',height),
     'TASKS'^duration>=0,
     'TASKS'^origin=<'TASKS'^end,
     'TASKS'^height>=0,
     'LIMIT'>=0]).

ctr_example(
    cumulative,
    cumulative(
        [[origin-1,duration-3,end-4,height-1],
         [origin-2,duration-9,end-11,height-2],
         [origin-3,duration-10,end-13,height-1],
         [origin-6,duration-6,end-12,height-1],
         [origin-7,duration-2,end-9,height-3]],
        8)).

ctr_typical(
    cumulative,
    [size('TASKS')>1,
     range('TASKS'^origin)>1,
     range('TASKS'^duration)>1,
```

```
     range('TASKS'^end)>1,
     range('TASKS'^height)>1,
     'TASKS'^duration>0,
     'TASKS'^height>0,
     'LIMIT'<sum('TASKS'^height)]).

ctr_exchangeable(
    cumulative,
    [items('TASKS',all),
     vals(['TASKS'^duration],int(>=(0)),>,dontcare,dontcare),
     vals(['TASKS'^height],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'^origin,'TASKS'^end]),
     vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_graph(
    cumulative,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin+tasks^duration=tasks^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    cumulative,
    ['TASKS','TASKS'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1^duration>0,
     tasks2^origin=<tasks1^origin,
     tasks1^origin<tasks2^end],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^height)])]],
    [sum_ctr(variables,=<,'LIMIT')]).

ctr_eval(cumulative,[builtin(cumulative_b)]).

ctr_contractible(cumulative,[],'TASKS',any).

ctr_cond_imply(
    cumulative,
```

```
    coloured_cumulative,
    ['TASKS'^height>0],
    [],
    [same('TASKS'),same('LIMIT')]).

ctr_application(cumulative,[1]).

cumulative_b(TASKS,LIMIT) :-
        collection(TASKS,[dvar,dvar_gteq(0),dvar,dvar_gteq(0)]),
        integer(LIMIT),
        LIMIT>=0,
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        get_attr3(TASKS,ENDS),
        get_attr4(TASKS,HEIGHTS),
        gen_cum_tasks(ORIGINS,DURATIONS,ENDS,HEIGHTS,1,Tasks),
        cumulative(Tasks,[limit(LIMIT)]).
```

B.99 cumulative_convex

♢ META-DATA:

```
ctr_date(cumulative_convex,['20050817','20060807']).

ctr_origin(cumulative_convex,'Derived from %c',[cumulative]).

ctr_types(cumulative_convex,['POINTS'-collection(var-dvar)]).

ctr_arguments(
    cumulative_convex,
    ['TASKS'-collection(points-'POINTS',height-dvar),
     'LIMIT'-int]).

ctr_restrictions(
    cumulative_convex,
    [required('POINTS',var),
     size('POINTS')>0,
     required('TASKS',[points,height]),
     'TASKS'^height>=0,
     'LIMIT'>=0]).

ctr_example(
    cumulative_convex,
    cumulative_convex(
        [[points-[[var-2],[var-1],[var-5]],height-1],
         [points-[[var-4],[var-5],[var-7]],height-2],
         [points-
          [[var-14],[var-13],[var-9],[var-11],[var-10]],
          height-2]],
        3)).

ctr_typical(
    cumulative_convex,
    [size('TASKS')>1,
     'TASKS'^height>0,
     'LIMIT'<sum('TASKS'^height)]).

ctr_exchangeable(
    cumulative_convex,
    [items('TASKS',all),
     items('TASKS'^points,all),
     vals(['TASKS'^height],int(>=(0)),>,dontcare,dontcare),
     vals(['LIMIT'],int,<,dontcare,dontcare)]).
```

```
ctr_derived_collections(
    cumulative_convex,
    [col('INSTANTS'-collection(instant-dvar),
         [item(instant-'TASKS'^points^var)])]).

ctr_graph(
    cumulative_convex,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [alldifferent(tasks^points)],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    cumulative_convex,
    ['INSTANTS','TASKS'],
    2,
    ['PRODUCT'>>collection(instants,tasks)],
    [between_min_max(instants^instant,tasks^points)],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^height)])]],
    [sum_ctr(variables,=<,'LIMIT')]).

ctr_contractible(cumulative_convex,[],'TASKS',any).

ctr_application(cumulative_convex,[1]).
```

B.100 cumulative_product

◇ META-DATA:

```
ctr_date(cumulative_product,['20030820','20060807','20081227']).

ctr_origin(cumulative_product,'Derived from %c.',[cumulative]).

ctr_arguments(
    cumulative_product,
    [TASKS-
     collection(
         origin-dvar,
         duration-dvar,
         end-dvar,
         height-dvar),
     'LIMIT'-int]).

ctr_restrictions(
    cumulative_product,
    [require_at_least(2,'TASKS',[origin,duration,end]),
     required('TASKS',height),
     'TASKS'^duration>=0,
     'TASKS'^origin=<'TASKS'^end,
     'TASKS'^height>=1,
     'LIMIT'>=0]).

ctr_example(
    cumulative_product,
    cumulative_product(
        [[origin-1,duration-3,end-4,height-1],
         [origin-2,duration-9,end-11,height-2],
         [origin-3,duration-10,end-13,height-1],
         [origin-6,duration-6,end-12,height-1],
         [origin-7,duration-2,end-9,height-3]],
        6)).

ctr_typical(
    cumulative_product,
    [size('TASKS')>1,
     range('TASKS'^origin)>1,
     range('TASKS'^duration)>1,
     range('TASKS'^end)>1,
     range('TASKS'^height)>1,
     'TASKS'^duration>0,
     'LIMIT'<prod('TASKS'^height)]).
```

```
ctr_exchangeable(
    cumulative_product,
    [items('TASKS',all),
     vals(['TASKS'^height],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'^origin,'TASKS'^end]),
     vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_graph(
    cumulative_product,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin+tasks^duration=tasks^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    cumulative_product,
    ['TASKS','TASKS'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1^duration>0,
     tasks2^origin=<tasks1^origin,
     tasks1^origin<tasks2^end],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'ITEMS'^height)])]],
    [product_ctr(variables,=<,'LIMIT')]).

ctr_eval(
    cumulative_product,
    [reformulation(cumulative_product_r)]).

ctr_contractible(cumulative_product,[],'TASKS',any).

ctr_application(cumulative_product,[1]).

cumulative_product_r(TASKS,LIMIT) :-
        integer(LIMIT),
        LIMIT>=1,
        collection(
```

```
            TASKS,
            [dvar,dvar_gteq(0),dvar,dvar(1,LIMIT)]),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        get_attr3(TASKS,ENDS),
        get_attr4(TASKS,HEIGHTS),
        ori_dur_end(ORIGINS,DURATIONS,ENDS),
        cumulative_product1(
            ORIGINS,
            ENDS,
            HEIGHTS,
            1,
            ORIGINS,
            ENDS,
            HEIGHTS,
            LIMIT).

cumulative_product1(
    [],
    [],
    [],
    _55154,
    _55201,
    _55248,
    _55295,
    _55342).

cumulative_product1(
    [Oi|RO],
    [Ei|RE],
    [Hi|RH],
    I,
    ORIGINS,
    ENDS,
    HEIGHTS,
    LIMIT) :-
        cumulative_product2(
            ORIGINS,
            ENDS,
            HEIGHTS,
            1,
            I,
            Oi,
            Ei,
            Hi,
            PRODi),
```

```
        call(PRODi#=<LIMIT),
        I1 is I+1,
        cumulative_product1(
            RO,
            RE,
            RH,
            I1,
            ORIGINS,
            ENDS,
            HEIGHTS,
            LIMIT).

cumulative_product2(
    [],
    [],
    [],
    _55157,
    _55204,
    _55251,
    _55298,
    _55345,
    1).

cumulative_product2(
    [_54787|RO],
    [_54791|RE],
    [_54795|RH],
    J,
    I,
    Oi,
    Ei,
    Hi,
    Hi*R) :-
        I=J,
        !,
        J1 is J+1,
        cumulative_product2(RO,RE,RH,J1,I,Oi,Ei,Hi,R).

cumulative_product2([Oj|RO],[Ej|RE],[Hj|RH],J,I,Oi,Ei,Hi,Hij*R) :-
        I=\=J,
        Hij in 1..Hj,
        Oj#=<Oi#/\Ej#>Oi#/\Hij#=Hj#\/
        (Oj#>Oi#\/Ej#=<Oi)#/\Hij#=1,
        J1 is J+1,
        cumulative_product2(RO,RE,RH,J1,I,Oi,Ei,Hi,R).
```

B.101 cumulative_two_d

◇ META-DATA:

```
ctr_predefined(cumulative_two_d).

ctr_date(cumulative_two_d,['20000128','20030820','20060807']).

ctr_origin(
    cumulative_two_d,
    Inspired by %c and %c.,
    [cumulative,diffn]).

ctr_arguments(
    cumulative_two_d,
    [RECTANGLES-
     collection(
         start1-dvar,
         size1-dvar,
         last1-dvar,
         start2-dvar,
         size2-dvar,
         last2-dvar,
         height-dvar),
     'LIMIT'-int]).

ctr_restrictions(
    cumulative_two_d,
    [require_at_least(2,'RECTANGLES',[start1,size1,last1]),
     require_at_least(2,'RECTANGLES',[start2,size2,last2]),
     required('RECTANGLES',height),
     'RECTANGLES'^size1>=0,
     'RECTANGLES'^size2>=0,
     'RECTANGLES'^height>=0,
     'LIMIT'>=0]).

ctr_example(
    cumulative_two_d,
    cumulative_two_d(
        [[start1-1,
          size1-4,
          last1-4,
          start2-3,
          size2-3,
          last2-5,
          height-4],
```

```
          [start1-3,
           size1-2,
           last1-4,
           start2-1,
           size2-2,
           last2-2,
           height-2],
          [start1-1,
           size1-2,
           last1-2,
           start2-1,
           size2-2,
           last2-2,
           height-3],
          [start1-4,
           size1-1,
           last1-4,
           start2-1,
           size2-1,
           last2-1,
           height-1]],
         4)).

ctr_typical(
    cumulative_two_d,
    [size('RECTANGLES')>1,
     'RECTANGLES'^size1>0,
     'RECTANGLES'^size2>0,
     'RECTANGLES'^height>0,
     'LIMIT'<sum('RECTANGLES'^height)]).

ctr_exchangeable(
    cumulative_two_d,
    [items('RECTANGLES',all),
     attrs_sync(
         RECTANGLES,
         [[start1,start2],
          [size1,size2],
          [last1,last2],
          [height]]),
     vals(
         ['RECTANGLES'^height],
         int(>=(0)),
         >,
         dontcare,
         dontcare),
```

```
    translate(['RECTANGLES'^start1,'RECTANGLES'^last1]),
    translate(['RECTANGLES'^start2,'RECTANGLES'^last2]),
    vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_contractible(cumulative_two_d,[],'RECTANGLES',any).

ctr_application(cumulative_two_d,[1]).
```

B.102 cumulative_with_level_of_priority

◇ **META-DATA:**

```
ctr_date(
    cumulative_with_level_of_priority,
    ['20040530','20060807']).

ctr_origin(cumulative_with_level_of_priority,'H.~Simonis',[]).

ctr_arguments(
    cumulative_with_level_of_priority,
    [TASKS-
     collection(
         priority-int,
         origin-dvar,
         duration-dvar,
         end-dvar,
         height-dvar),
     'PRIORITIES'-collection(id-int,capacity-int)]).

ctr_restrictions(
    cumulative_with_level_of_priority,
    [required('TASKS',[priority,height]),
     require_at_least(2,'TASKS',[origin,duration,end]),
     'TASKS'^priority>=1,
     'TASKS'^priority=<size('PRIORITIES'),
     'TASKS'^duration>=0,
     'TASKS'^origin=<'TASKS'^end,
     'TASKS'^height>=0,
     required('PRIORITIES',[id,capacity]),
     'PRIORITIES'^id>=1,
     'PRIORITIES'^id=<size('PRIORITIES'),
     increasing_seq('PRIORITIES',id),
     increasing_seq('PRIORITIES',capacity)]).

ctr_example(
    cumulative_with_level_of_priority,
    cumulative_with_level_of_priority(
        [[priority-1,origin-1,duration-2,end-3,height-1],
         [priority-1,origin-2,duration-3,end-5,height-1],
         [priority-1,origin-5,duration-2,end-7,height-2],
         [priority-2,origin-3,duration-2,end-5,height-2],
         [priority-2,origin-6,duration-3,end-9,height-1]],
        [[id-1,capacity-2],[id-2,capacity-3]])).
```

```
ctr_typical(
    cumulative_with_level_of_priority,
    [size('TASKS')>1,
     range('TASKS'^priority)>1,
     range('TASKS'^origin)>1,
     range('TASKS'^duration)>1,
     range('TASKS'^end)>1,
     range('TASKS'^height)>1,
     'TASKS'^duration>0,
     'TASKS'^height>0,
     size('PRIORITIES')>1,
     'PRIORITIES'^capacity>0,
     'PRIORITIES'^capacity<sum('TASKS'^height),
     size('TASKS')>size('PRIORITIES')]).

ctr_exchangeable(
    cumulative_with_level_of_priority,
    [items('TASKS',all),
     vals(
         ['TASKS'^priority],
         int(=<(size('PRIORITIES'))),
         <,
         dontcare,
         dontcare),
     vals(['TASKS'^height],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'^origin,'TASKS'^end]),
     vals(['PRIORITIES'^capacity],int,<,dontcare,dontcare)]).

ctr_derived_collections(
    cumulative_with_level_of_priority,
    [col(TIME_POINTS-
         collection(idp-int,duration-dvar,point-dvar),
         [item(
              idp-'TASKS'^priority,
              duration-'TASKS'^duration,
              point-'TASKS'^origin),
          item(
              idp-'TASKS'^priority,
              duration-'TASKS'^duration,
              point-'TASKS'^end)])]).

ctr_graph(
    cumulative_with_level_of_priority,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
```

```
    [tasks^origin+tasks^duration=tasks^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    cumulative_with_level_of_priority,
    ['TIME_POINTS','TASKS'],
    2,
    foreach(
        PRIORITIES,
        ['PRODUCT'>>collection(time_points,tasks)]),
    [time_points^idp='PRIORITIES'^id,
     time_points^idp>=tasks^priority,
     time_points^duration>0,
     tasks^origin=<time_points^point,
     time_points^point<tasks^end],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^height)])]],
    [sum_ctr(variables,=<,'PRIORITIES'^capacity)]).

ctr_contractible(
    cumulative_with_level_of_priority,
    [],
    TASKS,
    any).

ctr_application(cumulative_with_level_of_priority,[1]).
```

B.103 cumulatives

◇ **META-DATA:**

```
ctr_date(
    cumulatives,
    ['20000128','20030820','20040530','20060807']).

ctr_origin(cumulatives,'\\cite{BeldiceanuCarlsson02a}',[]).

ctr_arguments(
    cumulatives,
    [TASKS-
     collection(
         machine-dvar,
         origin-dvar,
         duration-dvar,
         end-dvar,
         height-dvar),
     'MACHINES'-collection(id-int,capacity-int),
     'CTR'-atom]).

ctr_restrictions(
    cumulatives,
    [required('TASKS',[machine,height]),
     require_at_least(2,'TASKS',[origin,duration,end]),
     in_attr('TASKS',machine,'MACHINES',id),
     'TASKS'^duration>=0,
     'TASKS'^origin=<'TASKS'^end,
     size('MACHINES')>0,
     required('MACHINES',[id,capacity]),
     distinct('MACHINES',id),
     in_list('CTR',[=<,>=])]).

ctr_example(
    cumulatives,
    cumulatives(
        [[machine-1,origin-2,duration-2,end-4,height- -2],
         [machine-1,origin-1,duration-4,end-5,height-1],
         [machine-1,origin-4,duration-2,end-6,height- -1],
         [machine-1,origin-2,duration-3,end-5,height-2],
         [machine-1,origin-5,duration-2,end-7,height-2],
         [machine-2,origin-3,duration-2,end-5,height- -1],
         [machine-2,origin-1,duration-4,end-5,height-1]],
        [[id-1,capacity-0],[id-2,capacity-0]],
        >=)).
```

```
ctr_typical(
    cumulatives,
    [size('TASKS')>1,
     range('TASKS'^machine)>1,
     range('TASKS'^origin)>1,
     range('TASKS'^duration)>1,
     range('TASKS'^end)>1,
     range('TASKS'^height)>1,
     'TASKS'^duration>0,
     'TASKS'^height=\=0,
     size('MACHINES')>1,
     'MACHINES'^capacity<sum('TASKS'^height),
     size('TASKS')>size('MACHINES')]).

ctr_exchangeable(
    cumulatives,
    [items('TASKS',all),
     items('MACHINES',all),
     vals(
         ['TASKS'^machine,'MACHINES'^id],
         int,
         =\=,
         all,
         dontcare)]).

ctr_derived_collections(
    cumulatives,
    [col(TIME_POINTS-
         collection(idm-int,duration-dvar,point-dvar),
         [item(
              idm-'TASKS'^machine,
              duration-'TASKS'^duration,
              point-'TASKS'^origin),
          item(
              idm-'TASKS'^machine,
              duration-'TASKS'^duration,
              point-'TASKS'^end)])]).

ctr_graph(
    cumulatives,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin+tasks^duration=tasks^end],
    ['NARC'=size('TASKS')],
```

```
    []).

ctr_graph(
    cumulatives,
    ['TIME_POINTS','TASKS'],
    2,
    foreach(
        MACHINES,
        ['PRODUCT'>>collection(time_points,tasks)]),
    [time_points^idm='MACHINES'^id,
     time_points^idm=tasks^machine,
     time_points^duration>0,
     tasks^origin=<time_points^point,
     time_points^point<tasks^end],
    [],
    ['ACYCLIC','BIPARTITE','NO_LOOP'],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^height)])]],
    [sum_ctr(variables,'CTR','MACHINES'^capacity)]).

ctr_eval(cumulatives,[builtin(cumulatives_b)]).

ctr_contractible(
    cumulatives,
    [in_list('RELOP',[=<]),minval('TASKS'^height)>=0],
    TASKS,
    any).

ctr_application(cumulatives,[1]).

cumulatives_b(TASKS,MACHINES,=<) :-
        !,
        collection(TASKS,[dvar,dvar,dvar_gteq(0),dvar,dvar]),
        get_attr1(TASKS,VMACHINES),
        get_attr2(TASKS,ORIGINS),
        get_attr3(TASKS,DURATIONS),
        get_attr4(TASKS,ENDS),
        get_attr5(TASKS,HEIGHTS),
        collection(MACHINES,[int,int]),
        get_attr1(MACHINES,IDS),
        get_attr2(MACHINES,CAPACITIES),
        all_different(IDS),
        cumulatives1(
```

```
            VMACHINES,
            ORIGINS,
            DURATIONS,
            ENDS,
            HEIGHTS,
            Tasks),
        cumulatives2(IDS,CAPACITIES,Machines),
        cumulatives(Tasks,Machines,[bound(upper)]).

cumulatives_b(TASKS,MACHINES,>=) :-
        collection(TASKS,[dvar,dvar,dvar_gteq(0),dvar,dvar]),
        get_attr1(TASKS,VMACHINES),
        get_attr2(TASKS,ORIGINS),
        get_attr3(TASKS,DURATIONS),
        get_attr4(TASKS,ENDS),
        get_attr5(TASKS,HEIGHTS),
        collection(MACHINES,[int,int]),
        get_attr1(MACHINES,IDS),
        get_attr2(MACHINES,CAPACITIES),
        all_different(IDS),
        cumulatives1(
            VMACHINES,
            ORIGINS,
            DURATIONS,
            ENDS,
            HEIGHTS,
            Tasks),
        cumulatives2(IDS,CAPACITIES,Machines),
        cumulatives(Tasks,Machines,[bound(lower)]).

cumulatives1([],[],[],[],[],[]).

cumulatives1(
    [M|RM],
    [O|RO],
    [D|RD],
    [E|RE],
    [H|RH],
    [task(O,D,E,H,M)|R]) :-
        cumulatives1(RM,RO,RD,RE,RH,R).

cumulatives2([],[],[]).

cumulatives2([I|RI],[C|RC],[machine(I,C)|R]) :-
        cumulatives2(RI,RC,R).
```

B.104 cutset

◇ META-DATA:

```
ctr_date(cutset,['20030820','20040530','20060807']).

ctr_origin(cutset,'\\cite{FagesLal03}',[]).

ctr_arguments(
    cutset,
    ['SIZE_CUTSET'-dvar,
     'NODES'-collection(index-int,succ-sint,bool-dvar)]).

ctr_restrictions(
    cutset,
    ['SIZE_CUTSET'>=0,
     'SIZE_CUTSET'=<size('NODES'),
     required('NODES',[index,succ,bool]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^bool>=0,
     'NODES'^bool=<1]).

ctr_example(
    cutset,
    cutset(
        1,
        [[index-1,succ-{2,3,4},bool-1],
         [index-2,succ-{3},bool-1],
         [index-3,succ-{4},bool-1],
         [index-4,succ-{1},bool-0]])).

ctr_typical(
    cutset,
    ['SIZE_CUTSET'>0,
     'SIZE_CUTSET'=<size('NODES'),
     size('NODES')>1]).

ctr_exchangeable(cutset,[items('NODES',all)]).

ctr_graph(
    cutset,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
```

```
[nodes2^index in_set nodes1^succ,
 nodes1^bool=1,
 nodes2^bool=1],
['MAX_NSCC'=<1,'NVERTEX'=size('NODES')-'SIZE_CUTSET'],
['ACYCLIC','NO_LOOP']).
```

B.105 cycle

◊ META-DATA:

```
ctr_date(cycle,['20000128','20030820','20060807','20111223']).

ctr_origin(cycle,'\\cite{BeldiceanuContejean94}',[]).

ctr_arguments(
    cycle,
    ['NCYCLE'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    cycle,
    ['NCYCLE'>=1,
     'NCYCLE'=<size('NODES'),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    cycle,
    [cycle(
         2,
         [[index-1,succ-2],
          [index-2,succ-1],
          [index-3,succ-5],
          [index-4,succ-3],
          [index-5,succ-4]]),
     cycle(
         1,
         [[index-1,succ-2],
          [index-2,succ-5],
          [index-3,succ-1],
          [index-4,succ-3],
          [index-5,succ-4]]),
     cycle(
         5,
         [[index-1,succ-1],
          [index-2,succ-2],
          [index-3,succ-3],
          [index-4,succ-4],
          [index-5,succ-5]])]).
```

```
ctr_typical(cycle,['NCYCLE'<size('NODES'),size('NODES')>2]).

ctr_exchangeable(
    cycle,
    [items('NODES',all),attrs_sync('NODES',[[index,succ]])]).

ctr_graph(
    cycle,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['NTREE'=0,'NCC'='NCYCLE'],
    ['ONE_SUCC']).

ctr_eval(cycle,[checker(cycle_c),reformulation(cycle_r)]).

ctr_functional_dependency(cycle,1,[2]).

ctr_cond_imply(
    cycle,
    balance_cycle,
    ['NCYCLE'=1],
    ['BALANCE'=0],
    [none,'NODES']).

ctr_cond_imply(cycle,permutation,[],[],[index_to_col('NODES')]).

ctr_application(cycle,[2]).

ctr_sol(cycle,2,0,2,2,[1-1,2-1]).

ctr_sol(cycle,3,0,3,6,[1-2,2-3,3-1]).

ctr_sol(cycle,4,0,4,24,[1-6,2-11,3-6,4-1]).

ctr_sol(cycle,5,0,5,120,[1-24,2-50,3-35,4-10,5-1]).

ctr_sol(cycle,6,0,6,720,[1-120,2-274,3-225,4-85,5-15,6-1]).

ctr_sol(
    cycle,
    7,
    0,
    7,
```

```
    5040,
    [1-720,2-1764,3-1624,4-735,5-175,6-21,7-1]).

ctr_sol(
    cycle,
    8,
    0,
    8,
    40320,
    [1-5040,2-13068,3-13132,4-6769,5-1960,6-322,7-28,8-1]).

ctr_sol(
    cycle,
    9,
    0,
    9,
    362880,
    [1-40320,
     2-109584,
     3-118124,
     4-67284,
     5-22449,
     6-4536,
     7-546,
     8-36,
     9-1]).

ctr_sol(
    cycle,
    10,
    0,
    10,
    3628800,
    [1-362880,
     2-1026576,
     3-1172700,
     4-723680,
     5-269325,
     6-63273,
     7-9450,
     8-870,
     9-45,
     10-1]).

cycle_c(NCYCLE,NODES) :-
        length(NODES,N),
```

```
        check_type(dvar(1,N),NCYCLE),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,IND),
        sort(IND,Js),
        length(Js,N),
        get_attr12(NODES,IND_SUCC),
        keysort(IND_SUCC,SIND_SUCC),
        remove_key_from_collection(SIND_SUCC,SUCCS),
        length(Term,N),
        list_to_tree(Term,Tree),
        (for(J,1,N),
         foreach(X,SUCCS),
         foreach(Free,Term),foreach(J,Js),param(Tree)do
         get_label(X,Tree,Free)),
        sort(SUCCS,Js),
        sort(Term,Cs),
        length(Cs,NCYCLE).

cycle_r(NCYCLE,NODES) :-
        length(NODES,N),
        check_type(dvar(1,N),NCYCLE),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,IND),
        sort(IND,SIND),
        length(SIND,N),
        get_attr12(NODES,IND_SUCC),
        keysort(IND_SUCC,SIND_SUCC),
        remove_key_from_collection(SIND_SUCC,Succ),
        all_distinct(Succ),
        (for(I,1,N),foreach(Min,Mins),param(Succ,N)do
         length([I|Ss],N),
         minimum(Min,[I|Ss]),
         (foreach(S2,Ss),fromto(I,S1,S2,_88223),param(Succ)do
          element(S1,Succ,S2))),
        nvalue(NCYCLE,Mins).
```

B.106 cycle_card_on_path

◇ META-DATA:

```
ctr_date(
    cycle_card_on_path,
    ['20000128','20030820','20040530','20060807']).

ctr_origin(cycle_card_on_path,'\\index{CHIP|indexuse}CHIP',[]).

ctr_arguments(
    cycle_card_on_path,
    ['NCYCLE'-dvar,
     'NODES'-collection(index-int,succ-dvar,colour-dvar),
     'ATLEAST'-int,
     'ATMOST'-int,
     'PATH_LEN'-int,
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    cycle_card_on_path,
    ['NCYCLE'>=1,
     'NCYCLE'=<size('NODES'),
     required('NODES',[index,succ,colour]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES'),
     'ATLEAST'>=0,
     'ATLEAST'=<'PATH_LEN',
     'ATMOST'>='ATLEAST',
     'PATH_LEN'>=0,
     size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    cycle_card_on_path,
    cycle_card_on_path(
        2,
        [[index-1,succ-7,colour-2],
         [index-2,succ-4,colour-3],
         [index-3,succ-8,colour-2],
         [index-4,succ-9,colour-1],
         [index-5,succ-1,colour-2],
```

```
          [index-6,succ-2,colour-1],
          [index-7,succ-5,colour-1],
          [index-8,succ-6,colour-1],
          [index-9,succ-3,colour-1]],
         1,
         2,
         3,
         [[val-1]]])).

ctr_typical(
    cycle_card_on_path,
    [size('NODES')>2,
     'NCYCLE'<size('NODES'),
     'ATLEAST'<'PATH_LEN',
     'ATMOST'>0,
     'PATH_LEN'>1,
     size('NODES')>size('VALUES'),
     'ATLEAST'>0#\/'ATMOST'<'PATH_LEN']).

ctr_exchangeable(
    cycle_card_on_path,
    [items('NODES',all),
     vals(
         ['NODES'^colour],
         comp('VALUES'^val),
         =,
         dontcare,
         dontcare),
     vals(['ATLEAST'],int(>=(0)),>,dontcare,dontcare),
     vals(['ATMOST'],int,<,dontcare,dontcare),
     items('VALUES',all)]).

ctr_graph(
    cycle_card_on_path,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['NTREE'=0,'NCC'='NCYCLE'],
    ['ONE_SUCC'],
    ['PATH_LENGTH'('PATH_LEN')>>
     [variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'NODES'^colour)])]],
    [among_low_up('ATLEAST','ATMOST',variables,'VALUES')]).
```

```
ctr_application(cycle_card_on_path,[2]).
```

B.107 cycle_or_accessibility

◊ **META-DATA:**

```
ctr_date(
    cycle_or_accessibility,
    ['20000128','20030820','20060807']).

ctr_origin(
    cycle_or_accessibility,
    Inspired by \cite{LabbeLaporteRodriguezMartin98}.,
    []).

ctr_arguments(
    cycle_or_accessibility,
    ['MAXDIST'-int,
     'NCYCLE'-dvar,
     'NODES'-collection(index-int,succ-dvar,x-int,y-int)]).

ctr_restrictions(
    cycle_or_accessibility,
    ['MAXDIST'>=0,
     'NCYCLE'>=1,
     'NCYCLE'=<size('NODES'),
     required('NODES',[index,succ,x,y]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=0,
     'NODES'^succ=<size('NODES'),
     'NODES'^x>=0,
     'NODES'^y>=0]).

ctr_example(
    cycle_or_accessibility,
    cycle_or_accessibility(
        3,
        2,
        [[index-1,succ-6,x-4,y-5],
         [index-2,succ-0,x-9,y-1],
         [index-3,succ-0,x-2,y-4],
         [index-4,succ-1,x-2,y-6],
         [index-5,succ-5,x-7,y-2],
         [index-6,succ-4,x-4,y-7],
         [index-7,succ-0,x-6,y-4]])).
```

```
ctr_typical(
    cycle_or_accessibility,
    ['MAXDIST'>0,'NCYCLE'<size('NODES'),size('NODES')>2]).

ctr_exchangeable(
    cycle_or_accessibility,
    [items('NODES',all),
     attrs_sync('NODES',[[index],[succ],[x,y]]),
     translate(['NODES'^x]),
     translate(['NODES'^y])]).

ctr_graph(
    cycle_or_accessibility,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['NTREE'=0,'NCC'='NCYCLE'],
    []).

ctr_graph(
    cycle_or_accessibility,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index#\/
     nodes1^succ=0#/\nodes2^succ=\=0#/\
     abs(nodes1^x-nodes2^x)+abs(nodes1^y-nodes2^y)=<'MAXDIST'],
    ['NVERTEX'=size('NODES')],
    [],
    [PRED>>
     [variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'NODES'^succ)]),
      destination]],
    [nvalues_except_0(variables,=,1)]).

ctr_functional_dependency(cycle_or_accessibility,2,[3]).

ctr_application(cycle_or_accessibility,[3]).
```

B.108 cycle_resource

◇ **META-DATA:**

```
ctr_date(cycle_resource,['20030820','20040530','20060807']).

ctr_origin(cycle_resource,'\\index{CHIP|indexuse}CHIP',[]).

ctr_arguments(
    cycle_resource,
    [RESOURCE-
     collection(id-int,first_task-dvar,nb_task-dvar),
     'TASK'-collection(id-int,next_task-dvar,resource-dvar)]).

ctr_restrictions(
    cycle_resource,
    [required('RESOURCE',[id,first_task,nb_task]),
     'RESOURCE'^id>=1,
     'RESOURCE'^id=<size('RESOURCE'),
     distinct('RESOURCE',id),
     'RESOURCE'^first_task>=1,
     'RESOURCE'^first_task=<size('RESOURCE')+size('TASK'),
     'RESOURCE'^nb_task>=0,
     'RESOURCE'^nb_task=<size('TASK'),
     required('TASK',[id,next_task,resource]),
     'TASK'^id>size('RESOURCE'),
     'TASK'^id=<size('RESOURCE')+size('TASK'),
     distinct('TASK',id),
     'TASK'^next_task>=1,
     'TASK'^next_task=<size('RESOURCE')+size('TASK'),
     'TASK'^resource>=1,
     'TASK'^resource=<size('RESOURCE')]).

ctr_example(
    cycle_resource,
    cycle_resource(
        [[id-1,first_task-5,nb_task-3],
         [id-2,first_task-2,nb_task-0],
         [id-3,first_task-8,nb_task-2]],
        [[id-4,next_task-7,resource-1],
         [id-5,next_task-4,resource-1],
         [id-6,next_task-3,resource-3],
         [id-7,next_task-1,resource-1],
         [id-8,next_task-6,resource-3]])).

ctr_typical(
```

```
    cycle_resource,
    [size('RESOURCE')>1,
     size('TASK')>1,
     size('TASK')>size('RESOURCE')]).

ctr_exchangeable(
    cycle_resource,
    [items('RESOURCE',all),
     items('TASK',all),
     vals(['RESOURCE'^id,'TASK'^resource],int,=\=,all,in)]).

ctr_derived_collections(
    cycle_resource,
    [col(RESOURCE_TASK-
         collection(index-int,succ-dvar,name-dvar),
         [item(
              index-'RESOURCE'^id,
              succ-'RESOURCE'^first_task,
              name-'RESOURCE'^id),
          item(
              index-'TASK'^id,
              succ-'TASK'^next_task,
              name-'TASK'^resource)])]).

ctr_graph(
    cycle_resource,
    ['RESOURCE_TASK'],
    2,
    ['CLIQUE'>>collection(resource_task1,resource_task2)],
    [resource_task1^succ=resource_task2^index,
     resource_task1^name=resource_task2^name],
    ['NTREE'=0,
     'NCC'=size('RESOURCE'),
     'NVERTEX'=size('RESOURCE')+size('TASK')],
    ['ONE_SUCC']).

ctr_graph(
    cycle_resource,
    ['RESOURCE_TASK'],
    2,
    foreach(
        RESOURCE,
        ['CLIQUE'>>collection(resource_task1,resource_task2)]),
    [resource_task1^succ=resource_task2^index,
     resource_task1^name=resource_task2^name,
     resource_task1^name='RESOURCE'^id],
```

```
    ['NVERTEX'='RESOURCE'^nb_task+1],
    []).

ctr_application(cycle_resource,[2]).
```

B.109 cyclic_change

♢ META-DATA:

```
ctr_date(
    cyclic_change,
    ['20000128','20030820','20040530','20060807']).

ctr_origin(cyclic_change,'Derived from %c.',[change]).

ctr_arguments(
    cyclic_change,
    ['NCHANGE'-dvar,
     'CYCLE_LENGTH'-int,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    cyclic_change,
    ['NCHANGE'>=0,
     'NCHANGE'<size('VARIABLES'),
     'CYCLE_LENGTH'>0,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var<'CYCLE_LENGTH',
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    cyclic_change,
    cyclic_change(
        2,
        4,
        [[var-3],[var-0],[var-2],[var-3],[var-1]],
        =\=)).

ctr_typical(
    cyclic_change,
    ['NCHANGE'>0,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=\=])]).

ctr_exchangeable(cyclic_change,[items('VARIABLES',shift)]).

ctr_graph(
    cyclic_change,
```

```
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'((variables1^var+1)mod 'CYCLE_LENGTH',
         variables2^var)],
    ['NARC'='NCHANGE'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(cyclic_change,[automaton(cyclic_change_a)]).

ctr_pure_functional_dependency(cyclic_change,[]).

ctr_functional_dependency(cyclic_change,1,[2,3,4]).

cyclic_change_a(FLAG,NCHANGE,CYCLE_LENGTH,VARIABLES,CTR) :-
        integer(CYCLE_LENGTH),
        CYCLE_LENGTH>0,
        CYCLE_LENGTH_1 is CYCLE_LENGTH-1,
        collection(VARIABLES,[dvar(0,CYCLE_LENGTH_1)]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        cyclic_change_signature(
            VARIABLES,
            SIGNATURE,
            CYCLE_LENGTH,
            CTR),
        automaton(
            SIGNATURE,
            _44114,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NCHANGE#<=>FLAG.

cyclic_change_signature([],[],_41520,_41521).

cyclic_change_signature([_41525],[],_41523,_41524) :-
        !.

cyclic_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
```

```
    [S|Ss],
    CYCLE_LENGTH,
    =) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#=VAR2#<=>S,
        cyclic_change_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            =).

cyclic_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    =\=) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#\=VAR2#<=>S,
        cyclic_change_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            =\=).

cyclic_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    <) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#<VAR2#<=>S,
        cyclic_change_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            <).

cyclic_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    >=) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#>=VAR2#<=>S,
        cyclic_change_signature(
            [[var-VAR2]|VARs],
```

```
            Ss,
            CYCLE_LENGTH,
            >=).

cyclic_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    >) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#>VAR2#<=>S,
        cyclic_change_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            >).

cyclic_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    =<) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#=<VAR2#<=>S,
        cyclic_change_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            =<).
```

B.110 cyclic_change_joker

◇ Meta-Data:

```
ctr_date(
    cyclic_change_joker,
    ['20000128','20030820','20040530','20060807']).

ctr_origin(
    cyclic_change_joker,
    Derived from %c.,
    [cyclic_change]).

ctr_arguments(
    cyclic_change_joker,
    ['NCHANGE'-dvar,
     'CYCLE_LENGTH'-int,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    cyclic_change_joker,
    ['NCHANGE'>=0,
     'NCHANGE'<size('VARIABLES'),
     'CYCLE_LENGTH'>0,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    cyclic_change_joker,
    cyclic_change_joker(
        2,
        4,
        [[var-3],
         [var-0],
         [var-2],
         [var-4],
         [var-4],
         [var-4],
         [var-3],
         [var-1],
         [var-4]],
        =\=)).

ctr_typical(
```

```
    cyclic_change_joker,
    ['NCHANGE'>0,
     'CYCLE_LENGTH'>1,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     maxval('VARIABLES'^var)>='CYCLE_LENGTH',
     in_list('CTR',[=\=])]).

ctr_typical_model(
    cyclic_change_joker,
    [atleast(2,'VARIABLES',0)]).

ctr_exchangeable(
    cyclic_change_joker,
    [items('VARIABLES',shift)]).

ctr_graph(
    cyclic_change_joker,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'((variables1^var+1)mod 'CYCLE_LENGTH',
          variables2^var),
     variables1^var<'CYCLE_LENGTH',
     variables2^var<'CYCLE_LENGTH'],
    ['NARC'='NCHANGE'],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    cyclic_change_joker,
    [automaton(cyclic_change_joker_a)]).

ctr_pure_functional_dependency(cyclic_change_joker,[]).

ctr_functional_dependency(cyclic_change_joker,1,[2,3,4]).

cyclic_change_joker_a(FLAG,NCHANGE,CYCLE_LENGTH,VARIABLES,CTR) :-
        integer(CYCLE_LENGTH),
        CYCLE_LENGTH>0,
        collection(VARIABLES,[dvar_gteq(0)]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        cyclic_change_joker_signature(
            VARIABLES,
```

```
            SIGNATURE,
            CYCLE_LENGTH,
            CTR),
        automaton(
            SIGNATURE,
            _47157,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=NCHANGE#<=>FLAG.

cyclic_change_joker_signature([],[],_44780,_44781).

cyclic_change_joker_signature([_44785],[],_44783,_44784) :-
        !.

cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    =) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#=VAR2#/\
        VAR1#<CYCLE_LENGTH#/\
        VAR2#<CYCLE_LENGTH#<=>
        S,
        cyclic_change_joker_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            =).

cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    =\=) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#\=VAR2#/\
        VAR1#<CYCLE_LENGTH#/\
        VAR2#<CYCLE_LENGTH#<=>
        S,
        cyclic_change_joker_signature(
```

```
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            =\=).

cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    <) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#<VAR2#/\
        VAR1#<CYCLE_LENGTH#/\
        VAR2#<CYCLE_LENGTH#<=>
        S,
        cyclic_change_joker_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            <).

cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    >=) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#>=VAR2#/\
        VAR1#<CYCLE_LENGTH#/\
        VAR2#<CYCLE_LENGTH#<=>
        S,
        cyclic_change_joker_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            >=).

cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    >) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#>VAR2#/\
        VAR1#<CYCLE_LENGTH#/\
        VAR2#<CYCLE_LENGTH#<=>
```

```
        S,
        cyclic_change_joker_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            >).

cyclic_change_joker_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    CYCLE_LENGTH,
    =<) :-
        !,
        (VAR1+1)mod CYCLE_LENGTH#=<VAR2#/\
        VAR1#<CYCLE_LENGTH#/\
        VAR2#<CYCLE_LENGTH#<=>
        S,
        cyclic_change_joker_signature(
            [[var-VAR2]|VARs],
            Ss,
            CYCLE_LENGTH,
            =<).
```

B.111 dag

◇ **META-DATA:**

```
ctr_date(dag,['20061001']).

ctr_origin(dag,'\\cite{Dooms06}',[]).

ctr_arguments(dag,['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    dag,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    dag,
    dag([[index-1,succ-{2,4}],
         [index-2,succ-{3,4}],
         [index-3,succ-{}],
         [index-4,succ-{}],
         [index-5,succ-{6}],
         [index-6,succ-{}]])).

ctr_typical(dag,[size('NODES')>2]).

ctr_exchangeable(dag,[items('NODES',all)]).

ctr_graph(
    dag,
    ['NODES'],
    1,
    ['SELF'>>collection(nodes)],
    [nodes^key in_set nodes^succ],
    ['NARC'=0],
    []).

ctr_graph(
    dag,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
```

```
    [nodes2^index in_set nodes1^succ],
    ['MAX_NSCC'=<1],
    []).

ctr_application(dag,[1]).
```

B.112 decreasing

◊ **META-DATA:**

```
ctr_date(decreasing,['20040814','20060808']).

ctr_origin(decreasing,'Inspired by %c.',[increasing]).

ctr_arguments(decreasing,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(decreasing,[required('VARIABLES',var)]).

ctr_example(
    decreasing,
    decreasing([[var-8],[var-4],[var-1],[var-1]])).

ctr_typical(
    decreasing,
    [size('VARIABLES')>2,range('VARIABLES'^var)>1]).

ctr_typical_model(decreasing,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(decreasing,[translate(['VARIABLES'^var])]).

ctr_graph(
    decreasing,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var>=variables2^var],
    ['NARC'=size('VARIABLES')-1],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    decreasing,
    [checker(decreasing_c),automaton(decreasing_a)]).

ctr_contractible(decreasing,[],'VARIABLES',any).

ctr_sol(decreasing,2,0,2,6,-).

ctr_sol(decreasing,3,0,3,20,-).

ctr_sol(decreasing,4,0,4,70,-).

ctr_sol(decreasing,5,0,5,252,-).
```

```
ctr_sol(decreasing,6,0,6,924,-).

ctr_sol(decreasing,7,0,7,3432,-).

ctr_sol(decreasing,8,0,8,12870,-).

decreasing_c([[var-X],[var-Y]|_47816]) :-
        X<Y,
        !,
        fail.

decreasing_c([]) :-
        !.

decreasing_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        decreasing_c1(VARS).

decreasing_c1([X,Y|R]) :-
        !,
        X>=Y,
        decreasing_c1([Y|R]).

decreasing_c1([_47806]) :-
        !.

decreasing_c1([]).

decreasing_a(1,[]) :-
        !.

decreasing_a(0,[]) :-
        !,
        fail.

decreasing_a(FLAG,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        decreasing_signature(VARIABLES,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _48949,
            SIGNATURE,
            [source(s),sink(s)],
```

```
            [arc(s,1,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

decreasing_signature([_47807],[]) :-
        !.

decreasing_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
        S in 0..1,
        VAR1#>=VAR2#<=>S,
        decreasing_signature([[var-VAR2]|VARs],Ss).
```

B.113 decreasing_peak

◇ META-DATA:

```
ctr_date(decreasing_peak,['20130209']).

ctr_origin(
    decreasing_peak,
    Derived from %c and %c.,
    [peak,decreasing]).

ctr_arguments(
    decreasing_peak,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    decreasing_peak,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    decreasing_peak,
    decreasing_peak(
        [[var-1],
         [var-7],
         [var-7],
         [var-4],
         [var-3],
         [var-7],
         [var-2],
         [var-2],
         [var-5],
         [var-4]])).

ctr_typical(
    decreasing_peak,
    [size('VARIABLES')>=7,
     range('VARIABLES'^var)>1,
     peak('VARIABLES'^var)>=3]).

ctr_typical_model(decreasing_peak,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    decreasing_peak,
    [translate(['VARIABLES'^var])]).

ctr_eval(
```

```
    decreasing_peak,
    [checker(decreasing_peak_c),
     automaton(decreasing_peak_a),
     automaton_with_signature(decreasing_peak_a_s)]).

ctr_contractible(decreasing_peak,[],'VARIABLES',prefix).

ctr_contractible(decreasing_peak,[],'VARIABLES',suffix).

ctr_cond_imply(
    decreasing_peak,
    not_all_equal,
    [peak('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(decreasing_peak,2,0,2,9,-).

ctr_sol(decreasing_peak,3,0,3,64,-).

ctr_sol(decreasing_peak,4,0,4,625,-).

ctr_sol(decreasing_peak,5,0,5,7553,-).

ctr_sol(decreasing_peak,6,0,6,105798,-).

ctr_sol(decreasing_peak,7,0,7,1666878,-).

ctr_sol(decreasing_peak,8,0,8,29090469,-).

decreasing_peak_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        L>0,
        get_attr1(VARIABLES,VARS),
        decreasing_peak_c(VARS,s,0).

decreasing_peak_c([V1,V2|R],s,A) :-
        V1>=V2,
        !,
        decreasing_peak_c([V2|R],s,A).

decreasing_peak_c([_32156,V2|R],s,A) :-
        !,
        decreasing_peak_c([V2|R],u,A).
```

```
decreasing_peak_c([V1,V2|R],u,A) :-
        V1=<V2,
        !,
        decreasing_peak_c([V2|R],u,A).

decreasing_peak_c([V1,V2|R],u,_32155) :-
        !,
        decreasing_peak_c([V2|R],v,V1).

decreasing_peak_c([V1,V2|R],v,A) :-
        V1>=V2,
        !,
        decreasing_peak_c([V2|R],v,A).

decreasing_peak_c([_32156,V2|R],v,A) :-
        !,
        decreasing_peak_c([V2|R],w,A).

decreasing_peak_c([V1,V2|R],w,A) :-
        V1=<V2,
        !,
        decreasing_peak_c([V2|R],w,A).

decreasing_peak_c([V1,V2|R],w,A) :-
        !,
        A>=V1,
        decreasing_peak_c([V2|R],v,V1).

decreasing_peak_c([_32156],_32154,_32155) :-
        !.

decreasing_peak_c([],_32151,_32152).

ctr_automaton_signature(
    decreasing_peak,
    decreasing_peak_a,
    pair_signature(1,signature)).

decreasing_peak_a(FLAG,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        decreasing_peak_a_s(FLAG,VARIABLES,SIGNATURE).

decreasing_peak_a_s(FLAG,VARIABLES,SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        L>=0,
```

```
pair_first_signature(VARIABLES,VARS),
automaton(
    VARS,
    Vi,
    SIGNATURE,
    [source(s),sink(s),sink(u),sink(v),sink(w)],
    [arc(s,2,s),
     arc(s,1,s),
     arc(s,0,u),
     arc(u,2,v,[Vi,F]),
     arc(u,1,u),
     arc(u,0,u),
     arc(v,2,v),
     arc(v,1,v),
     arc(v,0,w),
     arc(w,2,v,(A#>=Vi->[Vi,F])),
     arc(w,2,v,(A#<Vi->[A,0])),
     arc(w,1,w),
     arc(w,0,w)],
    [A,F],
    [0,1],
    [_A,FLAG]).
```

B.114 decreasing_valley

◇ META-DATA:

```
ctr_date(decreasing_valley,['20130210']).

ctr_origin(
    decreasing_valley,
    Derived from %c and %c.,
    [valley,decreasing]).

ctr_arguments(
    decreasing_valley,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    decreasing_valley,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    decreasing_valley,
    decreasing_valley(
        [[var-1],
         [var-7],
         [var-6],
         [var-8],
         [var-3],
         [var-7],
         [var-3],
         [var-3],
         [var-5],
         [var-4]])).

ctr_typical(
    decreasing_valley,
    [size('VARIABLES')>=7,
     range('VARIABLES'^var)>1,
     valley('VARIABLES'^var)>=3]).

ctr_typical_model(decreasing_valley,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    decreasing_valley,
    [translate(['VARIABLES'^var])]).

ctr_eval(
```

```
    decreasing_valley,
    [checker(decreasing_valley_c),
     automaton(decreasing_valley_a),
     automaton_with_signature(decreasing_valley_a_s)]).

ctr_contractible(decreasing_valley,[],'VARIABLES',prefix).

ctr_contractible(decreasing_valley,[],'VARIABLES',suffix).

ctr_cond_imply(
    decreasing_valley,
    not_all_equal,
    [valley('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(decreasing_valley,2,0,2,9,-).

ctr_sol(decreasing_valley,3,0,3,64,-).

ctr_sol(decreasing_valley,4,0,4,625,-).

ctr_sol(decreasing_valley,5,0,5,7553,-).

ctr_sol(decreasing_valley,6,0,6,105798,-).

ctr_sol(decreasing_valley,7,0,7,1666878,-).

ctr_sol(decreasing_valley,8,0,8,29090469,-).

decreasing_valley_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        L>0,
        get_attr1(VARIABLES,VARS),
        decreasing_valley_c(VARS,s,0).

decreasing_valley_c([V1,V2|R],s,A) :-
        V1=<V2,
        !,
        decreasing_valley_c([V2|R],s,A).

decreasing_valley_c([_32192,V2|R],s,A) :-
        !,
        decreasing_valley_c([V2|R],u,A).
```

```
decreasing_valley_c([V1,V2|R],u,A) :-
        V1>=V2,
        !,
        decreasing_valley_c([V2|R],u,A).

decreasing_valley_c([V1,V2|R],u,_32191) :-
        !,
        decreasing_valley_c([V2|R],v,V1).

decreasing_valley_c([V1,V2|R],v,A) :-
        V1=<V2,
        !,
        decreasing_valley_c([V2|R],v,A).

decreasing_valley_c([_32192,V2|R],v,A) :-
        !,
        decreasing_valley_c([V2|R],w,A).

decreasing_valley_c([V1,V2|R],w,A) :-
        V1>=V2,
        !,
        decreasing_valley_c([V2|R],w,A).

decreasing_valley_c([V1,V2|R],w,A) :-
        !,
        A>=V1,
        decreasing_valley_c([V2|R],v,V1).

decreasing_valley_c([_32192],_32190,_32191) :-
        !.

decreasing_valley_c([],_32187,_32188).

ctr_automaton_signature(
    decreasing_valley,
    decreasing_valley_a,
    pair_signature(1,signature)).

decreasing_valley_a(FLAG,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        decreasing_valley_a_s(FLAG,VARIABLES,SIGNATURE).

decreasing_valley_a_s(FLAG,VARIABLES,SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        L>=0,
```

```
pair_first_signature(VARIABLES,VARS),
automaton(
    VARS,
    Vi,
    SIGNATURE,
    [source(s),sink(s),sink(u),sink(v),sink(w)],
    [arc(s,0,s),
     arc(s,1,s),
     arc(s,2,u),
     arc(u,0,v,[Vi,F]),
     arc(u,1,u),
     arc(u,2,u),
     arc(v,0,v),
     arc(v,1,v),
     arc(v,2,w),
     arc(w,0,v,(A#>=Vi->[Vi,F])),
     arc(w,0,v,(A#<Vi->[A,0])),
     arc(w,1,w),
     arc(w,2,w)],
    [A,F],
    [0,1],
    [_A,FLAG]).
```

B.115 deepest_valley

◇ **META-DATA:**

```
ctr_date(deepest_valley,['20040530']).

ctr_origin(deepest_valley,'Derived from %c.',[valley]).

ctr_arguments(
    deepest_valley,
    ['DEPTH'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(deepest_valley,[required('VARIABLES',var)]).

ctr_example(
    deepest_valley,
    [deepest_valley(
         2,
         [[var-5],
          [var-3],
          [var-4],
          [var-8],
          [var-8],
          [var-2],
          [var-7],
          [var-1]]),
     deepest_valley(
         7,
         [[var-1],
          [var-3],
          [var-4],
          [var-8],
          [var-8],
          [var-8],
          [var-7],
          [var-8]])]).

ctr_typical(
    deepest_valley,
    [size('VARIABLES')>2,
     range('VARIABLES'^var)>2,
     valley('VARIABLES'^var)>0]).

ctr_typical_model(deepest_valley,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(deepest_valley,[items('VARIABLES',reverse)]).
```

```
ctr_eval(
    deepest_valley,
    [checker(deepest_valley_c),
     automaton(deepest_valley_a),
     automaton_with_signature(deepest_valley_a_s)]).

ctr_pure_functional_dependency(deepest_valley,[]).

ctr_functional_dependency(deepest_valley,1,[2]).

ctr_sol(deepest_valley,2,0,2,9,[1000000-9]).

ctr_sol(deepest_valley,3,0,3,64,[0-9,1-4,2-1,1000000-50]).

ctr_sol(
    deepest_valley,
    4,
    0,
    4,
    625,
    [0-176,1-99,2-44,3-11,1000000-295]).

ctr_sol(
    deepest_valley,
    5,
    0,
    5,
    7776,
    [0-2900,1-1712,2-900,3-380,4-92,1000000-1792]).

ctr_sol(
    deepest_valley,
    6,
    0,
    6,
    117649,
    [0-50472,
     1-29125,
     2-15680,
     3-7587,
     4-3000,
     5-697,
     1000000-11088]).

ctr_sol(
```

```
    deepest_valley,
    7,
    0,
    7,
    2097152,
    [0-976227,
     1-540576,
     2-283250,
     3-138544,
     4-61389,
     5-22632,
     6-5036,
     1000000-69498]).

ctr_sol(
    deepest_valley,
    8,
    0,
    8,
    43046721,
    [0-21133632,
     1-11233250,
     2-5665896,
     3-2693425,
     4-1195056,
     5-484020,
     6-166208,
     7-35443,
     1000000-439791]).

deepest_valley_c(DEPTH,VARIABLES) :-
        check_type(dvar,DEPTH),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        MAXINT=1000000,
        deepest_valley_c(VARS,s,MAXINT,DEPTH).

deepest_valley_c([V1,V2|R],s,C,DEPTH) :-
        V1=<V2,
        !,
        deepest_valley_c([V2|R],s,C,DEPTH).

deepest_valley_c([_V1,V2|R],s,C,DEPTH) :-
        !,
        deepest_valley_c([V2|R],u,C,DEPTH).
```

```
deepest_valley_c([V1,V2|R],u,C,DEPTH) :-
        V1>=V2,
        !,
        deepest_valley_c([V2|R],u,C,DEPTH).

deepest_valley_c([V1,V2|R],u,C,DEPTH) :-
        !,
        C1 is min(C,V1),
        deepest_valley_c([V2|R],s,C1,DEPTH).

deepest_valley_c([_48254],_48251,DEPTH,DEPTH) :-
        !.

deepest_valley_c([],_48248,DEPTH,DEPTH).

deepest_valley_counters_check([V1,V2|R],s,C,[C|S]) :-
        V1=<V2,
        !,
        deepest_valley_counters_check([V2|R],s,C,S).

deepest_valley_counters_check([_V1,V2|R],s,C,[C|S]) :-
        !,
        deepest_valley_counters_check([V2|R],u,C,S).

deepest_valley_counters_check([V1,V2|R],u,C,[C|S]) :-
        V1>=V2,
        !,
        deepest_valley_counters_check([V2|R],u,C,S).

deepest_valley_counters_check([V1,V2|R],u,C,[C1|S]) :-
        !,
        C1 is min(C,V1),
        deepest_valley_counters_check([V2|R],s,C1,S).

deepest_valley_counters_check([V|R],init,C,[C|S]) :-
        !,
        deepest_valley_counters_check([V|R],s,C,S).

deepest_valley_counters_check([_48251],_48248,_48249,[]).

ctr_automaton_signature(
    deepest_valley,
    deepest_valley_a,
    pair_signature(2,signature)).

deepest_valley_a(FLAG,DEPTH,VARIABLES) :-
```

```
        pair_signature(VARIABLES,SIGNATURE),
        deepest_valley_a_s(FLAG,DEPTH,VARIABLES,SIGNATURE).

deepest_valley_a_s(FLAG,DEPTH,VARIABLES,SIGNATURE) :-
        check_type(dvar,DEPTH),
        collection(VARIABLES,[dvar]),
        MAXINT=1000000,
        pair_first_signature(VARIABLES,VARS),
        automaton(
            VARS,
            VAR1,
            SIGNATURE,
            [source(s),sink(s),sink(u)],
            [arc(s,0,s),
             arc(s,1,s),
             arc(s,2,u),
             arc(u,0,s,[min(C,VAR1)]),
             arc(u,1,u),
             arc(u,2,u)],
            [C],
            [MAXINT],
            [COUNT]),
        COUNT#=DEPTH#<=>FLAG.
```

B.116 derangement

◇ **META-DATA:**

```
ctr_date(
    derangement,
    ['20000128','20030820','20040530','20060808']).

ctr_origin(derangement,'Derived from %c.',[cycle]).

ctr_arguments(
    derangement,
    ['NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    derangement,
    [size('NODES')>1,
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    derangement,
    derangement(
        [[index-1,succ-2],
         [index-2,succ-1],
         [index-3,succ-5],
         [index-4,succ-3],
         [index-5,succ-4]])).

ctr_typical(derangement,[size('NODES')>2]).

ctr_exchangeable(
    derangement,
    [items('NODES',all),attrs_sync('NODES',[[index,succ]])]).

ctr_graph(
    derangement,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index,nodes1^succ=\=nodes1^index],
    ['NTREE'=0],
```

```
    ['ONE_SUCC']).

ctr_eval(
    derangement,
    [checker(derangement_c),reformulation(derangement_r)]).

ctr_cond_imply(derangement,permutation,[],[],index_to_col).

ctr_sol(derangement,2,0,2,1,-).

ctr_sol(derangement,3,0,3,2,-).

ctr_sol(derangement,4,0,4,9,-).

ctr_sol(derangement,5,0,5,44,-).

ctr_sol(derangement,6,0,6,265,-).

ctr_sol(derangement,7,0,7,1854,-).

ctr_sol(derangement,8,0,8,14833,-).

ctr_sol(derangement,9,0,9,133496,-).

ctr_sol(derangement,10,0,10,1334961,-).

derangement_r(NODES) :-
        length(NODES,N),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        all_different(INDEXES),
        derangement1(SUCCS,INDEXES),
        all_different(SUCCS).

derangement_c([[_50427,succ-V],[_50438,succ-V]|_50437]) :-
        !,
        fail.

derangement_c(NODES) :-
        length(NODES,N),
        collection(NODES,[int(1,N),int(1,N)]),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        derangement1_fix(SUCCS,INDEXES),
        sort(SUCCS,SSUCCS),
```

```
length(SSUCCS,N),
sort(INDEXES,SINDEXES),
length(SINDEXES,N).
```

B.117 differ_from_at_least_k_pos

◊ META-DATA:

```
ctr_date(
    differ_from_at_least_k_pos,
    ['20030820','20040530','20060808']).

ctr_origin(
    differ_from_at_least_k_pos,
    Inspired by \cite{Frutos97}.,
    []).

ctr_types(
    differ_from_at_least_k_pos,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    differ_from_at_least_k_pos,
    ['K'-int,'VECTOR1'-'VECTOR','VECTOR2'-'VECTOR']).

ctr_restrictions(
    differ_from_at_least_k_pos,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     'K'>=0,
     'K'=<size('VECTOR1'),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    differ_from_at_least_k_pos,
    differ_from_at_least_k_pos(
        2,
        [[var-2],[var-5],[var-2],[var-0]],
        [[var-3],[var-6],[var-2],[var-1]])).

ctr_typical(
    differ_from_at_least_k_pos,
    ['K'>0,'K'<size('VECTOR1'),size('VECTOR1')>1]).

ctr_exchangeable(
    differ_from_at_least_k_pos,
    [args([['K'],['VECTOR1','VECTOR2']]),
     vals(['K'],int(>=(0)),>,dontcare,dontcare),
     items_sync('VECTOR1','VECTOR2',all)]).
```

```
ctr_graph(
    differ_from_at_least_k_pos,
    ['VECTOR1','VECTOR2'],
    2,
    ['PRODUCT'(=)>>collection(vector1,vector2)],
    [vector1^var=\=vector2^var],
    ['NARC'>='K'],
    []).

ctr_eval(
    differ_from_at_least_k_pos,
    [reformulation(differ_from_at_least_k_pos_r),
     automaton(differ_from_at_least_k_pos_a),
     checker(differ_from_at_least_k_pos_c)]).

ctr_extensible(
    differ_from_at_least_k_pos,
    [],
    ['VARIABLES1','VARIABLES2'],
    any).

differ_from_at_least_k_pos_r(K,VECTOR1,VECTOR2) :-
        integer(K),
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,N1),
        length(VECTOR2,N2),
        K>=0,
        K=<N1,
        N1=N2,
        N1>=1,
        differ_from_k_pos(VECTOR1,VECTOR2,SumBool),
        call(K#=<SumBool).

differ_from_at_least_k_pos_a(FLAG,K,VECTOR1,VECTOR2) :-
        integer(K),
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,N1),
        length(VECTOR2,N2),
        K>=0,
        K=<N1,
        N1=N2,
        N1>=1,
        differ_from_at_least_k_pos_signature(
            VECTOR1,
```

```
            VECTOR2,
            SIGNATURE),
        automaton(
            SIGNATURE,
            _45093,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s,[C+1]),arc(s,1,s)],
            [C],
            [0],
            [COUNT]),
        COUNT#>=K#<=>FLAG.

differ_from_at_least_k_pos_signature([],[],[]).

differ_from_at_least_k_pos_signature(
    [[var-VAR1]|VARS1],
    [[var-VAR2]|VARS2],
    [S|Ss]) :-
        VAR1#=VAR2#<=>S,
        differ_from_at_least_k_pos_signature(VARS1,VARS2,Ss).

differ_from_at_least_k_pos_c(K,VECTOR1,VECTOR2) :-
        integer(K),
        collection(VECTOR1,[int]),
        collection(VECTOR2,[int]),
        length(VECTOR1,N),
        length(VECTOR2,N),
        N>=1,
        K>=0,
        (   K=0 ->
            true
        ;   K=<N,
            differ_from_at_least_k_pos_check(
                VECTOR1,
                VECTOR2,
                N,
                K)
        ).

differ_from_at_least_k_pos_check([],[],_42497,0) :-
        !.

differ_from_at_least_k_pos_check(
    [[_42504-U]|R],
    [[_42515-V]|S],
```

```
N,
K) :-
    (   U=V ->
        NewK is K
    ;   NewK is K-1
    ),
    (   NewK=<0 ->
        true
    ;   NewN is N-1,
        NewK=<NewN,
        differ_from_at_least_k_pos_check(R,S,NewN,NewK)
    ).
```

B.118 differ_from_at_most_k_pos

◊ META-DATA:

```
ctr_date(differ_from_at_most_k_pos,['20120227']).

ctr_origin(
    differ_from_at_most_k_pos,
    Inspired by %c.,
    [differ_from_at_least_k_pos]).

ctr_types(
    differ_from_at_most_k_pos,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    differ_from_at_most_k_pos,
    ['K'-int,'VECTOR1'-'VECTOR','VECTOR2'-'VECTOR']).

ctr_restrictions(
    differ_from_at_most_k_pos,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     'K'>=0,
     'K'=<size('VECTOR1'),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    differ_from_at_most_k_pos,
    differ_from_at_most_k_pos(
        3,
        [[var-2],[var-5],[var-2],[var-0]],
        [[var-3],[var-6],[var-2],[var-0]])).

ctr_typical(
    differ_from_at_most_k_pos,
    ['K'>0,'K'<size('VECTOR1'),size('VECTOR1')>1]).

ctr_exchangeable(
    differ_from_at_most_k_pos,
    [args([['K'],['VECTOR1','VECTOR2']]),
     vals(['K'],int(=<(size('VECTOR1'))),<,dontcare,dontcare),
     items_sync('VECTOR1','VECTOR2',all)]).

ctr_graph(
    differ_from_at_most_k_pos,
```

```
    ['VECTOR1','VECTOR2'],
    2,
    ['PRODUCT'(=)>>collection(vector1,vector2)],
    [vector1^var=\=vector2^var],
    ['NARC'=<'K'],
    []).

ctr_eval(
    differ_from_at_most_k_pos,
    [reformulation(differ_from_at_most_k_pos_r),
     checker(differ_from_at_most_k_pos_c)]).

ctr_contractible(
    differ_from_at_most_k_pos,
    [],
    ['VARIABLES1','VARIABLES2'],
    any).

differ_from_at_most_k_pos_r(K,VECTOR1,VECTOR2) :-
        integer(K),
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,N1),
        length(VECTOR2,N2),
        K>=0,
        K=<N1,
        N1=N2,
        N1>=1,
        differ_from_k_pos(VECTOR1,VECTOR2,SumBool),
        call(K#>=SumBool).

differ_from_at_most_k_pos_c(K,VECTOR1,VECTOR2) :-
        integer(K),
        collection(VECTOR1,[int]),
        collection(VECTOR2,[int]),
        length(VECTOR1,N),
        length(VECTOR2,N),
        N>=1,
        K>=0,
        K=<N,
        (   N=<K ->
            true
        ;   differ_from_at_most_k_pos_check(
                VECTOR1,
                VECTOR2,
                N,
```

```
                K)
        ).

differ_from_at_most_k_pos_check([],[],_38043,0) :-
        !.

differ_from_at_most_k_pos_check(
    [[_38050-U]|R],
    [[_38061-V]|S],
    N,
    K) :-
        (   U=V ->
            NewK is K
        ;   NewK is K-1,
            NewK>=0
        ),
        NewN is N-1,
        (   NewN=<NewK ->
            true
        ;   differ_from_at_most_k_pos_check(R,S,NewN,NewK)
        ).
```

B.119 differ_from_exactly_k_pos

◇ **META-DATA:**

```
ctr_date(differ_from_exactly_k_pos,['20120227']).

ctr_origin(
    differ_from_exactly_k_pos,
    Inspired by %c.,
    [differ_from_at_least_k_pos]).

ctr_types(
    differ_from_exactly_k_pos,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    differ_from_exactly_k_pos,
    ['K'-int,'VECTOR1'-'VECTOR','VECTOR2'-'VECTOR']).

ctr_restrictions(
    differ_from_exactly_k_pos,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     'K'>=0,
     'K'=<size('VECTOR1'),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    differ_from_exactly_k_pos,
    differ_from_exactly_k_pos(
        2,
        [[var-3],[var-0],[var-2],[var-0]],
        [[var-3],[var-6],[var-2],[var-1]])).

ctr_typical(
    differ_from_exactly_k_pos,
    ['K'>0,'K'=<size('VECTOR1'),size('VECTOR1')>1]).

ctr_exchangeable(
    differ_from_exactly_k_pos,
    [args([['K'],['VECTOR1','VECTOR2']]),
     items_sync('VECTOR1','VECTOR2',all)]).

ctr_graph(
    differ_from_exactly_k_pos,
    ['VECTOR1','VECTOR2'],
```

```
    2,
    ['PRODUCT'(=)>>collection(vector1,vector2)],
    [vector1^var=\=vector2^var],
    ['NARC'='K'],
    []).

ctr_eval(
    differ_from_exactly_k_pos,
    [reformulation(differ_from_exactly_k_pos_r),
     checker(differ_from_exactly_k_pos_c)]).

ctr_pure_functional_dependency(differ_from_exactly_k_pos,[]).

ctr_functional_dependency(differ_from_exactly_k_pos,1,[2]).

differ_from_exactly_k_pos_r(K,VECTOR1,VECTOR2) :-
        integer(K),
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,N1),
        length(VECTOR2,N2),
        K>=0,
        K=<N1,
        N1=N2,
        N1>=1,
        differ_from_k_pos(VECTOR1,VECTOR2,SumBool),
        call(K#=SumBool).

differ_from_exactly_k_pos_c(K,VECTOR1,VECTOR2) :-
        integer(K),
        collection(VECTOR1,[int]),
        collection(VECTOR2,[int]),
        length(VECTOR1,N),
        length(VECTOR2,N),
        N>=1,
        K>=0,
        K=<N,
        differ_from_exactly_k_pos_check(VECTOR1,VECTOR2,N,K).

differ_from_exactly_k_pos_check([],[],_39223,0) :-
        !.

differ_from_exactly_k_pos_check(
    [[_39230-U]|R],
    [[_39241-V]|S],
    N,
```

```
K) :-
    (   U=V ->
        NewK is K
    ;   NewK is K-1,
        NewK>=0
    ),
    NewN is N-1,
    NewK=<NewN,
    differ_from_exactly_k_pos_check(R,S,NewN,NewK).
```

B.120 diffn

◇ META-DATA:

```
ctr_date(
    diffn,
    ['20000128','20030820','20040530','20051001','20060808']).

ctr_origin(diffn,'\\cite{BeldiceanuContejean94}',[]).

ctr_synonyms(diffn,[disjoint,disjoint1,disjoint2,diff2]).

ctr_types(
    diffn,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    diffn,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE')]).

ctr_restrictions(
    diffn,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     required('ORTHOTOPES',orth),
     same_size('ORTHOTOPES',orth)]).

ctr_example(
    diffn,
    diffn(
        [[orth-[[ori-2,siz-2,end-4],[ori-1,siz-2,end-3]]],
         [orth-[[ori-4,siz-4,end-8],[ori-2,siz-2,end-4]]],
         [orth-[[ori-6,siz-5,end-11],[ori-5,siz-2,end-7]]]])).

ctr_typical(
    diffn,
    [size('ORTHOTOPE')>1,
     'ORTHOTOPE'^siz>0,
     size('ORTHOTOPES')>1]).

ctr_exchangeable(
    diffn,
    [items('ORTHOTOPES',all),
     items_sync('ORTHOTOPES'^orth,all),
```

```
     vals(
          ['ORTHOTOPES'^orth^siz],
          int(>=(0)),
          >,
          dontcare,
          dontcare),
     translate(['ORTHOTOPES'^orth^ori,'ORTHOTOPES'^orth^end])]).

ctr_graph(
    diffn,
    ['ORTHOTOPES'],
    1,
    ['SELF'>>collection(orthotopes)],
    [orth_link_ori_siz_end(orthotopes^orth)],
    ['NARC'=size('ORTHOTOPES')],
    []).

ctr_graph(
    diffn,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE'(=\=)>>collection(orthotopes1,orthotopes2)],
    [two_orth_do_not_overlap(
         orthotopes1^orth,
         orthotopes2^orth)],
    [NARC=
     size('ORTHOTOPES')*size('ORTHOTOPES')-size('ORTHOTOPES')],
    []).

ctr_eval(diffn,[reformulation(diffn_r),density(diffn_d)]).

ctr_contractible(diffn,[],'ORTHOTOPES',any).

ctr_application(diffn,[1]).

diffn_r([]) :-
        !.

diffn_r(ORTHOTOPES) :-
        ORTHOTOPES=[[_77236-ORTH1]|_77232],
        length(ORTH1,K),
        collection(
            ORTHOTOPES,
            [col(K,[dvar,dvar_gteq(0),dvar])]),
        get_col_attr1(ORTHOTOPES,1,ORIS),
        get_col_attr1(ORTHOTOPES,2,SIZS),
```

```
        get_col_attr1(ORTHOTOPES,3,ENDS),
        (   K=2 ->
            diffn0(ORIS,SIZS,ENDS,RECTS),
            disjoint2(RECTS)
        ;   diffn_fixed_size(SIZS) ->
            length(Zeros,K),
            domain(Zeros,0,0),
            diffn5(ORIS,SIZS,ENDS,1,Zeros,OBJS,SHAPES),
            geost(OBJS,SHAPES)
        ;   diffn1(ORIS,SIZS,ENDS)
        ).

diffn_fixed_size([]).

diffn_fixed_size([L|R]) :-
        diffn_fixed_size1(L),
        diffn_fixed_size(R).

diffn_fixed_size1([]).

diffn_fixed_size1([S|R]) :-
        integer(S),
        S\=0,
        diffn_fixed_size1(R).

diffn0([],[],[],[]).

diffn0([[X,Y]|ORIS],[[L,H]|SIZS],[END|ENDS],[t(X,L,Y,H)|R]) :-
        diffn2([X,Y],[L,H],END),
        diffn0(ORIS,SIZS,ENDS,R).

diffn1([ORI1],[SIZ1],[END1]) :-
        !,
        diffn2(ORI1,SIZ1,END1).

diffn1([ORI1,ORI2|ORIS],[SIZ1,SIZ2|SIZS],[END1,END2|ENDS]) :-
        diffn2(ORI1,SIZ1,END1),
        diffn3([ORI2|ORIS],[END2|ENDS],ORI1,END1),
        diffn1([ORI2|ORIS],[SIZ2|SIZS],[END2|ENDS]).

diffn2([],[],[]).

diffn2([O|RO],[S|RS],[E|RE]) :-
        E#=O+S,
        diffn2(RO,RS,RE).
```

```
diffn3([],[],_77221,_77222).

diffn3([ORI2|ORIS],[END2|ENDS],ORI1,END1) :-
        diffn4(ORI1,END1,ORI2,END2,Disjunction),
        call(Disjunction),
        diffn3(ORIS,ENDS,ORI1,END1).

diffn4([],[],[],[],0).

diffn4([O1|R],[E1|S],[O2|T],[E2|U],E1#=<O2#\/E2#=<O1#\/V) :-
        diffn4(R,S,T,U,V).

diffn5([],[],[],_77222,_77223,[],[]).

diffn5(
    [ORI|ORIS],
    [SIZ|SIZS],
    [END|ENDS],
    I,
    Zeros,
    [object(I,I,ORI)|OBJS],
    [sbox(I,Zeros,SIZ)|SHAPES]) :-
        diffn2(ORI,SIZ,END),
        I1 is I+1,
        diffn5(ORIS,SIZS,ENDS,I1,Zeros,OBJS,SHAPES).

diffn_d(0,[]) :-
        !.

diffn_d(Density,[O|R]) :-
        O=[orth-L],
        length(L,N),
        length(LMin,N),
        length(LMax,N),
        diffn_minmax([O|R],LMin,LMax,Min,Max),
        diffn_availabel(Min,Max,1,Available),
        diffn_needed([O|R],0,Needed),
        Density is Needed/Available.

diffn_minmax([],Min,Max,Min,Max) :-
        !.

diffn_minmax([[orth-O]|R],LMin,LMax,Min,Max) :-
        diffn_minmax1(O,LMin,LMax,LMin1,LMax1),
        diffn_minmax(R,LMin1,LMax1,Min,Max).
```

```
diffn_minmax1([],[],[],[],[]) :-
        !.

diffn_minmax1(
    [[_77232-O,_77236,_77241-E]|R],
    [MinCur|S],
    [MaxCur|T],
    [MinNew|U],
    [MaxNew|V]) :-
        (   var(MinCur) ->
            MinNew=O
        ;   MinNew is min(O,MinCur)
        ),
        (   var(MaxCur) ->
            MaxNew=E
        ;   MaxNew is max(E,MaxCur)
        ),
        diffn_minmax1(R,S,T,U,V).

diffn_availabel([],[],A,A) :-
        !.

diffn_availabel([Min|R],[Max|S],Cur,Res) :-
        NewCur is Cur*(Max-Min),
        diffn_availabel(R,S,NewCur,Res).

diffn_needed([],N,N) :-
        !.

diffn_needed([[orth-O]|R],Cur,Res) :-
        diffn_vol(O,1,Vol),
        NewCur is Cur+Vol,
        diffn_needed(R,NewCur,Res).

diffn_vol([],V,V) :-
        !.

diffn_vol([[_77227,_77232-S,_77236]|R],Cur,Res) :-
        NewCur is Cur*S,
        diffn_vol(R,NewCur,Res).
```

B.121 diffn_column

◊ **META-DATA:**

```
ctr_date(diffn_column,['20030820']).

ctr_origin(
    diffn_column,
    \index{CHIP|indexuse}CHIP: option guillotine cut (column) of %c.,
    [diffn]).

ctr_types(
    diffn_column,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    diffn_column,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE'),'DIM'-int]).

ctr_restrictions(
    diffn_column,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     required('ORTHOTOPES',orth),
     same_size('ORTHOTOPES',orth),
     'DIM'>0,
     'DIM'=<size('ORTHOTOPE'),
     diffn('ORTHOTOPES')]).

ctr_example(
    diffn_column,
    diffn_column(
        [[orth-[[ori-1,siz-3,end-4],[ori-3,siz-2,end-5]]],
         [orth-[[ori-9,siz-1,end-10],[ori-4,siz-3,end-7]]],
         [orth-[[ori-4,siz-2,end-6],[ori-3,siz-4,end-7]]],
         [orth-[[ori-1,siz-3,end-4],[ori-6,siz-1,end-7]]],
         [orth-[[ori-6,siz-2,end-8],[ori-1,siz-4,end-5]]],
         [orth-[[ori-10,siz-1,end-11],[ori-1,siz-1,end-2]]],
         [orth-[[ori-9,siz-1,end-10],[ori-1,siz-1,end-2]]],
         [orth-[[ori-6,siz-2,end-8],[ori-6,siz-1,end-7]]]],
        1)).

ctr_typical(
    diffn_column,
```

```
    [size('ORTHOTOPE')>1,
     'ORTHOTOPE'^siz>0,
     size('ORTHOTOPES')>1]).

ctr_exchangeable(
    diffn_column,
    [items('ORTHOTOPES',all),
     translate(['ORTHOTOPES'^orth^ori,'ORTHOTOPES'^orth^end])]).

ctr_graph(
    diffn_column,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE'(<)>>collection(orthotopes1,orthotopes2)],
    [two_orth_column(orthotopes1^orth,orthotopes2^orth,'DIM')],
    ['NARC'=size('ORTHOTOPES')*(size('ORTHOTOPES')-1)/2],
    []).

ctr_eval(diffn_column,[reformulation(diffn_column_r)]).

ctr_contractible(diffn_column,[],'ORTHOTOPES',any).

ctr_application(diffn_column,[1]).

diffn_column_r([],DIM) :-
        integer(DIM),
        DIM>0.

diffn_column_r(ORTHOTOPES,DIM) :-
        ORTHOTOPES=[[_41235-ORTH1]|_41231],
        length(ORTH1,K),
        collection(
            ORTHOTOPES,
            [col(K,[dvar,dvar_gteq(0),dvar])]),
        check_type(int(1,K),DIM),
        eval(diffn(ORTHOTOPES)),
        get_attr1(ORTHOTOPES,ORTHOTOPES1),
        diffn_column1(ORTHOTOPES1,DIM).

diffn_column1([],_41216).

diffn_column1([_41220],_41219) :-
        !.

diffn_column1([O1,O2|R],DIM) :-
        eval(two_orth_column(O1,O2,DIM)),
```

```
diffn_column1([O2|R],DIM).
```

B.122 diffn_include

♢ META-DATA:

```
ctr_date(diffn_include,['20030820','20090523']).

ctr_origin(
    diffn_include,
    \index{CHIP|indexuse}CHIP: option guillotine cut (include) of %c.,
    [diffn]).

ctr_types(
    diffn_include,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    diffn_include,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE'),'DIM'-int]).

ctr_restrictions(
    diffn_include,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     required('ORTHOTOPES',orth),
     same_size('ORTHOTOPES',orth),
     'DIM'>0,
     'DIM'=<size('ORTHOTOPE'),
     diffn('ORTHOTOPES')]).

ctr_example(
    diffn_include,
    diffn_include(
        [[orth-[[ori-8,siz-1,end-9],[ori-4,siz-1,end-5]]],
         [orth-[[ori-9,siz-1,end-10],[ori-4,siz-3,end-7]]],
         [orth-[[ori-6,siz-3,end-9],[ori-5,siz-2,end-7]]],
         [orth-[[ori-1,siz-3,end-4],[ori-6,siz-1,end-7]]],
         [orth-[[ori-4,siz-2,end-6],[ori-3,siz-4,end-7]]],
         [orth-[[ori-6,siz-4,end-10],[ori-1,siz-1,end-2]]],
         [orth-[[ori-10,siz-1,end-11],[ori-1,siz-1,end-2]]],
         [orth-[[ori-6,siz-5,end-11],[ori-2,siz-2,end-4]]],
         [orth-[[ori-6,siz-2,end-8],[ori-4,siz-1,end-5]]],
         [orth-[[ori-1,siz-5,end-6],[ori-1,siz-2,end-3]]],
         [orth-[[ori-1,siz-3,end-4],[ori-3,siz-2,end-5]]],
         [orth-[[ori-1,siz-2,end-3],[ori-5,siz-1,end-6]]]],
```

```
        1)).

ctr_typical(
    diffn_include,
    [size('ORTHOTOPE')>1,
     'ORTHOTOPE'^siz>0,
     size('ORTHOTOPES')>1]).

ctr_exchangeable(
    diffn_include,
    [items('ORTHOTOPES',all),
     translate(['ORTHOTOPES'^orth^ori,'ORTHOTOPES'^orth^end])]).

ctr_graph(
    diffn_include,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE'(<)>>collection(orthotopes1,orthotopes2)],
    [two_orth_include(
         orthotopes1^orth,
         orthotopes2^orth,
         DIM)],
    ['NARC'=size('ORTHOTOPES')*(size('ORTHOTOPES')-1)/2],
    []).

ctr_eval(diffn_include,[reformulation(diffn_include_r)]).

ctr_contractible(diffn_include,[],'ORTHOTOPES',any).

ctr_application(diffn_include,[1]).

diffn_include_r([],DIM) :-
        integer(DIM),
        DIM>0.

diffn_include_r(ORTHOTOPES,DIM) :-
        ORTHOTOPES=[[_42725-ORTH1]|_42721],
        length(ORTH1,K),
        collection(
            ORTHOTOPES,
            [col(K,[dvar,dvar_gteq(0),dvar])]),
        check_type(int(1,K),DIM),
        eval(diffn(ORTHOTOPES)),
        get_attr1(ORTHOTOPES,ORTHOTOPES1),
        diffn_include1(ORTHOTOPES1,DIM).
```

```
diffn_include1([],_42706).

diffn_include1([_42710],_42709) :-
        !.

diffn_include1([O1,O2|R],DIM) :-
        eval(two_orth_include(O1,O2,DIM)),
        diffn_include1([O2|R],DIM).
```

B.123 discrepancy

◇ **META-DATA:**

```
ctr_date(discrepancy,['20050506','20060808']).

ctr_origin(
    discrepancy,
    \cite{Focacci01} and \cite{vanHoeve05},
    []).

ctr_arguments(
    discrepancy,
    ['VARIABLES'-collection(var-dvar,bad-sint),'K'-int]).

ctr_restrictions(
    discrepancy,
    [required('VARIABLES',var),
     required('VARIABLES',bad),
     'K'>=0,
     'K'=<size('VARIABLES')]).

ctr_example(
    discrepancy,
    discrepancy(
        [[var-4,bad-{1,4,6}],
         [var-5,bad-{0,1}],
         [var-5,bad-{1,6,9}],
         [var-4,bad-{1,4}],
         [var-1,bad-{}]],
        2)).

ctr_typical(
    discrepancy,
    [size('VARIABLES')>1,'K'<size('VARIABLES')]).

ctr_exchangeable(
    discrepancy,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var,'VARIABLES'^bad],
         int,
         =\=,
         all,
         dontcare)]).
```

```
ctr_graph(
    discrepancy,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var in_set variables^bad],
    ['NARC'='K'],
    []).

ctr_pure_functional_dependency(discrepancy,[]).

ctr_functional_dependency(discrepancy,2,[1]).

ctr_aggregate(discrepancy,[],[union,+]).
```

B.124 disj

◊ **META-DATA:**

```
ctr_date(disj,['20070527']).

ctr_origin(disj,'\\cite{MonetteDevilleDupont07}',[]).

ctr_arguments(
    disj,
    [TASKS-
     collection(
         start-dvar,
         duration-dvar,
         before-svar,
         position-dvar)]).

ctr_restrictions(
    disj,
    [required('TASKS',[start,duration,before,position]),
     'TASKS'^duration>=1,
     'TASKS'^position>=0,
     'TASKS'^position<size('TASKS')]).

ctr_example(
    disj,
    disj(
        [[start-1,duration-3,before-{},position-0],
         [start-9,duration-1,before-{1,3,4},position-3],
         [start-7,duration-2,before-{1,4},position-2],
         [start-4,duration-1,before-{1},position-1]])).

ctr_typical(disj,[size('TASKS')>1]).

ctr_exchangeable(
    disj,
    [translate(['TASKS'^start]),
     vals(['TASKS'^duration],int(>=(1)),>,dontcare,dontcare)]).

ctr_graph(
    disj,
    ['TASKS'],
    2,
    ['CLIQUE'(=\=)>>collection(tasks1,tasks2)],
    [tasks1^start+tasks1^duration=<tasks2^start#\/
     tasks2^start+tasks2^duration=<tasks1^start,
```

```
     tasks1^start+tasks1^duration=<tasks2^start#<=>
     tasks1^key in_set tasks2^before,
     tasks1^start+tasks1^duration=<tasks2^start#<=>
     tasks1^position<tasks2^position],
    ['NARC'=size('TASKS')*size('TASKS')-size('TASKS')],
    []).

ctr_application(disj,[1]).
```

B.125 disjoint

◇ **META-DATA:**

```
ctr_date(
    disjoint,
    ['20000315','20031017','20040530','20060808']).

ctr_origin(disjoint,'Derived from %c.',[alldifferent]).

ctr_arguments(
    disjoint,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    disjoint,
    [required('VARIABLES1',var),required('VARIABLES2',var)]).

ctr_example(
    disjoint,
    disjoint(
        [[var-1],[var-9],[var-1],[var-5]],
        [[var-2],[var-7],[var-7],[var-0],[var-6],[var-8]])).

ctr_typical(
    disjoint,
    [size('VARIABLES1')>1,size('VARIABLES2')>1]).

ctr_exchangeable(
    disjoint,
    [args([['VARIABLES1','VARIABLES2']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(['VARIABLES1'^var],int,=\=,dontcare,in),
     vals(['VARIABLES2'^var],int,=\=,dontcare,in),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    disjoint,
    ['VARIABLES1','VARIABLES2'],
```

```
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NARC'=0],
    []).

ctr_eval(disjoint,[reformulation(disjoint_r)]).

ctr_contractible(disjoint,[],'VARIABLES1',any).

ctr_contractible(disjoint,[],'VARIABLES2',any).

disjoint_r(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        disjoint1_(VARS1,VARS2).

disjoint1_([],_45459).

disjoint1_([V|R],VARS2) :-
        disjoint2_(VARS2,V),
        disjoint1_(R,VARS2).

disjoint2_([],_45459).

disjoint2_([U|R],V) :-
        U#\=V,
        disjoint2_(R,V).
```

B.126 disjoint_sboxes

◇ **META-DATA:**

```
ctr_date(disjoint_sboxes,['20070622','20090725']).

ctr_origin(
    disjoint_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(disjoint_sboxes,[disjoint]).

ctr_types(
    disjoint_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    disjoint_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    disjoint_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    disjoint_sboxes,
    disjoint_sboxes(
        2,
        {0,1},
        [[oid-1,sid-1,x-[[v-1],[v-1]]],
         [oid-2,sid-2,x-[[v-4],[v-1]]],
         [oid-3,sid-4,x-[[v-2],[v-4]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-1],[v-2]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-1],[v-1]]],
         [sid-2,t-[[v-1],[v-0]],l-[[v-1],[v-3]]],
         [sid-2,t-[[v-0],[v-2]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
         [sid-3,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-4,t-[[v-0],[v-0]],l-[[v-1],[v-1]]]])).

ctr_typical(disjoint_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    disjoint_sboxes,
    [items('OBJECTS',all),
     items('SBOXES',all),
     vals(['SBOXES'^l^v],int(>=(1)),>,dontcare,dontcare)]).

ctr_eval(disjoint_sboxes,[logic(disjoint_sboxes_g)]).

ctr_logic(
    disjoint_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (disjoint_sboxes(Dims,O1,S1,O2,S2)--->
      exists(
          D,
          Dims,
          origin(O1,S1,D)#>end(O2,S2,D)#\/
          origin(O2,S2,D)#>end(O1,S1,D))),
     (disjoint_objects(Dims,O1,O2)--->
```

```
      forall(
          S1,
          sboxes([O1^sid]),
          forall(
              S2,
              sboxes([O2^sid]),
              disjoint_sboxes(Dims,O1,S1,O2,S2)))),
     (all_disjoint(Dims,OIDS)--->
      forall(
          O1,
          objects(OIDS),
          forall(
              O2,
              objects(OIDS),
              O1^oid#<O2^oid#=>
              disjoint_objects(Dims,O1,O2)))),
     all_disjoint(DIMENSIONS,OIDS)]).

ctr_contractible(disjoint_sboxes,[],'OBJECTS',suffix).

ctr_application(disjoint_sboxes,[3]).

disjoint_sboxes_g(K,_40171,[],_40173) :-
        !,
        check_type(int_gteq(1),K).

disjoint_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        collection_increasing_seq(OBJECTS,[1]),
        geost1(OIDS,SIDS,XS,Objects),
        geost2(SIDES,TS,TL,Sboxes),
        geost_dims(1,K,DIMENSIONS),
```

```
ctr_logic(disjoint_sboxes,[DIMENSIONS,OIDS],Rules),
geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.127 disjoint_tasks

◊ **META-DATA:**

```
ctr_date(disjoint_tasks,['20030820','20060808']).

ctr_origin(disjoint_tasks,'Derived from %c.',[disjoint]).

ctr_arguments(
    disjoint_tasks,
    ['TASKS1'-collection(origin-dvar,duration-dvar,end-dvar),
     'TASKS2'-collection(origin-dvar,duration-dvar,end-dvar)]).

ctr_restrictions(
    disjoint_tasks,
    [require_at_least(2,'TASKS1',[origin,duration,end]),
     'TASKS1'^duration>=0,
     'TASKS1'^origin=<'TASKS1'^end,
     require_at_least(2,'TASKS2',[origin,duration,end]),
     'TASKS2'^duration>=0,
     'TASKS2'^origin=<'TASKS2'^end]).

ctr_example(
    disjoint_tasks,
    disjoint_tasks(
        [[origin-6,duration-5,end-11],
         [origin-8,duration-2,end-10]],
        [[origin-2,duration-2,end-4],
         [origin-3,duration-3,end-6],
         [origin-12,duration-1,end-13]])).

ctr_typical(
    disjoint_tasks,
    [size('TASKS1')>1,
     'TASKS1'^duration>0,
     size('TASKS2')>1,
     'TASKS2'^duration>0]).

ctr_exchangeable(
    disjoint_tasks,
    [args([['TASKS1','TASKS2']]),
     items('TASKS1',all),
     items('TASKS2',all),
     translate(
         ['TASKS1'^origin,
          'TASKS1'^end,
```

```
          'TASKS2'^origin,
          'TASKS2'^end])]).

ctr_graph(
    disjoint_tasks,
    ['TASKS1'],
    1,
    ['SELF'>>collection(tasks1)],
    [tasks1^origin+tasks1^duration=tasks1^end],
    ['NARC'=size('TASKS1')],
    []).

ctr_graph(
    disjoint_tasks,
    ['TASKS2'],
    1,
    ['SELF'>>collection(tasks2)],
    [tasks2^origin+tasks2^duration=tasks2^end],
    ['NARC'=size('TASKS2')],
    []).

ctr_graph(
    disjoint_tasks,
    ['TASKS1','TASKS2'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1^duration>0,
     tasks2^duration>0,
     tasks1^origin<tasks2^end,
     tasks2^origin<tasks1^end],
    ['NARC'=0],
    []).

ctr_eval(disjoint_tasks,[reformulation(disjoint_tasks_r)]).

ctr_contractible(disjoint_tasks,[],'TASKS1',any).

ctr_contractible(disjoint_tasks,[],'TASKS2',any).

ctr_application(disjoint_tasks,[1,2]).

disjoint_tasks_r(TASKS1,TASKS2) :-
        collection(TASKS1,[dvar,dvar_gteq(0),dvar]),
        collection(TASKS2,[dvar,dvar_gteq(0),dvar]),
        get_attr1(TASKS1,ORIGINS1),
        get_attr2(TASKS1,DURATIONS1),
```

```
        get_attr3(TASKS1,ENDS1),
        ori_dur_end(ORIGINS1,DURATIONS1,ENDS1),
        get_attr1(TASKS2,ORIGINS2),
        get_attr2(TASKS2,DURATIONS2),
        get_attr3(TASKS2,ENDS2),
        ori_dur_end(ORIGINS2,DURATIONS2,ENDS2),
        disjoint_tasks1(ORIGINS1,ENDS1,ORIGINS2,ENDS2).

disjoint_tasks1([],[],_44161,_44162).

disjoint_tasks1([O|R],[E|S],ORIGINS2,ENDS2) :-
        disjoint_tasks2(ORIGINS2,ENDS2,O,E),
        disjoint_tasks1(R,S,ORIGINS2,ENDS2).

disjoint_tasks2([],[],_44161,_44162).

disjoint_tasks2([Oj|R],[Ej|S],Oi,Ei) :-
        Ei#=<Oj#\/Ej#=<Oi,
        disjoint_tasks2(R,S,Oi,Ei).
```

B.128 disjunctive

♢ META-DATA:

```
ctr_date(disjunctive,['20050506','20060808']).

ctr_origin(disjunctive,'\\cite{Carlier82}',[]).

ctr_synonyms(disjunctive,[one_machine]).

ctr_arguments(
    disjunctive,
    ['TASKS'-collection(origin-dvar,duration-dvar)]).

ctr_restrictions(
    disjunctive,
    [required('TASKS',[origin,duration]),'TASKS'^duration>=0]).

ctr_example(
    disjunctive,
    disjunctive(
        [[origin-1,duration-3],
         [origin-2,duration-0],
         [origin-7,duration-2],
         [origin-4,duration-1]])).

ctr_typical(disjunctive,[size('TASKS')>2,'TASKS'^duration>=1]).

ctr_exchangeable(
    disjunctive,
    [items('TASKS',all),
     vals(['TASKS'^duration],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'^origin])]).

ctr_graph(
    disjunctive,
    ['TASKS'],
    2,
    ['CLIQUE'(<)>>collection(tasks1,tasks2)],
    [tasks1^duration=0#\/tasks2^duration=0#\/
     tasks1^origin+tasks1^duration=<tasks2^origin#\/
     tasks2^origin+tasks2^duration=<tasks1^origin],
    ['NARC'=size('TASKS')*(size('TASKS')-1)/2],
    []).

ctr_eval(
```

```
    disjunctive,
    [checker(disjunctive_c),builtin(disjunctive_b)]).

ctr_contractible(disjunctive,[],'TASKS',any).

ctr_cond_imply(
    disjunctive,
    alldifferent,
    [minval('TASKS'^duration)>0],
    [],
    ['TASKS'^origin]).

ctr_cond_imply(
    disjunctive,
    alldifferent_cst,
    [minval('TASKS'^duration)>0],
    [],
    same).

ctr_application(disjunctive,[1]).

disjunctive_b([]) :-
        !.

disjunctive_b(TASKS) :-
        collection(TASKS,[dvar,dvar_gteq(0)]),
        length(TASKS,N),
        (   N=1 ->
            true
        ;   get_attr1(TASKS,ORIGINS),
            get_attr2(TASKS,DURATIONS),
            length(ENDS,N),
            ori_dur_end(ORIGINS,DURATIONS,ENDS),
            length(HEIGHTS,N),
            domain(HEIGHTS,1,1),
            gen_cum_tasks(
                ORIGINS,
                DURATIONS,
                ENDS,
                HEIGHTS,
                1,
                Tasks),
            cumulative(Tasks,[limit(1)])
        ).

disjunctive_c([]) :-
```

```
        !.

disjunctive_c(TASKS) :-
        collection(TASKS,[int,int_gteq(0)]),
        (   TASKS=[_52660] ->
            true
        ;   get_attr12_diff20(TASKS,ORIS_DURS),
            keysort(ORIS_DURS,SORTED_NON_ZERO_TASKS),
            disjunctive_check_prec(SORTED_NON_ZERO_TASKS)
        ).

disjunctive_check_prec([]) :-
        !.

disjunctive_check_prec([_52637]) :-
        !.

disjunctive_check_prec([O1-D1,O2-D2|R]) :-
        E1 is O1+D1,
        E1=<O2,
        disjunctive_check_prec([O2-D2|R]).
```

B.129 disjunctive_or_same_end

◇ **META-DATA:**

```
ctr_date(disjunctive_or_same_end,['20120205']).

ctr_origin(disjunctive_or_same_end,'Scheduling.',[]).

ctr_synonyms(
    disjunctive_or_same_end,
    [same_end_or_disjunctive,
     non_overlap_or_same_end,
     same_end_or_non_overlap]).

ctr_arguments(
    disjunctive_or_same_end,
    ['TASKS'-collection(origin-dvar,duration-dvar)]).

ctr_restrictions(
    disjunctive_or_same_end,
    [required('TASKS',[origin,duration]),'TASKS'^duration>=0]).

ctr_example(
    disjunctive_or_same_end,
    disjunctive_or_same_end(
        [[origin-4,duration-3],
         [origin-7,duration-2],
         [origin-5,duration-2]])).

ctr_typical(
    disjunctive_or_same_end,
    [size('TASKS')>2,'TASKS'^duration>=1]).

ctr_exchangeable(
    disjunctive_or_same_end,
    [items('TASKS',all),
     vals(['TASKS'^duration],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'^origin])]).

ctr_graph(
    disjunctive_or_same_end,
    ['TASKS'],
    2,
    ['CLIQUE'(<)>>collection(tasks1,tasks2)],
    [tasks1^duration=0#\/tasks2^duration=0#\/
     tasks1^origin+tasks1^duration=<tasks2^origin#\/
```

```
     tasks2^origin+tasks2^duration=<tasks1^origin#\/
     tasks1^origin+tasks1^duration=
     tasks2^origin+tasks2^duration],
    ['NARC'=size('TASKS')*(size('TASKS')-1)/2],
    []).

ctr_eval(
    disjunctive_or_same_end,
    [checker(disjunctive_or_same_end_c),
     reformulation(disjunctive_or_same_end_r)]).

ctr_contractible(disjunctive_or_same_end,[],'TASKS',any).

ctr_application(disjunctive_or_same_end,[1]).

disjunctive_or_same_end_r([]) :-
        !.

disjunctive_or_same_end_r(TASKS) :-
        collection(TASKS,[dvar,dvar_gteq(0)]),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        disjunctive_or_same_end1(ORIGINS,DURATIONS).

disjunctive_or_same_end1([],[]).

disjunctive_or_same_end1([ORI|RO],[DUR|RD]) :-
        disjunctive_or_same_end2(RO,RD,ORI,DUR),
        disjunctive_or_same_end1(RO,RD).

disjunctive_or_same_end2([],[],_38596,_38597).

disjunctive_or_same_end2([O2|RO],[D2|RD],O1,D1) :-
        D1#=0#\/D2#=0#\/O1+D1#=<O2#\/O2+D2#=<O1#\/O1+D1#=O2+D2,
        disjunctive_or_same_end2(RO,RD,O1,D1).

disjunctive_or_same_end_c([]) :-
        !.

disjunctive_or_same_end_c(TASKS) :-
        collection(TASKS,[int,int_gteq(0)]),
        (   TASKS=[_38621] ->
            true
        ;   get_attr12_diff20_end(TASKS,ENDS_NEGDURS),
            sort(ENDS_NEGDURS,SORTED_NON_ZERO_TASKS),
            reverse(
```

```
                SORTED_NON_ZERO_TASKS,
                RSORTED_NON_ZERO_TASKS),
            disjunctive_or_same_end_check_prec(
                RSORTED_NON_ZERO_TASKS)
        ).

disjunctive_or_same_end_check_prec([]) :-
        !.

disjunctive_or_same_end_check_prec([_38598]) :-
        !.

disjunctive_or_same_end_check_prec([E-_D1,E-D2|R]) :-
        !,
        disjunctive_or_same_end_check_prec([E-D2|R]).

disjunctive_or_same_end_check_prec([E1-D1,E2-D2|R]) :-
        !,
        O1 is E1+D1,
        O1>=E2,
        disjunctive_or_same_end_check_prec([E2-D2|R]).
```

B.130 disjunctive_or_same_start

◇ META-DATA:

```
ctr_date(disjunctive_or_same_start,['20120205']).

ctr_origin(disjunctive_or_same_start,'Scheduling.',[]).

ctr_synonyms(
    disjunctive_or_same_start,
    [same_start_or_disjunctive,
     non_overlap_or_same_start,
     same_start_or_non_overlap]).

ctr_arguments(
    disjunctive_or_same_start,
    ['TASKS'-collection(origin-dvar,duration-dvar)]).

ctr_restrictions(
    disjunctive_or_same_start,
    [required('TASKS',[origin,duration]),'TASKS'^duration>=0]).

ctr_example(
    disjunctive_or_same_start,
    disjunctive_or_same_start(
        [[origin-4,duration-3],
         [origin-7,duration-2],
         [origin-4,duration-1]])).

ctr_typical(
    disjunctive_or_same_start,
    [size('TASKS')>2,'TASKS'^duration>=1]).

ctr_exchangeable(
    disjunctive_or_same_start,
    [items('TASKS',all),
     vals(['TASKS'^duration],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'^origin])]).

ctr_graph(
    disjunctive_or_same_start,
    ['TASKS'],
    2,
    ['CLIQUE'(<)>>collection(tasks1,tasks2)],
    [tasks1^duration=0#\/tasks2^duration=0#\/
     tasks1^origin+tasks1^duration=<tasks2^origin#\/
```

```
     tasks2^origin+tasks2^duration=<tasks1^origin#\/
     tasks1^origin=tasks2^origin],
    ['NARC'=size('TASKS')*(size('TASKS')-1)/2],
    []).

ctr_eval(
    disjunctive_or_same_start,
    [checker(disjunctive_or_same_start_c),
     reformulation(disjunctive_or_same_start_r)]).

ctr_contractible(disjunctive_or_same_start,[],'TASKS',any).

ctr_application(disjunctive_or_same_start,[1]).

disjunctive_or_same_start_r([]) :-
        !.

disjunctive_or_same_start_r(TASKS) :-
        collection(TASKS,[dvar,dvar_gteq(0)]),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        disjunctive_or_same_start1(ORIGINS,DURATIONS).

disjunctive_or_same_start1([],[]).

disjunctive_or_same_start1([ORI|RO],[DUR|RD]) :-
        disjunctive_or_same_start2(RO,RD,ORI,DUR),
        disjunctive_or_same_start1(RO,RD).

disjunctive_or_same_start2([],[],_38512,_38513).

disjunctive_or_same_start2([O2|RO],[D2|RD],O1,D1) :-
        D1#=0#\/D2#=0#\/O1+D1#=<O2#\/O2+D2#=<O1#\/O1#=O2,
        disjunctive_or_same_start2(RO,RD,O1,D1).

disjunctive_or_same_start_c([]) :-
        !.

disjunctive_or_same_start_c(TASKS) :-
        collection(TASKS,[int,int_gteq(0)]),
        (   TASKS=[_38537] ->
            true
        ;   get_attr12_diff20(TASKS,ORIS_DURS),
            sort(ORIS_DURS,SORTED_NON_ZERO_TASKS),
            disjunctive_or_same_start_check_prec(
                SORTED_NON_ZERO_TASKS)
```

```
        ).

disjunctive_or_same_start_check_prec([]) :-
        !.

disjunctive_or_same_start_check_prec([_38514]) :-
        !.

disjunctive_or_same_start_check_prec([O-_D1,O-D2|R]) :-
        !,
        disjunctive_or_same_start_check_prec([O-D2|R]).

disjunctive_or_same_start_check_prec([O1-D1,O2-D2|R]) :-
        !,
        E1 is O1+D1,
        E1=<O2,
        disjunctive_or_same_start_check_prec([O2-D2|R]).
```

B.131 distance

◊ **META-DATA:**

```
ctr_predefined(distance).

ctr_date(distance,['20090416']).

ctr_origin(distance,'Arithmetic constraint.',[]).

ctr_arguments(distance,['X'-dvar,'Y'-dvar,'Z'-dvar]).

ctr_restrictions(distance,['Z'>=0]).

ctr_example(distance,distance(5,7,2)).

ctr_typical(distance,['Z'>0]).

ctr_exchangeable(distance,[args([['X','Y'],['Z']])]).

ctr_eval(distance,[checker(distance_c),builtin(distance_b)]).

ctr_pure_functional_dependency(distance,[]).

ctr_functional_dependency(distance,3,[1,2]).

distance_c(X,Y,Z) :-
        check_type(int,X),
        check_type(int,Y),
        check_type(dvar_gteq(0),Z),
        Z#=abs(X-Y).

distance_b(X,Y,Z) :-
        check_type(dvar,X),
        check_type(dvar,Y),
        check_type(dvar_gteq(0),Z),
        Z#=abs(X-Y).
```

B.132 distance_between

◇ META-DATA:

```
ctr_date(
    distance_between,
    ['20000128','20030820','20060808','20090428']).

ctr_origin(distance_between,'N.~Beldiceanu',[]).

ctr_synonyms(distance_between,[distance]).

ctr_arguments(
    distance_between,
    ['DIST'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    distance_between,
    ['DIST'>=0,
     DIST=<
     size('VARIABLES1')*size('VARIABLES2')-size('VARIABLES1'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     size('VARIABLES1')=size('VARIABLES2'),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    distance_between,
    distance_between(
        2,
        [[var-3],[var-4],[var-6],[var-2],[var-4]],
        [[var-2],[var-6],[var-9],[var-3],[var-6]],
        <)).

ctr_typical(
    distance_between,
    ['DIST'>0,
     DIST<
     size('VARIABLES1')*size('VARIABLES2')-size('VARIABLES1'),
     size('VARIABLES1')>1,
     in_list('CTR',[=,=\=])]).

ctr_exchangeable(
```

```
    distance_between,
    [args([['DIST'],['VARIABLES1','VARIABLES2'],['CTR']]),
     items_sync('VARIABLES1','VARIABLES2',all),
     translate(['VARIABLES1'^var]),
     translate(['VARIABLES2'^var])]).

ctr_graph(
    distance_between,
    [['VARIABLES1'],['VARIABLES2']],
    2,
    ['CLIQUE'(=\=)>>collection(variables1,variables2)],
    ['CTR'(variables1^var,variables2^var)],
    ['DISTANCE'='DIST'],
    []).

ctr_eval(distance_between,[reformulation(distance_between_r)]).

ctr_pure_functional_dependency(distance_between,[]).

ctr_functional_dependency(distance_between,1,[2,3,4]).

distance_between_r(DIST,VARIABLES1,VARIABLES2,CTR) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1=L2,
        L12 is L1*L2-L1,
        check_type(dvar(0,L12),DIST),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        distance_between1(VARS1,VARS2,1,VARS1,VARS2,CTR,TERM),
        call(DIST#=TERM).

distance_between1([],[],_38533,_38534,_38535,_38536,0).

distance_between1(
    [VAR1|RVARS1],
    [VAR2|RVARS2],
    IVAR,
    VARS1,
    VARS2,
    CTR,
    TERM+R) :-
        distance_between2(
```

```
            VARS1,
            VARS2,
            VAR1,
            VAR2,
            IVAR,
            1,
            CTR,
            TERM),
        IVAR1 is IVAR+1,
        distance_between1(
            RVARS1,
            RVARS2,
            IVAR1,
            VARS1,
            VARS2,
            CTR,
            R).

distance_between2([],[],_38533,_38534,_38535,_38536,_38537,0).

distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    =,
    B12+S) :-
        !,
        (   IVAR=\=IUAR ->
            B12#<=>
            VAR1#=UAR1#/\VAR2#\=UAR2#\/VAR1#\=UAR1#/\VAR2#=UAR2
        ;   B12=0
        ),
        IUAR1 is IUAR+1,
        distance_between2(
            RUARS1,
            RUARS2,
            VAR1,
            VAR2,
            IVAR,
            IUAR1,
            =,
            S).
```

```
distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    =\=,
    B12+S) :-
        !,
        (   IVAR=\=IUAR ->
            B12#<=>
            VAR1#\=UAR1#/\VAR2#=UAR2#\/VAR1#=UAR1#/\VAR2#\=UAR2
        ;   B12=0
        ),
        IUAR1 is IUAR+1,
        distance_between2(
            RUARS1,
            RUARS2,
            VAR1,
            VAR2,
            IVAR,
            IUAR1,
            =\=,
            S).

distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    <,
    B12+S) :-
        !,
        (   IVAR=\=IUAR ->
            B12#<=>
            VAR1#<UAR1#/\VAR2#>=UAR2#\/VAR1#>=UAR1#/\VAR2#<UAR2
        ;   B12=0
        ),
        IUAR1 is IUAR+1,
        distance_between2(
            RUARS1,
            RUARS2,
            VAR1,
```

```
            VAR2,
            IVAR,
            IUAR1,
            <,
            S).

distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    >=,
    B12+S) :-
        !,
        (   IVAR=\=IUAR ->
            B12#<=>
            VAR1#>=UAR1#/\VAR2#<UAR2#\/VAR1#<UAR1#/\VAR2#>=UAR2
        ;   B12=0
        ),
        IUAR1 is IUAR+1,
        distance_between2(
            RUARS1,
            RUARS2,
            VAR1,
            VAR2,
            IVAR,
            IUAR1,
            >=,
            S).

distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    >,
    B12+S) :-
        !,
        (   IVAR=\=IUAR ->
            B12#<=>
            VAR1#>UAR1#/\VAR2#=<UAR2#\/VAR1#=<UAR1#/\VAR2#>UAR2
        ;   B12=0
```

```
        ),
        IUAR1 is IUAR+1,
        distance_between2(
            RUARS1,
            RUARS2,
            VAR1,
            VAR2,
            IVAR,
            IUAR1,
            >,
            S).

distance_between2(
    [UAR1|RUARS1],
    [UAR2|RUARS2],
    VAR1,
    VAR2,
    IVAR,
    IUAR,
    =<,
    B12+S) :-
        (   IVAR=\=IUAR ->
            B12#<=>
            VAR1#=<UAR1#/\VAR2#>UAR2#\/VAR1#>UAR1#/\VAR2#=<UAR2
        ;   B12=0
        ),
        IUAR1 is IUAR+1,
        distance_between2(
            RUARS1,
            RUARS2,
            VAR1,
            VAR2,
            IVAR,
            IUAR1,
            =<,
            S).
```

B.133 distance_change

◇ **META-DATA:**

```
ctr_date(
    distance_change,
    ['20000128','20030820','20040530','20060808']).

ctr_origin(distance_change,'Derived from %c.',[change]).

ctr_synonyms(distance_change,[distance]).

ctr_arguments(
    distance_change,
    ['DIST'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    distance_change,
    ['DIST'>=0,
     'DIST'<size('VARIABLES1'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     size('VARIABLES1')=size('VARIABLES2'),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    distance_change,
    distance_change(
        1,
        [[var-3],[var-3],[var-1],[var-2],[var-2]],
        [[var-4],[var-4],[var-3],[var-3],[var-3]],
        =\=)).

ctr_typical(
    distance_change,
    ['DIST'>0,size('VARIABLES1')>1,in_list('CTR',[=,=\=])]).

ctr_exchangeable(
    distance_change,
    [args([['DIST'],['VARIABLES1','VARIABLES2'],['CTR']]),
     translate(['VARIABLES1'^var]),
     translate(['VARIABLES2'^var])]).
```

```
ctr_graph(
    distance_change,
    [['VARIABLES1'],['VARIABLES2']],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'(variables1^var,variables2^var)],
    ['DISTANCE'='DIST'],
    []).

ctr_eval(
    distance_change,
    [reformulation(distance_change_r),
     automaton(distance_change_a)]).

ctr_pure_functional_dependency(distance_change,[]).

ctr_functional_dependency(distance_change,1,[2,3,4]).

distance_change_r(DIST,VARIABLES1,VARIABLES2,CTR) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1=L2,
        L is L1-1,
        check_type(dvar(0,L),DIST),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        distance_change1(VARS1,VARS2,CTR,TERM),
        call(DIST#=TERM).

distance_change1([],[],_38625,0).

distance_change1([_38630],[_38632],_38628,0) :-
        !.

distance_change1([UAR1,UAR2|R],[VAR1,VAR2|S],=,B12+T) :-
        !,
        B12#<=>
        UAR1#=UAR2#/\VAR1#\=VAR2#\/UAR1#\=UAR2#/\VAR1#=VAR2,
        distance_change1([UAR2|R],[VAR2|S],=,T).

distance_change1([UAR1,UAR2|R],[VAR1,VAR2|S],=\=,B12+T) :-
        !,
        B12#<=>
```

```
        UAR1#\=UAR2#/\VAR1#=VAR2#\/UAR1#=UAR2#/\VAR1#\=VAR2,
        distance_change1([UAR2|R],[VAR2|S],=\=,T).

distance_change1([UAR1,UAR2|R],[VAR1,VAR2|S],<,B12+T) :-
        !,
        B12#<=>
        UAR1#<UAR2#/\VAR1#>=VAR2#\/UAR1#>=UAR2#/\VAR1#<VAR2,
        distance_change1([UAR2|R],[VAR2|S],<,T).

distance_change1([UAR1,UAR2|R],[VAR1,VAR2|S],>=,B12+T) :-
        !,
        B12#<=>
        UAR1#>=UAR2#/\VAR1#<VAR2#\/UAR1#<UAR2#/\VAR1#>=VAR2,
        distance_change1([UAR2|R],[VAR2|S],>=,T).

distance_change1([UAR1,UAR2|R],[VAR1,VAR2|S],>,B12+T) :-
        !,
        B12#<=>
        UAR1#>UAR2#/\VAR1#=<VAR2#\/UAR1#=<UAR2#/\VAR1#>VAR2,
        distance_change1([UAR2|R],[VAR2|S],>,T).

distance_change1([UAR1,UAR2|R],[VAR1,VAR2|S],=<,B12+T) :-
        B12#<=>
        UAR1#=<UAR2#/\VAR1#>VAR2#\/UAR1#>UAR2#/\VAR1#=<VAR2,
        distance_change1([UAR2|R],[VAR2|S],=<,T).

distance_change_a(FLAG,DIST,VARIABLES1,VARIABLES2,CTR) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1=L2,
        L is L1-1,
        check_type(dvar(0,L),DIST),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        distance_change_signature(
            VARIABLES1,
            VARIABLES2,
            SIGNATURE,
            CTR),
        automaton(
            SIGNATURE,
            _41171,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
```

```
            [C],
            [0],
            [COUNT]),
        COUNT#=DIST#<=>FLAG.

distance_change_signature([],[],[],_38626).

distance_change_signature([_38630],[_38632],[],_38629) :-
        !.

distance_change_signature(
    [[var-VAR1i],[var-VAR1j]|VAR1s],
    [[var-VAR2i],[var-VAR2j]|VAR2s],
    [S|Ss],
    =) :-
        !,
        VAR1i#=VAR1j#/\VAR2i#\=VAR2j#\/
        VAR1i#\=VAR1j#/\VAR2i#=VAR2j#<=>
        S,
        distance_change_signature(
            [[var-VAR1j]|VAR1s],
            [[var-VAR2j]|VAR2s],
            Ss,
            =).

distance_change_signature(
    [[var-VAR1i],[var-VAR1j]|VAR1s],
    [[var-VAR2i],[var-VAR2j]|VAR2s],
    [S|Ss],
    =\=) :-
        !,
        VAR1i#\=VAR1j#/\VAR2i#=VAR2j#\/
        VAR1i#=VAR1j#/\VAR2i#\=VAR2j#<=>
        S,
        distance_change_signature(
            [[var-VAR1j]|VAR1s],
            [[var-VAR2j]|VAR2s],
            Ss,
            =\=).

distance_change_signature(
    [[var-VAR1i],[var-VAR1j]|VAR1s],
    [[var-VAR2i],[var-VAR2j]|VAR2s],
    [S|Ss],
    <) :-
        !,
```

```
        VAR1i#<VAR1j#/\VAR2i#>=VAR2j#\/
        VAR1i#>=VAR1j#/\VAR2i#<VAR2j#<=>
        S,
        distance_change_signature(
            [[var-VAR1j]|VAR1s],
            [[var-VAR2j]|VAR2s],
            Ss,
            <).

distance_change_signature(
    [[var-VAR1i],[var-VAR1j]|VAR1s],
    [[var-VAR2i],[var-VAR2j]|VAR2s],
    [S|Ss],
    >=) :-
        !,
        VAR1i#>=VAR1j#/\VAR2i#<VAR2j#\/
        VAR1i#<VAR1j#/\VAR2i#>=VAR2j#<=>
        S,
        distance_change_signature(
            [[var-VAR1j]|VAR1s],
            [[var-VAR2j]|VAR2s],
            Ss,
            >=).

distance_change_signature(
    [[var-VAR1i],[var-VAR1j]|VAR1s],
    [[var-VAR2i],[var-VAR2j]|VAR2s],
    [S|Ss],
    >) :-
        !,
        VAR1i#>VAR1j#/\VAR2i#=<VAR2j#\/
        VAR1i#=<VAR1j#/\VAR2i#>VAR2j#<=>
        S,
        distance_change_signature(
            [[var-VAR1j]|VAR1s],
            [[var-VAR2j]|VAR2s],
            Ss,
            >).

distance_change_signature(
    [[var-VAR1i],[var-VAR1j]|VAR1s],
    [[var-VAR2i],[var-VAR2j]|VAR2s],
    [S|Ss],
    =<) :-
        !,
        VAR1i#=<VAR1j#/\VAR2i#>VAR2j#\/
```

```
VAR1i#>VAR1j#/\VAR2i#=<VAR2j#<=>
S,
distance_change_signature(
    [[var-VAR1j]|VAR1s],
    [[var-VAR2j]|VAR2s],
    Ss,
    =<).
```

B.134 divisible

◊ META-DATA:

```
ctr_predefined(divisible).

ctr_date(divisible,['20110612']).

ctr_origin(divisible,'Arithmetic.',[]).

ctr_synonyms(divisible,[div]).

ctr_arguments(divisible,['Q'-dvar,'D'-dvar]).

ctr_restrictions(divisible,['Q'>=0,'D'>0]).

ctr_example(divisible,divisible(12,4)).

ctr_typical(divisible,['Q'>1,'D'<'Q']).

ctr_eval(divisible,[builtin(divisible_b)]).

divisible_b(Q,D) :-
        check_type(dvar,Q),
        check_type(dvar,D),
        Q#>=0,
        D#>0,
        Q mod D#=0.
```

B.135 divisible_or

◊ **META-DATA:**

```
ctr_predefined(divisible_or).

ctr_date(divisible_or,['20120212']).

ctr_origin(divisible_or,'Arithmetic.',[]).

ctr_synonyms(divisible_or,[div_or]).

ctr_arguments(divisible_or,['C'-dvar,'D'-dvar]).

ctr_restrictions(divisible_or,['C'>0,'D'>0]).

ctr_example(divisible_or,divisible_or(4,12)).

ctr_eval(divisible_or,[builtin(divisible_or_b)]).

divisible_or_b(C,D) :-
        check_type(dvar,C),
        check_type(dvar,D),
        C#>0,
        D#>0,
        C mod D#=0#\/D mod C#=0.
```

B.136 dom_reachability

◊ META-DATA:

```
ctr_predefined(dom_reachability).

ctr_date(dom_reachability,['20061011']).

ctr_origin(
    dom_reachability,
    \cite{QuesadaVanRoyDevilleCollet06},
    []).

ctr_arguments(
    dom_reachability,
    ['SOURCE'-int,
     'FLOW_GRAPH'-collection(index-int,succ-svar),
     'DOMINATOR_GRAPH'-collection(index-int,succ-sint),
     TRANSITIVE_CLOSURE_GRAPH-
     collection(index-int,succ-svar)]).

ctr_restrictions(
    dom_reachability,
    ['SOURCE'>=1,
     'SOURCE'=<size('FLOW_GRAPH'),
     required('FLOW_GRAPH',[index,succ]),
     'FLOW_GRAPH'^index>=1,
     'FLOW_GRAPH'^index=<size('FLOW_GRAPH'),
     'FLOW_GRAPH'^succ>=1,
     'FLOW_GRAPH'^succ=<size('FLOW_GRAPH'),
     distinct('FLOW_GRAPH',index),
     required('DOMINATOR_GRAPH',[index,succ]),
     size('DOMINATOR_GRAPH')=size('FLOW_GRAPH'),
     'DOMINATOR_GRAPH'^index>=1,
     'DOMINATOR_GRAPH'^index=<size('DOMINATOR_GRAPH'),
     'DOMINATOR_GRAPH'^succ>=1,
     'DOMINATOR_GRAPH'^succ=<size('DOMINATOR_GRAPH'),
     distinct('DOMINATOR_GRAPH',index),
     required('TRANSITIVE_CLOSURE_GRAPH',[index,succ]),
     size('TRANSITIVE_CLOSURE_GRAPH')=size('FLOW_GRAPH'),
     'TRANSITIVE_CLOSURE_GRAPH'^index>=1,
     'TRANSITIVE_CLOSURE_GRAPH'^index=<
     size('TRANSITIVE_CLOSURE_GRAPH'),
     'TRANSITIVE_CLOSURE_GRAPH'^succ>=1,
     'TRANSITIVE_CLOSURE_GRAPH'^succ=<
     size('TRANSITIVE_CLOSURE_GRAPH'),
```

```
     distinct('TRANSITIVE_CLOSURE_GRAPH',index)]).

ctr_example(
    dom_reachability,
    dom_reachability(
        1,
        [[index-1,succ-{2}],
         [index-2,succ-{3,4}],
         [index-3,succ-{}],
         [index-4,succ-{}]],
        [[index-1,succ-{2,3,4}],
         [index-2,succ-{3,4}],
         [index-3,succ-{}],
         [index-4,succ-{}]],
        [[index-1,succ-{1,2,3,4}],
         [index-2,succ-{2,3,4}],
         [index-3,succ-{3}],
         [index-4,succ-{4}]])).

ctr_typical(dom_reachability,[size('FLOW_GRAPH')>2]).

ctr_exchangeable(
    dom_reachability,
    [items('FLOW_GRAPH',all),
     items('DOMINATOR_GRAPH',all),
     items('TRANSITIVE_CLOSURE_GRAPH',all)]).
```

B.137 domain

◇ META-DATA:

```
ctr_predefined(domain).

ctr_date(domain,['20070821']).

ctr_origin(domain,'Domain definition.',[]).

ctr_synonyms(domain,[dom]).

ctr_arguments(
    domain,
    ['VARIABLES'-collection(var-dvar),'LOW'-int,'UP'-int]).

ctr_restrictions(
    domain,
    [required('VARIABLES',var),'LOW'=<'UP']).

ctr_example(domain,domain([[var-2],[var-8],[var-2]],1,9)).

ctr_typical(domain,[size('VARIABLES')>1,'LOW'<'UP']).

ctr_exchangeable(
    domain,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         int('LOW' in 'UP'),
         =\=,
         dontcare,
         dontcare),
     vals(['LOW'],int,>,dontcare,dontcare),
     vals(['UP'],int,<,dontcare,dontcare),
     translate(['VARIABLES'^var,'LOW','UP'])]).

ctr_eval(domain,[builtin(domain_b)]).

ctr_contractible(domain,[],'VARIABLES',any).

domain_b(VARIABLES,LOW,UP) :-
        check_type(int,LOW),
        check_type(int,UP),
        LOW=<UP,
        collection(VARIABLES,[fdvar(LOW,UP)]),
```

```
get_attr1(VARIABLES,VARS),
domain(VARS,LOW,UP).
```

B.138 domain_constraint

◇ **META-DATA:**

```
ctr_date(domain_constraint,['20030820','20040530','20060808']).

ctr_origin(domain_constraint,'\\cite{Refalo00}',[]).

ctr_synonyms(domain_constraint,[domain]).

ctr_arguments(
    domain_constraint,
    ['VAR'-dvar,'VALUES'-collection(var01-dvar,value-int)]).

ctr_restrictions(
    domain_constraint,
    [required('VALUES',[var01,value]),
     'VALUES'^var01>=0,
     'VALUES'^var01=<1,
     distinct('VALUES',value)]).

ctr_example(
    domain_constraint,
    domain_constraint(
        5,
        [[var01-0,value-9],
         [var01-1,value-5],
         [var01-0,value-2],
         [var01-0,value-7]])).

ctr_typical(domain_constraint,[size('VALUES')>1]).

ctr_exchangeable(domain_constraint,[items('VALUES',all)]).

ctr_derived_collections(
    domain_constraint,
    [col('VALUE'-collection(var01-int,value-dvar),
         [item(var01-1,value-'VAR')])]).

ctr_graph(
    domain_constraint,
    ['VALUE','VALUES'],
    2,
    ['PRODUCT'>>collection(value,values)],
    [value^value=values^value#<=>values^var01=1],
    ['NARC'=size('VALUES')],
```

```
    []).

ctr_eval(
    domain_constraint,
    [reformulation(domain_constraint_r),
     automaton(domain_constraint_a)]).

domain_constraint_r(VAR,VALUES) :-
        check_type(dvar,VAR),
        collection(VALUES,[dvar(0,1),int]),
        get_attr1(VALUES,VARS01),
        get_attr2(VALUES,VALS),
        all_different(VALS),
        domain_constraint1(VARS01,VALS,VAR,Term),
        call(Term).

domain_constraint1([],[],_46518,0).

domain_constraint1(
    [VAR01|R],
    [VAL|S],
    VAR,
    VAR#=VAL#/\VAR01#=1#\/T) :-
        domain_constraint1(R,S,VAR,T).

domain_constraint_a(FLAG,VAR,VALUES) :-
        check_type(dvar,VAR),
        collection(VALUES,[dvar(0,1),int]),
        get_attr2(VALUES,VALS),
        all_different(VALS),
        domain_constraint_signature(VALUES,SIGNATURE,VAR),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _48175,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

domain_constraint_signature([],[],_46518).

domain_constraint_signature(
```

```
[[var01-VAR01,value-VALUE]|VALUES],
[S|Ss],
VAR) :-
    VAR#=VALUE#<=>VAR01#<=>S,
    domain_constraint_signature(VALUES,Ss,VAR).
```

B.139 elem

◊ **META-DATA:**

```
ctr_date(elem,['20030820','20040530','20060808']).

ctr_origin(elem,'Derived from %c.',[element]).

ctr_usual_name(elem,element).

ctr_synonyms(elem,[nth,array]).

ctr_arguments(
    elem,
    ['ITEM'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-dvar)]).

ctr_restrictions(
    elem,
    [required('ITEM',[index,value]),
     'ITEM'^index>=1,
     'ITEM'^index=<size('TABLE'),
     size('ITEM')=1,
     size('TABLE')>0,
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     distinct('TABLE',index)]).

ctr_example(
    elem,
    elem(
        [[index-3,value-2]],
        [[index-1,value-6],
         [index-2,value-9],
         [index-3,value-2],
         [index-4,value-9]])).

ctr_typical(elem,[size('TABLE')>1,range('TABLE'^value)>1]).

ctr_exchangeable(
    elem,
    [items('TABLE',all),
     vals(['ITEM'^value,'TABLE'^value],int,=\=,all,dontcare)]).

ctr_graph(
```

```
    elem,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index=table^index,item^value=table^value],
    ['NARC'=1],
    []).

ctr_eval(elem,[builtin(elem_b),automaton(elem_a)]).

ctr_pure_functional_dependency(elem,[]).

ctr_functional_dependency(elem,1-2,[1-1,2]).

ctr_cond_imply(
    elem,
    bin_packing_capa,
    ['TABLE'^value>=0],
    [],
    ['TABLE','ITEM']).

elem_b(ITEM,TABLE) :-
        length(ITEM,1),
        length(TABLE,N),
        collection(ITEM,[dvar(1,N),dvar]),
        collection(TABLE,[int(1,N),dvar]),
        sort_collection(TABLE,index,SORTED_TABLE),
        get_attr1(SORTED_TABLE,INDEXES),
        increasing_values(INDEXES),
        get_attr2(SORTED_TABLE,VALUES),
        get_attr1(ITEM,[INDEX]),
        get_attr2(ITEM,[VALUE]),
        element(INDEX,VALUES,VALUE).

elem_a(FLAG,ITEM,TABLE) :-
        length(ITEM,1),
        length(TABLE,N),
        collection(ITEM,[dvar(1,N),dvar]),
        collection(TABLE,[int(1,N),dvar]),
        get_attr1(TABLE,INDEXES),
        all_different(INDEXES),
        ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
        elem_signature(TABLE,SIGNATURE,ITEM_INDEX,ITEM_VALUE),
        AUTOMATON=
        automaton(
            SIGNATURE,
```

```
            _55949,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

elem_signature([],[],_53605,_53606).

elem_signature(
    [[index-TABLE_INDEX,value-TABLE_VALUE]|TABLEs],
    [S|Ss],
    ITEM_INDEX,
    ITEM_VALUE) :-
        ITEM_INDEX#=TABLE_INDEX#/\ITEM_VALUE#=TABLE_VALUE#<=>S,
        elem_signature(TABLEs,Ss,ITEM_INDEX,ITEM_VALUE).
```

B.140 elem_from_to

◇ META-DATA:

```
ctr_date(elem_from_to,['20091115']).

ctr_origin(elem_from_to,'Derived from %c.',[elem]).

ctr_synonyms(elem_from_to,[element_from_to]).

ctr_arguments(
    elem_from_to,
    [ITEM-
     collection(
         from-dvar,
         cst_from-int,
         to-dvar,
         cst_to-int,
         value-dvar),
     'TABLE'-collection(index-int,value-dvar)]).

ctr_restrictions(
    elem_from_to,
    [required('ITEM',[from,cst_from,to,cst_to,value]),
     'ITEM'^from>=1,
     'ITEM'^from=<size('TABLE'),
     'ITEM'^to>=1,
     'ITEM'^to=<size('TABLE'),
     'ITEM'^from=<'ITEM'^to,
     size('ITEM')=1,
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     increasing_seq('TABLE',[index])]).

ctr_example(
    elem_from_to,
    elem_from_to(
        [[from-1,cst_from-1,to-4,cst_to- -1,value-2]],
        [[index-1,value-6],
         [index-2,value-2],
         [index-3,value-2],
         [index-4,value-9],
         [index-5,value-9]])).

ctr_typical(
```

```
    elem_from_to,
    ['ITEM'^cst_from>=0,
     'ITEM'^cst_from=<1,
     'ITEM'^cst_to>= -1,
     'ITEM'^cst_to=<1,
     size('TABLE')>1,
     range('TABLE'^value)>1]).

ctr_exchangeable(
    elem_from_to,
    [vals(['ITEM'^value,'TABLE'^value],int,=\=,all,dontcare)]).

ctr_eval(elem_from_to,[automaton(elem_from_to_a)]).

elem_from_to_a(FLAG,ITEM,TABLE) :-
        length(TABLE,N),
        collection(ITEM,[dvar(1,N),int,dvar(1,N),int,dvar]),
        collection(TABLE,[int(1,N),dvar]),
        collection_increasing_seq(TABLE,[1]),
        ITEM=
        [[from-FROM,
          cst_from-CST_FROM,
          to-TO,
          cst_to-CST_TO,
          value-VALUE]],
        FROM#=<TO,
        F#=max(1,FROM+CST_FROM),
        T#=min(N,TO+CST_TO),
        elem_from_to_signature(
            TABLE,
            SIGNATURE,
            N,
            FROM,
            TO,
            F,
            T,
            VALUE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _34879,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s),arc(s,2,s),arc(s,3,s)],
            [],
            [],
```

```
            []),
        automaton_bool(FLAG,[0,1,2,3,4],AUTOMATON).

elem_from_to_signature(
    [],
    [],
    _30376,
    _30423,
    _30470,
    _30517,
    _30564,
    _30611).

elem_from_to_signature(
    [[index-TABLE_INDEX,value-TABLE_VALUE]|TABLEs],
    [S|Ss],
    N,
    FROM,
    TO,
    F,
    T,
    VALUE) :-
        S in 0..4,
        1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#>T#<=>S#=0,
        1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#=<T#/\
        F#>TABLE_INDEX#<=>
        S#=1,
        1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#=<T#/\
        T#<TABLE_INDEX#<=>
        S#=2,
        1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#=<T#/\
        F#=<TABLE_INDEX#/\
        TABLE_INDEX#=<T#/\
        VALUE#=TABLE_VALUE#<=>
        S#=3,
        1#=<FROM#/\FROM#=<TO#/\TO#=<N#/\F#=<T#/\
        F#=<TABLE_INDEX#/\
        TABLE_INDEX#=<T#/\
        VALUE#\=TABLE_VALUE#<=>
        S#=4,
        elem_from_to_signature(TABLEs,Ss,N,FROM,TO,F,T,VALUE).
```

B.141 element

◊ **META-DATA:**

```
ctr_date(
    element,
    ['20000128','20030820','20040530','20060808','20090923']).

ctr_origin(element,'\\cite{VanHentenryckCarillon88}',[]).

ctr_synonyms(element,[nth,element_var,array]).

ctr_arguments(
    element,
    ['INDEX'-dvar,'TABLE'-collection(value-dvar),'VALUE'-dvar]).

ctr_restrictions(
    element,
    ['INDEX'>=1,
     'INDEX'=<size('TABLE'),
     size('TABLE')>0,
     required('TABLE',value)]).

ctr_example(
    element,
    element(3,[[value-6],[value-9],[value-2],[value-9]],2)).

ctr_typical(element,[size('TABLE')>1,range('TABLE'^value)>1]).

ctr_exchangeable(
    element,
    [vals(['TABLE'^value,'VALUE'],int,=\=,all,dontcare)]).

ctr_derived_collections(
    element,
    [col('ITEM'-collection(index-dvar,value-dvar),
         [item(index-'INDEX',value-'VALUE')])]).

ctr_graph(
    element,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index=table^key,item^value=table^value],
    ['NARC'=1],
    []).
```

```
ctr_eval(element,[builtin(element_b),automaton(element_a)]).

ctr_pure_functional_dependency(element,[]).

ctr_functional_dependency(element,3,[1,2]).

ctr_extensible(element,[],'TABLE',suffix).

element_b(INDEX,TABLE,VALUE) :-
        check_type(dvar,INDEX),
        collection(TABLE,[dvar]),
        check_type(dvar,VALUE),
        length(TABLE,N),
        N>0,
        INDEX#>=1,
        INDEX#=<N,
        get_attr1(TABLE,VALUES),
        element(INDEX,VALUES,VALUE).

element_a(FLAG,INDEX,TABLE,VALUE) :-
        check_type(dvar,INDEX),
        collection(TABLE,[dvar]),
        check_type(dvar,VALUE),
        length(TABLE,N),
        N>0,
        INDEX#>=1,
        INDEX#=<N,
        element_signature(TABLE,INDEX,VALUE,1,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _60301,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

element_signature([],_58021,_58022,_58023,[]).

element_signature(
    [[value-TABLE_VALUE]|Ts],
    INDEX,
```

```
VALUE,
TABLE_KEY,
[B|Bs]) :-
    INDEX#=TABLE_KEY#/\VALUE#=TABLE_VALUE#<=>B,
    TABLE_KEY1 is TABLE_KEY+1,
    element_signature(Ts,INDEX,VALUE,TABLE_KEY1,Bs).
```

B.142 element_greatereq

♢ META-DATA:

```
ctr_date(element_greatereq,['20030820','20040530','20060808']).

ctr_origin(
    element_greatereq,
    \cite{OttossonThorsteinssonHooker99},
    []).

ctr_arguments(
    element_greatereq,
    ['ITEM'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-int)]).

ctr_restrictions(
    element_greatereq,
    [required('ITEM',[index,value]),
     'ITEM'^index>=1,
     'ITEM'^index=<size('TABLE'),
     size('ITEM')=1,
     size('TABLE')>0,
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     distinct('TABLE',index)]).

ctr_example(
    element_greatereq,
    element_greatereq(
        [[index-1,value-8]],
        [[index-1,value-6],
         [index-2,value-9],
         [index-3,value-2],
         [index-4,value-9]])).

ctr_typical(
    element_greatereq,
    [size('TABLE')>1,range('TABLE'^value)>1]).

ctr_exchangeable(
    element_greatereq,
    [items('TABLE',all),
     vals(['ITEM'^value,'TABLE'^value],int,=\=,all,dontcare)]).
```

```
ctr_graph(
    element_greatereq,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index=table^index,item^value>=table^value],
    ['NARC'=1],
    []).

ctr_eval(
    element_greatereq,
    [reformulation(element_greatereq_r),
     automaton(element_greatereq_a)]).

element_greatereq_r(ITEM,TABLE) :-
        length(ITEM,1),
        length(TABLE,N),
        N>0,
        collection(ITEM,[dvar(1,N),dvar]),
        collection(TABLE,[int(1,N),dvar]),
        sort_collection(TABLE,index,SORTED_TABLE),
        get_attr1(SORTED_TABLE,INDEXES),
        increasing_values(INDEXES),
        get_attr2(SORTED_TABLE,VALUES),
        get_attr1(ITEM,[INDEX]),
        get_attr2(ITEM,[VALUE]),
        element(INDEX,VALUES,VAL),
        VALUE#>=VAL.

element_greatereq_a(FLAG,ITEM,TABLE) :-
        length(ITEM,1),
        length(TABLE,N),
        N>0,
        collection(ITEM,[dvar(1,N),dvar]),
        collection(TABLE,[int(1,N),dvar]),
        get_attr1(TABLE,INDEXES),
        all_different(INDEXES),
        ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
        element_greatereq_signature(
            TABLE,
            SIGNATURE,
            ITEM_INDEX,
            ITEM_VALUE),
        AUTOMATON=
        automaton(
            SIGNATURE,
```

```
            _46485,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

element_greatereq_signature([],[],_43640,_43641).

element_greatereq_signature(
    [[index-TABLE_INDEX,value-TABLE_VALUE]|TABLEs],
    [S|Ss],
    ITEM_INDEX,
    ITEM_VALUE) :-
        ITEM_INDEX#=TABLE_INDEX#/\ITEM_VALUE#>=TABLE_VALUE#<=>S,
        element_greatereq_signature(
            TABLEs,
            Ss,
            ITEM_INDEX,
            ITEM_VALUE).
```

B.143 element_lesseq

◊ **META-DATA:**

```
ctr_date(element_lesseq,['20030820','20040530','20060808']).

ctr_origin(
    element_lesseq,
    \cite{OttossonThorsteinssonHooker99},
    []).

ctr_arguments(
    element_lesseq,
    ['ITEM'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-int)]).

ctr_restrictions(
    element_lesseq,
    [required('ITEM',[index,value]),
     'ITEM'^index>=1,
     'ITEM'^index=<size('TABLE'),
     size('ITEM')=1,
     size('TABLE')>0,
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     distinct('TABLE',index)]).

ctr_example(
    element_lesseq,
    element_lesseq(
        [[index-3,value-1]],
        [[index-1,value-6],
         [index-2,value-9],
         [index-3,value-2],
         [index-4,value-9]])).

ctr_typical(
    element_lesseq,
    [size('TABLE')>1,range('TABLE'^value)>1]).

ctr_exchangeable(
    element_lesseq,
    [items('TABLE',all),
     vals(['ITEM'^value,'TABLE'^value],int,=\=,all,dontcare)]).
```

```
ctr_graph(
    element_lesseq,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index=table^index,item^value=<table^value],
    ['NARC'=1],
    []).

ctr_eval(
    element_lesseq,
    [reformulation(element_lesseq_r),
     automaton(element_lesseq_a)]).

ctr_cond_imply(
    element_lesseq,
    bin_packing_capa,
    [minval('ITEM'^value)>0,'TABLE'^value>0],
    [],
    [same('TABLE'),same('ITEM')]).

element_lesseq_r(ITEM,TABLE) :-
        length(ITEM,1),
        length(TABLE,N),
        N>0,
        collection(ITEM,[dvar(1,N),dvar]),
        collection(TABLE,[int(1,N),dvar]),
        sort_collection(TABLE,index,SORTED_TABLE),
        get_attr1(SORTED_TABLE,INDEXES),
        increasing_values(INDEXES),
        get_attr2(SORTED_TABLE,VALUES),
        get_attr1(ITEM,[INDEX]),
        get_attr2(ITEM,[VALUE]),
        element(INDEX,VALUES,VAL),
        VALUE#=<VAL.

element_lesseq_a(FLAG,ITEM,TABLE) :-
        length(ITEM,1),
        length(TABLE,N),
        N>0,
        collection(ITEM,[dvar(1,N),dvar]),
        collection(TABLE,[int(1,N),dvar]),
        get_attr1(TABLE,INDEXES),
        all_different(INDEXES),
        ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
        element_lesseq_signature(
```

```
            TABLE,
            SIGNATURE,
            ITEM_INDEX,
            ITEM_VALUE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _47472,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

element_lesseq_signature([],[],_44627,_44628).

element_lesseq_signature(
    [[index-TABLE_INDEX,value-TABLE_VALUE]|TABLEs],
    [S|Ss],
    ITEM_INDEX,
    ITEM_VALUE) :-
        ITEM_INDEX#=TABLE_INDEX#/\ITEM_VALUE#=<TABLE_VALUE#<=>S,
        element_lesseq_signature(
            TABLEs,
            Ss,
            ITEM_INDEX,
            ITEM_VALUE).
```

B.144 element_matrix

◇ META-DATA:

```
ctr_date(element_matrix,['20031101','20060808']).

ctr_origin(element_matrix,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(element_matrix,[elem_matrix,matrix]).

ctr_arguments(
    element_matrix,
    ['MAX_I'-int,
     'MAX_J'-int,
     'INDEX_I'-dvar,
     'INDEX_J'-dvar,
     'MATRIX'-collection(i-int,j-int,v-int),
     'VALUE'-dvar]).

ctr_restrictions(
    element_matrix,
    ['MAX_I'>=1,
     'MAX_J'>=1,
     'INDEX_I'>=1,
     'INDEX_I'=<'MAX_I',
     'INDEX_J'>=1,
     'INDEX_J'=<'MAX_J',
     required('MATRIX',[i,j,v]),
     increasing_seq('MATRIX',[i,j]),
     'MATRIX'^i>=1,
     'MATRIX'^i=<'MAX_I',
     'MATRIX'^j>=1,
     'MATRIX'^j=<'MAX_J',
     size('MATRIX')='MAX_I'*'MAX_J']).

ctr_example(
    element_matrix,
    element_matrix(
        4,
        3,
        1,
        3,
        [[i-1,j-1,v-4],
         [i-1,j-2,v-1],
         [i-1,j-3,v-7],
         [i-2,j-1,v-1],
```

```
        [i-2,j-2,v-0],
        [i-2,j-3,v-8],
        [i-3,j-1,v-3],
        [i-3,j-2,v-2],
        [i-3,j-3,v-1],
        [i-4,j-1,v-0],
        [i-4,j-2,v-0],
        [i-4,j-3,v-6]],
       7)).

ctr_typical(
    element_matrix,
    ['MAX_I'>1,
     'MAX_J'>1,
     size('MATRIX')>3,
     maxval('MATRIX'^i)>1,
     maxval('MATRIX'^j)>1,
     range('MATRIX'^v)>1]).

ctr_exchangeable(
    element_matrix,
    [vals(['MATRIX'^v,'VALUE'],int,=\=,all,dontcare)]).

ctr_derived_collections(
    element_matrix,
    [col(ITEM-
         collection(index_i-dvar,index_j-dvar,value-dvar),
         [item(
              index_i-'INDEX_I',
              index_j-'INDEX_J',
              value-'VALUE')])]).

ctr_graph(
    element_matrix,
    ['ITEM','MATRIX'],
    2,
    ['PRODUCT'>>collection(item,matrix)],
    [item^index_i=matrix^i,
     item^index_j=matrix^j,
     item^value=matrix^v],
    ['NARC'=1],
    []).

ctr_eval(
    element_matrix,
    [reformulation(element_matrix_r),
```

```
     automaton(element_matrix_a)]).

element_matrix_r(MAX_I,MAX_J,INDEX_I,INDEX_J,MATRIX,VALUE) :-
        check_type(int,MAX_I),
        MAX_I>=1,
        check_type(int,MAX_J),
        MAX_J>=1,
        check_type(dvar,INDEX_I),
        INDEX_I#>=1,
        INDEX_I#=<MAX_I,
        check_type(dvar,INDEX_J),
        INDEX_J#>=1,
        INDEX_J#=<MAX_J,
        collection(MATRIX,[int(1,MAX_I),int(1,MAX_J),int]),
        length(MATRIX,N),
        N is MAX_I*MAX_J,
        collection_increasing_seq(MATRIX,[1,2]),
        check_type(dvar,VALUE),
        get_attr3(MATRIX,VALUES),
        element_matrix1(MAX_I,MAX_J,INDEX_J,VALUES,TABLE_VARS),
        element(INDEX_I,TABLE_VARS,VALUE).

element_matrix1(0,_50891,_50892,_50893,[]) :-
        !.

element_matrix1(I,MAX_J,INDEX_J,VALUES,[V_J|R]) :-
        I>0,
        element_matrix2(MAX_J,VALUES,TABLE_VALS,REST_VALUES),
        element(INDEX_J,TABLE_VALS,V_J),
        I1 is I-1,
        element_matrix1(I1,MAX_J,INDEX_J,REST_VALUES,R).

element_matrix2(0,VALUES,[],VALUES) :-
        !.

element_matrix2(J,[V|R],[V|S],REST_VALUES) :-
        J>0,
        J1 is J-1,
        element_matrix2(J1,R,S,REST_VALUES).

element_matrix_a(FLAG,MAX_I,MAX_J,INDEX_I,INDEX_J,MATRIX,VALUE) :-
        check_type(int,MAX_I),
        MAX_I>=1,
        check_type(int,MAX_J),
        MAX_J>=1,
        check_type(dvar,INDEX_I),
```

```
        INDEX_I#>=1,
        INDEX_I#=<MAX_I,
        check_type(dvar,INDEX_J),
        INDEX_J#>=1,
        INDEX_J#=<MAX_J,
        collection(MATRIX,[int(1,MAX_I),int(1,MAX_J),int]),
        length(MATRIX,N),
        N is MAX_I*MAX_J,
        collection_increasing_seq(MATRIX,[1,2]),
        check_type(dvar,VALUE),
        element_matrix_signature(
            MATRIX,
            INDEX_I,
            INDEX_J,
            VALUE,
            SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _55011,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

element_matrix_signature([],_50888,_50889,_50890,[]).

element_matrix_signature(
    [[i-I,j-J,v-V]|Ms],
    INDEX_I,
    INDEX_J,
    VALUE,
    [S|Ss]) :-
        INDEX_I#=I#/\INDEX_J#=J#/\VALUE#=V#<=>S,
        element_matrix_signature(Ms,INDEX_I,INDEX_J,VALUE,Ss).
```

B.145 element_product

◊ META-DATA:

```
ctr_date(element_product,['20051229','20060808']).

ctr_origin(
    element_product,
    \cite{OttossonThorsteinsson00},
    []).

ctr_synonyms(element_product,[element]).

ctr_arguments(
    element_product,
    ['Y'-dvar,'TABLE'-collection(value-int),'X'-dvar,'Z'-dvar]).

ctr_restrictions(
    element_product,
    ['Y'>=1,
     'Y'=<size('TABLE'),
     'X'>=0,
     'Z'>=0,
     required('TABLE',value),
     'TABLE'^value>=0]).

ctr_example(
    element_product,
    element_product(
        3,
        [[value-6],[value-9],[value-2],[value-9]],
        5,
        10)).

ctr_typical(
    element_product,
    ['X'>0,
     'Z'>0,
     size('TABLE')>1,
     range('TABLE'^value)>1,
     'TABLE'^value>0]).

ctr_derived_collections(
    element_product,
    [col('ITEM'-collection(y-dvar,x-dvar,z-dvar),
         [item(y-'Y',x-'X',z-'Z')])]).
```

```
ctr_graph(
    element_product,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^y=table^key,item^z=item^x*table^value],
    ['NARC'=1],
    []).

ctr_eval(element_product,[reformulation(element_product_r)]).

ctr_pure_functional_dependency(element_product,[]).

ctr_functional_dependency(element_product,4,[1,2,3]).

ctr_extensible(element_product,[],'TABLE',suffix).

element_product_r(Y,TABLE,X,Z) :-
        check_type(dvar,Y),
        collection(TABLE,[int_gteq(0)]),
        check_type(dvar,X),
        check_type(dvar,Z),
        length(TABLE,N),
        Y#>=1,
        Y#=<N,
        X#>=0,
        Z#>=0,
        get_attr1(TABLE,VALUES),
        element(Y,VALUES,VAL),
        Z#=VAL*X.
```

B.146 element_sparse

◇ **META-DATA:**

```
ctr_date(element_sparse,['20030820','20040530','20060808']).

ctr_origin(element_sparse,'\\index{CHIP|indexuse}CHIP',[]).

ctr_usual_name(element_sparse,element).

ctr_arguments(
    element_sparse,
    ['ITEM'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-int),
     'DEFAULT'-int]).

ctr_restrictions(
    element_sparse,
    [required('ITEM',[index,value]),
     'ITEM'^index>=1,
     size('ITEM')=1,
     size('TABLE')>0,
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     distinct('TABLE',index)]).

ctr_example(
    element_sparse,
    element_sparse(
        [[index-2,value-5]],
        [[index-1,value-6],
         [index-2,value-5],
         [index-4,value-2],
         [index-8,value-9]],
        5)).

ctr_typical(
    element_sparse,
    [size('TABLE')>1,range('TABLE'^value)>1]).

ctr_exchangeable(
    element_sparse,
    [items('TABLE',all),
     vals(
         ['ITEM'^value,'TABLE'^value,'DEFAULT'],
         int,
```

```
          =\=,
          all,
          dontcare)]).

ctr_derived_collections(
    element_sparse,
    [col('DEF'-collection(index-int,value-int),
         [item(index-0,value-'DEFAULT')]),
     col('TABLE_DEF'-collection(index-dvar,value-dvar),
         [item(index-'TABLE'^index,value-'TABLE'^value),
          item(index-'DEF'^index,value-'DEF'^value)])]).

ctr_graph(
    element_sparse,
    ['ITEM','TABLE_DEF'],
    2,
    ['PRODUCT'>>collection(item,table_def)],
    [item^value=table_def^value,
     item^index=table_def^index#\/table_def^index=0],
    ['NARC'>=1],
    []).

ctr_eval(
    element_sparse,
    [reformulation(element_sparse_r),
     automaton(element_sparse_a)]).

element_sparse_r(ITEM,TABLE,DEFAULT) :-
        length(ITEM,1),
        length(TABLE,N),
        N>0,
        collection(ITEM,[dvar_gteq(1),dvar]),
        collection(TABLE,[int_gteq(1),dvar]),
        check_type(int,DEFAULT),
        get_attr1(ITEM,[I]),
        get_attr2(ITEM,[V]),
        get_attr1(TABLE,INDEXES),
        get_attr2(TABLE,VALUES),
        all_different(INDEXES),
        element_sparse1(INDEXES,VALUES,I,V,DEFAULT,Term1,Term2),
        call(Term1#\/Term2).

element_sparse1([],[],_47118,V,DEFAULT,0,V#=DEFAULT).

element_sparse1(
    [IND|R],
```

```
    [VAL|S],
    I,
    V,
    DEFAULT,
    I#=IND#/\V#=VAL#\/T,
    I#\=IND#/\U) :-
        element_sparse1(R,S,I,V,DEFAULT,T,U).

element_sparse_a(FLAG,ITEM,TABLE,DEFAULT) :-
        length(ITEM,1),
        length(TABLE,N),
        N>0,
        collection(ITEM,[dvar_gteq(1),dvar]),
        collection(TABLE,[int_gteq(1),dvar]),
        check_type(int,DEFAULT),
        get_attr1(TABLE,INDEXES),
        all_different(INDEXES),
        ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
        element_sparse_signature(
            TABLE,
            SIGNATURE,
            ITEM_INDEX,
            ITEM_VALUE,
            DEFAULT),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _50308,
            SIGNATURE,
            [source(s),sink(d),sink(t)],
            [arc(s,0,s),
             arc(s,1,t),
             arc(s,2,d),
             arc(d,1,t),
             arc(d,2,d),
             arc(t,0,t),
             arc(t,1,t),
             arc(t,2,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2],AUTOMATON).

element_sparse_signature([],[],_47118,_47119,_47120).

element_sparse_signature(
```

```
[[index-TABLE_INDEX,value-TABLE_VALUE]|TABLEs],
[S|Ss],
ITEM_INDEX,
ITEM_VALUE,
DEFAULT) :-
    S in 0..2,
    ITEM_INDEX#\=TABLE_INDEX#/\ITEM_VALUE#\=DEFAULT#<=>S#=0,
    ITEM_INDEX#=TABLE_INDEX#/\ITEM_VALUE#=TABLE_VALUE#<=>
    S#=1,
    ITEM_INDEX#\=TABLE_INDEX#/\ITEM_VALUE#=DEFAULT#<=>S#=2,
    element_sparse_signature(
        TABLEs,
        Ss,
        ITEM_INDEX,
        ITEM_VALUE,
        DEFAULT).
```

B.147 elementn

◇ **META-DATA:**

```
ctr_date(elementn,['20061004']).

ctr_origin(elementn,'P. Flener',[]).

ctr_arguments(
    elementn,
    ['INDEX'-dvar,
     'TABLE'-collection(value-int),
     'ENTRIES'-collection(entry-dvar)]).

ctr_restrictions(
    elementn,
    ['INDEX'>=1,
     'INDEX'=<size('TABLE')-size('ENTRIES')+1,
     size('TABLE')>0,
     size('ENTRIES')>0,
     size('TABLE')>=size('ENTRIES'),
     required('TABLE',value),
     required('ENTRIES',entry)]).

ctr_example(
    elementn,
    elementn(
        3,
        [[value-6],[value-9],[value-2],[value-9]],
        [[entry-2],[entry-9]])).

ctr_typical(
    elementn,
    [size('TABLE')>1,range('TABLE'^value)>1,size('ENTRIES')>1]).

ctr_exchangeable(
    elementn,
    [vals(
         ['TABLE'^value,'ENTRIES'^entry],
         int,
         =\=,
         all,
         dontcare)]).

ctr_eval(
    elementn,
```

```
    [reformulation(elementn_r),automaton(elementn_a)]).

ctr_extensible(elementn,[],'TABLE',suffix).

elementn_r(INDEX,TABLE,ENTRIES) :-
        length(TABLE,N),
        length(ENTRIES,M),
        N>0,
        M>0,
        N>=M,
        NM is N-M+1,
        check_type(dvar(1,NM),INDEX),
        collection(TABLE,[int]),
        collection(ENTRIES,[dvar]),
        get_attr1(TABLE,TAB),
        get_attr1(ENTRIES,VALS),
        elementn1(VALS,0,INDEX,TAB).

elementn1([],_28400,_28401,_28402).

elementn1([V|R],K,INDEX,TAB) :-
        IND#=INDEX+K,
        element(IND,TAB,V),
        K1 is K+1,
        elementn1(R,K1,INDEX,TAB).

elementn_a(FLAG,INDEX,TABLE,ENTRIES) :-
        length(TABLE,T),
        length(ENTRIES,E),
        T>0,
        E>0,
        T>=E,
        TE is T-E+1,
        check_type(dvar(1,TE),INDEX),
        collection(TABLE,[int]),
        collection(ENTRIES,[dvar]),
        elementn_get_para(TABLE,Table),
        elementn_get_para(ENTRIES,Entries),
        elementn_gen_val(1,TE,LV),
        elementn_gen_arc(1,TE,E,LV,Table,Arcs),
        append([INDEX],Entries,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _31696,
            SIGNATURE,
```

```
            [source(s),sink(t)],
            Arcs,
            [],
            [],
            []),
        union_dom_list_int(SIGNATURE,ALPHABET),
        automaton_bool(FLAG,ALPHABET,AUTOMATON).

elementn_get_para([],[]).

elementn_get_para([[_28409-P]|R],[P|S]) :-
        elementn_get_para(R,S).

elementn_gen_val(I,I,[I]) :-
        !.

elementn_gen_val(I,J,[I|R]) :-
        I<J,
        I1 is I+1,
        elementn_gen_val(I1,J,R).

elementn_gen_arc(I,J,_28404,_28405,_28406,[]) :-
        I>J,
        !.

elementn_gen_arc(I,J,E,[I|S],[F|T],Arcs) :-
        I=<J,
        K is 1+E*(I-1),
        A0=[arc(s,I,K)],
        elementn_gen_arc1(1,E,K,[F|T],A1),
        I1 is I+1,
        elementn_gen_arc(I1,J,E,S,T,A),
        append(A0,A1,A2),
        append(A2,A,Arcs).

elementn_gen_arc1(J,E,K,[F|T],[arc(K,F,K1)|R]) :-
        J<E,
        !,
        K1 is K+1,
        J1 is J+1,
        elementn_gen_arc1(J1,E,K1,T,R).

elementn_gen_arc1(E,E,K,[F|_28409],[arc(K,F,t)]).
```

B.148 elements

◊ **META-DATA:**

```
ctr_date(elements,['20030820','20060808']).

ctr_origin(elements,'Derived from %c.',[element]).

ctr_arguments(
    elements,
    ['ITEMS'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-dvar)]).

ctr_restrictions(
    elements,
    [required('ITEMS',[index,value]),
     'ITEMS'^index>=1,
     'ITEMS'^index=<size('TABLE'),
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     distinct('TABLE',index)]).

ctr_example(
    elements,
    elements(
        [[index-4,value-9],[index-1,value-6]],
        [[index-1,value-6],
         [index-2,value-9],
         [index-3,value-2],
         [index-4,value-9]])).

ctr_typical(
    elements,
    [size('ITEMS')>1,
     range('ITEMS'^index)>1,
     size('TABLE')>1,
     range('TABLE'^value)>1]).

ctr_exchangeable(
    elements,
    [items('ITEMS',all),
     items('TABLE',all),
     vals(['ITEMS'^value,'TABLE'^value],int,=\=,all,dontcare)]).

ctr_graph(
```

```
    elements,
    ['ITEMS','TABLE'],
    2,
    ['PRODUCT'>>collection(items,table)],
    [items^index=table^index,items^value=table^value],
    ['NARC'=size('ITEMS')],
    []).

ctr_eval(elements,[reformulation(elements_r)]).

ctr_pure_functional_dependency(elements,[]).

ctr_functional_dependency(elements,1-2,[1-1,2]).

ctr_cond_imply(
    elements,
    bin_packing_capa,
    [distinct('ITEMS',index),'TABLE'^value>=0],
    [],
    ['TABLE','ITEMS']).

elements_r(ITEMS,TABLE) :-
        length(TABLE,N),
        collection(ITEMS,[dvar(1,N),dvar]),
        collection(TABLE,[int(1,N),dvar]),
        sort_collection(TABLE,index,SORTED_TABLE),
        get_attr1(SORTED_TABLE,INDEXES),
        increasing_values(INDEXES),
        get_attr2(SORTED_TABLE,VALUES),
        get_attr1(ITEMS,INDS),
        get_attr2(ITEMS,VALS),
        elements1(INDS,VALS,VALUES).

elements1([],[],_43167).

elements1([IND|R],[VAL|S],VALUES) :-
        element(IND,VALUES,VAL),
        elements1(R,S,VALUES).
```

B.149 elements_alldifferent

◇ **META-DATA:**

```
ctr_date(elements_alldifferent,['20030820','20060809']).

ctr_origin(
    elements_alldifferent,
    Derived from %c and %c.,
    [elements,alldifferent]).

ctr_synonyms(
    elements_alldifferent,
    [elements_alldiff,elements_alldistinct,permutation]).

ctr_arguments(
    elements_alldifferent,
    ['ITEMS'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-dvar)]).

ctr_restrictions(
    elements_alldifferent,
    [required('ITEMS',[index,value]),
     'ITEMS'^index>=1,
     'ITEMS'^index=<size('TABLE'),
     size('ITEMS')=size('TABLE'),
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     distinct('TABLE',index)]).

ctr_example(
    elements_alldifferent,
    elements_alldifferent(
        [[index-2,value-9],
         [index-1,value-6],
         [index-4,value-9],
         [index-3,value-2]],
        [[index-1,value-6],
         [index-2,value-9],
         [index-3,value-2],
         [index-4,value-9]])).

ctr_typical(
    elements_alldifferent,
    [size('ITEMS')>1,
```

```
     range('ITEMS'^value)>1,
     size('TABLE')>1,
     range('TABLE'^value)>1]).

ctr_exchangeable(
    elements_alldifferent,
    [args([['ITEMS','TABLE']]),
     items('ITEMS',all),
     items('TABLE',all),
     vals(['ITEMS'^value,'TABLE'^value],int,=\=,all,dontcare)]).

ctr_graph(
    elements_alldifferent,
    ['ITEMS','TABLE'],
    2,
    ['PRODUCT'>>collection(items,table)],
    [items^index=table^index,items^value=table^value],
    ['NVERTEX'=size('ITEMS')+size('TABLE')],
    []).

ctr_eval(
    elements_alldifferent,
    [reformulation(elements_alldifferent_r)]).

ctr_functional_dependency(elements_alldifferent,1-2,[1-1,2]).

ctr_cond_imply(
    elements_alldifferent,
    bin_packing_capa,
    ['TABLE'^value>=0],
    [],
    ['TABLE','ITEMS']).

elements_alldifferent_r(ITEMS,TABLE) :-
        length(TABLE,N),
        collection(ITEMS,[dvar(1,N),dvar]),
        collection(TABLE,[int(1,N),dvar]),
        get_attr1(TABLE,INDEXES),
        all_different(INDEXES),
        sort(TABLE,STAB),
        get_attr2(STAB,VALUES),
        get_attr1(ITEMS,INDS),
        get_attr2(ITEMS,VALS),
        all_different(INDS),
        elements_alldifferent1(INDS,VALS,VALUES).
```

```
elements_alldifferent1([],[],_48954).

elements_alldifferent1([IND|R],[VAL|S],VALUES) :-
        element(IND,VALUES,VAL),
        elements_alldifferent1(R,S,VALUES).
```

B.150 elements_sparse

◇ META-DATA:

```
ctr_date(elements_sparse,['20030820','20060809']).

ctr_origin(elements_sparse,'Derived from %c.',[element_sparse]).

ctr_arguments(
    elements_sparse,
    ['ITEMS'-collection(index-dvar,value-dvar),
     'TABLE'-collection(index-int,value-int),
     'DEFAULT'-int]).

ctr_restrictions(
    elements_sparse,
    [required('ITEMS',[index,value]),
     'ITEMS'^index>=1,
     required('TABLE',[index,value]),
     'TABLE'^index>=1,
     distinct('TABLE',index)]).

ctr_example(
    elements_sparse,
    elements_sparse(
        [[index-8,value-9],
         [index-3,value-5],
         [index-2,value-5]],
        [[index-1,value-6],
         [index-2,value-5],
         [index-4,value-2],
         [index-8,value-9]],
        5)).

ctr_typical(
    elements_sparse,
    [size('ITEMS')>1,
     range('ITEMS'^value)>1,
     size('TABLE')>1,
     range('TABLE'^value)>1]).

ctr_exchangeable(
    elements_sparse,
    [items('ITEMS',all),
     items('TABLE',all),
     vals(
```

```
          ['ITEMS'^value,'TABLE'^value,'DEFAULT'],
          int,
          =\=,
          all,
          dontcare)]).

ctr_derived_collections(
    elements_sparse,
    [col('DEF'-collection(index-int,value-int),
         [item(index-0,value-'DEFAULT')]),
     col('TABLE_DEF'-collection(index-dvar,value-dvar),
         [item(index-'TABLE'^index,value-'TABLE'^index),
          item(index-'DEF'^index,value-'DEF'^value)])]).

ctr_graph(
    elements_sparse,
    ['ITEMS','TABLE_DEF'],
    2,
    ['PRODUCT'>>collection(items,table_def)],
    [items^value=table_def^value,
     items^index=table_def^index#\/table_def^index=0],
    ['NSOURCE'=size('ITEMS')],
    []).

ctr_eval(elements_sparse,[reformulation(elements_sparse_r)]).

elements_sparse_r(ITEMS,TABLE,DEFAULT) :-
        collection(ITEMS,[dvar_gteq(1),dvar]),
        collection(TABLE,[int_gteq(1),dvar]),
        check_type(int,DEFAULT),
        get_attr1(ITEMS,IS),
        get_attr2(ITEMS,VS),
        get_attr1(TABLE,INDEXES),
        get_attr2(TABLE,VALUES),
        all_different(INDEXES),
        elements_sparse1(IS,VS,INDEXES,VALUES,DEFAULT).

elements_sparse1([],[],_47321,_47322,_47323).

elements_sparse1([I|R],[V|S],INDEXES,VALUES,DEFAULT) :-
        elements_sparse2(
            INDEXES,
            VALUES,
            I,
            V,
            DEFAULT,
```

```
            Term1,
            Term2),
        call(Term1#\/Term2),
        elements_sparse1(R,S,INDEXES,VALUES,DEFAULT).

elements_sparse2([],[],_47321,V,DEFAULT,0,V#=DEFAULT).

elements_sparse2(
    [IND|R],
    [VAL|S],
    I,
    V,
    DEFAULT,
    I#=IND#/\V#=VAL#\/T,
    I#\=IND#/\U) :-
        elements_sparse2(R,S,I,V,DEFAULT,T,U).
```

B.151 eq

◇ **META-DATA:**

```
ctr_predefined(eq).

ctr_date(eq,['20070821']).

ctr_origin(eq,'Arithmetic.',[]).

ctr_synonyms(eq,[xeqy]).

ctr_arguments(eq,['VAR1'-dvar,'VAR2'-dvar]).

ctr_restrictions(eq,[]).

ctr_example(eq,eq(8,8)).

ctr_exchangeable(
    eq,
    [args([['VAR1','VAR2']]),
     vals(['VAR1','VAR2'],int,=\=,all,dontcare)]).

ctr_eval(eq,[checker(eq_c),builtin(eq_b)]).

ctr_pure_functional_dependency(eq,[]).

ctr_functional_dependency(eq,2,[1]).

ctr_functional_dependency(eq,1,[2]).

eq_c(VAR1,VAR2) :-
        check_type(int,VAR1),
        check_type(int,VAR2),
        VAR1=VAR2.

eq_b(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#=VAR2.
```

B.152 eq_cst

◇ META-DATA:

```
ctr_predefined(eq_cst).

ctr_date(eq_cst,['20090923']).

ctr_origin(eq_cst,'Arithmetic.',[]).

ctr_arguments(eq_cst,['VAR1'-dvar,'VAR2'-dvar,'CST2'-int]).

ctr_example(eq_cst,eq_cst(8,2,6)).

ctr_typical(eq_cst,['CST2'=\=0]).

ctr_exchangeable(
    eq_cst,
    [args([['VAR1'],['VAR2','CST2']]),
     translate(['VAR1','VAR2']),
     translate(['VAR1','CST2'])]).

ctr_eval(eq_cst,[checker(eq_cst_c),builtin(eq_cst_b)]).

ctr_pure_functional_dependency(eq_cst,[]).

ctr_functional_dependency(eq_cst,1,[2,3]).

ctr_functional_dependency(eq_cst,2,[1,3]).

ctr_functional_dependency(eq_cst,3,[1,2]).

eq_cst_c(VAR1,VAR2,CST2) :-
        check_type(int,VAR1),
        check_type(int,VAR2),
        check_type(dvar,CST2),
        CST2 is VAR1-VAR2.

eq_cst_b(VAR1,VAR2,CST2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        check_type(int,CST2),
        VAR1#=VAR2+CST2.
```

B.153 eq_set

◇ **META-DATA:**

```
ctr_predefined(eq_set).

ctr_date(eq_set,['20030820']).

ctr_origin(
    eq_set,
    Used for defining %c.,
    [alldifferent_between_sets]).

ctr_arguments(eq_set,['SET1'-svar,'SET2'-svar]).

ctr_example(eq_set,eq_set({3,5},{3,5})).

ctr_exchangeable(
    eq_set,
    [args([['SET1','SET2']]),
     vals(['SET1','SET2'],int,=\=,all,dontcare)]).
```

B.154 equal_sboxes

♢ META-DATA:

```
ctr_date(equal_sboxes,['20070622','20090725']).

ctr_origin(
    equal_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(equal_sboxes,[equal]).

ctr_types(
    equal_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    equal_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    equal_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    equal_sboxes,
    equal_sboxes(
        2,
        {0,1},
        [[oid-1,sid-2,x-[[v-4],[v-1]]],
         [oid-2,sid-2,x-[[v-4],[v-1]]],
         [oid-3,sid-2,x-[[v-4],[v-1]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-1],[v-2]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-1],[v-1]]],
         [sid-2,t-[[v-1],[v-0]],l-[[v-1],[v-3]]],
         [sid-2,t-[[v-0],[v-2]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
         [sid-3,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-4,t-[[v-0],[v-0]],l-[[v-1],[v-1]]]])).

ctr_typical(equal_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    equal_sboxes,
    [items('OBJECTS',all),
     items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all)]).

ctr_eval(equal_sboxes,[logic(equal_sboxes_g)]).

ctr_logic(
    equal_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (equal_sboxes(Dims,O1,S1,O2,S2)--->
      forall(
          D,
          Dims,
          origin(O1,S1,D)#=origin(O2,S2,D)#/\
          end(O1,S1,D)#=end(O2,S2,D))),
     (equal_objects(Dims,O1,O2)--->
```

```
      forall(
          S1,
          sboxes([O1^sid]),
          exists(
              S2,
              sboxes([O2^sid]),
              equal_sboxes(Dims,O1,S1,O2,S2)))),
     (all_equal(Dims,OIDS)--->
      forall(
          O1,
          objects(OIDS),
          forall(
              O2,
              objects(OIDS),
              O1^oid#=O2^oid-1#=>equal_objects(Dims,O1,O2)))),
     all_equal(DIMENSIONS,OIDS)]).

ctr_contractible(equal_sboxes,[],'OBJECTS',suffix).

ctr_application(equal_sboxes,[3]).

equal_sboxes_g(K,_40037,[],_40039) :-
        !,
        check_type(int_gteq(1),K).

equal_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        collection_increasing_seq(OBJECTS,[1]),
        geost1(OIDS,SIDS,XS,Objects),
        geost2(SIDES,TS,TL,Sboxes),
        geost_dims(1,K,DIMENSIONS),
        ctr_logic(equal_sboxes,[DIMENSIONS,OIDS],Rules),
```

```
geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.155 equilibrium

◇ META-DATA:

```
ctr_predefined(equilibrium).

ctr_date(equilibrium,['20130714']).

ctr_origin(
    equilibrium,
    Inspired by the Irish Collegiate Programming Competition 2012 (equilibri
    []).

ctr_synonyms(equilibrium,[balanced]).

ctr_arguments(
    equilibrium,
    ['VARIABLES'-collection(var-dvar),
     'INDEX1'-dvar,
     'INDEX2'-dvar,
     'EPSILON'-int,
     'COEF1'-int,
     'COEF2'-int,
     'TOLERANCE'-int,
     'CTR'-atom]).

ctr_restrictions(
    equilibrium,
    [size('VARIABLES')>=1,
     'INDEX1'>=1,
     'INDEX1'=<size('VARIABLES'),
     'INDEX2'>=1,
     'INDEX2'=<size('VARIABLES'),
     'INDEX1'=<'INDEX2',
     'EPSILON'>=0,
     'EPSILON'=<2,
     'EPSILON'='INDEX2'-'INDEX1',
     'COEF1'=\=0,
     'COEF2'=\=0,
     'TOLERANCE'>=0,
     in_list(
         CTR,
         [among_diff_0,
          and,
          change,
          deepest_valley,
```

```
          highest_peak,
          increasing_nvalue,
          inflexion,
          longest_change,
          longest_decreasing_sequence,
          longest_increasing_sequence,
          max_decreasing_slope,
          max_increasing_slope,
          min_decreasing_slope,
          min_increasing_slope,
          min_width_peak,
          min_width_valley,
          peak,
          sum_ctr,
          valley]])).

ctr_example(
    equilibrium,
    [equilibrium(
         [[var-4],[var-4],[var-3],[var-6],[var-2]],
         2,
         4,
         2,
         1,
         1,
         0,
         sum_ctr),
     equilibrium(
         [[var- -2],
          [var-5],
          [var- -2],
          [var-6],
          [var- -1],
          [var-0],
          [var- -3],
          [var-5],
          [var- -7],
          [var-6],
          [var- -1],
          [var-7],
          [var-0]],
         5,
         5,
         0,
         1,
         1,
```

```
        0,
        sum_ctr),
    equilibrium(
        [[var- -2],
         [var-5],
         [var- -2],
         [var-6],
         [var- -1],
         [var-0],
         [var- -3],
         [var-5],
         [var- -7],
         [var-6],
         [var- -1],
         [var-7],
         [var-0]],
        11,
        11,
        0,
        1,
        1,
        0,
        sum_ctr),
    equilibrium(
        [[var-0],
         [var-3],
         [var-2],
         [var-6],
         [var-2],
         [var-2],
         [var-5],
         [var-8],
         [var-7],
         [var-6],
         [var-7],
         [var-3]],
        5,
        7,
        2,
        1,
        1,
        0,
        peak),
    equilibrium(
        [[var-0],
         [var-5],
```

```
          [var-3],
          [var-8],
          [var-2],
          [var-2],
          [var-5],
          [var-5],
          [var-8],
          [var-7],
          [var-2],
          [var-7],
          [var-3]],
         7,
         7,
         0,
         1,
         1,
         0,
         change)]).

ctr_typical(
    equilibrium,
    [size('VARIABLES')>2,
     'INDEX1'>1,
     'INDEX1'<size('VARIABLES'),
     'INDEX2'>1,
     'INDEX2'<size('VARIABLES'),
     'COEF1'=1,
     'COEF2'=1,
     'EPSILON'=1,
     'TOLERANCE'=0]).

ctr_eval(
    equilibrium,
    [checker(equilibrium_c),
     reformulation(equilibrium_r),
     ground_typical(equilibrium_t)]).

equilibrium_t(
    VARIABLES,
    _30267,
    _30314,
    _30361,
    _30408,
    _30455,
    _30502,
    CTR) :-
```

```
(   CTR=among_diff_0 ->
    among_diff_0(C,VARIABLES),
    C#>0
;   CTR=change ->
    change(C,VARIABLES,=\=),
    C#>0
;   CTR=increasing_nvalue ->
    increasing_nvalue(C,VARIABLES),
    C#>0
;   CTR=inflexion ->
    inflexion(C,VARIABLES),
    C#>0
;   CTR=longest_change ->
    longest_change(C,VARIABLES,=\=),
    C#>0
;   CTR=longest_decreasing_sequence ->
    longest_decreasing_sequence(C,VARIABLES),
    C#>1
;   CTR=longest_increasing_sequence ->
    longest_increasing_sequence(C,VARIABLES),
    C#>1
;   CTR=max_decreasing_slope ->
    max_decreasing_slope(C,VARIABLES),
    C#>1
;   CTR=max_increasing_slope ->
    max_increasing_slope(C,VARIABLES),
    C#>1
;   CTR=min_decreasing_slope ->
    min_decreasing_slope(C,VARIABLES),
    C#>1
;   CTR=min_increasing_slope ->
    min_increasing_slope(C,VARIABLES),
    C#>1
;   CTR=min_width_peak ->
    min_width_peak(C,VARIABLES),
    C#>0
;   CTR=min_width_valley ->
    min_width_valley(C,VARIABLES),
    C#>0
;   CTR=peak ->
    peak(C,VARIABLES),
    C#>0
;   CTR=sum_ctr ->
    sum_ctr(C,VARIABLES),
    C#>0
;   CTR=valley ->
```

```
            valley(C,VARIABLES),
            C#>0
        ;   true
        ).

equilibrium_c(
    VARIABLES,
    INDEX1,
    INDEX2,
    EPSILON,
    COEF1,
    COEF2,
    TOLERANCE,
    CTR) :-
        integer(INDEX1),
        integer(INDEX2),
        !,
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        N>=1,
        INDEX1>=1,
        INDEX1=<N,
        INDEX2>=1,
        INDEX2=<N,
        INDEX1=<INDEX2,
        EPSILON>=0,
        EPSILON=<2,
        EPSILON is INDEX2-INDEX1,
        TOLERANCE>=0,
        memberchk(
            CTR,
            [among_diff_0,
             and,
             change,
             deepest_valley,
             highest_peak,
             increasing_nvalue,
             inflexion,
             longest_change,
             longest_decreasing_sequence,
             longest_increasing_sequence,
             max_decreasing_slope,
             max_increasing_slope,
             min_decreasing_slope,
             min_increasing_slope,
```

```
     min_width_peak,
     min_width_valley,
     peak,
     sum_ctr,
     valley]),
(   CTR=among_diff_0 ->
    prefix_length(VARS,PVARS,INDEX1),
    among_diff_0_c(PVARS,0,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    among_diff_0_c(SVARS,0,C2)
;   CTR=and ->
    prefix_length(VARIABLES,PVARS,INDEX1),
    C1 in 0..1,
    and_c(C1,PVARS),
    PREF is INDEX2-1,
    append_length(SVARS,VARIABLES,PREF),
    C2 in 0..1,
    and_c(C2,SVARS)
;   CTR=change ->
    prefix_length(VARS,PVARS,INDEX1),
    change_neq_c(PVARS,0,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    change_neq_c(SVARS,0,C2)
;   CTR=deepest_valley ->
    MAXINT=1000000,
    prefix_length(VARS,PVARS,INDEX1),
    deepest_valley_c(PVARS,s,MAXINT,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    deepest_valley_c(SVARS,s,MAXINT,C2)
;   CTR=highest_peak ->
    MININT= -1000000,
    prefix_length(VARS,PVARS,INDEX1),
    highest_peak_c(PVARS,s,MININT,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    highest_peak_c(SVARS,s,MININT,C2)
;   CTR=increasing_nvalue ->
    prefix_length(VARS,PVARS,INDEX1),
    PVARS=[PVAR|RPVARS],
    increasing_nvalue_c(RPVARS,PVAR,1,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    SVARS=[SVAR|RSVARS],
```

```
    increasing_nvalue_c(RSVARS,SVAR,1,C2)
;   CTR=inflexion ->
    prefix_length(VARS,PVARS,INDEX1),
    inflexion_c(PVARS,s,0,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    inflexion_c(SVARS,s,0,C2)
;   CTR=longest_change ->
    prefix_length(VARS,PVARS,INDEX1),
    longest_change_neq_c(PVARS,0,1,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    longest_change_neq_c(SVARS,0,1,C2)
;   CTR=longest_decreasing_sequence ->
    prefix_length(VARS,PVARS,INDEX1),
    longest_decreasing_sequence_c(PVARS,s,0,0,0,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    longest_decreasing_sequence_c(SVARS,s,0,0,0,C2)
;   CTR=longest_increasing_sequence ->
    prefix_length(VARS,PVARS,INDEX1),
    longest_increasing_sequence_c(PVARS,s,0,0,0,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    longest_increasing_sequence_c(SVARS,s,0,0,0,C2)
;   CTR=max_decreasing_slope ->
    prefix_length(VARS,PVARS,INDEX1),
    max_decreasing_slope_c1(PVARS,0,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    max_decreasing_slope_c1(SVARS,0,C2)
;   CTR=max_increasing_slope ->
    prefix_length(VARS,PVARS,INDEX1),
    max_increasing_slope_c1(PVARS,0,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    max_increasing_slope_c1(SVARS,0,C2)
;   CTR=min_decreasing_slope ->
    prefix_length(VARS,PVARS,INDEX1),
    min_decreasing_slope_c1(PVARS,0,C1),
    PREF is INDEX2-1,
    append_length(SVARS,VARS,PREF),
    min_decreasing_slope_c1(SVARS,0,C2)
;   CTR=min_increasing_slope ->
    prefix_length(VARS,PVARS,INDEX1),
    min_increasing_slope_c1(PVARS,0,C1),
```

```
            PREF is INDEX2-1,
            append_length(SVARS,VARS,PREF),
            min_increasing_slope_c1(SVARS,0,C2)
        ;   CTR=min_width_peak ->
            prefix_length(VARS,PVARS,INDEX1),
            min_width_peak_c(PVARS,s,1,0,0,0,INDEX1,C1),
            PREF is INDEX2-1,
            append_length(SVARS,VARS,PREF),
            LEN2 is N-INDEX1+1,
            min_width_peak_c(SVARS,s,1,0,0,0,LEN2,C2)
        ;   CTR=min_width_valley ->
            prefix_length(VARS,PVARS,INDEX1),
            min_width_valley_c(PVARS,s,1,0,0,0,INDEX1,C1),
            PREF is INDEX2-1,
            append_length(SVARS,VARS,PREF),
            LEN2 is N-INDEX1+1,
            min_width_valley_c(SVARS,s,1,0,0,0,LEN2,C2)
        ;   CTR=peak ->
            prefix_length(VARS,PVARS,INDEX1),
            peak_c(PVARS,s,0,C1),
            PREF is INDEX2-1,
            append_length(SVARS,VARS,PREF),
            peak_c(SVARS,s,0,C2)
        ;   CTR=sum_ctr ->
            sum_c(INDEX1,0,VARS,C1),
            PREF is INDEX2-1,
            append_length(SVARS,VARS,PREF),
            get_sum(SVARS,C2)
        ;   CTR=valley ->
            prefix_length(VARS,PVARS,INDEX1),
            valley_c(PVARS,s,0,C1),
            PREF is INDEX2-1,
            append_length(SVARS,VARS,PREF),
            valley_c(SVARS,s,0,C2)
        ),
        CC1 is COEF1*C1,
        CC2 is COEF2*C2,
        DIF is abs(CC1-CC2),
        DIF=<TOLERANCE.

equilibrium_c(
    VARIABLES,
    INDEX1,
    INDEX2,
    EPSILON,
    COEF1,
```

```
COEF2,
TOLERANCE,
CTR) :-
    collection(VARIABLES,[int]),
    get_attr1(VARIABLES,VARS),
    length(VARS,N),
    N>=1,
    EPSILON>=0,
    EPSILON=<2,
    TOLERANCE>=0,
    check_type(dvar(1,N),INDEX1),
    check_type(dvar(1,N),INDEX2),
    memberchk(
        CTR,
        [among_diff_0,
         and,
         change,
         deepest_valley,
         highest_peak,
         increasing_nvalue,
         inflexion,
         longest_change,
         longest_decreasing_sequence,
         longest_increasing_sequence,
         max_decreasing_slope,
         max_increasing_slope,
         min_decreasing_slope,
         min_increasing_slope,
         min_width_peak,
         min_width_valley,
         peak,
         sum_ctr,
         valley]),
    (   CTR=among_diff_0 ->
        equilibrium_among_diff_0(VARS,PREFIX,SUFFIX)
    ;   CTR=and ->
        equilibrium_and(VARS,PREFIX,SUFFIX)
    ;   CTR=change ->
        equilibrium_change(VARS,PREFIX,SUFFIX)
    ;   CTR=deepest_valley ->
        equilibrium_deepest_valley(VARS,PREFIX,SUFFIX)
    ;   CTR=highest_peak ->
        equilibrium_highest_peak(VARS,PREFIX,SUFFIX)
    ;   CTR=increasing_nvalue ->
        equilibrium_increasing_nvalue(VARS,PREFIX,SUFFIX)
    ;   CTR=inflexion ->
```

```
    equilibrium_inflexion(VARS,PREFIX,SUFFIX)
;   CTR=longest_change ->
    equilibrium_longest_change(VARS,PREFIX,SUFFIX)
;   CTR=longest_decreasing_sequence ->
    equilibrium_longest_decreasing_sequence(
        VARS,
        PREFIX,
        SUFFIX)
;   CTR=longest_increasing_sequence ->
    equilibrium_longest_increasing_sequence(
        VARS,
        PREFIX,
        SUFFIX)
;   CTR=max_decreasing_slope ->
    equilibrium_max_decreasing_slope(
        VARS,
        PREFIX,
        SUFFIX)
;   CTR=max_increasing_slope ->
    equilibrium_max_increasing_slope(
        VARS,
        PREFIX,
        SUFFIX)
;   CTR=min_decreasing_slope ->
    equilibrium_min_decreasing_slope(
        VARS,
        PREFIX,
        SUFFIX)
;   CTR=min_increasing_slope ->
    equilibrium_min_increasing_slope(
        VARS,
        PREFIX,
        SUFFIX)
;   CTR=min_width_peak ->
    equilibrium_min_width_peak(VARS,N,PREFIX,SUFFIX)
;   CTR=min_width_valley ->
    equilibrium_min_width_valley(VARS,N,PREFIX,SUFFIX)
;   CTR=peak ->
    equilibrium_peak(VARS,PREFIX,SUFFIX)
;   CTR=sum_ctr ->
    equilibrium_sum_ctr(VARS,1,PREFIX,SUFFIX)
;   CTR=valley ->
    equilibrium_valley(VARS,PREFIX,SUFFIX)
),
write(prefix(PREFIX)),
nl,
```

```
        write(suffix(SUFFIX)),
        nl,
        append_length(SUFFIX_INDEX2,SUFFIX,EPSILON),
        L2 is 1+EPSILON,
        equilibrium_search_sols(
            1,
            L2,
            PREFIX,
            SUFFIX_INDEX2,
            COEF1,
            COEF2,
            TOLERANCE,
            LIM1,
            LIM2),
        (   LIM1=[_30128|_30129] ->
            list_to_fdset(LIM1,SET1),
            Lim1 in_set SET1,
            Lim1=INDEX1,
            list_to_fdset(LIM2,SET2),
            Lim2 in_set SET2,
            Lim2=INDEX2,
            EPSILON#=INDEX2-INDEX1
        ;   fail
        ).

equilibrium_among_diff_0(VARS,PREFIX,SUFFIX) :-
        among_diff_0_counters_check(VARS,0,PREFIX),
        reverse(VARS,RVARS),
        among_diff_0_counters_check(RVARS,0,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_and(VARS,PREFIX,SUFFIX) :-
        and_counters_check(VARS,init,PREFIX),
        reverse(VARS,RVARS),
        and_counters_check(RVARS,init,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_change(VARS,PREFIX,SUFFIX) :-
        change_neq_counters_check(VARS,0,PREFIX),
        reverse(VARS,RVARS),
        change_neq_counters_check(RVARS,0,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_deepest_valley(VARS,PREFIX,SUFFIX) :-
        MAXINT=1000000,
        deepest_valley_counters_check(VARS,init,MAXINT,PREFIX),
```

```
        reverse(VARS,RVARS),
        deepest_valley_counters_check(RVARS,init,MAXINT,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_highest_peak(VARS,PREFIX,SUFFIX) :-
        MININT= -1000000,
        highest_peak_counters_check(VARS,init,MININT,PREFIX),
        reverse(VARS,RVARS),
        highest_peak_counters_check(RVARS,init,MININT,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_increasing_nvalue(VARS,PREFIX,SUFFIX) :-
        VARS=[V|R],
        increasing_nvalue_counters_check(R,V,1,RPREFIX),
        PREFIX=[1|RPREFIX],
        reverse(VARS,RVARS),
        RVARS=[U|S],
        decreasing_nvalue_counters_check(S,U,1,COUNTS),
        reverse([1|COUNTS],SUFFIX).

equilibrium_inflexion(VARS,PREFIX,SUFFIX) :-
        inflexion_counters_check(init,VARS,0,PREFIX),
        reverse(VARS,RVARS),
        inflexion_counters_check(init,RVARS,0,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_longest_change(VARS,PREFIX,SUFFIX) :-
        longest_change_neq_counters_check(VARS,0,1,PREFIX),
        reverse(VARS,RVARS),
        longest_change_neq_counters_check(RVARS,0,1,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_longest_decreasing_sequence(VARS,PREFIX,SUFFIX) :-
        longest_decreasing_sequence_counters_check(
            VARS,
            s,
            0,
            0,
            0,
            PREFIX),
        reverse(VARS,RVARS),
        longest_increasing_sequence_counters_check(
            RVARS,
            s,
            0,
            0,
```

```
            0,
            COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_longest_increasing_sequence(VARS,PREFIX,SUFFIX) :-
        longest_increasing_sequence_counters_check(
            VARS,
            s,
            0,
            0,
            0,
            PREFIX),
        reverse(VARS,RVARS),
        longest_decreasing_sequence_counters_check(
            RVARS,
            s,
            0,
            0,
            0,
            COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_max_decreasing_slope(VARS,PREFIX,SUFFIX) :-
        max_decreasing_slope_counters_check(VARS,PREFIX),
        reverse(VARS,RVARS),
        max_increasing_slope_counters_check(RVARS,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_max_increasing_slope(VARS,PREFIX,SUFFIX) :-
        max_increasing_slope_counters_check(VARS,PREFIX),
        reverse(VARS,RVARS),
        max_decreasing_slope_counters_check(RVARS,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_min_decreasing_slope(VARS,PREFIX,SUFFIX) :-
        min_decreasing_slope_counters_check(VARS,PREFIX),
        reverse(VARS,RVARS),
        min_increasing_slope_counters_check(RVARS,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_min_increasing_slope(VARS,PREFIX,SUFFIX) :-
        min_increasing_slope_counters_check(VARS,PREFIX),
        reverse(VARS,RVARS),
        min_decreasing_slope_counters_check(RVARS,COUNTS),
        reverse(COUNTS,SUFFIX).
```

```
equilibrium_min_width_peak(VARS,N,PREFIX,SUFFIX) :-
        min_width_peak_counters_check(VARS,N,PREFIX),
        reverse(VARS,RVARS),
        min_width_peak_counters_check(RVARS,N,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_min_width_valley(VARS,N,PREFIX,SUFFIX) :-
        min_width_valley_counters_check(VARS,N,PREFIX),
        reverse(VARS,RVARS),
        min_width_valley_counters_check(RVARS,N,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_peak(VARS,PREFIX,SUFFIX) :-
        peak_counters_check(VARS,init,0,PREFIX),
        reverse(VARS,RVARS),
        peak_counters_check(RVARS,init,0,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_sum_ctr(VARS,1,PREFIX,SUFFIX) :-
        sum_counters_check(VARS,0,PREFIX),
        reverse(VARS,RVARS),
        sum_counters_check(RVARS,0,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_sum_ctr(VARS,0,PREFIX,SUFFIX) :-
        sum_counters_ref(VARS,0,PREFIX),
        reverse(VARS,RVARS),
        sum_counters_ref(RVARS,0,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_valley(VARS,PREFIX,SUFFIX) :-
        valley_counters_check(VARS,init,0,PREFIX),
        reverse(VARS,RVARS),
        valley_counters_check(RVARS,init,0,COUNTS),
        reverse(COUNTS,SUFFIX).

equilibrium_search_sols(
    _29921,
    _29968,
    _30015,
    [],
    _30107,
    _30154,
    _30201,
    [],
    []) :-
```

```
        !.

equilibrium_search_sols(
    INDEX1,
    INDEX2,
    [C1|RC1],
    [C2|RC2],
    COEF1,
    COEF2,
    TOLERANCE,
    [INDEX1|RL1],
    [INDEX2|RL2]) :-
        integer(C1),
        integer(C2),
        CC1 is COEF1*C1,
        CC2 is COEF2*C2,
        DIF is abs(CC1-CC2),
        DIF=<TOLERANCE,
        !,
        NEXT_INDEX1 is INDEX1+1,
        NEXT_INDEX2 is INDEX2+1,
        equilibrium_search_sols(
            NEXT_INDEX1,
            NEXT_INDEX2,
            RC1,
            RC2,
            COEF1,
            COEF2,
            TOLERANCE,
            RL1,
            RL2).

equilibrium_search_sols(
    INDEX1,
    INDEX2,
    [_29687|RC1],
    [_29691|RC2],
    COEF1,
    COEF2,
    TOLERANCE,
    RL1,
    RL2) :-
        NEXT_INDEX1 is INDEX1+1,
        NEXT_INDEX2 is INDEX2+1,
        equilibrium_search_sols(
            NEXT_INDEX1,
```

```
            NEXT_INDEX2,
            RC1,
            RC2,
            COEF1,
            COEF2,
            TOLERANCE,
            RL1,
            RL2).

equilibrium_expose_prefix_suffix_counters(
    VARIABLES,
    EPSILON,
    CTR,
    PREFIX,
    SUFFIX_INDEX2) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        N>=1,
        EPSILON>=0,
        EPSILON=<2,
        memberchk(
            CTR,
            [among_diff_0,
             and,
             change,
             deepest_valley,
             highest_peak,
             increasing_nvalue,
             inflexion,
             longest_change,
             longest_decreasing_sequence,
             longest_increasing_sequence,
             max_decreasing_slope,
             max_increasing_slope,
             min_decreasing_slope,
             min_increasing_slope,
             min_width_peak,
             min_width_valley,
             peak,
             sum_ctr,
             valley]),
        (   CTR=among_diff_0 ->
            equilibrium_among_diff_0(VARS,PREFIX,SUFFIX)
        ;   CTR=and ->
            equilibrium_and(VARS,PREFIX,SUFFIX)
```

```
;    CTR=change ->
     equilibrium_change(VARS,PREFIX,SUFFIX)
;    CTR=deepest_valley ->
     equilibrium_deepest_valley(VARS,PREFIX,SUFFIX)
;    CTR=highest_peak ->
     equilibrium_highest_peak(VARS,PREFIX,SUFFIX)
;    CTR=increasing_nvalue ->
     equilibrium_increasing_nvalue(VARS,PREFIX,SUFFIX)
;    CTR=inflexion ->
     equilibrium_inflexion(VARS,PREFIX,SUFFIX)
;    CTR=longest_change ->
     equilibrium_longest_change(VARS,PREFIX,SUFFIX)
;    CTR=longest_decreasing_sequence ->
     equilibrium_longest_decreasing_sequence(
         VARS,
         PREFIX,
         SUFFIX)
;    CTR=longest_increasing_sequence ->
     equilibrium_longest_increasing_sequence(
         VARS,
         PREFIX,
         SUFFIX)
;    CTR=max_decreasing_slope ->
     equilibrium_max_decreasing_slope(
         VARS,
         PREFIX,
         SUFFIX)
;    CTR=max_increasing_slope ->
     equilibrium_max_increasing_slope(
         VARS,
         PREFIX,
         SUFFIX)
;    CTR=min_decreasing_slope ->
     equilibrium_min_decreasing_slope(
         VARS,
         PREFIX,
         SUFFIX)
;    CTR=min_increasing_slope ->
     equilibrium_min_increasing_slope(
         VARS,
         PREFIX,
         SUFFIX)
;    CTR=min_width_peak ->
     equilibrium_min_width_peak(VARS,N,PREFIX,SUFFIX)
;    CTR=min_width_valley ->
     equilibrium_min_width_valley(VARS,N,PREFIX,SUFFIX)
```

```
        ;   CTR=peak ->
            equilibrium_peak(VARS,PREFIX,SUFFIX)
        ;   CTR=sum_ctr ->
            equilibrium_sum_ctr(VARS,1,PREFIX,SUFFIX)
        ;   CTR=valley ->
            equilibrium_valley(VARS,PREFIX,SUFFIX)
        ),
        append_length(SUFFIX_INDEX2,SUFFIX,EPSILON).

equilibrium_r(
    VARIABLES,
    INDEX1,
    INDEX2,
    EPSILON,
    COEF1,
    COEF2,
    TOLERANCE,
    CTR) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        N>=1,
        check_type(dvar(1,N),INDEX1),
        check_type(dvar(1,N),INDEX2),
        EPSILON>=0,
        EPSILON=<2,
        TOLERANCE>=0,
        INDEX1#=<INDEX2,
        EPSILON#=INDEX2-INDEX1,
        memberchk(
            CTR,
            [among_diff_0,
             and,
             change,
             deepest_valley,
             highest_peak,
             increasing_nvalue,
             inflexion,
             longest_change,
             longest_decreasing_sequence,
             longest_increasing_sequence,
             max_decreasing_slope,
             max_increasing_slope,
             min_decreasing_slope,
             min_increasing_slope,
             min_width_peak,
```

```
     min_width_valley,
     peak,
     sum_ctr,
     valley]),
(   CTR=among_diff_0 ->
    true
;   CTR=and ->
    true
;   CTR=change ->
    true
;   CTR=deepest_valley ->
    true
;   CTR=highest_peak ->
    true
;   CTR=increasing_nvalue ->
    true
;   CTR=inflexion ->
    true
;   CTR=longest_change ->
    true
;   CTR=longest_decreasing_sequence ->
    true
;   CTR=longest_increasing_sequence ->
    true
;   CTR=max_decreasing_slope ->
    true
;   CTR=max_increasing_slope ->
    true
;   CTR=min_decreasing_slope ->
    true
;   CTR=min_increasing_slope ->
    true
;   CTR=min_width_peak ->
    true
;   CTR=min_width_valley ->
    true
;   CTR=peak ->
    true
;   CTR=sum_ctr ->
    equilibrium_sum_ctr(VARS,0,SUM_PREFIX,SUM_SUFFIX),
    append_length(SUM_SUFFIX_INDEX2,SUM_SUFFIX,EPSILON),
    equilibrium_sum_ctr1(
        1,
        INDEX1,
        SUM_PREFIX,
        SUM_SUFFIX_INDEX2,
```

```
                COEF1,
                COEF2,
                TOLERANCE)
        ;   CTR=valley ->
            true
        ).

equilibrium_sum_ctr1(
    _29915,
    _29962,
    _30009,
    [],
    _30101,
    _30148,
    _30195) :-
        !.

equilibrium_sum_ctr1(
    I,
    INDEX1,
    [C1|RC1],
    [C2|RC2],
    COEF1,
    COEF2,
    TOLERANCE) :-
        INDEX1#=I#=>abs(COEF1*C1-COEF2*C2)#=<TOLERANCE,
        I1 is I+1,
        equilibrium_sum_ctr1(
            I1,
            INDEX1,
            RC1,
            RC2,
            COEF1,
            COEF2,
            TOLERANCE).

sum_c(0,C,_29676,C) :-
        !.

sum_c(INDEX,C,[VAR|R],RES) :-
        INDEX>0,
        NextC is C+VAR,
        NextINDEX is INDEX-1,
        sum_c(NextINDEX,NextC,R,RES).

sum_counters_check([],_29672,[]).
```

```
sum_counters_check([VAR|R],SUM_CUR,[SUM_NEXT|S]) :-
        SUM_NEXT is SUM_CUR+VAR,
        sum_counters_check(R,SUM_NEXT,S).

sum_counters_ref([],_29672,[]).

sum_counters_ref([VAR|R],SUM_CUR,[SUM_NEXT|S]) :-
        SUM_NEXT#=SUM_CUR+VAR,
        sum_counters_ref(R,SUM_NEXT,S).
```

B.156 equivalent

◇ **META-DATA:**

```
ctr_date(equivalent,['20051226']).

ctr_origin(equivalent,'Logic',[]).

ctr_synonyms(equivalent,[eq]).

ctr_arguments(
    equivalent,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    equivalent,
    ['VAR'>=0,
     'VAR'=<1,
     size('VARIABLES')=2,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    equivalent,
    [equivalent(1,[[var-0],[var-0]]),
     equivalent(0,[[var-0],[var-1]]),
     equivalent(0,[[var-1],[var-0]]),
     equivalent(1,[[var-1],[var-1]])]).

ctr_exchangeable(
    equivalent,
    [items('VARIABLES',all),
     vals(['VAR','VARIABLES'^var],int(0 in 1),<,all,dontcare)]).

ctr_eval(
    equivalent,
    [reformulation(equivalent_r),automaton(equivalent_a)]).

ctr_pure_functional_dependency(equivalent,[]).

ctr_functional_dependency(equivalent,1,[2]).

ctr_sol(equivalent,2,0,2,4,[0-2,1-2]).

ctr_sol(equivalent,3,0,3,0,[]).
```

```
ctr_sol(equivalent,4,0,4,0,[]).

ctr_sol(equivalent,5,0,5,0,[]).

ctr_sol(equivalent,6,0,6,0,[]).

ctr_sol(equivalent,7,0,7,0,[]).

ctr_sol(equivalent,8,0,8,0,[]).

equivalent_r(VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,2),
        get_attr1(VARIABLES,[VAR1,VAR2]),
        VAR#<=>(VAR1#<=>VAR2).

equivalent_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,2),
        get_attr1(VARIABLES,LIST),
        append([VAR],LIST,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _41473,
            LIST_VARIABLES,
            [source(s),sink(t)],
            [arc(s,0,i),
             arc(s,1,j),
             arc(i,0,l),
             arc(i,1,k),
             arc(j,0,k),
             arc(j,1,l),
             arc(k,0,t),
             arc(l,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.157 exactly

◇ **META-DATA:**

```
ctr_date(exactly,['20040807','20060809']).

ctr_origin(exactly,'Derived from %c and %c.',[atleast,atmost]).

ctr_synonyms(exactly,[count]).

ctr_arguments(
    exactly,
    ['N'-int,'VARIABLES'-collection(var-dvar),'VALUE'-int]).

ctr_restrictions(
    exactly,
    ['N'>=0,'N'=<size('VARIABLES'),required('VARIABLES',var)]).

ctr_example(
    exactly,
    exactly(2,[[var-4],[var-2],[var-4],[var-5]],4)).

ctr_typical(
    exactly,
    ['N'>0,'N'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_exchangeable(
    exactly,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         int(=\=('VALUE')),
         =\=,
         dontcare,
         dontcare)]).

ctr_graph(
    exactly,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE'],
    ['NARC'='N'],
    []).

ctr_eval(
```

```
    exactly,
    [reformulation(exactly_r),automaton(exactly_a)]).

ctr_pure_functional_dependency(exactly,[]).

ctr_functional_dependency(exactly,1,[2,3]).

ctr_aggregate(exactly,[],[+,union,id]).

exactly_r(N,VARIABLES,VALUE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,NVAR),
        check_type(int(0,NVAR),N),
        integer(VALUE),
        get_attr1(VARIABLES,VARS),
        get_minimum(VARS,MINVARS),
        get_maximum(VARS,MAXVARS),
        MIN is min(MINVARS,VALUE),
        MAX is max(MAXVARS,VALUE),
        complete_card(MIN,MAX,NVAR,[VALUE],[N],VN),
        global_cardinality(VARS,VN).

exactly_a(FLAG,N,VARIABLES,VALUE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,NVAR),
        check_type(int(0,NVAR),N),
        integer(VALUE),
        exactly_signature(VARIABLES,SIGNATURE,VALUE),
        automaton(
            SIGNATURE,
            _45137,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [COUNT]),
        COUNT#=N#<=>FLAG.

exactly_signature([],[],_43672).

exactly_signature([[var-VAR]|VARs],[S|Ss],VALUE) :-
        VAR#=VALUE#<=>S,
        exactly_signature(VARs,Ss,VALUE).
```

B.158 first_value_diff_0

♢ META-DATA:

```
ctr_date(first_value_diff_0,['20120418']).

ctr_origin(first_value_diff_0,'Paparazzi puzzle',[]).

ctr_synonyms(
    first_value_diff_0,
    [first_value_diff_from_0,first_value_different_from_0]).

ctr_arguments(
    first_value_diff_0,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    first_value_diff_0,
    ['VAR'=\=0,size('VARIABLES')>=1,required('VARIABLES',var)]).

ctr_example(
    first_value_diff_0,
    [first_value_diff_0(
         8,
         [[var-0],[var-0],[var-8],[var-0],[var-5]]),
     first_value_diff_0(
         4,
         [[var-4],[var-0],[var-8],[var-0],[var-5]])]).

ctr_typical(
    first_value_diff_0,
    [size('VARIABLES')>1,
     minval('VARIABLES'^var)<0#\/maxval('VARIABLES'^var)>1,
     size('VARIABLES')-among_diff_0('VARIABLES'^var)>=1,
     size('VARIABLES')=<4#\/
     size('VARIABLES')-among_diff_0('VARIABLES'^var)>1]).

ctr_typical_model(
    first_value_diff_0,
    [nval('VARIABLES'^var)>2,atleast(2,'VARIABLES',0)]).

ctr_eval(
    first_value_diff_0,
    [checker(first_value_diff_0_c),
     automaton(first_value_diff_0_a)]).
```

```
ctr_functional_dependency(first_value_diff_0,1,[2]).

ctr_sol(first_value_diff_0,2,0,2,8,[1-4,2-4]).

ctr_sol(first_value_diff_0,3,0,3,63,[1-21,2-21,3-21]).

ctr_sol(first_value_diff_0,4,0,4,624,[1-156,2-156,3-156,4-156]).

ctr_sol(
    first_value_diff_0,
    5,
    0,
    5,
    7775,
    [1-1555,2-1555,3-1555,4-1555,5-1555]).

ctr_sol(
    first_value_diff_0,
    6,
    0,
    6,
    117648,
    [1-19608,2-19608,3-19608,4-19608,5-19608,6-19608]).

ctr_sol(
    first_value_diff_0,
    7,
    0,
    7,
    2097151,
    [1-299593,
     2-299593,
     3-299593,
     4-299593,
     5-299593,
     6-299593,
     7-299593]).

ctr_sol(
    first_value_diff_0,
    8,
    0,
    8,
    43046720,
    [1-5380840,
     2-5380840,
```

```
     3-5380840,
     4-5380840,
     5-5380840,
     6-5380840,
     7-5380840,
     8-5380840]).

first_value_diff_0_c(VAR,VARIABLES) :-
        check_type(dvar,VAR),
        VAR#\=0,
        collection(VARIABLES,[int]),
        first_value_diff_0_c1(VARIABLES,VAR).

first_value_diff_0_c1([[var-0]|R],VAR) :-
        !,
        first_value_diff_0_c1(R,VAR).

first_value_diff_0_c1([[var-VAR]|_45613],VAR).

first_value_diff_0_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar,VAR),
        VAR#\=0,
        collection(VARIABLES,[dvar]),
        VARIABLES=[_45648|_45649],
        first_value_diff_0_signature(VARIABLES,SIGNATURE,VARS),
        automaton(
            VARS,
            VARi,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),
             arc(s,1,t,[VARi]),
             arc(t,0,t),
             arc(t,1,t)],
            [_C],
            [0],
            [COUNT]),
        COUNT#=VAR#<=>FLAG.

first_value_diff_0_signature([],[],[]).

first_value_diff_0_signature([[var-VAR]|VARs],[S|Ss],[VAR|Ts]) :-
        VAR#\=0#<=>S,
        first_value_diff_0_signature(VARs,Ss,Ts).

first_value_diff_0_d(Density,_VAR,VARIABLES) :-
```

```
get_attr1(VARIABLES,VARS),
sort(VARS,SVARS),
length(VARS,N),
length(SVARS,S),
Density is S/N.
```

B.159 full_group

◇ META-DATA:

```
ctr_date(full_group,['20121025']).

ctr_origin(full_group,'Inspired by %c',[group]).

ctr_synonyms(full_group,[group_without_border]).

ctr_arguments(
    full_group,
    ['NGROUP'-dvar,
     'MIN_SIZE'-dvar,
     'MAX_SIZE'-dvar,
     'MIN_DIST'-dvar,
     'MAX_DIST'-dvar,
     'NVAL'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    full_group,
    ['NGROUP'>=0,
     'MIN_SIZE'>=0,
     'MAX_SIZE'>='MIN_SIZE',
     'MIN_DIST'>=0,
     'MAX_DIST'>='MIN_DIST',
     'MAX_DIST'=<size('VARIABLES')-2,
     'NVAL'>='MAX_SIZE',
     'NVAL'>='NGROUP',
     'NVAL'=<size('VARIABLES')-2,
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    full_group,
    full_group(
        2,
        2,
        3,
        1,
        1,
        5,
        [[var-0],
```

```
         [var-1],
         [var-2],
         [var-6],
         [var-2],
         [var-7],
         [var-4],
         [var-8],
         [var-9]],
        [[val-0],[val-2],[val-4],[val-6],[val-8]])).

ctr_typical(
    full_group,
    ['NGROUP'>0,
     'MIN_SIZE'>0,
     'MAX_SIZE'>'MIN_SIZE',
     'MIN_DIST'>0,
     'MAX_DIST'>'MIN_DIST',
     'MAX_DIST'<size('VARIABLES'),
     'NVAL'>'MAX_SIZE',
     'NVAL'>'NGROUP',
     'NVAL'<size('VARIABLES'),
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>0,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    full_group,
    [items('VARIABLES',reverse),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var],
         comp('VALUES'^val),
         =,
         dontcare,
         dontcare)]).

ctr_eval(
    full_group,
    [checker(full_group_c),automata(full_group_a)]).

ctr_pure_functional_dependency(full_group,[]).

ctr_functional_dependency(full_group,1,[7,8]).

ctr_functional_dependency(full_group,2,[7,8]).
```

```
ctr_functional_dependency(full_group,3,[7,8]).

ctr_functional_dependency(full_group,4,[7,8]).

ctr_functional_dependency(full_group,5,[7,8]).

ctr_functional_dependency(full_group,6,[7,8]).

full_group_a(
    NGROUP,
    MIN_SIZE,
    MAX_SIZE,
    MIN_DIST,
    MAX_DIST,
    NVAL,
    VARIABLES,
    VALUES) :-
        check_type(dvar,NGROUP),
        check_type(dvar,MIN_SIZE),
        check_type(dvar,MAX_SIZE),
        check_type(dvar,MIN_DIST),
        check_type(dvar,MAX_DIST),
        check_type(dvar,NVAL),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int]),
        length(VARIABLES,N),
        get_attr1(VALUES,VALS),
        NGROUP#>=0,
        MIN_SIZE#>=0,
        MAX_SIZE#>=MIN_SIZE,
        MIN_DIST#>=0,
        MAX_DIST#>=MIN_DIST,
        MAX_DIST#=<N-2,
        NVAL#>=MAX_SIZE,
        NVAL#>=NGROUP,
        NVAL#=<N-2,
        all_different(VALS),
        full_group_ngroup(NGROUP,VARIABLES,VALUES),
        full_group_min_size(MIN_SIZE,VARIABLES,VALUES),
        full_group_max_size(MAX_SIZE,VARIABLES,VALUES),
        full_group_min_dist(MIN_DIST,VARIABLES,VALUES),
        full_group_max_dist(MAX_DIST,VARIABLES,VALUES),
        full_group_nval(NVAL,VARIABLES,VALUES).

full_group_ngroup(NGROUP,VARIABLES,VALUES) :-
```

```
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        full_group_signature_in(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _34506,
            SIGNATURE,
            [source(s),sink(i),sink(j),sink(s)],
            [arc(s,1,s),
             arc(s,0,i),
             arc(i,0,i),
             arc(i,1,j),
             arc(j,1,j),
             arc(j,0,i,[C+1])],
            [C],
            [0],
            [NGROUP]).

full_group_min_size(MIN_SIZE,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        full_group_signature_in(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _34784,
            SIGNATURE,
            [source(s),
             sink(i),
             sink(j),
             sink(k),
             sink(l),
             sink(s)],
            [arc(s,1,s),
             arc(s,0,i),
             arc(i,0,i),
             arc(i,1,j,[C,D+1]),
             arc(j,1,j,[C,D+1]),
             arc(j,0,k,[D,0]),
             arc(k,0,k),
             arc(k,1,l,[C,D+1]),
```

```
             arc(l,1,l,[C,D+1]),
             arc(l,0,k,[min(C,D),0])],
            [C,D],
            [0,0],
            [MIN_SIZE,_33229]).

full_group_max_size(MAX_SIZE,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        full_group_signature_in(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _34784,
            SIGNATURE,
            [source(s),
             sink(i),
             sink(j),
             sink(k),
             sink(l),
             sink(s)],
            [arc(s,1,s),
             arc(s,0,i),
             arc(i,0,i),
             arc(i,1,j,[C,D+1]),
             arc(j,1,j,[C,D+1]),
             arc(j,0,k,[D,0]),
             arc(k,0,k),
             arc(k,1,l,[C,D+1]),
             arc(l,1,l,[C,D+1]),
             arc(l,0,k,[max(C,D),0])],
            [C,D],
            [0,0],
            [MAX_SIZE,_33229]).

full_group_min_dist(MIN_DIST,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        full_group_signature_not_in(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        automaton(
            SIGNATURE,
```

```
            _34784,
            SIGNATURE,
            [source(s),
             sink(i),
             sink(j),
             sink(k),
             sink(l),
             sink(s)],
            [arc(s,1,s),
             arc(s,0,i),
             arc(i,0,i),
             arc(i,1,j,[C,D+1]),
             arc(j,1,j,[C,D+1]),
             arc(j,0,k,[D,0]),
             arc(k,0,k),
             arc(k,1,l,[C,D+1]),
             arc(l,1,l,[C,D+1]),
             arc(l,0,k,[min(C,D),0])],
            [C,D],
            [0,0],
            [MIN_DIST,_33229]).

full_group_max_dist(MAX_DIST,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        full_group_signature_not_in(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _34784,
            SIGNATURE,
            [source(s),
             sink(i),
             sink(j),
             sink(k),
             sink(l),
             sink(s)],
            [arc(s,1,s),
             arc(s,0,i),
             arc(i,0,i),
             arc(i,1,j,[C,D+1]),
             arc(j,1,j,[C,D+1]),
             arc(j,0,k,[D,0]),
             arc(k,0,k),
```

```
              arc(k,1,l,[C,D+1]),
              arc(l,1,l,[C,D+1]),
              arc(l,0,k,[max(C,D),0])],
             [C,D],
             [0,0],
             [MAX_DIST,_33229]).

full_group_nval(NVAL,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        full_group_signature_in(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _34616,
            SIGNATURE,
            [source(s),sink(i),sink(j),sink(s)],
            [arc(s,1,s),
             arc(s,0,i),
             arc(i,0,i),
             arc(i,1,j,[C,D+1]),
             arc(j,1,j,[C,D+1]),
             arc(j,0,i,[C+D,0])],
            [C,D],
            [0,0],
            [NVAL,_33176]).

full_group_signature_in([],[],_33042).

full_group_signature_in([[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
        VAR in_set SET_OF_VALUES#<=>S,
        full_group_signature_in(VARs,Ss,SET_OF_VALUES).

full_group_signature_not_in([],[],_33042).

full_group_signature_not_in(
    [[var-VAR]|VARs],
    [S|Ss],
    SET_OF_VALUES) :-
        VAR in_set SET_OF_VALUES#<=> #\S,
        full_group_signature_not_in(VARs,Ss,SET_OF_VALUES).

full_group_c(
    NGROUP,
```

```
    MIN_SIZE,
    MAX_SIZE,
    MIN_DIST,
    MAX_DIST,
    NVAL,
    VARIABLES,
    VALUES) :-
        check_type(dvar,NGROUP),
        check_type(dvar,MIN_SIZE),
        check_type(dvar,MAX_SIZE),
        check_type(dvar,MIN_DIST),
        check_type(dvar,MAX_DIST),
        check_type(dvar,NVAL),
        collection(VARIABLES,[int]),
        collection(VALUES,[int]),
        length(VARIABLES,N),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        NGROUP#>=0,
        MIN_SIZE#>=0,
        MAX_SIZE#>=MIN_SIZE,
        MIN_DIST#>=0,
        MAX_DIST#>=MIN_DIST,
        MAX_DIST#=<N-2,
        NVAL#>=MAX_SIZE,
        NVAL#>=NGROUP,
        NVAL#=<N-2,
        sort(VALS,SVALS),
        length(VALS,M),
        length(SVALS,M),
        group_convert(VARS,BOOLS,NBOOLS,VALS),
        full_group_ngroup_c(BOOLS,s,0,NGROUP),
        full_group_min_size_c(BOOLS,s,0,0,MIN_SIZE),
        full_group_max_size_c(BOOLS,s,0,0,MAX_SIZE),
        full_group_min_size_c(NBOOLS,s,0,0,MIN_DIST),
        full_group_max_size_c(NBOOLS,s,0,0,MAX_DIST),
        full_group_nval_c(BOOLS,s,0,0,NVAL).

full_group_ngroup_c([1|R],s,C,NGROUP) :-
        !,
        full_group_ngroup_c(R,s,C,NGROUP).

full_group_ngroup_c([0|R],s,C,NGROUP) :-
        !,
        full_group_ngroup_c(R,i,C,NGROUP).
```

```
full_group_ngroup_c([0|R],i,C,NGROUP) :-
        !,
        full_group_ngroup_c(R,i,C,NGROUP).

full_group_ngroup_c([1|R],i,C,NGROUP) :-
        !,
        full_group_ngroup_c(R,j,C,NGROUP).

full_group_ngroup_c([1|R],j,C,NGROUP) :-
        !,
        full_group_ngroup_c(R,j,C,NGROUP).

full_group_ngroup_c([0|R],j,C,NGROUP) :-
        !,
        C1 is C+1,
        full_group_ngroup_c(R,i,C1,NGROUP).

full_group_ngroup_c([],_33041,C,C).

full_group_min_size_c([1|R],s,C,D,MIN_SIZE) :-
        !,
        full_group_min_size_c(R,s,C,D,MIN_SIZE).

full_group_min_size_c([0|R],s,C,D,MIN_SIZE) :-
        !,
        full_group_min_size_c(R,i,C,D,MIN_SIZE).

full_group_min_size_c([0|R],i,C,D,MIN_SIZE) :-
        !,
        full_group_min_size_c(R,i,C,D,MIN_SIZE).

full_group_min_size_c([1|R],i,C,D,MIN_SIZE) :-
        !,
        D1 is D+1,
        full_group_min_size_c(R,j,C,D1,MIN_SIZE).

full_group_min_size_c([1|R],j,C,D,MIN_SIZE) :-
        !,
        D1 is D+1,
        full_group_min_size_c(R,j,C,D1,MIN_SIZE).

full_group_min_size_c([0|R],j,_C,D,MIN_SIZE) :-
        !,
        full_group_min_size_c(R,k,D,0,MIN_SIZE).

full_group_min_size_c([0|R],k,C,D,MIN_SIZE) :-
```

```
        !,
        full_group_min_size_c(R,k,C,D,MIN_SIZE).

full_group_min_size_c([1|R],k,C,D,MIN_SIZE) :-
        !,
        D1 is D+1,
        full_group_min_size_c(R,l,C,D1,MIN_SIZE).

full_group_min_size_c([1|R],l,C,D,MIN_SIZE) :-
        !,
        D1 is D+1,
        full_group_min_size_c(R,l,C,D1,MIN_SIZE).

full_group_min_size_c([0|R],l,C,D,MIN_SIZE) :-
        !,
        C1 is min(C,D),
        full_group_min_size_c(R,k,C1,0,MIN_SIZE).

full_group_min_size_c([],_33041,C,_33043,C).

full_group_max_size_c([1|R],s,C,D,MAX_SIZE) :-
        !,
        full_group_max_size_c(R,s,C,D,MAX_SIZE).

full_group_max_size_c([0|R],s,C,D,MAX_SIZE) :-
        !,
        full_group_max_size_c(R,i,C,D,MAX_SIZE).

full_group_max_size_c([0|R],i,C,D,MAX_SIZE) :-
        !,
        full_group_max_size_c(R,i,C,D,MAX_SIZE).

full_group_max_size_c([1|R],i,C,D,MAX_SIZE) :-
        !,
        D1 is D+1,
        full_group_max_size_c(R,j,C,D1,MAX_SIZE).

full_group_max_size_c([1|R],j,C,D,MAX_SIZE) :-
        !,
        D1 is D+1,
        full_group_max_size_c(R,j,C,D1,MAX_SIZE).

full_group_max_size_c([0|R],j,_C,D,MAX_SIZE) :-
        !,
        full_group_max_size_c(R,k,D,0,MAX_SIZE).
```

```
full_group_max_size_c([0|R],k,C,D,MAX_SIZE) :-
        !,
        full_group_max_size_c(R,k,C,D,MAX_SIZE).

full_group_max_size_c([1|R],k,C,D,MAX_SIZE) :-
        !,
        D1 is D+1,
        full_group_max_size_c(R,l,C,D1,MAX_SIZE).

full_group_max_size_c([1|R],l,C,D,MAX_SIZE) :-
        !,
        D1 is D+1,
        full_group_max_size_c(R,l,C,D1,MAX_SIZE).

full_group_max_size_c([0|R],l,C,D,MAX_SIZE) :-
        !,
        C1 is max(C,D),
        full_group_max_size_c(R,k,C1,0,MAX_SIZE).

full_group_max_size_c([],_33041,C,_33043,C).

full_group_nval_c([1|R],s,C,D,NVAL) :-
        !,
        full_group_nval_c(R,s,C,D,NVAL).

full_group_nval_c([0|R],s,C,D,NVAL) :-
        !,
        full_group_nval_c(R,i,C,D,NVAL).

full_group_nval_c([0|R],i,C,D,NVAL) :-
        !,
        full_group_nval_c(R,i,C,D,NVAL).

full_group_nval_c([1|R],i,C,D,NVAL) :-
        !,
        D1 is D+1,
        full_group_nval_c(R,j,C,D1,NVAL).

full_group_nval_c([1|R],j,C,D,NVAL) :-
        !,
        D1 is D+1,
        full_group_nval_c(R,j,C,D1,NVAL).

full_group_nval_c([0|R],j,C,D,NVAL) :-
        !,
        C1 is C+D,
```

```
        full_group_nval_c(R,i,C1,0,NVAL).

full_group_nval_c([],_33041,C,_33043,C).
```

B.160 gcd

◇ META-DATA:

```
ctr_predefined(gcd).

ctr_date(gcd,['20070930']).

ctr_origin(gcd,'\\cite{DenmatGotliebDucasse07}',[]).

ctr_arguments(gcd,['X'-dvar,'Y'-dvar,'Z'-dvar]).

ctr_restrictions(gcd,['X'>0,'Y'>0,'Z'>0]).

ctr_example(gcd,gcd(24,60,12)).

ctr_typical(gcd,['X'>1,'Y'>1]).

ctr_exchangeable(gcd,[args([['X','Y'],['Z']])]).

ctr_eval(gcd,[checker(gcd_c)]).

ctr_pure_functional_dependency(gcd,[]).

ctr_functional_dependency(gcd,1,[2,3]).

gcd_c(X,Y,Z) :-
        check_type(int_gteq(1),X),
        check_type(int_gteq(1),Y),
        check_type(dvar_gteq(1),Z),
        GCD is gcd(X,Y),
        Z#=GCD.
```

B.161 geost

◇ **META-DATA:**

```
ctr_predefined(geost).

ctr_date(geost,['20060919','20080609','20090116','20090725']).

ctr_origin(geost,'Generalisation of %c.',[diffn]).

ctr_types(
    geost,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    geost,
    ['K'-int,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    geost,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     required('OBJECTS',[oid,sid,x]),
     distinct('OBJECTS',oid),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
     'OBJECTS'^sid=<size('SBOXES'),
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).
```

```
ctr_example(
    geost,
    geost(
        2,
        [[oid-1,sid-1,x-[[v-1],[v-2]]],
         [oid-2,sid-5,x-[[v-2],[v-1]]],
         [oid-3,sid-8,x-[[v-4],[v-1]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-2],[v-1]]],
         [sid-1,t-[[v-0],[v-1]],l-[[v-1],[v-2]]],
         [sid-1,t-[[v-1],[v-2]],l-[[v-3],[v-1]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
         [sid-2,t-[[v-0],[v-1]],l-[[v-1],[v-3]]],
         [sid-2,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-2],[v-1]]],
         [sid-3,t-[[v-1],[v-1]],l-[[v-1],[v-2]]],
         [sid-3,t-[[v- -2],[v-2]],l-[[v-3],[v-1]]],
         [sid-4,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
         [sid-4,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
         [sid-4,t-[[v-2],[v-1]],l-[[v-1],[v-3]]],
         [sid-5,t-[[v-0],[v-0]],l-[[v-2],[v-1]]],
         [sid-5,t-[[v-1],[v-1]],l-[[v-1],[v-1]]],
         [sid-5,t-[[v-0],[v-2]],l-[[v-2],[v-1]]],
         [sid-6,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
         [sid-6,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
         [sid-6,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-7,t-[[v-0],[v-0]],l-[[v-3],[v-2]]],
         [sid-8,t-[[v-0],[v-0]],l-[[v-2],[v-3]]]])).

ctr_typical(geost,[size('OBJECTS')>1]).

ctr_exchangeable(
    geost,
    [items('OBJECTS',all),
     items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all),
     vals(['SBOXES'^l^v],int(>=(1)),>,dontcare,dontcare)]).

ctr_eval(geost,[builtin(geost_b)]).

ctr_application(geost,[2]).

geost_b(K,[],_53718) :-
        !,
        check_type(int_gteq(1),K).
```

```
geost_b(K,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        geost1(OIDS,SIDS,XS,Objects),
        geost2(SIDES,TS,TL,Sboxes),
        catch(geost(Objects,Sboxes),_Flag,fail).
```

B.162 geost_time

◊ META-DATA:

```
ctr_predefined(geost_time).

ctr_date(geost_time,['20060919']).

ctr_origin(geost_time,'Generalisation of %c.',[diffn]).

ctr_types(
    geost_time,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    geost_time,
    ['K'-int,
     'DIMS'-sint,
     OBJECTS-
     collection(
         oid-int,
         sid-dvar,
         x-'VARIABLES',
         start-dvar,
         duration-dvar,
         end-dvar),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    geost_time,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>=0,
     'DIMS'>=0,
     'DIMS'<'K',
     distinct('OBJECTS',oid),
```

```
      required('OBJECTS',[oid,sid,x]),
      require_at_least(2,'OBJECTS',[start,duration,end]),
      'OBJECTS'^oid>=1,
      'OBJECTS'^oid=<size('OBJECTS'),
      'OBJECTS'^sid>=1,
      'OBJECTS'^sid=<size('SBOXES'),
      'OBJECTS'^duration>=0,
      size('SBOXES')>=1,
      required('SBOXES',[sid,t,l]),
      'SBOXES'^sid>=1,
      'SBOXES'^sid=<size('SBOXES'),
      do_not_overlap('SBOXES')]).

ctr_example(
    geost_time,
    geost_time(
        2,
        {0,1},
        [[oid-1,
          sid-1,
          x-[[v-1],[v-2]],
          start-0,
          duration-1,
          end-1],
         [oid-2,
          sid-5,
          x-[[v-2],[v-1]],
          start-0,
          duration-1,
          end-1],
         [oid-3,
          sid-8,
          x-[[v-4],[v-1]],
          start-0,
          duration-1,
          end-1]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-2],[v-1]]],
         [sid-1,t-[[v-0],[v-1]],l-[[v-1],[v-2]]],
         [sid-1,t-[[v-1],[v-2]],l-[[v-3],[v-1]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
         [sid-2,t-[[v-0],[v-1]],l-[[v-1],[v-3]]],
         [sid-2,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-2],[v-1]]],
         [sid-3,t-[[v-1],[v-1]],l-[[v-1],[v-2]]],
         [sid-3,t-[[v- -2],[v-2]],l-[[v-3],[v-1]]],
         [sid-4,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
```

```
          [sid-4,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
          [sid-4,t-[[v-2],[v-1]],l-[[v-1],[v-3]]],
          [sid-5,t-[[v-0],[v-0]],l-[[v-2],[v-1]]],
          [sid-5,t-[[v-1],[v-1]],l-[[v-1],[v-1]]],
          [sid-5,t-[[v-0],[v-2]],l-[[v-2],[v-1]]],
          [sid-6,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
          [sid-6,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
          [sid-6,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
          [sid-7,t-[[v-0],[v-0]],l-[[v-3],[v-2]]],
          [sid-8,t-[[v-0],[v-0]],l-[[v-2],[v-3]]]]])).

ctr_typical(geost_time,[size('OBJECTS')>1]).

ctr_exchangeable(
    geost_time,
    [items('OBJECTS',all),
     items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all),
     vals(['SBOXES'^l^v],int(>=(1)),>,dontcare,dontcare),
     translate(['OBJECTS'^start,'OBJECTS'^end])]).

ctr_application(geost_time,[3]).
```

B.163 geq

◊ **META-DATA:**

```
ctr_predefined(geq).

ctr_date(geq,['20070821']).

ctr_origin(geq,'Arithmetic.',[]).

ctr_synonyms(geq,[rel,xgteqy]).

ctr_arguments(geq,['VAR1'-dvar,'VAR2'-dvar]).

ctr_example(geq,geq(8,1)).

ctr_exchangeable(
    geq,
    [vals(['VAR1'],int(>=('VAR2')),=\=,all,dontcare),
     vals(['VAR2'],int(=<('VAR1')),=\=,all,dontcare)]).

ctr_eval(geq,[builtin(geq_b)]).

geq_b(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#>=VAR2.
```

B.164 geq_cst

◇ META-DATA:

```
ctr_predefined(geq_cst).

ctr_date(geq_cst,['20090912']).

ctr_origin(geq_cst,'Arithmetic.',[]).

ctr_arguments(geq_cst,['VAR1'-dvar,'VAR2'-dvar,'CST2'-int]).

ctr_example(geq_cst,geq_cst(8,1,7)).

ctr_typical(geq_cst,['CST2'=\=0,'VAR1'>'VAR2'+'CST2']).

ctr_exchangeable(
    geq_cst,
    [args([['VAR1'],['VAR2','CST2']]),
     vals(['VAR1'],int(>=('VAR2'+'CST2')),=\=,all,dontcare),
     vals(['VAR2'],int(=<('VAR1'-'CST2')),=\=,all,dontcare),
     vals(['CST2'],int(=<('VAR1'-'VAR2')),=\=,all,dontcare)]).

ctr_eval(geq_cst,[builtin(geq_cst_b)]).

geq_cst_b(VAR1,VAR2,CST2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        check_type(int,CST2),
        VAR1#>=VAR2+CST2.
```

B.165 global_cardinality

◇ **META-DATA:**

```
ctr_date(
    global_cardinality,
    ['20030820','20040530','20060809','20091218']).

ctr_origin(
    global_cardinality,
    \index{CHARME|indexuse}CHARME \cite{OplobeduMarcovitchTourbier89},
    []).

ctr_synonyms(
    global_cardinality,
    [count,
     distribute,
     distribution,
     gcc,
     card_var_gcc,
     egcc,
     extended_global_cardinality]).

ctr_arguments(
    global_cardinality,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,noccurrence-dvar)]).

ctr_restrictions(
    global_cardinality,
    [required('VARIABLES',var),
     required('VALUES',[val,noccurrence]),
     distinct('VALUES',val),
     'VALUES'^noccurrence>=0,
     'VALUES'^noccurrence=<size('VARIABLES')]).

ctr_example(
    global_cardinality,
    global_cardinality(
        [[var-3],[var-3],[var-8],[var-6]],
        [[val-3,noccurrence-2],
         [val-5,noccurrence-0],
         [val-6,noccurrence-1]])).

ctr_typical(
    global_cardinality,
```

```
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
     size('VARIABLES')>=size('VALUES'),
     minval('VARIABLES'^var)=0#\/
     in_attr('VARIABLES',var,'VALUES',val)]).

ctr_exchangeable(
    global_cardinality,
    [items('VARIABLES',all),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var],
         all(notin('VALUES'^val)),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    global_cardinality,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val],
    ['NVERTEX'='VALUES'^noccurrence],
    []).

ctr_eval(
    global_cardinality,
    [builtin(global_cardinality_b),
     checker(global_cardinality_c)]).

ctr_pure_functional_dependency(global_cardinality,[]).

ctr_functional_dependency(global_cardinality,2-2,[1,2-1]).

ctr_contractible(global_cardinality,[],'VALUES',any).

ctr_cond_imply(
    global_cardinality,
```

```
    and,
    [minval('VARIABLES'^var)=0],
    ['VAR'=0],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    or,
    [maxval('VARIABLES'^var)=1],
    ['VAR'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    min_size_full_zero_stretch,
    [minval('VARIABLES'^var)>0],
    ['MINSIZE'=size('VARIABLES')],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    min_size_full_zero_stretch,
    [maxval('VARIABLES'^var)<0],
    ['MINSIZE'=size('VARIABLES')],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    among_diff_0,
    [range('VALUES'^val)=nval('VALUES'^val),
     minval('VALUES'^val)=<minval('VARIABLES'^var),
     maxval('VALUES'^val)>=maxval('VARIABLES'^var)],
    [],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    atmost_nvalue,
    [range('VALUES'^val)=nval('VALUES'^val),
     minval('VALUES'^val)=<minval('VARIABLES'^var),
     maxval('VALUES'^val)>=maxval('VARIABLES'^var)],
    [],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
```

```
    balance,
    [range('VALUES'^noccurrence)=1,
     range('VALUES'^val)=nval('VALUES'^val),
     minval('VALUES'^val)=minval('VARIABLES'^var),
     maxval('VALUES'^val)=maxval('VARIABLES'^var)],
    ['BALANCE'=0],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    max_n,
    [range('VALUES'^val)=nval('VALUES'^val),
     minval('VALUES'^val)=<minval('VARIABLES'^var),
     maxval('VALUES'^val)>=maxval('VARIABLES'^var)],
    [],
    [none,none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    max_nvalue,
    [range('VALUES'^val)=nval('VALUES'^val),
     minval('VALUES'^val)=<minval('VARIABLES'^var),
     maxval('VALUES'^val)>=maxval('VARIABLES'^var)],
    [],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    min_n,
    [range('VALUES'^val)=nval('VALUES'^val),
     minval('VALUES'^val)=<minval('VARIABLES'^var),
     maxval('VALUES'^val)>=maxval('VARIABLES'^var)],
    [],
    [none,none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
    min_nvalue,
    [range('VALUES'^val)=nval('VALUES'^val),
     minval('VALUES'^val)=<minval('VARIABLES'^var),
     maxval('VALUES'^val)>=maxval('VARIABLES'^var)],
    [],
    [none,'VARIABLES']).

ctr_cond_imply(
    global_cardinality,
```

```
    range_ctr,
    [range('VALUES'^val)=nval('VALUES'^val),
     minval('VALUES'^val)=<minval('VARIABLES'^var),
     maxval('VALUES'^val)>=maxval('VARIABLES'^var)],
    [],
    ['VARIABLES',none,none]).

global_cardinality_b(VARIABLES,VALUES) :-
        length(VARIABLES,N),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int,dvar(0,N)]),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        get_attr2(VALUES,NOCCS),
        all_different(VALS),
        get_minimum(VARS,MINVARS),
        get_maximum(VARS,MAXVARS),
        get_minimum(VALS,MINVALS),
        get_maximum(VALS,MAXVALS),
        MIN is min(MINVARS,MINVALS),
        MAX is max(MAXVARS,MAXVALS),
        complete_card(MIN,MAX,N,VALS,NOCCS,VN),
        global_cardinality(VARS,VN).

global_cardinality_c(VARIABLES,VALUES) :-
        length(VARIABLES,N),
        collection(VARIABLES,[int]),
        collection(VALUES,[int,int(0,N)]),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        sort(VALS,SVALS),
        length(VALS,M),
        length(SVALS,M),
        create_pairs(VARS,PVARS),
        keysort(PVARS,SVARS),
        get_attr12(VALUES,VALOCCS),
        keysort(VALOCCS,SVALOCCS),
        global_cardinality_c1(SVARS,SVALOCCS).

global_cardinality_c1(_83208,[]) :-
        !.

global_cardinality_c1([],[_83213-0|R]) :-
        !,
        global_cardinality_c1([],R).
```

```
global_cardinality_c1([U-U|R],[V-O|S]) :-
        U<V,
        !,
        global_cardinality_c1(R,[V-O|S]).

global_cardinality_c1([U-U|R],[V-0|S]) :-
        U>V,
        !,
        global_cardinality_c1([U-U|R],S).

global_cardinality_c1([U-U|R],[U-1|S]) :-
        !,
        global_cardinality_c1(R,S).

global_cardinality_c1([U-U|R],[U-O|S]) :-
        O1 is O-1,
        global_cardinality_c1(R,[U-O1|S]).
```

B.166 global_cardinality_low_up

◇ **META-DATA:**

```
ctr_date(
    global_cardinality_low_up,
    ['20031008','20040530','20060809','20090521']).

ctr_origin(
    global_cardinality_low_up,
    Used for defining %c.,
    [sliding_distribution]).

ctr_synonyms(global_cardinality_low_up,[gcc_low_up,gcc]).

ctr_arguments(
    global_cardinality_low_up,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
    global_cardinality_low_up,
    [required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,omin,omax]),
     distinct('VALUES',val),
     'VALUES'^omin>=0,
     'VALUES'^omax=<size('VARIABLES'),
     'VALUES'^omin=<'VALUES'^omax]).

ctr_example(
    global_cardinality_low_up,
    global_cardinality_low_up(
        [[var-3],[var-3],[var-8],[var-6]],
        [[val-3,omin-2,omax-3],
         [val-5,omin-0,omax-1],
         [val-6,omin-1,omax-2]])).

ctr_typical(
    global_cardinality_low_up,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
     'VALUES'^omin=<size('VARIABLES'),
     'VALUES'^omax>0,
     'VALUES'^omax<size('VARIABLES'),
```

```
     size('VARIABLES')>size('VALUES'),
     in_attr('VARIABLES',var,'VALUES',val)]).

ctr_exchangeable(
    global_cardinality_low_up,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         all(notin('VALUES'^val)),
         =,
         dontcare,
         dontcare),
     items('VALUES',all),
     vals(['VALUES'^omin],int(>=(0)),>,dontcare,dontcare),
     vals(
         ['VALUES'^omax],
         int(=<(size('VARIABLES'))),
         <,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    global_cardinality_low_up,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val],
    ['NVERTEX'>='VALUES'^omin,'NVERTEX'=<'VALUES'^omax],
    []).

ctr_eval(
    global_cardinality_low_up,
    [reformulation(global_cardinality_low_up_r)]).

ctr_contractible(global_cardinality_low_up,[],'VALUES',any).

ctr_cond_imply(
    global_cardinality_low_up,
    increasing_global_cardinality,
    [increasing('VARIABLES')],
```

```
    [],
    id).

global_cardinality_low_up_r(VARIABLES,VALUES) :-
        length(VARIABLES,N),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int,int(0,N),int(0,N)]),
        length(VALUES,M),
        M>0,
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        get_attr2(VALUES,OMINS),
        get_attr3(VALUES,OMAXS),
        all_different(VALS),
        get_minimum(VARS,MINVARS),
        get_maximum(VARS,MAXVARS),
        get_minimum(VALS,MINVALS),
        get_maximum(VALS,MAXVALS),
        MIN is min(MINVARS,MINVALS),
        MAX is max(MAXVARS,MAXVALS),
        complete_card_low_up(MIN,MAX,N,VALS,OMINS,OMAXS,VN),
        global_cardinality(VARS,VN).
```

B.167 global_cardinality_low_up_no_loop

♢ META-DATA:

```
ctr_date(
    global_cardinality_low_up_no_loop,
    ['20051218','20060809']).

ctr_origin(
    global_cardinality_low_up_no_loop,
    Derived from %c and %c.,
    [global_cardinality_low_up,tree]).

ctr_synonyms(
    global_cardinality_low_up_no_loop,
    [gcc_low_up_no_loop]).

ctr_arguments(
    global_cardinality_low_up_no_loop,
    ['MINLOOP'-int,
     'MAXLOOP'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
    global_cardinality_low_up_no_loop,
    ['MINLOOP'>=0,
     'MINLOOP'=<'MAXLOOP',
     'MAXLOOP'=<size('VARIABLES'),
     required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,omin,omax]),
     distinct('VALUES',val),
     'VALUES'^omin>=0,
     'VALUES'^omax=<size('VARIABLES'),
     'VALUES'^omin=<'VALUES'^omax]).

ctr_example(
    global_cardinality_low_up_no_loop,
    global_cardinality_low_up_no_loop(
        1,
        1,
        [[var-1],[var-1],[var-8],[var-6]],
        [[val-1,omin-1,omax-1],
         [val-5,omin-0,omax-0],
         [val-6,omin-1,omax-2]])).
```

```
ctr_typical(
    global_cardinality_low_up_no_loop,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
     'VALUES'^omin=<size('VARIABLES'),
     'VALUES'^omax>0,
     'VALUES'^omax<size('VARIABLES'),
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    global_cardinality_low_up_no_loop,
    [items('VALUES',all),
     vals(['VALUES'^omin],int(>=(0)),>,dontcare,dontcare),
     vals(
         ['VALUES'^omax],
         int(=<(size('VARIABLES'))),
         <,
         dontcare,
         dontcare)]).

ctr_graph(
    global_cardinality_low_up_no_loop,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val,variables^key=\='VALUES'^val],
    ['NVERTEX'>='VALUES'^omin,'NVERTEX'=<'VALUES'^omax],
    []).

ctr_graph(
    global_cardinality_low_up_no_loop,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var=variables^key],
    ['NARC'>='MINLOOP','NARC'=<'MAXLOOP'],
    []).

ctr_eval(
    global_cardinality_low_up_no_loop,
    [reformulation(global_cardinality_low_up_no_loop_r)]).

global_cardinality_low_up_no_loop_r(
    MINLOOP,
```

```
    MAXLOOP,
    VARIABLES,
    VALUES) :-
        check_type(int_gteq(0),MINLOOP),
        check_type(int_gteq(MINLOOP),MAXLOOP),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        collection(VALUES,[int,int(0,N),int(0,N)]),
        length(VALUES,M),
        M>0,
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        get_attr2(VALUES,OMINS),
        get_attr3(VALUES,OMAXS),
        all_different(VALS),
        gcc_no_loop1(VARS,1,SUMLOOP),
        call(SUMLOOP#>=MINLOOP),
        call(SUMLOOP#=<MAXLOOP),
        global_cardinality_low_up_no_loop1(
            1,
            M,
            N,
            VALS,
            OMINS,
            OMAXS,
            VARS).

global_cardinality_low_up_no_loop1(I,M,_46173,[],[],[],_46177) :-
        I>M,
        !.

global_cardinality_low_up_no_loop1(
    I,
    M,
    N,
    [VAL|RVAL],
    [OMIN|ROMIN],
    [OMAX|ROMAX],
    VARS) :-
        I=<M,
        gcc_no_loop2(1,N,I,VARS,VAL,SUMI),
        call(SUMI#>=OMIN),
        call(SUMI#=<OMAX),
        I1 is I+1,
        global_cardinality_low_up_no_loop1(
            I1,
```

```
M,
N,
RVAL,
ROMIN,
ROMAX,
VARS).
```

B.168 global_cardinality_no_loop

◇ **META-DATA:**

```
ctr_date(global_cardinality_no_loop,['20051104','20060809']).

ctr_origin(
    global_cardinality_no_loop,
    Derived from %c and %c.,
    [global_cardinality,tree]).

ctr_synonyms(global_cardinality_no_loop,[gcc_no_loop]).

ctr_arguments(
    global_cardinality_no_loop,
    ['NLOOP'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,noccurrence-dvar)]).

ctr_restrictions(
    global_cardinality_no_loop,
    ['NLOOP'>=0,
     'NLOOP'=<size('VARIABLES'),
     required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,noccurrence]),
     distinct('VALUES',val),
     'VALUES'^noccurrence>=0,
     'VALUES'^noccurrence=<size('VARIABLES')]).

ctr_example(
    global_cardinality_no_loop,
    global_cardinality_no_loop(
        1,
        [[var-1],[var-1],[var-8],[var-6]],
        [[val-1,noccurrence-1],
         [val-5,noccurrence-0],
         [val-6,noccurrence-1]])).

ctr_typical(
    global_cardinality_no_loop,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
     size('VARIABLES')>size('VALUES')]).
```

```
ctr_exchangeable(
    global_cardinality_no_loop,
    [items('VALUES',all)]).

ctr_graph(
    global_cardinality_no_loop,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val,variables^key=\='VALUES'^val],
    ['NVERTEX'='VALUES'^noccurrence],
    []).

ctr_graph(
    global_cardinality_no_loop,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var=variables^key],
    ['NARC'='NLOOP'],
    []).

ctr_eval(
    global_cardinality_no_loop,
    [reformulation(global_cardinality_no_loop_r)]).

ctr_pure_functional_dependency(global_cardinality_no_loop,[]).

ctr_functional_dependency(global_cardinality_no_loop,1,[2]).

ctr_functional_dependency(
    global_cardinality_no_loop,
    3-2,
    [2,3-1]).

global_cardinality_no_loop_r(NLOOP,VARIABLES,VALUES) :-
        check_type(dvar_gteq(0),NLOOP),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        NLOOP#=<N,
        collection(VALUES,[int,dvar(0,N)]),
        length(VALUES,M),
        M>0,
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        get_attr2(VALUES,NOCCURRENCES),
```

```
        all_different(VALS),
        gcc_no_loop1(VARS,1,SUMLOOP),
        call(SUMLOOP#=NLOOP),
        global_cardinality_no_loop1(
            1,
            M,
            N,
            VALS,
            NOCCURRENCES,
            VARS).

global_cardinality_no_loop1(I,M,_45802,[],[],_45805) :-
        I>M,
        !.

global_cardinality_no_loop1(
    I,
    M,
    N,
    [VAL|RVAL],
    [NOCCURRENCE|RNOCCURRENCE],
    VARS) :-
        I=<M,
        gcc_no_loop2(1,N,I,VARS,VAL,SUMI),
        call(SUMI#=NOCCURRENCE),
        I1 is I+1,
        global_cardinality_no_loop1(
            I1,
            M,
            N,
            RVAL,
            RNOCCURRENCE,
            VARS).
```

B.169 global_cardinality_with_costs

◇ **META-DATA:**

```
ctr_date(
    global_cardinality_with_costs,
    ['20030820','20040530','20060809','20090425']).

ctr_origin(global_cardinality_with_costs,'\\cite{Regin99a}',[]).

ctr_synonyms(global_cardinality_with_costs,[gccc,cost_gcc]).

ctr_arguments(
    global_cardinality_with_costs,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,noccurrence-dvar),
     'MATRIX'-collection(i-int,j-int,c-int),
     'COST'-dvar]).

ctr_restrictions(
    global_cardinality_with_costs,
    [required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,noccurrence]),
     distinct('VALUES',val),
     'VALUES'^noccurrence>=0,
     'VALUES'^noccurrence=<size('VARIABLES'),
     required('MATRIX',[i,j,c]),
     increasing_seq('MATRIX',[i,j]),
     'MATRIX'^i>=1,
     'MATRIX'^i=<size('VARIABLES'),
     'MATRIX'^j>=1,
     'MATRIX'^j=<size('VALUES'),
     size('MATRIX')=size('VARIABLES')*size('VALUES')]).

ctr_example(
    global_cardinality_with_costs,
    global_cardinality_with_costs(
        [[var-3],[var-3],[var-3],[var-6]],
        [[val-3,noccurrence-3],
         [val-5,noccurrence-0],
         [val-6,noccurrence-1]],
        [[i-1,j-1,c-4],
         [i-1,j-2,c-1],
         [i-1,j-3,c-7],
         [i-2,j-1,c-1],
```

```
         [i-2,j-2,c-0],
         [i-2,j-3,c-8],
         [i-3,j-1,c-3],
         [i-3,j-2,c-2],
         [i-3,j-3,c-1],
         [i-4,j-1,c-0],
         [i-4,j-2,c-0],
         [i-4,j-3,c-6]],
        14)).

ctr_typical(
    global_cardinality_with_costs,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
     range('VALUES'^noccurrence)>1,
     range('MATRIX'^c)>1,
     size('VARIABLES')>size('VALUES')]).

ctr_graph(
    global_cardinality_with_costs,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val],
    ['NVERTEX'='VALUES'^noccurrence],
    []).

ctr_graph(
    global_cardinality_with_costs,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['SUM_WEIGHT_ARC'(
         MATRIX@
         ((variables^key-1)*size('VALUES')+values^key)^
         c)=
     COST],
    []).

ctr_eval(
    global_cardinality_with_costs,
    [reformulation(global_cardinality_with_costs_r)]).

ctr_pure_functional_dependency(
```

```
    global_cardinality_with_costs,
    []).

ctr_functional_dependency(
    global_cardinality_with_costs,
    2-2,
    [1]).

ctr_functional_dependency(
    global_cardinality_with_costs,
    4,
    [1,2,3]).

global_cardinality_with_costs_r(VARIABLES,VALUES,MATRIX,COST) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        collection(VALUES,[int,dvar(0,N)]),
        length(VALUES,M),
        M>0,
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        all_different(VALS),
        collection(MATRIX,[int(1,N),int(1,M),int]),
        collection_increasing_seq(MATRIX,[1,2]),
        eval(global_cardinality(VARIABLES,VALUES)),
        get_attr3(MATRIX,CS),
        global_cardinality_with_costs1(VARS,VALS,M,CS,TERM),
        call(TERM#=COST).

global_cardinality_with_costs1([],_57475,_57476,_57477,0).

global_cardinality_with_costs1([VAR|R],VALS,M,CMAT,C+S) :-
        global_cardinality_with_costs2(
            M,
            CMAT,
            ELEMTABLE,
            RESTCMAT),
        element(IVAL,VALS,VAR),
        element(IVAL,ELEMTABLE,C),
        global_cardinality_with_costs1(R,VALS,M,RESTCMAT,S).

global_cardinality_with_costs2(0,CMAT,[],CMAT) :-
        !.

global_cardinality_with_costs2(I,[C|R],[C|S],T) :-
        I>0,
```

```
I1 is I-1,
global_cardinality_with_costs2(I1,R,S,T).
```

B.170 global_contiguity

◇ **META-DATA:**

```
ctr_date(global_contiguity,['20030820','20040530','20060809']).

ctr_origin(global_contiguity,'\\cite{Maher02}',[]).

ctr_synonyms(global_contiguity,[contiguity]).

ctr_arguments(
    global_contiguity,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    global_contiguity,
    [required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    global_contiguity,
    global_contiguity([[var-0],[var-1],[var-1],[var-0]])).

ctr_typical(global_contiguity,[size('VARIABLES')>2]).

ctr_typical_model(
    global_contiguity,
    [range('VARIABLES'^var)>1,atleast(2,'VARIABLES',1)]).

ctr_exchangeable(
    global_contiguity,
    [items('VARIABLES',reverse)]).

ctr_graph(
    global_contiguity,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2),
     'LOOP'>>collection(variables1,variables2)],
    [variables1^var=variables2^var,variables1^var=1],
    ['NCC'=<1],
    []).

ctr_eval(
    global_contiguity,
```

```
    [checker(global_contiguity_c),
     automaton(global_contiguity_a)]).

ctr_contractible(global_contiguity,[],'VARIABLES',any).

ctr_cond_imply(
    global_contiguity,
    some_equal,
    [size('VARIABLES')>2],
    [],
    id).

ctr_sol(global_contiguity,2,0,1,4,-).

ctr_sol(global_contiguity,3,0,1,7,-).

ctr_sol(global_contiguity,4,0,1,11,-).

ctr_sol(global_contiguity,5,0,1,16,-).

ctr_sol(global_contiguity,6,0,1,22,-).

ctr_sol(global_contiguity,7,0,1,29,-).

ctr_sol(global_contiguity,8,0,1,37,-).

ctr_sol(global_contiguity,9,0,1,46,-).

ctr_sol(global_contiguity,10,0,1,56,-).

ctr_sol(global_contiguity,11,0,1,67,-).

ctr_sol(global_contiguity,12,0,1,79,-).

ctr_sol(global_contiguity,13,0,1,92,-).

ctr_sol(global_contiguity,14,0,1,106,-).

ctr_sol(global_contiguity,15,0,1,121,-).

ctr_sol(global_contiguity,16,0,1,137,-).

ctr_sol(global_contiguity,17,0,1,154,-).

ctr_sol(global_contiguity,18,0,1,172,-).
```

```
ctr_sol(global_contiguity,19,0,1,191,-).

ctr_sol(global_contiguity,20,0,1,211,-).

ctr_sol(global_contiguity,21,0,1,232,-).

ctr_sol(global_contiguity,22,0,1,254,-).

ctr_sol(global_contiguity,23,0,1,277,-).

ctr_sol(global_contiguity,24,0,1,301,-).

global_contiguity_c([]) :-
        !.

global_contiguity_c(VARIABLES) :-
        collection(VARIABLES,[int(0,1)]),
        get_attr1(VARIABLES,VARS),
        global_contiguity_c1(VARS).

global_contiguity_c1([]) :-
        !.

global_contiguity_c1([0|R]) :-
        !,
        global_contiguity_c1(R).

global_contiguity_c1([1|R]) :-
        global_contiguity_c2(R).

global_contiguity_c2([]) :-
        !.

global_contiguity_c2([1|R]) :-
        !,
        global_contiguity_c2(R).

global_contiguity_c2([0|R]) :-
        global_contiguity_c3(R).

global_contiguity_c3([]) :-
        !.

global_contiguity_c3([0|R]) :-
        global_contiguity_c3(R).
```

```
global_contiguity_a(1,[]) :-
        !.

global_contiguity_a(0,[]) :-
        !,
        fail.

global_contiguity_a(FLAG,VARIABLES) :-
        collection(VARIABLES,[dvar(0,1)]),
        get_attr1(VARIABLES,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _54592,
            LIST_VARIABLES,
            [source(s),sink(m),sink(z),sink(s)],
            [arc(s,0,s),
             arc(s,1,m),
             arc(m,0,z),
             arc(m,1,m),
             arc(z,0,z)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.171 golomb

◇ **META-DATA:**

```
ctr_date(golomb,['20000128','20030820','20040530','20060809']).

ctr_origin(golomb,'Inspired by \\cite{Golomb72}.',[]).

ctr_arguments(golomb,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    golomb,
    [required('VARIABLES',var),
     'VARIABLES'^var>=0,
     strictly_increasing('VARIABLES')]).

ctr_example(golomb,golomb([[var-0],[var-1],[var-4],[var-6]])).

ctr_typical(golomb,[size('VARIABLES')>2]).

ctr_exchangeable(golomb,[translate(['VARIABLES'^var])]).

ctr_derived_collections(
    golomb,
    [col('PAIRS'-collection(x-dvar,y-dvar),
         [> -item(x-'VARIABLES'^var,y-'VARIABLES'^var)])]).

ctr_graph(
    golomb,
    ['PAIRS'],
    2,
    ['CLIQUE'>>collection(pairs1,pairs2)],
    [pairs1^y-pairs1^x=pairs2^y-pairs2^x],
    ['MAX_NSCC'=<1],
    []).

ctr_eval(golomb,[checker(golomb_c),reformulation(golomb_r)]).

ctr_contractible(golomb,[],'VARIABLES',any).

ctr_cond_imply(
    golomb,
    increasing_nvalue,
    [],
    ['NVAL'=nval('VARIABLES'^var)],
    [none,'VARIABLES']).
```

```
ctr_cond_imply(
    golomb,
    soft_alldifferent_ctr,
    [],
    [],
    [none,'VARIABLES']).

ctr_sol(golomb,2,0,2,3,-).

ctr_sol(golomb,3,0,3,2,-).

ctr_sol(golomb,4,0,6,2,-).

ctr_sol(golomb,5,0,11,4,-).

ctr_sol(golomb,6,0,17,8,-).

ctr_sol(golomb,7,0,25,10,-).

ctr_sol(golomb,8,0,34,2,-).

ctr_sol(golomb,9,0,44,2,-).

ctr_sol(golomb,10,0,55,2,-).

ctr_sol(golomb,11,0,72,4,-).

golomb_c([]) :-
        !.

golomb_c(VARIABLES) :-
        collection(VARIABLES,[int_gteq(0)]),
        golomb_increasing(VARIABLES),
        get_attr1(VARIABLES,VARS),
        golomb3(VARS,D),
        sort(D,SD),
        length(D,N),
        length(SD,N).

golomb_increasing([]) :-
        !.

golomb_increasing([_51817]) :-
        !.
```

```
golomb_increasing([[var-X],[var-Y]|R]) :-
        X<Y,
        golomb_increasing([[var-Y]|R]).

golomb_r([]) :-
        !.

golomb_r(VARIABLES) :-
        collection(VARIABLES,[dvar_gteq(0)]),
        collection_increasing_seq(VARIABLES,[1]),
        get_attr1(VARIABLES,VARS),
        golomb1(VARS,D),
        all_different(D).

golomb1([_51818],[]) :-
        !.

golomb1([U,V|R],Diffs) :-
        golomb2([V|R],U,D),
        golomb1([V|R],Diff),
        append(D,Diff,Diffs).

golomb2([],_51814,[]).

golomb2([Vi|R],Vj,[D|S]) :-
        D#=Vi-Vj,
        golomb2(R,Vj,S).

golomb3([_51818],[]) :-
        !.

golomb3([U,V|R],Diffs) :-
        golomb4([V|R],U,D),
        sort(D,SD),
        length(D,N),
        length(SD,N),
        golomb3([V|R],Diff),
        append(SD,Diff,Diffs).

golomb4([],_51814,[]).

golomb4([Vi|R],Vj,[D|S]) :-
        D is Vi-Vj,
        golomb4(R,Vj,S).
```

B.172 graph_crossing

◇ META-DATA:

```
ctr_date(
    graph_crossing,
    ['20000128','20030820','20040530','20060809']).

ctr_origin(graph_crossing,'N.~Beldiceanu',[]).

ctr_synonyms(graph_crossing,[crossing,ncross]).

ctr_arguments(
    graph_crossing,
    ['NCROSS'-dvar,'NODES'-collection(succ-dvar,x-int,y-int)]).

ctr_restrictions(
    graph_crossing,
    ['NCROSS'>=0,
     required('NODES',[succ,x,y]),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    graph_crossing,
    graph_crossing(
        2,
        [[succ-1,x-4,y-7],
         [succ-1,x-2,y-5],
         [succ-1,x-7,y-6],
         [succ-2,x-1,y-2],
         [succ-3,x-2,y-2],
         [succ-2,x-5,y-3],
         [succ-3,x-8,y-2],
         [succ-9,x-6,y-2],
         [succ-10,x-10,y-6],
         [succ-8,x-10,y-1]])).

ctr_typical(
    graph_crossing,
    [size('NODES')>1,
     range('NODES'^succ)>1,
     range('NODES'^x)>1,
     range('NODES'^y)>1]).

ctr_exchangeable(
```

```
    graph_crossing,
    [attrs_sync('NODES',[[succ],[x,y]]),
     translate(['NODES'^x]),
     translate(['NODES'^y])]).

ctr_graph(
    graph_crossing,
    ['NODES'],
    2,
    ['CLIQUE'(<)>>collection(n1,n2)],
    [max(n1^x,'NODES'@(n1^succ)^x)>=
     min(n2^x,'NODES'@(n2^succ)^x),
     max(n2^x,'NODES'@(n2^succ)^x)>=
     min(n1^x,'NODES'@(n1^succ)^x),
     max(n1^y,'NODES'@(n1^succ)^y)>=
     min(n2^y,'NODES'@(n2^succ)^y),
     max(n2^y,'NODES'@(n2^succ)^y)>=
     min(n1^y,'NODES'@(n1^succ)^y),
     (n2^x-'NODES'@(n1^succ)^x)*
     ('NODES'@(n1^succ)^y-n1^y)-
     ('NODES'@(n1^succ)^x-n1^x)*(n2^y-'NODES'@(n1^succ)^y)=\=
     0,
     ('NODES'@(n2^succ)^x-'NODES'@(n1^succ)^x)*
     (n2^y-n1^y)-
     (n2^x-n1^x)*('NODES'@(n2^succ)^y-'NODES'@(n1^succ)^y)=\=
     0,
     sign(
         (n2^x-'NODES'@(n1^succ)^x)*
         ('NODES'@(n1^succ)^y-n1^y)-
         ('NODES'@(n1^succ)^x-n1^x)*
         (n2^y-'NODES'@(n1^succ)^y))=\=
     sign(
         ('NODES'@(n2^succ)^x-'NODES'@(n1^succ)^x)*
         (n2^y-n1^y)-
         (n2^x-n1^x)*
         ('NODES'@(n2^succ)^y-'NODES'@(n1^succ)^y))],
    ['NARC'='NCROSS'],
    []).

ctr_pure_functional_dependency(graph_crossing,[]).

ctr_functional_dependency(graph_crossing,1,[2]).

ctr_application(graph_crossing,[2]).
```

B.173 graph_isomorphism

◇ **META-DATA:**

```
ctr_predefined(graph_isomorphism).

ctr_date(graph_isomorphism,['20090822']).

ctr_origin(graph_isomorphism,'\\cite{Gregor79}',[]).

ctr_arguments(
    graph_isomorphism,
    ['NODES_PATTERN'-collection(index-int,succ-sint),
     'NODES_TARGET'-collection(index-int,succ-sint),
     'FUNCTION'-collection(image-dvar)]).

ctr_restrictions(
    graph_isomorphism,
    [required('NODES_PATTERN',[index,succ]),
     'NODES_PATTERN'^index>=1,
     'NODES_PATTERN'^index=<size('NODES_PATTERN'),
     distinct('NODES_PATTERN',index),
     'NODES_PATTERN'^succ>=1,
     'NODES_PATTERN'^succ=<size('NODES_PATTERN'),
     required('NODES_TARGET',[index,succ]),
     'NODES_TARGET'^index>=1,
     'NODES_TARGET'^index=<size('NODES_TARGET'),
     distinct('NODES_TARGET',index),
     'NODES_TARGET'^succ>=1,
     'NODES_TARGET'^succ=<size('NODES_TARGET'),
     size('NODES_TARGET')=size('NODES_PATTERN'),
     required('FUNCTION',[image]),
     'FUNCTION'^image>=1,
     'FUNCTION'^image=<size('NODES_TARGET'),
     distinct('FUNCTION',image),
     size('FUNCTION')=size('NODES_PATTERN')]).

ctr_example(
    graph_isomorphism,
    graph_isomorphism(
        [[index-1,succ-{2,4}],
         [index-2,succ-{1,3,4}],
         [index-3,succ-{}],
         [index-4,succ-{}]],
        [[index-1,succ-{}],
         [index-2,succ-{1,3,4}],
```

```
         [index-3,succ-{}],
         [index-4,succ-{1,2}]],
        [[image-4],[image-2],[image-3],[image-1]])).

ctr_typical(graph_isomorphism,[size('NODES_PATTERN')>1]).

ctr_exchangeable(
    graph_isomorphism,
    [items('NODES_PATTERN',all),items('NODES_TARGET',all)]).
```

B.174 group

◇ META-DATA:

```
ctr_date(group,['20000128','20030820','20040530','20060809']).

ctr_origin(group,'\\index{CHIP|indexuse}CHIP',[]).

ctr_arguments(
    group,
    ['NGROUP'-dvar,
     'MIN_SIZE'-dvar,
     'MAX_SIZE'-dvar,
     'MIN_DIST'-dvar,
     'MAX_DIST'-dvar,
     'NVAL'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    group,
    ['NGROUP'>=0,
     'MIN_SIZE'>=0,
     'MAX_SIZE'>='MIN_SIZE',
     'MIN_DIST'>=0,
     'MAX_DIST'>='MIN_DIST',
     'MAX_DIST'=<size('VARIABLES'),
     'NVAL'>='MAX_SIZE',
     'NVAL'>='NGROUP',
     'NVAL'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    group,
    group(
        2,
        1,
        2,
        2,
        4,
        3,
        [[var-2],
         [var-8],
         [var-1],
```

```
          [var-7],
          [var-4],
          [var-5],
          [var-1],
          [var-1],
          [var-1]],
         [[val-0],[val-2],[val-4],[val-6],[val-8]])).

ctr_typical(
    group,
    ['NGROUP'>0,
     'MIN_SIZE'>0,
     'MAX_SIZE'>'MIN_SIZE',
     'MIN_DIST'>0,
     'MAX_DIST'>'MIN_DIST',
     'MAX_DIST'<size('VARIABLES'),
     'NVAL'>'MAX_SIZE',
     'NVAL'>'NGROUP',
     'NVAL'<size('VARIABLES'),
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>0,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    group,
    [items('VARIABLES',reverse),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var],
         comp('VALUES'^val),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    group,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2),
     'LOOP'>>collection(variables1,variables2)],
    [variables1^var in 'VALUES',variables2^var in 'VALUES'],
    ['NCC'='NGROUP',
     'MIN_NCC'='MIN_SIZE',
     'MAX_NCC'='MAX_SIZE',
     'NVERTEX'='NVAL'],
```

```
    []).

ctr_graph(
    group,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2),
     'LOOP'>>collection(variables1,variables2)],
    [not_in(variables1^var,'VALUES'),
     not_in(variables2^var,'VALUES')],
    ['MIN_NCC'='MIN_DIST','MAX_NCC'='MAX_DIST'],
    []).

ctr_eval(group,[checker(group_c),automata(group_a)]).

ctr_pure_functional_dependency(group,[]).

ctr_functional_dependency(group,1,[7,8]).

ctr_functional_dependency(group,2,[7,8]).

ctr_functional_dependency(group,3,[7,8]).

ctr_functional_dependency(group,4,[7,8]).

ctr_functional_dependency(group,5,[7,8]).

ctr_functional_dependency(group,6,[7,8]).

group_a(
    NGROUP,
    MIN_SIZE,
    MAX_SIZE,
    MIN_DIST,
    MAX_DIST,
    NVAL,
    VARIABLES,
    VALUES) :-
        check_type(dvar,NGROUP),
        check_type(dvar,MIN_SIZE),
        check_type(dvar,MAX_SIZE),
        check_type(dvar,MIN_DIST),
        check_type(dvar,MAX_DIST),
        check_type(dvar,NVAL),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int]),
```

```
        length(VARIABLES,N),
        get_attr1(VALUES,VALS),
        NGROUP#>=0,
        MIN_SIZE#>=0,
        MAX_SIZE#>=MIN_SIZE,
        MIN_DIST#>=0,
        MAX_DIST#>=MIN_DIST,
        MAX_DIST#=<N,
        NVAL#>=MAX_SIZE,
        NVAL#>=NGROUP,
        NVAL#=<N,
        all_different(VALS),
        group_ngroup(NGROUP,VARIABLES,VALUES),
        group_min_size(MIN_SIZE,VARIABLES,VALUES),
        group_max_size(MAX_SIZE,VARIABLES,VALUES),
        group_min_dist(MIN_DIST,VARIABLES,VALUES),
        group_max_dist(MAX_DIST,VARIABLES,VALUES),
        group_nval(NVAL,VARIABLES,VALUES).

group_ngroup(NGROUP,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        group_signature_in(VARIABLES,SIGNATURE,SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _62024,
            SIGNATURE,
            [source(s),sink(i),sink(s)],
            [arc(s,0,s),
             arc(s,1,i,[C+1]),
             arc(i,1,i),
             arc(i,0,s)],
            [C],
            [0],
            [NGROUP]).

group_min_size(MIN_SIZE,VARIABLES,VALUES) :-
        length(VARIABLES,NVAR),
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        group_signature_in(VARIABLES,SIGNATURE,SET_OF_VALUES),
        MIN_SIZE#=min(C1,D1),
        automaton(
            SIGNATURE,
            _62463,
            SIGNATURE,
```

```
            [source(s),sink(i),sink(s)],
            [arc(s,0,s),
             arc(s,1,i,[C,1]),
             arc(i,1,i,[C,D+1]),
             arc(i,0,s,[min(C,D),D])],
            [C,D],
            [NVAR,0],
            [C1,D1]).

group_max_size(MAX_SIZE,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        group_signature_in(VARIABLES,SIGNATURE,SET_OF_VALUES),
        MAX_SIZE#=max(C1,D1),
        automaton(
            SIGNATURE,
            _62234,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s,[C,D+1]),arc(s,0,s,[max(C,D),0])],
            [C,D],
            [0,0],
            [C1,D1]).

group_min_dist(MIN_DIST,VARIABLES,VALUES) :-
        length(VARIABLES,NVAR),
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        group_signature_not_in(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        MIN_DIST#=min(C1,D1),
        automaton(
            SIGNATURE,
            _62726,
            SIGNATURE,
            [source(s),sink(i),sink(s)],
            [arc(s,0,s),
             arc(s,1,i,[C,1]),
             arc(i,1,i,[C,D+1]),
             arc(i,0,s,[min(C,D),D])],
            [C,D],
            [NVAR,0],
            [C1,D1]).
```

```
group_max_dist(MAX_DIST,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        group_signature_not_in(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        MAX_DIST#=max(C1,D1),
        automaton(
            SIGNATURE,
            _62497,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s,[C,D+1]),arc(s,0,s,[max(C,D),0])],
            [C,D],
            [0,0],
            [C1,D1]).

group_nval(NVAL,VARIABLES,VALUES) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        group_signature_in(VARIABLES,SIGNATURE,SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _61982,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [NVAL]).

group_signature_in([],[],_60865).

group_signature_in([[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
        VAR in_set SET_OF_VALUES#<=>S,
        group_signature_in(VARs,Ss,SET_OF_VALUES).

group_signature_not_in([],[],_60865).

group_signature_not_in([[var-VAR]|VARs],[S|Ss],SET_OF_VALUES) :-
        VAR in_set SET_OF_VALUES#<=> #\S,
        group_signature_not_in(VARs,Ss,SET_OF_VALUES).

group_c(
    NGROUP,
```

```
    MIN_SIZE,
    MAX_SIZE,
    MIN_DIST,
    MAX_DIST,
    NVAL,
    VARIABLES,
    VALUES) :-
        check_type(dvar,NGROUP),
        check_type(dvar,MIN_SIZE),
        check_type(dvar,MAX_SIZE),
        check_type(dvar,MIN_DIST),
        check_type(dvar,MAX_DIST),
        check_type(dvar,NVAL),
        collection(VARIABLES,[int]),
        collection(VALUES,[int]),
        length(VARIABLES,N),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        NGROUP#>=0,
        MIN_SIZE#>=0,
        MAX_SIZE#>=MIN_SIZE,
        MIN_DIST#>=0,
        MAX_DIST#>=MIN_DIST,
        MAX_DIST#=<N,
        NVAL#>=MAX_SIZE,
        NVAL#>=NGROUP,
        NVAL#=<N,
        sort(VALS,SVALS),
        length(VALS,M),
        length(SVALS,M),
        group_convert(VARS,BOOLS,NBOOLS,VALS),
        group_ngroup_c(BOOLS,s,0,NGROUP),
        group_min_size_c(BOOLS,s,N,0,MIN_SIZE),
        group_max_size_c(BOOLS,0,0,MAX_SIZE),
        group_min_size_c(NBOOLS,s,N,0,MIN_DIST),
        group_max_size_c(NBOOLS,0,0,MAX_DIST),
        group_nval_c(BOOLS,0,NVAL).

group_ngroup_c([0|R],s,C,NGROUP) :-
        !,
        group_ngroup_c(R,s,C,NGROUP).

group_ngroup_c([1|R],s,C,NGROUP) :-
        !,
        C1 is C+1,
        group_ngroup_c(R,i,C1,NGROUP).
```

```
group_ngroup_c([1|R],i,C,NGROUP) :-
        !,
        group_ngroup_c(R,i,C,NGROUP).

group_ngroup_c([0|R],i,C,NGROUP) :-
        !,
        group_ngroup_c(R,s,C,NGROUP).

group_ngroup_c([],_60864,C,C).

group_min_size_c([0|R],s,C,D,MIN_SIZE) :-
        !,
        group_min_size_c(R,s,C,D,MIN_SIZE).

group_min_size_c([1|R],s,C,_D,MIN_SIZE) :-
        !,
        group_min_size_c(R,i,C,1,MIN_SIZE).

group_min_size_c([1|R],i,C,D,MIN_SIZE) :-
        !,
        D1 is D+1,
        group_min_size_c(R,i,C,D1,MIN_SIZE).

group_min_size_c([0|R],i,C,D,MIN_SIZE) :-
        !,
        C1 is min(C,D),
        group_min_size_c(R,s,C1,D,MIN_SIZE).

group_min_size_c([],_60867,C,D,MIN_SIZE) :-
        M is min(C,D),
        MIN_SIZE#=M.

group_max_size_c([1|R],C,D,MAX_SIZE) :-
        !,
        D1 is D+1,
        group_max_size_c(R,C,D1,MAX_SIZE).

group_max_size_c([0|R],C,D,MAX_SIZE) :-
        !,
        C1 is max(C,D),
        group_max_size_c(R,C1,0,MAX_SIZE).

group_max_size_c([],C,D,MAX_SIZE) :-
        M is max(C,D),
        MAX_SIZE#=M.
```

```
group_nval_c([0|R],C,NVAL) :-
        !,
        group_nval_c(R,C,NVAL).

group_nval_c([1|R],C,NVAL) :-
        !,
        C1 is C+1,
        group_nval_c(R,C1,NVAL).

group_nval_c([],C,C).
```

B.175 group_skip_isolated_item

◇ **META-DATA:**

```
ctr_date(
    group_skip_isolated_item,
    ['20000128','20030820','20040530','20060809','20130301']).

ctr_origin(group_skip_isolated_item,'Derived from %c.',[group]).

ctr_arguments(
    group_skip_isolated_item,
    ['NGROUP'-dvar,
     'MIN_SIZE'-dvar,
     'MAX_SIZE'-dvar,
     'NVAL'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    group_skip_isolated_item,
    ['NGROUP'>=0,
     3*'NGROUP'=<size('VARIABLES')+1,
     'MIN_SIZE'>=0,
     'MIN_SIZE'=\=1,
     'MAX_SIZE'>='MIN_SIZE',
     'NVAL'>='MAX_SIZE',
     'NVAL'>='NGROUP',
     'NVAL'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    group_skip_isolated_item,
    group_skip_isolated_item(
        1,
        2,
        2,
        3,
        [[var-2],
         [var-8],
         [var-1],
         [var-7],
         [var-4],
         [var-5],
```

```
         [var-1],
         [var-1],
         [var-1]],
        [[val-0],[val-2],[val-4],[val-6],[val-8]])).

ctr_typical(
    group_skip_isolated_item,
    ['NGROUP'>0,
     'MIN_SIZE'>0,
     'NVAL'>'MAX_SIZE',
     'NVAL'>'NGROUP',
     'NVAL'<size('VARIABLES'),
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>0,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    group_skip_isolated_item,
    [items('VARIABLES',reverse),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var],
         comp('VALUES'^val),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    group_skip_isolated_item,
    ['VARIABLES'],
    2,
    ['CHAIN'>>collection(variables1,variables2)],
    [variables1^var in 'VALUES',variables2^var in 'VALUES'],
    ['NSCC'='NGROUP',
     'MIN_NSCC'='MIN_SIZE',
     'MAX_NSCC'='MAX_SIZE',
     'NVERTEX'='NVAL'],
    []).

ctr_eval(
    group_skip_isolated_item,
    [automata(group_skip_isolated_item_a)]).

ctr_functional_dependency(group_skip_isolated_item,1,[5,6]).
```

```
ctr_functional_dependency(group_skip_isolated_item,2,[5,6]).

ctr_functional_dependency(group_skip_isolated_item,3,[5,6]).

ctr_functional_dependency(group_skip_isolated_item,4,[5,6]).

group_skip_isolated_item_a(
    NGROUP,
    MIN_SIZE,
    MAX_SIZE,
    NVAL,
    VARIABLES,
    VALUES) :-
        check_type(dvar,NGROUP),
        check_type(dvar,MIN_SIZE),
        check_type(dvar,MAX_SIZE),
        check_type(dvar,NVAL),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int]),
        length(VARIABLES,N),
        get_attr1(VALUES,VALS),
        NGROUP#>=0,
        MIN_SIZE#>=0,
        MIN_SIZE#\=1,
        MAX_SIZE#>=MIN_SIZE,
        MAX_SIZE#\=1,
        NVAL#>=MAX_SIZE,
        NVAL#>=NGROUP,
        NVAL#=<N,
        all_different(VALS),
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        group_skip_isolated_item_signature(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        group_skip_isolated_item_ngroup(NGROUP,SIGNATURE),
        group_skip_isolated_item_min_size(MIN_SIZE,N,SIGNATURE),
        group_skip_isolated_item_max_size(MAX_SIZE,SIGNATURE),
        group_skip_isolated_item_nval(NVAL,SIGNATURE).

group_skip_isolated_item_ngroup(NGROUP,SIGNATURE) :-
        automaton(
            SIGNATURE,
            _50654,
            SIGNATURE,
```

```
            [source(s),sink(i),sink(j),sink(s)],
            [arc(s,0,s),
             arc(s,1,i),
             arc(i,0,s),
             arc(i,1,j,[C+1]),
             arc(j,1,j),
             arc(j,0,s)],
            [C],
            [0],
            [NGROUP]).

group_skip_isolated_item_min_size(MIN_SIZE,NVAR,SIGNATURE) :-
        MIN_SIZE#=min(C1,D1),
        automaton(
            SIGNATURE,
            _50936,
            SIGNATURE,
            [source(s),sink(t),sink(r),sink(s)],
            [arc(s,0,s),
             arc(s,1,t),
             arc(t,0,s),
             arc(t,1,r,[C,2]),
             arc(r,0,s,[min(C,D),D]),
             arc(r,1,r,[C,D+1])],
            [C,D],
            [NVAR,0],
            [C1,D1]).

group_skip_isolated_item_max_size(MAX_SIZE,SIGNATURE) :-
        MAX_SIZE#=max(C1,D1),
        automaton(
            SIGNATURE,
            _50943,
            SIGNATURE,
            [source(s),sink(t),sink(r),sink(s)],
            [arc(s,0,s),
             arc(s,1,t),
             arc(t,0,s),
             arc(t,1,r,[max(C,2),2]),
             arc(r,0,s),
             arc(r,1,r,[max(C,D+1),D+1])],
            [C,D],
            [0,0],
            [C1,D1]).

group_skip_isolated_item_nval(NVAL,SIGNATURE) :-
```

```
        automaton(
            SIGNATURE,
            _50570,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[C+1])],
            [C],
            [0],
            [NVAL]).

group_skip_isolated_item_signature([],[],_49967).

group_skip_isolated_item_signature(
    [[var-VAR]|VARs],
    [S|Ss],
    SET_OF_VALUES) :-
        VAR in_set SET_OF_VALUES#<=>S,
        group_skip_isolated_item_signature(
            VARs,
            Ss,
            SET_OF_VALUES).
```

B.176 gt

◇ META-DATA:

```
ctr_predefined(gt).

ctr_date(gt,['20070821']).

ctr_origin(gt,'Arithmetic.',[]).

ctr_synonyms(gt,[rel,xgty]).

ctr_arguments(gt,['VAR1'-dvar,'VAR2'-dvar]).

ctr_example(gt,gt(8,1)).

ctr_exchangeable(
    gt,
    [vals(['VAR1'],int(>('VAR2')),=\=,all,dontcare),
     vals(['VAR2'],int(<('VAR1')),=\=,all,dontcare)]).

ctr_eval(gt,[builtin(gt_b)]).

gt_b(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#>VAR2.
```

B.177 highest_peak

◇ **META-DATA:**

```
ctr_date(highest_peak,['20040530']).

ctr_origin(highest_peak,'Derived from %c.',[peak]).

ctr_arguments(
    highest_peak,
    ['HEIGHT'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(highest_peak,[required('VARIABLES',var)]).

ctr_example(
    highest_peak,
    [highest_peak(
         8,
         [[var-1],
          [var-1],
          [var-4],
          [var-8],
          [var-6],
          [var-2],
          [var-7],
          [var-1]]),
     highest_peak(
         1,
         [[var-0],
          [var-1],
          [var-1],
          [var-0],
          [var-0],
          [var-1],
          [var-0],
          [var-1]])]).

ctr_typical(
    highest_peak,
    [size('VARIABLES')>2,
     range('VARIABLES'^var)>2,
     peak('VARIABLES'^var)>0]).

ctr_typical_model(highest_peak,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(highest_peak,[items('VARIABLES',reverse)]).
```

```
ctr_eval(
    highest_peak,
    [checker(highest_peak_c),
     automaton(highest_peak_a),
     automaton_with_signature(highest_peak_a_s)]).

ctr_pure_functional_dependency(highest_peak,[]).

ctr_functional_dependency(highest_peak,1,[2]).

ctr_sol(highest_peak,2,0,2,9,[-1000000-9]).

ctr_sol(highest_peak,3,0,3,64,[-1000000-50,1-1,2-4,3-9]).

ctr_sol(
    highest_peak,
    4,
    0,
    4,
    625,
    [-1000000-295,1-11,2-44,3-99,4-176]).

ctr_sol(
    highest_peak,
    5,
    0,
    5,
    7776,
    [-1000000-1792,1-92,2-380,3-900,4-1712,5-2900]).

ctr_sol(
    highest_peak,
    6,
    0,
    6,
    117649,
    [-1000000-11088,
     1-697,
     2-3000,
     3-7587,
     4-15680,
     5-29125,
     6-50472]).

ctr_sol(
```

```
    highest_peak,
    7,
    0,
    7,
    2097152,
    [-1000000-69498,
     1-5036,
     2-22632,
     3-61389,
     4-138544,
     5-283250,
     6-540576,
     7-976227]).

ctr_sol(
    highest_peak,
    8,
    0,
    8,
    43046721,
    [-1000000-439791,
     1-35443,
     2-166208,
     3-484020,
     4-1195056,
     5-2693425,
     6-5665896,
     7-11233250,
     8-21133632]).

highest_peak_c(HEIGHT,VARIABLES) :-
        check_type(dvar,HEIGHT),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        MININT= -1000000,
        highest_peak_c(VARS,s,MININT,HEIGHT).

highest_peak_c([V1,V2|R],s,C,HEIGHT) :-
        V1>=V2,
        !,
        highest_peak_c([V2|R],s,C,HEIGHT).

highest_peak_c([_V1,V2|R],s,C,HEIGHT) :-
        !,
        highest_peak_c([V2|R],u,C,HEIGHT).
```

```
highest_peak_c([V1,V2|R],u,C,HEIGHT) :-
        V1=<V2,
        !,
        highest_peak_c([V2|R],u,C,HEIGHT).

highest_peak_c([V1,V2|R],u,C,HEIGHT) :-
        !,
        C1 is max(C,V1),
        highest_peak_c([V2|R],s,C1,HEIGHT).

highest_peak_c([_48268],_48265,HEIGHT,HEIGHT) :-
        !.

highest_peak_c([],_48262,HEIGHT,HEIGHT).

highest_peak_counters_check([V1,V2|R],s,C,[C|S]) :-
        V1>=V2,
        !,
        highest_peak_counters_check([V2|R],s,C,S).

highest_peak_counters_check([_V1,V2|R],s,C,[C|S]) :-
        !,
        highest_peak_counters_check([V2|R],u,C,S).

highest_peak_counters_check([V1,V2|R],u,C,[C|S]) :-
        V1=<V2,
        !,
        highest_peak_counters_check([V2|R],u,C,S).

highest_peak_counters_check([V1,V2|R],u,C,[C1|S]) :-
        !,
        C1 is max(C,V1),
        highest_peak_counters_check([V2|R],s,C1,S).

highest_peak_counters_check([V|R],init,C,[C|S]) :-
        !,
        highest_peak_counters_check([V|R],s,C,S).

highest_peak_counters_check([_48265],_48262,_48263,[]).

ctr_automaton_signature(
    highest_peak,
    highest_peak_a,
    pair_signature(2,signature)).

highest_peak_a(FLAG,HEIGHT,VARIABLES) :-
```

```
        pair_signature(VARIABLES,SIGNATURE),
        highest_peak_a_s(FLAG,HEIGHT,VARIABLES,SIGNATURE).

highest_peak_a_s(FLAG,HEIGHT,VARIABLES,SIGNATURE) :-
        check_type(dvar,HEIGHT),
        collection(VARIABLES,[dvar]),
        MININT= -1000000,
        pair_first_signature(VARIABLES,VARS),
        automaton(
            VARS,
            VAR1,
            SIGNATURE,
            [source(s),sink(u),sink(s)],
            [arc(s,2,s),
             arc(s,1,s),
             arc(s,0,u),
             arc(u,2,s,[max(C,VAR1)]),
             arc(u,1,u),
             arc(u,0,u)],
            [C],
            [MININT],
            [COUNT]),
        COUNT#=HEIGHT#<=>FLAG.
```

B.178 imply

◇ META-DATA:

```
ctr_date(imply,['20051226','20091016']).

ctr_origin(imply,'Logic',[]).

ctr_synonyms(imply,[rel,ifthen]).

ctr_arguments(
    imply,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    imply,
    ['VAR'>=0,
     'VAR'=<1,
     size('VARIABLES')=2,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    imply,
    [imply(1,[[var-0],[var-0]]),
     imply(1,[[var-0],[var-1]]),
     imply(0,[[var-1],[var-0]]),
     imply(1,[[var-1],[var-1]])]).

ctr_exchangeable(
    imply,
    [vals(['VAR','VARIABLES'^var],int(0 in 1),<,all,dontcare)]).

ctr_eval(imply,[reformulation(imply_r),automaton(imply_a)]).

ctr_pure_functional_dependency(imply,[]).

ctr_functional_dependency(imply,1,[2]).

ctr_sol(imply,2,0,2,4,[0-1,1-3]).

ctr_sol(imply,3,0,3,0,[]).

ctr_sol(imply,4,0,4,0,[]).
```

```
ctr_sol(imply,5,0,5,0,[]).

ctr_sol(imply,6,0,6,0,[]).

ctr_sol(imply,7,0,7,0,[]).

ctr_sol(imply,8,0,8,0,[]).

imply_r(VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,2),
        get_attr1(VARIABLES,VARS),
        VARS=[VAR1,VAR2],
        VAR#<=>VAR1#=>VAR2.

imply_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,2),
        get_attr1(VARIABLES,LIST),
        append([VAR],LIST,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _41518,
            LIST_VARIABLES,
            [source(s),sink(t)],
            [arc(s,0,i),
             arc(s,1,j),
             arc(i,1,k),
             arc(j,0,t),
             arc(j,1,l),
             arc(k,0,t),
             arc(l,1,t),
             arc(t,0,t),
             arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.179 in

◇ META-DATA:

```
ctr_date(in,['20030820','20040530','20060810']).

ctr_origin(in,'Domain definition.',[]).

ctr_synonyms(in,[dom,in_set,member]).

ctr_arguments(in,['VAR'-dvar,'VALUES'-collection(val-int)]).

ctr_restrictions(
    in,
    [size('VALUES')>0,
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(in,3 in[[val-1],[val-3]]).

ctr_typical(in,[size('VALUES')>1]).

ctr_exchangeable(
    in,
    [items('VALUES',all),
     vals(['VAR'],int(['VAR','VALUES'^val]),=\=,all,dontcare),
     translate(['VAR','VALUES'^val])]).

ctr_derived_collections(
    in,
    [col('VARIABLES'-collection(var-dvar),[item(var-'VAR')])]).

ctr_graph(
    in,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['NARC'=1],
    []).

ctr_eval(in,[reformulation(in_r),automaton(in_a)]).

ctr_extensible(in,[],'VALUES',any).

in_r(VAR,VALUES) :-
```

```
        check_type(dvar,VAR),
        collection(VALUES,[int]),
        length(VALUES,L),
        L>0,
        get_attr1(VALUES,VALS),
        all_different(VALS),
        build_or_var_in_values(VALS,VAR,TERM),
        call(TERM).

in_a(FLAG,VAR,VALUES) :-
        check_type(dvar,VAR),
        collection(VALUES,[int]),
        length(VALUES,L),
        L>0,
        get_attr1(VALUES,VALS),
        all_different(VALS),
        in_signature(VALUES,SIGNATURE,VAR),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _50267,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

in_signature([],[],_48228).

in_signature([[val-VAL]|VALs],[S|Ss],VAR) :-
        VAR#=VAL#<=>S,
        in_signature(VALs,Ss,VAR).
```

B.180 in_interval

◇ **META-DATA:**

```
ctr_date(in_interval,['20060317','20060810']).

ctr_origin(in_interval,'Domain definition.',[]).

ctr_synonyms(in_interval,[dom,in]).

ctr_arguments(in_interval,['VAR'-dvar,'LOW'-int,'UP'-int]).

ctr_restrictions(in_interval,['LOW'=<'UP']).

ctr_example(in_interval,in_interval(3,2,5)).

ctr_typical(in_interval,['LOW'<'UP','VAR'>'LOW','VAR'<'UP']).

ctr_exchangeable(
    in_interval,
    [vals(['LOW'],int,>,dontcare,dontcare),
     vals(['UP'],int,<,dontcare,dontcare),
     vals(['VAR'],int('LOW' in 'UP'),=\=,dontcare,dontcare),
     translate(['VAR','LOW','UP'])]).

ctr_derived_collections(
    in_interval,
    [col('VARIABLE'-collection(var-dvar),[item(var-'VAR')]),
     col('INTERVAL'-collection(low-int,up-int),
         [item(low-'LOW',up-'UP')])]).

ctr_graph(
    in_interval,
    ['VARIABLE','INTERVAL'],
    2,
    ['PRODUCT'>>collection(variable,interval)],
    [variable^var>=interval^low,variable^var=<interval^up],
    ['NARC'=1],
    []).

ctr_eval(
    in_interval,
    [reformulation(in_interval_r),automaton(in_interval_a)]).

in_interval_r(VAR,LOW,UP) :-
        check_type(fdvar,VAR),
```

```
        check_type(int,LOW),
        check_type(int,UP),
        LOW=<UP,
        VAR#>=LOW,
        VAR#=<UP.

in_interval_a(FLAG,VAR,LOW,UP) :-
        check_type(fdvar,VAR),
        check_type(int,LOW),
        check_type(int,UP),
        LOW=<UP,
        VAR#>=LOW#/\VAR#=<UP#<=>S,
        AUTOMATON=
        automaton(
            [S],
            _44933,
            [S],
            [source(s),sink(t)],
            [arc(s,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.181 in_interval_reified

♢ Meta-Data:

```
ctr_predefined(in_interval_reified).

ctr_date(in_interval_reified,['20100916']).

ctr_origin(
    in_interval_reified,
    Reified version of %c.,
    [in_interval]).

ctr_synonyms(in_interval_reified,[dom_reified,in_reified]).

ctr_arguments(
    in_interval_reified,
    ['VAR'-dvar,'LOW'-int,'UP'-int,'B'-dvar]).

ctr_restrictions(
    in_interval_reified,
    ['LOW'=<'UP','B'>=0,'B'=<1]).

ctr_example(in_interval_reified,in_interval_reified(3,2,5,1)).

ctr_typical(
    in_interval_reified,
    ['VAR'=\='LOW','VAR'=\='UP','LOW'<'UP']).

ctr_exchangeable(
    in_interval_reified,
    [vals(['VAR'],comp('LOW' in 'UP'),=,dontcare,dontcare),
     translate(['VAR','LOW','UP'])]).

ctr_eval(
    in_interval_reified,
    [reformulation(in_interval_reified_r)]).

in_interval_reified_r(VAR,LOW,UP,B) :-
        check_type(dvar,VAR),
        check_type(int,LOW),
        check_type(int,UP),
        check_type(dvar(0,1),B),
        LOW=<UP,
        VAR in LOW..UP#<=>B.
```

B.182 in_intervals

◇ **META-DATA:**

```
ctr_predefined(in_intervals).

ctr_date(in_intervals,['20080610']).

ctr_origin(in_intervals,'Domain definition.',[]).

ctr_synonyms(in_intervals,[in]).

ctr_arguments(
    in_intervals,
    ['VAR'-dvar,'INTERVALS'-collection(low-int,up-int)]).

ctr_restrictions(
    in_intervals,
    [required('INTERVALS',[low,up]),
     'INTERVALS'^low=<'INTERVALS'^up,
     size('INTERVALS')>0]).

ctr_example(
    in_intervals,
    in_intervals(5,[[low-1,up-1],[low-3,up-5],[low-8,up-8]])).

ctr_typical(in_intervals,[size('INTERVALS')>1]).

ctr_exchangeable(
    in_intervals,
    [items('INTERVALS',all),
     vals(['INTERVALS'^low],int,>,dontcare,dontcare),
     vals(['INTERVALS'^up],int,<,dontcare,dontcare),
     translate(['VAR','INTERVALS'^low,'INTERVALS'^up])]).

ctr_eval(in_intervals,[reformulation(in_intervals_r)]).

ctr_extensible(in_intervals,[],'INTERVALS',any).

in_intervals_r(VAR,INTERVALS) :-
        check_type(dvar,VAR),
        collection(INTERVALS,[int,int]),
        length(INTERVALS,L),
        L>0,
        get_attr1(INTERVALS,LOWS),
        get_attr2(INTERVALS,UPS),
```

```
        check_lesseq(LOWS,UPS),
        in_intervals1(LOWS,UPS,VAR,TERM),
        call(TERM).

in_intervals1([],[],_27574,0).

in_intervals1([LOW|RLOW],[UP|RUP],VAR,VAR#>=LOW#/\VAR#=<UP#\/R) :-
        in_intervals1(RLOW,RUP,VAR,R).
```

B.183 in_relation

◊ **META-DATA:**

```
ctr_date(in_relation,['20030820','20040530','20060810']).

ctr_origin(
    in_relation,
    Constraint explicitly defined by tuples of values.,
    []).

ctr_synonyms(
    in_relation,
    [case,
     extension,
     extensional,
     extensional_support,
     extensional_supportva,
     extensional_supportmdd,
     extensional_supportstr,
     feastupleac,
     table]).

ctr_types(
    in_relation,
    ['TUPLE_OF_VARS'-collection(var-dvar),
     'TUPLE_OF_VALS'-collection(val-int)]).

ctr_arguments(
    in_relation,
    ['VARIABLES'-'TUPLE_OF_VARS',
     'TUPLES_OF_VALS'-collection(tuple-'TUPLE_OF_VALS')]).

ctr_restrictions(
    in_relation,
    [required('TUPLE_OF_VARS',var),
     size('TUPLE_OF_VARS')>=1,
     size('TUPLE_OF_VALS')>=1,
     size('TUPLE_OF_VALS')=size('VARIABLES'),
     required('TUPLE_OF_VALS',val),
     required('TUPLES_OF_VALS',tuple)]).

ctr_example(
    in_relation,
    in_relation(
        [[var-5],[var-3],[var-3]],
```

```
        [[tuple-[[val-5],[val-2],[val-3]]],
         [tuple-[[val-5],[val-2],[val-6]]],
         [tuple-[[val-5],[val-3],[val-3]]]]])).

ctr_typical(in_relation,[size('TUPLE_OF_VARS')>1]).

ctr_exchangeable(
    in_relation,
    [items('TUPLES_OF_VALS',all),
     items_sync('VARIABLES','TUPLES_OF_VALS'^tuple,all),
     vals(
         ['VARIABLES','TUPLES_OF_VALS'^tuple],
         int,
         =\=,
         all,
         dontcare)]).

ctr_derived_collections(
    in_relation,
    [col('TUPLES_OF_VARS'-collection(vec-'TUPLE_OF_VARS'),
         [item(vec-'VARIABLES')])]).

ctr_graph(
    in_relation,
    ['TUPLES_OF_VARS','TUPLES_OF_VALS'],
    2,
    ['PRODUCT'>>collection(tuples_of_vars,tuples_of_vals)],
    [vec_eq_tuple(tuples_of_vars^vec,tuples_of_vals^tuple)],
    ['NARC'>=1],
    []).

ctr_eval(in_relation,[reformulation(in_relation_r)]).

ctr_extensible(in_relation,[],'TUPLES_OF_VALS',any).

in_relation_r(VARIABLES,TUPLES_OF_VALS) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        collection(TUPLES_OF_VALS,[col(N,[int])]),
        get_attr1(VARIABLES,VARS),
        get_col_attr1(TUPLES_OF_VALS,1,TUPLES),
        table([VARS],TUPLES).
```

B.184 in_same_partition

◇ **META-DATA:**

```
ctr_date(in_same_partition,['20030820','20040530','20060810']).

ctr_origin(
    in_same_partition,
    Used for defining several entries of this catalog.,
    []).

ctr_types(in_same_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    in_same_partition,
    ['VAR1'-dvar,
     'VAR2'-dvar,
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    in_same_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    in_same_partition,
    in_same_partition(
        6,
        2,
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(in_same_partition,['VAR1'=\='VAR2']).

ctr_exchangeable(
    in_same_partition,
    [args([['VAR1','VAR2'],['PARTITIONS']]),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all)]).

ctr_derived_collections(
    in_same_partition,
```

```
    [col('VARIABLES'-collection(var-dvar),
         [item(var-'VAR1'),item(var-'VAR2')])]).

ctr_graph(
    in_same_partition,
    ['VARIABLES','PARTITIONS'],
    2,
    ['PRODUCT'>>collection(variables,partitions)],
    [variables^var in partitions^p],
    ['NSOURCE'=2,'NSINK'=1],
    []).

ctr_eval(
    in_same_partition,
    [reformulation(in_same_partition_r),
     automaton(in_same_partition_a)]).

ctr_extensible(in_same_partition,[],'PARTITIONS',any).

in_same_partition_r(VAR1,VAR2,PARTITIONS) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        length(PARTITIONS,P),
        P>1,
        collection_distinct(PARTITIONS,1),
        get_col_attr1(PARTITIONS,1,PVALS),
        in_same_partition1(PVALS,VAR1,VAR2,TERM),
        call(TERM).

in_same_partition1([],_45642,_45643,0).

in_same_partition1([VALS|R],VAR1,VAR2,TERM1#/\TERM2#\/TERM) :-
        build_or_var_in_values(VALS,VAR1,TERM1),
        build_or_var_in_values(VALS,VAR2,TERM2),
        in_same_partition1(R,VAR1,VAR2,TERM).

in_same_partition_a(FLAG,VAR1,VAR2,PARTITIONS) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        length(PARTITIONS,P),
        P>1,
        collection_distinct(PARTITIONS,1),
        in_same_partition_signature(
            PARTITIONS,
```

```
            SIGNATURE,
            VAR1,
            VAR2),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _48058,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

in_same_partition_signature([],[],_45643,_45644).

in_same_partition_signature(
    [[p-VALUES]|PARTITIONs],
    [S|Ss],
    VAR1,
    VAR2) :-
        get_attr1(VALUES,LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        VAR1 in_set SET_OF_VALUES#/\
        VAR2 in_set SET_OF_VALUES#<=>
        S,
        in_same_partition_signature(PARTITIONs,Ss,VAR1,VAR2).
```

B.185 in_set

◇ **META-DATA:**

```
ctr_predefined(in_set).

ctr_date(in_set,['20030820']).

ctr_origin(
    in_set,
    Used for defining constraints with set variables.,
    []).

ctr_synonyms(in_set,[dom,member]).

ctr_arguments(in_set,['VAL'-dvar,'SET'-svar]).

ctr_example(in_set,3 in_set{1,3}).
```

B.186 incomparable

◇ **META-DATA:**

```
ctr_predefined(incomparable).

ctr_date(incomparable,['20120202']).

ctr_origin(
    incomparable,
    Inspired by incomparable rectangles.,
    []).

ctr_synonyms(incomparable,[incomparables]).

ctr_arguments(
    incomparable,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    incomparable,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')>=1,
     size('VECTOR2')>=1,
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    incomparable,
    [incomparable([[var-16],[var-2]],[[var-4],[var-11]])]).

ctr_typical(incomparable,[size('VECTOR1')>1]).

ctr_exchangeable(
    incomparable,
    [items('VECTOR1',all),
     items('VECTOR2',all),
     args([['VECTOR1','VECTOR2']])]).

ctr_eval(
    incomparable,
    [reformulation(incomparable_r),checker(incomparable_c)]).

ctr_cond_imply(
    incomparable,
```

```
    disjoint,
    [size('VECTOR1')=2],
    [],
    [same('VECTOR1'),same('VECTOR2')]).

ctr_cond_imply(
    incomparable,
    int_value_precede_chain,
    [size('VECTOR1')=2],
    [],
    [same('VECTOR1'),same('VECTOR2')]).

incomparable_r(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        incomparable(VECT1,VECT2).

incomparable(U,V) :-
        length(U,N),
        length(V,N),
        N>1,
        length(PU,N),
        length(PV,N),
        domain(PU,1,N),
        domain(PV,1,N),
        get_minimum(U,MinU),
        get_maximum(U,MaxU),
        get_minimum(V,MinV),
        get_maximum(V,MaxV),
        length(SU,N),
        length(SV,N),
        domain(SU,MinU,MaxU),
        domain(SV,MinV,MaxV),
        sorting(U,PU,SU),
        sorting(V,PV,SV),
        incomparable(SU,SV,Cond1),
        incomparable(SV,SU,Cond2),
        call(Cond1),
        call(Cond2),
        append(U,V,UV),
        append(PU,PV,PUV),
        when(ground(UV),once(labeling([],PUV))).
```

```
incomparable([],[],0).

incomparable([U|R],[V|S],U#>V#\/T) :-
        incomparable(R,S,T).

incomparable_c(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[int]),
        collection(VECTOR2,[int]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        incomparablec(VECT1,VECT2).
```

B.187 increasing

◇ META-DATA:

```
ctr_date(increasing,['20040814','20060810','20091105']).

ctr_origin(increasing,'KOALOG',[]).

ctr_arguments(increasing,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(increasing,[required('VARIABLES',var)]).

ctr_example(
    increasing,
    increasing([[var-1],[var-1],[var-4],[var-8]])).

ctr_typical(
    increasing,
    [size('VARIABLES')>2,range('VARIABLES'^var)>1]).

ctr_typical_model(increasing,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(increasing,[translate(['VARIABLES'^var])]).

ctr_graph(
    increasing,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var=<variables2^var],
    ['NARC'=size('VARIABLES')-1],
    []).

ctr_eval(
    increasing,
    [checker(increasing_c),
     reformulation(increasing_r),
     automaton(increasing_a)]).

ctr_contractible(increasing,[],'VARIABLES',any).

ctr_sol(increasing,2,0,2,6,-).

ctr_sol(increasing,3,0,3,20,-).

ctr_sol(increasing,4,0,4,70,-).
```

```
ctr_sol(increasing,5,0,5,252,-).

ctr_sol(increasing,6,0,6,924,-).

ctr_sol(increasing,7,0,7,3432,-).

ctr_sol(increasing,8,0,8,12870,-).

increasing_c([[var-X],[var-Y]|_49102]) :-
        X>Y,
        !,
        fail.

increasing_c([]) :-
        !.

increasing_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        increasing_c1(VARS).

increasing_c1([X,Y|R]) :-
        !,
        X=<Y,
        increasing_c1([Y|R]).

increasing_c1([_49092]) :-
        !.

increasing_c1([]).

increasing_r([]) :-
        !.

increasing_r(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        increasing1(VARS).

increasing1([_49092]) :-
        !.

increasing1([V1,V2|R]) :-
        V1#=<V2,
        increasing1([V2|R]).
```

```
increasing_a(1,[]) :-
        !.

increasing_a(0,[]) :-
        !,
        fail.

increasing_a(FLAG,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        increasing_signature(VARIABLES,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _50235,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

increasing_signature([_49093],[]) :-
        !.

increasing_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
        S in 0..1,
        VAR1#=<VAR2#<=>S,
        increasing_signature([[var-VAR2]|VARs],Ss).
```

B.188 increasing_global_cardinality

◇ **META-DATA:**

```
ctr_date(increasing_global_cardinality,['20091015']).

ctr_origin(
    increasing_global_cardinality,
    Conjoin %c and %c.,
    [global_cardinality_low_up,increasing]).

ctr_synonyms(
    increasing_global_cardinality,
    [increasing_global_cardinality_low_up,
     increasing_gcc,
     increasing_gcc_low_up]).

ctr_arguments(
    increasing_global_cardinality,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
    increasing_global_cardinality,
    [required('VARIABLES',var),
     increasing('VARIABLES'),
     size('VALUES')>0,
     required('VALUES',[val,omin,omax]),
     distinct('VALUES',val),
     'VALUES'^omin>=0,
     'VALUES'^omax=<size('VARIABLES'),
     'VALUES'^omin=<'VALUES'^omax]).

ctr_example(
    increasing_global_cardinality,
    increasing_global_cardinality(
        [[var-3],[var-3],[var-6],[var-8]],
        [[val-3,omin-2,omax-3],
         [val-5,omin-0,omax-1],
         [val-6,omin-1,omax-2]])).

ctr_typical(
    increasing_global_cardinality,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
```

```
     'VALUES'^omin=<size('VARIABLES'),
     'VALUES'^omax>0,
     'VALUES'^omax=<size('VARIABLES'),
     size('VARIABLES')>size('VALUES')]).

ctr_typical_model(
    increasing_global_cardinality,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    increasing_global_cardinality,
    [items('VALUES',all)]).

ctr_graph(
    increasing_global_cardinality,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val],
    ['NVERTEX'>='VALUES'^omin,'NVERTEX'=<'VALUES'^omax],
    []).

ctr_eval(
    increasing_global_cardinality,
    [reformulation(increasing_global_cardinality_r),
     automaton(increasing_global_cardinality_a)]).

ctr_functional_dependency(increasing_nvalue,1,[2]).

increasing_global_cardinality_r(VARIABLES,VALUES) :-
        eval(increasing(VARIABLES)),
        eval(global_cardinality_low_up(VARIABLES,VALUES)).

increasing_global_cardinality_a(FLAG,VARIABLES,VALUES) :-
        increasing_global_cardinality_get_a(
            VARIABLES,
            VALUES,
            AUTOMATON,
            ALPHABET),
        automaton_bool(FLAG,ALPHABET,AUTOMATON).

increasing_global_cardinality_get_a(
    VARIABLES,
    VALUES,
    AUTOMATON,
    ALPHABET) :-
```

```
length(VARIABLES,N),
collection(VARIABLES,[dvar]),
collection(VALUES,[int,int(0,N),int(0,N)]),
length(VALUES,M),
M>0,
sort_collection(VALUES,val,SVALUES),
get_attr1(VARIABLES,VARS),
get_attr1(SVALUES,VALS),
get_attr2(SVALUES,OMINS),
get_attr3(SVALUES,OMAXS),
all_different(VALS),
check_lesseq(OMINS,OMAXS),
increasing_gcc_normalize(VALS,OMAXS,VARS),
get_minimum(VARS,MINVARS),
get_maximum(VARS,MAXVARS),
get_minimum(VALS,MINVALS),
get_maximum(VALS,MAXVALS),
MIN is min(MINVARS,MINVALS),
MAX is max(MAXVARS,MAXVALS),
get_sum(OMINS,SUM_OMINS),
REST is N-SUM_OMINS,
REST>=0,
increasing_global_cardinality_complete_values(
    MIN,
    MAX,
    SVALUES,
    REST,
    CVALUES,
    SUM_OMAXS),
reverse(CVALUES,RVALUES),
increasing_global_cardinality_term_states(
    RVALUES,
    SUM_OMAXS,
    TERMINALS),
append([source(0)],TERMINALS,STATES),
increasing_global_cardinality_source_trans(
    CVALUES,
    1,
    TRANSITIONS_FROM_SOURCE),
increasing_global_cardinality_horiz_trans(
    CVALUES,
    1,
    TRANSITIONS_HORIZONTAL),
increasing_global_cardinality_vert_trans(
    CVALUES,
    1,
```

```
            TRANSITIONS_VERTICAL),
        append(
            TRANSITIONS_FROM_SOURCE,
            TRANSITIONS_HORIZONTAL,
            T1),
        append(T1,TRANSITIONS_VERTICAL,ALL_TRANSITIONS),
        AUTOMATON=
        automaton(
            VARS,
            _55755,
            VARS,
            STATES,
            ALL_TRANSITIONS,
            [],
            [],
            []),
        append(VARS,VALS,ALL),
        union_dom_list_int(ALL,ALPHABET).

increasing_gcc_normalize([],[],_47567) :-
        !.

increasing_gcc_normalize([VAL|R],[OMAX|S],VARS) :-
        (   OMAX=0 ->
            remove_value_from_vars(VARS,VAL)
        ;   true
        ),
        increasing_gcc_normalize(R,S,VARS).

increasing_global_cardinality_complete_values(
    MIN,
    MAX,
    VALUES,
    _REST,
    VALUES,
    0) :-
        MIN>MAX,
        (   VALUES=[] ->
            true
        ;   write(problem),
            nl,
            abort
        ),
        !.

increasing_global_cardinality_complete_values(
```

```
    MIN,
    MAX,
    [],
    REST,
    [[constrained-CTR,val-MIN,omin-0,omax-OOMAX]|S],
    SUM) :-
        MIN=<MAX,
        !,
        (   REST>1 ->
            CTR=0,
            OOMAX=1
        ;   CTR=1,
            OOMAX is max(1,REST)
        ),
        MIN1 is MIN+1,
        increasing_global_cardinality_complete_values(
            MIN1,
            MAX,
            [],
            REST,
            S,
            TSUM),
        SUM is TSUM+OOMAX.

increasing_global_cardinality_complete_values(
    MIN,
    MAX,
    [[val-VAL,omin-OMIN,omax-OMAX]|R],
    REST,
    [[constrained-CTR,val-VAL,omin-OMIN,omax-OOMAX]|S],
    SUM) :-
        MIN=<MAX,
        MIN=VAL,
        !,
        (   OMAX>1,
            OMAX>=REST+OMIN ->
            CTR=0,
            OOMAX is max(1,OMIN)
        ;   CTR=1,
            OOMAX is max(1,OMAX)
        ),
        MIN1 is MIN+1,
        increasing_global_cardinality_complete_values(
            MIN1,
            MAX,
            R,
```

```
            REST,
            S,
            TSUM),
        SUM is TSUM+OOMAX.

increasing_global_cardinality_complete_values(
    MIN,
    MAX,
    [[val-VAL,omin-OMIN,omax-OMAX]|R],
    REST,
    [[constrained-CTR,val-MIN,omin-0,omax-OOMAX]|S],
    SUM) :-
        MIN=<MAX,
        MIN<VAL,
        (   REST>1 ->
            CTR=0,
            OOMAX=1
        ;   CTR=1,
            OOMAX is max(1,REST)
        ),
        MIN1 is MIN+1,
        increasing_global_cardinality_complete_values(
            MIN1,
            MAX,
            [[val-VAL,omin-OMIN,omax-OMAX]|R],
            REST,
            S,
            TSUM),
        SUM is TSUM+OOMAX.

increasing_global_cardinality_term_states([],_47563,[]).

increasing_global_cardinality_term_states(
    [[constrained-_47574,val-_VAL,omin-OMIN,omax-OMAX]|R],
    LAST_STATE_ID,
    RES) :-
        I is LAST_STATE_ID-OMAX+max(1,OMIN),
        increasing_global_cardinality_term_states1(
            I,
            LAST_STATE_ID,
            TERMS),
        LAST_STATE_ID1 is LAST_STATE_ID-OMAX,
        (   OMIN=0 ->
            increasing_global_cardinality_term_states(
                R,
                LAST_STATE_ID1,
```

```
                S),
            append(S,TERMS,RES)
        ;   RES=TERMS
        ).

increasing_global_cardinality_term_states1(I,MAX,[]) :-
        I>MAX,
        !.

increasing_global_cardinality_term_states1(I,MAX,[sink(I)|R]) :-
        I=<MAX,
        I1 is I+1,
        increasing_global_cardinality_term_states1(I1,MAX,R).

increasing_global_cardinality_source_trans([],_47563,[]).

increasing_global_cardinality_source_trans(
    [[constrained-_47574,val-VAL,omin-OMIN,omax-OMAX]|R],
    CUR_ID,
    [arc(0,VAL,CUR_ID)|S]) :-
        CUR_ID1 is CUR_ID+OMAX,
        (   OMIN=0 ->
            increasing_global_cardinality_source_trans(
                R,
                CUR_ID1,
                S)
        ;   S=[]
        ).

increasing_global_cardinality_horiz_trans([],_47563,[]).

increasing_global_cardinality_horiz_trans(
    [[constrained-CTR,val-VAL,omin-_47588,omax-OMAX]|R],
    CUR_ID,
    RESULT) :-
        increasing_global_cardinality_horiz_trans1(
            1,
            OMAX,
            CTR,
            VAL,
            CUR_ID,
            TR),
        CUR_ID1 is CUR_ID+OMAX,
        increasing_global_cardinality_horiz_trans(R,CUR_ID1,S),
        append(TR,S,RESULT).
```

```
increasing_global_cardinality_horiz_trans1(
    I,
    OMAX,
    1,
    _47956,
    _48003,
    []) :-
        I>=OMAX,
        !.

increasing_global_cardinality_horiz_trans1(
    I,
    OMAX,
    0,
    VAL,
    ID,
    [arc(ID,VAL,ID)]) :-
        I>=OMAX,
        !.

increasing_global_cardinality_horiz_trans1(
    I,
    OMAX,
    CTR,
    VAL,
    ID,
    [arc(ID,VAL,ID1)|R]) :-
        I<OMAX,
        ID1 is ID+1,
        I1 is I+1,
        increasing_global_cardinality_horiz_trans1(
            I1,
            OMAX,
            CTR,
            VAL,
            ID1,
            R).

increasing_global_cardinality_vert_trans([_47568],_47566,[]) :-
        !.

increasing_global_cardinality_vert_trans(
    [[constrained-_47574,val-_VAL,omin-OMIN,omax-OMAX]|R],
    CUR_ID,
    RESULT) :-
        I is CUR_ID+max(0,OMIN-1),
```

```
        CUR_ID1 is CUR_ID+OMAX,
        increasing_global_cardinality_vert_trans1(
            R,
            CUR_ID1,
            I,
            CUR_ID1,
            S),
        increasing_global_cardinality_vert_trans(R,CUR_ID1,T),
        append(S,T,RESULT).

increasing_global_cardinality_vert_trans1(
    [],
    _47842,
    _47889,
    _47936,
    []).

increasing_global_cardinality_vert_trans1(
    [[constrained-_47576,val-VAL,omin-OMIN,omax-OMAX]|R],
    CUR_ID,
    I,
    MAX,
    RESULT) :-
        increasing_global_cardinality_vert_trans2(
            I,
            MAX,
            CUR_ID,
            VAL,
            RES1),
        CUR_ID1 is CUR_ID+OMAX,
        (   OMIN=0 ->
            increasing_global_cardinality_vert_trans1(
                R,
                CUR_ID1,
                I,
                MAX,
                RES2),
            append(RES1,RES2,RESULT)
        ;   RESULT=RES1
        ).

increasing_global_cardinality_vert_trans2(
    MAX,
    MAX,
    _47900,
    _47947,
```

```
    []) :-
        !.

increasing_global_cardinality_vert_trans2(
    I,
    MAX,
    CUR_ID,
    VAL,
    [arc(I,VAL,CUR_ID)|R]) :-
        I<MAX,
        I1 is I+1,
        increasing_global_cardinality_vert_trans2(
            I1,
            MAX,
            CUR_ID,
            VAL,
            R).
```

B.189 increasing_nvalue

◊ **META-DATA:**

```
ctr_date(increasing_nvalue,['20091104']).

ctr_origin(
    increasing_nvalue,
    Conjoin %c and %c.,
    [nvalue,increasing]).

ctr_arguments(
    increasing_nvalue,
    ['NVAL'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    increasing_nvalue,
    ['NVAL'>=min(1,size('VARIABLES')),
     'NVAL'=<size('VARIABLES'),
     required('VARIABLES',var),
     increasing('VARIABLES')]).

ctr_example(
    increasing_nvalue,
    [increasing_nvalue(
         2,
         [[var-6],[var-6],[var-8],[var-8],[var-8]]),
     increasing_nvalue(
         1,
         [[var-6],[var-6],[var-6],[var-6],[var-6]]),
     increasing_nvalue(
         5,
         [[var-0],[var-2],[var-3],[var-6],[var-7]])]).

ctr_typical(
    increasing_nvalue,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_typical_model(increasing_nvalue,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    increasing_nvalue,
    [translate(['VARIABLES'^var])]).

ctr_graph(
    increasing_nvalue,
```

```
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSCC'='NVAL'],
    ['EQUIVALENCE']).

ctr_eval(
    increasing_nvalue,
    [checker(increasing_nvalue_c),
     builtin(increasing_nvalue_b),
     reformulation(increasing_nvalue_r),
     automata(increasing_nvalue_a)]).

ctr_sol(increasing_nvalue,2,0,2,6,[1-3,2-3]).

ctr_sol(increasing_nvalue,3,0,3,20,[1-4,2-12,3-4]).

ctr_sol(increasing_nvalue,4,0,4,70,[1-5,2-30,3-30,4-5]).

ctr_sol(increasing_nvalue,5,0,5,252,[1-6,2-60,3-120,4-60,5-6]).

ctr_sol(
    increasing_nvalue,
    6,
    0,
    6,
    924,
    [1-7,2-105,3-350,4-350,5-105,6-7]).

ctr_sol(
    increasing_nvalue,
    7,
    0,
    7,
    3432,
    [1-8,2-168,3-840,4-1400,5-840,6-168,7-8]).

ctr_sol(
    increasing_nvalue,
    8,
    0,
    8,
    12870,
    [1-9,2-252,3-1764,4-4410,5-4410,6-1764,7-252,8-9]).
```

```
increasing_nvalue_c(_71212,[[var-X],[var-Y]|_71224]) :-
        X>Y,
        !,
        fail.

increasing_nvalue_c(0,[]) :-
        !.

increasing_nvalue_c(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        (   N=1 ->
            NVAL=1
        ;   VARS=[VAR|R],
            increasing_nvalue_c(R,VAR,1,NVAL)
        ).

increasing_nvalue_c([],_71213,NVAL,NVAL) :-
        !.

increasing_nvalue_c([V|R],Prev,Count,NVAL) :-
        Prev=<V,
        (   V=Prev ->
            Count1=Count
        ;   Count1 is Count+1
        ),
        increasing_nvalue_c(R,V,Count1,NVAL).

increasing_nvalue_counters_check([],_71213,_71214,[]) :-
        !.

increasing_nvalue_counters_check([V|R],Prev,Count,[Count1|S]) :-
        integer(Prev),
        Prev=<V,
        !,
        (   V=Prev ->
            Count1=Count
        ;   Count1 is Count+1
        ),
        increasing_nvalue_counters_check(R,V,Count1,S).

increasing_nvalue_counters_check(
    [_71216|R],
    _71611,
```

```
    _71658,
    [-|S]) :-
        increasing_nvalue_counters_check(R,-,_71227,S).

decreasing_nvalue_counters_check([],_71213,_71214,[]) :-
        !.

decreasing_nvalue_counters_check([V|R],Prev,Count,[Count1|S]) :-
        integer(Prev),
        Prev>=V,
        !,
        (   V=Prev ->
            Count1=Count
        ;   Count1 is Count+1
        ),
        decreasing_nvalue_counters_check(R,V,Count1,S).

decreasing_nvalue_counters_check(
    [_71216|R],
    _71611,
    _71658,
    [-|S]) :-
        decreasing_nvalue_counters_check(R,-,_71227,S).

increasing_nvalue_b(0,[]) :-
        !.

increasing_nvalue_b(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        NVAL#>=min(1,N),
        NVAL#=<N,
        increasing_nvalue_b1(VARS,S),
        call(NVAL#=S).

increasing_nvalue_b1([_71214],1) :-
        !.

increasing_nvalue_b1([V1,V2|R],B+S) :-
        V1#=<V2,
        B#<=>V1#<V2,
        increasing_nvalue_b1([V2|R],S).

increasing_nvalue_r(0,[]) :-
```

```
        !.

increasing_nvalue_r(NVAL,VARIABLES) :-
        eval(increasing(VARIABLES)),
        eval(nvalue(NVAL,VARIABLES)).

increasing_nvalue_a(0,[]) :-
        !.

increasing_nvalue_a(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        NVAL#>=min(1,N),
        NVAL#=<N,
        get_minimum(VARS,MINVARS),
        get_maximum(VARS,MAXVARS),
        SIZE is MAXVARS-MINVARS+1,
        fd_min(NVAL,MINNVAL),
        fd_max(NVAL,MAXNVAL),
        D is min(N,min(SIZE,MAXNVAL)),
        fd_set(NVAL,SVAL),
        fdset_to_list(SVAL,VALUES),
        increasing_nvalue_states(VALUES,SIZE,MINNVAL,STATES),
        gen_automaton_state(s,0,0,S_00),
        increasing_nvalue_class1(
            1,
            SIZE,
            MINNVAL,
            MINVARS,
            S_00,
            TRANS1),
        increasing_nvalue_class2(
            1,
            D,
            SIZE,
            MINNVAL,
            MINVARS,
            TRANS2),
        increasing_nvalue_class3(
            1,
            D,
            SIZE,
            MINNVAL,
            MINVARS,
```

```
            TRANS3),
        append(TRANS1,TRANS2,TRANS12),
        append(TRANS12,TRANS3,ALL_TRANSITIONS),
        automaton(
            VARS,
            _76799,
            VARS,
            STATES,
            ALL_TRANSITIONS,
            [],
            [],
            []),
        eval(nvalue(NVAL,VARIABLES)).

increasing_nvalue_states([],_71213,_71214,[source(S_00)]) :-
        gen_automaton_state(s,0,0,S_00).

increasing_nvalue_states([V|R],SIZE,MINNVAL,STATES) :-
        increasing_nvalue_states1(V,SIZE,V,MINNVAL,STATES1),
        increasing_nvalue_states(R,SIZE,MINNVAL,STATES2),
        append(STATES1,STATES2,STATES).

increasing_nvalue_states1(J,SIZE,_71214,_71215,[]) :-
        J>SIZE,
        !.

increasing_nvalue_states1(J,SIZE,I,MINNVAL,[sink(S_IJ)|STATES]) :-
        J=<SIZE,
        I_SIZE_J is I+SIZE-J,
        I_SIZE_J>=MINNVAL,
        !,
        gen_automaton_state(s,I,J,S_IJ),
        J1 is J+1,
        increasing_nvalue_states1(J1,SIZE,I,MINNVAL,STATES).

increasing_nvalue_states1(J,SIZE,I,MINNVAL,STATES) :-
        J=<SIZE,
        J1 is J+1,
        increasing_nvalue_states1(J1,SIZE,I,MINNVAL,STATES).

increasing_nvalue_class1(J,SIZE,_71214,_71215,_71216,[]) :-
        J>SIZE,
        !.

increasing_nvalue_class1(
    J,
```

```
    SIZE,
    MINNVAL,
    MINVARS,
    S_00,
    [arc(S_00,LABEL,S_1J)|TRANS]) :-
        J=<SIZE,
        I_SIZE_J is 1+SIZE-J,
        I_SIZE_J>=MINNVAL,
        !,
        gen_automaton_state(s,1,J,S_1J),
        LABEL is MINVARS+J-1,
        J1 is J+1,
        increasing_nvalue_class1(
            J1,
            SIZE,
            MINNVAL,
            MINVARS,
            S_00,
            TRANS).

increasing_nvalue_class1(J,SIZE,MINNVAL,MINVARS,S_00,TRANS) :-
        J=<SIZE,
        J1 is J+1,
        increasing_nvalue_class1(
            J1,
            SIZE,
            MINNVAL,
            MINVARS,
            S_00,
            TRANS).

increasing_nvalue_class2(I,D,_SIZE,_MINNVAL,_MINVARS,[]) :-
        I>D,
        !.

increasing_nvalue_class2(I,D,SIZE,MINNVAL,MINVARS,TRANS) :-
        I=<D,
        increasing_nvalue_class21(
            I,
            SIZE,
            I,
            MINNVAL,
            MINVARS,
            TRANS1),
        I1 is I+1,
        increasing_nvalue_class2(
```

```
            I1,
            D,
            SIZE,
            MINNVAL,
            MINVARS,
            TRANS2),
        append(TRANS1,TRANS2,TRANS).

increasing_nvalue_class21(J,SIZE,_I,_MINNVAL,_MINVARS,[]) :-
        J>SIZE,
        !.

increasing_nvalue_class21(
    J,
    SIZE,
    I,
    MINNVAL,
    MINVARS,
    [arc(S_IJ,LABEL,S_IJ)|TRANS]) :-
        J=<SIZE,
        I_SIZE_J is I+SIZE-J,
        I_SIZE_J>=MINNVAL,
        !,
        gen_automaton_state(s,I,J,S_IJ),
        LABEL is MINVARS+J-1,
        J1 is J+1,
        increasing_nvalue_class21(
            J1,
            SIZE,
            I,
            MINNVAL,
            MINVARS,
            TRANS).

increasing_nvalue_class21(J,SIZE,I,MINNVAL,MINVARS,TRANS) :-
        J=<SIZE,
        J1 is J+1,
        increasing_nvalue_class21(
            J1,
            SIZE,
            I,
            MINNVAL,
            MINVARS,
            TRANS).

increasing_nvalue_class3(I,D,_71214,_71215,_71216,[]) :-
```

```
        I>=D,
        !.

increasing_nvalue_class3(I,D,SIZE,MINNVAL,MINVARS,TRANS) :-
        I<D,
        increasing_nvalue_class31(
            I,
            SIZE,
            I,
            MINNVAL,
            MINVARS,
            TRANS1),
        I1 is I+1,
        increasing_nvalue_class3(
            I1,
            D,
            SIZE,
            MINNVAL,
            MINVARS,
            TRANS2),
        append(TRANS1,TRANS2,TRANS).

increasing_nvalue_class31(J,SIZE,_71214,_71215,_71216,[]) :-
        J>SIZE,
        !.

increasing_nvalue_class31(J,SIZE,I,MINNVAL,MINVARS,TRANS) :-
        J=<SIZE,
        I_SIZE_J is I+SIZE-J,
        I_SIZE_J>=MINNVAL,
        !,
        gen_automaton_state(s,I,J,S_IJ),
        J1 is J+1,
        increasing_nvalue_class32(
            J1,
            SIZE,
            I,
            J,
            S_IJ,
            MINNVAL,
            MINVARS,
            TRANS1),
        increasing_nvalue_class31(
            J1,
            SIZE,
            I,
```

```
            MINNVAL,
            MINVARS,
            TRANS2),
        append(TRANS1,TRANS2,TRANS).

increasing_nvalue_class31(J,SIZE,I,MINNVAL,MINVARS,TRANS) :-
        J=<SIZE,
        J1 is J+1,
        increasing_nvalue_class31(
            J1,
            SIZE,
            I,
            MINNVAL,
            MINVARS,
            TRANS).

increasing_nvalue_class32(
    K,
    SIZE,
    _71564,
    _71611,
    _71658,
    _71705,
    _71752,
    []) :-
        K>SIZE,
        !.

increasing_nvalue_class32(
    K,
    SIZE,
    I,
    J,
    S_IJ,
    MINNVAL,
    MINVARS,
    [arc(S_IJ,LABEL,S_I1K)|TRANS]) :-
        K=<SIZE,
        I1 is I+1,
        I1_SIZE_K is I1+SIZE-K,
        I1_SIZE_K>=MINNVAL,
        !,
        gen_automaton_state(s,I1,K,S_I1K),
        LABEL is MINVARS+K-1,
        K1 is K+1,
        increasing_nvalue_class32(
```

```
            K1,
            SIZE,
            I,
            J,
            S_IJ,
            MINNVAL,
            MINVARS,
            TRANS).

increasing_nvalue_class32(
    K,
    SIZE,
    I,
    J,
    S_IJ,
    MINNVAL,
    MINVARS,
    TRANS) :-
        K=<SIZE,
        K1 is K+1,
        increasing_nvalue_class32(
            K1,
            SIZE,
            I,
            J,
            S_IJ,
            MINNVAL,
            MINVARS,
            TRANS).
```

B.190 increasing_nvalue_chain

◇ META-DATA:

```
ctr_date(increasing_nvalue_chain,['20091118']).

ctr_origin(
    increasing_nvalue_chain,
    Derived from %c.,
    [increasing_nvalue]).

ctr_arguments(
    increasing_nvalue_chain,
    ['NVAL'-dvar,'VARIABLES'-collection(b-dvar,var-dvar)]).

ctr_restrictions(
    increasing_nvalue_chain,
    ['NVAL'>=min(1,size('VARIABLES')),
     'NVAL'=<size('VARIABLES'),
     required('VARIABLES',[b,var]),
     'VARIABLES'^b>=0,
     'VARIABLES'^b=<1]).

ctr_example(
    increasing_nvalue_chain,
    increasing_nvalue_chain(
        6,
        [[b-0,var-2],
         [b-1,var-4],
         [b-1,var-4],
         [b-1,var-4],
         [b-0,var-4],
         [b-1,var-8],
         [b-0,var-1],
         [b-0,var-7],
         [b-1,var-7]])).

ctr_typical(
    increasing_nvalue_chain,
    [size('VARIABLES')>1,
     range('VARIABLES'^b)>1,
     range('VARIABLES'^var)>1]).

ctr_graph(
    increasing_nvalue_chain,
    ['VARIABLES'],
```

```
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables2^b=0#\/variables1^var=<variables2^var],
    ['NARC'=size('VARIABLES')-1],
    []).

ctr_graph(
    increasing_nvalue_chain,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables2^b=0#\/variables1^var<variables2^var],
    ['NARC'='NVAL'-1],
    []).

ctr_eval(
    increasing_nvalue_chain,
    [reformulation(increasing_nvalue_chain_r)]).

increasing_nvalue_chain_r(_42781,_42782).
```

B.191 increasing_peak

◊ META-DATA:

```
ctr_date(increasing_peak,['20130209']).

ctr_origin(
    increasing_peak,
    Derived from %c and %c.,
    [peak,increasing]).

ctr_arguments(
    increasing_peak,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    increasing_peak,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    increasing_peak,
    increasing_peak(
        [[var-1],
         [var-5],
         [var-5],
         [var-3],
         [var-5],
         [var-2],
         [var-2],
         [var-7],
         [var-4]])).

ctr_typical(
    increasing_peak,
    [size('VARIABLES')>=7,
     range('VARIABLES'^var)>1,
     peak('VARIABLES'^var)>=3]).

ctr_typical_model(increasing_peak,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    increasing_peak,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    increasing_peak,
```

```
    [checker(increasing_peak_c),
     automaton(increasing_peak_a),
     automaton_with_signature(increasing_peak_a_s)]).

ctr_contractible(increasing_peak,[],'VARIABLES',prefix).

ctr_contractible(increasing_peak,[],'VARIABLES',suffix).

ctr_cond_imply(
    increasing_peak,
    not_all_equal,
    [peak('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(increasing_peak,2,0,2,9,-).

ctr_sol(increasing_peak,3,0,3,64,-).

ctr_sol(increasing_peak,4,0,4,625,-).

ctr_sol(increasing_peak,5,0,5,7553,-).

ctr_sol(increasing_peak,6,0,6,105798,-).

ctr_sol(increasing_peak,7,0,7,1666878,-).

ctr_sol(increasing_peak,8,0,8,29090469,-).

increasing_peak_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        L>0,
        get_attr1(VARIABLES,VARS),
        increasing_peak_c(VARS,s,0).

increasing_peak_c([V1,V2|R],s,A) :-
        V1>=V2,
        !,
        increasing_peak_c([V2|R],s,A).

increasing_peak_c([_32458,V2|R],s,A) :-
        !,
        increasing_peak_c([V2|R],u,A).

increasing_peak_c([V1,V2|R],u,A) :-
```

```
        V1=<V2,
        !,
        increasing_peak_c([V2|R],u,A).

increasing_peak_c([V1,V2|R],u,_32457) :-
        !,
        increasing_peak_c([V2|R],v,V1).

increasing_peak_c([V1,V2|R],v,A) :-
        V1>=V2,
        !,
        increasing_peak_c([V2|R],v,A).

increasing_peak_c([_32458,V2|R],v,A) :-
        !,
        increasing_peak_c([V2|R],w,A).

increasing_peak_c([V1,V2|R],w,A) :-
        V1=<V2,
        !,
        increasing_peak_c([V2|R],w,A).

increasing_peak_c([V1,V2|R],w,A) :-
        !,
        A=<V1,
        increasing_peak_c([V2|R],v,V1).

increasing_peak_c([_32458],_32456,_32457) :-
        !.

increasing_peak_c([],_32453,_32454).

ctr_automaton_signature(
    increasing_peak,
    increasing_peak_a,
    pair_signature(1,signature)).

increasing_peak_a(FLAG,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        increasing_peak_a_s(FLAG,VARIABLES,SIGNATURE).

increasing_peak_a_s(FLAG,VARIABLES,SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        L>=0,
        pair_first_signature(VARIABLES,VARS),
```

```
automaton(
    VARS,
    Vi,
    SIGNATURE,
    [source(s),sink(s),sink(u),sink(v),sink(w)],
    [arc(s,2,s),
     arc(s,1,s),
     arc(s,0,u),
     arc(u,2,v,[Vi,F]),
     arc(u,1,u),
     arc(u,0,u),
     arc(v,2,v),
     arc(v,1,v),
     arc(v,0,w),
     arc(w,2,v,(A#=<Vi->[Vi,F])),
     arc(w,2,v,(A#>Vi->[A,0])),
     arc(w,1,w),
     arc(w,0,w)],
    [A,F],
    [0,1],
    [_A,FLAG]).
```

B.192 increasing_sum

◇ META-DATA:

```
ctr_predefined(increasing_sum).

ctr_date(increasing_sum,['20110617']).

ctr_origin(
    increasing_sum,
    Conjoin %c and %c.,
    [increasing,sum_ctr]).

ctr_synonyms(
    increasing_sum,
    [increasing_sum_ctr,increasing_sum_eq]).

ctr_arguments(
    increasing_sum,
    ['VARIABLES'-collection(var-dvar),'S'-dvar]).

ctr_restrictions(
    increasing_sum,
    [required('VARIABLES',var),increasing('VARIABLES')]).

ctr_example(
    increasing_sum,
    increasing_sum([[var-3],[var-3],[var-6],[var-8]],20)).

ctr_typical(
    increasing_sum,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_typical_model(increasing_sum,[nval('VARIABLES'^var)>2]).

ctr_eval(increasing_sum,[reformulation(increasing_sum_r)]).

ctr_functional_dependency(increasing_sum,2,[1]).

ctr_cond_imply(
    increasing_sum,
    atmost_nvalue,
    [minval('VARIABLES'^var)>0],
    [],
    ['S','VARIABLES']).
```

```
ctr_cond_imply(
    increasing_sum,
    sum_of_increments,
    [minval('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(increasing_sum,2,0,2,6,[0-1,1-1,2-2,3-1,4-1]).

ctr_sol(
    increasing_sum,
    3,
    0,
    3,
    20,
    [0-1,1-1,2-2,3-3,4-3,5-3,6-3,7-2,8-1,9-1]).

ctr_sol(
    increasing_sum,
    4,
    0,
    4,
    70,
    [0-1,
     1-1,
     2-2,
     3-3,
     4-5,
     5-5,
     6-7,
     7-7,
     8-8,
     9-7,
     10-7,
     11-5,
     12-5,
     13-3,
     14-2,
     15-1,
     16-1]).

ctr_sol(
    increasing_sum,
    5,
    0,
    5,
```

```
    252,
    [0-1,
     1-1,
     2-2,
     3-3,
     4-5,
     5-7,
     6-9,
     7-11,
     8-14,
     9-16,
     10-18,
     11-19,
     12-20,
     13-20,
     14-19,
     15-18,
     16-16,
     17-14,
     18-11,
     19-9,
     20-7,
     21-5,
     22-3,
     23-2,
     24-1,
     25-1]).

ctr_sol(
    increasing_sum,
    6,
    0,
    6,
    924,
    [0-1,
     1-1,
     2-2,
     3-3,
     4-5,
     5-7,
     6-11,
     7-13,
     8-18,
     9-22,
     10-28,
     11-32,
```

```
     12-39,
     13-42,
     14-48,
     15-51,
     16-55,
     17-55,
     18-58,
     19-55,
     20-55,
     21-51,
     22-48,
     23-42,
     24-39,
     25-32,
     26-28,
     27-22,
     28-18,
     29-13,
     30-11,
     31-7,
     32-5,
     33-3,
     34-2,
     35-1,
     36-1]).

ctr_sol(
    increasing_sum,
    7,
    0,
    7,
    3432,
    [0-1,
     1-1,
     2-2,
     3-3,
     4-5,
     5-7,
     6-11,
     7-15,
     8-20,
     9-26,
     10-34,
     11-42,
     12-53,
     13-63,
```

```
      14-75,
      15-87,
      16-100,
      17-112,
      18-125,
      19-136,
      20-146,
      21-155,
      22-162,
      23-166,
      24-169,
      25-169,
      26-166,
      27-162,
      28-155,
      29-146,
      30-136,
      31-125,
      32-112,
      33-100,
      34-87,
      35-75,
      36-63,
      37-53,
      38-42,
      39-34,
      40-26,
      41-20,
      42-15,
      43-11,
      44-7,
      45-5,
      46-3,
      47-2,
      48-1,
      49-1]).

ctr_sol(
    increasing_sum,
    8,
    0,
    8,
    12870,
    [0-1,
     1-1,
     2-2,
```

3-3,
4-5,
5-7,
6-11,
7-15,
8-22,
9-28,
10-38,
11-48,
12-63,
13-77,
14-97,
15-116,
16-141,
17-164,
18-194,
19-221,
20-255,
21-284,
22-319,
23-348,
24-383,
25-409,
26-440,
27-461,
28-486,
29-499,
30-515,
31-519,
32-526,
33-519,
34-515,
35-499,
36-486,
37-461,
38-440,
39-409,
40-383,
41-348,
42-319,
43-284,
44-255,
45-221,
46-194,
47-164,
48-141,

```
    49-116,
    50-97,
    51-77,
    52-63,
    53-48,
    54-38,
    55-28,
    56-22,
    57-15,
    58-11,
    59-7,
    60-5,
    61-3,
    62-2,
    63-1,
    64-1]).

increasing_sum_r(VARIABLES,S) :-
        eval(increasing(VARIABLES)),
        eval(sum_ctr(VARIABLES,=,S)).
```

B.193 increasing_valley

◇ **META-DATA:**

```
ctr_date(increasing_valley,['20130210']).

ctr_origin(
    increasing_valley,
    Derived from %c and %c.,
    [valley,increasing]).

ctr_arguments(
    increasing_valley,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    increasing_valley,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    increasing_valley,
    increasing_valley(
        [[var-3],
         [var-5],
         [var-1],
         [var-4],
         [var-3],
         [var-5],
         [var-3],
         [var-3],
         [var-7],
         [var-2]])).

ctr_typical(
    increasing_valley,
    [size('VARIABLES')>=7,
     range('VARIABLES'^var)>1,
     valley('VARIABLES'^var)>=3]).

ctr_typical_model(increasing_valley,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    increasing_valley,
    [translate(['VARIABLES'^var])]).

ctr_eval(
```

```
    increasing_valley,
    [checker(increasing_valley_c),
     automaton(increasing_valley_a),
     automaton_with_signature(increasing_valley_a_s)]).

ctr_contractible(increasing_valley,[],'VARIABLES',prefix).

ctr_contractible(increasing_valley,[],'VARIABLES',suffix).

ctr_cond_imply(
    increasing_valley,
    not_all_equal,
    [valley('VARIABLES'^var)>0],
    [],
    id).

ctr_sol(increasing_valley,2,0,2,9,-).

ctr_sol(increasing_valley,3,0,3,64,-).

ctr_sol(increasing_valley,4,0,4,625,-).

ctr_sol(increasing_valley,5,0,5,7553,-).

ctr_sol(increasing_valley,6,0,6,105798,-).

ctr_sol(increasing_valley,7,0,7,1666878,-).

ctr_sol(increasing_valley,8,0,8,29090469,-).

increasing_valley_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        L>0,
        get_attr1(VARIABLES,VARS),
        increasing_valley_c(VARS,s,0).

increasing_valley_c([V1,V2|R],s,A) :-
        V1=<V2,
        !,
        increasing_valley_c([V2|R],s,A).

increasing_valley_c([_32528,V2|R],s,A) :-
        !,
        increasing_valley_c([V2|R],u,A).
```

```
increasing_valley_c([V1,V2|R],u,A) :-
        V1>=V2,
        !,
        increasing_valley_c([V2|R],u,A).

increasing_valley_c([V1,V2|R],u,_32527) :-
        !,
        increasing_valley_c([V2|R],v,V1).

increasing_valley_c([V1,V2|R],v,A) :-
        V1=<V2,
        !,
        increasing_valley_c([V2|R],v,A).

increasing_valley_c([_32528,V2|R],v,A) :-
        !,
        increasing_valley_c([V2|R],w,A).

increasing_valley_c([V1,V2|R],w,A) :-
        V1>=V2,
        !,
        increasing_valley_c([V2|R],w,A).

increasing_valley_c([V1,V2|R],w,A) :-
        !,
        A=<V1,
        increasing_valley_c([V2|R],v,V1).

increasing_valley_c([_32528],_32526,_32527) :-
        !.

increasing_valley_c([],_32523,_32524).

ctr_automaton_signature(
    increasing_valley,
    increasing_valley_a,
    pair_signature(1,signature)).

increasing_valley_a(FLAG,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        increasing_valley_a_s(FLAG,VARIABLES,SIGNATURE).

increasing_valley_a_s(FLAG,VARIABLES,SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        L>=0,
```

```
pair_first_signature(VARIABLES,VARS),
automaton(
    VARS,
    Vi,
    SIGNATURE,
    [source(s),sink(s),sink(u),sink(v),sink(w)],
    [arc(s,0,s),
     arc(s,1,s),
     arc(s,2,u),
     arc(u,0,v,[Vi,F]),
     arc(u,1,u),
     arc(u,2,u),
     arc(v,0,v),
     arc(v,1,v),
     arc(v,2,w),
     arc(w,0,v,(A#=<Vi->[Vi,F])),
     arc(w,0,v,(A#>Vi->[A,0])),
     arc(w,1,w),
     arc(w,2,w)],
    [A,F],
    [0,1],
    [_A,FLAG]).
```

B.194 indexed_sum

◇ **META-DATA:**

```
ctr_date(indexed_sum,['20040814','20060810','20090422']).

ctr_origin(indexed_sum,'N.~Beldiceanu',[]).

ctr_arguments(
    indexed_sum,
    ['ITEMS'-collection(index-dvar,weight-dvar),
     'TABLE'-collection(index-int,summation-dvar)]).

ctr_restrictions(
    indexed_sum,
    [size('ITEMS')>0,
     size('TABLE')>0,
     required('ITEMS',[index,weight]),
     'ITEMS'^index>=1,
     'ITEMS'^index=<size('TABLE'),
     required('TABLE',[index,summation]),
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     increasing_seq('TABLE',index)]).

ctr_example(
    indexed_sum,
    indexed_sum(
        [[index-3,weight- -4],
         [index-1,weight-6],
         [index-3,weight-1]],
        [[index-1,summation-6],
         [index-2,summation-0],
         [index-3,summation- -3]])).

ctr_typical(
    indexed_sum,
    [size('ITEMS')>1,
     range('ITEMS'^index)>1,
     size('TABLE')>1,
     range('TABLE'^summation)>1]).

ctr_exchangeable(
    indexed_sum,
    [items('ITEMS',all),items('TABLE',all)]).
```

```
ctr_graph(
    indexed_sum,
    ['ITEMS','TABLE'],
    2,
    foreach('TABLE',['PRODUCT'>>collection(items,table)]),
    [items^index=table^index],
    [],
    [],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'ITEMS'^weight)])]],
    [sum_ctr(variables,=,'TABLE'^summation)]).

ctr_eval(indexed_sum,[reformulation(indexed_sum_r)]).

indexed_sum_r(ITEMS,TABLE) :-
        length(ITEMS,I),
        length(TABLE,T),
        I>0,
        T>0,
        collection(ITEMS,[dvar(1,T),dvar]),
        collection(TABLE,[int(1,T),dvar]),
        collection_increasing_seq(TABLE,[1]),
        get_attr1(ITEMS,ITEMS_INDEXES),
        get_attr2(ITEMS,ITEMS_WEIGHTS),
        get_attr2(TABLE,TABLE_TSUMS),
        indexed_sum1(
            1,
            T,
            TABLE_TSUMS,
            ITEMS_INDEXES,
            ITEMS_WEIGHTS).

indexed_sum1(I,T,[],_43186,_43187) :-
        I>T,
        !.

indexed_sum1(I,T,[SUM|R],ITEMS_INDEXES,ITEMS_WEIGHTS) :-
        indexed_sum2(ITEMS_INDEXES,ITEMS_WEIGHTS,I,TERM),
        call(SUM#=TERM),
        I1 is I+1,
        indexed_sum1(I1,T,R,ITEMS_INDEXES,ITEMS_WEIGHTS).

indexed_sum2([],[],_43182,0).
```

```
indexed_sum2([J|R],[W|S],I,W*B+T) :-
        B#<=>J#=I,
        indexed_sum2(R,S,I,T).
```

B.195 inflexion

◊ **META-DATA:**

```
ctr_date(inflexion,['20000128','20030820','20040530']).

ctr_origin(inflexion,'N.~Beldiceanu',[]).

ctr_arguments(
    inflexion,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    inflexion,
    ['N'>=0,
     'N'=<max(0,size('VARIABLES')-2),
     required('VARIABLES',var)]).

ctr_example(
    inflexion,
    [inflexion(
         3,
         [[var-1],
          [var-1],
          [var-4],
          [var-8],
          [var-8],
          [var-2],
          [var-7],
          [var-1]]),
     inflexion(
         0,
         [[var-1],
          [var-1],
          [var-4],
          [var-4],
          [var-6],
          [var-6],
          [var-7],
          [var-9]]),
     inflexion(
         7,
         [[var-1],
          [var-0],
          [var-2],
          [var-0],
```

```
          [var-7],
          [var-2],
          [var-7],
          [var-1],
          [var-2]])]).

ctr_typical(
    inflexion,
    ['N'>0,size('VARIABLES')>2,range('VARIABLES'^var)>1]).

ctr_typical_model(inflexion,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    inflexion,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    inflexion,
    [checker(inflexion_c),
     automaton(inflexion_a),
     automaton_with_signature(inflexion_a_s)]).

ctr_pure_functional_dependency(inflexion,[]).

ctr_functional_dependency(inflexion,1,[2]).

ctr_cond_imply(
    inflexion,
    atleast_nvalue,
    ['N'>0],
    ['NVAL'=2],
    [none,'VARIABLES']).

ctr_cond_imply(
    inflexion,
    peak,
    [valley('VARIABLES'^var)=0],
    [],
    id).

ctr_cond_imply(
    inflexion,
    valley,
    [peak('VARIABLES'^var)=0],
    [],
    id).
```

```
ctr_sol(inflexion,2,0,2,9,[0-9]).

ctr_sol(inflexion,3,0,3,64,[0-36,1-28]).

ctr_sol(inflexion,4,0,4,625,[0-135,1-320,2-170]).

ctr_sol(inflexion,5,0,5,7776,[0-498,1-2588,2-3348,3-1342]).

ctr_sol(
    inflexion,
    6,
    0,
    6,
    117649,
    [0-1841,1-18494,2-44058,3-40446,4-12810]).

ctr_sol(
    inflexion,
    7,
    0,
    7,
    2097152,
    [0-6856,1-125284,2-492320,3-778936,4-549152,5-144604]).

ctr_sol(
    inflexion,
    8,
    0,
    8,
    43046721,
    [0-25731,
     1-828120,
     2-5069970,
     3-12341184,
     4-14547186,
     5-8354520,
     6-1880010]).

inflexion_c(N,VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        MAX is max(0,L-2),
        check_type(dvar(0,MAX),N),
        get_attr1(VARIABLES,VARS),
        inflexion_c(s,VARS,0,N).
```

```
inflexion_c(s,[V,V|R],C,N) :-
        !,
        inflexion_c(s,[V|R],C,N).

inflexion_c(s,[V1,V2|R],C,N) :-
        V1<V2,
        !,
        inflexion_c(i,[V2|R],C,N).

inflexion_c(s,[_51811,V2|R],C,N) :-
        !,
        inflexion_c(j,[V2|R],C,N).

inflexion_c(i,[V1,V2|R],C,N) :-
        V1=<V2,
        !,
        inflexion_c(i,[V2|R],C,N).

inflexion_c(i,[_51811,V2|R],C,N) :-
        !,
        C1 is C+1,
        inflexion_c(j,[V2|R],C1,N).

inflexion_c(j,[V1,V2|R],C,N) :-
        V1>=V2,
        !,
        inflexion_c(j,[V2|R],C,N).

inflexion_c(j,[_51811,V2|R],C,N) :-
        !,
        C1 is C+1,
        inflexion_c(i,[V2|R],C1,N).

inflexion_c(_51807,[_51811],N,N) :-
        !.

inflexion_c(_51804,[],N,N).

inflexion_counters_check(s,[V,V|R],C,[C|S]) :-
        !,
        inflexion_counters_check(s,[V|R],C,S).

inflexion_counters_check(s,[V1,V2|R],C,[C|S]) :-
        V1<V2,
        !,
```

```
        inflexion_counters_check(i,[V2|R],C,S).

inflexion_counters_check(s,[_51811,V2|R],C,[C|S]) :-
        !,
        inflexion_counters_check(j,[V2|R],C,S).

inflexion_counters_check(i,[V1,V2|R],C,[C|S]) :-
        V1=<V2,
        !,
        inflexion_counters_check(i,[V2|R],C,S).

inflexion_counters_check(i,[_51811,V2|R],C,[C1|S]) :-
        !,
        C1 is C+1,
        inflexion_counters_check(j,[V2|R],C1,S).

inflexion_counters_check(j,[V1,V2|R],C,[C|S]) :-
        V1>=V2,
        !,
        inflexion_counters_check(j,[V2|R],C,S).

inflexion_counters_check(j,[_51811,V2|R],C,[C1|S]) :-
        !,
        C1 is C+1,
        inflexion_counters_check(i,[V2|R],C1,S).

inflexion_counters_check(init,[V|R],C,[0|S]) :-
        !,
        inflexion_counters_check(s,[V|R],C,S).

inflexion_counters_check(_51804,[_51808],_51806,[]).

ctr_automaton_signature(
    inflexion,
    inflexion_a,
    pair_signature(2,signature)).

inflexion_a(FLAG,N,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        inflexion_a_s(FLAG,N,VARIABLES,SIGNATURE).

inflexion_a_s(FLAG,N,VARIABLES,SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        MAX is max(0,L-2),
        check_type(dvar(0,MAX),N),
```

```
automaton(
    SIGNATURE,
    _53330,
    SIGNATURE,
    [source(s),sink(i),sink(j),sink(s)],
    [arc(s,1,s),
     arc(s,0,i),
     arc(s,2,j),
     arc(i,1,i),
     arc(i,0,i),
     arc(i,2,j,[C+1]),
     arc(j,1,j),
     arc(j,2,j),
     arc(j,0,i,[C+1])],
    [C],
    [0],
    [COUNT]),
COUNT#=N#<=>FLAG.
```

B.196 inside_sboxes

◇ META-DATA:

```
ctr_date(inside_sboxes,['20070622','20090725']).

ctr_origin(
    inside_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(inside_sboxes,[inside]).

ctr_types(
    inside_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    inside_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    inside_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    inside_sboxes,
    inside_sboxes(
        2,
        {0,1},
        [[oid-1,sid-1,x-[[v-3],[v-3]]],
         [oid-2,sid-2,x-[[v-2],[v-2]]],
         [oid-3,sid-3,x-[[v-1],[v-1]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-1],[v-1]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-3]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-5],[v-5]]]])).

ctr_typical(inside_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    inside_sboxes,
    [items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all)]).

ctr_eval(inside_sboxes,[logic(inside_sboxes_g)]).

ctr_logic(
    inside_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (inside_sboxes(Dims,O1,S1,O2,S2)--->
      forall(
          D,
          Dims,
          origin(O2,S2,D)#<origin(O1,S1,D)#/\
          end(O1,S1,D)#<end(O2,S2,D))),
     (inside_objects(Dims,O1,O2)--->
      forall(
          S1,
          sboxes([O1^sid]),
          exists(
              S2,
              sboxes([O2^sid]),
```

```
                inside_sboxes(Dims,O1,S1,O2,S2)))),
      (all_inside(Dims,OIDS)--->
       forall(
           O1,
           objects(OIDS),
           forall(
               O2,
               objects(OIDS),
               O1^oid#<O2^oid#=>inside_objects(Dims,O1,O2)))),
     all_inside(DIMENSIONS,OIDS)]).

ctr_contractible(inside_sboxes,[],'OBJECTS',suffix).

ctr_application(inside_sboxes,[3]).

inside_sboxes_g(K,_38639,[],_38641) :-
        !,
        check_type(int_gteq(1),K).

inside_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        collection_increasing_seq(OBJECTS,[1]),
        geost1(OIDS,SIDS,XS,Objects),
        geost2(SIDES,TS,TL,Sboxes),
        geost_dims(1,K,DIMENSIONS),
        ctr_logic(inside_sboxes,[DIMENSIONS,OIDS],Rules),
        geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.197 int_value_precede

◇ **META-DATA:**

```
ctr_date(int_value_precede,['20041003']).

ctr_origin(int_value_precede,'\\cite{YatChiuLawJimmyLee04}',[]).

ctr_synonyms(
    int_value_precede,
    [precede,precedence,value_precede]).

ctr_arguments(
    int_value_precede,
    ['S'-int,'T'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    int_value_precede,
    ['S'=\='T',required('VARIABLES',var)]).

ctr_example(
    int_value_precede,
    int_value_precede(
        0,
        1,
        [[var-4],[var-0],[var-6],[var-1],[var-0]])).

ctr_typical(
    int_value_precede,
    ['S'<'T',
     size('VARIABLES')>1,
     atleast(1,'VARIABLES','S'),
     atleast(1,'VARIABLES','T')]).

ctr_exchangeable(
    int_value_precede,
    [vals(
         ['VARIABLES'^var],
         int(notin(['S','T'])),
         =\=,
         dontcare,
         dontcare),
     vals(
         ['S','T','VARIABLES'^var],
         int(['S','T']),
         =\=,
```

```
          all,
          in)]).

ctr_eval(int_value_precede,[automaton(int_value_precede_a)]).

ctr_contractible(int_value_precede,[],'VARIABLES',suffix).

ctr_aggregate(int_value_precede,[],[id,id,union]).

int_value_precede_a(1,S,T,[]) :-
        !,
        check_type(int,S),
        check_type(int,T),
        S=\=T.

int_value_precede_a(0,_S,_T,[]) :-
        !,
        fail.

int_value_precede_a(FLAG,S,T,VARIABLES) :-
        check_type(int,S),
        check_type(int,T),
        S=\=T,
        collection(VARIABLES,[dvar]),
        int_value_precede_signature(VARIABLES,SIGNATURE,S,T),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _34091,
            SIGNATURE,
            [source(s),sink(s),sink(t)],
            [arc(s,3,s),
             arc(s,1,t),
             arc(t,1,t),
             arc(t,2,t),
             arc(t,3,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[1,2,3],AUTOMATON).

int_value_precede_signature([],[],_32291,_32292).

int_value_precede_signature([[var-VAR]|VARs],[SI|SIs],S,T) :-
        SI in 1..3,
        VAR#=S#<=>SI#=1,
```

```
VAR#=T#<=>SI#=2,
VAR#\=S#/\VAR#\=T#<=>SI#=3,
int_value_precede_signature(VARs,SIs,S,T).
```

B.198 int_value_precede_chain

◇ META-DATA:

```
ctr_date(
    int_value_precede_chain,
    ['20041003','20090728','20090822']).

ctr_origin(
    int_value_precede_chain,
    \cite{YatChiuLawJimmyLee04},
    []).

ctr_synonyms(
    int_value_precede_chain,
    [precede,precedence,value_precede_chain]).

ctr_arguments(
    int_value_precede_chain,
    ['VALUES'-collection(var-int),
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    int_value_precede_chain,
    [required('VALUES',var),
     distinct('VALUES',var),
     required('VARIABLES',var)]).

ctr_example(
    int_value_precede_chain,
    int_value_precede_chain(
        [[var-4],[var-0],[var-1]],
        [[var-4],[var-0],[var-6],[var-1],[var-0]])).

ctr_typical(
    int_value_precede_chain,
    [size('VALUES')>1,
     strictly_increasing('VALUES'),
     size('VARIABLES')>size('VALUES'),
     range('VARIABLES'^var)>1,
     used_by('VARIABLES','VALUES')]).

ctr_exchangeable(
    int_value_precede_chain,
    [vals(
         ['VARIABLES'^var],
```

```
         int(notin('VALUES'^var)),
         =\=,
         dontcare,
         dontcare)]).

ctr_eval(
    int_value_precede_chain,
    [automaton(int_value_precede_chain_a)]).

ctr_contractible(int_value_precede_chain,[],'VALUES',any).

ctr_contractible(int_value_precede_chain,[],'VARIABLES',suffix).

ctr_aggregate(int_value_precede_chain,[],[id,union]).

int_value_precede_chain_a(FLAG,[],VARIABLES) :-
        !,
        collection(VARIABLES,[dvar]),
        (   FLAG=1 ->
            true
        ;   fail
        ).

int_value_precede_chain_a(FLAG,VALUES,[]) :-
        !,
        collection(VALUES,[int]),
        get_attr1(VALUES,VALS),
        all_different(VALS),
        (   FLAG=1 ->
            true
        ;   fail
        ).

int_value_precede_chain_a(FLAG,VALUES,VARIABLES) :-
        collection(VALUES,[int]),
        collection(VARIABLES,[dvar]),
        length(VALUES,1),
        !,
        (   FLAG=1 ->
            true
        ;   fail
        ).

int_value_precede_chain_a(FLAG,VALUES,VARIABLES) :-
        collection(VALUES,[int]),
        collection(VARIABLES,[dvar]),
```

```
        get_attr1(VALUES,VALS),
        all_different(VALS),
        length(VALS,N),
        get_attr1(VARIABLES,VARS),
        int_value_precede_chain_gen_complement(
            VARS,
            VALS,
            COMPLEMENT),
        int_value_precede_chain_gen_states1(0,N,STATES),
        int_value_precede_chain_gen_transitions(
            N,
            VALS,
            COMPLEMENT,
            STATES,
            TRANSITIONS),
        nth0(0,STATES,S0),
        int_value_precede_chain_gen_states2(
            STATES,
            S0,
            AUTOMATON_STATES),
        AUTOMATON=
        automaton(
            VARS,
            _37646,
            VARS,
            AUTOMATON_STATES,
            TRANSITIONS,
            [],
            [],
            []),
        append(VALS,COMPLEMENT,ALPHABET),
        automaton_bool(FLAG,ALPHABET,AUTOMATON).

int_value_precede_chain_gen_states2([],S0,[source(S0)]) :-
        !.

int_value_precede_chain_gen_states2([S|R],S0,[sink(S)|T]) :-
        int_value_precede_chain_gen_states2(R,S0,T).

int_value_precede_chain_gen_complement(VARS,VALS,COMPLEMENT) :-
        union_dom_set(VARS,UNION),
        list_to_fdset(VALS,VALUES),
        fdset_subtract(UNION,VALUES,DIFFERENCE),
        fdset_to_list(DIFFERENCE,COMPLEMENT).

int_value_precede_chain_gen_states1(I,N,[]) :-
```

```
        I>N,
        !.

int_value_precede_chain_gen_states1(I,N,[INAME|R]) :-
        I=<N,
        number_codes(I,ICODE),
        atom_codes(IATOM,ICODE),
        atom_concat(s,IATOM,INAME),
        I1 is I+1,
        int_value_precede_chain_gen_states1(I1,N,R).

int_value_precede_chain_gen_transitions(
    N,
    VALS,
    COMPLEMENT,
    STATES,
    TRANSITIONS) :-
        N1 is N-1,
        int_value_precede_chain_gen_transitions1(
            0,
            N1,
            VALS,
            STATES,
            TR1),
        int_value_precede_chain_gen_transitions2(
            1,
            N,
            VALS,
            STATES,
            TR2),
        int_value_precede_chain_gen_transitions3(
            0,
            N,
            VALS,
            STATES,
            COMPLEMENT,
            TR3),
        append(TR1,TR2,TR12),
        append(TR12,TR3,TRANSITIONS).

int_value_precede_chain_gen_transitions1(I,N1,_34098,_34099,[]) :-
        I>N1,
        !.

int_value_precede_chain_gen_transitions1(
    I,
```

```
    N1,
    VALS,
    STATES,
    [arc(Si,Vii,Sii)|R]) :-
        I=<N1,
        I1 is I+1,
        nth0(I,STATES,Si),
        nth1(I1,VALS,Vii),
        nth0(I1,STATES,Sii),
        int_value_precede_chain_gen_transitions1(
            I1,
            N1,
            VALS,
            STATES,
            R).

int_value_precede_chain_gen_transitions2(I,N1,_34098,_34099,[]) :-
        I>N1,
        !.

int_value_precede_chain_gen_transitions2(
    I,
    N1,
    VALS,
    STATES,
    TRANSITIONS) :-
        I=<N1,
        int_value_precede_chain_gen_transitions21(
            1,
            I,
            VALS,
            STATES,
            TR),
        I1 is I+1,
        int_value_precede_chain_gen_transitions2(
            I1,
            N1,
            VALS,
            STATES,
            R),
        append(TR,R,TRANSITIONS).

int_value_precede_chain_gen_transitions21(J,I,_34098,_34099,[]) :-
        J>I,
        !.
```

```
int_value_precede_chain_gen_transitions21(
    J,
    I,
    VALS,
    STATES,
    [arc(Si,Vj,Si)|R]) :-
        J=<I,
        nth0(I,STATES,Si),
        nth1(J,VALS,Vj),
        J1 is J+1,
        int_value_precede_chain_gen_transitions21(
            J1,
            I,
            VALS,
            STATES,
            R).

int_value_precede_chain_gen_transitions3(
    _34334,
    _34381,
    _34428,
    _34475,
    [],
    []) :-
        !.

int_value_precede_chain_gen_transitions3(
    I,
    N1,
    _34442,
    _34489,
    _34536,
    []) :-
        I>N1,
        !.

int_value_precede_chain_gen_transitions3(
    I,
    N1,
    VALS,
    STATES,
    [C|CC],
    TRANSITIONS) :-
        I=<N1,
        length([C|CC],LC),
        int_value_precede_chain_gen_transitions31(
```

```
            1,
            LC,
            I,
            [C|CC],
            STATES,
            TR),
        I1 is I+1,
        int_value_precede_chain_gen_transitions3(
            I1,
            N1,
            VALS,
            STATES,
            [C|CC],
            R),
        append(TR,R,TRANSITIONS).

int_value_precede_chain_gen_transitions31(
    J,
    LC,
    _34442,
    _34489,
    _34536,
    []) :-
        J>LC,
        !.

int_value_precede_chain_gen_transitions31(
    J,
    LC,
    I,
    C,
    STATES,
    [arc(Si,Cj,Si)|R]) :-
        J=<LC,
        nth0(I,STATES,Si),
        nth1(J,C,Cj),
        J1 is J+1,
        int_value_precede_chain_gen_transitions31(
            J1,
            LC,
            I,
            C,
            STATES,
            R).
```

B.199 interval_and_count

◇ **META-DATA:**

```
ctr_date(
    interval_and_count,
    ['20000128','20030820','20040530','20060810']).

ctr_origin(interval_and_count,'\\cite{Cousin93}',[]).

ctr_arguments(
    interval_and_count,
    ['ATMOST'-int,
     'COLOURS'-collection(val-int),
     'TASKS'-collection(origin-dvar,colour-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    interval_and_count,
    ['ATMOST'>=0,
     required('COLOURS',val),
     distinct('COLOURS',val),
     required('TASKS',[origin,colour]),
     'TASKS'^origin>=0,
     'SIZE_INTERVAL'>0]).

ctr_example(
    interval_and_count,
    interval_and_count(
        2,
        [[val-4]],
        [[origin-1,colour-4],
         [origin-0,colour-9],
         [origin-10,colour-4],
         [origin-4,colour-4]],
        5)).

ctr_typical(
    interval_and_count,
    ['ATMOST'>0,
     'ATMOST'<size('TASKS'),
     size('COLOURS')>0,
     size('TASKS')>1,
     range('TASKS'^origin)>1,
     range('TASKS'^colour)>1,
     'SIZE_INTERVAL'>1]).
```

```
ctr_exchangeable(
    interval_and_count,
    [vals(['ATMOST'],int,<,dontcare,dontcare),
     items('COLOURS',all),
     items('TASKS',all),
     translate(['TASKS'^origin]),
     vals(
         ['TASKS'^origin],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare),
     vals(
         ['TASKS'^colour],
         comp('COLOURS'^val),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    interval_and_count,
    ['TASKS','TASKS'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1^origin/'SIZE_INTERVAL'=
     tasks2^origin/'SIZE_INTERVAL'],
    [],
    [],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^colour)])]],
    [among_low_up(0,'ATMOST',variables,'COLOURS')]).

ctr_eval(
    interval_and_count,
    [reformulation(interval_and_count_r)]).

ctr_contractible(interval_and_count,[],'COLOURS',any).

ctr_contractible(interval_and_count,[],'TASKS',any).

ctr_application(interval_and_count,[3]).
```

```
interval_and_count_r(ATMOST,COLOURS,TASKS,SIZE_INTERVAL) :-
        check_type(int_gteq(0),ATMOST),
        collection(COLOURS,[int]),
        get_attr1(COLOURS,COLS),
        all_different(COLS),
        collection(TASKS,[dvar_gteq(0),dvar]),
        check_type(int_gteq(1),SIZE_INTERVAL),
        (   COLOURS=[] ->
            true
        ;   TASKS=[] ->
            true
        ;   get_attr1(TASKS,TORIS),
            get_attr2(TASKS,TCOLS),
            interval_and_count1(TCOLS,COLS,LB),
            get_maximum(TORIS,MAX),
            MAXK is(MAX+SIZE_INTERVAL-1)//SIZE_INTERVAL,
            interval_and_count2(
                0,
                MAXK,
                SIZE_INTERVAL,
                ATMOST,
                LB,
                TORIS)
        ).

interval_and_count1([],_51307,[]).

interval_and_count1([TC|R],COLS,[B|S]) :-
        build_or_var_in_values(COLS,TC,TERM),
        call(B#<=>TERM),
        interval_and_count1(R,COLS,S).

interval_and_count2(K,MAXK,_51311,_51312,_51313,_51314) :-
        K>MAXK,
        !.

interval_and_count2(K,MAXK,SIZE_INTERVAL,ATMOST,LB,TORIS) :-
        K=<MAXK,
        interval_and_count3(LB,TORIS,K,SIZE_INTERVAL,SUMB),
        call(SUMB#=<ATMOST),
        K1 is K+1,
        interval_and_count2(
            K1,
            MAXK,
            SIZE_INTERVAL,
            ATMOST,
```

```
            LB,
            TORIS).

interval_and_count3([],[],_51308,_51309,0).

interval_and_count3([B|R],[O|S],K,SIZE_INTERVAL,BK+T) :-
        SK is K*SIZE_INTERVAL,
        TK is SK+SIZE_INTERVAL-1,
        BK#<=>B#/\O#>=SK#/\O#=<TK,
        interval_and_count3(R,S,K,SIZE_INTERVAL,T).
```

B.200 interval_and_sum

◇ **META-DATA:**

```
ctr_date(interval_and_sum,['20000128','20030820','20060810']).

ctr_origin(interval_and_sum,'Derived from %c.',[cumulative]).

ctr_arguments(
    interval_and_sum,
    ['SIZE_INTERVAL'-int,
     'TASKS'-collection(origin-dvar,height-dvar),
     'LIMIT'-int]).

ctr_restrictions(
    interval_and_sum,
    ['SIZE_INTERVAL'>0,
     required('TASKS',[origin,height]),
     'TASKS'^origin>=0,
     'TASKS'^height>=0,
     'LIMIT'>=0]).

ctr_example(
    interval_and_sum,
    interval_and_sum(
        5,
        [[origin-1,height-2],
         [origin-10,height-2],
         [origin-10,height-3],
         [origin-4,height-1]],
        5)).

ctr_typical(
    interval_and_sum,
    ['SIZE_INTERVAL'>1,
     size('TASKS')>1,
     range('TASKS'^origin)>1,
     range('TASKS'^height)>1,
     'LIMIT'<sum('TASKS'^height)]).

ctr_exchangeable(
    interval_and_sum,
    [items('TASKS',all),
     translate(['TASKS'^origin]),
     vals(
         ['TASKS'^origin],
```

```
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare),
     vals(['TASKS'^height],int(>=(0)),>,dontcare,dontcare),
     vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_graph(
    interval_and_sum,
    ['TASKS','TASKS'],
    2,
    ['PRODUCT'>>collection(tasks1,tasks2)],
    [tasks1^origin/'SIZE_INTERVAL'=
     tasks2^origin/'SIZE_INTERVAL'],
    [],
    [],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^height)])]],
    [sum_ctr(variables,=<,'LIMIT')]).

ctr_eval(interval_and_sum,[reformulation(interval_and_sum_r)]).

ctr_contractible(interval_and_sum,[],'TASKS',any).

ctr_application(interval_and_sum,[2]).

interval_and_sum_r(SIZE_INTERVAL,TASKS,LIMIT) :-
        check_type(int_gteq(1),SIZE_INTERVAL),
        collection(TASKS,[dvar_gteq(0),dvar_gteq(0)]),
        check_type(int_gteq(0),LIMIT),
        (   TASKS=[] ->
            true
        ;   get_attr1(TASKS,ORIS),
            get_attr2(TASKS,HEIGHTS),
            get_maximum(ORIS,MAX),
            MAXK is(MAX+SIZE_INTERVAL-1)//SIZE_INTERVAL,
            interval_and_sum1(
                0,
                MAXK,
                SIZE_INTERVAL,
                LIMIT,
                ORIS,
                HEIGHTS)
```

```
        ).

interval_and_sum1(K,MAXK,_48750,_48751,_48752,_48753) :-
        K>MAXK,
        !.

interval_and_sum1(K,MAXK,SIZE_INTERVAL,LIMIT,ORIS,HEIGHTS) :-
        K=<MAXK,
        interval_and_sum2(ORIS,HEIGHTS,K,SIZE_INTERVAL,SUM),
        call(SUM#=<LIMIT),
        K1 is K+1,
        interval_and_sum1(
            K1,
            MAXK,
            SIZE_INTERVAL,
            LIMIT,
            ORIS,
            HEIGHTS).

interval_and_sum2([],[],_48747,_48748,0).

interval_and_sum2([O|R],[H|S],K,SIZE_INTERVAL,H*B+T) :-
        SK is K*SIZE_INTERVAL,
        TK is SK+SIZE_INTERVAL-1,
        B#<=>O#>=SK#/\O#=<TK,
        interval_and_sum2(R,S,K,SIZE_INTERVAL,T).
```

B.201 inverse

◇ **META-DATA:**

```
ctr_date(inverse,['20000128','20030820','20040530','20060810']).

ctr_origin(inverse,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(inverse,[assignment,channel,inverse_channeling]).

ctr_arguments(
    inverse,
    ['NODES'-collection(index-int,succ-dvar,pred-dvar)]).

ctr_restrictions(
    inverse,
    [required('NODES',[index,succ,pred]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES'),
     'NODES'^pred>=1,
     'NODES'^pred=<size('NODES')]).

ctr_example(
    inverse,
    inverse(
        [[index-1,succ-2,pred-2],
         [index-2,succ-1,pred-1],
         [index-3,succ-5,pred-4],
         [index-4,succ-3,pred-5],
         [index-5,succ-4,pred-3]])).

ctr_typical(inverse,[size('NODES')>1]).

ctr_exchangeable(
    inverse,
    [items('NODES',all),
     attrs_sync('NODES',[[index],[succ,pred]])]).

ctr_graph(
    inverse,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
```

```
    [nodes1^succ=nodes2^index,nodes2^pred=nodes1^index],
    ['NARC'=size('NODES')],
    []).

ctr_eval(inverse,[reformulation(inverse_r)]).

ctr_pure_functional_dependency(inverse,[]).

ctr_functional_dependency(inverse,1-2,[1-1,1-3]).

ctr_functional_dependency(inverse,1-3,[1-1,1-2]).

inverse_r([]) :-
        !.

inverse_r(NODES) :-
        length(NODES,N),
        collection(NODES,[int(1,N),dvar(1,N),dvar(1,N)]),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        get_attr3(NODES,PREDS),
        all_different(INDEXES),
        all_different(SUCCS),
        all_different(PREDS),
        inverse1(SUCCS,INDEXES,PREDS,INDEXES).

inverse1([],[],_56704,_56705).

inverse1([S_I|R],[I|S],PREDS,INDEXES) :-
        inverse2(PREDS,INDEXES,S_I,I),
        inverse1(R,S,PREDS,INDEXES).

inverse2([],[],_56704,_56705).

inverse2([P_J|R],[J|S],S_I,I) :-
        S_I#=J#<=>P_J#=I,
        inverse2(R,S,S_I,I).
```

B.202 inverse_except_loop

◇ META-DATA:

```
ctr_predefined(inverse_except_loop).

ctr_date(inverse_except_loop,['20141028']).

ctr_origin(inverse_except_loop,'Derived from %c',[inverse]).

ctr_arguments(
    inverse_except_loop,
    ['NODES'-collection(index-int,succ-dvar,pred-dvar)]).

ctr_restrictions(
    inverse_except_loop,
    [required('NODES',[index,succ,pred]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES'),
     'NODES'^pred>=1,
     'NODES'^pred=<size('NODES')]).

ctr_example(
    inverse_except_loop,
    inverse_except_loop(
        [[index-1,succ-3,pred-1],
         [index-2,succ-4,pred-2],
         [index-3,succ-3,pred-1],
         [index-4,succ-5,pred-2],
         [index-5,succ-5,pred-4]])).

ctr_typical(inverse_except_loop,[size('NODES')>1]).

ctr_eval(
    inverse_except_loop,
    [reformulation(inverse_except_loop_r)]).

ctr_pure_functional_dependency(inverse_except_loop,[]).

ctr_functional_dependency(inverse_except_loop,1-2,[1-1,1-3]).

ctr_functional_dependency(inverse_except_loop,1-3,[1-1,1-2]).

inverse_except_loop_r([]) :-
```

```
        !.

inverse_except_loop_r(NODES) :-
        length(NODES,N),
        collection(NODES,[int(1,N),dvar(1,N),dvar(1,N)]),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        get_attr3(NODES,PREDS),
        all_different(INDEXES),
        create_collection(SUCCS,var,VARS),
        create_collection(PREDS,var,VARP),
        inverse_except_loop0(1,N,OCCS),
        inverse_except_loop0(1,N,OCCP),
        eval(global_cardinality(VARS,OCCS)),
        eval(global_cardinality(VARP,OCCP)),
        inverse_except_loop1(SUCCS,INDEXES,PREDS,INDEXES),
        inverse_except_loop3(SUCCS,OCCP),
        inverse_except_loop3(PREDS,OCCS).

inverse_except_loop0(I,N,[]) :-
        I>N,
        !.

inverse_except_loop0(I,N,[[val-I,occ-O]|R]) :-
        I=<N,
        O in 0..N,
        I1 is I+1,
        inverse_except_loop0(I1,N,R).

inverse_except_loop1([],[],_25694,_25695) :-
        !.

inverse_except_loop1([S_I|R],[I|S],PREDS,INDEXES) :-
        inverse_except_loop2(PREDS,INDEXES,S_I,I),
        inverse_except_loop1(R,S,PREDS,INDEXES).

inverse_except_loop2([],[],_25694,_25695) :-
        !.

inverse_except_loop2([P_J|R],[J|S],S_I,I) :-
        S_I#=J#/\I#\=J#<=>P_J#=I#/\J#\=I,
        inverse_except_loop2(R,S,S_I,I).

inverse_except_loop3([],[]) :-
        !.
```

```
inverse_except_loop3([SP|R],[[val-I,occ-O]|T]) :-
        SP#=I#<=>O#=0,
        inverse_except_loop3(R,T).
```

B.203 inverse_offset

◇ **META-DATA:**

```
ctr_date(inverse_offset,['20091404']).

ctr_origin(inverse_offset,'\\index{Gecode|indexuse}Gecode',[]).

ctr_synonyms(inverse_offset,[channel]).

ctr_arguments(
    inverse_offset,
    ['SOFFSET'-int,
     'POFFSET'-int,
     'NODES'-collection(index-int,succ-dvar,pred-dvar)]).

ctr_restrictions(
    inverse_offset,
    [required('NODES',[index,succ,pred]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1+'SOFFSET',
     'NODES'^succ=<size('NODES')+'SOFFSET',
     'NODES'^pred>=1+'POFFSET',
     'NODES'^pred=<size('NODES')+'POFFSET']).

ctr_example(
    inverse_offset,
    inverse_offset(
        -1,
        0,
        [[index-1,succ-4,pred-3],
         [index-2,succ-2,pred-5],
         [index-3,succ-0,pred-2],
         [index-4,succ-6,pred-8],
         [index-5,succ-1,pred-1],
         [index-6,succ-7,pred-7],
         [index-7,succ-5,pred-4],
         [index-8,succ-3,pred-6]])).

ctr_typical(
    inverse_offset,
    ['SOFFSET'>= -1,
     'SOFFSET'=<1,
     'POFFSET'>= -1,
```

```
     'POFFSET'=<1,
     size('NODES')>1]).

ctr_exchangeable(inverse_offset,[items('NODES',all)]).

ctr_graph(
    inverse_offset,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ-'SOFFSET'=nodes2^index,
     nodes2^pred-'POFFSET'=nodes1^index],
    ['NARC'=size('NODES')],
    []).

ctr_pure_functional_dependency(inverse_offset,[]).

ctr_functional_dependency(inverse_offset,3-2,[1,2,3-1,3-3]).

ctr_functional_dependency(inverse_offset,3-3,[1,2,3-1,3-2]).
```

B.204 inverse_set

◇ **META-DATA:**

```
ctr_date(inverse_set,['20041211','20060810']).

ctr_origin(inverse_set,'Derived from %c.',[inverse]).

ctr_arguments(
    inverse_set,
    ['X'-collection(index-int,set-svar),
     'Y'-collection(index-int,set-svar)]).

ctr_restrictions(
    inverse_set,
    [required('X',[index,set]),
     required('Y',[index,set]),
     increasing_seq('X',index),
     increasing_seq('Y',index),
     'X'^index>=1,
     'X'^index=<size('X'),
     'Y'^index>=1,
     'Y'^index=<size('Y'),
     'X'^set>=1,
     'X'^set=<size('Y'),
     'Y'^set>=1,
     'Y'^set=<size('X')]).

ctr_example(
    inverse_set,
    inverse_set(
        [[index-1,set-{2,4}],
         [index-2,set-{4}],
         [index-3,set-{1}],
         [index-4,set-{4}]],
        [[index-1,set-{3}],
         [index-2,set-{1}],
         [index-3,set-{}],
         [index-4,set-{1,2,4}],
         [index-5,set-{}]])).

ctr_typical(inverse_set,[size('X')>1,size('Y')>1]).

ctr_exchangeable(
    inverse_set,
    [args([['X','Y']]),items('X',all),items('Y',all)]).
```

```
ctr_graph(
    inverse_set,
    ['X','Y'],
    2,
    ['PRODUCT'>>collection(x,y)],
    [y^index in_set x^set#<=>x^index in_set y^set],
    ['NARC'=size('X')*size('Y')],
    []).
```

B.205 inverse_within_range

◊ **META-DATA:**

```
ctr_date(inverse_within_range,['20060517','20060810']).

ctr_origin(inverse_within_range,'Derived from %c.',[inverse]).

ctr_synonyms(
    inverse_within_range,
    [inverse_in_range,inverse_range]).

ctr_arguments(
    inverse_within_range,
    ['X'-collection(var-dvar),'Y'-collection(var-dvar)]).

ctr_restrictions(
    inverse_within_range,
    [required('X',var),required('Y',var)]).

ctr_example(
    inverse_within_range,
    inverse_within_range(
        [[var-9],[var-4],[var-2]],
        [[var-9],[var-3],[var-9],[var-2]])).

ctr_typical(
    inverse_within_range,
    [size('X')>1,
     range('X'^var)>1,
     size('Y')>1,
     range('Y'^var)>1]).

ctr_exchangeable(inverse_within_range,[args([['X','Y']])]).

ctr_graph(
    inverse_within_range,
    ['X','Y'],
    2,
    ['SYMMETRIC_PRODUCT'>>collection(s1,s2)],
    [s1^var=s2^key],
    [],
    ['BIPARTITE','NO_LOOP','SYMMETRIC']).
```

B.206 ith_pos_different_from_0

◇ META-DATA:

```
ctr_date(ith_pos_different_from_0,['20040811']).

ctr_origin(ith_pos_different_from_0,'N.~Beldiceanu',[]).

ctr_arguments(
    ith_pos_different_from_0,
    ['ITH'-int,'POS'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    ith_pos_different_from_0,
    ['ITH'>=1,
     'ITH'=<size('VARIABLES'),
     'POS'>='ITH',
     'POS'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    ith_pos_different_from_0,
    ith_pos_different_from_0(
        2,
        4,
        [[var-3],[var-0],[var-0],[var-8],[var-6]])).

ctr_typical(
    ith_pos_different_from_0,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     atleast(1,'VARIABLES',0)]).

ctr_typical_model(
    ith_pos_different_from_0,
    [atleast(2,'VARIABLES',0)]).

ctr_exchangeable(
    ith_pos_different_from_0,
    [vals(
         ['VARIABLES'^var],
         int(=\=(0)),
         =\=,
         dontcare,
         dontcare)]).
```

```
ctr_eval(
    ith_pos_different_from_0,
    [automaton(ith_pos_different_from_0_a)]).

ctr_extensible(ith_pos_different_from_0,[],'VARIABLES',suffix).

ith_pos_different_from_0_a(FLAG,ITH,POS,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        integer(ITH),
        ITH>=1,
        ITH=<N,
        check_type(dvar(ITH,N),POS),
        ith_pos_different_from_0_signature(VARIABLES,SIGNATURE),
        automaton(
            SIGNATURE,
            _28275,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s,(C#<ITH->[C+1,D+1];C#>=ITH->[C,D])),
             arc(s,1,s,(C#<ITH->[C,D+1];C#>=ITH->[C,D]))],
            [C,D],
            [0,0],
            [C1,D1]),
        C1#=ITH#/\D1#=POS#<=>FLAG.

ith_pos_different_from_0_signature([],[]).

ith_pos_different_from_0_signature([[var-V]|VARs],[S|Ss]) :-
        V#=0#<=>S,
        ith_pos_different_from_0_signature(VARs,Ss).
```

B.207 k_alldifferent

◇ **META-DATA:**

```
ctr_date(k_alldifferent,['20050618','20060811']).

ctr_origin(
    k_alldifferent,
    \cite{ElbassioniKatrielKutzMahajan05},
    []).

ctr_synonyms(
    k_alldifferent,
    [k_alldiff,k_alldistinct,some_different]).

ctr_types(k_alldifferent,['X'-collection(x-dvar)]).

ctr_arguments(k_alldifferent,['VARS'-collection(vars-'X')]).

ctr_restrictions(
    k_alldifferent,
    [size('X')>=1,
     required('X',x),
     required('VARS',vars),
     size('VARS')>=1]).

ctr_example(
    k_alldifferent,
    k_alldifferent(
        [[vars-[[x-5],[x-6],[x-0],[x-9],[x-3]]],
         [vars-[[x-5],[x-6],[x-1],[x-2]]]]))).

ctr_typical(k_alldifferent,[size('X')>1,size('VARS')>1]).

ctr_exchangeable(
    k_alldifferent,
    [items('VARS',all),
     items('VARS'^vars,all),
     vals(['VARS'^vars^x],int,=\=,all,dontcare)]).

ctr_graph(
    k_alldifferent,
    ['VARS'^vars],
    2,
    foreach('VARS',['CLIQUE'>>collection(x1,x2)]),
    [x1^x=x2^x],
```

```
    ['MAX_NSCC'=<1],
    []).

ctr_eval(
    k_alldifferent,
    [checker(k_alldifferent_c),
     reformulation(k_alldifferent_r)]).

ctr_contractible(k_alldifferent,[],'VARS',any).

k_alldifferent_c(VARS) :-
        length(VARS,N),
        N>0,
        collection(VARS,[non_empty_col([int])]),
        get_col_attr1(VARS,1,VS),
        k_alldifferent0(VS).

k_alldifferent0([]).

k_alldifferent0([V|R]) :-
        sort(V,S),
        length(V,N),
        length(S,N),
        k_alldifferent0(R).

k_alldifferent_r(VARS) :-
        length(VARS,N),
        N>0,
        collection(VARS,[non_empty_col([dvar])]),
        get_col_attr1(VARS,1,VS),
        k_alldifferent1(VS).

k_alldifferent1([]).

k_alldifferent1([V|R]) :-
        all_different(V),
        k_alldifferent1(R).
```

B.208 k_cut

◇ META-DATA:

```
ctr_date(k_cut,['20030820','20041230','20060811']).

ctr_origin(k_cut,'E.~Althaus',[]).

ctr_arguments(
    k_cut,
    ['K'-int,'NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    k_cut,
    ['K'>=1,
     'K'=<size('NODES'),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    k_cut,
    k_cut(
        3,
        [[index-1,succ-{}],
         [index-2,succ-{3,5}],
         [index-3,succ-{5}],
         [index-4,succ-{}],
         [index-5,succ-{2,3}]])).

ctr_typical(k_cut,[size('NODES')>1]).

ctr_exchangeable(
    k_cut,
    [vals(['K'],int(>=(1)),>,dontcare,dontcare),
     items('NODES',all)]).

ctr_graph(
    k_cut,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^index=nodes2^index#\/
```

```
 nodes2^index in_set nodes1^succ],
['NCC'>='K'],
[]).
```

B.209 k_disjoint

◇ META-DATA:

```
ctr_date(k_disjoint,['20050816','20060811']).

ctr_origin(k_disjoint,'Derived from %c',[disjoint]).

ctr_types(k_disjoint,['VARIABLES'-collection(var-dvar)]).

ctr_arguments(k_disjoint,['SETS'-collection(set-'VARIABLES')]).

ctr_restrictions(
    k_disjoint,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1]).

ctr_example(
    k_disjoint,
    k_disjoint(
        [[set-[[var-1],[var-9],[var-1],[var-5]]],
         [set-
          [[var-2],[var-7],[var-7],[var-0],[var-6],[var-8]]],
         [set-[[var-4],[var-4],[var-3]]]])).

ctr_typical(k_disjoint,[size('VARIABLES')>1]).

ctr_exchangeable(
    k_disjoint,
    [items('SETS',all),
     items('SETS'^set,all),
     vals(['VARIABLES'^var],int,=\=,dontcare,in),
     vals(['SETS'^set^var],int,=\=,all,dontcare)]).

ctr_graph(
    k_disjoint,
    ['SETS'],
    2,
    ['CLIQUE'(<)>>collection(set1,set2)],
    [disjoint(set1^set,set2^set)],
    ['NARC'=size('SETS')*(size('SETS')-1)/2],
    []).

ctr_eval(k_disjoint,[reformulation(k_disjoint_r)]).
```

```
ctr_contractible(k_disjoint,[],'SETS',any).

k_disjoint_r(SETS) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        get_attr1(SETS,VARS),
        k_disjoint1(VARS).

k_disjoint1([_36141]) :-
        !.

k_disjoint1([V1,V2|R]) :-
        k_disjoint2([V2|R],V1),
        k_disjoint1([V2|R]).

k_disjoint2([],_36138).

k_disjoint2([U|R],V) :-
        eval(disjoint(V,U)),
        k_disjoint2(R,V).
```

B.210 k_same

♢ META-DATA:

```
ctr_date(k_same,['20050808','20060811']).

ctr_origin(k_same,'\\cite{ElbassioniKatrielKutzMahajan05}',[]).

ctr_types(k_same,['VARIABLES'-collection(var-dvar)]).

ctr_arguments(k_same,['SETS'-collection(set-'VARIABLES')]).

ctr_restrictions(
    k_same,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1,
     same_size('SETS',set)]).

ctr_example(
    k_same,
    k_same(
        [[set-
          [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]]],
         [set-
          [[var-9],[var-1],[var-1],[var-1],[var-2],[var-5]]],
         [set-
          [[var-5],[var-2],[var-1],[var-1],[var-9],[var-1]]]])).

ctr_typical(k_same,[size('VARIABLES')>1]).

ctr_exchangeable(
    k_same,
    [items('SETS',all),
     items('SETS'^set,all),
     vals(['SETS'^set^var],int,=\=,all,dontcare)]).

ctr_graph(
    k_same,
    ['SETS'],
    2,
    ['PATH'>>collection(set1,set2)],
    [same(set1^set,set2^set)],
    ['NARC'=size('SETS')-1],
    []).
```

```
ctr_eval(k_same,[reformulation(k_same_r)]).

ctr_contractible(k_same,[],'SETS',any).

k_same_r(SETS) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        get_attr1(SETS,VARS),
        k_same1(VARS).

k_same1([_39647]) :-
        !.

k_same1([V1,V2|R]) :-
        eval(same(V1,V2)),
        k_same1([V2|R]).
```

B.211 k_same_interval

◇ META-DATA:

```
ctr_date(k_same_interval,['20050810','20060811']).

ctr_origin(
    k_same_interval,
    Derived from %c and from %c.,
    [same_interval,k_same]).

ctr_types(k_same_interval,['VARIABLES'-collection(var-dvar)]).

ctr_arguments(
    k_same_interval,
    ['SETS'-collection(set-'VARIABLES'),'SIZE_INTERVAL'-int]).

ctr_restrictions(
    k_same_interval,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1,
     same_size('SETS',set),
     'SIZE_INTERVAL'>0]).

ctr_example(
    k_same_interval,
    k_same_interval(
        [[set-
          [[var-1],[var-1],[var-6],[var-0],[var-1],[var-7]]],
         [set-
          [[var-8],[var-8],[var-0],[var-0],[var-1],[var-2]]],
         [set-
          [[var-2],[var-1],[var-1],[var-2],[var-6],[var-6]]]],
        3)).

ctr_typical(
    k_same_interval,
    [size('VARIABLES')>1,'SIZE_INTERVAL'>1]).

ctr_exchangeable(
    k_same_interval,
    [items('SETS',all),
     items('SETS'^set,all),
     vals(
```

```
          ['SETS'^set^var],
          intervals('SIZE_INTERVAL'),
          =,
          dontcare,
          dontcare)]).

ctr_graph(
    k_same_interval,
    ['SETS'],
    2,
    ['PATH'>>collection(set1,set2)],
    [same_interval(set1^set,set2^set,'SIZE_INTERVAL')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(k_same_interval,[reformulation(k_same_interval_r)]).

ctr_contractible(k_same_interval,[],'SETS',any).

k_same_interval_r(SETS,SIZE_INTERVAL) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
        get_attr1(SETS,VARS),
        k_same_interval1(VARS,SIZE_INTERVAL).

k_same_interval1([_37967],_37966) :-
        !.

k_same_interval1([V1,V2|R],SIZE_INTERVAL) :-
        eval(same_interval(V1,V2,SIZE_INTERVAL)),
        k_same_interval1([V2|R],SIZE_INTERVAL).
```

B.212 k_same_modulo

◇ META-DATA:

```
ctr_date(k_same_modulo,['20050810','20060811']).

ctr_origin(
    k_same_modulo,
    Derived from %c and from %c.,
    [same_modulo,k_same]).

ctr_types(k_same_modulo,['VARIABLES'-collection(var-dvar)]).

ctr_arguments(
    k_same_modulo,
    ['SETS'-collection(set-'VARIABLES'),'M'-int]).

ctr_restrictions(
    k_same_modulo,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1,
     same_size('SETS',set),
     'M'>0]).

ctr_example(
    k_same_modulo,
    k_same_modulo(
        [[set-
          [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]]],
         [set-
          [[var-6],[var-4],[var-1],[var-1],[var-5],[var-5]]],
         [set-
          [[var-1],[var-3],[var-4],[var-2],[var-8],[var-7]]]],
        3)).

ctr_typical(k_same_modulo,[size('VARIABLES')>1,'M'>1]).

ctr_exchangeable(
    k_same_modulo,
    [items('SETS',all),
     items('SETS'^set,all),
     vals(['SETS'^set^var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
```

```
    k_same_modulo,
    ['SETS'],
    2,
    ['PATH'>>collection(set1,set2)],
    [same_modulo(set1^set,set2^set,'M')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(k_same_modulo,[reformulation(k_same_modulo_r)]).

ctr_contractible(k_same_modulo,[],'SETS',any).

k_same_modulo_r(SETS,M) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        integer(M),
        M=\=0,
        get_attr1(SETS,VARS),
        k_same_modulo1(VARS,M).

k_same_modulo1([_37607],_37606) :-
        !.

k_same_modulo1([V1,V2|R],M) :-
        eval(same_modulo(V1,V2,M)),
        k_same_modulo1([V2|R],M).
```

B.213 k_same_partition

◇ **META-DATA:**

```
ctr_date(k_same_partition,['20050810','20060811']).

ctr_origin(
    k_same_partition,
    Derived from %c and from %c.,
    [same_partition,k_same]).

ctr_types(
    k_same_partition,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_arguments(
    k_same_partition,
    ['SETS'-collection(set-'VARIABLES'),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    k_same_partition,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     required('SETS',set),
     size('SETS')>1,
     same_size('SETS',set),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    k_same_partition,
    k_same_partition(
        [[set-
           [[var-1],[var-2],[var-6],[var-3],[var-1],[var-2]]],
         [set-
           [[var-6],[var-6],[var-2],[var-3],[var-1],[var-3]]],
         [set-
           [[var-2],[var-2],[var-2],[var-1],[var-1],[var-1]]]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).
```

```
ctr_typical(k_same_partition,[size('VARIABLES')>1]).

ctr_exchangeable(
    k_same_partition,
    [items('SETS',all),
     items('SETS'^set,all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['SETS'^set^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    k_same_partition,
    ['SETS'],
    2,
    ['PATH'>>collection(set1,set2)],
    [same_partition(set1^set,set2^set,'PARTITIONS')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(k_same_partition,[reformulation(k_same_partition_r)]).

ctr_contractible(k_same_partition,[],'SETS',any).

k_same_partition_r(SETS,PARTITIONS) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        length(PARTITIONS,P),
        P>1,
        get_attr1(SETS,VARS),
        k_same_partition1(VARS,PARTITIONS).

k_same_partition1([_38961],_38960) :-
        !.

k_same_partition1([V1,V2|R],PARTITIONS) :-
        eval(same_partition(V1,V2,PARTITIONS)),
        k_same_partition1([V2|R],PARTITIONS).
```

B.214 k_used_by

♢ META-DATA:

```
ctr_date(k_used_by,['20050814','20060811']).

ctr_origin(k_used_by,'Derived from %c',[used_by]).

ctr_types(k_used_by,['VARIABLES'-collection(var-dvar)]).

ctr_arguments(k_used_by,['SETS'-collection(set-'VARIABLES')]).

ctr_restrictions(
    k_used_by,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1,
     non_increasing_size('SETS',set)]).

ctr_example(
    k_used_by,
    k_used_by(
        [[set-
          [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]]],
         [set-
          [[var-9],[var-1],[var-1],[var-1],[var-2],[var-5]]],
         [set-[[var-1],[var-1],[var-2],[var-5]]]])).

ctr_typical(k_used_by,[size('VARIABLES')>1]).

ctr_exchangeable(
    k_used_by,
    [items('SETS',all),
     items('SETS'^set,all),
     vals(['SETS'^set^var],int,=\=,all,dontcare)]).

ctr_graph(
    k_used_by,
    ['SETS'],
    2,
    ['PATH'>>collection(set1,set2)],
    [used_by(set1^set,set2^set)],
    ['NARC'=size('SETS')-1],
    []).
```

```
ctr_eval(k_used_by,[reformulation(k_used_by_r)]).

ctr_contractible(k_used_by,[],'SETS',any).

k_used_by_r(SETS) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        get_attr1(SETS,VARS),
        k_used_by1(VARS).

k_used_by1([_39054]) :-
        !.

k_used_by1([V1,V2|R]) :-
        eval(used_by(V1,V2)),
        k_used_by1([V2|R]).
```

B.215 k_used_by_interval

♢ META-DATA:

```
ctr_date(k_used_by_interval,['20050814','20060811']).

ctr_origin(
    k_used_by_interval,
    Derived from %c and from %c.,
    [used_by_interval,k_used_by]).

ctr_types(
    k_used_by_interval,
    ['VARIABLES'-collection(var-dvar)]).

ctr_arguments(
    k_used_by_interval,
    ['SETS'-collection(set-'VARIABLES'),'SIZE_INTERVAL'-int]).

ctr_restrictions(
    k_used_by_interval,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1,
     non_increasing_size('SETS',set),
     'SIZE_INTERVAL'>0]).

ctr_example(
    k_used_by_interval,
    k_used_by_interval(
        [[set-
          [[var-1],[var-1],[var-1],[var-8],[var-6],[var-2]]],
         [set-[[var-1],[var-0],[var-7],[var-7]]],
         [set-[[var-1],[var-2]]]],
        3)).

ctr_typical(
    k_used_by_interval,
    [size('VARIABLES')>1,'SIZE_INTERVAL'>0]).

ctr_exchangeable(
    k_used_by_interval,
    [items('SETS',all),
     items('SETS'^set,all),
     vals(
```

```
          ['SETS'^set^var],
          intervals('SIZE_INTERVAL'),
          =,
          dontcare,
          dontcare)]).

ctr_graph(
    k_used_by_interval,
    ['SETS'],
    2,
    ['PATH'>>collection(set1,set2)],
    [used_by_interval(set1^set,set2^set,'SIZE_INTERVAL')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(
    k_used_by_interval,
    [reformulation(k_used_by_interval_r)]).

ctr_contractible(k_used_by_interval,[],'SETS',any).

k_used_by_interval_r(SETS,SIZE_INTERVAL) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
        get_attr1(SETS,VARS),
        k_used_by_interval1(VARS,SIZE_INTERVAL).

k_used_by_interval1([_37435],_37434) :-
        !.

k_used_by_interval1([V1,V2|R],SIZE_INTERVAL) :-
        eval(used_by_interval(V1,V2,SIZE_INTERVAL)),
        k_used_by_interval1([V2|R],SIZE_INTERVAL).
```

B.216 k_used_by_modulo

◇ **META-DATA:**

```
ctr_date(k_used_by_modulo,['20050814','20060811']).

ctr_origin(
    k_used_by_modulo,
    Derived from %c and from %c.,
    [used_by_modulo,k_used_by]).

ctr_types(k_used_by_modulo,['VARIABLES'-collection(var-dvar)]).

ctr_arguments(
    k_used_by_modulo,
    ['SETS'-collection(set-'VARIABLES'),'M'-int]).

ctr_restrictions(
    k_used_by_modulo,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     required('SETS',set),
     size('SETS')>1,
     non_increasing_size('SETS',set),
     'M'>0]).

ctr_example(
    k_used_by_modulo,
    k_used_by_modulo(
        [[set-
          [[var-1],[var-9],[var-4],[var-5],[var-2],[var-1]]],
         [set-[[var-7],[var-1],[var-2],[var-5]]],
         [set-[[var-1],[var-1]]]],
        3)).

ctr_typical(k_used_by_modulo,[size('VARIABLES')>1,'M'>1]).

ctr_exchangeable(
    k_used_by_modulo,
    [items('SETS',all),
     items('SETS'^set,all),
     vals(['SETS'^set^var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
    k_used_by_modulo,
    ['SETS'],
```

```
    2,
    ['PATH'>>collection(set1,set2)],
    [used_by_modulo(set1^set,set2^set,'M')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(k_used_by_modulo,[reformulation(k_used_by_modulo_r)]).

ctr_contractible(k_used_by_modulo,[],'SETS',any).

k_used_by_modulo_r(SETS,M) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        integer(M),
        M=\=0,
        get_attr1(SETS,VARS),
        k_used_by_modulo1(VARS,M).

k_used_by_modulo1([_37354],_37353) :-
        !.

k_used_by_modulo1([V1,V2|R],M) :-
        eval(used_by_modulo(V1,V2,M)),
        k_used_by_modulo1([V2|R],M).
```

B.217 k_used_by_partition

◇ META-DATA:

```
ctr_date(k_used_by_partition,['20050814','20060811']).

ctr_origin(
    k_used_by_partition,
    Derived from %c and from %c.,
    [used_by_partition,k_used_by]).

ctr_types(
    k_used_by_partition,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_arguments(
    k_used_by_partition,
    ['SETS'-collection(set-'VARIABLES'),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    k_used_by_partition,
    [required('VARIABLES',var),
     size('VARIABLES')>=1,
     size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     required('SETS',set),
     size('SETS')>1,
     non_increasing_size('SETS',set),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    k_used_by_partition,
    k_used_by_partition(
        [[set-
           [[var-1],[var-9],[var-1],[var-6],[var-2],[var-3]]],
         [set-[[var-1],[var-3],[var-6],[var-6]]],
         [set-[[var-2],[var-2]]]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(k_used_by_partition,[size('VARIABLES')>1]).
```

```
ctr_exchangeable(
    k_used_by_partition,
    [items('SETS',all),
     items('SETS'^set,all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['SETS'^set^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    k_used_by_partition,
    ['SETS'],
    2,
    ['PATH'>>collection(set1,set2)],
    [used_by_partition(set1^set,set2^set,'PARTITIONS')],
    ['NARC'=size('SETS')-1],
    []).

ctr_eval(
    k_used_by_partition,
    [reformulation(k_used_by_partition_r)]).

ctr_contractible(k_used_by_partition,[],'SETS',any).

k_used_by_partition_r(SETS,PARTITIONS) :-
        length(SETS,N),
        N>1,
        collection(SETS,[non_empty_col([dvar])]),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        length(PARTITIONS,P),
        P>1,
        get_attr1(SETS,VARS),
        k_used_by_partition1(VARS,PARTITIONS).

k_used_by_partition1([_38220],_38219) :-
        !.

k_used_by_partition1([V1,V2|R],PARTITIONS) :-
        eval(used_by_partition(V1,V2,PARTITIONS)),
        k_used_by_partition1([V2|R],PARTITIONS).
```

B.218 length_first_sequence

◇ META-DATA:

```
ctr_date(length_first_sequence,['20081123']).

ctr_origin(
    length_first_sequence,
    Inspired by %c,
    [stretch_path]).

ctr_synonyms(length_first_sequence,[length_first_stretch]).

ctr_arguments(
    length_first_sequence,
    ['LEN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    length_first_sequence,
    ['LEN'>=0,
     'LEN'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    length_first_sequence,
    [length_first_sequence(
         3,
         [[var-4],[var-4],[var-4],[var-5],[var-5],[var-4]]),
     length_first_sequence(
         6,
         [[var-4],[var-4],[var-4],[var-4],[var-4],[var-4]]),
     length_first_sequence(
         5,
         [[var-4],[var-4],[var-4],[var-4],[var-4],[var-1]])]).

ctr_typical(
    length_first_sequence,
    ['LEN'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_typical_model(
    length_first_sequence,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    length_first_sequence,
    [vals(['VARIABLES'^var],int,=\=,all,dontcare)]).
```

```
ctr_eval(
    length_first_sequence,
    [checker(length_first_sequence_c),
     reformulation(length_first_sequence_r),
     automaton(length_first_sequence_a)]).

ctr_pure_functional_dependency(length_first_sequence,[]).

ctr_functional_dependency(length_first_sequence,1,[2]).

ctr_sol(length_first_sequence,2,0,2,9,[1-6,2-3]).

ctr_sol(length_first_sequence,3,0,3,64,[1-48,2-12,3-4]).

ctr_sol(length_first_sequence,4,0,4,625,[1-500,2-100,3-20,4-5]).

ctr_sol(
    length_first_sequence,
    5,
    0,
    5,
    7776,
    [1-6480,2-1080,3-180,4-30,5-6]).

ctr_sol(
    length_first_sequence,
    6,
    0,
    6,
    117649,
    [1-100842,2-14406,3-2058,4-294,5-42,6-7]).

ctr_sol(
    length_first_sequence,
    7,
    0,
    7,
    2097152,
    [1-1835008,2-229376,3-28672,4-3584,5-448,6-56,7-8]).

ctr_sol(
    length_first_sequence,
    8,
    0,
    8,
```

```
    43046721,
    [1-38263752,
     2-4251528,
     3-472392,
     4-52488,
     5-5832,
     6-648,
     7-72,
     8-9]).

length_first_sequence_c(LEN,VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),LEN),
        get_attr1(VARIABLES,VARS),
        length_first_sequence_c(N,LEN,VARS).

length_first_sequence_c(0,0,_49510) :-
        !.

length_first_sequence_c(1,1,_49510) :-
        !.

length_first_sequence_c(_49508,LEN,VARS) :-
        length_first_eq_sequence(VARS,1,LEN).

length_first_sequence_r(LEN,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        check_type(dvar(0,N),LEN),
        get_attr1(VARIABLES,VARS),
        (   N=0 ->
            LEN#=0
        ;   N=1 ->
            LEN#=1
        ;   reverse(VARS,RVARS),
            length_first_sequence1(RVARS,_49584,TERM),
            call(LEN#=TERM)
        ).

length_first_sequence1([_49511],1,1) :-
        !.

length_first_sequence1([VAR1,VAR2|R],AND1,AND1+S) :-
        length_first_sequence1([VAR2|R],AND2,S),
        B12#<=>VAR1#=VAR2,
```

```
        AND1#<=>AND2#/\B12.

length_first_sequence_a(1,0,[]) :-
        !.

length_first_sequence_a(0,0,[]) :-
        !,
        fail.

length_first_sequence_a(1,1,[_49511]) :-
        !.

length_first_sequence_a(0,1,[_49511]) :-
        !,
        fail.

length_first_sequence_a(FLAG,LEN,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        check_type(dvar(0,N),LEN),
        length_first_sequence_signature(VARIABLES,SIGNATURE),
        automaton(
            SIGNATURE,
            _50841,
            SIGNATURE,
            [source(s),sink(s),sink(t)],
            [arc(s,0,t),
             arc(s,1,s,[C+1]),
             arc(t,0,t),
             arc(t,1,t)],
            [C],
            [1],
            [COUNT]),
        COUNT#=LEN#<=>FLAG.

length_first_sequence_signature([],[]).

length_first_sequence_signature([_49510],[]) :-
        !.

length_first_sequence_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss]) :-
        S in 0..1,
        VAR1#=VAR2#<=>S,
        length_first_sequence_signature([[var-VAR2]|VARs],Ss).
```

B.219 length_last_sequence

◇ **META-DATA:**

```
ctr_date(length_last_sequence,['20081123']).

ctr_origin(
    length_last_sequence,
    Inspired by %c,
    [stretch_path]).

ctr_synonyms(length_last_sequence,[length_last_stretch]).

ctr_arguments(
    length_last_sequence,
    ['LEN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    length_last_sequence,
    ['LEN'>=0,
     'LEN'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    length_last_sequence,
    [length_last_sequence(
         1,
         [[var-4],[var-4],[var-4],[var-5],[var-5],[var-4]]),
     length_last_sequence(
         6,
         [[var-4],[var-4],[var-4],[var-4],[var-4],[var-4]]),
     length_last_sequence(
         5,
         [[var-2],[var-4],[var-4],[var-4],[var-4],[var-4]])]).

ctr_typical(
    length_last_sequence,
    ['LEN'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_typical_model(
    length_last_sequence,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    length_last_sequence,
    [vals(['VARIABLES'^var],int,=\=,all,dontcare)]).
```

```
ctr_eval(
    length_last_sequence,
    [checker(length_last_sequence_c),
     reformulation(length_last_sequence_r),
     automaton(length_last_sequence_a)]).

ctr_pure_functional_dependency(length_last_sequence,[]).

ctr_functional_dependency(length_last_sequence,1,[2]).

ctr_sol(length_last_sequence,2,0,2,9,[1-6,2-3]).

ctr_sol(length_last_sequence,3,0,3,64,[1-48,2-12,3-4]).

ctr_sol(length_last_sequence,4,0,4,625,[1-500,2-100,3-20,4-5]).

ctr_sol(
    length_last_sequence,
    5,
    0,
    5,
    7776,
    [1-6480,2-1080,3-180,4-30,5-6]).

ctr_sol(
    length_last_sequence,
    6,
    0,
    6,
    117649,
    [1-100842,2-14406,3-2058,4-294,5-42,6-7]).

ctr_sol(
    length_last_sequence,
    7,
    0,
    7,
    2097152,
    [1-1835008,2-229376,3-28672,4-3584,5-448,6-56,7-8]).

ctr_sol(
    length_last_sequence,
    8,
    0,
    8,
```

```
    43046721,
    [1-38263752,
     2-4251528,
     3-472392,
     4-52488,
     5-5832,
     6-648,
     7-72,
     8-9]).

length_last_sequence_c(LEN,VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),LEN),
        get_attr1(VARIABLES,VARS),
        length_last_sequence_c(N,LEN,VARS).

length_last_sequence_c(0,0,_49510) :-
        !.

length_last_sequence_c(1,1,_49510) :-
        !.

length_last_sequence_c(_49508,LEN,VARS) :-
        reverse(VARS,RVARS),
        length_first_eq_sequence(RVARS,1,LEN).

length_last_sequence_r(LEN,VARIABLES) :-
        check_type(dvar,LEN),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        (   N=0 ->
            LEN#=0
        ;   N=1 ->
            LEN#=1
        ;   length_last_sequence1(VARS,_49573,TERM),
            call(LEN#=TERM)
        ).

length_last_sequence1([_49511],1,1) :-
        !.

length_last_sequence1([VAR1,VAR2|R],AND1,AND1+S) :-
        length_last_sequence1([VAR2|R],AND2,S),
        B12#<=>VAR1#=VAR2,
```

```
        AND1#<=>AND2#/\B12.

length_last_sequence_a(1,0,[]) :-
        !.

length_last_sequence_a(0,0,[]) :-
        !,
        fail.

length_last_sequence_a(1,1,[_49511]) :-
        !.

length_last_sequence_a(0,1,[_49511]) :-
        !,
        fail.

length_last_sequence_a(FLAG,LEN,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        check_type(dvar(0,N),LEN),
        length_last_sequence_signature(VARIABLES,SIGNATURE),
        automaton(
            SIGNATURE,
            _50805,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s,[1]),arc(s,1,s,[C+1])],
            [C],
            [1],
            [COUNT]),
        COUNT#=LEN#<=>FLAG.

length_last_sequence_signature([],[]).

length_last_sequence_signature([_49510],[]) :-
        !.

length_last_sequence_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss]) :-
        S in 0..1,
        VAR1#=VAR2#<=>S,
        length_last_sequence_signature([[var-VAR2]|VARs],Ss).
```

B.220 leq

◊ **META-DATA:**

```
ctr_predefined(leq).

ctr_date(leq,['20070821']).

ctr_origin(leq,'Arithmetic.',[]).

ctr_synonyms(leq,[rel,xlteqy]).

ctr_arguments(leq,['VAR1'-dvar,'VAR2'-dvar]).

ctr_example(leq,leq(1,8)).

ctr_exchangeable(
    leq,
    [vals(['VAR1'],int(=<('VAR2')),=\=,all,dontcare),
     vals(['VAR2'],int(>=('VAR1')),=\=,all,dontcare)]).

ctr_eval(leq,[builtin(leq_b)]).

leq_b(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#=<VAR2.
```

B.221 leq_cst

◇ **META-DATA:**

```
ctr_predefined(leq_cst).

ctr_date(leq_cst,['20090912']).

ctr_origin(leq_cst,'Arithmetic.',[]).

ctr_arguments(leq_cst,['VAR1'-dvar,'VAR2'-dvar,'CST2'-int]).

ctr_example(leq_cst,leq_cst(5,2,4)).

ctr_typical(leq_cst,['CST2'=\=0,'VAR1'<'VAR2'+'CST2']).

ctr_exchangeable(
    leq_cst,
    [args([['VAR1'],['VAR2','CST2']]),
     vals(['VAR1'],int(=<('VAR2'+'CST2')),=\=,all,dontcare),
     vals(['VAR2'],int(>=('VAR1'-'CST2')),=\=,all,dontcare),
     vals(['CST2'],int(>=('VAR1'-'VAR2')),=\=,all,dontcare)]).

ctr_eval(leq_cst,[builtin(leq_cst_b)]).

leq_cst_b(VAR1,VAR2,CST2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        check_type(int,CST2),
        VAR1#=<VAR2+CST2.
```

B.222 lex2

◊ **META-DATA:**

```
ctr_predefined(lex2).

ctr_date(lex2,['20031008','20040530','20060811']).

ctr_origin(
    lex2,
    \cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02},
    []).

ctr_synonyms(lex2,[double_lex,row_and_column_lex]).

ctr_types(lex2,['VECTOR'-collection(var-dvar)]).

ctr_arguments(lex2,['MATRIX'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex2,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('MATRIX',vec),
     same_size('MATRIX',vec)]).

ctr_example(
    lex2,
    lex2(
        [[vec-[[var-2],[var-2],[var-3]]],
         [vec-[[var-2],[var-3],[var-1]]]])).

ctr_typical(lex2,[size('VECTOR')>1,size('MATRIX')>1]).

ctr_exchangeable(lex2,[translate(['MATRIX'^vec^var])]).

ctr_eval(lex2,[checker(lex2_c),reformulation(lex2_r)]).

lex2_c(MATRIX) :-
        collection(MATRIX,[col([int])]),
        same_size(MATRIX),
        get_attr11(MATRIX,MAT),
        lex_chain_lesseq_c1(MAT),
        transpose(MAT,TMAT),
        lex_chain_lesseq_c1(TMAT).
```

```
lex2_r(MATRIX) :-
        collection(MATRIX,[col([dvar])]),
        same_size(MATRIX),
        get_attr11(MATRIX,MAT),
        lex_chain(MAT,[op(#=<)]),
        transpose(MAT,TMAT),
        lex_chain(TMAT,[op(#=<)]).
```

B.223 lex_alldifferent

◇ **META-DATA:**

```
ctr_date(
    lex_alldifferent,
    ['20030820','20040530','20051008','20060811','20111102']).

ctr_origin(lex_alldifferent,'J.~Pearson',[]).

ctr_synonyms(
    lex_alldifferent,
    [lex_alldiff,
     lex_alldistinct,
     alldiff_on_tuples,
     alldifferent_on_tuples,
     alldistinct_on_tuples]).

ctr_types(lex_alldifferent,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_alldifferent,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_alldifferent,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    lex_alldifferent,
    lex_alldifferent(
        [[vec-[[var-5],[var-2],[var-3]]],
         [vec-[[var-5],[var-2],[var-6]]],
         [vec-[[var-5],[var-3],[var-3]]]])).

ctr_typical(
    lex_alldifferent,
    [size('VECTOR')>1,size('VECTORS')>1]).

ctr_exchangeable(
    lex_alldifferent,
    [items('VECTORS',all),
     items_sync('VECTORS'^vec,all),
```

```
     vals(['VECTORS'^vec],int,=\=,all,dontcare)]).

ctr_graph(
    lex_alldifferent,
    ['VECTORS'],
    2,
    ['CLIQUE'(<)>>collection(vectors1,vectors2)],
    [lex_different(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')*(size('VECTORS')-1)/2],
    []).

ctr_eval(
    lex_alldifferent,
    [checker(lex_alldifferent_c),
     reformulation(lex_alldifferent_r),
     density(lex_alldifferent_d)]).

ctr_contractible(lex_alldifferent,[],'VECTORS',any).

ctr_extensible(lex_alldifferent,[],'VECTORS'^vec,any).

lex_alldifferent_c([]) :-
        !.

lex_alldifferent_c(VECTORS) :-
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        length(VECTORS,L),
        sort(VECTORS,SVECTORS),
        length(SVECTORS,L).

lex_alldifferent_r([]) :-
        !.

lex_alldifferent_r(VECTORS) :-
        collection(VECTORS,[col([dvar])]),
        lex_alldifferent1(VECTORS).

lex_alldifferent1([]).

lex_alldifferent1([[_46050-VECTOR]|R]) :-
        lex_alldifferent2(R,VECTOR),
        lex_alldifferent1(R).

lex_alldifferent2([],_46042).
```

```
lex_alldifferent2([[_46051-VECTORi]|R],VECTOR) :-
        eval(lex_different(VECTOR,VECTORi)),
        lex_alldifferent2(R,VECTOR).

lex_alldifferent_d(Density,VECTORS) :-
        lex_alldifferent_density(Density,VECTORS).
```

B.224 lex_alldifferent_except_0

♢ META-DATA:

```
ctr_predefined(lex_alldifferent_except_0).

ctr_date(lex_alldifferent_except_0,['20120515']).

ctr_origin(lex_alldifferent_except_0,'H.~Simonis',[]).

ctr_synonyms(
    lex_alldifferent_except_0,
    [lex_alldiff_except_0,
     lex_alldistinct_except_0,
     alldiff_on_tuples_except_0,
     alldifferent_on_tuples_except_0,
     alldistinct_on_tuples_except_0]).

ctr_types(
    lex_alldifferent_except_0,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_alldifferent_except_0,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_alldifferent_except_0,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    lex_alldifferent_except_0,
    lex_alldifferent_except_0(
        [[vec-[[var-0],[var-0],[var-0]]],
         [vec-[[var-5],[var-2],[var-0]]],
         [vec-[[var-5],[var-8],[var-0]]],
         [vec-[[var-0],[var-0],[var-0]]]])).

ctr_typical(
    lex_alldifferent_except_0,
    [size('VECTOR')>1,size('VECTORS')>1]).

ctr_eval(
```

```
    lex_alldifferent_except_0,
    [reformulation(lex_alldifferent_except_0_r),
     checker(lex_alldifferent_except_0_c),
     density(lex_alldifferent_except_0_d)]).

ctr_contractible(lex_alldifferent_except_0,[],'VECTORS',any).

lex_alldifferent_except_0_c([]) :-
        !.

lex_alldifferent_except_0_c(VECTORS) :-
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        length(VECTORS,L),
        count_zero_vectors(VECTORS,0,Z),
        sort(VECTORS,SVECTORS),
        length(SVECTORS,S),
        (   Z=0 ->
            S1=S
        ;   S1 is S-1
        ),
        L=:=S1+Z.

lex_alldifferent_except_0_d(Density,VECTORS) :-
        remove_zeros(VECTORS,VECTORS1),
        count_zero_vectors(VECTORS1,0,Z),
        length(VECTORS1,L),
        VECTORS1=[[vec-V]|_26782],
        length(V,M),
        Density1 is Z/(L*M),
        lex_alldifferent_density(Density2,VECTORS1),
        Density is min(Density1,Density2).

count_zero_vectors([],Z,Z) :-
        !.

count_zero_vectors([[vec-V]|R],Cur,Z) :-
        V=[[var-_26774]|_26769],
        (   zero_vector(V) ->
            Next is Cur+1
        ;   Next is Cur
        ),
        count_zero_vectors(R,Next,Z).

remove_zeros([],[]) :-
        !.
```

```
remove_zeros([[vec-V]|R],S) :-
        zero_vector(V),
        !,
        remove_zeros(R,S).

remove_zeros([VEC|R],[VEC|S]) :-
        remove_zeros(R,S).

lex_alldifferent_except_0_r([]) :-
        !.

lex_alldifferent_except_0_r(VECTORS) :-
        collection(VECTORS,[col([dvar])]),
        lex_alldifferent_except_01(VECTORS).

lex_alldifferent_except_01([]) :-
        !.

lex_alldifferent_except_01([[_26750-VECTOR]|R]) :-
        VECTOR=[[var-_26768]|_26763],
        lex_different_except_zero(VECTOR,ZERO),
        lex_alldifferent_except_02(R,VECTOR,ZERO),
        lex_alldifferent_except_01(R).

lex_alldifferent_except_02([],_26745,_26746) :-
        !.

lex_alldifferent_except_02([[_26752-VECTORi]|R],VECTOR,ZERO) :-
        lex_different_except_03(VECTOR,VECTORi,DIFF),
        call(ZERO#\/DIFF),
        lex_alldifferent_except_02(R,VECTOR,ZERO).

lex_different_except_03([],[],0) :-
        !.

lex_different_except_03([[var-U]|R],[[var-V]|S],U#\=V#\/T) :-
        lex_different_except_03(R,S,T).

lex_different_except_zero([],1) :-
        !.

lex_different_except_zero([[var-V]|R],V#=0#/\S) :-
        lex_different_except_zero(R,S).
```

B.225 lex_between

◊ **META-DATA:**

```
ctr_date(lex_between,['20030820','20040530','20060811']).

ctr_origin(lex_between,'\\cite{BeldiceanuCarlsson02c}',[]).

ctr_synonyms(lex_between,[between]).

ctr_arguments(
    lex_between,
    ['LOWER_BOUND'-collection(var-int),
     'VECTOR'-collection(var-dvar),
     'UPPER_BOUND'-collection(var-int)]).

ctr_restrictions(
    lex_between,
    [required('LOWER_BOUND',var),
     required('VECTOR',var),
     required('UPPER_BOUND',var),
     size('LOWER_BOUND')=size('VECTOR'),
     size('UPPER_BOUND')=size('VECTOR'),
     lex_lesseq('LOWER_BOUND','VECTOR'),
     lex_lesseq('VECTOR','UPPER_BOUND')]).

ctr_example(
    lex_between,
    lex_between(
        [[var-5],[var-2],[var-3],[var-9]],
        [[var-5],[var-2],[var-6],[var-2]],
        [[var-5],[var-2],[var-6],[var-3]])).

ctr_typical(
    lex_between,
    [size('LOWER_BOUND')>1,
     lex_lesseq('LOWER_BOUND','UPPER_BOUND')]).

ctr_exchangeable(
    lex_between,
    [vals(['LOWER_BOUND'^var],int,>,dontcare,dontcare),
     vals(['UPPER_BOUND'^var],int,<,dontcare,dontcare)]).

ctr_eval(
    lex_between,
    [reformulation(lex_between_r),automaton(lex_between_a)]).
```

```
ctr_contractible(
    lex_between,
    [],
    ['LOWER_BOUND','VECTOR','UPPER_BOUND'],
    suffix).

lex_between_r(LOWER_BOUND,VECTOR,UPPER_BOUND) :-
        collection(LOWER_BOUND,[int]),
        collection(VECTOR,[dvar]),
        collection(UPPER_BOUND,[int]),
        length(LOWER_BOUND,LB),
        length(VECTOR,LV),
        length(UPPER_BOUND,LU),
        LB=LV,
        LU=LV,
        eval(lex_lesseq(LOWER_BOUND,VECTOR)),
        eval(lex_lesseq(VECTOR,UPPER_BOUND)).

lex_between_a(FLAG,LOWER_BOUND,VECTOR,UPPER_BOUND) :-
        collection(LOWER_BOUND,[int]),
        collection(VECTOR,[dvar]),
        collection(UPPER_BOUND,[int]),
        length(LOWER_BOUND,LB),
        length(VECTOR,LV),
        length(UPPER_BOUND,LU),
        LB=LV,
        LU=LV,
        lex_between_signature(
            LOWER_BOUND,
            VECTOR,
            UPPER_BOUND,
            SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _38035,
            SIGNATURE,
            [source(s),sink(a),sink(b),sink(s),sink(t)],
            [arc(s,4,s),
             arc(s,0,t),
             arc(s,3,a),
             arc(s,1,b),
             arc(a,3,a),
             arc(a,4,a),
             arc(a,5,a),
```

```
            arc(a,0,t),
            arc(a,1,t),
            arc(a,2,t),
            arc(b,1,b),
            arc(b,4,b),
            arc(b,7,b),
            arc(b,0,t),
            arc(b,3,t),
            arc(b,6,t),
            arc(t,0,t),
            arc(t,1,t),
            arc(t,2,t),
            arc(t,3,t),
            arc(t,4,t),
            arc(t,5,t),
            arc(t,6,t),
            arc(t,7,t),
            arc(t,8,t)],
           [],
           [],
           []),
        automaton_bool(FLAG,[0,1,2,3,4,5,6,7,8],AUTOMATON).

lex_between_signature([],[],[],[]).

lex_between_signature(
    [[var-A1]|As],
    [[var-X1]|Xs],
    [[var-B1]|Bs],
    [L1|Ls]) :-
        Adown is A1-1,
        Aup is A1+1,
        Bdown is B1-1,
        Bup is B1+1,
        (   A1+1<B1 ->
            case(
                X-L,
                [X1-L1],
                [node(
                     -1,
                     X,
                     [(inf..Adown)-6,
                      (A1..A1)-3,
                      (Aup..Bdown)-0,
                      (B1..B1)-1,
                      (Bup..sup)-2]),
```

```
            node(0,L,[0..0]),
            node(1,L,[1..1]),
            node(2,L,[2..2]),
            node(3,L,[3..3]),
            node(6,L,[6..6])])
;   A1<B1 ->
    case(
        X-L,
        [X1-L1],
        [node(
             -1,
             X,
             [(inf..Adown)-6,
              (A1..A1)-3,
              (B1..B1)-1,
              (Bup..sup)-2]),
            node(1,L,[1..1]),
            node(2,L,[2..2]),
            node(3,L,[3..3]),
            node(6,L,[6..6])])
;   A1=:=B1 ->
    case(
        X-L,
        [X1-L1],
        [node(
             -1,
             X,
             [(inf..Adown)-6,
              (A1..A1)-4,
              (Aup..sup)-2]),
            node(2,L,[2..2]),
            node(4,L,[4..4]),
            node(6,L,[6..6])])
;   A1=:=B1+1 ->
    case(
        X-L,
        [X1-L1],
        [node(
             -1,
             X,
             [(inf..Bdown)-6,
              (B1..B1)-7,
              (A1..A1)-5,
              (Aup..sup)-2]),
            node(2,L,[2..2]),
            node(5,L,[5..5]),
```

```
          node(6,L,[6..6]),
          node(7,L,[7..7])])
;   A1>B1 ->
    case(
        X-L,
        [X1-L1],
        [node(
             -1,
             X,
             [(inf..Bdown)-6,
              (B1..B1)-7,
              (Bup..Adown)-8,
              (A1..A1)-5,
              (Aup..sup)-2]),
         node(2,L,[2..2]),
         node(5,L,[5..5]),
         node(6,L,[6..6]),
         node(7,L,[7..7]),
         node(8,L,[8..8])])
),
lex_between_signature(As,Xs,Bs,Ls).
```

B.226 lex_chain_greater

◇ **META-DATA:**

```
ctr_date(lex_chain_greater,['20130730']).

ctr_origin(
    lex_chain_greater,
    Derived from %c,
    [lex_chain_less]).

ctr_usual_name(lex_chain_greater,lex_chain).

ctr_types(lex_chain_greater,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_chain_greater,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_chain_greater,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    lex_chain_greater,
    lex_chain_greater(
        [[vec-[[var-5],[var-2],[var-6],[var-3]]],
         [vec-[[var-5],[var-2],[var-6],[var-2]]],
         [vec-[[var-5],[var-2],[var-3],[var-9]]]])).

ctr_typical(
    lex_chain_greater,
    [size('VECTOR')>1,size('VECTORS')>1]).

ctr_graph(
    lex_chain_greater,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_greater(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')-1],
    []).
```

```
ctr_eval(
    lex_chain_greater,
    [checker(lex_chain_greater_c),
     builtin(lex_chain_greater_b)]).

ctr_contractible(lex_chain_greater,[],'VECTORS',any).

ctr_extensible(lex_chain_greater,[],'VECTORS'^vec,suffix).

lex_chain_greater_c(VECTORS) :-
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        reverse(VECTS,RVECTS),
        lex_chain_less_c1(RVECTS).

lex_chain_greater_b(VECTORS) :-
        collection(VECTORS,[col([dvar])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        reverse(VECTS,RVECTS),
        lex_chain(RVECTS,[op(#<)]).
```

B.227 lex_chain_greatereq

◇ META-DATA:

```
ctr_date(lex_chain_greatereq,['20130730']).

ctr_origin(
    lex_chain_greatereq,
    Derived from %c,
    [lex_chain_lesseq]).

ctr_usual_name(lex_chain_greatereq,lex_chain).

ctr_types(lex_chain_greatereq,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_chain_greatereq,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_chain_greatereq,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    lex_chain_greatereq,
    lex_chain_greatereq(
        [[vec-[[var-5],[var-2],[var-6],[var-2]]],
         [vec-[[var-5],[var-2],[var-6],[var-2]]],
         [vec-[[var-5],[var-2],[var-3],[var-9]]]])).

ctr_typical(
    lex_chain_greatereq,
    [size('VECTOR')>1,size('VECTORS')>1]).

ctr_graph(
    lex_chain_greatereq,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_lesseq(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')-1],
    []).
```

```
ctr_eval(
    lex_chain_greatereq,
    [checker(lex_chain_greatereq_c),
     builtin(lex_chain_greatereq_b)]).

ctr_contractible(lex_chain_greatereq,[],'VECTORS',any).

ctr_contractible(lex_chain_greatereq,[],'VECTORS'^vec,suffix).

lex_chain_greatereq_c(VECTORS) :-
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        reverse(VECTS,RVECTS),
        lex_chain_lesseq_c1(RVECTS).

lex_chain_greatereq_b(VECTORS) :-
        collection(VECTORS,[col([dvar])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        reverse(VECTS,RVECTS),
        lex_chain(RVECTS,[op(#=<)]).
```

B.228 lex_chain_less

◇ META-DATA:

```
ctr_date(
    lex_chain_less,
    ['20030820','20040530','20060811','20090116']).

ctr_origin(lex_chain_less,'\\cite{BeldiceanuCarlsson02c}',[]).

ctr_usual_name(lex_chain_less,lex_chain).

ctr_types(lex_chain_less,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_chain_less,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_chain_less,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    lex_chain_less,
    lex_chain_less(
        [[vec-[[var-5],[var-2],[var-3],[var-9]]],
         [vec-[[var-5],[var-2],[var-6],[var-2]]],
         [vec-[[var-5],[var-2],[var-6],[var-3]]]])).

ctr_typical(
    lex_chain_less,
    [size('VECTOR')>1,size('VECTORS')>1]).

ctr_graph(
    lex_chain_less,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_less(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')-1],
    []).

ctr_eval(
```

```
    lex_chain_less,
    [checker(lex_chain_less_c),builtin(lex_chain_less_b)]).

ctr_contractible(lex_chain_less,[],'VECTORS',any).

ctr_extensible(lex_chain_less,[],'VECTORS'^vec,suffix).

lex_chain_less_c(VECTORS) :-
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        lex_chain_less_c1(VECTS).

lex_chain_less_b(VECTORS) :-
        collection(VECTORS,[col([dvar])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        lex_chain(VECTS,[op(#<)]).
```

B.229 lex_chain_lesseq

◇ **META-DATA:**

```
ctr_date(
    lex_chain_lesseq,
    ['20030820','20040530','20060811','20090116']).

ctr_origin(lex_chain_lesseq,'\\cite{BeldiceanuCarlsson02c}',[]).

ctr_usual_name(lex_chain_lesseq,lex_chain).

ctr_types(lex_chain_lesseq,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    lex_chain_lesseq,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    lex_chain_lesseq,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    lex_chain_lesseq,
    lex_chain_lesseq(
        [[vec-[[var-5],[var-2],[var-3],[var-9]]],
         [vec-[[var-5],[var-2],[var-6],[var-2]]],
         [vec-[[var-5],[var-2],[var-6],[var-2]]]])).

ctr_typical(
    lex_chain_lesseq,
    [size('VECTOR')>1,size('VECTORS')>1]).

ctr_graph(
    lex_chain_lesseq,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_lesseq(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')-1],
    []).

ctr_eval(
```

```
    lex_chain_lesseq,
    [checker(lex_chain_lesseq_c),builtin(lex_chain_lesseq_b)]).

ctr_contractible(lex_chain_lesseq,[],'VECTORS',any).

ctr_contractible(lex_chain_lesseq,[],'VECTORS'^vec,suffix).

lex_chain_lesseq_c(VECTORS) :-
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        lex_chain_lesseq_c1(VECTS).

lex_chain_lesseq_b(VECTORS) :-
        collection(VECTORS,[col([dvar])]),
        same_size(VECTORS),
        get_attr11(VECTORS,VECTS),
        lex_chain(VECTS,[op(#=<)]).
```

B.230 lex_different

◊ **META-DATA:**

```
ctr_date(lex_different,['20030820','20040530']).

ctr_origin(
    lex_different,
    Used for defining %c.,
    [lex_alldifferent]).

ctr_synonyms(lex_different,[different,diff]).

ctr_arguments(
    lex_different,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_different,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')>0,
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_different,
    lex_different(
        [[var-5],[var-2],[var-7],[var-1]],
        [[var-5],[var-3],[var-7],[var-1]])).

ctr_typical(
    lex_different,
    [size('VECTOR1')>1,
     range('VECTOR1'^var)>1,
     range('VECTOR2'^var)>1]).

ctr_exchangeable(
    lex_different,
    [args([['VECTOR1','VECTOR2']]),
     items_sync('VECTOR1','VECTOR2',all)]).

ctr_graph(
    lex_different,
    ['VECTOR1','VECTOR2'],
    2,
```

```
    ['PRODUCT'(=)>>collection(vector1,vector2)],
    [vector1^var=\=vector2^var],
    ['NARC'>=1],
    []).

ctr_eval(
    lex_different,
    [reformulation(lex_different_r),
     automaton(lex_different_a)]).

ctr_extensible(lex_different,[],['VECTOR1','VECTOR2'],any).

lex_different_r(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L1),
        L1>0,
        length(VECTOR2,L2),
        L1=L2,
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_different1(VECT1,VECT2,Term),
        call(Term).

lex_different1([],[],0).

lex_different1([V1|R1],[V2|R2],V1#\=V2#\/T) :-
        lex_different1(R1,R2,T).

lex_different_a(FLAG,VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L1),
        L1>0,
        length(VECTOR2,L2),
        L1=L2,
        lex_different_signature(VECTOR1,VECTOR2,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _45691,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,1,s),arc(s,0,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
```

```
          []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

lex_different_signature([],[],[]).

lex_different_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
        VAR1#=VAR2#<=>S,
        lex_different_signature(Xs,Ys,Ss).
```

B.231 lex_equal

◇ **META-DATA:**

```
ctr_date(lex_equal,['20081220']).

ctr_origin(
    lex_equal,
    Initially introduced for defining %c,
    [nvector]).

ctr_synonyms(lex_equal,[equal,eq]).

ctr_arguments(
    lex_equal,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_equal,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_equal,
    lex_equal(
        [[var-1],[var-9],[var-1],[var-5]],
        [[var-1],[var-9],[var-1],[var-5]])).

ctr_typical(
    lex_equal,
    [size('VECTOR1')>1,
     range('VECTOR1'^var)>1,
     range('VECTOR2'^var)>1]).

ctr_exchangeable(
    lex_equal,
    [args([['VECTOR1','VECTOR2']]),
     items_sync('VECTOR1','VECTOR2',all)]).

ctr_graph(
    lex_equal,
    ['VECTOR1','VECTOR2'],
    2,
    ['PRODUCT'(=)>>collection(vector1,vector2)],
```

```
    [vector1^var=vector2^var],
    ['NARC'=size('VECTOR1')],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(
    lex_equal,
    [reformulation(lex_equal_r),automaton(lex_equal_a)]).

ctr_contractible(lex_equal,[],['VECTOR1','VECTOR2'],any).

lex_equal_r(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L1),
        length(VECTOR2,L2),
        L1=L2,
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_equal1(VECT1,VECT2).

lex_equal1([],[]).

lex_equal1([V1|R1],[V2|R2]) :-
        V1#=V2,
        lex_equal1(R1,R2).

lex_equal_a(FLAG,VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L1),
        length(VECTOR2,L2),
        L1=L2,
        lex_equal_signature(VECTOR1,VECTOR2,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _50141,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

lex_equal_signature([],[],[]).
```

```
lex_equal_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
        S in 0..1,
        VAR1#=VAR2#<=>S,
        lex_equal_signature(Xs,Ys,Ss).
```

B.232 lex_greater

◇ META-DATA:

```
ctr_date(lex_greater,['20030820','20040530','20060811']).

ctr_origin(lex_greater,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(lex_greater,[lex,lex_chain,rel,greater,gt]).

ctr_arguments(
    lex_greater,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_greater,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_greater,
    lex_greater(
        [[var-5],[var-2],[var-7],[var-1]],
        [[var-5],[var-2],[var-6],[var-2]])).

ctr_typical(
    lex_greater,
    [size('VECTOR1')>1,
     size('VECTOR1')<5#\/
     nval(['VECTOR1'^var,'VECTOR2'^var])<2*size('VECTOR1'),
     maxval(['VECTOR1'^var,'VECTOR2'^var])=<1#\/
     2*size('VECTOR1')-
     max_nvalue(['VECTOR1'^var,'VECTOR2'^var])>
     2]).

ctr_exchangeable(
    lex_greater,
    [vals(['VECTOR1'^var],int,<,dontcare,dontcare),
     vals(['VECTOR2'^var],int,>,dontcare,dontcare)]).

ctr_derived_collections(
    lex_greater,
    [col('DESTINATION'-collection(index-int,x-int,y-int),
         [item(index-0,x-0,y-0)]),
```

```
     col('COMPONENTS'-collection(index-int,x-dvar,y-dvar),
         [item(
              index-'VECTOR1'^key,
              x-'VECTOR1'^var,
              y-'VECTOR2'^var)])]).

ctr_graph(
    lex_greater,
    ['COMPONENTS','DESTINATION'],
    2,
    ['PRODUCT'('PATH','VOID')>>collection(item1,item2)],
    [item2^index>0#/\item1^x=item1^y#\/
     item2^index=0#/\item1^x>item1^y],
    ['PATH_FROM_TO'(index,1,0)=1],
    []).

ctr_eval(
    lex_greater,
    [checker(lex_greater_c),
     builtin(lex_greater_b),
     automaton(lex_greater_a)]).

ctr_extensible(lex_greater,[],['VECTOR1','VECTOR2'],suffix).

lex_greater_c(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[int]),
        collection(VECTOR2,[int]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_greater_c1(VECT1,VECT2).

lex_greater_c1([V|R],[V|S]) :-
        !,
        lex_greater_c1(R,S).

lex_greater_c1([V1|_59724],[V2|_59728]) :-
        V1>V2.

lex_greater_b(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
```

```
        get_attr1(VECTOR2,VECT2),
        lex_chain([VECT2,VECT1],[op(#<)]).

lex_greater_a(FLAG,VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        lex_greater_signature(VECTOR1,VECTOR2,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _61480,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,2,s),
             arc(s,3,t),
             arc(t,1,t),
             arc(t,2,t),
             arc(t,3,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[1,2,3],AUTOMATON).

lex_greater_signature([],[],[]).

lex_greater_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
        S in 1..3,
        VAR1#<VAR2#<=>S#=1,
        VAR1#=VAR2#<=>S#=2,
        VAR1#>VAR2#<=>S#=3,
        lex_greater_signature(Xs,Ys,Ss).
```

B.233 lex_greatereq

◊ **META-DATA:**

```
ctr_date(lex_greatereq,['20030820','20040530','20060811']).

ctr_origin(lex_greatereq,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(
    lex_greatereq,
    [lexeq,lex_chain,rel,greatereq,geq,lex_geq]).

ctr_arguments(
    lex_greatereq,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_greatereq,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_greatereq,
    [lex_greatereq(
         [[var-5],[var-2],[var-8],[var-9]],
         [[var-5],[var-2],[var-6],[var-2]]),
     lex_greatereq(
         [[var-5],[var-2],[var-3],[var-9]],
         [[var-5],[var-2],[var-3],[var-9]])]).

ctr_typical(
    lex_greatereq,
    [size('VECTOR1')>1,
     size('VECTOR1')<5#\/
     nval(['VECTOR1'^var,'VECTOR2'^var])<2*size('VECTOR1'),
     maxval(['VECTOR1'^var,'VECTOR2'^var])=<1#\/
     2*size('VECTOR1')-
     max_nvalue(['VECTOR1'^var,'VECTOR2'^var])>
     2]).

ctr_exchangeable(
    lex_greatereq,
    [vals(['VECTOR1'^var],int,<,dontcare,dontcare),
     vals(['VECTOR2'^var],int,>,dontcare,dontcare)]).
```

```
ctr_derived_collections(
    lex_greatereq,
    [col('DESTINATION'-collection(index-int,x-int,y-int),
         [item(index-0,x-0,y-0)]),
     col('COMPONENTS'-collection(index-int,x-dvar,y-dvar),
         [item(
              index-'VECTOR1'^key,
              x-'VECTOR1'^var,
              y-'VECTOR2'^var)])]).

ctr_graph(
    lex_greatereq,
    ['COMPONENTS','DESTINATION'],
    2,
    ['PRODUCT'('PATH','VOID')>>collection(item1,item2)],
    [item2^index>0#/\item1^x=item1^y#\/
     item1^index<size('VECTOR1')#/\item2^index=0#/\
     item1^x>item1^y#\/
     item1^index=size('VECTOR1')#/\item2^index=0#/\
     item1^x>=item1^y],
    ['PATH_FROM_TO'(index,1,0)=1],
    []).

ctr_eval(
    lex_greatereq,
    [checker(lex_greatereq_c),
     builtin(lex_greatereq_b),
     automaton(lex_greatereq_a)]).

ctr_contractible(lex_greatereq,[],['VECTOR1','VECTOR2'],suffix).

lex_greatereq_c(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[int]),
        collection(VECTOR2,[int]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_greatereq_c1(VECT1,VECT2).

lex_greatereq_c1([],[]) :-
        !.

lex_greatereq_c1([V|R],[V|S]) :-
        !,
```

```
        lex_greatereq_c1(R,S).

lex_greatereq_c1([V1|_62328],[V2|_62332]) :-
        V1>V2.

lex_greatereq_b(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_chain([VECT2,VECT1],[op(#=<)]).

lex_greatereq_a(FLAG,VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        lex_greatereq_signature(VECTOR1,VECTOR2,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _64100,
            SIGNATURE,
            [source(s),sink(s),sink(t)],
            [arc(s,2,s),
             arc(s,3,t),
             arc(t,1,t),
             arc(t,2,t),
             arc(t,3,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[1,2,3],AUTOMATON).

lex_greatereq_signature([],[],[]).

lex_greatereq_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
        S in 1..3,
        VAR1#<VAR2#<=>S#=1,
        VAR1#=VAR2#<=>S#=2,
        VAR1#>VAR2#<=>S#=3,
        lex_greatereq_signature(Xs,Ys,Ss).
```

B.234 lex_less

◇ **META-DATA:**

```
ctr_date(lex_less,['20030820','20040530','20060811']).

ctr_origin(lex_less,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(lex_less,[lex,lex_chain,rel,less]).

ctr_arguments(
    lex_less,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_less,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_less,
    lex_less(
        [[var-5],[var-2],[var-3],[var-9]],
        [[var-5],[var-2],[var-6],[var-2]])).

ctr_typical(
    lex_less,
    [size('VECTOR1')>1,
     size('VECTOR1')<5#\/
     nval(['VECTOR1'^var,'VECTOR2'^var])<2*size('VECTOR1'),
     maxval(['VECTOR1'^var,'VECTOR2'^var])=<1#\/
     2*size('VECTOR1')-
     max_nvalue(['VECTOR1'^var,'VECTOR2'^var])>
     2]).

ctr_exchangeable(
    lex_less,
    [vals(['VECTOR1'^var],int,>,dontcare,dontcare),
     vals(['VECTOR2'^var],int,<,dontcare,dontcare)]).

ctr_derived_collections(
    lex_less,
    [col('DESTINATION'-collection(index-int,x-int,y-int),
         [item(index-0,x-0,y-0)]),
```

```
     col('COMPONENTS'-collection(index-int,x-dvar,y-dvar),
         [item(
              index-'VECTOR1'^key,
              x-'VECTOR1'^var,
              y-'VECTOR2'^var)])]).

ctr_graph(
    lex_less,
    ['COMPONENTS','DESTINATION'],
    2,
    ['PRODUCT'('PATH','VOID')>>collection(item1,item2)],
    [item2^index>0#/\item1^x=item1^y#\/
     item2^index=0#/\item1^x<item1^y],
    ['PATH_FROM_TO'(index,1,0)=1],
    []).

ctr_eval(
    lex_less,
    [checker(lex_less_c),
     builtin(lex_less_b),
     automaton(lex_less_a)]).

ctr_extensible(lex_less,[],['VECTOR1','VECTOR2'],suffix).

lex_less_c(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[int]),
        collection(VECTOR2,[int]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_less_c1(VECT1,VECT2).

lex_less_b(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_chain([VECT1,VECT2],[op(#<)]).

lex_less_a(FLAG,VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
```

```
        length(VECTOR2,L),
        lex_less_signature(VECTOR1,VECTOR2,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _61873,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,2,s),
             arc(s,1,t),
             arc(t,1,t),
             arc(t,2,t),
             arc(t,3,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[1,2,3],AUTOMATON).

lex_less_signature([],[],[]).

lex_less_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
        S in 1..3,
        VAR1#<VAR2#<=>S#=1,
        VAR1#=VAR2#<=>S#=2,
        VAR1#>VAR2#<=>S#=3,
        lex_less_signature(Xs,Ys,Ss).
```

B.235 lex_lesseq

◇ **META-DATA:**

```
ctr_date(lex_lesseq,['20030820','20040530','20060811']).

ctr_origin(lex_lesseq,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(
    lex_lesseq,
    [lexeq,lex_chain,rel,lesseq,leq,lex_leq]).

ctr_arguments(
    lex_lesseq,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_lesseq,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_lesseq,
    [lex_lesseq(
         [[var-5],[var-2],[var-3],[var-1]],
         [[var-5],[var-2],[var-6],[var-2]]),
     lex_lesseq(
         [[var-5],[var-2],[var-3],[var-9]],
         [[var-5],[var-2],[var-3],[var-9]])]).

ctr_typical(
    lex_lesseq,
    [size('VECTOR1')>1,
     size('VECTOR1')<5#\/
     nval(['VECTOR1'^var,'VECTOR2'^var])<2*size('VECTOR1'),
     maxval(['VECTOR1'^var,'VECTOR2'^var])=<1#\/
     2*size('VECTOR1')-
     max_nvalue(['VECTOR1'^var,'VECTOR2'^var])>
     2]).

ctr_exchangeable(
    lex_lesseq,
    [vals(['VECTOR1'^var],int,>,dontcare,dontcare),
     vals(['VECTOR2'^var],int,<,dontcare,dontcare)]).
```

```
ctr_derived_collections(
    lex_lesseq,
    [col('DESTINATION'-collection(index-int,x-int,y-int),
         [item(index-0,x-0,y-0)]),
     col('COMPONENTS'-collection(index-int,x-dvar,y-dvar),
         [item(
              index-'VECTOR1'^key,
              x-'VECTOR1'^var,
              y-'VECTOR2'^var)])]).

ctr_graph(
    lex_lesseq,
    ['COMPONENTS','DESTINATION'],
    2,
    ['PRODUCT'('PATH','VOID')>>collection(item1,item2)],
    [item2^index>0#/\item1^x=item1^y#\/
     item1^index<size('VECTOR1')#/\item2^index=0#/\
     item1^x<item1^y#\/
     item1^index=size('VECTOR1')#/\item2^index=0#/\
     item1^x=<item1^y],
    ['PATH_FROM_TO'(index,1,0)=1],
    []).

ctr_eval(
    lex_lesseq,
    [checker(lex_lesseq_c),
     builtin(lex_lesseq_b),
     automaton(lex_lesseq_a)]).

ctr_contractible(lex_lesseq,[],['VECTOR1','VECTOR2'],suffix).

lex_lesseq_c(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[int]),
        collection(VECTOR2,[int]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_lesseq_c1(VECT1,VECT2).

lex_lesseq_b(VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
        length(VECTOR2,L),
```

```
        get_attr1(VECTOR1,VECT1),
        get_attr1(VECTOR2,VECT2),
        lex_chain([VECT1,VECT2],[op(#=<)]).

lex_lesseq_a(FLAG,VECTOR1,VECTOR2) :-
        collection(VECTOR1,[dvar]),
        collection(VECTOR2,[dvar]),
        length(VECTOR1,L),
        length(VECTOR2,L),
        lex_lesseq_signature(VECTOR1,VECTOR2,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _66771,
            SIGNATURE,
            [source(s),sink(s),sink(t)],
            [arc(s,2,s),
             arc(s,1,t),
             arc(t,1,t),
             arc(t,2,t),
             arc(t,3,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[1,2,3],AUTOMATON).

lex_lesseq_signature([],[],[]).

lex_lesseq_signature([[var-VAR1]|Xs],[[var-VAR2]|Ys],[S|Ss]) :-
        S in 1..3,
        VAR1#<VAR2#<=>S#=1,
        VAR1#=VAR2#<=>S#=2,
        VAR1#>VAR2#<=>S#=3,
        lex_lesseq_signature(Xs,Ys,Ss).
```

B.236 lex_lesseq_allperm

◇ META-DATA:

```
ctr_predefined(lex_lesseq_allperm).

ctr_date(lex_lesseq_allperm,['20070916']).

ctr_origin(
    lex_lesseq_allperm,
    Inspired by \cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02},
    []).

ctr_synonyms(lex_lesseq_allperm,[leximin]).

ctr_arguments(
    lex_lesseq_allperm,
    ['VECTOR1'-collection(var-dvar),
     'VECTOR2'-collection(var-dvar)]).

ctr_restrictions(
    lex_lesseq_allperm,
    [required('VECTOR1',var),
     required('VECTOR2',var),
     size('VECTOR1')=size('VECTOR2')]).

ctr_example(
    lex_lesseq_allperm,
    lex_lesseq_allperm(
        [[var-1],[var-2],[var-3]],
        [[var-3],[var-1],[var-2]])).

ctr_typical(lex_lesseq_allperm,[size('VECTOR1')>1]).

ctr_exchangeable(
    lex_lesseq_allperm,
    [vals(['VECTOR1'^var,'VECTOR2'^var],int,=\=,all,dontcare)]).

ctr_contractible(
    lex_lesseq_allperm,
    [],
    ['VECTOR1','VECTOR2'],
    suffix).
```

B.237 link_set_to_booleans

◇ **META-DATA:**

```
ctr_date(link_set_to_booleans,['20030820','20060811']).

ctr_origin(
    link_set_to_booleans,
    Inspired by %c.,
    [domain_constraint]).

ctr_arguments(
    link_set_to_booleans,
    ['SVAR'-svar,'BOOLEANS'-collection(bool-dvar,val-int)]).

ctr_restrictions(
    link_set_to_booleans,
    [required('BOOLEANS',[bool,val]),
     'BOOLEANS'^bool>=0,
     'BOOLEANS'^bool=<1,
     distinct('BOOLEANS',val)]).

ctr_example(
    link_set_to_booleans,
    link_set_to_booleans(
        {1,3,4},
        [[bool-0,val-0],
         [bool-1,val-1],
         [bool-0,val-2],
         [bool-1,val-3],
         [bool-1,val-4],
         [bool-0,val-5]])).

ctr_typical(
    link_set_to_booleans,
    [size('BOOLEANS')>1,range('BOOLEANS'^bool)>1]).

ctr_exchangeable(link_set_to_booleans,[items('BOOLEANS',all)]).

ctr_derived_collections(
    link_set_to_booleans,
    [col('SET'-collection(one-int,setvar-svar),
         [item(one-1,setvar-'SVAR')])]).

ctr_graph(
    link_set_to_booleans,
```

```
['SET','BOOLEANS'],
2,
['PRODUCT'>>collection(set,booleans)],
[booleans^bool=set^one#<=>booleans^val in_set set^setvar],
['NARC'=size('BOOLEANS')],
[]).
```

B.238 longest_change

◇ **META-DATA:**

```
ctr_date(
    longest_change,
    ['20000128','20030820','20040530','20060811']).

ctr_origin(longest_change,'Derived from %c.',[change]).

ctr_arguments(
    longest_change,
    ['SIZE'-dvar,'VARIABLES'-collection(var-dvar),'CTR'-atom]).

ctr_restrictions(
    longest_change,
    ['SIZE'>=0,
     'SIZE'=<size('VARIABLES'),
     required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    longest_change,
    longest_change(
        4,
        [[var-8],
         [var-8],
         [var-3],
         [var-4],
         [var-1],
         [var-1],
         [var-5],
         [var-5],
         [var-2]],
        =\=)).

ctr_typical(
    longest_change,
    [size('VARIABLES')>2,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=\=])]).

ctr_exchangeable(longest_change,[translate(['VARIABLES'^var])]).

ctr_graph(
    longest_change,
```

```
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    ['CTR'(variables1^var,variables2^var)],
    ['MAX_NCC'='SIZE'],
    []).

ctr_eval(
    longest_change,
    [checker(longest_change_c),automaton(longest_change_a)]).

ctr_pure_functional_dependency(longest_change,[]).

ctr_functional_dependency(longest_change,1,[2,3]).

longest_change_c(SIZE,VARIABLES,=) :-
        !,
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        get_attr1(VARIABLES,VARS),
        longest_change_eq_c(VARS,0,0,SIZE).

longest_change_c(SIZE,VARIABLES,=\=) :-
        !,
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        get_attr1(VARIABLES,VARS),
        longest_change_neq_c(VARS,0,0,SIZE).

longest_change_c(SIZE,VARIABLES,<) :-
        !,
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        get_attr1(VARIABLES,VARS),
        longest_change_lt_c(VARS,0,0,SIZE).

longest_change_c(SIZE,VARIABLES,>=) :-
        !,
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        get_attr1(VARIABLES,VARS),
        longest_change_geq_c(VARS,0,0,SIZE).
```

```
longest_change_c(SIZE,VARIABLES,>) :-
        !,
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        get_attr1(VARIABLES,VARS),
        longest_change_gt_c(VARS,0,0,SIZE).

longest_change_c(SIZE,VARIABLES,=<) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        get_attr1(VARIABLES,VARS),
        longest_change_leq_c(VARS,0,0,SIZE).

longest_change_eq_c([V,V|R],C,_D,SIZE) :-
        !,
        longest_change_eq_c1([V|R],C,2,SIZE).

longest_change_eq_c([_42112,V|R],C,D,SIZE) :-
        !,
        longest_change_eq_c([V|R],C,D,SIZE).

longest_change_eq_c(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_eq_c1([V,V|R],C,D,SIZE) :-
        !,
        D1 is D+1,
        longest_change_eq_c1([V|R],C,D1,SIZE).

longest_change_eq_c1([_42112,V|R],C,D,SIZE) :-
        !,
        C1 is max(C,D),
        longest_change_eq_c1([V|R],C1,1,SIZE).

longest_change_eq_c1(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_neq_c([V,V|R],C,D,SIZE) :-
        !,
        longest_change_neq_c([V|R],C,D,SIZE).
```

```
longest_change_neq_c([_42112,V|R],C,_D,SIZE) :-
        !,
        longest_change_neq_c1([V|R],C,2,SIZE).

longest_change_neq_c(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_neq_c1([V,V|R],C,D,SIZE) :-
        !,
        C1 is max(C,D),
        longest_change_neq_c1([V|R],C1,1,SIZE).

longest_change_neq_c1([_42112,V|R],C,D,SIZE) :-
        !,
        D1 is D+1,
        longest_change_neq_c1([V|R],C,D1,SIZE).

longest_change_neq_c1(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_lt_c([V1,V2|R],C,_D,SIZE) :-
        V1<V2,
        !,
        longest_change_lt_c1([V2|R],C,2,SIZE).

longest_change_lt_c([_42112,V|R],C,D,SIZE) :-
        !,
        longest_change_lt_c([V|R],C,D,SIZE).

longest_change_lt_c(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_lt_c1([V1,V2|R],C,D,SIZE) :-
        V1<V2,
        !,
        D1 is D+1,
        longest_change_lt_c1([V2|R],C,D1,SIZE).

longest_change_lt_c1([_42112,V|R],C,D,SIZE) :-
        !,
        C1 is max(C,D),
        longest_change_lt_c1([V|R],C1,1,SIZE).
```

```
longest_change_lt_c1(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_geq_c([V1,V2|R],C,_D,SIZE) :-
        V1>=V2,
        !,
        longest_change_geq_c1([V2|R],C,2,SIZE).

longest_change_geq_c([_42112,V|R],C,D,SIZE) :-
        !,
        longest_change_geq_c([V|R],C,D,SIZE).

longest_change_geq_c(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_geq_c1([V1,V2|R],C,D,SIZE) :-
        V1>=V2,
        !,
        D1 is D+1,
        longest_change_geq_c1([V2|R],C,D1,SIZE).

longest_change_geq_c1([_42112,V|R],C,D,SIZE) :-
        !,
        C1 is max(C,D),
        longest_change_geq_c1([V|R],C1,1,SIZE).

longest_change_geq_c1(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_gt_c([V1,V2|R],C,_D,SIZE) :-
        V1>V2,
        !,
        longest_change_gt_c1([V2|R],C,2,SIZE).

longest_change_gt_c([_42112,V|R],C,D,SIZE) :-
        !,
        longest_change_gt_c([V|R],C,D,SIZE).

longest_change_gt_c(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_gt_c1([V1,V2|R],C,D,SIZE) :-
```

```
        V1>V2,
        !,
        D1 is D+1,
        longest_change_gt_c1([V2|R],C,D1,SIZE).

longest_change_gt_c1([_42112,V|R],C,D,SIZE) :-
        !,
        C1 is max(C,D),
        longest_change_gt_c1([V|R],C1,1,SIZE).

longest_change_gt_c1(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_leq_c([V1,V2|R],C,_D,SIZE) :-
        V1=<V2,
        !,
        longest_change_leq_c1([V2|R],C,2,SIZE).

longest_change_leq_c([_42112,V|R],C,D,SIZE) :-
        !,
        longest_change_leq_c([V|R],C,D,SIZE).

longest_change_leq_c(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_leq_c1([V1,V2|R],C,D,SIZE) :-
        V1=<V2,
        !,
        D1 is D+1,
        longest_change_leq_c1([V2|R],C,D1,SIZE).

longest_change_leq_c1([_42112,V|R],C,D,SIZE) :-
        !,
        C1 is max(C,D),
        longest_change_leq_c1([V|R],C1,1,SIZE).

longest_change_leq_c1(_42108,C,D,SIZE) :-
        M is max(C,D),
        SIZE#=M.

longest_change_neq_counters_check([V,V|R],C,D,[C1|S]) :-
        !,
        C1 is max(C,D),
        longest_change_neq_counters_check([V|R],C,D,S).
```

```
longest_change_neq_counters_check([_42112,V|R],C,D,[C1|S]) :-
        !,
        C1 is max(C,D),
        longest_change_neq_counters_check1([V|R],C,2,S).

longest_change_neq_counters_check(_42105,_42106,_42107,[0]).

longest_change_neq_counters_check1([V,V|R],C,D,[C1|S]) :-
        !,
        C1 is max(C,D),
        longest_change_neq_counters_check1([V|R],C1,1,S).

longest_change_neq_counters_check1([_42112,V|R],C,D,[C1|S]) :-
        !,
        C1 is max(C,D),
        D1 is D+1,
        longest_change_neq_counters_check1([V|R],C,D1,S).

longest_change_neq_counters_check1(_42105,_42106,_42107,[0]).

longest_change_a(FLAG,SIZE,VARIABLES,CTR) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        longest_change_signature(VARIABLES,SIGNATURE,CTR),
        automaton(
            SIGNATURE,
            _43706,
            SIGNATURE,
            [source(s),sink(i),sink(s)],
            [arc(s,0,s),
             arc(s,1,i,[C,2]),
             arc(i,1,i,[C,D+1]),
             arc(i,0,i,[max(C,D),1])],
            [C,D],
            [0,0],
            [C1,D1]),
        SIZE#=max(C1,D1)#<=>FLAG.

longest_change_signature([],[],_42107).

longest_change_signature([_42111],[],_42110) :-
        !.
```

```
longest_change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],=) :-
        !,
        VAR1#=VAR2#<=>S,
        longest_change_signature([[var-VAR2]|VARs],Ss,=).

longest_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    =\=) :-
        !,
        VAR1#\=VAR2#<=>S,
        longest_change_signature([[var-VAR2]|VARs],Ss,=\=).

longest_change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],<) :-
        !,
        VAR1#<VAR2#<=>S,
        longest_change_signature([[var-VAR2]|VARs],Ss,<).

longest_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    >=) :-
        !,
        VAR1#>=VAR2#<=>S,
        longest_change_signature([[var-VAR2]|VARs],Ss,>=).

longest_change_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],>) :-
        !,
        VAR1#>VAR2#<=>S,
        longest_change_signature([[var-VAR2]|VARs],Ss,>).

longest_change_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss],
    =<) :-
        !,
        VAR1#=<VAR2#<=>S,
        longest_change_signature([[var-VAR2]|VARs],Ss,=<).
```

B.239 longest_decreasing_sequence

◇ **META-DATA:**

```
ctr_date(longest_decreasing_sequence,['20121124']).

ctr_origin(
    longest_decreasing_sequence,
    constraint on sequences,
    []).

ctr_synonyms(
    longest_decreasing_sequence,
    [size_longest_decreasing_sequence]).

ctr_arguments(
    longest_decreasing_sequence,
    ['L'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    longest_decreasing_sequence,
    ['L'>=0,
     'L'<range('VARIABLES'^var),
     required('VARIABLES',var)]).

ctr_example(
    longest_decreasing_sequence,
    [longest_decreasing_sequence(
         0,
         [[var-0],[var-1],[var-2],[var-5]]),
     longest_decreasing_sequence(0,[[var-8],[var-8]]),
     longest_decreasing_sequence(
         6,
         [[var-10],
          [var-8],
          [var-8],
          [var-6],
          [var-4],
          [var-9],
          [var-10],
          [var-8]])]).

ctr_typical(
    longest_decreasing_sequence,
    ['L'>0,size('VARIABLES')>1,nval('VARIABLES'^var)>2]).
```

```
ctr_typical_model(
    longest_decreasing_sequence,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    longest_decreasing_sequence,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    longest_decreasing_sequence,
    [checker(longest_decreasing_sequence_c),
     automaton(longest_decreasing_sequence_a)]).

ctr_pure_functional_dependency(longest_decreasing_sequence,[]).

ctr_functional_dependency(longest_decreasing_sequence,1,[2]).

ctr_sol(longest_decreasing_sequence,2,0,2,9,[0-6,1-2,2-1]).

ctr_sol(
    longest_decreasing_sequence,
    3,
    0,
    3,
    64,
    [0-20,1-18,2-16,3-10]).

ctr_sol(
    longest_decreasing_sequence,
    4,
    0,
    4,
    625,
    [0-70,1-122,2-161,3-162,4-110]).

ctr_sol(
    longest_decreasing_sequence,
    5,
    0,
    5,
    7776,
    [0-252,1-750,2-1398,3-1942,4-2024,5-1410]).

ctr_sol(
    longest_decreasing_sequence,
    6,
```

```
    0,
    6,
    117649,
    [0-924,1-4412,2-11361,3-20816,4-28930,5-30134,6-21072]).

ctr_sol(
    longest_decreasing_sequence,
    7,
    0,
    7,
    2097152,
    [0-3432,
     1-25382,
     2-89132,
     3-211106,
     4-375084,
     5-506766,
     6-522648,
     7-363602]).

ctr_sol(
    longest_decreasing_sequence,
    8,
    0,
    8,
    43046721,
    [0-12870,
     1-144314,
     2-685090,
     3-2074365,
     4-4603682,
     5-7792840,
     6-10197174,
     7-10379696,
     8-7156690]).

longest_decreasing_sequence_c(0,[]) :-
        !.

longest_decreasing_sequence_c(L,VARIABLES) :-
        check_type(dvar,L),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        longest_decreasing_sequence_c(VARS,s,0,0,0,L).

longest_decreasing_sequence_c([V|R],s,_48682,_48683,Max,L) :-
```

```
        !,
        longest_decreasing_sequence_c(R,t,V,V,Max,L).

longest_decreasing_sequence_c([V|R],t,_48682,Last,Max,L) :-
        Last<V,
        !,
        longest_decreasing_sequence_c(R,t,V,V,Max,L).

longest_decreasing_sequence_c([V|R],t,First,_48683,Max,L) :-
        !,
        Max1 is max(Max,First-V),
        longest_decreasing_sequence_c(R,t,First,V,Max1,L).

longest_decreasing_sequence_c([],_48678,_48679,_48680,L,L).

longest_decreasing_sequence_counters_check(
    [V|R],
    s,
    _49155,
    _49202,
    Max,
    [Max|S]) :-
        !,
        longest_decreasing_sequence_counters_check(
            R,
            t,
            V,
            V,
            Max,
            S).

longest_decreasing_sequence_counters_check(
    [V|R],
    t,
    _49167,
    Last,
    Max,
    [Max|S]) :-
        Last<V,
        !,
        longest_decreasing_sequence_counters_check(
            R,
            t,
            V,
            V,
            Max,
```

```
            S).

longest_decreasing_sequence_counters_check(
    [V|R],
    t,
    First,
    _49220,
    Max,
    [Max1|S]) :-
        !,
        Max1 is max(Max,First-V),
        longest_decreasing_sequence_counters_check(
            R,
            t,
            First,
            V,
            Max1,
            S).

longest_decreasing_sequence_counters_check(
    [],
    _48960,
    _49007,
    _49054,
    _49101,
    []).

longest_decreasing_sequence_a(FLAG,L,VARIABLES) :-
        check_type(dvar,L),
        length(VARIABLES,N),
        (   N=0 ->
            Max=0
        ;   collection(VARIABLES,[dvar]),
            longest_decreasing_sequence_signature(
                VARIABLES,
                SIGNATURE,
                DIFFERENCES),
            automaton(
                DIFFERENCES,
                Di,
                SIGNATURE,
                [source(s),sink(s),sink(t)],
                [arc(s,0,s),
                 arc(s,1,s),
                 arc(s,2,t,[max(M,Di),Di]),
                 arc(t,0,s),
```

```
                 arc(t,1,t),
                 arc(t,2,t,[max(M,C+Di),C+Di])],
                [M,C],
                [0,0],
                [Max,_48834])
        ),
        Max#=L#<=>FLAG.

longest_decreasing_sequence_signature([_48683],[],[]) :-
        !.

longest_decreasing_sequence_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|RS],
    [DIFFERENCE|RD]) :-
        VAR1#<VAR2#<=>S#=0,
        VAR1#=VAR2#<=>S#=1,
        VAR1#>VAR2#<=>S#=2,
        VAR1#=DIFFERENCE+VAR2,
        longest_decreasing_sequence_signature(
            [[var-VAR2]|VARs],
            RS,
            RD).
```

B.240 longest_increasing_sequence

◇ **META-DATA:**

```
ctr_date(longest_increasing_sequence,['20121124']).

ctr_origin(
    longest_increasing_sequence,
    constraint on sequences,
    []).

ctr_synonyms(
    longest_increasing_sequence,
    [size_longest_increasing_sequence]).

ctr_arguments(
    longest_increasing_sequence,
    ['L'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    longest_increasing_sequence,
    ['L'>=0,
     'L'<range('VARIABLES'^var),
     required('VARIABLES',var)]).

ctr_example(
    longest_increasing_sequence,
    [longest_increasing_sequence(
         7,
         [[var-10],
          [var-8],
          [var-8],
          [var-6],
          [var-4],
          [var-9],
          [var-11],
          [var-8]]),
     longest_increasing_sequence(
         0,
         [[var-10],
          [var-8],
          [var-7],
          [var-5],
          [var-4],
          [var-3],
          [var-1],
```

```
          [var-0]])]).

ctr_typical(
    longest_increasing_sequence,
    ['L'>0,size('VARIABLES')>1,nval('VARIABLES'^var)>2]).

ctr_typical_model(
    longest_increasing_sequence,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    longest_increasing_sequence,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    longest_increasing_sequence,
    [checker(longest_increasing_sequence_c),
     automaton(longest_increasing_sequence_a)]).

ctr_pure_functional_dependency(longest_increasing_sequence,[]).

ctr_functional_dependency(longest_increasing_sequence,1,[2]).

ctr_sol(longest_increasing_sequence,2,0,2,9,[0-6,1-2,2-1]).

ctr_sol(
    longest_increasing_sequence,
    3,
    0,
    3,
    64,
    [0-20,1-18,2-16,3-10]).

ctr_sol(
    longest_increasing_sequence,
    4,
    0,
    4,
    625,
    [0-70,1-122,2-161,3-162,4-110]).

ctr_sol(
    longest_increasing_sequence,
    5,
    0,
    5,
```

```
    7776,
    [0-252,1-750,2-1398,3-1942,4-2024,5-1410]).

ctr_sol(
    longest_increasing_sequence,
    6,
    0,
    6,
    117649,
    [0-924,1-4412,2-11361,3-20816,4-28930,5-30134,6-21072]).

ctr_sol(
    longest_increasing_sequence,
    7,
    0,
    7,
    2097152,
    [0-3432,
     1-25382,
     2-89132,
     3-211106,
     4-375084,
     5-506766,
     6-522648,
     7-363602]).

ctr_sol(
    longest_increasing_sequence,
    8,
    0,
    8,
    43046721,
    [0-12870,
     1-144314,
     2-685090,
     3-2074365,
     4-4603682,
     5-7792840,
     6-10197174,
     7-10379696,
     8-7156690]).

longest_increasing_sequence_c(0,[]) :-
        !.

longest_increasing_sequence_c(L,VARIABLES) :-
```

```
        check_type(dvar,L),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        longest_increasing_sequence_c(VARS,s,0,0,0,L).

longest_increasing_sequence_c([V|R],s,_48663,_48664,Max,L) :-
        !,
        longest_increasing_sequence_c(R,t,V,V,Max,L).

longest_increasing_sequence_c([V|R],t,_48663,Last,Max,L) :-
        Last>V,
        !,
        longest_increasing_sequence_c(R,t,V,V,Max,L).

longest_increasing_sequence_c([V|R],t,First,_48664,Max,L) :-
        !,
        Max1 is max(Max,V-First),
        longest_increasing_sequence_c(R,t,First,V,Max1,L).

longest_increasing_sequence_c([],_48659,_48660,_48661,L,L).

longest_increasing_sequence_counters_check(
    [V|R],
    s,
    _49136,
    _49183,
    Max,
    [Max|S]) :-
        !,
        longest_increasing_sequence_counters_check(
            R,
            t,
            V,
            V,
            Max,
            S).

longest_increasing_sequence_counters_check(
    [V|R],
    t,
    _49148,
    Last,
    Max,
    [Max|S]) :-
        Last>V,
        !,
```

```
        longest_increasing_sequence_counters_check(
            R,
            t,
            V,
            V,
            Max,
            S).

longest_increasing_sequence_counters_check(
    [V|R],
    t,
    First,
    _49201,
    Max,
    [Max1|S]) :-
        !,
        Max1 is max(Max,V-First),
        longest_increasing_sequence_counters_check(
            R,
            t,
            First,
            V,
            Max1,
            S).

longest_increasing_sequence_counters_check(
    [],
    _48941,
    _48988,
    _49035,
    _49082,
    []).

longest_increasing_sequence_a(FLAG,L,VARIABLES) :-
        check_type(dvar,L),
        length(VARIABLES,N),
        (   N=0 ->
            Max=0
        ;   collection(VARIABLES,[dvar]),
            longest_increasing_sequence_signature(
                VARIABLES,
                SIGNATURE,
                DIFFERENCES),
            automaton(
                DIFFERENCES,
                Di,
```

```
                SIGNATURE,
                [source(s),sink(s),sink(t)],
                [arc(s,2,s),
                 arc(s,1,s),
                 arc(s,0,t,[max(M,Di),Di]),
                 arc(t,2,s),
                 arc(t,1,t),
                 arc(t,0,t,[max(M,C+Di),C+Di])],
                [M,C],
                [0,0],
                [Max,_48815])
        ),
        Max#=L#<=>FLAG.

longest_increasing_sequence_signature([_48664],[],[]) :-
        !.

longest_increasing_sequence_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|RS],
    [DIFFERENCE|RD]) :-
        VAR1#>VAR2#<=>S#=2,
        VAR1#=VAR2#<=>S#=1,
        VAR1#<VAR2#<=>S#=0,
        VAR2#=DIFFERENCE+VAR1,
        longest_increasing_sequence_signature(
            [[var-VAR2]|VARs],
            RS,
            RD).
```

B.241 lt

◊ **META-DATA:**

```
ctr_predefined(lt).

ctr_date(lt,['20070821']).

ctr_origin(lt,'Arithmetic.',[]).

ctr_synonyms(lt,[rel,xlty]).

ctr_arguments(lt,['VAR1'-dvar,'VAR2'-dvar]).

ctr_example(lt,lt(1,8)).

ctr_exchangeable(
    lt,
    [vals(['VAR1'],int(<('VAR2')),=\=,all,dontcare),
     vals(['VAR2'],int(>('VAR1')),=\=,all,dontcare)]).

ctr_eval(lt,[builtin(lt_b)]).

lt_b(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#<VAR2.
```

B.242 map

◇ META-DATA:

```
ctr_date(map,['20000128','20030820','20060811']).

ctr_origin(map,'Inspired by \\cite{SedgewickFlajolet96}',[]).

ctr_arguments(
    map,
    ['NBCYCLE'-dvar,
     'NBTREE'-dvar,
     'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    map,
    ['NBCYCLE'>=0,
     'NBTREE'>=0,
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    map,
    map(2,
        3,
        [[index-1,succ-5],
         [index-2,succ-9],
         [index-3,succ-8],
         [index-4,succ-2],
         [index-5,succ-9],
         [index-6,succ-2],
         [index-7,succ-9],
         [index-8,succ-8],
         [index-9,succ-1]])).

ctr_typical(
    map,
    ['NBCYCLE'>0,
     'NBTREE'>0,
     'NBCYCLE'<size('NODES'),
     'NBCYCLE'<'NBTREE',
     size('NODES')>2]).
```

```
ctr_exchangeable(map,[items('NODES',all)]).

ctr_graph(
    map,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['NCC'='NBCYCLE','NTREE'='NBTREE'],
    []).

ctr_pure_functional_dependency(map,[]).

ctr_functional_dependency(map,1,[3]).

ctr_functional_dependency(map,2,[3]).

ctr_application(map,[3]).
```

B.243 max_decreasing_slope

◇ META-DATA:

```
ctr_date(max_decreasing_slope,['20130317']).

ctr_origin(max_decreasing_slope,'Motivated by time series.',[]).

ctr_arguments(
    max_decreasing_slope,
    ['MAX'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    max_decreasing_slope,
    ['MAX'>=0,
     'MAX'<range('VARIABLES'^var),
     required('VARIABLES',var),
     size('VARIABLES')>0]).

ctr_example(
    max_decreasing_slope,
    [max_decreasing_slope(
         4,
         [[var-1],
          [var-1],
          [var-5],
          [var-8],
          [var-6],
          [var-2],
          [var-4],
          [var-1],
          [var-2]]),
     max_decreasing_slope(
         0,
         [[var-1],[var-3],[var-5],[var-8]]),
     max_decreasing_slope(
         8,
         [[var-3],[var-1],[var-9],[var-1]])]).

ctr_typical(
    max_decreasing_slope,
    ['MAX'>0,
     'MAX'<range('VARIABLES'^var)-1,
     size('VARIABLES')>2,
     range('VARIABLES'^var)>2]).
```

```
ctr_typical_model(
    max_decreasing_slope,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    max_decreasing_slope,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    max_decreasing_slope,
    [checker(max_decreasing_slope_c),
     automaton(max_decreasing_slope_a),
     automaton_with_signature(max_decreasing_slope_a_s)]).

ctr_pure_functional_dependency(max_decreasing_slope,[]).

ctr_functional_dependency(max_decreasing_slope,1,[2]).

ctr_cond_imply(
    max_decreasing_slope,
    longest_decreasing_sequence,
    [range('VARIABLES'^var)='MAX'+1],
    [range('VARIABLES'^var)='L'+1],
    [none,'VARIABLES']).

ctr_cond_imply(
    max_decreasing_slope,
    min_decreasing_slope,
    ['MAX'=1],
    ['MIN'=1],
    id).

ctr_sol(max_decreasing_slope,2,0,2,9,[0-6,1-2,2-1]).

ctr_sol(max_decreasing_slope,3,0,3,64,[0-20,1-20,2-16,3-8]).

ctr_sol(
    max_decreasing_slope,
    4,
    0,
    4,
    625,
    [0-70,1-151,2-188,3-142,4-74]).

ctr_sol(
    max_decreasing_slope,
```

```
    5,
    0,
    5,
    7776,
    [0-252,1-1036,2-1952,3-2106,4-1584,5-846]).

ctr_sol(
    max_decreasing_slope,
    6,
    0,
    6,
    117649,
    [0-924,1-6828,2-19200,3-29035,4-28266,5-21684,6-11712]).

ctr_sol(
    max_decreasing_slope,
    7,
    0,
    7,
    2097152,
    [0-3432,
     1-44220,
     2-183304,
     3-380116,
     4-483840,
     5-457632,
     6-353088,
     7-191520]).

ctr_sol(
    max_decreasing_slope,
    8,
    0,
    8,
    43046721,
    [0-12870,
     1-284405,
     2-1721425,
     3-4847301,
     4-8021350,
     5-9208124,
     6-8654931,
     7-6673834,
     8-3622481]).

max_decreasing_slope_c(MAX,VARIABLES) :-
```

```
        check_type(dvar_gteq(0),MAX),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        N>0,
        max_decreasing_slope_c1(VARS,0,MAX).

max_decreasing_slope_c1([_49388],MAX,MAX) :-
        !.

max_decreasing_slope_c1([V1,V2|R],M,MAX) :-
        V1=<V2,
        !,
        max_decreasing_slope_c1([V2|R],M,MAX).

max_decreasing_slope_c1([V1,V2|R],M,MAX) :-
        N is max(M,V1-V2),
        max_decreasing_slope_c1([V2|R],N,MAX).

max_decreasing_slope_counters_check(L,[0|S]) :-
        max_decreasing_slope_counters_check(L,0,S).

max_decreasing_slope_counters_check([V1,V2|R],M,[M|S]) :-
        V1=<V2,
        !,
        max_decreasing_slope_counters_check([V2|R],M,S).

max_decreasing_slope_counters_check([V1,V2|R],M,[N|S]) :-
        N is max(M,V1-V2),
        max_decreasing_slope_counters_check([V2|R],N,S).

max_decreasing_slope_counters_check([_49388],_49386,[]) :-
        !.

ctr_automaton_signature(
    max_decreasing_slope,
    max_decreasing_slope_a,
    pair_signature(2,signature)).

max_decreasing_slope_a(FLAG,MAX,VARIABLES) :-
        check_type(dvar_gteq(0),MAX),
        collection(VARIABLES,[dvar]),
        max_decreasing_slope_signature(
            VARIABLES,
            SIGNATURE,
            DIFFERENCES),
```

```
        automaton(
            DIFFERENCES,
            Di,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[max(M,Di)])],
            [M],
            [0],
            [MAXIMUM]),
        MAXIMUM#=MAX#<=>FLAG.

max_decreasing_slope_signature([_49388],[],[]) :-
        !.

max_decreasing_slope_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|RS],
    [DIFFERENCE|RD]) :-
        VAR1#=<VAR2#<=>S#=0,
        VAR1#>VAR2#<=>S#=1,
        VAR1#=DIFFERENCE+VAR2,
        max_decreasing_slope_signature([[var-VAR2]|VARs],RS,RD).

max_decreasing_slope_a_s(FLAG,MAX,VARIABLES,SIGNATURE) :-
        check_type(dvar_gteq(0),MAX),
        collection(VARIABLES,[dvar]),
        difference_decreasing_slope_signature(
            VARIABLES,
            DIFFERENCES),
        automaton(
            DIFFERENCES,
            Di,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s),arc(s,2,s,[max(M,Di)])],
            [M],
            [0],
            [MAXIMUM]),
        MAXIMUM#=MAX#<=>FLAG.
```

B.244 max_increasing_slope

◇ **META-DATA:**

```
ctr_date(max_increasing_slope,['20130317']).

ctr_origin(max_increasing_slope,'Motivated by time series.',[]).

ctr_arguments(
    max_increasing_slope,
    ['MAX'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    max_increasing_slope,
    ['MAX'>=0,
     'MAX'<range('VARIABLES'^var),
     required('VARIABLES',var),
     size('VARIABLES')>0]).

ctr_example(
    max_increasing_slope,
    [max_increasing_slope(
         4,
         [[var-1],
          [var-1],
          [var-5],
          [var-8],
          [var-6],
          [var-2],
          [var-2],
          [var-1],
          [var-2]]),
     max_increasing_slope(
         0,
         [[var-9],[var-8],[var-6],[var-4],[var-1],[var-0]]),
     max_increasing_slope(
         8,
         [[var-9],[var-6],[var-6],[var-4],[var-1],[var-9]])]).

ctr_typical(
    max_increasing_slope,
    ['MAX'>0,
     'MAX'<range('VARIABLES'^var)-1,
     size('VARIABLES')>2,
     range('VARIABLES'^var)>2]).
```

```
ctr_typical_model(
    max_increasing_slope,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    max_increasing_slope,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    max_increasing_slope,
    [checker(max_increasing_slope_c),
     automaton(max_increasing_slope_a),
     automaton_with_signature(max_increasing_slope_a_s)]).

ctr_pure_functional_dependency(max_increasing_slope,[]).

ctr_functional_dependency(max_increasing_slope,1,[2]).

ctr_cond_imply(
    max_increasing_slope,
    longest_increasing_sequence,
    [range('VARIABLES'^var)='MAX'+1],
    [range('VARIABLES'^var)='L'+1],
    [none,'VARIABLES']).

ctr_cond_imply(
    max_increasing_slope,
    min_increasing_slope,
    ['MAX'=1],
    ['MIN'=1],
    id).

ctr_sol(max_increasing_slope,2,0,2,9,[0-6,1-2,2-1]).

ctr_sol(max_increasing_slope,3,0,3,64,[0-20,1-20,2-16,3-8]).

ctr_sol(
    max_increasing_slope,
    4,
    0,
    4,
    625,
    [0-70,1-151,2-188,3-142,4-74]).

ctr_sol(
    max_increasing_slope,
```

```
    5,
    0,
    5,
    7776,
    [0-252,1-1036,2-1952,3-2106,4-1584,5-846]).

ctr_sol(
    max_increasing_slope,
    6,
    0,
    6,
    117649,
    [0-924,1-6828,2-19200,3-29035,4-28266,5-21684,6-11712]).

ctr_sol(
    max_increasing_slope,
    7,
    0,
    7,
    2097152,
    [0-3432,
     1-44220,
     2-183304,
     3-380116,
     4-483840,
     5-457632,
     6-353088,
     7-191520]).

ctr_sol(
    max_increasing_slope,
    8,
    0,
    8,
    43046721,
    [0-12870,
     1-284405,
     2-1721425,
     3-4847301,
     4-8021350,
     5-9208124,
     6-8654931,
     7-6673834,
     8-3622481]).

max_increasing_slope_c(MAX,VARIABLES) :-
```

```
        check_type(dvar_gteq(0),MAX),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        N>0,
        max_increasing_slope_c1(VARS,0,MAX).

max_increasing_slope_c1([_49512],MAX,MAX) :-
        !.

max_increasing_slope_c1([V1,V2|R],M,MAX) :-
        V1>=V2,
        !,
        max_increasing_slope_c1([V2|R],M,MAX).

max_increasing_slope_c1([V1,V2|R],M,MAX) :-
        N is max(M,V2-V1),
        max_increasing_slope_c1([V2|R],N,MAX).

max_increasing_slope_counters_check(L,[0|S]) :-
        max_increasing_slope_counters_check(L,0,S).

max_increasing_slope_counters_check([V1,V2|R],M,[M|S]) :-
        V1>=V2,
        !,
        max_increasing_slope_counters_check([V2|R],M,S).

max_increasing_slope_counters_check([V1,V2|R],M,[N|S]) :-
        N is max(M,V2-V1),
        max_increasing_slope_counters_check([V2|R],N,S).

max_increasing_slope_counters_check([_49512],_49510,[]) :-
        !.

ctr_automaton_signature(
    max_increasing_slope,
    max_increasing_slope_a,
    pair_signature(2,signature)).

max_increasing_slope_a(FLAG,MAX,VARIABLES) :-
        check_type(dvar_gteq(0),MAX),
        collection(VARIABLES,[dvar]),
        max_increasing_slope_signature(
            VARIABLES,
            SIGNATURE,
            DIFFERENCES),
```

```
        automaton(
            DIFFERENCES,
            Di,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s,[max(M,Di)])],
            [M],
            [0],
            [MAXIMUM]),
        MAXIMUM#=MAX#<=>FLAG.

max_increasing_slope_signature([_49512],[],[]) :-
        !.

max_increasing_slope_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|RS],
    [DIFFERENCE|RD]) :-
        VAR1#>=VAR2#<=>S#=0,
        VAR1#<VAR2#<=>S#=1,
        VAR2#=DIFFERENCE+VAR1,
        max_increasing_slope_signature([[var-VAR2]|VARs],RS,RD).

max_increasing_slope_a_s(FLAG,MAX,VARIABLES,SIGNATURE) :-
        check_type(dvar_gteq(0),MAX),
        collection(VARIABLES,[dvar]),
        difference_increasing_slope_signature(
            VARIABLES,
            DIFFERENCES),
        automaton(
            DIFFERENCES,
            Di,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s),arc(s,2,s),arc(s,0,s,[max(M,Di)])],
            [M],
            [0],
            [MAXIMUM]),
        MAXIMUM#=MAX#<=>FLAG.
```

B.245 max_index

◇ META-DATA:

```
ctr_date(
    max_index,
    ['20030820','20040530','20041230','20060811']).

ctr_origin(max_index,'N.~Beldiceanu',[]).

ctr_arguments(
    max_index,
    ['MAX_INDEX'-dvar,
     'VARIABLES'-collection(index-int,var-dvar)]).

ctr_restrictions(
    max_index,
    [size('VARIABLES')>0,
     'MAX_INDEX'>=0,
     'MAX_INDEX'=<size('VARIABLES'),
     required('VARIABLES',[index,var]),
     'VARIABLES'^index>=1,
     'VARIABLES'^index=<size('VARIABLES'),
     distinct('VARIABLES',index)]).

ctr_example(
    max_index,
    max_index(
        3,
        [[index-1,var-3],
         [index-2,var-2],
         [index-3,var-7],
         [index-4,var-2],
         [index-5,var-7]])).

ctr_typical(
    max_index,
    [size('VARIABLES')>0,range('VARIABLES'^var)>1]).

ctr_exchangeable(
    max_index,
    [items('VARIABLES',all),translate(['VARIABLES'^var])]).

ctr_graph(
    max_index,
    ['VARIABLES'],
```

```
2,
['CLIQUE'>>collection(variables1,variables2)],
[variables1^key=variables2^key#\/
 variables1^var>variables2^var],
['ORDER'(0,0,index)='MAX_INDEX'],
[]).
```

B.246 max_n

◇ META-DATA:

```
ctr_date(max_n,['20000128','20030820','20041230','20060811']).

ctr_origin(max_n,'\\cite{Beldiceanu01}',[]).

ctr_arguments(
    max_n,
    ['MAX'-dvar,'RANK'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    max_n,
    ['RANK'>=0,
     'RANK'<size('VARIABLES'),
     size('VARIABLES')>0,
     required('VARIABLES',var)]).

ctr_example(
    max_n,
    max_n(6,1,[[var-3],[var-1],[var-7],[var-1],[var-6]])).

ctr_typical(
    max_n,
    ['RANK'>0,
     'RANK'<3,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1]).

ctr_typical_model(max_n,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    max_n,
    [items('VARIABLES',all),
     translate(['MAX','VARIABLES'^var])]).

ctr_graph(
    max_n,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^key=variables2^key#\/
     variables1^var>variables2^var],
    ['ORDER'('RANK','MININT',var)='MAX'],
    []).
```

```
ctr_eval(max_n,[checker(max_n_c),reformulation(max_n_r)]).

ctr_pure_functional_dependency(max_n,[]).

ctr_functional_dependency(max_n,1,[2,3]).

max_n_c(MAX,RANK,VARIABLES) :-
        length(VARIABLES,N),
        N>0,
        N1 is N-1,
        check_type(dvar,MAX),
        check_type(int(0,N1),RANK),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        sort(VARS,SVARS),
        length(SVARS,NN),
        Pos is NN-RANK,
        nth1(Pos,SVARS,MAX).

max_n_r(MAX,RANK,VARIABLES) :-
        length(VARIABLES,N),
        N>0,
        N1 is N-1,
        check_type(dvar,MAX),
        check_type(int(0,N1),RANK),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        create_collection([MAX],var,VMAX),
        create_collection(VARS,val,VALUES),
        eval(among_var(1,VMAX,VALUES)),
        NVAL in 0..N,
        eval(nvalue(NVAL,VARIABLES)),
        length(RANKS,N),
        domain(RANKS,0,N1),
        max_n1(VARS,RANKS,MAX,RANK,NVAL).

max_n1([],[],_43026,_43027,_43028).

max_n1([V|RV],[R|RR],MAX,RANK,NVAL) :-
        R#<NVAL,
        R#=RANK#<=>V#=MAX,
        max_n2(RV,RR,V,R),
        max_n1(RV,RR,MAX,RANK,NVAL).

max_n2([],[],_43026,_43027).
```

```
max_n2([Vj|RV],[Rj|RR],Vi,Ri) :-
        Vi#>Vj#<=>Ri#<Rj,
        Vi#=Vj#<=>Ri#=Rj,
        Vi#<Vj#<=>Ri#>Rj,
        max_n2(RV,RR,Vi,Ri).
```

B.247 max_nvalue

◇ **META-DATA:**

```
ctr_date(max_nvalue,['20000128','20030820','20060811']).

ctr_origin(max_nvalue,'Derived from %c.',[nvalue]).

ctr_arguments(
    max_nvalue,
    ['MAX'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    max_nvalue,
    ['MAX'>=1,
     'MAX'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    max_nvalue,
    [max_nvalue(
         3,
         [[var-9],
          [var-1],
          [var-7],
          [var-1],
          [var-1],
          [var-6],
          [var-7],
          [var-7],
          [var-4],
          [var-9]]),
     max_nvalue(
         1,
         [[var-9],[var-1],[var-7],[var-3],[var-2],[var-6]]),
     max_nvalue(
         6,
         [[var-5],[var-5],[var-5],[var-5],[var-5],[var-5]])]).

ctr_typical(
    max_nvalue,
    ['MAX'>1,
     'MAX'<size('VARIABLES'),
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1]).
```

```
ctr_exchangeable(
    max_nvalue,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    max_nvalue,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['MAX_NSCC'='MAX'],
    []).

ctr_eval(
    max_nvalue,
    [checker(max_nvalue_c),reformulation(max_nvalue_r)]).

ctr_pure_functional_dependency(max_nvalue,[]).

ctr_functional_dependency(max_nvalue,1,[2]).

ctr_sol(max_nvalue,2,0,2,9,[1-6,2-3]).

ctr_sol(max_nvalue,3,0,3,64,[1-24,2-36,3-4]).

ctr_sol(max_nvalue,4,0,4,625,[1-120,2-420,3-80,4-5]).

ctr_sol(max_nvalue,5,0,5,7776,[1-720,2-5400,3-1500,4-150,5-6]).

ctr_sol(
    max_nvalue,
    6,
    0,
    6,
    117649,
    [1-5040,2-78750,3-29820,4-3780,5-252,6-7]).

ctr_sol(
    max_nvalue,
    7,
    0,
    7,
    2097152,
    [1-40320,2-1305360,3-646800,4-96040,5-8232,6-392,7-8]).
```

```
ctr_sol(
    max_nvalue,
    8,
    0,
    8,
    43046721,
    [1-362880,
     2-24449040,
     3-15382080,
     4-2577960,
     5-258048,
     6-16128,
     7-576,
     8-9]).

max_nvalue_c(0,[]) :-
        !.

max_nvalue_c(MAX,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(1,N),MAX),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        samsort(VARS,SVARS),
        SVARS=[V|R],
        max_nvalue_seq_size(R,1,V,1,M),
        MAX#=M.

max_nvalue_r(0,[]) :-
        !.

max_nvalue_r(MAX,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(1,N),MAX),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        union_dom_list_int(VARS,UnionDomainsVARS),
        NSquare is N*N,
        length(UnionDomainsVARS,SizeUnion),
        (   SizeUnion=<NSquare ->
            balance1(UnionDomainsVARS,N,VALS,OCCS,_OCCS1),
            eval(global_cardinality(VARIABLES,VALS))
        ;   balance2(VARS,N,VARS,OCCS)
        ),
        eval(maximum(MAX,OCCS)).
```

B.248 max_occ_of_consecutive_tuples_of_values

◇ META-DATA:

```
ctr_predefined(max_occ_of_consecutive_tuples_of_values).

ctr_date(max_occ_of_consecutive_tuples_of_values,['20120319']).

ctr_origin(
    max_occ_of_consecutive_tuples_of_values,
    Design.,
    []).

ctr_types(
    max_occ_of_consecutive_tuples_of_values,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    max_occ_of_consecutive_tuples_of_values,
    ['MAX'-int,'K'-int,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    max_occ_of_consecutive_tuples_of_values,
    [required('VECTOR',var),
     size('VECTOR')>=2,
     alldifferent('VECTOR'),
     'MAX'>=1,
     'K'>=2,
     'K'<size('VECTOR'),
     required('VECTORS',vec),
     size('VECTORS')>=1,
     same_size('VECTORS',vec)]).

ctr_example(
    max_occ_of_consecutive_tuples_of_values,
    max_occ_of_consecutive_tuples_of_values(
        1,
        2,
        [[vec-[[var-4],[var-1],[var-3]]],
         [vec-[[var-2],[var-7],[var-6]]],
         [vec-[[var-5],[var-9],[var-8]]]])).

ctr_typical(
    max_occ_of_consecutive_tuples_of_values,
    ['MAX'=1,'K'=2,size('VECTORS')>2]).
```

```
ctr_eval(
    max_occ_of_consecutive_tuples_of_values,
    [reformulation(max_occ_of_consecutive_tuples_of_values_r),
     checker(max_occ_of_consecutive_tuples_of_values_c),
     density(max_occ_of_consecutive_tuples_of_values_d)]).

ctr_functional_dependency(
    max_occ_of_consecutive_tuples_of_values,
    1,
    [2,3]).

ctr_contractible(
    max_occ_of_consecutive_tuples_of_values,
    ['MAX'=1],
    VECTORS,
    any).

max_occ_of_consecutive_tuples_of_values_r(MAX,K,VECTORS) :-
        check_type(dvar_gteq(1),MAX),
        integer(K),
        K>=2,
        collection(VECTORS,[col([dvar])]),
        same_size(VECTORS),
        VECTORS=[[vec-VECTOR]|_26561],
        length(VECTOR,N),
        N>=2,
        K<N,
        get_attr11(VECTORS,VECTS),
        gen_alldifferents(VECTS,1),
        generate_consec_subtuples(VECTORS,K,SUBTUPLES),
        length(MIN0,K),
        length(MAX0,K),
        get_min_max_vectors(SUBTUPLES,0,K,MIN0,MAX0,MINS,MAXS),
        get_max_val_vec_vars(MINS,MAXS,1,MAX_VAL),
        MAX_VAL1 is MAX_VAL-1,
        create_vectors_vars(SUBTUPLES,MINS,MAXS,MAX_VAL1,VARS),
        length(SUBTUPLES,LEN_SUBTUPLES),
        MAX#=<LEN_SUBTUPLES,
        fd_max(MAX,MAX_MAX),
        create_occ_vars(0,MAX_VAL1,MAX_MAX,VALS_OCCS,OCCS),
        global_cardinality(VARS,VALS_OCCS),
        maximum(MAX,OCCS).

max_occ_of_consecutive_tuples_of_values_c(MAX,K,VECTORS) :-
        (   integer(MAX) ->
            MAX>=1
```

```
        ;   true
        ),
        integer(K),
        K>=2,
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        VECTORS=[[vec-VECTOR]|_26567],
        length(VECTOR,N),
        N>=2,
        K<N,
        get_attr11(VECTORS,VECTS),
        gen_alldifferents(VECTS,0),
        generate_consec_subtuples(VECTORS,K,SUBTUPLES),
        create_pairs(SUBTUPLES,PSUBTUPLES),
        keysort(PSUBTUPLES,SORTED),
        (   integer(MAX) ->
            Limit is MAX
        ;   length(SORTED,Limit)
        ),
        get_max_occ_tuples_of_values(SORTED,Limit,0,M),
        MAX=M,
        MAX>=1.

generate_consec_subtuples([],_26510,[]) :-
        !.

generate_consec_subtuples([[_26517-Tuple]|R],K,Result) :-
        remove_key_from_col(Tuple,Vars),
        length(Vars,N),
        gen_consec_sub_tuples(Vars,N,K,SubTuples1),
        generate_consec_subtuples(R,K,SubTuples2),
        append(SubTuples1,SubTuples2,Result).

gen_consec_sub_tuples(_26509,Len,K,[]) :-
        Len<K,
        !.

gen_consec_sub_tuples(Vars,Len,K,[Vars,RVars]) :-
        Len=K,
        !,
        reverse(Vars,RVars).

gen_consec_sub_tuples(Vars,Len,K,[Prefix,RPrefix|S]) :-
        Len>K,
        get_prefix_of_given_length(K,Vars,Prefix),
        reverse(Prefix,RPrefix),
```

```
        Len1 is Len-1,
        Vars=[_26565|RVars],
        gen_consec_sub_tuples(RVars,Len1,K,S).

get_prefix_of_given_length(0,_26510,[]) :-
        !.

get_prefix_of_given_length(N,[V|R],[V|S]) :-
        N>0,
        N1 is N-1,
        get_prefix_of_given_length(N1,R,S).

max_occ_of_consecutive_tuples_of_values_d(0,_26510,_26511,[]) :-
        !.

max_occ_of_consecutive_tuples_of_values_d(
    Density,
    MAX,
    K,
    VECTORS) :-
        VECTORS=[[vec-VECTOR]|_26528],
        length(VECTORS,N),
        length(VECTOR,D),
        Needed is 2*(D-K+1)*N,
        get_min_vectors(VECTORS,_26577,MINI),
        get_max_vectors(VECTORS,_26586,MAXI),
        RANGE is MAXI-MINI+1,
        Available is RANGE*RANGE*MAX,
        Density is Needed/Available.

get_min_vectors([],Min,Min) :-
        !.

get_min_vectors([[_26517-V]|R],Cur,Res) :-
        get_min_vector(V,Cur,New),
        get_min_vectors(R,New,Res).

get_min_vector([],Min,Min) :-
        !.

get_min_vector([[_26517-V]|R],Cur,Res) :-
        (   var(Cur) ->
            New is V
        ;   New is min(Cur,V)
        ),
        get_min_vector(R,New,Res).
```

```
get_max_vectors([],Max,Max) :-
        !.

get_max_vectors([[_26517-V]|R],Cur,Res) :-
        get_max_vector(V,Cur,New),
        get_max_vectors(R,New,Res).

get_max_vector([],Max,Max) :-
        !.

get_max_vector([[_26517-V]|R],Cur,Res) :-
        (   var(Cur) ->
            New is V
        ;   New is max(Cur,V)
        ),
        get_max_vector(R,New,Res).

gen_alldifferents([],_26510) :-
        !.

gen_alldifferents([VARS|R],FLAG) :-
        (   FLAG=1 ->
            all_different(VARS)
        ;   sort(VARS,SVARS),
            length(VARS,N),
            length(SVARS,N)
        ),
        gen_alldifferents(R,FLAG).
```

B.249 max_occ_of_sorted_tuples_of_values

◊ **META-DATA:**

```
ctr_predefined(max_occ_of_sorted_tuples_of_values).

ctr_date(max_occ_of_sorted_tuples_of_values,['20120327']).

ctr_origin(max_occ_of_sorted_tuples_of_values,'Design.',[]).

ctr_types(
    max_occ_of_sorted_tuples_of_values,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    max_occ_of_sorted_tuples_of_values,
    ['MAX'-int,'K'-int,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    max_occ_of_sorted_tuples_of_values,
    [required('VECTOR',var),
     size('VECTOR')>=2,
     alldifferent('VECTOR'),
     'MAX'>=1,
     'K'>=2,
     'K'<size('VECTOR'),
     required('VECTORS',vec),
     size('VECTORS')>=1,
     same_size('VECTORS',vec)]).

ctr_example(
    max_occ_of_sorted_tuples_of_values,
    max_occ_of_sorted_tuples_of_values(
        1,
        2,
        [[vec-[[var-4],[var-2],[var-1]]],
         [vec-[[var-2],[var-3],[var-5]]],
         [vec-[[var-3],[var-6],[var-4]]],
         [vec-[[var-5],[var-4],[var-7]]],
         [vec-[[var-6],[var-5],[var-1]]],
         [vec-[[var-7],[var-6],[var-2]]],
         [vec-[[var-3],[var-1],[var-7]]]])).

ctr_typical(
    max_occ_of_sorted_tuples_of_values,
    ['MAX'=1,'K'+1=size('VECTOR'),size('VECTORS')>2]).
```

```
ctr_eval(
    max_occ_of_sorted_tuples_of_values,
    [checker(max_occ_of_sorted_tuples_of_values_c)]).

ctr_functional_dependency(
    max_occ_of_sorted_tuples_of_values,
    1,
    [2,3]).

ctr_contractible(
    max_occ_of_sorted_tuples_of_values,
    ['MAX'=1],
    VECTORS,
    any).

max_occ_of_sorted_tuples_of_values_c(MAX,K,VECTORS) :-
        (   integer(MAX) ->
            MAX>=1
        ;   true
        ),
        integer(K),
        K>=2,
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        VECTORS=[[vec-VECTOR]|_27468],
        length(VECTOR,N),
        N>=2,
        K<N,
        generate_subtuples(VECTORS,K,1,SUBTUPLES),
        create_pairs(SUBTUPLES,PSUBTUPLES),
        keysort(PSUBTUPLES,SORTED),
        (   integer(MAX) ->
            Limit is MAX
        ;   length(SORTED,Limit)
        ),
        get_max_occ_tuples_of_values(SORTED,Limit,0,M),
        MAX=M,
        MAX>=1.
```

B.250 max_occ_of_tuples_of_values

◇ **META-DATA:**

```
ctr_predefined(max_occ_of_tuples_of_values).

ctr_date(max_occ_of_tuples_of_values,['20120228']).

ctr_origin(max_occ_of_tuples_of_values,'Design.',[]).

ctr_types(
    max_occ_of_tuples_of_values,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    max_occ_of_tuples_of_values,
    ['MAX'-int,'K'-int,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    max_occ_of_tuples_of_values,
    [required('VECTOR',var),
     size('VECTOR')>=2,
     strictly_increasing('VECTOR'),
     'MAX'>=1,
     'K'>=2,
     'K'<size('VECTOR'),
     required('VECTORS',vec),
     size('VECTORS')>=1,
     same_size('VECTORS',vec)]).

ctr_example(
    max_occ_of_tuples_of_values,
    max_occ_of_tuples_of_values(
        1,
        2,
        [[vec-[[var-1],[var-2],[var-4]]],
         [vec-[[var-2],[var-3],[var-5]]],
         [vec-[[var-3],[var-4],[var-6]]],
         [vec-[[var-4],[var-5],[var-7]]],
         [vec-[[var-1],[var-5],[var-6]]],
         [vec-[[var-2],[var-6],[var-7]]],
         [vec-[[var-1],[var-3],[var-7]]]])).

ctr_typical(
    max_occ_of_tuples_of_values,
    ['MAX'=<2,
```

```
     size('VECTOR')<'K'+5,
     'K'=2#\/'K'+1=size('VECTOR'),
     size('VECTORS')>2]).

ctr_eval(
    max_occ_of_tuples_of_values,
    [reformulation(max_occ_of_tuples_of_values_r),
     checker(max_occ_of_tuples_of_values_c)]).

ctr_functional_dependency(max_occ_of_tuples_of_values,1,[2,3]).

ctr_contractible(
    max_occ_of_tuples_of_values,
    ['MAX'=1],
    VECTORS,
    any).

max_occ_of_tuples_of_values_r(MAX,K,VECTORS) :-
        check_type(dvar_gteq(1),MAX),
        integer(K),
        K>=2,
        collection(VECTORS,[col([dvar])]),
        same_size(VECTORS),
        VECTORS=[[vec-VECTOR]|_27962],
        length(VECTOR,N),
        N>=2,
        K<N,
        max_occ_of_tuples_of_values_strictly_increasing_vectors(
            VECTORS,
            1),
        generate_subtuples(VECTORS,K,0,SUBTUPLES),
        length(MIN0,K),
        length(MAX0,K),
        get_min_max_vectors(SUBTUPLES,0,K,MIN0,MAX0,MINS,MAXS),
        get_max_val_vec_vars(MINS,MAXS,1,MAX_VAL),
        MAX_VAL1 is MAX_VAL-1,
        create_vectors_vars(SUBTUPLES,MINS,MAXS,MAX_VAL1,VARS),
        length(SUBTUPLES,LEN_SUBTUPLES),
        MAX#=<LEN_SUBTUPLES,
        fd_max(MAX,MAX_MAX),
        create_occ_vars(0,MAX_VAL1,MAX_MAX,VALS_OCCS,OCCS),
        global_cardinality(VARS,VALS_OCCS),
        maximum(MAX,OCCS).

max_occ_of_tuples_of_values_c(MAX,K,VECTORS) :-
        (   integer(MAX) ->
```

```
            MAX>=1
        ;   true
        ),
        integer(K),
        K>=2,
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        VECTORS=[[vec-VECTOR]|_27968],
        length(VECTOR,N),
        N>=2,
        K<N,
        max_occ_of_tuples_of_values_strictly_increasing_vectors(
            VECTORS,
            0),
        generate_subtuples(VECTORS,K,0,SUBTUPLES),
        create_pairs(SUBTUPLES,PSUBTUPLES),
        keysort(PSUBTUPLES,SORTED),
        (   integer(MAX) ->
            Limit is MAX
        ;   length(SORTED,Limit)
        ),
        get_max_occ_tuples_of_values(SORTED,Limit,0,M),
        MAX=M,
        MAX>=1.

max_occ_of_tuples_of_values_strictly_increasing_vectors(
    [],
    _28181) :-
        !.

max_occ_of_tuples_of_values_strictly_increasing_vectors(
    [[vec-VECTOR]|R],
    FLAG) :-
        max_occ_of_tuples_of_values_strictly_increasing_vector(
            VECTOR,
            FLAG),
        max_occ_of_tuples_of_values_strictly_increasing_vectors(
            R,
            FLAG).

max_occ_of_tuples_of_values_strictly_increasing_vector(
    [],
    _28181) :-
        !.

max_occ_of_tuples_of_values_strictly_increasing_vector(
```

```
    [_27912],
    _28276) :-
        !.

max_occ_of_tuples_of_values_strictly_increasing_vector(
    [[var-V1],[var-V2]|R],
    FLAG) :-
        (   FLAG=1 ->
            V1#<V2
        ;   V1<V2
        ),
        max_occ_of_tuples_of_values_strictly_increasing_vector(
            [[var-V2]|R],
            FLAG).
```

B.251 max_size_set_of_consecutive_var

◊ **META-DATA:**

```
ctr_date(
    max_size_set_of_consecutive_var,
    ['20030820','20040530','20060811']).

ctr_origin(max_size_set_of_consecutive_var,'N.~Beldiceanu',[]).

ctr_arguments(
    max_size_set_of_consecutive_var,
    ['MAX'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    max_size_set_of_consecutive_var,
    ['MAX'>=1,
     'MAX'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    max_size_set_of_consecutive_var,
    [max_size_set_of_consecutive_var(
         6,
         [[var-3],
          [var-1],
          [var-3],
          [var-7],
          [var-4],
          [var-1],
          [var-2],
          [var-8],
          [var-7],
          [var-6]]),
     max_size_set_of_consecutive_var(
         2,
         [[var-2],[var-6],[var-7],[var-3],[var-0],[var-9]])]).

ctr_typical(
    max_size_set_of_consecutive_var,
    ['MAX'<size('VARIABLES'),
     size('VARIABLES')>0,
     range('VARIABLES'^var)>1]).

ctr_exchangeable(
    max_size_set_of_consecutive_var,
```

```
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     translate(['VARIABLES'^var])]).

ctr_graph(
    max_size_set_of_consecutive_var,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [abs(variables1^var-variables2^var)=<1],
    ['MAX_NSCC'='MAX'],
    []).

ctr_eval(
    max_size_set_of_consecutive_var,
    [checker(max_size_set_of_consecutive_var_c)]).

ctr_pure_functional_dependency(
    max_size_set_of_consecutive_var,
    []).

ctr_functional_dependency(
    max_size_set_of_consecutive_var,
    1,
    [2]).

ctr_sol(max_size_set_of_consecutive_var,2,0,2,9,[1-2,2-7]).

ctr_sol(max_size_set_of_consecutive_var,3,0,3,64,[2-30,3-34]).

ctr_sol(
    max_size_set_of_consecutive_var,
    4,
    0,
    4,
    625,
    [2-168,3-240,4-217]).

ctr_sol(
    max_size_set_of_consecutive_var,
    5,
    0,
    5,
    7776,
    [2-720,3-3080,4-2260,5-1716]).
```

```
ctr_sol(
    max_size_set_of_consecutive_var,
    6,
    0,
    6,
    117649,
    [2-5220,3-35580,4-36030,5-24660,6-16159]).

ctr_sol(
    max_size_set_of_consecutive_var,
    7,
    0,
    7,
    2097152,
    [2-27720,3-426720,4-683550,5-477162,6-305634,7-176366]).

ctr_sol(
    max_size_set_of_consecutive_var,
    8,
    0,
    8,
    43046721,
    [2-249480,
     3-6059760,
     4-12672940,
     5-10592848,
     6-7044632,
     7-4239424,
     8-2187637]).

max_size_set_of_consecutive_var_c(MAX,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(1,N),MAX),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        samsort(VARS,SVARS),
        SVARS=[V|R],
        max_size_set_of_consecutive_var_c(R,V,1,0,M),
        MAX#=M.

max_size_set_of_consecutive_var_c([V|R],Prev,Occ,MaxOcc,Res) :-
        Diff is V-Prev,
        Diff=<1,
        !,
        Occ1 is Occ+1,
        max_size_set_of_consecutive_var_c(R,V,Occ1,MaxOcc,Res).
```

```
max_size_set_of_consecutive_var_c([V|R],_76005,Occ,MaxOcc,Res) :-
        !,
        Max is max(Occ,MaxOcc),
        max_size_set_of_consecutive_var_c(R,V,1,Max,Res).

max_size_set_of_consecutive_var_c([],_76005,Occ,MaxOcc,Res) :-
        Res is max(Occ,MaxOcc).
```

B.252 maximum

◇ **META-DATA:**

```
ctr_date(
    maximum,
    [20000128,
     20030820,
     20040530,
     20041230,
     20060811,
     20090416]).

ctr_origin(maximum,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(maximum,[max]).

ctr_arguments(
    maximum,
    ['MAX'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    maximum,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    maximum,
    [maximum(7,[[var-3],[var-2],[var-7],[var-2],[var-6]]),
     maximum(1,[[var-0],[var-0],[var-1],[var-0],[var-1]])]).

ctr_typical(
    maximum,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_typical_model(maximum,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    maximum,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     translate(['MAX','VARIABLES'^var])]).

ctr_graph(
    maximum,
    ['VARIABLES'],
    2,
```

```
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^key=variables2^key#\/
     variables1^var>variables2^var],
    ['ORDER'(0,'MININT',var)='MAX'],
    []).

ctr_eval(
    maximum,
    [builtin(maximum_b),
     automaton(maximum_a),
     automaton(maximum_ca)]).

ctr_pure_functional_dependency(maximum,[]).

ctr_functional_dependency(maximum,1,[2]).

ctr_aggregate(maximum,[],[max,union]).

ctr_cond_imply(
    maximum,
    highest_peak,
    [first('VARIABLES'^var)<'MAX',last('VARIABLES'^var)<'MAX'],
    [],
    id).

ctr_sol(maximum,2,0,2,9,[0-1,1-3,2-5]).

ctr_sol(maximum,3,0,3,64,[0-1,1-7,2-19,3-37]).

ctr_sol(maximum,4,0,4,625,[0-1,1-15,2-65,3-175,4-369]).

ctr_sol(
    maximum,
    5,
    0,
    5,
    7776,
    [0-1,1-31,2-211,3-781,4-2101,5-4651]).

ctr_sol(
    maximum,
    6,
    0,
    6,
    117649,
    [0-1,1-63,2-665,3-3367,4-11529,5-31031,6-70993]).
```

```
ctr_sol(
    maximum,
    7,
    0,
    7,
    2097152,
    [0-1,
     1-127,
     2-2059,
     3-14197,
     4-61741,
     5-201811,
     6-543607,
     7-1273609]).

ctr_sol(
    maximum,
    8,
    0,
    8,
    43046721,
    [0-1,
     1-255,
     2-6305,
     3-58975,
     4-325089,
     5-1288991,
     6-4085185,
     7-11012415,
     8-26269505]).

maximum_b(MAX,VARIABLES) :-
        check_type(dvar,MAX),
        collection(VARIABLES,[dvar]),
        VARIABLES=[_74914|_74915],
        get_attr1(VARIABLES,VARS),
        maximum(MAX,VARS).

maximum_a(FLAG,MAX,VARIABLES) :-
        check_type(dvar,MAX),
        collection(VARIABLES,[dvar]),
        VARIABLES=[_74917|_74918],
        maximum_signature(VARIABLES,SIGNATURE,MAX),
        AUTOMATON=
        automaton(
```

```
            SIGNATURE,
            _76468,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,1,t),arc(t,0,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2],AUTOMATON).

maximum_signature([],[],_74887).

maximum_signature([[var-VAR]|VARs],[S|Ss],MAX) :-
        S in 0..2,
        MAX#>VAR#<=>S#=0,
        MAX#=VAR#<=>S#=1,
        MAX#<VAR#<=>S#=2,
        maximum_signature(VARs,Ss,MAX).

maximum_ca(FLAG,MAX,VARIABLES) :-
        check_type(dvar,MAX),
        collection(VARIABLES,[dvar]),
        maximum_signature1(VARIABLES,VARS,Zeros),
        VARS=[VAR1|_74929],
        automaton(
            VARS,
            VARi,
            Zeros,
            [source(s),sink(s)],
            [arc(s,0,s,[max(C,VARi)])],
            [C],
            [VAR1],
            [CC]),
        CC#=MAX#<=>FLAG.

maximum_signature1([],[],[]).

maximum_signature1([[var-VAR]|VARs],[VAR|R],[0|S]) :-
        maximum_signature1(VARs,R,S).
```

B.253 maximum_modulo

◇ **META-DATA:**

```
ctr_date(
    maximum_modulo,
    ['20000128','20030820','20041230','20060811']).

ctr_origin(maximum_modulo,'Derived from %c.',[maximum]).

ctr_arguments(
    maximum_modulo,
    ['MAX'-dvar,'VARIABLES'-collection(var-dvar),'M'-int]).

ctr_restrictions(
    maximum_modulo,
    [size('VARIABLES')>0,'M'>0,required('VARIABLES',var)]).

ctr_example(
    maximum_modulo,
    maximum_modulo(
        5,
        [[var-9],[var-1],[var-7],[var-6],[var-5]],
        3)).

ctr_typical(
    maximum_modulo,
    ['M'>1,
     'M'<maxval('VARIABLES'^var),
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1]).

ctr_exchangeable(maximum_modulo,[items('VARIABLES',all)]).

ctr_graph(
    maximum_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^key=variables2^key#\/
     variables1^var mod 'M'>variables2^var mod 'M'],
    ['ORDER'(0,'MININT',var)='MAX'],
    []).

ctr_pure_functional_dependency(maximum_modulo,[]).
```

```
ctr_functional_dependency(maximum_modulo,1,[2,3]).
```

B.254 meet_sboxes

◊ **META-DATA:**

```
ctr_date(meet_sboxes,['20070622','20090725']).

ctr_origin(
    meet_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(meet_sboxes,[meet]).

ctr_types(
    meet_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    meet_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    meet_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    meet_sboxes,
    meet_sboxes(
        2,
        {0,1},
        [[oid-1,sid-1,x-[[v-3],[v-2]]],
         [oid-2,sid-2,x-[[v-4],[v-1]]],
         [oid-3,sid-4,x-[[v-3],[v-4]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-1],[v-2]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-1],[v-1]]],
         [sid-2,t-[[v-1],[v-0]],l-[[v-1],[v-3]]],
         [sid-2,t-[[v-0],[v-2]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
         [sid-3,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-4,t-[[v-0],[v-0]],l-[[v-1],[v-1]]]])).

ctr_typical(meet_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    meet_sboxes,
    [items('OBJECTS',all),
     items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all)]).

ctr_eval(meet_sboxes,[logic(meet_sboxes_g)]).

ctr_logic(
    meet_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (non_overlap_sboxes(Dims,O1,S1,O2,S2)--->
      exists(
          D,
          Dims,
          end(O1,S1,D)#=<origin(O2,S2,D)#\/
          end(O2,S2,D)#=<origin(O1,S1,D))),
     (meet_sboxes(Dims,O1,S1,O2,S2)--->
```

```
     exists(
         D,
         Dims,
         end(O1,S1,D)#=origin(O2,S2,D)#\/
         end(O2,S2,D)#=origin(O1,S1,D))),
    (meet_objects(Dims,O1,O2)--->
     forall(
         S1,
         sboxes([O1^sid]),
         forall(
             S2,
             sboxes([O2^sid]),
             non_overlap_sboxes(Dims,O1,S1,O2,S2)))#/\
     exists(
         S1,
         sboxes([O1^sid]),
         exists(
             S2,
             sboxes([O2^sid]),
             meet_sboxes(Dims,O1,S1,O2,S2)))),
    (all_meet(Dims,OIDS)--->
     forall(
         O1,
         objects(OIDS),
         forall(
             O2,
             objects(OIDS),
             O1^oid#<O2^oid#=>meet_objects(Dims,O1,O2)))),
    all_meet(DIMENSIONS,OIDS)]).

ctr_contractible(meet_sboxes,[],'OBJECTS',suffix).

ctr_application(meet_sboxes,[3]).

meet_sboxes_g(K,_41795,[],_41797) :-
        !,
        check_type(int_gteq(1),K).

meet_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
```

```
        SBOXES,
        [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]]),
    get_attr1(OBJECTS,OIDS),
    get_attr2(OBJECTS,SIDS),
    get_col_attr3(OBJECTS,1,XS),
    get_attr1(SBOXES,SIDES),
    get_col_attr2(SBOXES,1,TS),
    get_col_attr3(SBOXES,1,TL),
    collection_increasing_seq(OBJECTS,[1]),
    geost1(OIDS,SIDS,XS,Objects),
    geost2(SIDES,TS,TL,Sboxes),
    geost_dims(1,K,DIMENSIONS),
    ctr_logic(meet_sboxes,[DIMENSIONS,OIDS],Rules),
    geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.255 min_decreasing_slope

◇ **META-DATA:**

```
ctr_date(min_decreasing_slope,['20130317']).

ctr_origin(min_decreasing_slope,'Motivated by time series.',[]).

ctr_arguments(
    min_decreasing_slope,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_decreasing_slope,
    ['MIN'>=0,
     'MIN'<range('VARIABLES'^var),
     required('VARIABLES',var),
     size('VARIABLES')>0]).

ctr_example(
    min_decreasing_slope,
    [min_decreasing_slope(
         2,
         [[var-1],
          [var-1],
          [var-5],
          [var-8],
          [var-6],
          [var-2],
          [var-4],
          [var-1],
          [var-5]]),
     min_decreasing_slope(
         0,
         [[var-1],
          [var-1],
          [var-1],
          [var-3],
          [var-4],
          [var-7],
          [var-7],
          [var-7],
          [var-9]]),
     min_decreasing_slope(
         9,
         [[var-1],
```

```
          [var-1],
          [var-9],
          [var-0],
          [var-4],
          [var-7],
          [var-7],
          [var-7],
          [var-9]])]).

ctr_typical(
    min_decreasing_slope,
    ['MIN'>1,size('VARIABLES')>2,range('VARIABLES'^var)>2]).

ctr_typical_model(
    min_decreasing_slope,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    min_decreasing_slope,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    min_decreasing_slope,
    [checker(min_decreasing_slope_c),
     automaton(min_decreasing_slope_a),
     automaton_with_signature(min_decreasing_slope_a_s)]).

ctr_pure_functional_dependency(min_decreasing_slope,[]).

ctr_functional_dependency(min_decreasing_slope,1,[2]).

ctr_cond_imply(
    min_decreasing_slope,
    max_decreasing_slope,
    [range('VARIABLES'^var)='MIN'+1],
    [range('VARIABLES'^var)='MAX'+1],
    [none,'VARIABLES']).

ctr_sol(min_decreasing_slope,2,0,2,9,[0-6,1-2,2-1]).

ctr_sol(min_decreasing_slope,3,0,3,64,[0-20,1-22,2-14,3-8]).

ctr_sol(
    min_decreasing_slope,
    4,
    0,
```

```
    4,
    625,
    [0-70,1-256,2-145,3-98,4-56]).

ctr_sol(
    min_decreasing_slope,
    5,
    0,
    5,
    7776,
    [0-252,1-3512,2-1864,3-1062,4-704,5-382]).

ctr_sol(
    min_decreasing_slope,
    6,
    0,
    6,
    117649,
    [0-924,1-56537,2-28728,3-14729,4-8853,5-5266,6-2612]).

ctr_sol(
    min_decreasing_slope,
    7,
    0,
    7,
    2097152,
    [0-3432,
     1-1051936,
     2-515372,
     3-255076,
     4-133672,
     5-78198,
     6-41330,
     7-18136]).

ctr_sol(
    min_decreasing_slope,
    8,
    0,
    8,
    43046721,
    [0-12870,
     1-22280084,
     2-10601773,
     3-5106480,
     4-2475484,
```

```
     5-1369232,
     6-730161,
     7-341618,
     8-129019]).

min_decreasing_slope_c(MIN,VARIABLES) :-
        check_type(dvar_gteq(0),MIN),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        N>0,
        min_decreasing_slope_c1(VARS,0,MIN).

min_decreasing_slope_c1([_48784],MIN,MIN) :-
        !.

min_decreasing_slope_c1(_48781,1,MIN) :-
        !,
        MIN=1.

min_decreasing_slope_c1([V1,V2|R],M,MIN) :-
        V1=<V2,
        !,
        min_decreasing_slope_c1([V2|R],M,MIN).

min_decreasing_slope_c1([V1,V2|R],M,MIN) :-
        (   M=0 ->
            N is V1-V2
        ;   N is min(M,V1-V2)
        ),
        min_decreasing_slope_c1([V2|R],N,MIN).

min_decreasing_slope_counters_check(L,[0|S]) :-
        min_decreasing_slope_counters_check(L,0,S).

min_decreasing_slope_counters_check([V1,V2|R],M,[M|S]) :-
        V1=<V2,
        !,
        min_decreasing_slope_counters_check([V2|R],M,S).

min_decreasing_slope_counters_check([V1,V2|R],M,[N|S]) :-
        !,
        (   M=0 ->
            N is V1-V2
        ;   N is min(M,V1-V2)
        ),
```

```
        min_decreasing_slope_counters_check([V2|R],N,S).

min_decreasing_slope_counters_check([_48781],_48779,[]).

ctr_automaton_signature(
    min_decreasing_slope,
    min_decreasing_slope_a,
    pair_signature(2,signature)).

min_decreasing_slope_a(FLAG,MIN,VARIABLES) :-
        check_type(dvar_gteq(0),MIN),
        collection(VARIABLES,[dvar]),
        min_decreasing_slope_signature(
            VARIABLES,
            SIGNATURE,
            DIFFERENCES),
        automaton(
            DIFFERENCES,
            Di,
            SIGNATURE,
            [source(s),sink(t),sink(s)],
            [arc(s,0,s),
             arc(s,1,t,[Di]),
             arc(t,0,t),
             arc(t,1,t,[min(M,Di)])],
            [M],
            [0],
            [MINIMUM]),
        MINIMUM#=MIN#<=>FLAG.

min_decreasing_slope_signature([_48784],[],[]) :-
        !.

min_decreasing_slope_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|RS],
    [DIFFERENCE|RD]) :-
        VAR1#=<VAR2#<=>S#=0,
        VAR1#>VAR2#<=>S#=1,
        VAR1#=DIFFERENCE+VAR2,
        min_decreasing_slope_signature([[var-VAR2]|VARs],RS,RD).

min_decreasing_slope_a_s(FLAG,MIN,VARIABLES,SIGNATURE) :-
        check_type(dvar_gteq(0),MIN),
        collection(VARIABLES,[dvar]),
        difference_decreasing_slope_signature(
```

```
        VARIABLES,
        DIFFERENCES),
    automaton(
        DIFFERENCES,
        Di,
        SIGNATURE,
        [source(s),sink(t),sink(s)],
        [arc(s,0,s),
         arc(s,1,s),
         arc(s,2,t,[Di]),
         arc(t,0,t),
         arc(t,1,t),
         arc(t,2,t,[min(M,Di)])],
        [M],
        [0],
        [MINIMUM]),
    MINIMUM#=MIN#<=>FLAG.
```

B.256 min_dist_between_inflexion

◊ **META-DATA:**

```
ctr_date(min_dist_between_inflexion,['20121023']).

ctr_origin(
    min_dist_between_inflexion,
    Derived from %c,
    [inflexion]).

ctr_arguments(
    min_dist_between_inflexion,
    ['MINDIST'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_dist_between_inflexion,
    ['MINDIST'>=0,
     'MINDIST'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    min_dist_between_inflexion,
    min_dist_between_inflexion(
        2,
        [[var-2],
         [var-2],
         [var-3],
         [var-3],
         [var-2],
         [var-2],
         [var-1],
         [var-4],
         [var-4],
         [var-3]])).

ctr_typical(
    min_dist_between_inflexion,
    ['MINDIST'>1,size('VARIABLES')>3,range('VARIABLES'^var)>1]).

ctr_typical_model(
    min_dist_between_inflexion,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    min_dist_between_inflexion,
```

```
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    min_dist_between_inflexion,
    [checker(min_dist_between_inflexion_c),
     automaton(min_dist_between_inflexion_a),
     automaton_with_signature(min_dist_between_inflexion_a_s)]).

ctr_total_relation(min_dist_between_inflexion).

ctr_sol(min_dist_between_inflexion,2,0,2,9,[2-9]).

ctr_sol(min_dist_between_inflexion,3,0,3,64,[3-64]).

ctr_sol(
    min_dist_between_inflexion,
    4,
    0,
    4,
    1135,
    [1-170,2-170,3-170,4-625]).

ctr_sol(
    min_dist_between_inflexion,
    5,
    0,
    5,
    25444,
    [1-3598,2-4690,3-4690,4-4690,5-7776]).

ctr_sol(
    min_dist_between_inflexion,
    6,
    0,
    6,
    574483,
    [1-73794,2-91098,3-97314,4-97314,5-97314,6-117649]).

ctr_sol(
    min_dist_between_inflexion,
    7,
    0,
    7,
    13287476,
    [1-1543512,
     2-1819764,
```

```
     3-1932012,
     4-1965012,
     5-1965012,
     6-1965012,
     7-2097152]).

ctr_sol(
    min_dist_between_inflexion,
    8,
    0,
    8,
    328156407,
    [1-35152278,
     2-39992562,
     3-41360676,
     4-42025560,
     5-42192870,
     6-42192870,
     7-42192870,
     8-43046721]).

min_dist_between_inflexion_c(MINDIST,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        check_type(dvar(0,L),MINDIST),
        get_attr1(VARIABLES,VARS),
        min_dist_between_inflexion_c(VARS,s,L,1,MINDIST).

min_dist_between_inflexion_c([V,V|VARS],s,D,C,MINDIST) :-
        !,
        min_dist_between_inflexion_c([V|VARS],s,D,C,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],s,D,C,MINDIST) :-
        Vi<Vj,
        !,
        min_dist_between_inflexion_c([Vj|VARS],i0,D,C,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],s,D,C,MINDIST) :-
        Vi>Vj,
        !,
        min_dist_between_inflexion_c([Vj|VARS],d0,D,C,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],i0,D,C,MINDIST) :-
        Vi=<Vj,
        !,
        min_dist_between_inflexion_c([Vj|VARS],i0,D,C,MINDIST).
```

```
min_dist_between_inflexion_c([Vi,Vj|VARS],i0,D,C,MINDIST) :-
        Vi>Vj,
        !,
        min_dist_between_inflexion_c([Vj|VARS],d1,D,C,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],d0,D,C,MINDIST) :-
        Vi>=Vj,
        !,
        min_dist_between_inflexion_c([Vj|VARS],d0,D,C,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],d0,D,C,MINDIST) :-
        Vi<Vj,
        !,
        min_dist_between_inflexion_c([Vj|VARS],i1,D,C,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],d1,D,C,MINDIST) :-
        Vi>=Vj,
        !,
        C1 is C+1,
        min_dist_between_inflexion_c([Vj|VARS],d1,D,C1,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],d1,D,C,MINDIST) :-
        Vi<Vj,
        !,
        D1 is min(D,C),
        min_dist_between_inflexion_c([Vj|VARS],i1,D1,1,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],i1,D,C,MINDIST) :-
        Vi=<Vj,
        !,
        C1 is C+1,
        min_dist_between_inflexion_c([Vj|VARS],i1,D,C1,MINDIST).

min_dist_between_inflexion_c([Vi,Vj|VARS],i1,D,C,MINDIST) :-
        Vi>Vj,
        !,
        D1 is min(D,C),
        min_dist_between_inflexion_c([Vj|VARS],d1,D1,1,MINDIST).

min_dist_between_inflexion_c(_45979,_45980,D,_45982,MINDIST) :-
        (   integer(MINDIST) ->
            MINDIST>=D
        ;   MINDIST#=D
        ).
```

```
ctr_automaton_signature(
    min_dist_between_inflexion,
    min_dist_between_inflexion_a,
    pair_signature(2,signature)).

min_dist_between_inflexion_a(FLAG,MINDIST,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        min_dist_between_inflexion_a_s(
            FLAG,
            MINDIST,
            VARIABLES,
            SIGNATURE).

min_dist_between_inflexion_a_s(
    FLAG,
    MINDIST,
    VARIABLES,
    SIGNATURE) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        check_type(dvar(0,L),MINDIST),
        automaton(
            SIGNATURE,
            _47856,
            SIGNATURE,
            [source(s),
             sink(s),
             sink(i0),
             sink(d0),
             sink(i1),
             sink(d1)],
            [arc(s,1,s),
             arc(s,0,i0),
             arc(s,2,d0),
             arc(i0,1,i0),
             arc(i0,0,i0),
             arc(i0,2,d1),
             arc(d0,1,d0),
             arc(d0,0,i1),
             arc(d0,2,d0),
             arc(i1,1,i1,[D,C+1]),
             arc(i1,0,i1,[D,C+1]),
             arc(i1,2,d1,[min(D,C),1]),
             arc(d1,1,d1,[D,C+1]),
             arc(d1,0,i1,[min(D,C),1]),
             arc(d1,2,d1,[D,C+1])],
```

```
    [D,C],
    [L,1],
    [DIST,_46206]),
MINDIST#>=DIST#<=>FLAG,
(   integer(MINDIST) ->
    true
;   FLAG=0 ->
    true
;   MINDIST#=DIST
).
```

B.257 min_increasing_slope

◇ **META-DATA:**

```
ctr_date(min_increasing_slope,['20130315']).

ctr_origin(min_increasing_slope,'Motivated by time series.',[]).

ctr_arguments(
    min_increasing_slope,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_increasing_slope,
    ['MIN'>=0,
     'MIN'<range('VARIABLES'^var),
     required('VARIABLES',var),
     size('VARIABLES')>0]).

ctr_example(
    min_increasing_slope,
    [min_increasing_slope(
         3,
         [[var-1],
          [var-1],
          [var-5],
          [var-8],
          [var-6],
          [var-2],
          [var-2],
          [var-1],
          [var-5]]),
     min_increasing_slope(
         0,
         [[var-8],[var-8],[var-2],[var-0],[var-0]]),
     min_increasing_slope(
         9,
         [[var-1],[var-1],[var-0],[var-9],[var-6]])]).

ctr_typical(
    min_increasing_slope,
    ['MIN'>1,size('VARIABLES')>2,range('VARIABLES'^var)>2]).

ctr_typical_model(
    min_increasing_slope,
    [nval('VARIABLES'^var)>2]).
```

```
ctr_exchangeable(
    min_increasing_slope,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    min_increasing_slope,
    [checker(min_increasing_slope_c),
     automaton(min_increasing_slope_a),
     automaton_with_signature(min_increasing_slope_a_s)]).

ctr_pure_functional_dependency(min_increasing_slope,[]).

ctr_functional_dependency(min_increasing_slope,1,[2]).

ctr_cond_imply(
    min_increasing_slope,
    max_increasing_slope,
    [range('VARIABLES'^var)='MIN'+1],
    [range('VARIABLES'^var)='MAX'+1],
    [none,'VARIABLES']).

ctr_sol(min_increasing_slope,2,0,2,9,[0-6,1-2,2-1]).

ctr_sol(min_increasing_slope,3,0,3,64,[0-20,1-22,2-14,3-8]).

ctr_sol(
    min_increasing_slope,
    4,
    0,
    4,
    625,
    [0-70,1-256,2-145,3-98,4-56]).

ctr_sol(
    min_increasing_slope,
    5,
    0,
    5,
    7776,
    [0-252,1-3512,2-1864,3-1062,4-704,5-382]).

ctr_sol(
    min_increasing_slope,
    6,
    0,
```

```
    6,
    117649,
    [0-924,1-56537,2-28728,3-14729,4-8853,5-5266,6-2612]).

ctr_sol(
    min_increasing_slope,
    7,
    0,
    7,
    2097152,
    [0-3432,
     1-1051936,
     2-515372,
     3-255076,
     4-133672,
     5-78198,
     6-41330,
     7-18136]).

ctr_sol(
    min_increasing_slope,
    8,
    0,
    8,
    43046721,
    [0-12870,
     1-22280084,
     2-10601773,
     3-5106480,
     4-2475484,
     5-1369232,
     6-730161,
     7-341618,
     8-129019]).

min_increasing_slope_c(MIN,VARIABLES) :-
        check_type(dvar_gteq(0),MIN),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        N>0,
        min_increasing_slope_c1(VARS,0,MIN).

min_increasing_slope_c1([_48552],MIN,MIN) :-
        !.
```

```
min_increasing_slope_c1(_48549,1,MIN) :-
        !,
        MIN=1.

min_increasing_slope_c1([V1,V2|R],M,MIN) :-
        V1>=V2,
        !,
        min_increasing_slope_c1([V2|R],M,MIN).

min_increasing_slope_c1([V1,V2|R],M,MIN) :-
        (   M=0 ->
            N is V2-V1
        ;   N is min(M,V2-V1)
        ),
        min_increasing_slope_c1([V2|R],N,MIN).

min_increasing_slope_counters_check(L,[0|S]) :-
        min_increasing_slope_counters_check(L,0,S).

min_increasing_slope_counters_check([V1,V2|R],M,[M|S]) :-
        V1>=V2,
        !,
        min_increasing_slope_counters_check([V2|R],M,S).

min_increasing_slope_counters_check([V1,V2|R],M,[N|S]) :-
        !,
        (   M=0 ->
            N is V2-V1
        ;   N is min(M,V2-V1)
        ),
        min_increasing_slope_counters_check([V2|R],N,S).

min_increasing_slope_counters_check([_48549],_48547,[]).

ctr_automaton_signature(
    min_increasing_slope,
    min_increasing_slope_a,
    pair_signature(2,signature)).

min_increasing_slope_a(FLAG,MIN,VARIABLES) :-
        check_type(dvar_gteq(0),MIN),
        collection(VARIABLES,[dvar]),
        min_increasing_slope_signature(
            VARIABLES,
            SIGNATURE,
            DIFFERENCES),
```

```
        automaton(
            DIFFERENCES,
            Di,
            SIGNATURE,
            [source(s),sink(t),sink(s)],
            [arc(s,0,s),
             arc(s,1,t,[Di]),
             arc(t,0,t),
             arc(t,1,t,[min(M,Di)])],
            [M],
            [0],
            [MINIMUM]),
        MINIMUM#=MIN#<=>FLAG.

min_increasing_slope_signature([_48552],[],[]) :-
        !.

min_increasing_slope_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|RS],
    [DIFFERENCE|RD]) :-
        VAR1#>=VAR2#<=>S#=0,
        VAR1#<VAR2#<=>S#=1,
        VAR2#=DIFFERENCE+VAR1,
        min_increasing_slope_signature([[var-VAR2]|VARs],RS,RD).

min_increasing_slope_a_s(FLAG,MIN,VARIABLES,SIGNATURE) :-
        check_type(dvar_gteq(0),MIN),
        collection(VARIABLES,[dvar]),
        difference_increasing_slope_signature(
            VARIABLES,
            DIFFERENCES),
        automaton(
            DIFFERENCES,
            Di,
            SIGNATURE,
            [source(s),sink(t),sink(s)],
            [arc(s,1,s),
             arc(s,2,s),
             arc(s,0,t,[Di]),
             arc(t,1,t),
             arc(t,2,t),
             arc(t,0,t,[min(M,Di)])],
            [M],
            [0],
            [MINIMUM]),
```

```
MINIMUM#=MIN#<=>FLAG.
```

B.258 min_index

◇ **META-DATA:**

```
ctr_date(
    min_index,
    ['20030820','20040530','20041230','20060811']).

ctr_origin(min_index,'N.~Beldiceanu',[]).

ctr_arguments(
    min_index,
    ['MIN_INDEX'-dvar,
     'VARIABLES'-collection(index-int,var-dvar)]).

ctr_restrictions(
    min_index,
    [size('VARIABLES')>0,
     'MIN_INDEX'>=0,
     'MIN_INDEX'=<size('VARIABLES'),
     required('VARIABLES',[index,var]),
     'VARIABLES'^index>=1,
     'VARIABLES'^index=<size('VARIABLES'),
     distinct('VARIABLES',index)]).

ctr_example(
    min_index,
    [min_index(
         2,
         [[index-1,var-3],
          [index-2,var-2],
          [index-3,var-7],
          [index-4,var-2],
          [index-5,var-6]]),
     min_index(
         4,
         [[index-1,var-3],
          [index-2,var-2],
          [index-3,var-7],
          [index-4,var-2],
          [index-5,var-6]])]).

ctr_typical(
    min_index,
    [size('VARIABLES')>0,range('VARIABLES'^var)>1]).
```

```
ctr_exchangeable(
    min_index,
    [items('VARIABLES',all),translate(['VARIABLES'^var])]).

ctr_graph(
    min_index,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^key=variables2^key#\/
     variables1^var<variables2^var],
    ['ORDER'(0,0,index)='MIN_INDEX'],
    []).
```

B.259 min_n

◇ **META-DATA:**

```
ctr_date(
    min_n,
    ['20000128','20030820','20040530','20041230','20060811']).

ctr_origin(min_n,'\\cite{Beldiceanu01}',[]).

ctr_arguments(
    min_n,
    ['MIN'-dvar,'RANK'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_n,
    [size('VARIABLES')>0,
     'RANK'>=0,
     'RANK'<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    min_n,
    min_n(3,1,[[var-3],[var-1],[var-7],[var-1],[var-6]])).

ctr_typical(
    min_n,
    ['RANK'>0,
     'RANK'<3,
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1]).

ctr_typical_model(min_n,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    min_n,
    [items('VARIABLES',all),
     translate(['MIN','VARIABLES'^var])]).

ctr_graph(
    min_n,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^key=variables2^key#\/
     variables1^var<variables2^var],
```

```
    ['ORDER'('RANK','MAXINT',var)='MIN'],
    []).

ctr_eval(min_n,[checker(min_n_c),reformulation(min_n_r)]).

ctr_pure_functional_dependency(min_n,[]).

ctr_functional_dependency(min_n,1,[2,3]).

ctr_cond_imply(
    min_n,
    atleast,
    [],
    ['N'=1],
    [none,'VARIABLES','MIN']).

ctr_cond_imply(
    min_n,
    minimum_greater_than,
    ['RANK'=1,minval('VARIABLES'^var)=1],
    [],
    id).

min_n_c(MIN,RANK,VARIABLES) :-
        length(VARIABLES,N),
        N>0,
        N1 is N-1,
        check_type(dvar,MIN),
        check_type(int(0,N1),RANK),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        sort(VARS,SVARS),
        nth0(RANK,SVARS,MIN).

min_n_r(MIN,RANK,VARIABLES) :-
        length(VARIABLES,N),
        N>0,
        N1 is N-1,
        check_type(dvar,MIN),
        check_type(int(0,N1),RANK),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        create_collection([MIN],var,VMIN),
        create_collection(VARS,val,VALUES),
        eval(among_var(1,VMIN,VALUES)),
        NVAL in 0..N,
```

```
        eval(nvalue(NVAL,VARIABLES)),
        length(RANKS,N),
        domain(RANKS,0,N1),
        min_n1(VARS,RANKS,MIN,RANK,NVAL).

min_n1([],[],_47302,_47303,_47304).

min_n1([V|RV],[R|RR],MIN,RANK,NVAL) :-
        R#<NVAL,
        R#=RANK#<=>V#=MIN,
        min_n2(RV,RR,V,R),
        min_n1(RV,RR,MIN,RANK,NVAL).

min_n2([],[],_47302,_47303).

min_n2([Vj|RV],[Rj|RR],Vi,Ri) :-
        Vi#<Vj#<=>Ri#<Rj,
        Vi#=Vj#<=>Ri#=Rj,
        Vi#>Vj#<=>Ri#>Rj,
        min_n2(RV,RR,Vi,Ri).
```

B.260 min_nvalue

♢ META-DATA:

```
ctr_date(min_nvalue,['20000128','20030820','20060811']).

ctr_origin(min_nvalue,'N.~Beldiceanu',[]).

ctr_arguments(
    min_nvalue,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_nvalue,
    ['MIN'>=1,
     'MIN'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    min_nvalue,
    [min_nvalue(
         2,
         [[var-9],
          [var-1],
          [var-7],
          [var-1],
          [var-1],
          [var-7],
          [var-7],
          [var-7],
          [var-7],
          [var-9]]),
     min_nvalue(5,[[var-8],[var-8],[var-8],[var-8],[var-8]]),
     min_nvalue(2,[[var-1],[var-8],[var-1],[var-8],[var-1]])]).

ctr_typical(
    min_nvalue,
    [2*'MIN'=<size('VARIABLES'),
     size('VARIABLES')>1,
     range('VARIABLES'^var)>1]).

ctr_exchangeable(
    min_nvalue,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).
```

```
ctr_graph(
    min_nvalue,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['MIN_NSCC'='MIN'],
    []).

ctr_eval(
    min_nvalue,
    [checker(min_nvalue_c),reformulation(min_nvalue_r)]).

ctr_pure_functional_dependency(min_nvalue,[]).

ctr_functional_dependency(min_nvalue,1,[2]).

ctr_cond_imply(
    min_nvalue,
    atleast_nvalue,
    ['MIN'<size('VARIABLES')],
    ['NVAL'=2],
    [none,'VARIABLES']).

ctr_sol(min_nvalue,2,0,2,9,[1-6,2-3]).

ctr_sol(min_nvalue,3,0,3,64,[1-60,3-4]).

ctr_sol(min_nvalue,4,0,4,625,[1-560,2-60,4-5]).

ctr_sol(min_nvalue,5,0,5,7776,[1-7470,2-300,5-6]).

ctr_sol(min_nvalue,6,0,6,117649,[1-113442,2-3780,3-420,6-7]).

ctr_sol(
    min_nvalue,
    7,
    0,
    7,
    2097152,
    [1-2058728,2-36456,3-1960,7-8]).

ctr_sol(
    min_nvalue,
    8,
    0,
```

```
    8,
    43046721,
    [1-42473664,2-566496,3-4032,4-2520,8-9]).

min_nvalue_c(0,[]) :-
        !.

min_nvalue_c(MIN,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(1,N),MIN),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        samsort(VARS,SVARS),
        SVARS=[V|R],
        min_nvalue_seq_size(R,1,V,N,M),
        MIN#=M.

min_nvalue_seq_size([],C,_81666,Best,Res) :-
        !,
        Res is min(C,Best).

min_nvalue_seq_size([V|R],C,V,Best,Res) :-
        !,
        C1 is C+1,
        min_nvalue_seq_size(R,C1,V,Best,Res).

min_nvalue_seq_size([V|R],C,Prev,Best,Res) :-
        C>0,
        V=\=Prev,
        NewBest is min(C,Best),
        min_nvalue_seq_size(R,1,V,NewBest,Res).

min_nvalue_r(0,[]) :-
        !.

min_nvalue_r(MIN,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(1,N),MIN),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        union_dom_list_int(VARS,UnionDomainsVARS),
        NSquare is N*N,
        length(UnionDomainsVARS,SizeUnion),
        (   SizeUnion=<NSquare ->
            balance1(UnionDomainsVARS,N,VALS,_OCCS,OCCS1),
            eval(global_cardinality(VARIABLES,VALS))
```

```
;   balance2(VARS,N,VARS,OCCS1)
),
eval(minimum(MIN,OCCS1)).
```

B.261 min_size_full_zero_stretch

◊ **META-DATA:**

```
ctr_date(min_size_full_zero_stretch,['20121023']).

ctr_origin(
    min_size_full_zero_stretch,
    Derived from the unit commitment problem,
    []).

ctr_arguments(
    min_size_full_zero_stretch,
    ['MINSIZE'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_size_full_zero_stretch,
    ['MINSIZE'>=0,
     'MINSIZE'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    min_size_full_zero_stretch,
    min_size_full_zero_stretch(
        2,
        [[var-0],
         [var-2],
         [var-0],
         [var-0],
         [var-0],
         [var-2],
         [var-1],
         [var-0],
         [var-0],
         [var-3]])).

ctr_typical(
    min_size_full_zero_stretch,
    [size('VARIABLES')>2,
     range('VARIABLES'^var)>1,
     size('VARIABLES')-among_diff_0('VARIABLES'^var)>1]).

ctr_typical_model(
    min_size_full_zero_stretch,
    [atleast(2,'VARIABLES',0)]).
```

```
ctr_exchangeable(
    min_size_full_zero_stretch,
    [items('VARIABLES',reverse),
     vals(
         ['VARIABLES'^var],
         int(=\=(0)),
         =\=,
         dontcare,
         dontcare)]).

ctr_eval(
    min_size_full_zero_stretch,
    [checker(min_size_full_zero_stretch_c),
     automaton(min_size_full_zero_stretch_a)]).

ctr_total_relation(min_size_full_zero_stretch).

ctr_sol(min_size_full_zero_stretch,2,0,2,9,[2-9]).

ctr_sol(min_size_full_zero_stretch,3,0,3,82,[1-9,2-9,3-64]).

ctr_sol(
    min_size_full_zero_stretch,
    4,
    0,
    4,
    1137,
    [1-160,2-176,3-176,4-625]).

ctr_sol(
    min_size_full_zero_stretch,
    5,
    0,
    5,
    19026,
    [1-2575,2-2875,3-2900,4-2900,5-7776]).

ctr_sol(
    min_size_full_zero_stretch,
    6,
    0,
    6,
    364033,
    [1-45072,2-49932,3-50436,4-50472,5-50472,6-117649]).

ctr_sol(
```

```
    min_size_full_zero_stretch,
    7,
    0,
    7,
    7850291,
    [1-882441,
     2-966672,
     3-975394,
     4-976178,
     5-976227,
     6-976227,
     7-2097152]).

ctr_sol(
    min_size_full_zero_stretch,
    8,
    0,
    8,
    188987201,
    [1-19330432,
     2-20958912,
     3-21117888,
     4-21132416,
     5-21133568,
     6-21133632,
     7-21133632,
     8-43046721]).

min_size_full_zero_stretch_a(FLAG,MINSIZE,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        check_type(dvar(0,L),MINSIZE),
        min_size_full_zero_stretch_signature(
            VARIABLES,
            SIGNATURE),
        automaton(
            SIGNATURE,
            _47129,
            SIGNATURE,
            [source(s),sink(s),sink(i),sink(j)],
            [arc(s,0,s),
             arc(s,1,i),
             arc(i,1,i),
             arc(i,0,j,[M,C+1]),
             arc(j,0,j,[M,C+1]),
             arc(j,1,i,[min(M,C),0])],
```

```
            [M,C],
            [L,0],
            [MIN,_45578]),
        MINSIZE#>=MIN#<=>FLAG,
        (   integer(MINSIZE) ->
            true
        ;   FLAG=0 ->
            true
        ;   MINSIZE#=MIN
        ).

min_size_full_zero_stretch_signature([],[]).

min_size_full_zero_stretch_signature([[var-VAR]|VARs],[S|Ss]) :-
        VAR#\=0#<=>S,
        min_size_full_zero_stretch_signature(VARs,Ss).

min_size_full_zero_stretch_c(MINSIZE,VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        check_type(dvar(0,L),MINSIZE),
        get_attr1(VARIABLES,VARS),
        min_size_full_zero_stretch_c(VARS,s,L,0,MINSIZE).

min_size_full_zero_stretch_c([0|R],s,M,C,MINSIZE) :-
        !,
        min_size_full_zero_stretch_c(R,s,M,C,MINSIZE).

min_size_full_zero_stretch_c([_45437|R],s,M,C,MINSIZE) :-
        !,
        min_size_full_zero_stretch_c(R,i,M,C,MINSIZE).

min_size_full_zero_stretch_c([0|R],i,M,C,MINSIZE) :-
        !,
        C1 is C+1,
        min_size_full_zero_stretch_c(R,j,M,C1,MINSIZE).

min_size_full_zero_stretch_c([_45437|R],i,M,C,MINSIZE) :-
        !,
        min_size_full_zero_stretch_c(R,i,M,C,MINSIZE).

min_size_full_zero_stretch_c([0|R],j,M,C,MINSIZE) :-
        !,
        C1 is C+1,
        min_size_full_zero_stretch_c(R,j,M,C1,MINSIZE).
```

```
min_size_full_zero_stretch_c([_45437|R],j,M,C,MINSIZE) :-
        !,
        M1 is min(M,C),
        min_size_full_zero_stretch_c(R,i,M1,0,MINSIZE).

min_size_full_zero_stretch_c([],_45433,M,_45435,MINSIZE) :-
        (   integer(MINSIZE) ->
            MINSIZE>=M
        ;   MINSIZE#=M
        ).
```

B.262 min_size_set_of_consecutive_var

◇ **META-DATA:**

```
ctr_date(
    min_size_set_of_consecutive_var,
    ['20030820','20040530','20060811']).

ctr_origin(min_size_set_of_consecutive_var,'N.~Beldiceanu',[]).

ctr_arguments(
    min_size_set_of_consecutive_var,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_size_set_of_consecutive_var,
    ['MIN'>=1,
     'MIN'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    min_size_set_of_consecutive_var,
    [min_size_set_of_consecutive_var(
         4,
         [[var-3],
          [var-1],
          [var-3],
          [var-7],
          [var-4],
          [var-1],
          [var-2],
          [var-8],
          [var-7],
          [var-6]]),
     min_size_set_of_consecutive_var(
         4,
         [[var-3],[var-1],[var-3],[var-2]])]).

ctr_typical(
    min_size_set_of_consecutive_var,
    ['MIN'>1,
     'MIN'<size('VARIABLES'),
     size('VARIABLES')>0,
     range('VARIABLES'^var)>1]).

ctr_exchangeable(
```

```
    min_size_set_of_consecutive_var,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     translate(['VARIABLES'^var])]).

ctr_graph(
    min_size_set_of_consecutive_var,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [abs(variables1^var-variables2^var)=<1],
    ['MIN_NSCC'='MIN'],
    []).

ctr_eval(
    min_size_set_of_consecutive_var,
    [checker(min_size_set_of_consecutive_var_c)]).

ctr_pure_functional_dependency(
    min_size_set_of_consecutive_var,
    []).

ctr_functional_dependency(
    min_size_set_of_consecutive_var,
    1,
    [2]).

ctr_sol(min_size_set_of_consecutive_var,2,0,2,9,[1-2,2-7]).

ctr_sol(min_size_set_of_consecutive_var,3,0,3,64,[1-30,3-34]).

ctr_sol(
    min_size_set_of_consecutive_var,
    4,
    0,
    4,
    625,
    [1-276,2-132,4-217]).

ctr_sol(
    min_size_set_of_consecutive_var,
    5,
    0,
    5,
    7776,
    [1-3580,2-2480,5-1716]).
```

```
ctr_sol(
    min_size_set_of_consecutive_var,
    6,
    0,
    6,
    117649,
    [1-57000,2-30990,3-13500,6-16159]).

ctr_sol(
    min_size_set_of_consecutive_var,
    7,
    0,
    7,
    2097152,
    [1-1065834,2-522522,3-332430,7-176366]).

ctr_sol(
    min_size_set_of_consecutive_var,
    8,
    0,
    8,
    43046721,
    [1-22894984,2-11080412,3-4590208,4-2293480,8-2187637]).

min_size_set_of_consecutive_var_c(MIN,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(1,N),MIN),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        samsort(VARS,SVARS),
        SVARS=[V|R],
        min_size_set_of_consecutive_var_c(R,V,1,N,M),
        MIN#=M.

min_size_set_of_consecutive_var_c([V|R],Prev,Occ,MinOcc,Res) :-
        Diff is V-Prev,
        Diff=<1,
        !,
        Occ1 is Occ+1,
        min_size_set_of_consecutive_var_c(R,V,Occ1,MinOcc,Res).

min_size_set_of_consecutive_var_c([V|R],_75649,Occ,MinOcc,Res) :-
        !,
        Min is min(Occ,MinOcc),
        min_size_set_of_consecutive_var_c(R,V,1,Min,Res).
```

```
min_size_set_of_consecutive_var_c([],_75649,Occ,MinOcc,Res) :-
        Res is min(Occ,MinOcc).
```

B.263 min_surf_peak

◇ **META-DATA:**

```
ctr_date(min_surf_peak,['20141110']).

ctr_origin(min_surf_peak,'derived from %c',[peak]).

ctr_arguments(
    min_surf_peak,
    ['MIN_SURF'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_surf_peak,
    ['MIN_SURF'>=0,
     'MIN_SURF'=<sum('VARIABLES'^var),
     required('VARIABLES',var)]).

ctr_example(
    min_surf_peak,
    [min_surf_peak(
         12,
         [[var-4],
          [var-4],
          [var-2],
          [var-2],
          [var-3],
          [var-5],
          [var-5],
          [var-6],
          [var-3],
          [var-1],
          [var-1],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-1]]),
     min_surf_peak(
         35,
         [[var-4],
          [var-6],
          [var-7],
          [var-9],
```

```
           [var-8],
           [var-5],
           [var-4]]),
      min_surf_peak(
          0,
          [[var-4],
           [var-4],
           [var-2],
           [var-0],
           [var-0],
           [var-4],
           [var-5]])]).

ctr_typical(min_surf_peak,['MIN_SURF'>1,size('VARIABLES')>2]).

ctr_typical_model(min_surf_peak,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    min_surf_peak,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    min_surf_peak,
    [checker(min_surf_peak_c),automaton(min_surf_peak_a)]).

ctr_pure_functional_dependency(min_surf_peak,[]).

ctr_functional_dependency(min_surf_peak,1,[2]).

min_surf_peak_c(0,[]) :-
        !.

min_surf_peak_c(MIN_SURF,VARIABLES) :-
        check_type(dvar,MIN_SURF),
        MIN_SURF#>=0,
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        get_sum(VARS,SUM),
        MIN_SURF#=<SUM,
        min_surf_peak_c(VARS,s,1,SUM,0,0,MIN_SURF).

min_surf_peak_c([VAR1,VAR2|R],s,I,S,C,D,MIN_SURF) :-
        VAR1>=VAR2,
        !,
        I1 is I+1,
        min_surf_peak_c([VAR2|R],s,I1,S,C,D,MIN_SURF).
```

```
min_surf_peak_c([_VAR1,VAR2|R],s,I,S,C,_D,MIN_SURF) :-
        !,
        I1 is I+1,
        min_surf_peak_c([VAR2|R],j,I1,S,C,VAR2,MIN_SURF).

min_surf_peak_c([VAR1,VAR2|R],j,I,S,C,D,MIN_SURF) :-
        VAR1=<VAR2,
        !,
        I1 is I+1,
        D1 is D+VAR2,
        min_surf_peak_c([VAR2|R],j,I1,S,C,D1,MIN_SURF).

min_surf_peak_c([_VAR1,VAR2|R],j,I,S,_C,D,MIN_SURF) :-
        !,
        I1 is I+1,
        min_surf_peak_c([VAR2|R],k,I1,S,D,0,MIN_SURF).

min_surf_peak_c([VAR1,VAR2|R],k,I,S,C,D,MIN_SURF) :-
        VAR1=VAR2,
        !,
        I1 is I+1,
        D1 is D+VAR1,
        min_surf_peak_c([VAR2|R],k,I1,S,C,D1,MIN_SURF).

min_surf_peak_c([VAR1,VAR2|R],k,I,S,C,D,MIN_SURF) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        C1 is C+D+VAR1,
        min_surf_peak_c([VAR2|R],k,I1,S,C1,0,MIN_SURF).

min_surf_peak_c([_VAR1,VAR2|R],k,I,S,C,_D,MIN_SURF) :-
        !,
        I1 is I+1,
        S1 is min(S,C),
        min_surf_peak_c([VAR2|R],j,I1,S1,C,VAR2,MIN_SURF).

min_surf_peak_c([_28137],_28131,_28132,S,C,_28135,MIN_SURF) :-
        MIN_SURF#=min(S,C).

min_surf_peak_a(FLAG,MIN_SURF,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        min_surf_peak_a_s(FLAG,MIN_SURF,VARIABLES,SIGNATURE).

min_surf_peak_a_s(FLAG,MIN_SURF,VARIABLES,SIGNATURE) :-
```

```
check_type(dvar,MIN_SURF),
length(VARIABLES,N),
(   N=0 ->
    Sn=0,
    Cn=0
;   MIN_SURF#>=0,
    collection(VARIABLES,[dvar]),
    get_attr1(VARIABLES,VARS),
    get_sum(VARS,SUM),
    MIN_SURF#=<SUM,
    gen_pairs(VARS,PAIRS),
    automaton(
        PAIRS,
        VAR1-VAR2,
        SIGNATURE,
        [source(s),sink(s),sink(j),sink(k)],
        [arc(s,2,s),
         arc(s,1,s),
         arc(s,0,j,[S,C,VAR2]),
         arc(j,2,k,[S,D,0]),
         arc(j,1,j,[S,C,D+VAR2]),
         arc(j,0,j,[S,C,D+VAR2]),
         arc(k,2,k,[S,C+D+VAR1,0]),
         arc(k,1,k,[S,C,D+VAR1]),
         arc(k,0,j,[min(S,C),C,VAR2])],
        [S,C,D],
        [SUM,0,0],
        [Sn,Cn,_28400])
),
MIN_SURF#=min(Sn,Cn)#<=>FLAG.
```

B.264 min_width_peak

◇ **META-DATA:**

```
ctr_date(min_width_peak,['20121201']).

ctr_origin(min_width_peak,'derived from %c',[peak]).

ctr_synonyms(min_width_peak,[min_base_peak]).

ctr_arguments(
    min_width_peak,
    ['MIN_WIDTH'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_width_peak,
    ['MIN_WIDTH'>=0,
     'MIN_WIDTH'=<size('VARIABLES')-2,
     required('VARIABLES',var)]).

ctr_example(
    min_width_peak,
    [min_width_peak(
         5,
         [[var-4],
          [var-4],
          [var-2],
          [var-2],
          [var-3],
          [var-5],
          [var-5],
          [var-6],
          [var-3],
          [var-1],
          [var-1],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-1]]),
     min_width_peak(
         5,
         [[var-4],
          [var-6],
```

```
          [var-7],
          [var-9],
          [var-8],
          [var-5],
          [var-4]]),
     min_width_peak(
         0,
         [[var-4],
          [var-4],
          [var-2],
          [var-0],
          [var-0],
          [var-4],
          [var-5]])]).

ctr_typical(min_width_peak,['MIN_WIDTH'>1,size('VARIABLES')>2]).

ctr_typical_model(min_width_peak,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    min_width_peak,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    min_width_peak,
    [checker(min_width_peak_c),
     automaton(min_width_peak_a),
     automaton_with_signature(min_width_peak_a_s)]).

ctr_pure_functional_dependency(min_width_peak,[]).

ctr_functional_dependency(min_width_peak,1,[2]).

ctr_sol(min_width_peak,2,0,2,9,[0-9]).

ctr_sol(min_width_peak,3,0,3,64,[0-50,1-14]).

ctr_sol(min_width_peak,4,0,4,625,[0-295,1-230,2-100]).

ctr_sol(min_width_peak,5,0,5,7776,[0-1792,1-3205,2-2100,3-679]).

ctr_sol(
    min_width_peak,
    6,
    0,
    6,
```

```
    117649,
    [0-11088,1-56637,2-28420,3-17024,4-4480]).

ctr_sol(
    min_width_peak,
    7,
    0,
    7,
    2097152,
    [0-69498,1-1174398,2-424928,3-268722,4-130452,5-29154]).

ctr_sol(
    min_width_peak,
    8,
    0,
    8,
    43046721,
    [0-439791,
     1-26327058,
     2-9363060,
     3-3413256,
     4-2345982,
     5-968946,
     6-188628]).

min_width_peak_c(0,[]) :-
        !.

min_width_peak_c(MIN_WIDTH,VARIABLES) :-
        check_type(dvar,MIN_WIDTH),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        min_width_peak_c(VARS,s,1,0,N,0,N,MIN_WIDTH).

min_width_peak_c([VAR1,VAR2|R],s,I,C,W,F,N,MIN_WIDTH) :-
        VAR1>=VAR2,
        !,
        I1 is I+1,
        min_width_peak_c([VAR2|R],s,I1,C,W,F,N,MIN_WIDTH).

min_width_peak_c([VAR1,VAR2|R],s,I,C,W,_47615,N,MIN_WIDTH) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        min_width_peak_c([VAR2|R],j,I1,C,W,I,N,MIN_WIDTH).
```

```
min_width_peak_c([VAR1,VAR2|R],j,I,C,W,F,N,MIN_WIDTH) :-
        VAR1=<VAR2,
        !,
        I1 is I+1,
        min_width_peak_c([VAR2|R],j,I1,C,W,F,N,MIN_WIDTH).

min_width_peak_c([VAR1,VAR2|R],j,I,_47613,W,F,N,MIN_WIDTH) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        C1 is I-F,
        min_width_peak_c([VAR2|R],k,I1,C1,W,F,N,MIN_WIDTH).

min_width_peak_c([VAR1,VAR2|R],k,I,C,W,F,N,MIN_WIDTH) :-
        VAR1=VAR2,
        !,
        I1 is I+1,
        min_width_peak_c([VAR2|R],k,I1,C,W,F,N,MIN_WIDTH).

min_width_peak_c([VAR1,VAR2|R],k,I,_47613,W,F,N,MIN_WIDTH) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        C1 is I-F,
        min_width_peak_c([VAR2|R],k,I1,C1,W,F,N,MIN_WIDTH).

min_width_peak_c([VAR1,VAR2|R],k,I,C,W,_47615,N,MIN_WIDTH) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        W1 is min(W,C),
        min_width_peak_c([VAR2|R],j,I1,C,W1,I,N,MIN_WIDTH).

min_width_peak_c(
    [_47618],
    _48018,
    _48065,
    C,
    W,
    _48202,
    _48249,
    MIN_WIDTH) :-
        MIN_WIDTH#=min(W,C).

min_width_peak_counters_check(VARS,N,[0|COUNTERS]) :-
```

```
        min_width_peak_counters_check(
            VARS,
            s,
            1,
            0,
            N,
            0,
            N,
            COUNTERS).

min_width_peak_counters_check(
    [VAR1,VAR2|R],
    s,
    I,
    C,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1>=VAR2,
        !,
        I1 is I+1,
        MIN_WIDTH#=min(W,C),
        min_width_peak_counters_check(
            [VAR2|R],
            s,
            I1,
            C,
            W,
            F,
            N,
            REST).

min_width_peak_counters_check(
    [VAR1,VAR2|R],
    s,
    I,
    C,
    W,
    _48290,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        MIN_WIDTH#=min(W,C),
```

```
        min_width_peak_counters_check(
            [VAR2|R],
            j,
            I1,
            C,
            W,
            I,
            N,
            REST).

min_width_peak_counters_check(
    [VAR1,VAR2|R],
    j,
    I,
    C,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1=<VAR2,
        !,
        I1 is I+1,
        MIN_WIDTH#=min(W,C),
        min_width_peak_counters_check(
            [VAR2|R],
            j,
            I1,
            C,
            W,
            F,
            N,
            REST).

min_width_peak_counters_check(
    [VAR1,VAR2|R],
    j,
    I,
    _48211,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        C1 is I-F,
```

```
        MIN_WIDTH#=min(W,C1),
        min_width_peak_counters_check(
            [VAR2|R],
            k,
            I1,
            C1,
            W,
            F,
            N,
            REST).

min_width_peak_counters_check(
    [VAR1,VAR2|R],
    k,
    I,
    C,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1=VAR2,
        !,
        I1 is I+1,
        MIN_WIDTH#=min(W,C),
        min_width_peak_counters_check(
            [VAR2|R],
            k,
            I1,
            C,
            W,
            F,
            N,
            REST).

min_width_peak_counters_check(
    [VAR1,VAR2|R],
    k,
    I,
    _48211,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
```

```
        C1 is I-F,
        MIN_WIDTH#=min(W,C1),
        min_width_peak_counters_check(
            [VAR2|R],
            k,
            I1,
            C1,
            W,
            F,
            N,
            REST).

min_width_peak_counters_check(
    [VAR1,VAR2|R],
    k,
    I,
    C,
    W,
    _48301,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        W1 is min(W,C),
        MIN_WIDTH#=min(W1,C),
        min_width_peak_counters_check(
            [VAR2|R],
            j,
            I1,
            C,
            W1,
            I,
            N,
            REST).

min_width_peak_counters_check(
    [_47615],
    _47991,
    _48038,
    _48085,
    _48132,
    _48179,
    _48226,
    []).
```

```
ctr_automaton_signature(
    min_width_peak,
    min_width_peak_a,
    pair_signature(2,signature)).

min_width_peak_a(FLAG,MIN_WIDTH,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        min_width_peak_a_s(FLAG,MIN_WIDTH,VARIABLES,SIGNATURE).

min_width_peak_a_s(FLAG,MIN_WIDTH,VARIABLES,SIGNATURE) :-
        check_type(dvar,MIN_WIDTH),
        length(VARIABLES,N),
        (   N=0 ->
            Wn=0,
            Cn=0
        ;   collection(VARIABLES,[dvar]),
            pair_index_signature(VARIABLES,1,INDICES),
            automaton(
                INDICES,
                I,
                SIGNATURE,
                [source(s),sink(s),sink(j),sink(k)],
                [arc(s,2,s),
                 arc(s,1,s),
                 arc(s,0,j,[C,W,I]),
                 arc(j,2,k,[I-F,W,F]),
                 arc(j,1,j),
                 arc(j,0,j),
                 arc(k,2,k,[I-F,W,F]),
                 arc(k,1,k),
                 arc(k,0,j,[C,min(W,C),I])],
                [C,W,F],
                [0,N,0],
                [Cn,Wn,_47818])
        ),
        MIN_WIDTH#=min(Wn,Cn)#<=>FLAG.
```

B.265 min_width_plateau

◇ META-DATA:

```
ctr_date(min_width_plateau,['20141108']).

ctr_origin(min_width_plateau,'Derived from %c.',[peak]).

ctr_arguments(
    min_width_plateau,
    ['MIN_WIDTH'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_width_plateau,
    ['MIN_WIDTH'>=0,
     'MIN_WIDTH'=<size('VARIABLES')-2,
     required('VARIABLES',var)]).

ctr_example(
    min_width_plateau,
    [min_width_plateau(
         3,
         [[var-4],
          [var-4],
          [var-2],
          [var-2],
          [var-3],
          [var-5],
          [var-6],
          [var-6],
          [var-6],
          [var-1],
          [var-1],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-1]]),
     min_width_plateau(
         1,
         [[var-4],
          [var-6],
          [var-7],
          [var-9],
```

```
          [var-8],
          [var-5],
          [var-4]]),
     min_width_plateau(
         0,
         [[var-4],
          [var-4],
          [var-2],
          [var-0],
          [var-0],
          [var-4],
          [var-5]])]).

ctr_typical(
    min_width_plateau,
    [size('VARIABLES')>2,range('VARIABLES'^var)>1]).

ctr_typical_model(min_width_plateau,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    min_width_plateau,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    min_width_plateau,
    [checker(min_width_plateau_c),
     automaton(min_width_plateau_a)]).

ctr_pure_functional_dependency(min_width_plateau,[]).

ctr_functional_dependency(min_width_plateau,1,[2]).

min_width_plateau_c(MIN_WIDTH,VARIABLES) :-
        check_type(dvar,MIN_WIDTH),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        min_width_plateau_c(VARS,s,1,N,0,0,MIN_WIDTH).

min_width_plateau_c([VAR1,VAR2|R],s,I,C,D,P,MIN_WIDTH) :-
        VAR1>=VAR2,
        !,
        I1 is I+1,
        min_width_plateau_c([VAR2|R],s,I1,C,D,P,MIN_WIDTH).

min_width_plateau_c([_VAR1,VAR2|R],s,I,C,D,P,MIN_WIDTH) :-
```

```
        !,
        I1 is I+1,
        min_width_plateau_c([VAR2|R],t,I1,C,D,P,MIN_WIDTH).

min_width_plateau_c([VAR1,VAR2|R],t,I,C,D,P,MIN_WIDTH) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        min_width_plateau_c([VAR2|R],t,I1,C,D,P,MIN_WIDTH).

min_width_plateau_c([VAR1,VAR2|R],t,I,_C,_D,_P,MIN_WIDTH) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        min_width_plateau_c([VAR2|R],s,I1,1,1,1,MIN_WIDTH).

min_width_plateau_c([_VAR1,VAR2|R],t,I,C,_D,P,MIN_WIDTH) :-
        !,
        I1 is I+1,
        min_width_plateau_c([VAR2|R],r,I1,C,2,P,MIN_WIDTH).

min_width_plateau_c([VAR1,VAR2|R],r,I,C,D,P,MIN_WIDTH) :-
        VAR1=VAR2,
        !,
        I1 is I+1,
        D1 is D+1,
        min_width_plateau_c([VAR2|R],r,I1,C,D1,P,MIN_WIDTH).

min_width_plateau_c([VAR1,VAR2|R],r,I,C,_D,P,MIN_WIDTH) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        min_width_plateau_c([VAR2|R],t,I1,C,C,P,MIN_WIDTH).

min_width_plateau_c([_VAR1,VAR2|R],r,I,C,D,_P,MIN_WIDTH) :-
        !,
        I1 is I+1,
        C1 is min(C,D),
        min_width_plateau_c([VAR2|R],s,I1,C1,D,1,MIN_WIDTH).

min_width_plateau_c([_27772],_27766,_27767,C,D,P,MIN_WIDTH) :-
        MIN_WIDTH#=P*min(C,D).

min_width_plateau_a(FLAG,MIN_WIDTH,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        min_width_plateau_a_s(
```

```
            FLAG,
            MIN_WIDTH,
            VARIABLES,
            SIGNATURE).

min_width_plateau_a_s(FLAG,MIN_WIDTH,VARIABLES,SIGNATURE) :-
        check_type(dvar,MIN_WIDTH),
        length(VARIABLES,N),
        (   N=0 ->
            Cn=N,
            Dn=0,
            Pn=0
        ;   collection(VARIABLES,[dvar]),
            pair_index_signature(VARIABLES,1,INDICES),
            automaton(
                INDICES,
                _30112,
                SIGNATURE,
                [source(s),sink(s),sink(t),sink(r)],
                [arc(s,0,t),
                 arc(s,1,s),
                 arc(s,2,s),
                 arc(t,0,t),
                 arc(t,1,r,[C,2,P]),
                 arc(t,2,s,[1,1,1]),
                 arc(r,0,t,[C,C,P]),
                 arc(r,1,r,[C,D+1,P]),
                 arc(r,2,s,[min(C,D),D,1])],
                [C,D,P],
                [N,0,0],
                [Cn,Dn,Pn])
        ),
        MIN_WIDTH#=Pn*min(Cn,Dn)#<=>FLAG.
```

B.266 min_width_valley

◇ **META-DATA:**

```
ctr_date(min_width_valley,['20121202']).

ctr_origin(min_width_valley,'derived from %c',[valley]).

ctr_synonyms(min_width_valley,[min_base_valley]).

ctr_arguments(
    min_width_valley,
    ['MIN_WIDTH'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    min_width_valley,
    ['MIN_WIDTH'>=0,
     'MIN_WIDTH'=<size('VARIABLES')-2,
     required('VARIABLES',var)]).

ctr_example(
    min_width_valley,
    [min_width_valley(
         5,
         [[var-3],
          [var-3],
          [var-5],
          [var-5],
          [var-4],
          [var-2],
          [var-2],
          [var-3],
          [var-4],
          [var-6],
          [var-6],
          [var-5],
          [var-5],
          [var-5],
          [var-5],
          [var-5],
          [var-5],
          [var-6]]),
     min_width_valley(
         0,
         [[var-3],[var-8],[var-8],[var-5],[var-0],[var-0]]),
     min_width_valley(
```

```
        4,
        [[var-9],[var-8],[var-8],[var-0],[var-0],[var-2]])]).

ctr_typical(
    min_width_valley,
    ['MIN_WIDTH'>1,size('VARIABLES')>2]).

ctr_typical_model(min_width_valley,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    min_width_valley,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    min_width_valley,
    [checker(min_width_valley_c),
     automaton(min_width_valley_a),
     automaton_with_signature(min_width_valley_a_s)]).

ctr_pure_functional_dependency(min_width_valley,[]).

ctr_functional_dependency(min_width_valley,1,[2]).

ctr_sol(min_width_valley,2,0,2,9,[0-9]).

ctr_sol(min_width_valley,3,0,3,64,[0-50,1-14]).

ctr_sol(min_width_valley,4,0,4,625,[0-295,1-230,2-100]).

ctr_sol(
    min_width_valley,
    5,
    0,
    5,
    7776,
    [0-1792,1-3205,2-2100,3-679]).

ctr_sol(
    min_width_valley,
    6,
    0,
    6,
    117649,
    [0-11088,1-56637,2-28420,3-17024,4-4480]).

ctr_sol(
```

```
    min_width_valley,
    7,
    0,
    7,
    2097152,
    [0-69498,1-1174398,2-424928,3-268722,4-130452,5-29154]).

ctr_sol(
    min_width_valley,
    8,
    0,
    8,
    43046721,
    [0-439791,
     1-26327058,
     2-9363060,
     3-3413256,
     4-2345982,
     5-968946,
     6-188628]).

min_width_valley_c(0,[]) :-
        !.

min_width_valley_c(MIN_WIDTH,VARIABLES) :-
        check_type(dvar,MIN_WIDTH),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,N),
        min_width_valley_c(VARS,s,1,0,N,0,N,MIN_WIDTH).

min_width_valley_c([VAR1,VAR2|R],s,I,C,W,F,N,MIN_WIDTH) :-
        VAR1=<VAR2,
        !,
        I1 is I+1,
        min_width_valley_c([VAR2|R],s,I1,C,W,F,N,MIN_WIDTH).

min_width_valley_c([VAR1,VAR2|R],s,I,C,W,_46817,N,MIN_WIDTH) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        min_width_valley_c([VAR2|R],j,I1,C,W,I,N,MIN_WIDTH).

min_width_valley_c([VAR1,VAR2|R],j,I,C,W,F,N,MIN_WIDTH) :-
        VAR1>=VAR2,
        !,
```

```
        I1 is I+1,
        min_width_valley_c([VAR2|R],j,I1,C,W,F,N,MIN_WIDTH).

min_width_valley_c([VAR1,VAR2|R],j,I,_46815,W,F,N,MIN_WIDTH) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        C1 is I-F,
        min_width_valley_c([VAR2|R],k,I1,C1,W,F,N,MIN_WIDTH).

min_width_valley_c([VAR1,VAR2|R],k,I,C,W,F,N,MIN_WIDTH) :-
        VAR1=VAR2,
        !,
        I1 is I+1,
        min_width_valley_c([VAR2|R],k,I1,C,W,F,N,MIN_WIDTH).

min_width_valley_c([VAR1,VAR2|R],k,I,_46815,W,F,N,MIN_WIDTH) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        C1 is I-F,
        min_width_valley_c([VAR2|R],k,I1,C1,W,F,N,MIN_WIDTH).

min_width_valley_c([VAR1,VAR2|R],k,I,C,W,_46817,N,MIN_WIDTH) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        W1 is min(W,C),
        min_width_valley_c([VAR2|R],j,I1,C,W1,I,N,MIN_WIDTH).

min_width_valley_c(
    [_46820],
    _47220,
    _47267,
    C,
    W,
    _47404,
    _47451,
    MIN_WIDTH) :-
        MIN_WIDTH#=min(W,C).

min_width_valley_counters_check(VARS,N,[0|COUNTERS]) :-
        min_width_valley_counters_check(
            VARS,
            s,
            1,
```

```
            0,
            N,
            0,
            N,
            COUNTERS).

min_width_valley_counters_check(
    [VAR1,VAR2|R],
    s,
    I,
    C,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1=<VAR2,
        !,
        I1 is I+1,
        MIN_WIDTH#=min(W,C),
        min_width_valley_counters_check(
            [VAR2|R],
            s,
            I1,
            C,
            W,
            F,
            N,
            REST).

min_width_valley_counters_check(
    [VAR1,VAR2|R],
    s,
    I,
    C,
    W,
    _47492,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        MIN_WIDTH#=min(W,C),
        min_width_valley_counters_check(
            [VAR2|R],
            j,
            I1,
```

```
            C,
            W,
            I,
            N,
            REST).

min_width_valley_counters_check(
    [VAR1,VAR2|R],
    j,
    I,
    C,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1>=VAR2,
        !,
        I1 is I+1,
        MIN_WIDTH#=min(W,C),
        min_width_valley_counters_check(
            [VAR2|R],
            j,
            I1,
            C,
            W,
            F,
            N,
            REST).

min_width_valley_counters_check(
    [VAR1,VAR2|R],
    j,
    I,
    _47413,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        C1 is I-F,
        MIN_WIDTH#=min(W,C1),
        min_width_valley_counters_check(
            [VAR2|R],
            k,
```

```
            I1,
            C1,
            W,
            F,
            N,
            REST).

min_width_valley_counters_check(
    [VAR1,VAR2|R],
    k,
    I,
    C,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1=VAR2,
        !,
        I1 is I+1,
        MIN_WIDTH#=min(W,C),
        min_width_valley_counters_check(
            [VAR2|R],
            k,
            I1,
            C,
            W,
            F,
            N,
            REST).

min_width_valley_counters_check(
    [VAR1,VAR2|R],
    k,
    I,
    _47413,
    W,
    F,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1<VAR2,
        !,
        I1 is I+1,
        C1 is I-F,
        MIN_WIDTH#=min(W,C1),
        min_width_valley_counters_check(
            [VAR2|R],
```

```
            k,
            I1,
            C1,
            W,
            F,
            N,
            REST).

min_width_valley_counters_check(
    [VAR1,VAR2|R],
    k,
    I,
    C,
    W,
    _47503,
    N,
    [MIN_WIDTH|REST]) :-
        VAR1>VAR2,
        !,
        I1 is I+1,
        W1 is min(W,C),
        MIN_WIDTH#=min(W1,C),
        min_width_valley_counters_check(
            [VAR2|R],
            j,
            I1,
            C,
            W1,
            I,
            N,
            REST).

min_width_valley_counters_check(
    [_46817],
    _47193,
    _47240,
    _47287,
    _47334,
    _47381,
    _47428,
    []).

ctr_automaton_signature(
    min_width_valley,
    min_width_valley_a,
    pair_signature(2,signature)).
```

```
min_width_valley_a(FLAG,MIN_WIDTH,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        min_width_valley_a_s(
            FLAG,
            MIN_WIDTH,
            VARIABLES,
            SIGNATURE).

min_width_valley_a_s(FLAG,MIN_WIDTH,VARIABLES,SIGNATURE) :-
        check_type(dvar,MIN_WIDTH),
        length(VARIABLES,N),
        (   N=0 ->
            Wn=0,
            Cn=0
        ;   collection(VARIABLES,[dvar]),
            pair_index_signature(VARIABLES,1,INDICES),
            automaton(
                INDICES,
                I,
                SIGNATURE,
                [source(s),sink(s),sink(j),sink(k)],
                [arc(s,0,s),
                 arc(s,1,s),
                 arc(s,2,j,[C,W,I]),
                 arc(j,0,k,[I-F,W,F]),
                 arc(j,1,j),
                 arc(j,2,j),
                 arc(k,0,k,[I-F,W,F]),
                 arc(k,1,k),
                 arc(k,2,j,[C,min(W,C),I])],
                [C,W,F],
                [0,N,0],
                [Cn,Wn,_47020])
        ),
        MIN_WIDTH#=min(Wn,Cn)#<=>FLAG.
```

B.267 minimum

◇ **META-DATA:**

```
ctr_date(
    minimum,
    [20000128,
     20030820,
     20040530,
     20041230,
     20060811,
     20090416]).

ctr_origin(minimum,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(minimum,[min]).

ctr_arguments(
    minimum,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    minimum,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    minimum,
    [minimum(2,[[var-3],[var-2],[var-7],[var-2],[var-6]]),
     minimum(7,[[var-8],[var-8],[var-7],[var-8],[var-7]])]).

ctr_typical(
    minimum,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_typical_model(minimum,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    minimum,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     translate(['MIN','VARIABLES'^var])]).

ctr_graph(
    minimum,
    ['VARIABLES'],
    2,
```

```
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^key=variables2^key#\/
     variables1^var<variables2^var],
    ['ORDER'(0,'MAXINT',var)='MIN'],
    []).

ctr_eval(
    minimum,
    [builtin(minimum_b),
     automaton(minimum_a),
     automaton(minimum_ca)]).

ctr_pure_functional_dependency(minimum,[]).

ctr_functional_dependency(minimum,1,[2]).

ctr_aggregate(minimum,[],[min,union]).

ctr_cond_imply(
    minimum,
    deepest_valley,
    [first('VARIABLES'^var)>'MIN',last('VARIABLES'^var)>'MIN'],
    [],
    id).

ctr_sol(minimum,2,0,2,9,[0-5,1-3,2-1]).

ctr_sol(minimum,3,0,3,64,[0-37,1-19,2-7,3-1]).

ctr_sol(minimum,4,0,4,625,[0-369,1-175,2-65,3-15,4-1]).

ctr_sol(
    minimum,
    5,
    0,
    5,
    7776,
    [0-4651,1-2101,2-781,3-211,4-31,5-1]).

ctr_sol(
    minimum,
    6,
    0,
    6,
    117649,
    [0-70993,1-31031,2-11529,3-3367,4-665,5-63,6-1]).
```

```
ctr_sol(
    minimum,
    7,
    0,
    7,
    2097152,
    [0-1273609,
     1-543607,
     2-201811,
     3-61741,
     4-14197,
     5-2059,
     6-127,
     7-1]).

ctr_sol(
    minimum,
    8,
    0,
    8,
    43046721,
    [0-26269505,
     1-11012415,
     2-4085185,
     3-1288991,
     4-325089,
     5-58975,
     6-6305,
     7-255,
     8-1]).

minimum_b(MIN,VARIABLES) :-
        check_type(dvar,MIN),
        collection(VARIABLES,[dvar]),
        VARIABLES=[_76212|_76213],
        get_attr1(VARIABLES,VARS),
        minimum(MIN,VARS).

minimum_a(FLAG,MIN,VARIABLES) :-
        check_type(dvar,MIN),
        collection(VARIABLES,[dvar]),
        VARIABLES=[_76215|_76216],
        minimum_signature(VARIABLES,SIGNATURE,MIN),
        AUTOMATON=
        automaton(
```

```
            SIGNATURE,
            _77766,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,1,t),arc(t,0,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2],AUTOMATON).

minimum_signature([],[],_76185).

minimum_signature([[var-VAR]|VARs],[S|Ss],MIN) :-
        S in 0..2,
        MIN#<VAR#<=>S#=0,
        MIN#=VAR#<=>S#=1,
        MIN#>VAR#<=>S#=2,
        minimum_signature(VARs,Ss,MIN).

minimum_ca(FLAG,MIN,VARIABLES) :-
        check_type(dvar,MIN),
        collection(VARIABLES,[dvar]),
        maximum_signature1(VARIABLES,VARS,Zeros),
        VARS=[VAR1|_76227],
        automaton(
            VARS,
            VARi,
            Zeros,
            [source(s),sink(s)],
            [arc(s,0,s,[min(C,VARi)])],
            [C],
            [VAR1],
            [CC]),
        CC#=MIN#<=>FLAG.

minimum_signature1([],[],[]).

minimum_signature1([[var-VAR]|VARs],[VAR|R],[0|S]) :-
        minimum_signature1(VARs,R,S).
```

B.268 minimum_except_0

◇ **META-DATA:**

```
ctr_date(
    minimum_except_0,
    ['20030820','20040530','20041230','20060812','20090101']).

ctr_origin(minimum_except_0,'Derived from %c.',[minimum]).

ctr_arguments(
    minimum_except_0,
    ['MIN'-dvar,
     'VARIABLES'-collection(var-dvar),
     'DEFAULT'-int]).

ctr_restrictions(
    minimum_except_0,
    ['MIN'>0,
     'MIN'=<'DEFAULT',
     size('VARIABLES')>0,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<'DEFAULT',
     'DEFAULT'>0]).

ctr_example(
    minimum_except_0,
    [minimum_except_0(
        3,
        [[var-3],[var-7],[var-6],[var-7],[var-4],[var-7]],
        1000000),
     minimum_except_0(
        2,
        [[var-3],[var-2],[var-0],[var-7],[var-2],[var-6]],
        1000000),
     minimum_except_0(
        1000000,
        [[var-0],[var-0],[var-0],[var-0],[var-0],[var-0]],
        1000000)]).

ctr_typical(
    minimum_except_0,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     atleast(1,'VARIABLES',0)]).
```

```
ctr_typical_model(
    minimum_except_0,
    [nval('VARIABLES'^var)>2,atleast(2,'VARIABLES',0)]).

ctr_exchangeable(
    minimum_except_0,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in)]).

ctr_graph(
    minimum_except_0,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=\=0,
     variables2^var=\=0,
     variables1^key=variables2^key#\/
     variables1^var<variables2^var],
    ['ORDER'(0,'DEFAULT',var)='MIN'],
    []).

ctr_eval(
    minimum_except_0,
    [checker(minimum_except_0_c),
     reformulation(minimum_except_0_r),
     automaton(minimum_except_0_a)]).

ctr_pure_functional_dependency(minimum_except_0,[]).

ctr_functional_dependency(minimum_except_0,1,[2,3]).

ctr_cond_imply(
    minimum_except_0,
    atmost,
    [maxval('VARIABLES'^var)<'DEFAULT'],
    [],
    id).

minimum_except_0_c(MIN,VARIABLES,DEFAULT) :-
        check_type(int_gteq(1),DEFAULT),
        check_type(dvar(1,DEFAULT),MIN),
        collection(VARIABLES,[int(0,DEFAULT)]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
```

```
        minimum_except_0_c(VARS,1,DEFAULT,DEFAULT,MIN).

minimum_except_0_c([V|R],AllZero,Min,DEFAULT,RESULT) :-
        !,
        V>=0,
        V=<DEFAULT,
        (   V>0 ->
            NextAllZero=0,
            NextMin is min(Min,V)
        ;   NextAllZero=AllZero,
            NextMin=Min
        ),
        minimum_except_0_c(
            R,
            NextAllZero,
            NextMin,
            DEFAULT,
            RESULT).

minimum_except_0_c([],1,_49217,RESULT,RESULT) :-
        !.

minimum_except_0_c([],0,RESULT,_49215,RESULT).

minimum_except_0_r(MIN,VARIABLES,DEFAULT) :-
        check_type(int_gteq(1),DEFAULT),
        check_type(dvar(1,DEFAULT),MIN),
        collection(VARIABLES,[dvar(0,DEFAULT)]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        minimum_except_01(VARS,ALLZEROS),
        call(ALLZEROS#=>MIN#=DEFAULT),
        append([0],VARS,VARS0),
        N1 is N+1,
        length(RANKS,N1),
        domain(RANKS,0,N),
        min_n1(VARS0,RANKS,MIN,1).

minimum_except_01([],1).

minimum_except_01([V|R],V#=0#/\S) :-
        minimum_except_01(R,S).

minimum_except_0_a(FLAG,MIN,VARIABLES,DEFAULT) :-
        check_type(int_gteq(1),DEFAULT),
```

```
        check_type(dvar(1,DEFAULT),MIN),
        collection(VARIABLES,[dvar(0,DEFAULT)]),
        length(VARIABLES,N),
        N>0,
        minimum_except_0_signature(
            VARIABLES,
            SIGNATURE,
            MIN,
            DEFAULT),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _51643,
            SIGNATURE,
            [source(s),sink(j),sink(k)],
            [arc(s,0,s),
             arc(s,3,s),
             arc(s,2,j),
             arc(s,1,k),
             arc(j,0,j),
             arc(j,1,j),
             arc(j,2,j),
             arc(j,3,j),
             arc(k,1,k)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2,3,4],AUTOMATON).

minimum_except_0_signature([],[],_49214,_49215).

minimum_except_0_signature([[var-VAR]|VARs],[S|Ss],MIN,DEFAULT) :-
        S in 0..4,
        VAR#=0#/\MIN#\=DEFAULT#<=>S#=0,
        VAR#=0#/\MIN#=DEFAULT#<=>S#=1,
        VAR#\=0#/\MIN#=VAR#<=>S#=2,
        VAR#\=0#/\MIN#<VAR#<=>S#=3,
        VAR#\=0#/\MIN#>VAR#<=>S#=4,
        minimum_except_0_signature(VARs,Ss,MIN,DEFAULT).
```

B.269 minimum_greater_than

◇ META-DATA:

```
ctr_date(minimum_greater_than,['20030820','20060812']).

ctr_origin(minimum_greater_than,'N.~Beldiceanu',[]).

ctr_arguments(
    minimum_greater_than,
    ['VAR1'-dvar,'VAR2'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    minimum_greater_than,
    ['VAR1'>'VAR2',
     size('VARIABLES')>0,
     required('VARIABLES',var)]).

ctr_example(
    minimum_greater_than,
    minimum_greater_than(
        5,
        3,
        [[var-8],[var-5],[var-3],[var-8]])).

ctr_typical(
    minimum_greater_than,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_typical_model(
    minimum_greater_than,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(minimum_greater_than,[items('VARIABLES',all)]).

ctr_derived_collections(
    minimum_greater_than,
    [col('ITEM'-collection(var-dvar),[item(var-'VAR2')])]).

ctr_graph(
    minimum_greater_than,
    ['ITEM','VARIABLES'],
    2,
    ['PRODUCT'>>collection(item,variables)],
    [item^var<variables^var],
    ['NARC'>0],
```

```
    [],
    ['SUCC'>>[source,variables]],
    [minimum('VAR1',variables)]).

ctr_eval(
    minimum_greater_than,
    [reformulation(minimum_greater_than_r),
     automaton(minimum_greater_than_a)]).

ctr_aggregate(minimum_greater_than,[],[min,id,union]).

minimum_greater_than_r(VAR1,VAR2,VARIABLES) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        maximum(MAX,VARS),
        VAR1#>VAR2,
        VAR1#=<MAX,
        minimum_greater_than1(VARS,VAR2,MAX,UARS),
        minimum(VAR1,UARS).

minimum_greater_than1([],_42747,_42748,[]).

minimum_greater_than1([V|R],VAR2,MAX,[U|S]) :-
        fd_min(V,Min),
        fd_max(MAX,Max),
        U in Min..Max,
        V#=<VAR2#=>U#=MAX,
        V#>VAR2#=>U#=V,
        minimum_greater_than1(R,VAR2,MAX,S).

minimum_greater_than_a(FLAG,VAR1,VAR2,VARIABLES) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        VAR1#>VAR2,
        minimum_greater_than_signature(
            VARIABLES,
            SIGNATURE,
            VAR1,
            VAR2),
```

```
        AUTOMATON=
        automaton(
            SIGNATURE,
            _45335,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),
             arc(s,1,s),
             arc(s,2,s),
             arc(s,5,s),
             arc(s,4,t),
             arc(t,0,t),
             arc(t,1,t),
             arc(t,2,t),
             arc(t,4,t),
             arc(t,5,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2,3,4,5],AUTOMATON).

minimum_greater_than_signature([],[],_42748,_42749).

minimum_greater_than_signature(
    [[var-VAR]|VARs],
    [S|Ss],
    VAR1,
    VAR2) :-
        S in 0..5,
        VAR#<VAR1#/\VAR#=<VAR2#<=>S#=0,
        VAR#=VAR1#/\VAR#=<VAR2#<=>S#=1,
        VAR#>VAR1#/\VAR#=<VAR2#<=>S#=2,
        VAR#<VAR1#/\VAR#>VAR2#<=>S#=3,
        VAR#=VAR1#/\VAR#>VAR2#<=>S#=4,
        VAR#>VAR1#/\VAR#>VAR2#<=>S#=5,
        minimum_greater_than_signature(VARs,Ss,VAR1,VAR2).
```

B.270 minimum_modulo

◊ META-DATA:

```
ctr_date(
    minimum_modulo,
    ['20000128','20030820','20041230','20060812']).

ctr_origin(minimum_modulo,'Derived from %c.',[minimum]).

ctr_arguments(
    minimum_modulo,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar),'M'-int]).

ctr_restrictions(
    minimum_modulo,
    [size('VARIABLES')>0,'M'>0,required('VARIABLES',var)]).

ctr_example(
    minimum_modulo,
    [minimum_modulo(
         6,
         [[var-9],[var-1],[var-7],[var-6],[var-5]],
         3),
     minimum_modulo(
         9,
         [[var-9],[var-1],[var-7],[var-6],[var-5]],
         3)]).

ctr_typical(
    minimum_modulo,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     'M'>1,
     'M'<maxval('VARIABLES'^var)]).

ctr_exchangeable(minimum_modulo,[items('VARIABLES',all)]).

ctr_graph(
    minimum_modulo,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^key=variables2^key#\/
     variables1^var mod 'M'<variables2^var mod 'M'],
    ['ORDER'(0,'MAXINT',var)='MIN'],
```

```
    []).

ctr_pure_functional_dependency(minimum_modulo,[]).

ctr_functional_dependency(minimum_modulo,1,[2,3]).
```

B.271 minimum_weight_alldifferent

◊ META-DATA:

```
ctr_date(
    minimum_weight_alldifferent,
    ['20030820','20040530','20060812']).

ctr_origin(
    minimum_weight_alldifferent,
    \cite{FocacciLodiMilano99},
    []).

ctr_synonyms(
    minimum_weight_alldifferent,
    [minimum_weight_alldiff,
     minimum_weight_alldistinct,
     min_weight_alldiff,
     min_weight_alldifferent,
     min_weight_alldistinct]).

ctr_arguments(
    minimum_weight_alldifferent,
    ['VARIABLES'-collection(var-dvar),
     'MATRIX'-collection(i-int,j-int,c-int),
     'COST'-dvar]).

ctr_restrictions(
    minimum_weight_alldifferent,
    [size('VARIABLES')>0,
     required('VARIABLES',var),
     'VARIABLES'^var>=1,
     'VARIABLES'^var=<size('VARIABLES'),
     required('MATRIX',[i,j,c]),
     increasing_seq('MATRIX',[i,j]),
     'MATRIX'^i>=1,
     'MATRIX'^i=<size('VARIABLES'),
     'MATRIX'^j>=1,
     'MATRIX'^j=<size('VARIABLES'),
     size('MATRIX')=size('VARIABLES')*size('VARIABLES')]).

ctr_example(
    minimum_weight_alldifferent,
    minimum_weight_alldifferent(
        [[var-2],[var-3],[var-1],[var-4]],
        [[i-1,j-1,c-4],
```

```
         [i-1,j-2,c-1],
         [i-1,j-3,c-7],
         [i-1,j-4,c-0],
         [i-2,j-1,c-1],
         [i-2,j-2,c-0],
         [i-2,j-3,c-8],
         [i-2,j-4,c-2],
         [i-3,j-1,c-3],
         [i-3,j-2,c-2],
         [i-3,j-3,c-1],
         [i-3,j-4,c-6],
         [i-4,j-1,c-0],
         [i-4,j-2,c-0],
         [i-4,j-3,c-6],
         [i-4,j-4,c-5]],
        17)).

ctr_typical(
    minimum_weight_alldifferent,
    [size('VARIABLES')>1,range('MATRIX'^c)>1,'MATRIX'^c>0]).

ctr_graph(
    minimum_weight_alldifferent,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^key],
    ['NTREE'=0,
     'SUM_WEIGHT_ARC'(
         MATRIX@
         ((variables1^key-1)*size('VARIABLES')+
          variables1^var)^
         c)=
     COST],
    []).

ctr_functional_dependency(minimum_weight_alldifferent,3,[1,2]).
```

B.272 multi_global_contiguity

◇ **META-DATA:**

```
ctr_predefined(multi_global_contiguity).

ctr_date(multi_global_contiguity,['20120212']).

ctr_origin(
    multi_global_contiguity,
    Derived from %c.,
    [global_contiguity]).

ctr_synonyms(multi_global_contiguity,[multi_contiguity]).

ctr_arguments(
    multi_global_contiguity,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    multi_global_contiguity,
    [required('VARIABLES',var),'VARIABLES'^var>=0]).

ctr_example(
    multi_global_contiguity,
    multi_global_contiguity(
        [[var-0],
         [var-2],
         [var-2],
         [var-1],
         [var-1],
         [var-0],
         [var-0],
         [var-5]])).

ctr_typical(multi_global_contiguity,[size('VARIABLES')>3]).

ctr_typical_model(
    multi_global_contiguity,
    [nval('VARIABLES'^var)>2,atleast(2,'VARIABLES',0)]).

ctr_exchangeable(
    multi_global_contiguity,
    [items('VARIABLES',reverse)]).

ctr_eval(
```

```
    multi_global_contiguity,
    [checker(multi_global_contiguity_c)]).

ctr_contractible(multi_global_contiguity,[],'VARIABLES',any).

ctr_sol(multi_global_contiguity,2,0,2,9,-).

ctr_sol(multi_global_contiguity,3,0,3,55,-).

ctr_sol(multi_global_contiguity,4,0,4,413,-).

ctr_sol(multi_global_contiguity,5,0,5,3656,-).

ctr_sol(multi_global_contiguity,6,0,6,37147,-).

ctr_sol(multi_global_contiguity,7,0,7,425069,-).

ctr_sol(multi_global_contiguity,8,0,8,5400481,-).

multi_global_contiguity_c([]) :-
        !.

multi_global_contiguity_c(VARIABLES) :-
        collection(VARIABLES,[int_gteq(0)]),
        get_kattr1(VARIABLES,1,VARKEYS),
        sort(VARKEYS,SVARKEYS),
        multi_global_contiguity_c1(SVARKEYS).

multi_global_contiguity_c1([]) :-
        !.

multi_global_contiguity_c1([_31176]) :-
        !.

multi_global_contiguity_c1([0-_31180|R]) :-
        !,
        multi_global_contiguity_c1(R).

multi_global_contiguity_c1([I-P,I-Q|R]) :-
        !,
        Q is P+1,
        multi_global_contiguity_c1([I-Q|R]).

multi_global_contiguity_c1([_31176,J-Q|R]) :-
        multi_global_contiguity_c1([J-Q|R]).
```

B.273 multi_inter_distance

◇ META-DATA:

```
ctr_predefined(multi_inter_distance).

ctr_date(multi_inter_distance,['20110814']).

ctr_origin(multi_inter_distance,'\\cite{OuelletQuimper11}',[]).

ctr_synonyms(
    multi_inter_distance,
    [multi_all_min_distance,
     multi_all_min_dist,
     sliding_atmost,
     atmost_sliding]).

ctr_arguments(
    multi_inter_distance,
    ['VARIABLES'-collection(var-dvar),'LIMIT'-int,'DIST'-int]).

ctr_restrictions(
    multi_inter_distance,
    [required('VARIABLES',var),'LIMIT'>0,'DIST'>0]).

ctr_example(
    multi_inter_distance,
    multi_inter_distance(
        [[var-4],[var-0],[var-9],[var-4],[var-7]],
        2,
        3)).

ctr_typical(
    multi_inter_distance,
    ['LIMIT'>1,
     'LIMIT'<size('VARIABLES'),
     'DIST'>1,
     'DIST'<range('VARIABLES'^var)]).

ctr_exchangeable(
    multi_inter_distance,
    [items('VARIABLES',all),
     translate(['VARIABLES'^var]),
     vals(['LIMIT'],int,<,dontcare,dontcare),
     vals(['MINDIST'],int(>=(1)),>,dontcare,dontcare)]).
```

```
ctr_eval(
    multi_inter_distance,
    [reformulation(multi_inter_distance_r)]).

ctr_contractible(multi_inter_distance,[],'VARIABLES',any).

multi_inter_distance_r([],LIMIT,DIST) :-
        !,
        integer(LIMIT),
        integer(DIST),
        LIMIT>0,
        DIST>0.

multi_inter_distance_r(VARIABLES,LIMIT,DIST) :-
        collection(VARIABLES,[dvar]),
        integer(LIMIT),
        integer(DIST),
        LIMIT>0,
        DIST>0,
        get_attr1(VARIABLES,ORIGINS),
        length(VARIABLES,N),
        length(DURATIONS,N),
        length(ENDS,N),
        length(HEIGHTS,N),
        domain(DURATIONS,DIST,DIST),
        domain(HEIGHTS,1,1),
        ori_dur_end(ORIGINS,DURATIONS,ENDS),
        gen_cum_tasks(ORIGINS,DURATIONS,ENDS,HEIGHTS,1,Tasks),
        cumulative(Tasks,[limit(LIMIT)]).
```

B.274 multiple

◊ META-DATA:

```
ctr_predefined(multiple).

ctr_date(multiple,['20120501']).

ctr_origin(multiple,'Arithmetic.',[]).

ctr_arguments(multiple,['X'-dvar,'Y'-dvar,'C'-int]).

ctr_restrictions(multiple,['X'=\=0,'Y'=\=0,'C'>0]).

ctr_example(multiple,multiple(8,-2,4)).

ctr_typical(multiple,['C'>1]).

ctr_eval(multiple,[checker(multiple_c),builtin(multiple_b)]).

ctr_functional_dependency(multiple,3,[1,2]).

multiple_c(X,Y,C) :-
        check_type(int,X),
        X=\=0,
        check_type(int,Y),
        Y=\=0,
        check_type(dvar,C),
        AX is abs(X),
        AY is abs(Y),
        MAX is max(AX,AY),
        MIN is min(AX,AY),
        DIV is MAX//MIN,
        C#=DIV,
        MAX=:=C*MIN.

multiple_b(X,Y,C) :-
        check_type(dvar,X),
        X#\=0,
        check_type(dvar,Y),
        Y#\=0,
        check_type(int,C),
        max(abs(X),abs(Y))#=C*min(abs(X),abs(Y)).
```

B.275 nand

◇ **META-DATA:**

```
ctr_date(nand,['20051226']).

ctr_origin(nand,'Logic',[]).

ctr_synonyms(nand,[clause]).

ctr_arguments(
    nand,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nand,
    ['VAR'>=0,
     'VAR'=<1,
     size('VARIABLES')>=2,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    nand,
    [nand(1,[[var-0],[var-0]]),
     nand(1,[[var-0],[var-1]]),
     nand(1,[[var-1],[var-0]]),
     nand(0,[[var-1],[var-1]]),
     nand(1,[[var-1],[var-0],[var-1]])]).

ctr_exchangeable(nand,[items('VARIABLES',all)]).

ctr_eval(nand,[automaton(nand_a)]).

ctr_pure_functional_dependency(nand,[]).

ctr_functional_dependency(nand,1,[2]).

ctr_contractible(nand,['VAR'=0],'VARIABLES',any).

ctr_extensible(nand,['VAR'=1],'VARIABLES',any).

ctr_aggregate(nand,[],[#\/,union]).

ctr_cond_imply(
```

```
    nand,
    some_equal,
    [size('VARIABLES')>2],
    [],
    ['VARIABLES']).

ctr_sol(nand,2,0,2,4,[0-1,1-3]).

ctr_sol(nand,3,0,3,8,[0-1,1-7]).

ctr_sol(nand,4,0,4,16,[0-1,1-15]).

ctr_sol(nand,5,0,5,32,[0-1,1-31]).

ctr_sol(nand,6,0,6,64,[0-1,1-63]).

ctr_sol(nand,7,0,7,128,[0-1,1-127]).

ctr_sol(nand,8,0,8,256,[0-1,1-255]).

nand_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,L),
        L>1,
        get_attr1(VARIABLES,LIST),
        append([VAR],LIST,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _47385,
            LIST_VARIABLES,
            [source(s),sink(j),sink(k)],
            [arc(s,1,i),
             arc(s,0,j),
             arc(i,0,k),
             arc(i,1,i),
             arc(k,0,k),
             arc(k,1,k),
             arc(j,1,j)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.276 nclass

◇ **META-DATA:**

```
ctr_date(nclass,['20000128','20030820','20060812']).

ctr_origin(nclass,'Derived from %c.',[nvalue]).

ctr_types(nclass,['VALUES'-collection(val-int)]).

ctr_arguments(
    nclass,
    ['NCLASS'-dvar,
     'VARIABLES'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    nclass,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     'NCLASS'>=0,
     'NCLASS'=<min(size('VARIABLES'),size('PARTITIONS')),
     'NCLASS'=<range('VARIABLES'^var),
     required('VARIABLES',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    nclass,
    nclass(
        2,
        [[var-3],[var-2],[var-7],[var-2],[var-6]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(
    nclass,
    ['NCLASS'>1,
     'NCLASS'<size('VARIABLES'),
     'NCLASS'<range('VARIABLES'^var),
     size('VARIABLES')>size('PARTITIONS')]).

ctr_exchangeable(
    nclass,
```

```
    [items('VARIABLES',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['VARIABLES'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES'^var,'PARTITIONS'^p^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    nclass,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    ['NSCC'='NCLASS'],
    []).

ctr_pure_functional_dependency(nclass,[]).

ctr_functional_dependency(nclass,1,[2,3]).

ctr_extensible(
    nclass,
    ['NCLASS'=size('PARTITIONS')],
    VARIABLES,
    any).
```

B.277 neq

◊ **META-DATA:**

```
ctr_predefined(neq).

ctr_date(neq,['20070821']).

ctr_origin(neq,'Arithmetic.',[]).

ctr_synonyms(neq,[rel]).

ctr_arguments(neq,['VAR1'-dvar,'VAR2'-dvar]).

ctr_example(neq,neq(1,8)).

ctr_exchangeable(
    neq,
    [args([['VAR1','VAR2']]),
     vals(['VAR1','VAR2'],int,=\=,all,dontcare)]).

ctr_eval(neq,[builtin(neq_b)]).

neq_b(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#\=VAR2.
```

B.278 neq_cst

◇ **META-DATA:**

```
ctr_predefined(neq_cst).

ctr_date(neq_cst,['20090923']).

ctr_origin(neq_cst,'Arithmetic.',[]).

ctr_arguments(neq_cst,['VAR1'-dvar,'VAR2'-dvar,'CST2'-int]).

ctr_example(neq_cst,neq_cst(8,2,7)).

ctr_typical(neq_cst,['CST2'=\=0,'VAR1'=\='VAR2'+'CST2']).

ctr_exchangeable(
    neq_cst,
    [args([['VAR1'],['VAR2','CST2']]),
     translate(['VAR1','VAR2']),
     translate(['VAR1','CST2'])]).

ctr_eval(neq_cst,[builtin(neq_cst_b)]).

neq_cst_b(VAR1,VAR2,CST2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        check_type(int,CST2),
        VAR1#\=VAR2+CST2.
```

B.279 nequivalence

◇ **META-DATA:**

```
ctr_date(nequivalence,['20000128','20030820','20060812']).

ctr_origin(nequivalence,'Derived from %c.',[nvalue]).

ctr_arguments(
    nequivalence,
    ['NEQUIV'-dvar,'M'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nequivalence,
    [required('VARIABLES',var),
     'NEQUIV'>=min(1,size('VARIABLES')),
     'NEQUIV'=<min('M',size('VARIABLES')),
     'NEQUIV'=<range('VARIABLES'^var),
     'M'>0]).

ctr_example(
    nequivalence,
    nequivalence(
        2,
        3,
        [[var-3],
         [var-2],
         [var-5],
         [var-6],
         [var-15],
         [var-3],
         [var-3]])).

ctr_typical(
    nequivalence,
    ['NEQUIV'>1,
     'NEQUIV'<size('VARIABLES'),
     'NEQUIV'<range('VARIABLES'^var),
     'M'>1,
     'M'<maxval('VARIABLES'^var)]).

ctr_exchangeable(
    nequivalence,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],mod('M'),=,dontcare,dontcare)]).
```

```
ctr_graph(
    nequivalence,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['NSCC'='NEQUIV'],
    []).

ctr_pure_functional_dependency(nequivalence,[]).

ctr_functional_dependency(nequivalence,1,[2,3]).

ctr_contractible(
    nequivalence,
    ['NEQUIV'=1,size('VARIABLES')>0],
    VARIABLES,
    any).

ctr_contractible(
    nequivalence,
    ['NEQUIV'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_extensible(nequivalence,['NEQUIV'='M'],'VARIABLES',any).
```

B.280 next_element

◇ **META-DATA:**

```
ctr_date(next_element,['20030820','20040530','20060812']).

ctr_origin(next_element,'N.~Beldiceanu',[]).

ctr_arguments(
    next_element,
    ['THRESHOLD'-dvar,
     'INDEX'-dvar,
     'TABLE'-collection(index-int,value-dvar),
     'VAL'-dvar]).

ctr_restrictions(
    next_element,
    ['INDEX'>=1,
     'INDEX'=<size('TABLE'),
     'THRESHOLD'<'INDEX',
     required('TABLE',[index,value]),
     size('TABLE')>0,
     'TABLE'^index>=1,
     'TABLE'^index=<size('TABLE'),
     distinct('TABLE',index)]).

ctr_example(
    next_element,
    next_element(
        2,
        3,
        [[index-1,value-1],
         [index-2,value-8],
         [index-3,value-9],
         [index-4,value-5],
         [index-5,value-9]],
        9)).

ctr_typical(
    next_element,
    [size('TABLE')>1,range('TABLE'^value)>1]).

ctr_derived_collections(
    next_element,
    [col('ITEM'-collection(index-dvar,value-dvar),
         [item(index-'THRESHOLD',value-'VAL')])]).
```

```
ctr_graph(
    next_element,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index<table^index,item^value=table^value],
    ['NARC'>0],
    [],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TABLE'^index)])]],
    [minimum('INDEX',variables)]).

ctr_eval(next_element,[automaton(next_element_a)]).

next_element_a(FLAG,THRESHOLD,INDEX,TABLE,VAL) :-
        length(TABLE,N),
        N>0,
        check_type(dvar,THRESHOLD),
        check_type(dvar(1,N),INDEX),
        collection(TABLE,[int(1,N),dvar]),
        check_type(dvar,VAL),
        THRESHOLD#<INDEX,
        get_attr1(TABLE,INDEXES),
        all_different(INDEXES),
        next_element_signature(
            TABLE,
            SIGNATURE,
            THRESHOLD,
            INDEX,
            VAL),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _46195,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),
             arc(s,1,s),
             arc(s,2,s),
             arc(s,3,s),
             arc(s,4,s),
             arc(s,5,s),
```

```
             arc(s,7,s),
             arc(s,9,s),
             arc(s,10,s),
             arc(s,11,s),
             arc(s,8,t),
             arc(t,0,t),
             arc(t,1,t),
             arc(t,2,t),
             arc(t,3,t),
             arc(t,4,t),
             arc(t,5,t),
             arc(t,7,t),
             arc(t,8,t),
             arc(t,9,t),
             arc(t,10,t),
             arc(t,11,t)],
            [],
            [],
            []),
        automaton_bool(
            FLAG,
            [0,1,2,3,4,5,6,7,8,9,10,11],
            AUTOMATON).

next_element_signature([],[],_42693,_42694,_42695).

next_element_signature(
    [[index-I,value-V]|Ts],
    [S|Ss],
    THRESHOLD,
    INDEX,
    VAL) :-
        S in 0..11,
        I#=<THRESHOLD#/\I#<INDEX#/\V#=VAL#<=>S#=0,
        I#=<THRESHOLD#/\I#<INDEX#/\V#\=VAL#<=>S#=1,
        I#=<THRESHOLD#/\I#=INDEX#/\V#=VAL#<=>S#=2,
        I#=<THRESHOLD#/\I#=INDEX#/\V#\=VAL#<=>S#=3,
        I#=<THRESHOLD#/\I#>INDEX#/\V#=VAL#<=>S#=4,
        I#=<THRESHOLD#/\I#>INDEX#/\V#\=VAL#<=>S#=5,
        I#>THRESHOLD#/\I#<INDEX#/\V#=VAL#<=>S#=6,
        I#>THRESHOLD#/\I#<INDEX#/\V#\=VAL#<=>S#=7,
        I#>THRESHOLD#/\I#=INDEX#/\V#=VAL#<=>S#=8,
        I#>THRESHOLD#/\I#=INDEX#/\V#\=VAL#<=>S#=9,
        I#>THRESHOLD#/\I#>INDEX#/\V#=VAL#<=>S#=10,
        I#>THRESHOLD#/\I#>INDEX#/\V#\=VAL#<=>S#=11,
        next_element_signature(Ts,Ss,THRESHOLD,INDEX,VAL).
```

B.281 next_greater_element

◇ **META-DATA:**

```
ctr_date(
    next_greater_element,
    ['20030820','20040530','20060812']).

ctr_origin(next_greater_element,'M.~Carlsson',[]).

ctr_arguments(
    next_greater_element,
    ['VAR1'-dvar,'VAR2'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    next_greater_element,
    ['VAR1'<'VAR2',
     size('VARIABLES')>0,
     required('VARIABLES',var)]).

ctr_example(
    next_greater_element,
    next_greater_element(
        7,
        8,
        [[var-3],[var-5],[var-8],[var-9]])).

ctr_typical(
    next_greater_element,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_derived_collections(
    next_greater_element,
    [col('V'-collection(var-dvar),[item(var-'VAR1')])]).

ctr_graph(
    next_greater_element,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var<variables2^var],
    ['NARC'=size('VARIABLES')-1],
    []).

ctr_graph(
    next_greater_element,
```

```
    ['V','VARIABLES'],
    2,
    ['PRODUCT'>>collection(v,variables)],
    [v^var<variables^var],
    ['NARC'>0],
    [],
    ['SUCC'>>[source,variables]],
    [minimum('VAR2',variables)]).

ctr_eval(
    next_greater_element,
    [reformulation(next_greater_element_r)]).

next_greater_element_r(VAR1,VAR2,VARIABLES) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        maximum(MAX,VARS),
        VAR2#>VAR1,
        VAR2#=<MAX,
        next_greater_element1(VARS,VAR1,MAX,UARS),
        minimum(VAR2,UARS).

next_greater_element1([V],VAR1,MAX,[U]) :-
        !,
        fd_min(V,Min),
        fd_max(MAX,Max),
        U in Min..Max,
        V#=<VAR1#=>U#=MAX,
        V#>VAR1#=>U#=V.

next_greater_element1([V1,V2|R],VAR1,MAX,[U1|S]) :-
        V1#<V2,
        fd_min(V1,Min),
        fd_max(MAX,Max),
        U1 in Min..Max,
        V1#=<VAR1#=>U1#=MAX,
        V1#>VAR1#=>U1#=V1,
        next_greater_element1([V2|R],VAR1,MAX,S).
```

B.282 ninterval

◇ **META-DATA:**

```
ctr_date(ninterval,['20030820','20040530','20060812']).

ctr_origin(ninterval,'Derived from %c.',[nvalue]).

ctr_arguments(
    ninterval,
    ['NVAL'-dvar,
     'VARIABLES'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    ninterval,
    ['NVAL'>=min(1,size('VARIABLES')),
     'NVAL'=<size('VARIABLES'),
     required('VARIABLES',var),
     'SIZE_INTERVAL'>0]).

ctr_example(
    ninterval,
    ninterval(2,[[var-3],[var-1],[var-9],[var-1],[var-9]],4)).

ctr_typical(
    ninterval,
    ['NVAL'>1,
     'NVAL'<size('VARIABLES'),
     'SIZE_INTERVAL'>1,
     'SIZE_INTERVAL'<range('VARIABLES'^var),
     (nval('VARIABLES'^var)+'SIZE_INTERVAL'-1)/
     SIZE_INTERVAL<
     NVAL]).

ctr_exchangeable(
    ninterval,
    [items('VARIABLES',all),
     vals(
         ['VARIABLES'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
```

```
    ninterval,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var/'SIZE_INTERVAL'=
     variables2^var/'SIZE_INTERVAL'],
    ['NSCC'='NVAL'],
    []).

ctr_eval(ninterval,[checker(ninterval_c)]).

ctr_pure_functional_dependency(ninterval,[]).

ctr_functional_dependency(ninterval,1,[2,3]).

ctr_contractible(
    ninterval,
    ['NVAL'=1,size('VARIABLES')>0],
    VARIABLES,
    any).

ctr_contractible(
    ninterval,
    ['NVAL'=size('VARIABLES')],
    VARIABLES,
    any).

ninterval_c(NVAL,VARIABLES,SIZE_INTERVAL) :-
        collection(VARIABLES,[int]),
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
        get_attr1(VARIABLES,VARS),
        length(VARS,L),
        MIN_NVAL is min(1,L),
        check_type(dvar(MIN_NVAL,L),NVAL),
        (   L=0 ->
            NVAL#=0
        ;   gen_quotient_fix(VARS,SIZE_INTERVAL,QUOTIENT),
            sort(QUOTIENT,SORTED),
            length(SORTED,N),
            NVAL#=N
        ).
```

B.283 no_peak

◇ **META-DATA:**

```
ctr_date(no_peak,['20031101','20040530']).

ctr_origin(no_peak,'Derived from %c.',[peak]).

ctr_arguments(no_peak,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    no_peak,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    no_peak,
    no_peak([[var-1],[var-1],[var-4],[var-8],[var-8]])).

ctr_typical(
    no_peak,
    [size('VARIABLES')>3,range('VARIABLES'^var)>1]).

ctr_typical_model(no_peak,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    no_peak,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(no_peak,[checker(no_peak_c),automaton(no_peak_a)]).

ctr_contractible(no_peak,[],'VARIABLES',any).

ctr_sol(no_peak,2,0,2,9,-).

ctr_sol(no_peak,3,0,3,50,-).

ctr_sol(no_peak,4,0,4,295,-).

ctr_sol(no_peak,5,0,5,1792,-).

ctr_sol(no_peak,6,0,6,11088,-).

ctr_sol(no_peak,7,0,7,69498,-).

ctr_sol(no_peak,8,0,8,439791,-).
```

```
no_peak_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N>0,
        no_peak_c(VARIABLES,0).

no_peak_c([],_33096) :-
        !.

no_peak_c([_33097],_33096) :-
        !.

no_peak_c([[var-X],[var-Y]|R],0) :-
        X>=Y,
        !,
        no_peak_c([[var-Y]|R],0).

no_peak_c([_33097,[var-Y]|R],0) :-
        !,
        no_peak_c([[var-Y]|R],1).

no_peak_c([[var-X],[var-Y]|R],1) :-
        X=<Y,
        no_peak_c([[var-Y]|R],1).

no_peak_a(FLAG,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        no_peak_signature(VARIABLES,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _34704,
            SIGNATURE,
            [source(s),sink(t),sink(s)],
            [arc(s,1,s),
             arc(s,2,s),
             arc(s,0,t),
             arc(t,0,t),
             arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2],AUTOMATON).
```

```
no_peak_signature([],[]).

no_peak_signature([_33097],[]) :-
        !.

no_peak_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
        S in 0..2,
        VAR1#<VAR2#<=>S#=0,
        VAR1#=VAR2#<=>S#=1,
        VAR1#>VAR2#<=>S#=2,
        no_peak_signature([[var-VAR2]|VARs],Ss).
```

B.284 no_valley

◇ **META-DATA:**

```
ctr_date(no_valley,['20031101','20040530']).

ctr_origin(no_valley,'Derived from %c.',[valley]).

ctr_arguments(no_valley,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    no_valley,
    [size('VARIABLES')>0,required('VARIABLES',var)]).

ctr_example(
    no_valley,
    no_valley(
        [[var-1],[var-1],[var-4],[var-8],[var-8],[var-2]])).

ctr_typical(
    no_valley,
    [size('VARIABLES')>3,range('VARIABLES'^var)>1]).

ctr_typical_model(no_valley,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    no_valley,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    no_valley,
    [checker(no_valley_c),automaton(no_valley_a)]).

ctr_contractible(no_valley,[],'VARIABLES',any).

ctr_sol(no_valley,2,0,2,9,-).

ctr_sol(no_valley,3,0,3,50,-).

ctr_sol(no_valley,4,0,4,295,-).

ctr_sol(no_valley,5,0,5,1792,-).

ctr_sol(no_valley,6,0,6,11088,-).

ctr_sol(no_valley,7,0,7,69498,-).
```

```
ctr_sol(no_valley,8,0,8,439791,-).

no_valley_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        N>0,
        no_valley_c(VARIABLES,0).

no_valley_c([],_33411) :-
        !.

no_valley_c([_33412],_33411) :-
        !.

no_valley_c([[var-X],[var-Y]|R],0) :-
        X=<Y,
        !,
        no_valley_c([[var-Y]|R],0).

no_valley_c([_33412,[var-Y]|R],0) :-
        !,
        no_valley_c([[var-Y]|R],1).

no_valley_c([[var-X],[var-Y]|R],1) :-
        X>=Y,
        no_valley_c([[var-Y]|R],1).

no_valley_a(FLAG,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>0,
        no_valley_signature(VARIABLES,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _35019,
            SIGNATURE,
            [source(s),sink(t),sink(s)],
            [arc(s,0,s),
             arc(s,1,s),
             arc(s,2,t),
             arc(t,1,t),
             arc(t,2,t)],
            [],
            [],
```

```
            []),
        automaton_bool(FLAG,[0,1,2],AUTOMATON).

no_valley_signature([],[]).

no_valley_signature([_33412],[]) :-
        !.

no_valley_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
        S in 0..2,
        VAR1#<VAR2#<=>S#=0,
        VAR1#=VAR2#<=>S#=1,
        VAR1#>VAR2#<=>S#=2,
        no_valley_signature([[var-VAR2]|VARs],Ss).
```

B.285 non_overlap_sboxes

◇ **META-DATA:**

```
ctr_date(non_overlap_sboxes,['20070622','20090725']).

ctr_origin(
    non_overlap_sboxes,
    Geometry, derived from \cite{BeldiceanuCarlssonPoderSadekTruchet07},
    []).

ctr_synonyms(non_overlap_sboxes,[non_overlap,non_overlapping]).

ctr_types(
    non_overlap_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    non_overlap_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    non_overlap_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES')]).

ctr_example(
    non_overlap_sboxes,
    non_overlap_sboxes(
        2,
        {0,1},
        [[oid-1,sid-1,x-[[v-4],[v-1]]],
         [oid-2,sid-3,x-[[v-2],[v-2]]],
         [oid-3,sid-4,x-[[v-5],[v-4]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-1],[v-1]]],
         [sid-1,t-[[v-1],[v-0]],l-[[v-1],[v-3]]],
         [sid-1,t-[[v-0],[v-2]],l-[[v-1],[v-1]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-1]]],
         [sid-2,t-[[v-0],[v-1]],l-[[v-1],[v-1]]],
         [sid-2,t-[[v-2],[v-1]],l-[[v-1],[v-1]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-1],[v-2]]],
         [sid-4,t-[[v-0],[v-0]],l-[[v-1],[v-1]]]])).

ctr_typical(non_overlap_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    non_overlap_sboxes,
    [items('OBJECTS',all),
     items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all),
     vals(['SBOXES'^l^v],int(>=(1)),>,dontcare,dontcare)]).

ctr_eval(non_overlap_sboxes,[logic(non_overlap_sboxes_g)]).

ctr_logic(
    non_overlap_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (non_overlap_sboxes(Dims,O1,S1,O2,S2)--->
      exists(
          D,
          Dims,
          end(O1,S1,D)#=<origin(O2,S2,D)#\/
          end(O2,S2,D)#=<origin(O1,S1,D))),
     (non_overlap_objects(Dims,O1,O2)--->
      forall(
```

```
            S1,
            sboxes([O1^sid]),
            forall(
                S2,
                sboxes([O2^sid]),
                non_overlap_sboxes(Dims,O1,S1,O2,S2)))),
       (all_non_overlap(Dims,OIDS)--->
        forall(
            O1,
            objects(OIDS),
            forall(
                O2,
                objects(OIDS),
                O1^oid#<O2^oid#=>
                non_overlap_objects(Dims,O1,O2)))),
       all_non_overlap(DIMENSIONS,OIDS)]).

ctr_contractible(non_overlap_sboxes,[],'OBJECTS',suffix).

ctr_application(non_overlap_sboxes,[3]).

non_overlap_sboxes_g(K,_42412,[],_42414) :-
        !,
        check_type(int_gteq(1),K).

non_overlap_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        collection_increasing_seq(OBJECTS,[1]),
        geost1(OIDS,SIDS,XS,Objects),
        geost2(SIDES,TS,TL,Sboxes),
        geost_dims(1,K,DIMENSIONS),
        ctr_logic(non_overlap_sboxes,[DIMENSIONS,OIDS],Rules),
```

```
geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.286 nor

◇ **META-DATA:**

```
ctr_date(nor,['20051226']).

ctr_origin(nor,'Logic',[]).

ctr_synonyms(nor,[clause]).

ctr_arguments(
    nor,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nor,
    ['VAR'>=0,
     'VAR'=<1,
     size('VARIABLES')>=2,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    nor,
    [nor(1,[[var-0],[var-0]]),
     nor(0,[[var-0],[var-1]]),
     nor(0,[[var-1],[var-0]]),
     nor(0,[[var-1],[var-1]]),
     nor(0,[[var-1],[var-0],[var-1]])]).

ctr_exchangeable(nor,[items('VARIABLES',all)]).

ctr_eval(nor,[automaton(nor_a)]).

ctr_pure_functional_dependency(nor,[]).

ctr_functional_dependency(nor,1,[2]).

ctr_contractible(nor,['VAR'=1],'VARIABLES',any).

ctr_extensible(nor,['VAR'=0],'VARIABLES',any).

ctr_aggregate(nor,[],[#/\,union]).

ctr_cond_imply(
```

```
    nor,
    some_equal,
    [size('VARIABLES')>2],
    [],
    ['VARIABLES']).

ctr_sol(nor,2,0,2,4,[0-3,1-1]).

ctr_sol(nor,3,0,3,8,[0-7,1-1]).

ctr_sol(nor,4,0,4,16,[0-15,1-1]).

ctr_sol(nor,5,0,5,32,[0-31,1-1]).

ctr_sol(nor,6,0,6,64,[0-63,1-1]).

ctr_sol(nor,7,0,7,128,[0-127,1-1]).

ctr_sol(nor,8,0,8,256,[0-255,1-1]).

nor_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,L),
        L>1,
        get_attr1(VARIABLES,LIST),
        append([VAR],LIST,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _47809,
            LIST_VARIABLES,
            [source(s),sink(i),sink(k)],
            [arc(s,0,j),
             arc(s,1,i),
             arc(i,0,i),
             arc(j,0,j),
             arc(j,1,k),
             arc(k,0,k),
             arc(k,1,k)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.287 not_all_equal

◇ **META-DATA:**

```
ctr_date(
    not_all_equal,
    ['20030820','20040530','20040726','20060812','20100418']).

ctr_origin(not_all_equal,'\\index{CHIP|indexuse}CHIP',[]).

ctr_arguments(not_all_equal,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    not_all_equal,
    [required('VARIABLES',var),size('VARIABLES')>1]).

ctr_example(
    not_all_equal,
    not_all_equal([[var-3],[var-1],[var-3],[var-3],[var-3]])).

ctr_typical(
    not_all_equal,
    [size('VARIABLES')>2,nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    not_all_equal,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    not_all_equal,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSCC'>1],
    []).

ctr_eval(
    not_all_equal,
    [checker(not_all_equal_c),
     reformulation(not_all_equal_r),
     automaton(not_all_equal_a)]).

ctr_extensible(not_all_equal,[],'VARIABLES',any).
```

```
ctr_sol(not_all_equal,2,0,2,6,-).

ctr_sol(not_all_equal,3,0,3,60,-).

ctr_sol(not_all_equal,4,0,4,620,-).

ctr_sol(not_all_equal,5,0,5,7770,-).

ctr_sol(not_all_equal,6,0,6,117642,-).

ctr_sol(not_all_equal,7,0,7,2097144,-).

ctr_sol(not_all_equal,8,0,8,43046712,-).

not_all_equal_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        not_all_equal_c1(VARIABLES).

not_all_equal_c1([V,V|R]) :-
        !,
        not_all_equal_c1([V|R]).

not_all_equal_c1([_52698,_52700|_52701]).

not_all_equal_r(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>1,
        get_attr1(VARIABLES,VARS),
        NVAL in 2..N,
        nvalue(NVAL,VARS).

not_all_equal_a(FLAG,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N>1,
        not_all_equal_signature(VARIABLES,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _54265,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,1,s),arc(s,0,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
```

```
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

not_all_equal_signature([],[]).

not_all_equal_signature([_52702],[]) :-
        !.

not_all_equal_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss]) :-
        VAR1#=VAR2#<=>S,
        not_all_equal_signature([[var-VAR2]|VARs],Ss).
```

B.288 not_in

♢ META-DATA:

```
ctr_date(not_in,['20030820','20040530','20060812']).

ctr_origin(not_in,'Derived from %c.',[in]).

ctr_arguments(not_in,['VAR'-dvar,'VALUES'-collection(val-int)]).

ctr_restrictions(
    not_in,
    [required('VALUES',val),distinct('VALUES',val)]).

ctr_example(not_in,not_in(2,[[val-1],[val-3]])).

ctr_typical(not_in,[size('VALUES')>1]).

ctr_exchangeable(
    not_in,
    [items('VALUES',all),translate(['VAR','VALUES'^val])]).

ctr_derived_collections(
    not_in,
    [col('VARIABLES'-collection(var-dvar),[item(var-'VAR')])]).

ctr_graph(
    not_in,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['NARC'=0],
    []).

ctr_eval(not_in,[automaton(not_in_a)]).

ctr_contractible(not_in,[],'VALUES',any).

not_in_a(FLAG,VAR,VALUES) :-
        check_type(dvar,VAR),
        collection(VALUES,[int]),
        get_attr1(VALUES,VALS),
        all_different(VALS),
        not_in_signature(VALUES,SIGNATURE,VAR),
        AUTOMATON=
```

```
        automaton(
            SIGNATURE,
            _41459,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

not_in_signature([],[],_39815).

not_in_signature([[val-VAL]|VALs],[S|Ss],VAR) :-
        VAR#=VAL#<=>S,
        not_in_signature(VALs,Ss,VAR).
```

B.289 npair

◊ **META-DATA:**

```
ctr_date(npair,['20030820','20060812']).

ctr_origin(npair,'Derived from %c.',[nvalue]).

ctr_arguments(
    npair,
    ['NPAIRS'-dvar,'PAIRS'-collection(x-dvar,y-dvar)]).

ctr_restrictions(
    npair,
    ['NPAIRS'>=min(1,size('PAIRS')),
     'NPAIRS'=<size('PAIRS'),
     required('PAIRS',[x,y])]).

ctr_example(
    npair,
    npair(
        2,
        [[x-3,y-1],[x-1,y-5],[x-3,y-1],[x-3,y-1],[x-1,y-5]])).

ctr_typical(
    npair,
    ['NPAIRS'>1,
     'NPAIRS'<size('PAIRS'),
     size('PAIRS')>1,
     range('PAIRS'^x)>1,
     range('PAIRS'^y)>1]).

ctr_exchangeable(
    npair,
    [items('PAIRS',all),
     attrs_sync('PAIRS',[[x,y]]),
     vals(['NPAIRS'],int,=\=,all,dontcare)]).

ctr_graph(
    npair,
    ['PAIRS'],
    2,
    ['CLIQUE'>>collection(pairs1,pairs2)],
    [pairs1^x=pairs2^x,pairs1^y=pairs2^y],
    ['NSCC'='NPAIRS'],
    []).
```

```
ctr_pure_functional_dependency(npair,[]).

ctr_functional_dependency(npair,1,[2]).

ctr_contractible(
    npair,
    ['NPAIRS'=1,size('PAIRS')>0],
    PAIRS,
    any).

ctr_contractible(npair,['NPAIRS'=size('PAIRS')],'PAIRS',any).
```

B.290 nset_of_consecutive_values

◊ META-DATA:

```
ctr_date(
    nset_of_consecutive_values,
    ['20030820','20040530','20060812']).

ctr_origin(nset_of_consecutive_values,'N.~Beldiceanu',[]).

ctr_arguments(
    nset_of_consecutive_values,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nset_of_consecutive_values,
    ['N'>=1,'N'=<size('VARIABLES'),required('VARIABLES',var)]).

ctr_example(
    nset_of_consecutive_values,
    [nset_of_consecutive_values(
         2,
         [[var-3],
          [var-1],
          [var-7],
          [var-1],
          [var-1],
          [var-2],
          [var-8]]),
     nset_of_consecutive_values(
         7,
         [[var-3],
          [var-1],
          [var-5],
          [var-7],
          [var-9],
          [var-11],
          [var-13]]),
     nset_of_consecutive_values(
         1,
         [[var-3],
          [var-3],
          [var-3],
          [var-3],
          [var-3],
          [var-3],
```

```
        [var-3]])]).

ctr_typical(
    nset_of_consecutive_values,
    ['N'>1,size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_exchangeable(
    nset_of_consecutive_values,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     translate(['VARIABLES'^var])]).

ctr_graph(
    nset_of_consecutive_values,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [abs(variables1^var-variables2^var)=<1],
    ['NSCC'='N'],
    []).

ctr_eval(
    nset_of_consecutive_values,
    [checker(nset_of_consecutive_values_c)]).

ctr_pure_functional_dependency(nset_of_consecutive_values,[]).

ctr_functional_dependency(nset_of_consecutive_values,1,[2]).

ctr_sol(nset_of_consecutive_values,2,0,2,9,[1-7,2-2]).

ctr_sol(nset_of_consecutive_values,3,0,3,64,[1-34,2-30]).

ctr_sol(
    nset_of_consecutive_values,
    4,
    0,
    4,
    625,
    [1-217,2-372,3-36]).

ctr_sol(
    nset_of_consecutive_values,
    5,
    0,
    5,
```

```
    7776,
    [1-1716,2-4740,3-1320]).

ctr_sol(
    nset_of_consecutive_values,
    6,
    0,
    6,
    117649,
    [1-16159,2-65010,3-34920,4-1560]).

ctr_sol(
    nset_of_consecutive_values,
    7,
    0,
    7,
    2097152,
    [1-176366,2-969066,3-842520,4-109200]).

ctr_sol(
    nset_of_consecutive_values,
    8,
    0,
    8,
    43046721,
    [1-2187637,2-15695624,3-19989900,4-5047560,5-126000]).

nset_of_consecutive_values_c(N,VARIABLES) :-
        length(VARIABLES,L),
        check_type(dvar(1,L),N),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        samsort(VARS,SVARS),
        SVARS=[V|R],
        nset_of_consecutive_values_c(R,V,1,NSet),
        N#=NSet.

nset_of_consecutive_values_c([V|R],Prev,NSet,Res) :-
        Diff is V-Prev,
        Diff=<1,
        !,
        nset_of_consecutive_values_c(R,V,NSet,Res).

nset_of_consecutive_values_c([V|R],_65584,NSet,Res) :-
        !,
        NSet1 is NSet+1,
```

```
        nset_of_consecutive_values_c(R,V,NSet1,Res).

nset_of_consecutive_values_c([],_65581,Res,Res).
```

B.291 number_digit

◊ **META-DATA:**

```
ctr_predefined(number_digit).

ctr_date(number_digit,['20141011']).

ctr_origin(number_digit,'Arithmetic.',[]).

ctr_arguments(
    number_digit,
    ['N'-dvar,'VARIABLES'-collection(var-dvar),'B'-int]).

ctr_restrictions(
    number_digit,
    ['N'>=0,
     size('VARIABLES')>=1,
     size('VARIABLES')=<9,
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<'B'-1,
     'B'>=2,
     'B'=<10]).

ctr_example(
    number_digit,
    number_digit(1234,[[var-1],[var-2],[var-3],[var-4]],10)).

ctr_eval(
    number_digit,
    [checker(number_digit_c),reformulation(number_digit_r)]).

ctr_pure_functional_dependency(number_digit,[]).

ctr_functional_dependency(number_digit,1,[2,3]).

number_digit_c(N,VARIABLES,B) :-
        check_type(int_gteq(0),N),
        check_type(int(2,10),B),
        B1 is B-1,
        collection(VARIABLES,[int(0,B1)]),
        length(VARIABLES,L),
        L>=0,
        L=<9,
        number_digit_c(VARIABLES,B,0,N).
```

```
number_digit_c([],_22417,N,N) :-
        !.

number_digit_c([[var-V]|R],B,C,N) :-
        C1 is B*C+V,
        number_digit_c(R,B,C1,N).

number_digit_r(N,VARIABLES,B) :-
        check_type(dvar_gteq(0),N),
        check_type(int(2,10),B),
        B1 is B-1,
        collection(VARIABLES,[dvar(0,B1)]),
        length(VARIABLES,L),
        L>=0,
        L=<9,
        number_digit_r(VARIABLES,B,0,N).

number_digit_r([],_22417,C,N) :-
        !,
        C#=N.

number_digit_r([[var-V]|R],B,C,N) :-
        C1#=B*C+V,
        number_digit_r(R,B,C1,N).
```

B.292 nvalue

◇ **META-DATA:**

```
ctr_date(
    nvalue,
    [20000128,
     20030820,
     20040530,
     20051001,
     20060812,
     20091105]).

ctr_origin(nvalue,'\\cite{PachetRoy99}',[]).

ctr_synonyms(nvalue,[cardinality_on_attributes_values,values]).

ctr_arguments(
    nvalue,
    ['NVAL'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nvalue,
    [required('VARIABLES',var),
     'NVAL'>=min(1,size('VARIABLES')),
     'NVAL'=<size('VARIABLES'),
     'NVAL'=<range('VARIABLES'^var)]).

ctr_example(
    nvalue,
    [nvalue(4,[[var-3],[var-1],[var-7],[var-1],[var-6]]),
     nvalue(1,[[var-6],[var-6],[var-6],[var-6],[var-6]]),
     nvalue(5,[[var-6],[var-3],[var-0],[var-2],[var-9]])]).

ctr_typical(
    nvalue,
    ['NVAL'>1,'NVAL'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_exchangeable(
    nvalue,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    nvalue,
    ['VARIABLES'],
```

```
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSCC'='NVAL'],
    ['EQUIVALENCE']).

ctr_eval(nvalue,[checker(nvalue_c),builtin(nvalue_b)]).

ctr_pure_functional_dependency(nvalue,[]).

ctr_functional_dependency(nvalue,1,[2]).

ctr_contractible(
    nvalue,
    ['NVAL'=1,size('VARIABLES')>0],
    VARIABLES,
    any).

ctr_contractible(
    nvalue,
    ['NVAL'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_cond_imply(
    nvalue,
    increasing_nvalue,
    [increasing('VARIABLES')],
    [],
    id).

ctr_sol(nvalue,2,0,2,9,[1-3,2-6]).

ctr_sol(nvalue,3,0,3,64,[1-4,2-36,3-24]).

ctr_sol(nvalue,4,0,4,625,[1-5,2-140,3-360,4-120]).

ctr_sol(nvalue,5,0,5,7776,[1-6,2-450,3-3000,4-3600,5-720]).

ctr_sol(
    nvalue,
    6,
    0,
    6,
    117649,
    [1-7,2-1302,3-18900,4-54600,5-37800,6-5040]).
```

```
ctr_sol(
    nvalue,
    7,
    0,
    7,
    2097152,
    [1-8,2-3528,3-101136,4-588000,5-940800,6-423360,7-40320]).

ctr_sol(
    nvalue,
    8,
    0,
    8,
    43046721,
    [1-9,
     2-9144,
     3-486864,
     4-5143824,
     5-15876000,
     6-16087680,
     7-5080320,
     8-362880]).

nvalue_b(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        NVAL#>=min(1,N),
        NVAL#=<N,
        list_dvar_range(VARS,R),
        NVAL#=<R,
        nvalue(NVAL,VARS).

nvalue_c(NVAL,VARIABLES) :-
        check_type(dvar,NVAL),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARIABLES,N),
        (   integer(NVAL) ->
            MIN is min(1,N),
            NVAL>=MIN,
            NVAL=<N
        ;   NVAL#>=min(1,N),
            NVAL#=<N
```

```
),
sort(VARS,SVARS),
length(SVARS,M),
NVAL#=M.
```

B.293 nvalue_on_intersection

◊ **META-DATA:**

```
ctr_date(nvalue_on_intersection,['20040530','20060812']).

ctr_origin(
    nvalue_on_intersection,
    Derived from %c and %c.,
    [common,nvalue]).

ctr_arguments(
    nvalue_on_intersection,
    ['NVAL'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    nvalue_on_intersection,
    [required('VARIABLES1',var),
     required('VARIABLES2',var),
     'NVAL'>=0,
     'NVAL'=<size('VARIABLES1'),
     'NVAL'=<size('VARIABLES2'),
     'NVAL'=<range('VARIABLES1'^var),
     'NVAL'=<range('VARIABLES2'^var)]).

ctr_example(
    nvalue_on_intersection,
    nvalue_on_intersection(
        2,
        [[var-1],[var-9],[var-1],[var-5]],
        [[var-2],[var-1],[var-9],[var-9],[var-6],[var-9]])).

ctr_typical(
    nvalue_on_intersection,
    ['NVAL'>0,
     'NVAL'<size('VARIABLES1'),
     'NVAL'<size('VARIABLES2'),
     'NVAL'<range('VARIABLES1'^var),
     'NVAL'<range('VARIABLES2'^var),
     size('VARIABLES1')>1,
     size('VARIABLES2')>1]).

ctr_exchangeable(
    nvalue_on_intersection,
```

```
    [args([['NVAL'],['VARIABLES1','VARIABLES2']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    nvalue_on_intersection,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NCC'='NVAL'],
    []).

ctr_pure_functional_dependency(nvalue_on_intersection,[]).

ctr_functional_dependency(nvalue_on_intersection,1,[2,3]).

ctr_contractible(
    nvalue_on_intersection,
    ['NVAL'=0],
    VARIABLES1,
    any).

ctr_contractible(
    nvalue_on_intersection,
    ['NVAL'=0],
    VARIABLES2,
    any).
```

B.294 nvalues

◊ META-DATA:

```
ctr_date(nvalues,['20030820','20060812']).

ctr_origin(nvalues,'Inspired by %c and %c.',[nvalue,count]).

ctr_arguments(
    nvalues,
    ['VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    nvalues,
    [required('VARIABLES',var),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    nvalues,
    nvalues(
        [[var-4],[var-5],[var-5],[var-4],[var-1],[var-5]],
        =,
        3)).

ctr_typical(
    nvalues,
    [size('VARIABLES')>1,
     'LIMIT'>1,
     'LIMIT'<size('VARIABLES'),
     in_list('RELOP',[=,<,>=,>,=<])]).

ctr_exchangeable(
    nvalues,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    nvalues,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['RELOP'('NSCC','LIMIT')],
    ['EQUIVALENCE']).
```

```
ctr_eval(nvalues,[reformulation(nvalues_r)]).

ctr_pure_functional_dependency(nvalues,[in_list('RELOP',[=])]).

ctr_contractible(
    nvalues,
    [in_list('RELOP',[<,=<])],
    VARIABLES,
    any).

ctr_contractible(
    nvalues,
    [in_list('RELOP',[=]),'LIMIT'=1,size('VARIABLES')>0],
    VARIABLES,
    any).

ctr_contractible(
    nvalues,
    [in_list('RELOP',[=]),'LIMIT'=size('VARIABLES')],
    VARIABLES,
    any).

ctr_extensible(
    nvalues,
    [in_list('RELOP',[>=,>])],
    VARIABLES,
    any).

ctr_cond_imply(
    nvalues,
    nvalues_except_0,
    [minval('VARIABLES'^var)>0],
    [],
    id).

nvalues_r(VARIABLES,RELOP,LIMIT) :-
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        length(VARIABLES,N),
        NVAL in 0..N,
        get_attr1(VARIABLES,VARS),
        nvalue(NVAL,VARS),
        call_term_relop_value(NVAL,RELOP,LIMIT).
```

B.295 nvalues_except_0

◇ META-DATA:

```
ctr_date(nvalues_except_0,['20030820','20060812']).

ctr_origin(nvalues_except_0,'Derived from %c.',[nvalues]).

ctr_arguments(
    nvalues_except_0,
    ['VARIABLES'-collection(var-dvar),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    nvalues_except_0,
    [required('VARIABLES',var),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    nvalues_except_0,
    nvalues_except_0(
        [[var-4],[var-5],[var-5],[var-4],[var-0],[var-1]],
        =,
        3)).

ctr_typical(
    nvalues_except_0,
    [size('VARIABLES')>1,
     'LIMIT'>1,
     'LIMIT'<size('VARIABLES'),
     atleast(1,'VARIABLES',0),
     in_list('RELOP',[=,<,>=,>,=<])]).

ctr_typical_model(nvalues_except_0,[atleast(2,'VARIABLES',0)]).

ctr_exchangeable(
    nvalues_except_0,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int(=\=(0)),=\=,all,dontcare)]).

ctr_graph(
    nvalues_except_0,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
```

```
    [variables1^var=\=0,variables1^var=variables2^var],
    ['RELOP'('NSCC','LIMIT')],
    []).

ctr_eval(nvalues_except_0,[reformulation(nvalues_except_0_r)]).

ctr_contractible(
    nvalues_except_0,
    [in_list('RELOP',[<,=<])],
    VARIABLES,
    any).

ctr_extensible(
    nvalues_except_0,
    [in_list('RELOP',[>=,>])],
    VARIABLES,
    any).

nvalues_except_0_r(VARIABLES,RELOP,LIMIT) :-
        collection(VARIABLES,[dvar]),
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        length(VARIABLES,N),
        N1 is N+1,
        NVAL1 in 1..N1,
        get_attr1(VARIABLES,VARS),
        append([0],VARS,VARS0),
        nvalue(NVAL1,VARS0),
        NVAL1#=NVAL+1,
        call_term_relop_value(NVAL,RELOP,LIMIT).
```

B.296 nvector

◇ **META-DATA:**

```
ctr_date(nvector,['20081220']).

ctr_origin(
    nvector,
    Introduced by G.~Chabert as a generalisation of %c,
    [nvalue]).

ctr_synonyms(nvector,[nvectors,npoint,npoints]).

ctr_types(nvector,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    nvector,
    ['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    nvector,
    [size('VECTOR')>=1,
     'NVEC'>=min(1,size('VECTORS')),
     'NVEC'=<size('VECTORS'),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    nvector,
    nvector(
        2,
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]]])).

ctr_typical(
    nvector,
    [size('VECTOR')>1,
     'NVEC'>1,
     'NVEC'<size('VECTORS'),
     size('VECTORS')>1]).

ctr_exchangeable(
    nvector,
```

```
    [items('VECTORS',all),
     items_sync('VECTORS'^vec,all),
     vals(['VECTORS'^vec],int,=\=,all,dontcare)]).

ctr_graph(
    nvector,
    ['VECTORS'],
    2,
    ['CLIQUE'>>collection(vectors1,vectors2)],
    [lex_equal(vectors1^vec,vectors2^vec)],
    ['NSCC'='NVEC'],
    ['EQUIVALENCE']).

ctr_eval(nvector,[reformulation(nvector_r)]).

ctr_pure_functional_dependency(nvector,[]).

ctr_functional_dependency(nvector,1,[2]).

ctr_contractible(
    nvector,
    ['NVEC'=1,size('VECTORS')>0],
    VECTORS,
    any).

ctr_contractible(
    nvector,
    ['NVEC'=size('VECTORS')],
    VECTORS,
    any).

nvector_r(0,[]) :-
        !.

nvector_r(NVEC,VECTORS) :-
        check_type(dvar,NVEC),
        collection(VECTORS,[col([dvar])]),
        same_size(VECTORS),
        length(VECTORS,N),
        NVEC#>=min(1,N),
        NVEC#=<N,
        nvector_common(NVEC,VECTORS).
```

B.297 nvectors

♢ Meta-Data:

```
ctr_date(nvectors,['20081226']).

ctr_origin(nvectors,'Inspired by %c and %c.',[nvector,count]).

ctr_synonyms(nvectors,[npoints]).

ctr_types(nvectors,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    nvectors,
    ['VECTORS'-collection(vec-'VECTOR'),
     'RELOP'-atom,
     'LIMIT'-dvar]).

ctr_restrictions(
    nvectors,
    [size('VECTOR')>=1,
     required('VECTORS',vec),
     same_size('VECTORS',vec),
     in_list('RELOP',[=,=\=,<,>=,>,=<])]).

ctr_example(
    nvectors,
    nvectors(
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]]],
        =,
        2)).

ctr_typical(
    nvectors,
    [size('VECTOR')>1,
     size('VECTORS')>1,
     in_list('RELOP',[=,<,>=,>,=<]),
     'LIMIT'>1,
     'LIMIT'<size('VECTORS')]).

ctr_exchangeable(
    nvectors,
```

```
    [items('VECTORS',all),
     items_sync('VECTORS'^vec,all),
     vals(['VECTORS'^vec],int,=\=,all,dontcare)]).

ctr_graph(
    nvectors,
    ['VECTORS'],
    2,
    ['CLIQUE'>>collection(vectors1,vectors2)],
    [lex_equal(vectors1^vec,vectors2^vec)],
    ['RELOP'('NSCC','LIMIT')],
    ['EQUIVALENCE']).

ctr_eval(nvectors,[reformulation(nvectors_r)]).

ctr_pure_functional_dependency(nvectors,[in_list('RELOP',[=])]).

ctr_contractible(
    nvectors,
    [in_list('RELOP',[<,=<])],
    VECTORS,
    any).

ctr_extensible(
    nvectors,
    [in_list('RELOP',[>=,>])],
    VECTORS,
    any).

nvectors_r(VECTORS,RELOP,LIMIT) :-
        memberchk(RELOP,[=,=\=,<,>=,>,=<]),
        check_type(dvar,LIMIT),
        length(VECTORS,N),
        NV in 0..N,
        eval(nvector(NV,VECTORS)),
        call_term_relop_value(NV,RELOP,LIMIT).
```

B.298 nvisible_from_end

◇ META-DATA:

```
ctr_date(nvisible_from_end,['20111228']).

ctr_origin(
    nvisible_from_end,
    Derived from %c,
    [nvisible_from_start]).

ctr_synonyms(nvisible_from_end,[nvisible,nvisible_from_right]).

ctr_arguments(
    nvisible_from_end,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nvisible_from_end,
    [required('VARIABLES',var),
     'N'>=min(1,size('VARIABLES')),
     'N'=<size('VARIABLES')]).

ctr_example(
    nvisible_from_end,
    [nvisible_from_end(
         2,
         [[var-1],
          [var-6],
          [var-2],
          [var-1],
          [var-4],
          [var-8],
          [var-2]]),
     nvisible_from_end(
         1,
         [[var-3],
          [var-6],
          [var-2],
          [var-1],
          [var-4],
          [var-8],
          [var-8]]),
     nvisible_from_end(
         7,
         [[var-9],
```

```
            [var-8],
            [var-7],
            [var-5],
            [var-4],
            [var-3],
            [var-2]])]).

ctr_typical(
    nvisible_from_end,
    [size('VARIABLES')>2,range('VARIABLES'^var)>2]).

ctr_typical_model(nvisible_from_end,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    nvisible_from_end,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    nvisible_from_end,
    [checker(nvisible_from_end_c),
     automaton(nvisible_from_end_a)]).

ctr_pure_functional_dependency(nvisible_from_end,[]).

ctr_functional_dependency(nvisible_from_end,1,[2]).

ctr_sol(nvisible_from_end,2,0,2,9,[1-6,2-3]).

ctr_sol(nvisible_from_end,3,0,3,64,[1-30,2-30,3-4]).

ctr_sol(nvisible_from_end,4,0,4,625,[1-225,2-305,3-90,4-5]).

ctr_sol(
    nvisible_from_end,
    5,
    0,
    5,
    7776,
    [1-2275,2-3675,3-1610,4-210,5-6]).

ctr_sol(
    nvisible_from_end,
    6,
    0,
    6,
    117649,
```

```
    [1-29008,2-52794,3-29400,4-6020,5-420,6-7]).

ctr_sol(
    nvisible_from_end,
    7,
    0,
    7,
    2097152,
    [1-446964,2-889056,3-583548,4-158760,5-18060,6-756,7-8]).

ctr_sol(
    nvisible_from_end,
    8,
    0,
    8,
    43046721,
    [1-8080425,
     2-17238570,
     3-12780180,
     4-4238367,
     5-661500,
     6-46410,
     7-1260,
     8-9]).

nvisible_from_end_c(N,VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        MIN is min(1,L),
        check_type(dvar(MIN,L),N),
        get_attr1(VARIABLES,VARS),
        reverse(VARS,RVARS),
        nvisible_from_start(s,RVARS,0,0,N).

nvisible_from_end_a(FLAG,N,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        MIN is min(1,L),
        check_type(dvar(MIN,L),N),
        get_attr1(VARIABLES,VARS),
        reverse(VARS,RVARS),
        (foreach(_46920,VARS),foreach(0,SIGNATURE)do true),
        automaton(
            RVARS,
            Vi,
            SIGNATURE,
```

```
    [source(s),sink(s),sink(t)],
    [arc(s,0,t,[Vi,1]),
     arc(t,0,t,(M#<Vi->[Vi,C+1];M#>=Vi->[M,C]))],
    [M,C],
    [0,0],
    [_47009,COUNT]),
COUNT#=N#<=>FLAG.
```

B.299 nvisible_from_start

♢ META-DATA:

```
ctr_date(nvisible_from_start,['20111227']).

ctr_origin(
    nvisible_from_start,
    Derived from a puzzle called skyscraper,
    []).

ctr_synonyms(nvisible_from_start,[nvisible,nvisible_from_left]).

ctr_arguments(
    nvisible_from_start,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    nvisible_from_start,
    [required('VARIABLES',var),
     'N'>=min(1,size('VARIABLES')),
     'N'=<size('VARIABLES')]).

ctr_example(
    nvisible_from_start,
    [nvisible_from_start(
         3,
         [[var-1],
          [var-6],
          [var-2],
          [var-1],
          [var-4],
          [var-8],
          [var-2]]),
     nvisible_from_start(
         1,
         [[var-8],
          [var-6],
          [var-2],
          [var-1],
          [var-4],
          [var-8],
          [var-2]]),
     nvisible_from_start(
         7,
         [[var-0],
```

```
        [var-2],
        [var-3],
        [var-5],
        [var-6],
        [var-7],
        [var-9]])]).

ctr_typical(
    nvisible_from_start,
    [size('VARIABLES')>2,range('VARIABLES'^var)>2]).

ctr_typical_model(
    nvisible_from_start,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    nvisible_from_start,
    [translate(['VARIABLES'^var])]).

ctr_eval(
    nvisible_from_start,
    [checker(nvisible_from_start_c),
     automaton(nvisible_from_start_a)]).

ctr_pure_functional_dependency(nvisible_from_start,[]).

ctr_functional_dependency(nvisible_from_start,1,[2]).

ctr_sol(nvisible_from_start,2,0,2,9,[1-6,2-3]).

ctr_sol(nvisible_from_start,3,0,3,64,[1-30,2-30,3-4]).

ctr_sol(nvisible_from_start,4,0,4,625,[1-225,2-305,3-90,4-5]).

ctr_sol(
    nvisible_from_start,
    5,
    0,
    5,
    7776,
    [1-2275,2-3675,3-1610,4-210,5-6]).

ctr_sol(
    nvisible_from_start,
    6,
    0,
```

```
    6,
    117649,
    [1-29008,2-52794,3-29400,4-6020,5-420,6-7]).

ctr_sol(
    nvisible_from_start,
    7,
    0,
    7,
    2097152,
    [1-446964,2-889056,3-583548,4-158760,5-18060,6-756,7-8]).

ctr_sol(
    nvisible_from_start,
    8,
    0,
    8,
    43046721,
    [1-8080425,
     2-17238570,
     3-12780180,
     4-4238367,
     5-661500,
     6-46410,
     7-1260,
     8-9]).

nvisible_from_start_c(N,VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        MIN is min(1,L),
        check_type(dvar(MIN,L),N),
        get_attr1(VARIABLES,VARS),
        nvisible_from_start(s,VARS,0,0,N).

nvisible_from_start_a(FLAG,N,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        MIN is min(1,L),
        check_type(dvar(MIN,L),N),
        get_attr1(VARIABLES,VARS),
        (foreach(_47232,VARS),foreach(0,SIGNATURE)do true),
        automaton(
            VARS,
            Vi,
            SIGNATURE,
```

```
    [source(s),sink(s),sink(t)],
    [arc(s,0,t,[Vi,1]),
     arc(t,0,t,(M#<Vi->[Vi,C+1];M#>=Vi->[M,C]))],
    [M,C],
    [0,0],
    [_47321,COUNT]),
COUNT#=N#<=>FLAG.
```

B.300 open_alldifferent

◇ **META-DATA:**

```
ctr_date(open_alldifferent,['20060824','20090524']).

ctr_origin(open_alldifferent,'\\cite{HoeveRegin06}',[]).

ctr_synonyms(
    open_alldifferent,
    [open_alldiff,open_alldistinct,open_distinct]).

ctr_arguments(
    open_alldifferent,
    ['S'-svar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    open_alldifferent,
    ['S'>=1,'S'=<size('VARIABLES'),required('VARIABLES',var)]).

ctr_example(
    open_alldifferent,
    open_alldifferent(
        {2,3,4},
        [[var-9],[var-1],[var-9],[var-3]])).

ctr_typical(open_alldifferent,[size('VARIABLES')>2]).

ctr_exchangeable(
    open_alldifferent,
    [vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    open_alldifferent,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var,
     variables1^key in_set 'S',
     variables2^key in_set 'S'],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).

ctr_contractible(open_alldifferent,[],'VARIABLES',suffix).
```

B.301 open_among

◇ **META-DATA:**

```
ctr_date(open_among,['20060824']).

ctr_origin(
    open_among,
    Derived from %c and %c.,
    [among,open_global_cardinality]).

ctr_arguments(
    open_among,
    ['S'-svar,
     'NVAR'-dvar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    open_among,
    ['S'>=1,
     'S'=<size('VARIABLES'),
     'NVAR'>=0,
     'NVAR'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    open_among,
    open_among(
        {2,3,4,5},
        3,
        [[var-8],[var-5],[var-5],[var-4],[var-1]],
        [[val-1],[val-5],[val-8]])).

ctr_typical(
    open_among,
    ['NVAR'>0,
     'NVAR'<size('VARIABLES'),
     size('VARIABLES')>1,
     size('VALUES')>1,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    open_among,
```

```
    [items('VALUES',all),
     vals(
         ['VARIABLES'^var],
         comp('VALUES'^val),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    open_among,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var in 'VALUES',variables^key in_set 'S'],
    ['NARC'='NVAR'],
    []).

ctr_functional_dependency(open_among,2,[1,3,4]).

ctr_contractible(open_among,['NVAR'=0],'VARIABLES',suffix).
```

B.302 open_atleast

◊ **META-DATA:**

```
ctr_date(open_atleast,['20060824']).

ctr_origin(
    open_atleast,
    Derived from %c and %c.,
    [atleast,open_global_cardinality]).

ctr_arguments(
    open_atleast,
    ['S'-svar,
     'N'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUE'-int]).

ctr_restrictions(
    open_atleast,
    ['S'>=1,
     'S'=<size('VARIABLES'),
     'N'>=0,
     'N'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    open_atleast,
    open_atleast(
        {2,3,4},
        2,
        [[var-4],[var-2],[var-4],[var-4]],
        4)).

ctr_typical(
    open_atleast,
    ['N'>0,'N'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_exchangeable(
    open_atleast,
    [vals(['N'],int(>=(0)),>,dontcare,dontcare),
     vals(
         ['VARIABLES'^var],
         comp('VALUE'),
         >=,
         dontcare,
```

```
         dontcare)]).

ctr_graph(
    open_atleast,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE',variables^key in_set 'S'],
    ['NARC'>='N'],
    []).

ctr_extensible(open_atleast,[],'VARIABLES',suffix).
```

B.303 open_atmost

◊ **META-DATA:**

```
ctr_date(open_atmost,['20060824']).

ctr_origin(
    open_atmost,
    Derived from %c and %c.,
    [atmost,open_global_cardinality]).

ctr_arguments(
    open_atmost,
    ['S'-svar,
     'N'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUE'-int]).

ctr_restrictions(
    open_atmost,
    ['S'>=1,
     'S'=<size('VARIABLES'),
     'N'>=0,
     required('VARIABLES',var)]).

ctr_example(
    open_atmost,
    open_atmost({2,3,4},1,[[var-2],[var-2],[var-4],[var-5]],2)).

ctr_typical(
    open_atmost,
    ['N'>0,'N'<size('VARIABLES'),size('VARIABLES')>1]).

ctr_exchangeable(
    open_atmost,
    [vals(['N'],int,<,dontcare,dontcare),
     vals(
         ['VARIABLES'^var],
         comp('VALUE'),
         =<,
         dontcare,
         dontcare)]).

ctr_graph(
    open_atmost,
    ['VARIABLES'],
```

```
    1,
    ['SELF'>>collection(variables)],
    [variables^var='VALUE',variables^key in_set 'S'],
    ['NARC'=<'N'],
    []).

ctr_contractible(open_atmost,[],'VARIABLES',suffix).
```

B.304 open_global_cardinality

◇ **META-DATA:**

```
ctr_date(open_global_cardinality,['20060824']).

ctr_origin(open_global_cardinality,'\\cite{HoeveRegin06}',[]).

ctr_synonyms(open_global_cardinality,[open_gcc,ogcc]).

ctr_arguments(
    open_global_cardinality,
    ['S'-svar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,noccurrence-dvar)]).

ctr_restrictions(
    open_global_cardinality,
    ['S'>=1,
     'S'=<size('VARIABLES'),
     required('VARIABLES',var),
     required('VALUES',[val,noccurrence]),
     distinct('VALUES',val),
     'VALUES'^noccurrence>=0,
     'VALUES'^noccurrence=<size('VARIABLES')]).

ctr_example(
    open_global_cardinality,
    open_global_cardinality(
        {2,3,4},
        [[var-3],[var-3],[var-8],[var-6]],
        [[val-3,noccurrence-1],
         [val-5,noccurrence-0],
         [val-6,noccurrence-1]])).

ctr_typical(
    open_global_cardinality,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
     range('VALUES'^noccurrence)>1,
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    open_global_cardinality,
    [items('VALUES',all),
```

```
     vals(
         ['VARIABLES'^var],
         all(notin('VALUES'^val)),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    open_global_cardinality,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val,variables^key in_set 'S'],
    ['NVERTEX'='VALUES'^noccurrence],
    []).
```

B.305 open_global_cardinality_low_up

◇ **META-DATA:**

```
ctr_date(open_global_cardinality_low_up,['20060824']).

ctr_origin(
    open_global_cardinality_low_up,
    \cite{HoeveRegin06},
    []).

ctr_arguments(
    open_global_cardinality_low_up,
    ['S'-svar,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
    open_global_cardinality_low_up,
    ['S'>=1,
     'S'=<size('VARIABLES'),
     required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,omin,omax]),
     distinct('VALUES',val),
     'VALUES'^omin>=0,
     'VALUES'^omax=<size('VARIABLES'),
     'VALUES'^omin=<'VALUES'^omax]).

ctr_example(
    open_global_cardinality_low_up,
    open_global_cardinality_low_up(
        {2,3,4},
        [[var-3],[var-3],[var-8],[var-6]],
        [[val-3,omin-1,omax-3],
         [val-5,omin-0,omax-1],
         [val-6,omin-1,omax-2]])).

ctr_typical(
    open_global_cardinality_low_up,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
     'VALUES'^omin=<size('VARIABLES'),
     'VALUES'^omax>0,
     'VALUES'^omax=<size('VARIABLES'),
```

```
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    open_global_cardinality_low_up,
    [items('VALUES',all),
     vals(
         ['VARIABLES'^var],
         all(notin('VALUES'^val)),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    open_global_cardinality_low_up,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val,variables^key in_set 'S'],
    ['NVERTEX'>='VALUES'^omin,'NVERTEX'=<'VALUES'^omax],
    []).
```

B.306 open_maximum

◇ **META-DATA:**

```
ctr_date(open_maximum,['20090507']).

ctr_origin(open_maximum,'Derived from %c',[maximum]).

ctr_arguments(
    open_maximum,
    ['MAX'-dvar,'VARIABLES'-collection(var-dvar,bool-dvar)]).

ctr_restrictions(
    open_maximum,
    [size('VARIABLES')>0,
     required('VARIABLES',[var,bool]),
     'VARIABLES'^bool>=0,
     'VARIABLES'^bool=<1]).

ctr_example(
    open_maximum,
    open_maximum(
        5,
        [[var-3,bool-1],
         [var-1,bool-0],
         [var-7,bool-0],
         [var-5,bool-1],
         [var-5,bool-1]])).

ctr_typical(
    open_maximum,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_exchangeable(
    open_maximum,
    [items('VARIABLES',all),
     translate(['MAX','VARIABLES'^var])]).

ctr_eval(open_maximum,[automaton(open_maximum_a)]).

open_maximum_a(FLAG,MAX,VARIABLES) :-
        check_type(dvar,MAX),
        collection(VARIABLES,[dvar,dvar(0,1)]),
        length(VARIABLES,N),
        N>0,
        open_maximum_signature(VARIABLES,SIGNATURE,MAX),
```

```
        AUTOMATON=
        automaton(
            SIGNATURE,
            _29077,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),
             arc(s,1,t),
             arc(s,3,s),
             arc(s,4,s),
             arc(s,5,s),
             arc(t,1,t),
             arc(t,0,t),
             arc(t,3,t),
             arc(t,4,t),
             arc(t,5,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2,3,4,5],AUTOMATON).

open_maximum_signature([],[],_27158).

open_maximum_signature([[var-VAR,bool-B]|VARs],[S|Ss],MAX) :-
        S in 0..5,
        B#=1#/\MAX#>VAR#<=>S#=0,
        B#=1#/\MAX#=VAR#<=>S#=1,
        B#=1#/\MAX#<VAR#<=>S#=2,
        B#=0#/\MAX#>VAR#<=>S#=3,
        B#=0#/\MAX#=VAR#<=>S#=4,
        B#=0#/\MAX#<VAR#<=>S#=5,
        open_maximum_signature(VARs,Ss,MAX).
```

B.307 open_minimum

◊ **META-DATA:**

```
ctr_date(open_minimum,['20090506']).

ctr_origin(open_minimum,'Derived from %c',[minimum]).

ctr_arguments(
    open_minimum,
    ['MIN'-dvar,'VARIABLES'-collection(var-dvar,bool-dvar)]).

ctr_restrictions(
    open_minimum,
    [size('VARIABLES')>0,
     required('VARIABLES',[var,bool]),
     'VARIABLES'^bool>=0,
     'VARIABLES'^bool=<1]).

ctr_example(
    open_minimum,
    open_minimum(
        3,
        [[var-3,bool-1],
         [var-1,bool-0],
         [var-7,bool-0],
         [var-5,bool-1],
         [var-5,bool-1]])).

ctr_typical(
    open_minimum,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_exchangeable(
    open_minimum,
    [items('VARIABLES',all),
     translate(['MIN','VARIABLES'^var])]).

ctr_eval(open_minimum,[automaton(open_minimum_a)]).

open_minimum_a(FLAG,MIN,VARIABLES) :-
        check_type(dvar,MIN),
        collection(VARIABLES,[dvar,dvar(0,1)]),
        length(VARIABLES,N),
        N>0,
        open_minimum_signature(VARIABLES,SIGNATURE,MIN),
```

```
        AUTOMATON=
        automaton(
            SIGNATURE,
            _30293,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),
             arc(s,1,t),
             arc(s,3,s),
             arc(s,4,s),
             arc(s,5,s),
             arc(t,1,t),
             arc(t,0,t),
             arc(t,3,t),
             arc(t,4,t),
             arc(t,5,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2,3,4,5],AUTOMATON).

open_minimum_signature([],[],_28374).

open_minimum_signature([[var-VAR,bool-B]|VARs],[S|Ss],MIN) :-
        S in 0..5,
        B#=1#/\MIN#<VAR#<=>S#=0,
        B#=1#/\MIN#=VAR#<=>S#=1,
        B#=1#/\MIN#>VAR#<=>S#=2,
        B#=0#/\MIN#<VAR#<=>S#=3,
        B#=0#/\MIN#=VAR#<=>S#=4,
        B#=0#/\MIN#>VAR#<=>S#=5,
        open_minimum_signature(VARs,Ss,MIN).
```

B.308 opposite_sign

◇ **META-DATA:**

```
ctr_predefined(opposite_sign).

ctr_date(opposite_sign,['20100821']).

ctr_origin(opposite_sign,'Arithmetic.',[]).

ctr_arguments(opposite_sign,['VAR1'-dvar,'VAR2'-dvar]).

ctr_restrictions(opposite_sign,[]).

ctr_example(opposite_sign,opposite_sign(6,-3)).

ctr_typical(opposite_sign,['VAR1'=\=0]).

ctr_exchangeable(opposite_sign,[args([['VAR1','VAR2']])]).

ctr_eval(
    opposite_sign,
    [checker(opposite_sign_c),builtin(opposite_sign_b)]).

opposite_sign_c(VAR1,VAR2) :-
        check_type(int,VAR1),
        check_type(int,VAR2),
        (   VAR1=0 ->
            true
        ;   VAR2=0 ->
            true
        ;   VAR1>0 ->
            VAR2<0
        ;   VAR2>0
        ).

opposite_sign_b(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#>=0#/\VAR2#=<0#\/VAR2#>=0#/\VAR1#=<0.
```

B.309 or

◇ META-DATA:

```
ctr_date(or,['20051226']).

ctr_origin(or,'Logic',[]).

ctr_synonyms(or,[rel]).

ctr_arguments(or,['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    or,
    ['VAR'>=0,
     'VAR'=<1,
     size('VARIABLES')>=2,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    or,
    [or(0,[[var-0],[var-0]]),
     or(1,[[var-0],[var-1]]),
     or(1,[[var-1],[var-0]]),
     or(1,[[var-1],[var-1]]),
     or(1,[[var-1],[var-0],[var-1]])]).

ctr_exchangeable(or,[items('VARIABLES',all)]).

ctr_eval(or,[automaton(or_a)]).

ctr_pure_functional_dependency(or,[]).

ctr_functional_dependency(or,1,[2]).

ctr_contractible(or,['VAR'=0],'VARIABLES',any).

ctr_extensible(or,['VAR'=1],'VARIABLES',any).

ctr_aggregate(or,[],[#\/,union]).

ctr_cond_imply(
    or,
    some_equal,
```

```
    [size('VARIABLES')>2],
    [],
    ['VARIABLES']).

ctr_cond_imply(or,nor,['VAR'=0],['VAR'=1],[none,'VARIABLES']).

ctr_cond_imply(or,nor,['VAR'=1],['VAR'=0],[none,'VARIABLES']).

ctr_sol(or,2,0,2,4,[0-1,1-3]).

ctr_sol(or,3,0,3,8,[0-1,1-7]).

ctr_sol(or,4,0,4,16,[0-1,1-15]).

ctr_sol(or,5,0,5,32,[0-1,1-31]).

ctr_sol(or,6,0,6,64,[0-1,1-63]).

ctr_sol(or,7,0,7,128,[0-1,1-127]).

ctr_sol(or,8,0,8,256,[0-1,1-255]).

or_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,L),
        L>1,
        get_attr1(VARIABLES,LIST),
        append([VAR],LIST,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _50031,
            LIST_VARIABLES,
            [source(s),sink(i),sink(k)],
            [arc(s,0,i),
             arc(s,1,j),
             arc(i,0,i),
             arc(j,0,j),
             arc(j,1,k),
             arc(k,0,k),
             arc(k,1,k)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.310 orchard

◊ **META-DATA:**

```
ctr_date(orchard,['20000128','20030820']).

ctr_origin(orchard,'\\cite{Jackson1821}',[]).

ctr_arguments(
    orchard,
    ['NROW'-dvar,'TREES'-collection(index-int,x-dvar,y-dvar)]).

ctr_restrictions(
    orchard,
    ['NROW'>=0,
     'TREES'^index>=1,
     'TREES'^index=<size('TREES'),
     required('TREES',[index,x,y]),
     distinct('TREES',index),
     'TREES'^x>=0,
     'TREES'^y>=0]).

ctr_example(
    orchard,
    orchard(
        10,
        [[index-1,x-0,y-0],
         [index-2,x-4,y-0],
         [index-3,x-8,y-0],
         [index-4,x-2,y-4],
         [index-5,x-4,y-4],
         [index-6,x-6,y-4],
         [index-7,x-0,y-8],
         [index-8,x-4,y-8],
         [index-9,x-8,y-8]])).

ctr_typical(orchard,['NROW'>0,size('TREES')>3]).

ctr_exchangeable(
    orchard,
    [items('TREES',all),
     attrs_sync('TREES',[[index],[x,y]]),
     translate(['TREES'^x]),
     translate(['TREES'^y])]).

ctr_graph(
```

```
    orchard,
    ['TREES'],
    3,
    ['CLIQUE'(<)>>collection(trees1,trees2,trees3)],
    [trees1^x*trees2^y-trees1^x*trees3^y+
     trees1^y*trees3^x-trees1^y*trees2^x+
     trees2^x*trees3^y-trees2^y*trees3^x=
     0],
    ['NARC'='NROW'],
    []).

ctr_pure_functional_dependency(orchard,[]).

ctr_functional_dependency(orchard,1,[2]).

ctr_application(orchard,[2]).
```

B.311 order

◊ **META-DATA:**

```
ctr_predefined(order).

ctr_date(order,['20120502']).

ctr_origin(order,'Derived from %c',[sort_permutation]).

ctr_types(order,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    order,
    ['VECTORS'-collection(vec-'VECTOR'),
     'PERMUTATION'-collection(var-dvar)]).

ctr_restrictions(
    order,
    [size('VECTOR')>=1,
     size('VECTORS')>=1,
     required('VECTORS',vec),
     same_size('VECTORS',vec),
     required('PERMUTATION',var),
     'PERMUTATION'^var>=1,
     'PERMUTATION'^var=<size('PERMUTATION'),
     size('PERMUTATION')=size('VECTORS')]).

ctr_example(
    order,
    order(
        [[vec-[[var-1],[var-1],[var-2],[var-2]]],
         [vec-[[var-2],[var-1],[var-2],[var-1]]],
         [vec-[[var-2],[var-1],[var-1],[var-1]]],
         [vec-[[var-1],[var-1],[var-1],[var-2]]],
         [vec-[[var-1],[var-2],[var-2],[var-1]]],
         [vec-[[var-1],[var-1],[var-1],[var-1]]],
         [vec-[[var-2],[var-2],[var-1],[var-1]]],
         [vec-[[var-2],[var-1],[var-1],[var-2]]]],
        [[var-3],
         [var-7],
         [var-5],
         [var-2],
         [var-4],
         [var-1],
         [var-8],
```

```
         [var-6]])).

ctr_typical(order,[size('VECTOR')>1,size('VECTORS')>1]).

ctr_eval(order,[reformulation(order_c)]).

ctr_functional_dependency(order,2,[1]).

order_c(VECTORS,PERMUTATION) :-
        collection(VECTORS,[col([int])]),
        VECTORS=[[vec-VECTOR]|_27219],
        same_size(VECTORS),
        length(VECTOR,M),
        M>=1,
        length(VECTORS,N),
        collection(PERMUTATION,[int(1,N)]),
        length(PERMUTATION,N),
        get_attr1(PERMUTATION,P),
        sort(P,S),
        length(S,N),
        order_c1(VECTORS,P,VP),
        sort(VP,SVP),
        order_c2(SVP,1).

order_c1([],[],[]) :-
        !.

order_c1([VEC|R],[O|S],[VEC-O|T]) :-
        order_c1(R,S,T).

order_c2([],_27195) :-
        !.

order_c2([_VEC-O],O) :-
        !.

order_c2([VEC1-O1,VEC2-O2|R],O1) :-
        VEC1\==VEC2,
        Next is O1+1,
        order_c2([VEC2-O2|R],Next).
```

B.312 ordered_atleast_nvector

◊ **META-DATA:**

```
ctr_date(ordered_atleast_nvector,['20080921']).

ctr_origin(
    ordered_atleast_nvector,
    Conjoin %c and %c.,
    [atleast_nvector,lex_chain_lesseq]).

ctr_synonyms(
    ordered_atleast_nvector,
    [ordered_atleast_nvectors,
     ordered_atleast_npoint,
     ordered_atleast_npoints]).

ctr_types(
    ordered_atleast_nvector,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    ordered_atleast_nvector,
    ['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    ordered_atleast_nvector,
    [size('VECTOR')>=1,
     'NVEC'>=0,
     'NVEC'=<size('VECTORS'),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    ordered_atleast_nvector,
    ordered_atleast_nvector(
        2,
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-9],[var-4]]]])).

ctr_typical(
    ordered_atleast_nvector,
    [size('VECTOR')>1,
```

```
     'NVEC'>0,
     'NVEC'<size('VECTORS'),
     size('VECTORS')>1]).

ctr_exchangeable(
    ordered_atleast_nvector,
    [vals(['NVEC'],int(>=(0)),>,dontcare,dontcare)]).

ctr_graph(
    ordered_atleast_nvector,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_lesseq(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')-1],
    []).

ctr_graph(
    ordered_atleast_nvector,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_less(vectors1^vec,vectors2^vec)],
    ['NCC'>='NVEC'],
    []).

ctr_eval(
    ordered_atleast_nvector,
    [reformulation(ordered_atleast_nvector_r)]).

ordered_atleast_nvector_r(0,[]) :-
        !.

ordered_atleast_nvector_r(NVEC,VECTORS) :-
        eval(atleast_nvector(NVEC,VECTORS)),
        eval(lex_chain_lesseq(VECTORS)).
```

B.313 ordered_atmost_nvector

◇ **META-DATA:**

```
ctr_date(ordered_atmost_nvector,['20080921']).

ctr_origin(
    ordered_atmost_nvector,
    Conjoin %c and %c.,
    [atmost_nvector,lex_chain_lesseq]).

ctr_synonyms(
    ordered_atmost_nvector,
    [ordered_atmost_nvectors,
     ordered_atmost_npoint,
     ordered_atmost_npoints]).

ctr_types(
    ordered_atmost_nvector,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    ordered_atmost_nvector,
    ['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    ordered_atmost_nvector,
    [size('VECTOR')>=1,
     'NVEC'>=min(1,size('VECTORS')),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    ordered_atmost_nvector,
    ordered_atmost_nvector(
        3,
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-9],[var-3]]]])).

ctr_typical(
    ordered_atmost_nvector,
    [size('VECTOR')>1,
     'NVEC'>1,
```

```
     'NVEC'<size('VECTORS'),
     size('VECTORS')>1]).

ctr_exchangeable(
    ordered_atmost_nvector,
    [vals(['NVEC'],int,<,dontcare,dontcare)]).

ctr_graph(
    ordered_atmost_nvector,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_lesseq(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')-1],
    []).

ctr_graph(
    ordered_atmost_nvector,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_less(vectors1^vec,vectors2^vec)],
    ['NCC'=<'NVEC'],
    []).

ctr_eval(
    ordered_atmost_nvector,
    [reformulation(ordered_atmost_nvector_r)]).

ctr_contractible(ordered_atmost_nvector,[],'VECTORS',any).

ordered_atmost_nvector_r(0,[]) :-
        !.

ordered_atmost_nvector_r(NVEC,VECTORS) :-
        eval(atmost_nvector(NVEC,VECTORS)),
        eval(lex_chain_lesseq(VECTORS)).
```

B.314 ordered_global_cardinality

◇ **META-DATA:**

```
ctr_date(ordered_global_cardinality,['20090911']).

ctr_origin(
    ordered_global_cardinality,
    \cite{PetitRegin09},
    []).

ctr_usual_name(ordered_global_cardinality,ordgcc).

ctr_synonyms(ordered_global_cardinality,[ordered_gcc]).

ctr_arguments(
    ordered_global_cardinality,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,omax-int)]).

ctr_restrictions(
    ordered_global_cardinality,
    [required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,omax]),
     increasing_seq('VALUES',[val]),
     'VALUES'^omax>=0,
     'VALUES'^omax=<size('VARIABLES')]).

ctr_example(
    ordered_global_cardinality,
    ordered_global_cardinality(
        [[var-2],[var-0],[var-1],[var-0],[var-0]],
        [[val-0,omax-5],[val-1,omax-3],[val-2,omax-1]])).

ctr_exchangeable(
    ordered_global_cardinality,
    [items('VARIABLES',all)]).

ctr_graph(
    ordered_global_cardinality,
    ['VARIABLES'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var>='VALUES'^val],
    ['NVERTEX'=<'VALUES'^omax],
```

```
    []).

ctr_eval(
    ordered_global_cardinality,
    [reformulation(ordered_global_cardinality_r)]).

ctr_contractible(ordered_global_cardinality,[],'VALUES',any).

ordered_global_cardinality_r(VARIABLES,VALUES) :-
        length(VARIABLES,N),
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int,int(0,N)]),
        length(VALUES,M),
        M>0,
        collection_increasing_seq(VALUES,[1]),
        (   N=0 ->
            true
        ;   get_attr1(VALUES,VALS),
            get_attr2(VALUES,OMAXS),
            length(OCCS,M),
            domain(OCCS,0,N),
            create_collection(
                VALS,
                OCCS,
                val,
                noccurrence,
                VALUES_GC),
            eval(global_cardinality(VARIABLES,VALUES_GC)),
            reverse(OCCS,ROCCS),
            build_sliding_sums(ROCCS,0,SUMS),
            reverse(OMAXS,ROMAXS),
            ordered_global_cardinality1(SUMS,ROMAXS)
        ).

ordered_global_cardinality1([],[]).

ordered_global_cardinality1([V|R],[L|S]) :-
        V#=<L,
        ordered_global_cardinality1(R,S).
```

B.315 ordered_nvector

◇ **META-DATA:**

```
ctr_date(ordered_nvector,['20080919']).

ctr_origin(ordered_nvector,'Derived from %c.',[nvector]).

ctr_synonyms(
    ordered_nvector,
    [ordered_nvectors,ordered_npoint,ordered_npoints]).

ctr_types(ordered_nvector,['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    ordered_nvector,
    ['NVEC'-dvar,'VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    ordered_nvector,
    [size('VECTOR')>=1,
     'NVEC'>=min(1,size('VECTORS')),
     'NVEC'=<size('VECTORS'),
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    ordered_nvector,
    ordered_nvector(
        2,
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-9],[var-3]]]])).

ctr_typical(
    ordered_nvector,
    [size('VECTOR')>1,
     'NVEC'>1,
     'NVEC'<size('VECTORS'),
     size('VECTORS')>1]).

ctr_graph(
    ordered_nvector,
    ['VECTORS'],
```

```
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_lesseq(vectors1^vec,vectors2^vec)],
    ['NARC'=size('VECTORS')-1],
    []).

ctr_graph(
    ordered_nvector,
    ['VECTORS'],
    2,
    ['PATH'>>collection(vectors1,vectors2)],
    [lex_less(vectors1^vec,vectors2^vec)],
    ['NCC'='NVEC'],
    []).

ctr_eval(ordered_nvector,[reformulation(ordered_nvector_r)]).

ctr_functional_dependency(ordered_nvector,1,[2]).

ctr_contractible(
    ordered_nvector,
    ['NVEC'=1,size('VECTORS')>0],
    VECTORS,
    any).

ctr_contractible(
    ordered_nvector,
    ['NVEC'=size('VECTORS')],
    VECTORS,
    any).

ordered_nvector_r(0,[]) :-
        !.

ordered_nvector_r(NVEC,VECTORS) :-
        eval(nvector(NVEC,VECTORS)),
        eval(lex_chain_lesseq(VECTORS)).
```

B.316 orth_link_ori_siz_end

◇ **META-DATA:**

```
ctr_date(orth_link_ori_siz_end,['20030820','20060812']).

ctr_origin(
    orth_link_ori_siz_end,
    Used by several constraints between orthotopes,
    []).

ctr_arguments(
    orth_link_ori_siz_end,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_restrictions(
    orth_link_ori_siz_end,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end]).

ctr_example(
    orth_link_ori_siz_end,
    orth_link_ori_siz_end(
        [[ori-2,siz-2,end-4],[ori-1,siz-3,end-4]])).

ctr_typical(
    orth_link_ori_siz_end,
    [size('ORTHOTOPE')>1,'ORTHOTOPE'^siz>0]).

ctr_exchangeable(
    orth_link_ori_siz_end,
    [items('ORTHOTOPE',all),
     translate(['ORTHOTOPE'^ori,'ORTHOTOPE'^end]),
     translate(['ORTHOTOPE'^siz,'ORTHOTOPE'^end])]).

ctr_graph(
    orth_link_ori_siz_end,
    ['ORTHOTOPE'],
    1,
    ['SELF'>>collection(orthotope)],
    [orthotope^ori+orthotope^siz=orthotope^end],
    ['NARC'=size('ORTHOTOPE')],
    []).
```

```
ctr_eval(
    orth_link_ori_siz_end,
    [reformulation(orth_link_ori_siz_end_r)]).

ctr_pure_functional_dependency(orth_link_ori_siz_end,[]).

ctr_functional_dependency(orth_link_ori_siz_end,1-1,[1-2,1-3]).

ctr_functional_dependency(orth_link_ori_siz_end,1-2,[1-1,1-3]).

ctr_functional_dependency(orth_link_ori_siz_end,1-3,[1-1,1-2]).

ctr_contractible(orth_link_ori_siz_end,[],'ORTHOTOPE',any).

orth_link_ori_siz_end_r(ORTHOTOPE) :-
        collection(ORTHOTOPE,[dvar,dvar_gteq(0),dvar]),
        length(ORTHOTOPE,N),
        N>0,
        get_attr1(ORTHOTOPE,ORIGINS),
        get_attr2(ORTHOTOPE,SIZES),
        get_attr3(ORTHOTOPE,ENDS),
        gen_varcst(ORIGINS,SIZES,ENDS).
```

B.317 orth_on_the_ground

♢ META-DATA:

```
ctr_date(orth_on_the_ground,['20030820','20040726','20060812']).

ctr_origin(
    orth_on_the_ground,
    Used for defining %c.,
    [place_in_pyramid]).

ctr_arguments(
    orth_on_the_ground,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar),
     'VERTICAL_DIM'-int]).

ctr_restrictions(
    orth_on_the_ground,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     'VERTICAL_DIM'>=1,
     'VERTICAL_DIM'=<size('ORTHOTOPE'),
     orth_link_ori_siz_end('ORTHOTOPE')]).

ctr_example(
    orth_on_the_ground,
    orth_on_the_ground(
        [[ori-1,siz-2,end-3],[ori-2,siz-3,end-5]],
        1)).

ctr_typical(
    orth_on_the_ground,
    [size('ORTHOTOPE')>1,'ORTHOTOPE'^siz>0]).

ctr_graph(
    orth_on_the_ground,
    ['ORTHOTOPE'],
    1,
    ['SELF'>>collection(orthotope)],
    [orthotope^key='VERTICAL_DIM',orthotope^ori=1],
    ['NARC'=1],
    []).
```

B.318 orth_on_top_of_orth

♢ META-DATA:

```
ctr_date(
    orth_on_top_of_orth,
    ['20030820','20040726','20060812']).

ctr_origin(
    orth_on_top_of_orth,
    Used for defining %c.,
    [place_in_pyramid]).

ctr_types(
    orth_on_top_of_orth,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    orth_on_top_of_orth,
    ['ORTHOTOPE1'-'ORTHOTOPE',
     'ORTHOTOPE2'-'ORTHOTOPE',
     'VERTICAL_DIM'-int]).

ctr_restrictions(
    orth_on_top_of_orth,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     'VERTICAL_DIM'>=1,
     'VERTICAL_DIM'=<size('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2')]).

ctr_example(
    orth_on_top_of_orth,
    orth_on_top_of_orth(
        [[ori-5,siz-2,end-7],[ori-3,siz-3,end-6]],
        [[ori-3,siz-5,end-8],[ori-1,siz-2,end-3]],
        2)).

ctr_typical(
    orth_on_top_of_orth,
    [size('ORTHOTOPE')>1,'ORTHOTOPE'^siz>0]).
```

```
ctr_graph(
    orth_on_top_of_orth,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT'(=)>>collection(orthotope1,orthotope2)],
    [orthotope1^key=\='VERTICAL_DIM',
     orthotope2^ori=<orthotope1^ori,
     orthotope1^end=<orthotope2^end],
    ['NARC'=size('ORTHOTOPE1')-1],
    []).

ctr_graph(
    orth_on_top_of_orth,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT'(=)>>collection(orthotope1,orthotope2)],
    [orthotope1^key='VERTICAL_DIM',
     orthotope1^ori=orthotope2^end],
    ['NARC'=1],
    []).
```

B.319 orths_are_connected

♢ META-DATA:

```
ctr_date(
    orths_are_connected,
    ['20000128','20030820','20060812']).

ctr_origin(orths_are_connected,'N.~Beldiceanu',[]).

ctr_types(
    orths_are_connected,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    orths_are_connected,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE')]).

ctr_restrictions(
    orths_are_connected,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     required('ORTHOTOPES',orth),
     same_size('ORTHOTOPES',orth)]).

ctr_example(
    orths_are_connected,
    orths_are_connected(
        [[orth-[[ori-2,siz-4,end-6],[ori-2,siz-2,end-4]]],
         [orth-[[ori-1,siz-2,end-3],[ori-4,siz-3,end-7]]],
         [orth-[[ori-6,siz-3,end-9],[ori-1,siz-2,end-3]]],
         [orth-[[ori-6,siz-2,end-8],[ori-3,siz-2,end-5]]]])).

ctr_typical(
    orths_are_connected,
    [size('ORTHOTOPE')>1,size('ORTHOTOPES')>1]).

ctr_exchangeable(
    orths_are_connected,
    [items('ORTHOTOPES',all),
     items_sync('ORTHOTOPES'^orth,all),
     translate(['ORTHOTOPES'^orth^ori,'ORTHOTOPES'^orth^end])]).

ctr_graph(
```

```
    orths_are_connected,
    ['ORTHOTOPES'],
    1,
    ['SELF'>>collection(orthotopes)],
    [orth_link_ori_siz_end(orthotopes^orth)],
    ['NARC'=size('ORTHOTOPES')],
    []).

ctr_graph(
    orths_are_connected,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE'(=\=)>>collection(orthotopes1,orthotopes2)],
    [two_orth_are_in_contact(
         orthotopes1^orth,
         orthotopes2^orth)],
    ['NVERTEX'=size('ORTHOTOPES'),'NCC'=1],
    []).

ctr_application(orths_are_connected,[1]).
```

B.320 overlap_sboxes

◊ META-DATA:

```
ctr_date(overlap_sboxes,['20070622','20090725']).

ctr_origin(
    overlap_sboxes,
    Geometry, derived from \cite{RandellCuiCohn92},
    []).

ctr_synonyms(overlap_sboxes,[overlap]).

ctr_types(
    overlap_sboxes,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int)]).

ctr_arguments(
    overlap_sboxes,
    ['K'-int,
     'DIMS'-sint,
     'OBJECTS'-collection(oid-int,sid-dvar,x-'VARIABLES'),
     'SBOXES'-collection(sid-int,t-'INTEGERS',l-'POSITIVES')]).

ctr_restrictions(
    overlap_sboxes,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     'K'>0,
     'DIMS'>=0,
     'DIMS'<'K',
     increasing_seq('OBJECTS',[oid]),
     required('OBJECTS',[oid,sid,x]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
```

```
     'OBJECTS'^sid=<size('SBOXES'),
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    overlap_sboxes,
    overlap_sboxes(
        2,
        {0,1},
        [[oid-1,sid-1,x-[[v-1],[v-1]]],
         [oid-2,sid-2,x-[[v-3],[v-2]]],
         [oid-3,sid-3,x-[[v-2],[v-4]]]],
        [[sid-1,t-[[v-0],[v-0]],l-[[v-4],[v-5]]],
         [sid-2,t-[[v-0],[v-0]],l-[[v-3],[v-3]]],
         [sid-3,t-[[v-0],[v-0]],l-[[v-2],[v-1]]]])).

ctr_typical(overlap_sboxes,[size('OBJECTS')>1]).

ctr_exchangeable(
    overlap_sboxes,
    [items('OBJECTS',all),
     items('SBOXES',all),
     items_sync('OBJECTS'^x,'SBOXES'^t,'SBOXES'^l,all),
     vals(['SBOXES'^l^v],int,<,dontcare,dontcare)]).

ctr_eval(overlap_sboxes,[logic(overlap_sboxes_g)]).

ctr_logic(
    overlap_sboxes,
    [DIMENSIONS,OIDS],
    [(origin(O1,S1,D)--->O1^x(D)+S1^t(D)),
     (end(O1,S1,D)--->O1^x(D)+S1^t(D)+S1^l(D)),
     (overlap_sboxes(Dims,O1,S1,O2,S2)--->
      forall(
          D,
          Dims,
          end(O1,S1,D)#>origin(O2,S2,D)#/\
          end(O2,S2,D)#>origin(O1,S1,D))),
     (overlap_objects(Dims,O1,O2)--->
      forall(
          S1,
          sboxes([O1^sid]),
          exists(
```

```
                S2,
                sboxes([O2^sid]),
                overlap_sboxes(Dims,O1,S1,O2,S2)))),
       (all_overlap(Dims,OIDS)--->
        forall(
            O1,
            objects(OIDS),
            forall(
                O2,
                objects(OIDS),
                O1^oid#<O2^oid#=>overlap_objects(Dims,O1,O2)))),
     all_overlap(DIMENSIONS,OIDS)]).

ctr_contractible(overlap_sboxes,[],'OBJECTS',suffix).

ctr_application(overlap_sboxes,[3]).

overlap_sboxes_g(K,_39114,[],_39116) :-
        !,
        check_type(int_gteq(1),K).

overlap_sboxes_g(K,_DIMS,OBJECTS,SBOXES) :-
        length(OBJECTS,O),
        length(SBOXES,S),
        O>0,
        S>0,
        check_type(int_gteq(1),K),
        collection(OBJECTS,[int(1,O),dvar(1,S),col(K,[dvar])]),
        collection(
            SBOXES,
            [int(1,S),col(K,[int]),col(K,[int_gteq(1)])]),
        get_attr1(OBJECTS,OIDS),
        get_attr2(OBJECTS,SIDS),
        get_col_attr3(OBJECTS,1,XS),
        get_attr1(SBOXES,SIDES),
        get_col_attr2(SBOXES,1,TS),
        get_col_attr3(SBOXES,1,TL),
        collection_increasing_seq(OBJECTS,[1]),
        geost1(OIDS,SIDS,XS,Objects),
        geost2(SIDES,TS,TL,Sboxes),
        geost_dims(1,K,DIMENSIONS),
        ctr_logic(overlap_sboxes,[DIMENSIONS,OIDS],Rules),
        geost(Objects,Sboxes,[overlap(true)],Rules).
```

B.321 path

◊ **META-DATA:**

```
ctr_date(path,['20090101','20120219']).

ctr_origin(path,'Derived from %c.',[binary_tree]).

ctr_arguments(
    path,
    ['NPATH'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    path,
    ['NPATH'>=1,
     'NPATH'=<size('NODES'),
     required('NODES',[index,succ]),
     size('NODES')>0,
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    path,
    [path(
         3,
         [[index-1,succ-1],
          [index-2,succ-3],
          [index-3,succ-5],
          [index-4,succ-7],
          [index-5,succ-1],
          [index-6,succ-6],
          [index-7,succ-7],
          [index-8,succ-6]]),
     path(
         1,
         [[index-1,succ-8],
          [index-2,succ-7],
          [index-3,succ-6],
          [index-4,succ-5],
          [index-5,succ-5],
          [index-6,succ-4],
          [index-7,succ-3],
          [index-8,succ-2]]),
```

```
     path(
         8,
         [[index-1,succ-1],
          [index-2,succ-2],
          [index-3,succ-3],
          [index-4,succ-4],
          [index-5,succ-5],
          [index-6,succ-6],
          [index-7,succ-7],
          [index-8,succ-8]])]).

ctr_typical(path,['NPATH'<size('NODES'),size('NODES')>1]).

ctr_exchangeable(path,[items('NODES',all)]).

ctr_graph(
    path,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1,'NCC'='NPATH','MAX_ID'=<1],
    ['ONE_SUCC']).

ctr_eval(path,[reformulation(path_r),checker(path_c)]).

ctr_functional_dependency(path,1,[2]).

ctr_application(path,[2]).

ctr_sol(path,2,0,2,3,[1-2,2-1]).

ctr_sol(path,3,0,3,13,[1-6,2-6,3-1]).

ctr_sol(path,4,0,4,73,[1-24,2-36,3-12,4-1]).

ctr_sol(path,5,0,5,501,[1-120,2-240,3-120,4-20,5-1]).

ctr_sol(path,6,0,6,4051,[1-720,2-1800,3-1200,4-300,5-30,6-1]).

ctr_sol(
    path,
    7,
    0,
    7,
    37633,
```

```
    [1-5040,2-15120,3-12600,4-4200,5-630,6-42,7-1]).

ctr_sol(
    path,
    8,
    0,
    8,
    394353,
    [1-40320,
     2-141120,
     3-141120,
     4-58800,
     5-11760,
     6-1176,
     7-56,
     8-1]).

path_r(NPATH,NODES) :-
        eval(tree(NPATH,NODES)),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        k_ary_tree(INDEXES,INDEXES,SUCCS,1).

path_c(NPATH,NODES) :-
        length(NODES,N),
        check_type(dvar(1,N),NPATH),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,IND),
        sort(IND,SIND),
        length(SIND,N),
        get_attr12(NODES,IND_SUCC),
        keysort(IND_SUCC,SIND_SUCC),
        remove_key_from_collection(SIND_SUCC,Succ),
        (foreach(S,Succ),
         foreach(S-P,L1),
         foreach(_79261,Sink),count(P,1,_79268)do
         true),
        keysort(L1,L2),
        keyclumped(L2,L3),
        (foreach(Su-Ps,L3),param(Sink)do
         Ps=[Su]->nth1(Su,Sink,Su);
         Ps=[Other]->nth1(Su,Sink,X),nth1(Other,Sink,X);
         Ps=[Su,Other]->nth1(Su,Sink,Su),nth1(Other,Sink,Su);
         Ps=[Other,Su]->nth1(Su,Sink,Su),nth1(Other,Sink,Su)),
        ground(Sink),
        sort(Sink,Sinks),
```

```
length(Sinks,NPATH).
```

B.322 path_from_to

◇ **META-DATA:**

```
ctr_date(path_from_to,['20030820','20040530','20060812']).

ctr_origin(
    path_from_to,
    \cite{AlthausBockmayrElfKasperJungerMehlhorn02},
    []).

ctr_usual_name(path_from_to,path).

ctr_arguments(
    path_from_to,
    ['FROM'-int,
     'TO'-int,
     'NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    path_from_to,
    ['FROM'>=1,
     'FROM'=<size('NODES'),
     'TO'>=1,
     'TO'=<size('NODES'),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    path_from_to,
    path_from_to(
        4,
        3,
        [[index-1,succ-{}],
         [index-2,succ-{}],
         [index-3,succ-{5}],
         [index-4,succ-{5}],
         [index-5,succ-{2,3}]])).

ctr_typical(path_from_to,['FROM'=\='TO',size('NODES')>2]).

ctr_exchangeable(path_from_to,[items('NODES',all)]).
```

```
ctr_graph(
    path_from_to,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ],
    ['PATH_FROM_TO'(index,'FROM','TO')=1],
    []).

ctr_application(path_from_to,[3]).
```

B.323 pattern

◇ **META-DATA:**

```
ctr_date(pattern,['20031008','20090717']).

ctr_origin(pattern,'\\cite{BourdaisGalinierPesant03}',[]).

ctr_types(pattern,['PATTERN'-collection(var-int)]).

ctr_arguments(
    pattern,
    ['VARIABLES'-collection(var-dvar),
     'PATTERNS'-collection(pat-'PATTERN')]).

ctr_restrictions(
    pattern,
    [required('PATTERN',var),
     'PATTERN'^var>=0,
     change(0,'PATTERN',=),
     size('PATTERN')>1,
     required('VARIABLES',var),
     required('PATTERNS',pat),
     size('PATTERNS')>0,
     same_size('PATTERNS',pat)]).

ctr_example(
    pattern,
    pattern(
        [[var-1],
         [var-1],
         [var-2],
         [var-2],
         [var-2],
         [var-1],
         [var-3],
         [var-3]],
        [[pat-[[var-1],[var-2],[var-1]]],
         [pat-[[var-1],[var-2],[var-3]]],
         [pat-[[var-2],[var-1],[var-3]]]])).

ctr_typical(
    pattern,
    [size('VARIABLES')>2,range('VARIABLES'^var)>1]).

ctr_exchangeable(
```

```
    pattern,
    [items('PATTERNS',all),
     items_sync('VARIABLES','PATTERNS'^pat,reverse),
     vals(
         ['VARIABLES'^var,'PATTERNS'^pat^var],
         int,
         =\=,
         all,
         dontcare)]).

ctr_eval(pattern,[automaton(pattern_a)]).

ctr_contractible(pattern,[],'VARIABLES',prefix).

ctr_contractible(pattern,[],'VARIABLES',suffix).

pattern_a(FLAG,VARIABLES,PATTERNS) :-
        collection(VARIABLES,[dvar]),
        collection(PATTERNS,[col([int_gteq(0)])]),
        same_size(PATTERNS),
        length(PATTERNS,NPATTERNS),
        NPATTERNS>0,
        get_attr1(VARIABLES,VARS),
        get_col_attr1(PATTERNS,1,PATTS),
        PATTS=[PATT|_32739],
        length(PATT,K),
        K>1,
        pattern_change(PATTERNS),
        remove_duplicates(PATTS,PATTS_NO_DUPLICATES),
        pattern_build_tree(
            PATTS_NO_DUPLICATES,
            ID_PATTS,
            node(-1-0,[]),
            1,
            _35612,
            TREE),
        flattern(PATTS_NO_DUPLICATES,FLAT_PATTS),
        remove_duplicates(FLAT_PATTS,VALUES),
        pattern_next(
            ID_PATTS,
            ID_PATTS,
            VALUES,
            ADDITIONAL_TRANSITIONS),
        pattern_gen_states(TREE,STATES,TRANSITIONS),
        append(
            TRANSITIONS,
```

```
            ADDITIONAL_TRANSITIONS,
            ALL_TRANSITIONS),
        AUTOMATON=
        automaton(
            VARS,
            _37711,
            VARS,
            STATES,
            ALL_TRANSITIONS,
            [],
            [],
            []),
        automaton_bool(FLAG,VALUES,AUTOMATON).

pattern_gen_states(
    node(-1-0,LIST_SUNS),
    [source(NAME),sink(NAME)|R],
    TRANSITIONS) :-
        !,
        number_codes(-1,CODE),
        atom_codes(ATOM,CODE),
        atom_concat(s,ATOM,NAME),
        pattern_gen_states1(LIST_SUNS,-1,R,TRANSITIONS).

pattern_gen_states(
    node(ID-_VAL,LIST_SUNS),
    [sink(NAME)|R],
    TRANSITIONS) :-
        ID>=0,
        number_codes(ID,IDCODE),
        atom_codes(IDATOM,IDCODE),
        atom_concat(s,IDATOM,NAME),
        pattern_gen_states1(LIST_SUNS,ID,R,TRANSITIONS).

pattern_gen_states1([],_32663,[],[]).

pattern_gen_states1([N|R],ID1,ST,TRANSITIONS) :-
        N=node(ID2-VAL2,_32689),
        pattern_gen_states(N,S,TRANSITIONS1),
        pattern_gen_states1(R,ID1,T,TRANSITIONS2),
        append(S,T,ST),
        number_codes(ID1,IDCODE1),
        atom_codes(IDATOM1,IDCODE1),
        atom_concat(s,IDATOM1,IDNAME1),
        number_codes(ID2,IDCODE2),
        atom_codes(IDATOM2,IDCODE2),
```

```
        atom_concat(s,IDATOM2,IDNAME2),
        append(
            [arc(IDNAME1,VAL2,IDNAME2),
             arc(IDNAME2,VAL2,IDNAME2)],
            TRANSITIONS1,
            T1),
        append(T1,TRANSITIONS2,TRANSITIONS).

pattern_change([]).

pattern_change([[_32671-P]|R]) :-
        eval(change(0,P,=)),
        pattern_change(R).

pattern_next([],_32663,_32664,[]).

pattern_next([PID-P|R],ID_PATTS,VALUES,TRANSITIONS) :-
        P=[_32692|RP],
        pattern_next1(VALUES,PID,RP,ID_PATTS,TRANSITIONS1),
        pattern_next(R,ID_PATTS,VALUES,TRANSITIONS2),
        append(TRANSITIONS1,TRANSITIONS2,TRANSITIONS).

pattern_next1([],_32663,_32664,_32665,[]).

pattern_next1(
    [V|R],
    PID,
    RP,
    ID_PATTS,
    [arc(PIDNAME,V,NEWPIDNAME)|S]) :-
        append(RP,[V],NEWP),
        pattern_search(ID_PATTS,NEWP,NEWPID),
        number_codes(PID,PIDCODE),
        atom_codes(PIDATOM,PIDCODE),
        atom_concat(s,PIDATOM,PIDNAME),
        number_codes(NEWPID,NEWPIDCODE),
        atom_codes(NEWPIDATOM,NEWPIDCODE),
        atom_concat(s,NEWPIDATOM,NEWPIDNAME),
        !,
        pattern_next1(R,PID,RP,ID_PATTS,S).

pattern_next1([_32670|R],PID,RP,ID_PATTS,S) :-
        pattern_next1(R,PID,RP,ID_PATTS,S).

pattern_search([ID-PAT|_32669],PAT,ID) :-
        !.
```

```
pattern_search([_32668|R],PAT,ID) :-
        pattern_search(R,PAT,ID).

pattern_build_tree([],[],TREE,NODE_ID,NODE_ID,TREE).

pattern_build_tree(
    [PATTERN|R],
    [PATTERN_ID-PATTERN|S],
    OLD_TREE,
    OLD_NODE_ID,
    NEW_NODE_ID,
    NEW_TREE) :-
        pattern_insert(
            PATTERN,
            OLD_TREE,
            OLD_NODE_ID,
            CUR_NODE_ID,
            CUR_TREE,
            PATTERN_ID),
        pattern_build_tree(
            R,
            S,
            CUR_TREE,
            CUR_NODE_ID,
            NEW_NODE_ID,
            NEW_TREE).

pattern_insert([],TREE,NODE_ID,NODE_ID,TREE,_32667).

pattern_insert(
    [I|R],
    OLD_TREE,
    OLD_NODE_ID,
    NEW_NODE_ID,
    node(LABEL,NEW_TREE),
    PATTERN_ID) :-
        OLD_TREE=node(LABEL,LIST_NODES),
        pattern_occurs(I,LIST_NODES,[],BEFORE,SUBTREE,AFTER),
        !,
        pattern_insert(
            R,
            SUBTREE,
            OLD_NODE_ID,
            NEW_NODE_ID,
            NEW_SUBTREE,
```

```
            PATTERN_ID),
        append(BEFORE,[NEW_SUBTREE],TEMPO_TREE),
        append(TEMPO_TREE,AFTER,NEW_TREE).

pattern_insert(
    [I|R],
    node(LABEL,LIST_NODES),
    OLD_NODE_ID,
    NEW_NODE_ID,
    node(LABEL,[BRANCH|LIST_NODES]),
    PATTERN_ID) :-
        pattern_create_branch(
            [I|R],
            OLD_NODE_ID,
            NEW_NODE_ID,
            BRANCH,
            PATTERN_ID).

pattern_create_branch(
    [I],
    OLD_NODE_ID,
    NEW_NODE_ID,
    node(OLD_NODE_ID-I,[]),
    OLD_NODE_ID) :-
        !,
        NEW_NODE_ID is OLD_NODE_ID+1.

pattern_create_branch(
    [I,J|R],
    OLD_NODE_ID,
    NEW_NODE_ID,
    node(OLD_NODE_ID-I,[S]),
    PATTERN_ID) :-
        CUR_NODE_ID is OLD_NODE_ID+1,
        pattern_create_branch(
            [J|R],
            CUR_NODE_ID,
            NEW_NODE_ID,
            S,
            PATTERN_ID).

pattern_occurs(
    I,
    [node(Id-I,L)|AFTER],
    BEFORE,
    BEFORE,
```

```
    node(Id-I,L),
    AFTER) :-
        !.

pattern_occurs(
    I,
    [NODE|AFTER_CUR],
    BEFORE_CUR,
    BEFORE,
    NODE_FOUND,
    AFTER) :-
        pattern_occurs(
            I,
            AFTER_CUR,
            [NODE|BEFORE_CUR],
            BEFORE,
            NODE_FOUND,
            AFTER).
```

B.324 peak

◇ META-DATA:

```
ctr_date(peak,['20040530']).

ctr_origin(peak,'Derived from %c.',[inflexion]).

ctr_arguments(peak,['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    peak,
    ['N'>=0,
     2*'N'=<max(size('VARIABLES')-1,0),
     required('VARIABLES',var)]).

ctr_example(
    peak,
    [peak(
         2,
         [[var-1],
          [var-1],
          [var-4],
          [var-8],
          [var-6],
          [var-2],
          [var-7],
          [var-1]]),
     peak(
         0,
         [[var-1],
          [var-1],
          [var-4],
          [var-4],
          [var-4],
          [var-6],
          [var-7],
          [var-7]]),
     peak(
         4,
         [[var-1],
          [var-5],
          [var-4],
          [var-9],
          [var-4],
          [var-6],
```

```
        [var-2],
        [var-7],
        [var-6]])]).

ctr_typical(
    peak,
    [size('VARIABLES')>2,range('VARIABLES'^var)>1]).

ctr_typical_model(peak,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    peak,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    peak,
    [checker(peak_c),
     automaton(peak_a),
     automaton_with_signature(peak_a_s)]).

ctr_pure_functional_dependency(peak,[]).

ctr_functional_dependency(peak,1,[2]).

ctr_contractible(peak,['N'=0],'VARIABLES',any).

ctr_cond_imply(
    peak,
    atleast_nvalue,
    ['N'>0],
    ['NVAL'=2],
    [none,'VARIABLES']).

ctr_cond_imply(
    peak,
    inflexion,
    [],
    ['N'=peak('VARIABLES'^var)+valley('VARIABLES'^var)],
    [none,'VARIABLES']).

ctr_sol(peak,2,0,2,9,[0-9]).

ctr_sol(peak,3,0,3,64,[0-50,1-14]).

ctr_sol(peak,4,0,4,625,[0-295,1-330]).
```

```
ctr_sol(peak,5,0,5,7776,[0-1792,1-5313,2-671]).

ctr_sol(peak,6,0,6,117649,[0-11088,1-73528,2-33033]).

ctr_sol(
    peak,
    7,
    0,
    7,
    2097152,
    [0-69498,1-944430,2-1010922,3-72302]).

ctr_sol(
    peak,
    8,
    0,
    8,
    43046721,
    [0-439791,1-11654622,2-24895038,3-6057270]).

peak_c(N,VARIABLES) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        peak_c(VARS,s,0,N).

peak_c([V1,V2|R],s,C,N) :-
        V1>=V2,
        !,
        peak_c([V2|R],s,C,N).

peak_c([_V1,V2|R],s,C,N) :-
        !,
        peak_c([V2|R],u,C,N).

peak_c([V1,V2|R],u,C,N) :-
        V1=<V2,
        !,
        peak_c([V2|R],u,C,N).

peak_c([_V1,V2|R],u,C,N) :-
        !,
        C1 is C+1,
        peak_c([V2|R],s,C1,N).

peak_c([_52973],_52970,N,N) :-
```

```
        !.

peak_c([],_52967,N,N).

peak_counters_check([V1,V2|R],s,C,[C|S]) :-
        V1>=V2,
        !,
        peak_counters_check([V2|R],s,C,S).

peak_counters_check([_V1,V2|R],s,C,[C|S]) :-
        !,
        peak_counters_check([V2|R],u,C,S).

peak_counters_check([V1,V2|R],u,C,[C|S]) :-
        V1=<V2,
        !,
        peak_counters_check([V2|R],u,C,S).

peak_counters_check([_V1,V2|R],u,C,[C1|S]) :-
        !,
        C1 is C+1,
        peak_counters_check([V2|R],s,C1,S).

peak_counters_check([V|R],init,C,[0|S]) :-
        !,
        peak_counters_check([V|R],s,C,S).

peak_counters_check([_52970],_52967,_52968,[]).

ctr_automaton_signature(
    peak,
    peak_a,
    pair_signature(2,signature)).

peak_a(FLAG,N,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        peak_a_s(FLAG,N,VARIABLES,SIGNATURE).

peak_a_s(FLAG,N,VARIABLES,SIGNATURE) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        MAX is max(L-1,0),
        2*N#=<MAX,
        automaton(
            SIGNATURE,
```

```
        _54598,
        SIGNATURE,
        [source(s),sink(u),sink(s)],
        [arc(s,2,s),
         arc(s,1,s),
         arc(s,0,u),
         arc(u,2,s,[C+1]),
         arc(u,1,u),
         arc(u,0,u)],
        [C],
        [0],
        [COUNT]),
    COUNT#=N#<=>FLAG.
```

B.325 period

◇ **META-DATA:**

```
ctr_predefined(period).

ctr_date(period,['20000128','20030820','20040530','20060812']).

ctr_origin(period,'N.~Beldiceanu',[]).

ctr_arguments(
    period,
    ['PERIOD'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    period,
    ['PERIOD'>=1,
     'PERIOD'=<size('VARIABLES'),
     required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    period,
    period(
        3,
        [[var-1],
         [var-1],
         [var-4],
         [var-1],
         [var-1],
         [var-4],
         [var-1],
         [var-1]],
        =)).

ctr_typical(
    period,
    ['PERIOD'>1,
     'PERIOD'<size('VARIABLES'),
     size('VARIABLES')>2,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=])]).

ctr_exchangeable(
```

```
    period,
    [items('VARIABLES',reverse),
     items('VARIABLES',shift),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_eval(period,[checker(period_c),reformulation(period_r)]).

ctr_pure_functional_dependency(period,[]).

ctr_functional_dependency(period,1,[2,3]).

ctr_contractible(
    period,
    [in_list('CTR',[=]),'PERIOD'=1],
    VARIABLES,
    any).

ctr_contractible(period,[],'VARIABLES',prefix).

ctr_contractible(period,[],'VARIABLES',suffix).

period_c(PERIOD,VARIABLES,CTR) :-
        check_type(dvar,PERIOD),
        collection(VARIABLES,[int]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        length(VARIABLES,N),
        PERIOD#>=1,
        PERIOD#=<N,
        fd_min(PERIOD,PMin),
        fd_max(PERIOD,PMax),
        get_attr1(VARIABLES,VARS),
        (   integer(PERIOD) ->
            MinPeriod is PMin
        ;   memberchk(CTR,[=]) ->
            VARS=[V|RVARS],
            compute_occ_consecutive_identical_values(
                RVARS,
                V,
                1,
                OCCS),
            compute_max_sliding2(OCCS,0,MinPeriod)
        ;   MinPeriod is PMin
        ),
        P in MinPeriod..PMax,
        indomain(P),
        PERIOD=P,
```

```
        (   P=N ->
            true
        ;   M is min(N,P),
            append_length(FIRSTS,REST,VARS,M),
            period_c(REST,FIRSTS,P,CTR)
        ),
        !.

compute_occ_consecutive_identical_values([V|R],Prev,Occ,Res) :-
        V=Prev,
        !,
        Occ1 is Occ+1,
        compute_occ_consecutive_identical_values(
            R,
            Prev,
            Occ1,
            Res).

compute_occ_consecutive_identical_values(
    [V|R],
    Prev,
    Occ,
    [Occ|S]) :-
        V=\=Prev,
        !,
        compute_occ_consecutive_identical_values(R,V,1,S).

compute_occ_consecutive_identical_values([],_28967,Occ,[Occ]).

compute_max_sliding2([],Max,Max) :-
        !.

compute_max_sliding2([_28972],Max,Max) :-
        !.

compute_max_sliding2([O1,O2|R],MaxCur,Max) :-
        M is O1+O2,
        (   M>MaxCur ->
            compute_max_sliding2([O2|R],M,Max)
        ;   compute_max_sliding2([O2|R],MaxCur,Max)
        ).

period_c([],_28970,_28971,_28972) :-
        !.

period_c(VARS,FIRSTS,P,CTR) :-
```

```
        length(VARS,N),
        M is min(N,P),
        append_length(NEXTS,REST,VARS,M),
        period_compare(CTR,FIRSTS,NEXTS),
        period_c(REST,NEXTS,P,CTR).

period_compare(_28969,[],_28971) :-
        !.

period_compare(_28969,_28970,[]) :-
        !.

period_compare(=,[U|R],[V|S]) :-
        !,
        U=V,
        period_compare(=,R,S).

period_compare(=\=,[U|R],[V|S]) :-
        !,
        U=\=V,
        period_compare(=\=,R,S).

period_compare(<,[U|R],[V|S]) :-
        !,
        U<V,
        period_compare(<,R,S).

period_compare(>=,[U|R],[V|S]) :-
        !,
        U>=V,
        period_compare(>=,R,S).

period_compare(>,[U|R],[V|S]) :-
        !,
        U>V,
        period_compare(>,R,S).

period_compare(=<,[U|R],[V|S]) :-
        U=<V,
        period_compare(=<,R,S).

period_r(PERIOD,VARIABLES,CTR) :-
        check_type(dvar,PERIOD),
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        length(VARIABLES,N),
```

```
PERIOD#>=1,
PERIOD#=<N,
get_attr1(VARIABLES,VARS),
period1(N,VARS,LISTS),
period4(LISTS,1,CTR,BOOLS),
reverse(BOOLS,RBOOLS),
period7(RBOOLS,1,PERIOD,1,EXPR),
call(EXPR).
```

B.326 period_except_0

◇ META-DATA:

```
ctr_predefined(period_except_0).

ctr_date(period_except_0,['20030820','20040530','20060813']).

ctr_origin(period_except_0,'Derived from %c.',[period]).

ctr_arguments(
    period_except_0,
    ['PERIOD'-dvar,
     'VARIABLES'-collection(var-dvar),
     'CTR'-atom]).

ctr_restrictions(
    period_except_0,
    ['PERIOD'>=1,
     'PERIOD'=<size('VARIABLES'),
     required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    period_except_0,
    period_except_0(
        3,
        [[var-1],
         [var-1],
         [var-4],
         [var-1],
         [var-1],
         [var-0],
         [var-1],
         [var-1]],
        =)).

ctr_typical(
    period_except_0,
    ['PERIOD'>1,
     'PERIOD'<size('VARIABLES'),
     size('VARIABLES')>2,
     range('VARIABLES'^var)>1,
     atleast(1,'VARIABLES',0),
     in_list('CTR',[=])]).
```

```
ctr_typical_model(period_except_0,[atleast(2,'VARIABLES',0)]).

ctr_exchangeable(
    period_except_0,
    [items('VARIABLES',reverse),
     items('VARIABLES',shift),
     vals(['VARIABLES'^var],int(=\=(0)),=\=,all,dontcare)]).

ctr_eval(
    period_except_0,
    [checker(period_except_0_c),
     reformulation(period_except_0_r)]).

ctr_pure_functional_dependency(period_except_0,[]).

ctr_functional_dependency(period_except_0,1,[2,3]).

ctr_contractible(
    period_except_0,
    [in_list('CTR',[=]),'PERIOD'=1],
    VARIABLES,
    any).

ctr_contractible(period_except_0,[],'VARIABLES',prefix).

ctr_contractible(period_except_0,[],'VARIABLES',suffix).

period_except_0_c(PERIOD,VARIABLES,CTR) :-
        check_type(dvar,PERIOD),
        collection(VARIABLES,[int]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        length(VARIABLES,N),
        PERIOD#>=1,
        PERIOD#=<N,
        fd_min(PERIOD,PMin),
        fd_max(PERIOD,PMax),
        get_attr1(VARIABLES,VARS),
        P in PMin..PMax,
        indomain(P),
        PERIOD=P,
        (   P=N ->
            true
        ;   M is min(N,P),
            append_length(FIRSTS,REST,VARS,M),
            period_except_0_c(REST,FIRSTS,P,CTR)
        ),
```

```
        !.

period_except_0_c([],_29808,_29809,_29810) :-
        !.

period_except_0_c(VARS,FIRSTS,P,CTR) :-
        length(VARS,N),
        M is min(N,P),
        append_length(NEXTS,REST,VARS,M),
        period_compare_except_0(CTR,FIRSTS,NEXTS),
        period_except_0_c(REST,NEXTS,P,CTR).

period_compare_except_0(_29807,[],_29809) :-
        !.

period_compare_except_0(_29807,_29808,[]) :-
        !.

period_compare_except_0(CTR,[0|R],[_29816|S]) :-
        !,
        period_compare_except_0(CTR,R,S).

period_compare_except_0(CTR,[_29812|R],[0|S]) :-
        !,
        period_compare_except_0(CTR,R,S).

period_compare_except_0(=,[U|R],[V|S]) :-
        !,
        U=V,
        period_compare_except_0(=,R,S).

period_compare_except_0(=\=,[U|R],[V|S]) :-
        !,
        U=\=V,
        period_compare_except_0(=\=,R,S).

period_compare_except_0(<,[U|R],[V|S]) :-
        !,
        U<V,
        period_compare_except_0(<,R,S).

period_compare_except_0(>=,[U|R],[V|S]) :-
        !,
        U>=V,
        period_compare_except_0(>=,R,S).
```

```
period_compare_except_0(>,[U|R],[V|S]) :-
        !,
        U>V,
        period_compare_except_0(>,R,S).

period_compare_except_0(=<,[U|R],[V|S]) :-
        U=<V,
        period_compare_except_0(=<,R,S).

period_except_0_r(PERIOD,VARIABLES,CTR) :-
        check_type(dvar,PERIOD),
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        length(VARIABLES,N),
        PERIOD#>=1,
        PERIOD#=<N,
        get_attr1(VARIABLES,VARS),
        period1(N,VARS,LISTS),
        period4(LISTS,0,CTR,BOOLS),
        reverse(BOOLS,RBOOLS),
        period7(RBOOLS,1,PERIOD,1,EXPR),
        call(EXPR).
```

B.327 period_vectors

◇ META-DATA:

```
ctr_predefined(period_vectors).

ctr_date(period_vectors,['20110614']).

ctr_origin(period_vectors,'Derived from %c',[period]).

ctr_types(
    period_vectors,
    ['VECTOR'-collection(var-dvar),'CTR'-atom]).

ctr_arguments(
    period_vectors,
    ['PERIOD'-dvar,
     'VECTORS'-collection(vec-'VECTOR'),
     'CTRS'-collection(ctr-'CTR')]).

ctr_restrictions(
    period_vectors,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     in_list('CTR',[=,=\=,<,>=,>,=<]),
     'PERIOD'>=1,
     'PERIOD'=<size('VECTORS'),
     required('VECTORS',vec),
     same_size('VECTORS',vec),
     required('CTRS',ctr),
     size('CTRS')=size('VECTOR')]).

ctr_example(
    period_vectors,
    period_vectors(
        3,
        [[vec-[[var-1],[var-0]]],
         [vec-[[var-1],[var-5]]],
         [vec-[[var-4],[var-4]]],
         [vec-[[var-1],[var-0]]],
         [vec-[[var-1],[var-5]]],
         [vec-[[var-4],[var-4]]],
         [vec-[[var-1],[var-0]]],
         [vec-[[var-1],[var-5]]]],
        [[ctr- =],[ctr- =]])).
```

```
ctr_typical(
    period_vectors,
    [in_list('CTR',[=]),
     size('VECTOR')>1,
     'PERIOD'>1,
     'PERIOD'<size('VECTORS'),
     size('VECTORS')>2]).

ctr_exchangeable(period_vectors,[items('VECTORS',reverse)]).

ctr_eval(period_vectors,[reformulation(period_vectors_r)]).

ctr_pure_functional_dependency(period_vectors,[]).

ctr_functional_dependency(period_vectors,1,[2,3]).

ctr_contractible(period_vectors,[],'VECTORS',prefix).

ctr_contractible(period_vectors,[],'VECTORS',suffix).

period_vectors_r(PERIOD,VECTORS,CTRS) :-
        check_type(dvar,PERIOD),
        collection(VECTORS,[col([dvar])]),
        collection(CTRS,[atom([=,=\=,<,>=,>,=<])]),
        length(VECTORS,N),
        PERIOD#>=1,
        PERIOD#=<N,
        get_attr11(VECTORS,VECTS),
        get_attr1(CTRS,LCTRS),
        period1(N,VECTS,LISTS),
        period4(LISTS,2,LCTRS,BOOLS),
        reverse(BOOLS,RBOOLS),
        period7(RBOOLS,1,PERIOD,1,EXPR),
        call(EXPR).
```

B.328 permutation

◇ **META-DATA:**

```
ctr_date(permutation,['20111210']).

ctr_origin(
    permutation,
    Derived from %c.,
    [alldifferent_consecutive_values]).

ctr_arguments(permutation,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    permutation,
    [required('VARIABLES',var),
     minval('VARIABLES'^var)=1,
     maxval('VARIABLES'^var)=size('VARIABLES')]).

ctr_example(
    permutation,
    permutation([[var-3],[var-2],[var-1],[var-4]])).

ctr_typical(permutation,[size('VARIABLES')>2]).

ctr_exchangeable(
    permutation,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in)]).

ctr_graph(
    permutation,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['MAX_NSCC'=<1],
    ['ONE_SUCC']).

ctr_eval(
    permutation,
    [checker(permutation_c),reformulation(permutation_r)]).

ctr_cond_imply(
    permutation,
    balance,
```

```
    [],
    ['BALANCE'=0],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    change,
    [],
    ['NCHANGE'=size('VARIABLES')-1,in_list('CTR',[=\=])],
    [none,'VARIABLES',none]).

ctr_cond_imply(
    permutation,
    circular_change,
    [],
    ['NCHANGE'=size('VARIABLES'),in_list('CTR',[=\=])],
    [none,'VARIABLES',none]).

ctr_cond_imply(
    permutation,
    length_last_sequence,
    [],
    ['LEN'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    length_first_sequence,
    [],
    ['LEN'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    longest_change,
    [],
    ['SIZE'=size('VARIABLES'),in_list('CTR',[=\=])],
    [none,'VARIABLES',none]).

ctr_cond_imply(
    permutation,
    max_n,
    [],
    ['MAX'=size('VARIABLES')-'RANK'],
    [none,none,'VARIABLES']).
```

```
ctr_cond_imply(
    permutation,
    min_n,
    [],
    ['MIN'='RANK'+1],
    [none,none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    min_nvalue,
    [],
    ['MIN'=1],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    min_size_full_zero_stretch,
    [],
    ['MINSIZE'=size('VARIABLES')],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    ninterval,
    [],
    [NVAL=
     (size('VARIABLES')+'SIZE_INTERVAL')/'SIZE_INTERVAL'],
    [none,'VARIABLES',none]).

ctr_cond_imply(
    permutation,
    range_ctr,
    [],
    [in_list('CTR',[=<]),'R'=size('VARIABLES')],
    ['VARIABLES',none,none]).

ctr_cond_imply(
    permutation,
    soft_alldifferent_ctr,
    [],
    [],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    soft_all_equal_max_var,
```

```
    [],
    ['N'=<size('VARIABLES')-1],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    soft_all_equal_min_var,
    [],
    ['N'>=size('VARIABLES')-1],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    sum_ctr,
    [],
    [in_list('CTR',[=]),
     'VAR'=size('VARIABLES')*(size('VARIABLES')+1)/2],
    ['VARIABLES',none,none]).

ctr_cond_imply(
    permutation,
    deepest_valley,
    [size('VARIABLES')>2,
     first('VARIABLES'^var)>minval('VARIABLES'^var),
     last('VARIABLES'^var)>minval('VARIABLES'^var)],
    ['DEPTH'=minval('VARIABLES'^var)],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    deepest_valley,
    [size('VARIABLES')>2,first('VARIABLES'^var)=1],
    ['DEPTH'=2],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    deepest_valley,
    [size('VARIABLES')>2,last('VARIABLES'^var)=1],
    ['DEPTH'=2],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    highest_peak,
    [size('VARIABLES')>2,
```

```
     first('VARIABLES'^var)<maxval('VARIABLES'^var),
     last('VARIABLES'^var)<maxval('VARIABLES'^var)],
    ['HEIGHT'=maxval('VARIABLES'^var)],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    highest_peak,
    [size('VARIABLES')>2,
     first('VARIABLES'^var)=size('VARIABLES')],
    ['HEIGHT'=size('VARIABLES')-1],
    [none,'VARIABLES']).

ctr_cond_imply(
    permutation,
    highest_peak,
    [size('VARIABLES')>2,
     last('VARIABLES'^var)=size('VARIABLES')],
    ['HEIGHT'=size('VARIABLES')-1],
    [none,'VARIABLES']).

ctr_sol(permutation,2,0,2,2,-).

ctr_sol(permutation,3,0,3,6,-).

ctr_sol(permutation,4,0,4,24,-).

ctr_sol(permutation,5,0,5,120,-).

ctr_sol(permutation,6,0,6,720,-).

ctr_sol(permutation,7,0,7,5040,-).

ctr_sol(permutation,8,0,8,40320,-).

ctr_sol(permutation,9,0,9,362880,-).

ctr_sol(permutation,10,0,10,3628800,-).

permutation_c([V,V|_61972]) :-
        !,
        fail.

permutation_c([]) :-
        !.
```

```
permutation_c(VARIABLES) :-
        length(VARIABLES,N),
        collection(VARIABLES,[int(1,N)]),
        get_attr1(VARIABLES,VARS),
        sort(VARS,SVARS),
        length(SVARS,N).

permutation_r([]) :-
        !.

permutation_r(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        all_different(VARS),
        minimum(1,VARS),
        length(VARIABLES,N),
        maximum(N,VARS).
```

B.329 place_in_pyramid

◇ META-DATA:

```
ctr_date(
    place_in_pyramid,
    ['20000128','20030820','20041230','20060813']).

ctr_origin(place_in_pyramid,'N.~Beldiceanu',[]).

ctr_types(
    place_in_pyramid,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    place_in_pyramid,
    ['ORTHOTOPES'-collection(orth-'ORTHOTOPE'),
     'VERTICAL_DIM'-int]).

ctr_restrictions(
    place_in_pyramid,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     required('ORTHOTOPES',orth),
     same_size('ORTHOTOPES',orth),
     'VERTICAL_DIM'>=1,
     diffn('ORTHOTOPES')]).

ctr_example(
    place_in_pyramid,
    place_in_pyramid(
        [[orth-[[ori-1,siz-3,end-4],[ori-1,siz-2,end-3]]],
         [orth-[[ori-1,siz-2,end-3],[ori-3,siz-3,end-6]]],
         [orth-[[ori-5,siz-6,end-11],[ori-1,siz-2,end-3]]],
         [orth-[[ori-5,siz-2,end-7],[ori-3,siz-2,end-5]]],
         [orth-[[ori-8,siz-3,end-11],[ori-3,siz-2,end-5]]],
         [orth-[[ori-8,siz-2,end-10],[ori-5,siz-2,end-7]]]],
        2)).

ctr_typical(
    place_in_pyramid,
    [size('ORTHOTOPE')>1,
     'ORTHOTOPE'^siz>0,
     size('ORTHOTOPES')>1]).
```

```
ctr_exchangeable(place_in_pyramid,[items('ORTHOTOPES',all)]).

ctr_graph(
    place_in_pyramid,
    ['ORTHOTOPES'],
    2,
    ['CLIQUE'>>collection(orthotopes1,orthotopes2)],
    [orthotopes1^key=orthotopes2^key#/\
     orth_on_the_ground(orthotopes1^orth,'VERTICAL_DIM')#\/
     orthotopes1^key=\=orthotopes2^key#/\
     orth_on_top_of_orth(
         orthotopes1^orth,
         orthotopes2^orth,
         VERTICAL_DIM)],
    ['NARC'=size('ORTHOTOPES')],
    []).

ctr_application(place_in_pyramid,[1]).
```

B.330 polyomino

◊ META-DATA:

```
ctr_date(polyomino,['20000128','20030820','20060813']).

ctr_origin(polyomino,'Inspired by \\cite{Golomb65}.',[]).

ctr_arguments(
    polyomino,
    [CELLS-
     collection(
         index-int,
         right-dvar,
         left-dvar,
         up-dvar,
         down-dvar)]).

ctr_restrictions(
    polyomino,
    ['CELLS'^index>=1,
     'CELLS'^index=<size('CELLS'),
     size('CELLS')>=1,
     required('CELLS',[index,right,left,up,down]),
     distinct('CELLS',index),
     'CELLS'^right>=0,
     'CELLS'^right=<size('CELLS'),
     'CELLS'^left>=0,
     'CELLS'^left=<size('CELLS'),
     'CELLS'^up>=0,
     'CELLS'^up=<size('CELLS'),
     'CELLS'^down>=0,
     'CELLS'^down=<size('CELLS')]).

ctr_example(
    polyomino,
    polyomino(
        [[index-1,right-0,left-0,up-2,down-0],
         [index-2,right-3,left-0,up-0,down-1],
         [index-3,right-0,left-2,up-4,down-0],
         [index-4,right-5,left-0,up-0,down-3],
         [index-5,right-0,left-4,up-0,down-0]])).

ctr_exchangeable(
    polyomino,
    [items('CELLS',all),
```

```
     attrs_sync('CELLS',[[index],[right,left],[up],[down]]),
     attrs_sync('CELLS',[[index],[right],[left],[up,down]]),
     attrs_sync('CELLS',[[index],[up,left,down,right]])]).

ctr_graph(
    polyomino,
    ['CELLS'],
    2,
    ['CLIQUE'(=\=)>>collection(cells1,cells2)],
    [cells1^right=cells2^index#/\
     cells2^left=cells1^index#\/
     cells1^left=cells2^index#/\
     cells2^right=cells1^index#\/
     cells1^up=cells2^index#/\cells2^down=cells1^index#\/
     cells1^down=cells2^index#/\cells2^up=cells1^index],
    ['NVERTEX'=size('CELLS'),'NCC'=1],
    []).

ctr_application(polyomino,[1]).
```

B.331 power

◇ META-DATA:

```
ctr_predefined(power).

ctr_date(power,['20070930']).

ctr_origin(power,'\\cite{DenmatGotliebDucasse07}',[]).

ctr_synonyms(power,[xexpyeqz]).

ctr_arguments(power,['X'-dvar,'N'-dvar,'Y'-dvar]).

ctr_restrictions(power,['X'>=0,'N'>=0,'Y'>=0]).

ctr_example(power,power(2,3,8)).

ctr_typical(power,['X'>1,'N'>1,'N'<5,'Y'>1]).

ctr_eval(power,[checker(power_c),reformulation(power_r)]).

ctr_pure_functional_dependency(power,[]).

ctr_functional_dependency(power,3,[1,2]).

power_c(X,0,Y) :-
        !,
        check_type(int_gteq(0),X),
        Y=1.

power_c(X,N,Y) :-
        check_type(int_gteq(0),X),
        check_type(int_gteq(0),N),
        check_type(int_gteq(0),Y),
        power_c(N,X,X,Y).

power_c(1,Cur,_26427,Y) :-
        !,
        Cur=Y.

power_c(N,Cur,X,Y) :-
        Next is Cur*X,
        N1 is N-1,
        power_c(N1,Next,X,Y).
```

```
power_r(X,0,Y) :-
        !,
        check_type(dvar_gteq(0),X),
        Y=1.

power_r(X,N,Y) :-
        check_type(dvar_gteq(0),X),
        check_type(dvar_gteq(0),N),
        check_type(dvar_gteq(0),Y),
        fd_min(N,Min),
        fd_max(N,Max),
        Min1 is max(1,Min),
        power1(0,Min1,Max,1,X,Y,N,Disj),
        call(Disj).

power1(I,_26426,Max,_26428,_26429,_26430,_26431,0) :-
        I>Max,
        !.

power1(I,Min,Max,P,X,Y,N,R) :-
        I<Min,
        !,
        I1 is I+1,
        power1(I1,Min,Max,P*X,X,Y,N,R).

power1(I,Min,Max,P,X,Y,N,P#=Y#/\N#=I#\/R) :-
        I>=Min,
        I=<Max,
        I1 is I+1,
        power1(I1,Min,Max,P*X,X,Y,N,R).
```

B.332 precedence

◇ META-DATA:

```
ctr_date(precedence,['20111015']).

ctr_origin(precedence,'Scheduling',[]).

ctr_arguments(
    precedence,
    ['TASKS'-collection(origin-dvar,duration-dvar)]).

ctr_restrictions(
    precedence,
    [required('TASKS',[origin,duration]),'TASKS'^duration>=0]).

ctr_example(
    precedence,
    precedence(
        [[origin-1,duration-3],
         [origin-4,duration-0],
         [origin-5,duration-2],
         [origin-8,duration-1]])).

ctr_typical(precedence,[size('TASKS')>2,'TASKS'^duration>=1]).

ctr_exchangeable(
    precedence,
    [vals(['TASKS'^duration],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'^origin])]).

ctr_graph(
    precedence,
    ['TASKS'],
    2,
    ['PATH'>>collection(tasks1,tasks2)],
    [tasks1^origin+tasks1^duration=<tasks2^origin],
    ['NARC'=size('TASKS')-1],
    []).

ctr_eval(
    precedence,
    [checker(precedence_c),reformulation(precedence_r)]).

ctr_contractible(precedence,[],'TASKS',any).
```

```
ctr_application(precedence,[1]).

precedence_r(TASKS) :-
        length(TASKS,N),
        N>1,
        collection(TASKS,[dvar,dvar_gteq(0)]),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        gen_precedences(ORIGINS,DURATIONS).

gen_precedences([_35581],[_35583]) :-
        !.

gen_precedences([O1,O2|R],[D1,D2|S]) :-
        O1+D1#=<O2,
        gen_precedences([O2|R],[D2|S]).

precedence_c(TASKS) :-
        length(TASKS,N),
        N>1,
        collection(TASKS,[int,int_gteq(0)]),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        gen_precedences_fix(ORIGINS,DURATIONS).

gen_precedences_fix([_35581],[_35583]) :-
        !.

gen_precedences_fix([O1,O2|R],[D1,D2|S]) :-
        E1 is O1+D1,
        E1=<O2,
        gen_precedences_fix([O2|R],[D2|S]).
```

B.333 product_ctr

◇ META-DATA:

```
ctr_date(product_ctr,['20030820','20060813','20070902']).

ctr_origin(product_ctr,'Arithmetic constraint.',[]).

ctr_arguments(
    product_ctr,
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'VAR'-dvar]).

ctr_restrictions(
    product_ctr,
    [required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    product_ctr,
    product_ctr([[var-2],[var-1],[var-4]],=,8)).

ctr_typical(
    product_ctr,
    [size('VARIABLES')>1,
     size('VARIABLES')<10,
     range('VARIABLES'^var)>1,
     'VARIABLES'^var=\=0,
     in_list('CTR',[=,<,>=,>,=<]),
     'VAR'=\=0]).

ctr_exchangeable(product_ctr,[items('VARIABLES',all)]).

ctr_graph(
    product_ctr,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['TRUE'],
    ['CTR'('PROD'('VARIABLES',var),'VAR')],
    []).

ctr_eval(
    product_ctr,
    [checker(product_ctr_c),reformulation(product_ctr_r)]).

ctr_pure_functional_dependency(
```

```
    product_ctr,
    [in_list('CTR',[=])]).

ctr_contractible(
    product_ctr,
    [in_list('CTR',[<,=<]),minval('VARIABLES'^var)>0],
    VARIABLES,
    any).

ctr_aggregate(product_ctr,[in_list('CTR',[=])],[union,id,*]).

product_ctr_r(VARIABLES,CTR,VAR) :-
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_prod_var(VARS,PROD),
        call_term_relop_value(PROD,CTR,VAR).

product_ctr_c(VARIABLES,=,VAR) :-
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        prodlist(VARS,VAR).

product_ctr_c(VARIABLES,=\=,VAR) :-
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        prodlist(VARS,PROD),
        PROD=\=VAR.

product_ctr_c(VARIABLES,<,VAR) :-
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        prodlist(VARS,PROD),
        PROD<VAR.

product_ctr_c(VARIABLES,>=,VAR) :-
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        prodlist(VARS,PROD),
        PROD>=VAR.
```

```
product_ctr_c(VARIABLES,>,VAR) :-
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        prodlist(VARS,PROD),
        PROD>VAR.

product_ctr_c(VARIABLES,=<,VAR) :-
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        prodlist(VARS,PROD),
        PROD=<VAR.

prodlist(Numbers,Prod) :-
        (foreach(X,Numbers),fromto(1,P0,P,Prod)do P is P0*X).
```

B.334 proper_circuit

◊ **META-DATA:**

```
ctr_predefined(proper_circuit).

ctr_date(proper_circuit,['20120429']).

ctr_origin(proper_circuit,'Derived from %c',[circuit]).

ctr_synonyms(proper_circuit,[circuit]).

ctr_arguments(
    proper_circuit,
    ['NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    proper_circuit,
    [size('NODES')>1,
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    proper_circuit,
    proper_circuit(
        [[index-1,succ-2],
         [index-2,succ-3],
         [index-3,succ-1],
         [index-4,succ-4]])).

ctr_typical(proper_circuit,[size('NODES')>2]).

ctr_exchangeable(proper_circuit,[items('NODES',all)]).

ctr_eval(
    proper_circuit,
    [checker(proper_circuit_c),
     reformulation(proper_circuit_r)]).

ctr_application(proper_circuit,[1]).

ctr_sol(proper_circuit,2,0,2,1,-).
```

```
ctr_sol(proper_circuit,3,0,3,5,-).

ctr_sol(proper_circuit,4,0,4,20,-).

ctr_sol(proper_circuit,5,0,5,84,-).

ctr_sol(proper_circuit,6,0,6,409,-).

ctr_sol(proper_circuit,7,0,7,2365,-).

ctr_sol(proper_circuit,8,0,8,16064,-).

ctr_sol(proper_circuit,9,0,9,125664,-).

ctr_sol(proper_circuit,10,0,10,1112073,-).

proper_circuit_c([[_35044,succ-V],[_35055,succ-V]|_35054]) :-
        !,
        fail.

proper_circuit_c(NODES) :-
        length(NODES,N),
        N>1,
        collection(NODES,[int(1,N),int(1,N)]),
        sort_collection(NODES,index,SORTED_NODES),
        get_attr1(SORTED_NODES,INDEXES),
        get_attr2(SORTED_NODES,SUCCS),
        (for(J,1,N),
         foreach(X,SUCCS),
         foreach(Free-1,KeyTerm),
         foreach(J,Js),param(KeyTerm)do
         nth1(X,KeyTerm,Free-1)),
        sort(INDEXES,Js),
        sort(SUCCS,Js),
        keysort(KeyTerm,KeySorted),
        keyclumped(KeySorted,KeyClumped),
        (foreach(_35193-Ones,KeyClumped),
         foreach(Count,Counts)do
         length(Ones,Count)),
        max_member(Max,Counts),
        Max>1,
        length(Counts,M),
        M+Max=:=N+1.

proper_circuit_r(NODES) :-
```

```
length(NODES,N),
collection(NODES,[int(1,N),dvar(1,N)]),
get_attr1(NODES,IND),
sort(IND,SIND),
length(SIND,N),
get_attr12(NODES,IND_SUCC),
keysort(IND_SUCC,SIND_SUCC),
remove_key_from_collection(SIND_SUCC,Succ),
all_distinct(Succ),
(for(I,1,N),foreach(Min,Mins),param(Succ,N)do
 length([I|Ss],N),
 minimum(Min,[I|Ss]),
  (foreach(S2,Ss),fromto(I,S1,S2,_35183),param(Succ)do
  element(S1,Succ,S2))),
(for(J,1,N),foreach(J-C,ICs),foreach(C,Cs)do true),
global_cardinality(Mins,ICs),
length(Ps,N),
length(Vs,N),
Max1 in 2..N,
Max2 in 0..1,
nth1(N,Vs,Max1),
N1 is N-1,
nth1(N1,Vs,Max2),
sorting(Cs,Ps,Vs).
```

B.335 proper_forest

◇ **META-DATA:**

```
ctr_date(proper_forest,['20050604','20060813']).

ctr_origin(
    proper_forest,
    Derived from %c, \cite{BeldiceanuKatrielLorca06}.,
    [tree]).

ctr_arguments(
    proper_forest,
    ['NTREES'-dvar,
     'NODES'-collection(index-int,neighbour-svar)]).

ctr_restrictions(
    proper_forest,
    ['NTREES'>=0,
     required('NODES',[index,neighbour]),
     size('NODES')mod 2=0,
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^neighbour>=1,
     'NODES'^neighbour=<size('NODES'),
     'NODES'^neighbour=\='NODES'^index]).

ctr_example(
    proper_forest,
    proper_forest(
        3,
        [[index-1,neighbour-{3,6}],
         [index-2,neighbour-{9}],
         [index-3,neighbour-{1,5,7}],
         [index-4,neighbour-{9}],
         [index-5,neighbour-{3}],
         [index-6,neighbour-{1}],
         [index-7,neighbour-{3}],
         [index-8,neighbour-{10}],
         [index-9,neighbour-{2,4}],
         [index-10,neighbour-{8}]])).

ctr_typical(proper_forest,['NTREES'>0,size('NODES')>1]).

ctr_exchangeable(proper_forest,[items('NODES',all)]).
```

```
ctr_graph(
    proper_forest,
    ['NODES'],
    2,
    ['CLIQUE'(=\=)>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^neighbour],
    ['NVERTEX'=('NARC'+2*'NTREES')/2,
     'NCC'='NTREES',
     'NVERTEX'=size('NODES')],
    ['SYMMETRIC']).

ctr_functional_dependency(proper_forest,1,[2]).

ctr_application(proper_forest,[2]).
```

B.336 range_ctr

◊ META-DATA:

```
ctr_date(range_ctr,['20030820','20060813']).

ctr_origin(range_ctr,'Arithmetic constraint.',[]).

ctr_arguments(
    range_ctr,
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'R'-dvar]).

ctr_restrictions(
    range_ctr,
    [size('VARIABLES')>0,
     required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(range_ctr,range_ctr([[var-1],[var-9],[var-4]],=,9)).

ctr_typical(
    range_ctr,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(
    range_ctr,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     translate(['VARIABLES'^var])]).

ctr_graph(
    range_ctr,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['TRUE'],
    ['CTR'('RANGE'('VARIABLES',var),'R')],
    []).

ctr_eval(range_ctr,[reformulation(range_ctr_r)]).

ctr_pure_functional_dependency(range_ctr,[in_list('CTR',[=])]).

ctr_contractible(
```

```
    range_ctr,
    [in_list('CTR',[<,=<])],
    VARIABLES,
    any).

ctr_extensible(
    range_ctr,
    [in_list('CTR',[>=,>])],
    VARIABLES,
    any).

range_ctr_r(VARIABLES,CTR,R) :-
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        check_type(dvar,R),
        length(VARIABLES,N),
        N>0,
        get_attr1(VARIABLES,VARS),
        minimum(MIN,VARS),
        maximum(MAX,VARS),
        call_term_relop_value(MAX-MIN+1,CTR,R).
```

B.337 relaxed_sliding_sum

◇ META-DATA:

```
ctr_date(
    relaxed_sliding_sum,
    ['20000128','20030820','20060813']).

ctr_origin(relaxed_sliding_sum,'\\index{CHIP|indexuse}CHIP',[]).

ctr_arguments(
    relaxed_sliding_sum,
    ['ATLEAST'-int,
     'ATMOST'-int,
     'LOW'-int,
     'UP'-int,
     'SEQ'-int,
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    relaxed_sliding_sum,
    ['ATLEAST'>=0,
     'ATMOST'>='ATLEAST',
     'ATMOST'=<size('VARIABLES')-'SEQ'+1,
     'UP'>='LOW',
     'SEQ'>0,
     'SEQ'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    relaxed_sliding_sum,
    relaxed_sliding_sum(
        3,
        4,
        3,
        7,
        4,
        [[var-2],
         [var-4],
         [var-2],
         [var-0],
         [var-0],
         [var-3],
         [var-4]])).

ctr_typical(
```

```
    relaxed_sliding_sum,
    ['SEQ'>1,
     'SEQ'<size('VARIABLES'),
     range('VARIABLES'^var)>1,
     'ATLEAST'>0#\/'ATMOST'<size('VARIABLES')-'SEQ'+1]).

ctr_exchangeable(
    relaxed_sliding_sum,
    [vals(['ATLEAST'],int(>=(0)),>,dontcare,dontcare),
     vals(
         ['ATMOST'],
         int(=<(size('VARIABLES')-'SEQ'+1)),
         <,
         dontcare,
         dontcare),
     items('VARIABLES',reverse)]).

ctr_graph(
    relaxed_sliding_sum,
    ['VARIABLES'],
    SEQ,
    ['PATH'>>collection],
    [sum_ctr(collection,>=,'LOW'),sum_ctr(collection,=<,'UP')],
    ['NARC'>='ATLEAST','NARC'=<'ATMOST'],
    []).

ctr_eval(
    relaxed_sliding_sum,
    [reformulation(relaxed_sliding_sum_r)]).

relaxed_sliding_sum_r(ATLEAST,ATMOST,LOW,UP,SEQ,VARIABLES) :-
        integer(ATLEAST),
        integer(ATMOST),
        integer(LOW),
        integer(UP),
        integer(SEQ),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        LIMIT is N-SEQ+1,
        ATLEAST>=0,
        ATMOST>=ATLEAST,
        ATMOST=<LIMIT,
        UP>=LOW,
        SEQ>0,
        SEQ=<N,
        get_attr1(VARIABLES,VARS),
```

```
        relaxed_sliding_sum1(VARS,[],LOW,UP,SEQ,SUMB),
        call(SUMB#>=ATLEAST),
        call(SUMB#=<ATMOST).

relaxed_sliding_sum1([],_31129,_31130,_31131,_31132,0).

relaxed_sliding_sum1([Last|R],Seq,LOW,UP,SEQ,B+RB) :-
        append(Seq,[Last],Sequence),
        length(Sequence,L),
        (   L>SEQ ->
            Sequence=[_31192|SeqCur],
            build_sum_var(SeqCur,SumVar),
            B in 0..1,
            call(SumVar#>=LOW#/\SumVar#=<UP#<=>B),
            relaxed_sliding_sum1(R,SeqCur,LOW,UP,SEQ,RB)
        ;   L=SEQ ->
            build_sum_var(Sequence,SumVar),
            B in 0..1,
            call(SumVar#>=LOW#/\SumVar#=<UP#<=>B),
            relaxed_sliding_sum1(R,Sequence,LOW,UP,SEQ,RB)
        ;   relaxed_sliding_sum1(R,Sequence,LOW,UP,SEQ,RB)
        ).
```

B.338 remainder

◇ **META-DATA:**

```
ctr_predefined(remainder).

ctr_date(remainder,['20110612']).

ctr_origin(remainder,'Arithmetic.',[]).

ctr_synonyms(remainder,[modulo,mod]).

ctr_arguments(remainder,['Q'-dvar,'D'-dvar,'R'-dvar]).

ctr_restrictions(remainder,['Q'>=0,'D'>0,'R'>=0,'R'<'D']).

ctr_example(remainder,remainder(15,2,1)).

ctr_eval(remainder,[checker(remainder_c),builtin(remainder_b)]).

ctr_pure_functional_dependency(remainder,[]).

ctr_functional_dependency(remainder,3,[1,2]).

remainder_c(Q,D,R) :-
        check_type(int,Q),
        check_type(int,D),
        check_type(dvar,R),
        Q>=0,
        D>0,
        R#>=0,
        R#<D,
        REM is Q mod D,
        R#=REM.

remainder_b(Q,D,R) :-
        check_type(dvar,Q),
        check_type(dvar,D),
        check_type(dvar,R),
        Q#>=0,
        D#>0,
        R#>=0,
        R#<D,
        Q mod D#=R.
```

B.339 roots

◊ META-DATA:

```
ctr_date(roots,['20070620']).

ctr_origin(
    roots,
    \cite{BessiereHebrardHnichKiziltanWalsh05IJCAI},
    []).

ctr_arguments(
    roots,
    ['S'-svar,'T'-svar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    roots,
    ['S'=<size('VARIABLES'),required('VARIABLES',var)]).

ctr_example(
    roots,
    roots(
        {2,4,5},
        {2,3,8},
        [[var-1],[var-3],[var-1],[var-2],[var-3]])).

ctr_typical(
    roots,
    [size('VARIABLES')>1,range('VARIABLES'^var)>1]).

ctr_derived_collections(
    roots,
    [col('SETS'-collection(s-svar,t-svar),
         [item(s-'S',t-'T')])]).

ctr_graph(
    roots,
    ['SETS','VARIABLES'],
    2,
    ['PRODUCT'>>collection(sets,variables)],
    [variables^key in_set sets^s#<=>
     variables^var in_set sets^t],
    ['NARC'=size('VARIABLES')],
    []).
```

B.340 same

◊ **META-DATA:**

```
ctr_date(same,['20000128','20030820','20040530','20060813']).

ctr_origin(same,'N.~Beldiceanu',[]).

ctr_arguments(
    same,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    same,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    same,
    same(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-9],[var-1],[var-1],[var-1],[var-2],[var-5]])).

ctr_typical(
    same,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1]).

ctr_exchangeable(
    same,
    [args([['VARIABLES1','VARIABLES2']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    same,
    ['VARIABLES1','VARIABLES2'],
```

```
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    [for_all('CC','NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_eval(same,[reformulation(same_r),checker(same_c)]).

ctr_aggregate(same,[],[union,union]).

same_r(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1=N2,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        same1(VARS1,VARS2).

same1(VARS1,VARS2) :-
        length(VARS1,N),
        length(PERMUTATION1,N),
        domain(PERMUTATION1,1,N),
        length(PERMUTATION2,N),
        domain(PERMUTATION2,1,N),
        length(SVARS,N),
        get_minimum(VARS1,MIN1),
        get_maximum(VARS1,MAX1),
        domain(SVARS,MIN1,MAX1),
        sorting(VARS1,PERMUTATION1,SVARS),
        sorting(VARS2,PERMUTATION2,SVARS),
        append(VARS1,VARS2,VARS12),
        append(PERMUTATION1,PERMUTATION2,PERMUTATION12),
        when(ground(VARS12),once(labeling([],PERMUTATION12))).

same_c(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[int]),
        collection(VARIABLES2,[int]),
        length(VARIABLES1,N),
        length(VARIABLES2,N),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        create_pairs(VARS1,PVARS1),
```

```
create_pairs(VARS2,PVARS2),
keysort(PVARS1,SORTED),
keysort(PVARS2,SORTED).
```

B.341 same_and_global_cardinality

◇ **META-DATA:**

```
ctr_date(same_and_global_cardinality,['20040530','20060813']).

ctr_origin(
    same_and_global_cardinality,
    Conjoin %c and %c,
    [same,global_cardinality]).

ctr_synonyms(
    same_and_global_cardinality,
    [sgcc,same_gcc,same_and_gcc,swc,same_with_cardinalities]).

ctr_arguments(
    same_and_global_cardinality,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'VALUES'-collection(val-int,noccurrence-dvar)]).

ctr_restrictions(
    same_and_global_cardinality,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     required('VALUES',[val,noccurrence]),
     distinct('VALUES',val),
     'VALUES'^noccurrence>=0,
     'VALUES'^noccurrence=<size('VARIABLES1')]).

ctr_example(
    same_and_global_cardinality,
    same_and_global_cardinality(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-9],[var-1],[var-1],[var-1],[var-2],[var-5]],
        [[val-1,noccurrence-3],
         [val-2,noccurrence-1],
         [val-5,noccurrence-1],
         [val-7,noccurrence-0],
         [val-9,noccurrence-1]])).

ctr_typical(
    same_and_global_cardinality,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
```

```
     range('VARIABLES2'^var)>1,
     size('VALUES')>1,
     range('VALUES'^noccurrence)>1,
     size('VARIABLES1')>size('VALUES')]).

ctr_exchangeable(
    same_and_global_cardinality,
    [args([['VARIABLES1','VARIABLES2'],['VALUES']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     items('VALUES',all),
     vals(
        ['VARIABLES1'^var,'VARIABLES2'^var],
        all(notin('VALUES'^val)),
        =,
        dontcare,
        dontcare),
     vals(
        ['VARIABLES1'^var,'VARIABLES2'^var,'VALUES'^val],
        int,
        =\=,
        all,
        dontcare)]).

ctr_graph(
    same_and_global_cardinality,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    [for_all('CC','NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_graph(
    same_and_global_cardinality,
    ['VARIABLES1'],
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val],
    ['NVERTEX'='VALUES'^noccurrence],
    []).

ctr_eval(
    same_and_global_cardinality,
```

```
    [reformulation(same_and_global_cardinality_r)]).

ctr_contractible(same_and_global_cardinality,[],'VALUES',any).

same_and_global_cardinality_r(VARIABLES1,VARIABLES2,VALUES) :-
        eval(same(VARIABLES1,VARIABLES2)),
        eval(global_cardinality(VARIABLES1,VALUES)).
```

B.342 same_and_global_cardinality_low_up

◇ **META-DATA:**

```
ctr_date(
    same_and_global_cardinality_low_up,
    ['20051104','20060813']).

ctr_origin(
    same_and_global_cardinality_low_up,
    Derived from %c and %c,
    [same,global_cardinality_low_up]).

ctr_arguments(
    same_and_global_cardinality_low_up,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
    same_and_global_cardinality_low_up,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     required('VALUES',[val,omin,omax]),
     distinct('VALUES',val),
     'VALUES'^omin>=0,
     'VALUES'^omax=<size('VARIABLES1'),
     'VALUES'^omin=<'VALUES'^omax]).

ctr_example(
    same_and_global_cardinality_low_up,
    same_and_global_cardinality_low_up(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-9],[var-1],[var-1],[var-1],[var-2],[var-5]],
        [[val-1,omin-2,omax-3],
         [val-2,omin-1,omax-1],
         [val-5,omin-1,omax-1],
         [val-7,omin-0,omax-2],
         [val-9,omin-1,omax-1]])).

ctr_typical(
    same_and_global_cardinality_low_up,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
```

```
     size('VALUES')>1,
     'VALUES'^omin=<size('VARIABLES1'),
     'VALUES'^omax>0,
     'VALUES'^omax<size('VARIABLES1'),
     size('VARIABLES1')>size('VALUES')]).

ctr_exchangeable(
    same_and_global_cardinality_low_up,
    [args([['VARIABLES1','VARIABLES2'],['VALUES']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         all(notin('VALUES'^val)),
         =,
         dontcare,
         dontcare),
     items('VALUES',all),
     vals(['VALUES'^omin],int(>=(0)),>,dontcare,dontcare),
     vals(
         ['VALUES'^omax],
         int(=<(size('VARIABLES1'))),
         <,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    same_and_global_cardinality_low_up,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    [for_all('CC','NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_graph(
    same_and_global_cardinality_low_up,
    ['VARIABLES1'],
```

```
    1,
    foreach('VALUES',['SELF'>>collection(variables)]),
    [variables^var='VALUES'^val],
    ['NVERTEX'>='VALUES'^omin,'NVERTEX'=<'VALUES'^omax],
    []).

ctr_eval(
    same_and_global_cardinality_low_up,
    [reformulation(same_and_global_cardinality_low_up_r)]).

ctr_contractible(
    same_and_global_cardinality_low_up,
    [],
    VALUES,
    any).

same_and_global_cardinality_low_up_r(
    VARIABLES1,
    VARIABLES2,
    VALUES) :-
        eval(same(VARIABLES1,VARIABLES2)),
        eval(global_cardinality_low_up(VARIABLES1,VALUES)).
```

B.343 same_intersection

◇ **META-DATA:**

```
ctr_date(same_intersection,['20040530','20060814']).

ctr_origin(
    same_intersection,
    Derived from %c and %c.,
    [same,common]).

ctr_arguments(
    same_intersection,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    same_intersection,
    [required('VARIABLES1',var),required('VARIABLES2',var)]).

ctr_example(
    same_intersection,
    same_intersection(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-9],
         [var-1],
         [var-1],
         [var-1],
         [var-3],
         [var-5],
         [var-8]])).

ctr_typical(
    same_intersection,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1]).

ctr_exchangeable(
    same_intersection,
    [args([['VARIABLES1','VARIABLES2']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
```

```
          int,
          =\=,
          all,
          dontcare)]).

ctr_graph(
    same_intersection,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    [for_all('CC','NSOURCE'='NSINK')],
    []).

ctr_eval(
    same_intersection,
    [reformulation(same_intersection_r)]).

same_intersection_r(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        (   N1=0 ->
            true
        ;   N2=0 ->
            true
        ;   get_attr1(VARIABLES1,VARS1),
            get_attr1(VARIABLES2,VARS2),
            get_minimum(VARS1,MINVARS1),
            get_minimum(VARS2,MINVARS2),
            get_maximum(VARS1,MAXVARS1),
            get_maximum(VARS2,MAXVARS2),
            MIN is min(MINVARS1,MINVARS2),
            MAX is max(MAXVARS1,MAXVARS2),
            complete_card(MIN,MAX,N1,[],[],VN1),
            complete_card(MIN,MAX,N2,[],[],VN2),
            global_cardinality(VARS1,VN1),
            global_cardinality(VARS2,VN2),
            same_intersection1(VN1,VN2)
        ).

same_intersection1([],[]).

same_intersection1([V-O1|R],[V-O2|S]) :-
        O1#>0#/\O2#>0#=>O2#=O1,
```

```
same_intersection1(R,S).
```

B.344 same_interval

◊ **META-DATA:**

```
ctr_date(same_interval,['20030820','20060814']).

ctr_origin(same_interval,'Derived from %c.',[same]).

ctr_arguments(
    same_interval,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    same_interval,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'SIZE_INTERVAL'>0]).

ctr_example(
    same_interval,
    same_interval(
        [[var-1],[var-7],[var-6],[var-0],[var-1],[var-7]],
        [[var-8],[var-8],[var-8],[var-0],[var-1],[var-2]],
        3)).

ctr_typical(
    same_interval,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     'SIZE_INTERVAL'>1,
     'SIZE_INTERVAL'<range('VARIABLES1'^var),
     'SIZE_INTERVAL'<range('VARIABLES2'^var)]).

ctr_exchangeable(
    same_interval,
    [args([['VARIABLES1','VARIABLES2'],['SIZE_INTERVAL']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES'^var],
         intervals('SIZE_INTERVAL'),
         =,
```

```
         dontcare,
         dontcare)]).

ctr_graph(
    same_interval,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var/'SIZE_INTERVAL'=
     variables2^var/'SIZE_INTERVAL'],
    [for_all('CC','NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_eval(same_interval,[reformulation(same_interval_r)]).

ctr_aggregate(same_interval,[],[union,union,id]).

same_interval_r(VARIABLES1,VARIABLES2,SIZE_INTERVAL) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1=N2,
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
        gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
        same1(QUOTVARS1,QUOTVARS2).
```

B.345 same_modulo

◇ **META-DATA:**

```
ctr_date(same_modulo,['20030820','20060814']).

ctr_origin(same_modulo,'Derived from %c.',[same]).

ctr_arguments(
    same_modulo,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    same_modulo,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'M'>0]).

ctr_example(
    same_modulo,
    same_modulo(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-6],[var-4],[var-1],[var-1],[var-5],[var-5]],
        3)).

ctr_typical(
    same_modulo,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     'M'>1,
     'M'<maxval('VARIABLES1'^var),
     'M'<maxval('VARIABLES2'^var)]).

ctr_exchangeable(
    same_modulo,
    [args([['VARIABLES1','VARIABLES2'],['M']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(['VARIABLES'^var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
    same_modulo,
```

```
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    [for_all('CC','NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_eval(same_modulo,[reformulation(same_modulo_r)]).

ctr_aggregate(same_modulo,[],[union,union,id]).

same_modulo_r(VARIABLES1,VARIABLES2,M) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1=N2,
        integer(M),
        M>0,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        gen_remainder(VARS1,M,REMVARS1),
        gen_remainder(VARS2,M,REMVARS2),
        same1(REMVARS1,REMVARS2).
```

B.346 same_partition

◊ **META-DATA:**

```
ctr_date(same_partition,['20030820','20060814']).

ctr_origin(same_partition,'Derived from %c.',[same]).

ctr_types(same_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    same_partition,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    same_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    same_partition,
    same_partition(
        [[var-1],[var-2],[var-6],[var-3],[var-1],[var-2]],
        [[var-6],[var-6],[var-2],[var-3],[var-1],[var-3]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(
    same_partition,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     size('VARIABLES1')>size('PARTITIONS'),
     size('VARIABLES2')>size('PARTITIONS')]).

ctr_exchangeable(
    same_partition,
```

```
    [args([['VARIABLES1','VARIABLES2'],['PARTITIONS']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['VARIABLES'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    same_partition,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    [for_all('CC','NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_eval(same_partition,[reformulation(same_partition_r)]).

ctr_aggregate(same_partition,[],[union,union,id]).

same_partition_r(VARIABLES1,VARIABLES2,PARTITIONS) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1=N2,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        get_col_attr1(PARTITIONS,1,PVALS),
        flattern(PVALS,VALS),
        all_different(VALS),
        length(PARTITIONS,P),
        P>1,
        length(PVALS,LPVALS),
        LPVALS1 is LPVALS+1,
        get_partition_var(VARS1,PVALS,PVARS1,LPVALS1,0),
```

```
get_partition_var(VARS2,PVALS,PVARS2,LPVALS1,0),
same1(PVARS1,PVARS2).
```

B.347 same_remainder

◇ **META-DATA:**

```
ctr_predefined(same_remainder).

ctr_date(same_remainder,['20140919']).

ctr_origin(same_remainder,learning,[]).

ctr_arguments(
    same_remainder,
    ['VARIABLES'-collection(var-dvar),'Q'-dvar,'R'-dvar]).

ctr_restrictions(
    same_remainder,
    [size('VARIABLES')>0,
     required('VARIABLES',[var]),
     'VARIABLES'^var>=0,
     'Q'>1,
     'Q'=<maxval('VARIABLES'^var),
     'R'>=0,
     'R'<'Q']).

ctr_example(
    same_remainder,
    [same_remainder([[var-4],[var-6],[var-4],[var-8]],2,0),
     same_remainder([[var-4],[var-1],[var-4],[var-7]],3,1)]).

ctr_typical(same_remainder,[size('VARIABLES')>2,'Q'<10]).

ctr_exchangeable(same_remainder,[items('VARIABLES',all)]).

ctr_eval(
    same_remainder,
    [checker(same_remainder_c),
     reformulation(same_remainder_r)]).

same_remainder_c(VARIABLES,Q,R) :-
        length(VARIABLES,N),
        N>0,
        collection(VARIABLES,[int_gteq(0)]),
        get_attr1(VARIABLES,VARS),
        (   integer(Q) ->
            Q>=2,
            same_remainder_r0(VARS,Q,R)
```

```
        ;   get_maximum(VARS,Maximum),
            Maximum1 is Maximum-1,
            Q in 2..Maximum,
            R in 0..Maximum1,
            same_remainder_r1(VARS,Q,R)
        ).

same_remainder_r(VARIABLES,Q,R) :-
        length(VARIABLES,N),
        N>0,
        collection(VARIABLES,[dvar_gteq(0)]),
        check_type(dvar_gteq(2),Q),
        Q#>=2,
        check_type(dvar_gteq(0),R),
        R#<Q,
        get_attr1(VARIABLES,VARS),
        same_remainder_r1(VARS,Q,R).

same_remainder_r0([],_22310,_22311).

same_remainder_r0([V|S],Q,R) :-
        R is V mod Q,
        same_remainder_r0(S,Q,R).

same_remainder_r1([],_22310,_22311).

same_remainder_r1([V|S],Q,R) :-
        R#=V mod Q,
        same_remainder_r1(S,Q,R).
```

B.348 same_sign

◇ META-DATA:

```
ctr_predefined(same_sign).

ctr_date(same_sign,['20100821']).

ctr_origin(same_sign,'Arithmetic.',[]).

ctr_arguments(same_sign,['VAR1'-dvar,'VAR2'-dvar]).

ctr_restrictions(same_sign,[]).

ctr_example(same_sign,same_sign(7,1)).

ctr_typical(same_sign,['VAR1'=\=0,'VAR2'=\=0]).

ctr_exchangeable(same_sign,[args([['VAR1','VAR2']])]).

ctr_eval(same_sign,[builtin(same_sign_b)]).

same_sign_b(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#>=0#/\VAR2#>=0#\/VAR1#=<0#/\VAR2#=<0.
```

B.349 scalar_product

◊ **META-DATA:**

```
ctr_predefined(scalar_product).

ctr_date(scalar_product,['20090415']).

ctr_origin(scalar_product,'Arithmetic constraint.',[]).

ctr_synonyms(
    scalar_product,
    [equation,linear,sum_weight,weightedSum]).

ctr_arguments(
    scalar_product,
    ['LINEARTERM'-collection(coeff-int,var-dvar),
     'CTR'-atom,
     'VAL'-dvar]).

ctr_restrictions(
    scalar_product,
    [required('LINEARTERM',[coeff,var]),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    scalar_product,
    scalar_product(
        [[coeff-1,var-1],[coeff-3,var-1],[coeff-1,var-4]],
        =,
        8)).

ctr_typical(
    scalar_product,
    [size('LINEARTERM')>1,
     range('LINEARTERM'^coeff)>1,
     range('LINEARTERM'^var)>1,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(
    scalar_product,
    [items('LINEARTERM',all),
     attrs('LINEARTERM',[[coeff,var]])]).

ctr_eval(scalar_product,[builtin(scalar_product_b)]).
```

```
ctr_pure_functional_dependency(
    scalar_product,
    [in_list('CTR',[=])]).

ctr_contractible(
    scalar_product,
    [in_list('CTR',[<,=<]),
     minval('LINEARTERM'^coeff)>=0,
     minval('LINEARTERM'^var)>=0],
    LINEARTERM,
    any).

ctr_extensible(
    scalar_product,
    [in_list('CTR',[>=,>]),
     minval('LINEARTERM'^coeff)>=0,
     minval('LINEARTERM'^var)>=0],
    LINEARTERM,
    any).

ctr_aggregate(scalar_product,[],[union,id,+]).

scalar_product_b(LINEARTERM,=,VAR) :-
        !,
        collection(LINEARTERM,[int,dvar]),
        check_type(dvar,VAR),
        get_attr1(LINEARTERM,COEFFS),
        get_attr2(LINEARTERM,VARS),
        scalar_product(COEFFS,VARS,#=,VAR).

scalar_product_b(LINEARTERM,=\=,VAR) :-
        !,
        collection(LINEARTERM,[int,dvar]),
        check_type(dvar,VAR),
        get_attr1(LINEARTERM,COEFFS),
        get_attr2(LINEARTERM,VARS),
        scalar_product(COEFFS,VARS,#\=,VAR).

scalar_product_b(LINEARTERM,<,VAR) :-
        !,
        collection(LINEARTERM,[int,dvar]),
        check_type(dvar,VAR),
        get_attr1(LINEARTERM,COEFFS),
        get_attr2(LINEARTERM,VARS),
        scalar_product(COEFFS,VARS,#<,VAR).
```

```
scalar_product_b(LINEARTERM,>=,VAR) :-
        !,
        collection(LINEARTERM,[int,dvar]),
        check_type(dvar,VAR),
        get_attr1(LINEARTERM,COEFFS),
        get_attr2(LINEARTERM,VARS),
        scalar_product(COEFFS,VARS,#>=,VAR).

scalar_product_b(LINEARTERM,>,VAR) :-
        !,
        collection(LINEARTERM,[int,dvar]),
        check_type(dvar,VAR),
        get_attr1(LINEARTERM,COEFFS),
        get_attr2(LINEARTERM,VARS),
        scalar_product(COEFFS,VARS,#>,VAR).

scalar_product_b(LINEARTERM,=<,VAR) :-
        collection(LINEARTERM,[int,dvar]),
        check_type(dvar,VAR),
        get_attr1(LINEARTERM,COEFFS),
        get_attr2(LINEARTERM,VARS),
        scalar_product(COEFFS,VARS,#=<,VAR).
```

B.350 sequence_folding

◇ META-DATA:

```
ctr_date(sequence_folding,['20030820','20040530','20060814']).

ctr_origin(sequence_folding,'J.~Pearson',[]).

ctr_arguments(
    sequence_folding,
    ['LETTERS'-collection(index-int,next-dvar)]).

ctr_restrictions(
    sequence_folding,
    [size('LETTERS')>=1,
     required('LETTERS',[index,next]),
     'LETTERS'^index>=1,
     'LETTERS'^index=<size('LETTERS'),
     increasing_seq('LETTERS',index),
     'LETTERS'^next>=1,
     'LETTERS'^next=<size('LETTERS')]).

ctr_example(
    sequence_folding,
    sequence_folding(
        [[index-1,next-1],
         [index-2,next-8],
         [index-3,next-3],
         [index-4,next-5],
         [index-5,next-5],
         [index-6,next-7],
         [index-7,next-7],
         [index-8,next-8],
         [index-9,next-9]])).

ctr_typical(
    sequence_folding,
    [size('LETTERS')>2,range('LETTERS'^next)>1]).

ctr_graph(
    sequence_folding,
    ['LETTERS'],
    1,
    ['SELF'>>collection(letters)],
    [letters^next>=letters^index],
    ['NARC'=size('LETTERS')],
```

```
    []).

ctr_graph(
    sequence_folding,
    ['LETTERS'],
    2,
    ['CLIQUE'(<)>>collection(letters1,letters2)],
    [letters2^index>=letters1^next#\/
     letters2^next=<letters1^next],
    ['NARC'=size('LETTERS')*(size('LETTERS')-1)/2],
    []).

ctr_eval(sequence_folding,[automaton(sequence_folding_a)]).

ctr_application(sequence_folding,[1]).

sequence_folding_a(FLAG,LETTERS) :-
        length(LETTERS,N),
        N>=1,
        collection(LETTERS,[int(1,N),dvar(1,N)]),
        collection_increasing_seq(LETTERS,[1]),
        sequence_folding_signature(LETTERS,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _58283,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s),arc(s,1,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2],AUTOMATON).

sequence_folding_signature([],[]).

sequence_folding_signature([_56576],[]) :-
        !.

sequence_folding_signature([L1,L2|R],S) :-
        sequence_folding_signature([L2|R],L1,S1),
        sequence_folding_signature([L2|R],S2),
        append(S1,S2,S).

sequence_folding_signature([],_56572,[]).
```

```
sequence_folding_signature([L2|R],L1,[S|Ss]) :-
        L1=[index-INDEX1,next-NEXT1],
        L2=[index-INDEX2,next-NEXT2],
        INDEX1#=<NEXT1#/\INDEX2#=<NEXT2#/\NEXT1#=<INDEX2#<=>
        S#=0,
        INDEX1#=<NEXT1#/\INDEX2#=<NEXT2#/\NEXT1#>INDEX2#/\
        NEXT2#=<NEXT1#<=>
        S#=1,
        sequence_folding_signature(R,L1,Ss).
```

B.351 set_value_precede

◊ **META-DATA:**

```
ctr_predefined(set_value_precede).

ctr_date(set_value_precede,['20041003']).

ctr_origin(set_value_precede,'\\cite{YatChiuLawJimmyLee04}',[]).

ctr_arguments(
    set_value_precede,
    ['S'-int,'T'-int,'VARIABLES'-collection(var-svar)]).

ctr_restrictions(
    set_value_precede,
    ['S'=\='T',required('VARIABLES',var)]).

ctr_example(
    set_value_precede,
    [set_value_precede(
         2,
         1,
         [[var-{0,2}],[var-{0,1}],[var-{}],[var-{1}]]),
     set_value_precede(
         0,
         1,
         [[var-{0,2}],[var-{0,1}],[var-{}],[var-{1}]]),
     set_value_precede(
         0,
         2,
         [[var-{0,2}],[var-{0,1}],[var-{}],[var-{1}]]),
     set_value_precede(
         0,
         4,
         [[var-{0,2}],[var-{0,1}],[var-{}],[var-{1}]])]).

ctr_typical(set_value_precede,['S'<'T',size('VARIABLES')>1]).

ctr_contractible(set_value_precede,[],'VARIABLES',suffix).
```

B.352 shift

♢ META-DATA:

```
ctr_date(shift,['20030820','20060814','20090531']).

ctr_origin(shift,'N.~Beldiceanu',[]).

ctr_arguments(
    shift,
    ['MIN_BREAK'-int,
     'MAX_RANGE'-int,
     'TASKS'-collection(origin-dvar,end-dvar)]).

ctr_restrictions(
    shift,
    ['MIN_BREAK'>0,
     'MAX_RANGE'>0,
     required('TASKS',[origin,end]),
     'TASKS'^origin<'TASKS'^end]).

ctr_example(
    shift,
    shift(
        6,
        8,
        [[origin-17,end-20],
         [origin-7,end-10],
         [origin-2,end-4],
         [origin-21,end-22],
         [origin-5,end-6]])).

ctr_typical(
    shift,
    ['MIN_BREAK'>1,
     'MAX_RANGE'>1,
     'MIN_BREAK'<'MAX_RANGE',
     size('TASKS')>2]).

ctr_exchangeable(
    shift,
    [items('TASKS',all),translate(['TASKS'^origin])]).

ctr_graph(
    shift,
    ['TASKS'],
```

```
    1,
    ['SELF'>>collection(tasks)],
    [tasks^end>=tasks^origin,
     tasks^end-tasks^origin=<'MAX_RANGE'],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    shift,
    ['TASKS'],
    2,
    ['CLIQUE'>>collection(tasks1,tasks2)],
    [tasks2^origin>=tasks1^end#/\
     tasks2^origin-tasks1^end=<'MIN_BREAK'#\/
     tasks1^origin>=tasks2^end#/\
     tasks1^origin-tasks2^end=<'MIN_BREAK'#\/
     tasks2^origin<tasks1^end#/\tasks1^origin<tasks2^end],
    [],
    [],
    [CC>>
     [variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^origin),item(var-'TASKS'^end)])]],
    [range_ctr(variables,=<,'MAX_RANGE')]).

ctr_eval(shift,[reformulation(shift_r)]).

ctr_application(shift,[3]).

shift_r(MIN_BREAK,MAX_RANGE,TASKS) :-
        integer(MIN_BREAK),
        MIN_BREAK>0,
        integer(MAX_RANGE),
        MAX_RANGE>0,
        collection(TASKS,[dvar,dvar]),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,ENDS),
        get_minimum(ORIGINS,MINO),
        get_maximum(ORIGINS,MAXO),
        get_minimum(ENDS,MINE),
        get_maximum(ENDS,MAXE),
        shift1(
            ORIGINS,
            ENDS,
            ORIGINS,
            ENDS,
```

```
            MINO,
            MAXO,
            MINE,
            MAXE,
            MIN_BREAK,
            MAX_RANGE).

shift1(
    [],
    [],
    _48315,
    _48362,
    _48409,
    _48456,
    _48503,
    _48550,
    _48597,
    _48644).

shift1(
    [O|RO],
    [E|RE],
    ORIGINS,
    ENDS,
    MINO,
    MAXO,
    MINE,
    MAXE,
    MIN_BREAK,
    MAX_RANGE) :-
        shift2(
            ORIGINS,
            ENDS,
            O,
            E,
            MIN_BREAK,
            MAX_RANGE,
            ORIBOOLS,
            ENDBOOLS),
        MIN in MINO..MAXO,
        MAX in MINE..MAXE,
        eval(open_minimum(MIN,ORIBOOLS)),
        eval(open_maximum(MAX,ENDBOOLS)),
        MAX-MIN#=<MAX_RANGE,
        shift1(
            RO,
```

```
            RE,
            ORIGINS,
            ENDS,
            MINO,
            MAXO,
            MINE,
            MAXE,
            MIN_BREAK,
            MAX_RANGE).

shift2([],[],_47977,_47978,_47979,_47980,[],[]).

shift2(
    [Oj|RO],
    [Ej|RE],
    Oi,
    Ei,
    MIN_BREAK,
    MAX_RANGE,
    [[var-Oj,bool-Bij]|ROB],
    [[var-Ej,bool-Bij]|REB]) :-
        Oi#<Ei,
        Bij#<=>
        Oj#>=Ei#/\Oj-Ei#=<MIN_BREAK#\/
        Oi#>=Ej#/\Oi-Ej#=<MIN_BREAK#\/
        Oj#<Ei#/\Oi#<Ej,
        shift2(RO,RE,Oi,Ei,MIN_BREAK,MAX_RANGE,ROB,REB).
```

B.353 sign_of

◇ META-DATA:

```
ctr_predefined(sign_of).

ctr_date(sign_of,['20110612']).

ctr_origin(sign_of,'Arithmetic.',[]).

ctr_usual_name(sign_of,sign).

ctr_arguments(sign_of,['S'-dvar,'X'-dvar]).

ctr_restrictions(sign_of,['S'>= -1,'S'=<1]).

ctr_example(sign_of,[sign_of(-1,-8),sign_of(0,0),sign_of(1,8)]).

ctr_typical(sign_of,['S'=\=0,'X'=\=0]).

ctr_eval(sign_of,[checker(sign_of_c),builtin(sign_of_b)]).

ctr_pure_functional_dependency(sign_of,[]).

ctr_functional_dependency(sign_of,1,[2]).

sign_of_c(S,X) :-
        check_type(int,S),
        check_type(int,X),
        (   S= -1 ->
            X<0
        ;   S=0 ->
            X=0
        ;   S=1 ->
            X>0
        ).

sign_of_b(S,X) :-
        check_type(dvar,S),
        check_type(dvar,X),
        S#>= -1,
        S#=<1,
        X#<0#/\S#= -1#\/X#=0#/\S#=0#\/X#>0#/\S#=1.
```

B.354 size_max_seq_alldifferent

◊ **META-DATA:**

```
ctr_date(
    size_max_seq_alldifferent,
    ['20030820','20060814','20121124']).

ctr_origin(size_max_seq_alldifferent,'N.~Beldiceanu',[]).

ctr_synonyms(
    size_max_seq_alldifferent,
    [size_maximal_sequence_alldiff,
     size_maximal_sequence_alldistinct,
     size_maximal_sequence_alldifferent]).

ctr_arguments(
    size_max_seq_alldifferent,
    ['SIZE'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    size_max_seq_alldifferent,
    ['SIZE'>=0,
     'SIZE'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    size_max_seq_alldifferent,
    [size_max_seq_alldifferent(
         4,
         [[var-2],
          [var-2],
          [var-4],
          [var-5],
          [var-2],
          [var-7],
          [var-4]]),
     size_max_seq_alldifferent(
         1,
         [[var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2],
          [var-2]]),
```

```
     size_max_seq_alldifferent(
         2,
         [[var-2],
          [var-2],
          [var-4],
          [var-4],
          [var-4],
          [var-7],
          [var-4]]),
     size_max_seq_alldifferent(
         7,
         [[var-2],
          [var-0],
          [var-4],
          [var-6],
          [var-5],
          [var-7],
          [var-3]]])]).

ctr_typical(
    size_max_seq_alldifferent,
    ['SIZE'>2,
     'SIZE'<size('VARIABLES'),
     range('VARIABLES'^var)>1]).

ctr_exchangeable(
    size_max_seq_alldifferent,
    [translate(['VARIABLES'^var])]).

ctr_graph(
    size_max_seq_alldifferent,
    ['VARIABLES'],
    *,
    ['PATH_N'>>collection],
    [alldifferent(collection)],
    ['NARC'='SIZE'],
    []).

ctr_eval(
    size_max_seq_alldifferent,
    [checker(size_max_seq_alldifferent_c),
     reformulation(size_max_seq_alldifferent_r)]).

ctr_pure_functional_dependency(size_max_seq_alldifferent,[]).

ctr_functional_dependency(size_max_seq_alldifferent,1,[2]).
```

```
ctr_sol(size_max_seq_alldifferent,2,0,2,9,[1-3,2-6]).

ctr_sol(size_max_seq_alldifferent,3,0,3,64,[1-4,2-36,3-24]).

ctr_sol(
    size_max_seq_alldifferent,
    4,
    0,
    4,
    625,
    [1-5,2-200,3-300,4-120]).

ctr_sol(
    size_max_seq_alldifferent,
    5,
    0,
    5,
    7776,
    [1-6,2-1050,3-3480,4-2520,5-720]).

ctr_sol(
    size_max_seq_alldifferent,
    6,
    0,
    6,
    117649,
    [1-7,2-5922,3-38640,4-45360,5-22680,6-5040]).

ctr_sol(
    size_max_seq_alldifferent,
    7,
    0,
    7,
    2097152,
    [1-8,2-34104,3-428400,4-801360,5-571200,6-221760,7-40320]).

ctr_sol(
    size_max_seq_alldifferent,
    8,
    0,
    8,
    43046721,
    [1-9,
     2-208224,
     3-4981032,
```

```
    4-14028336,
    5-13728960,
    6-7378560,
    7-2358720,
    8-362880]).

size_max_seq_alldifferent_c(SIZE,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        size_max_seq_alldifferent0(VARS,VARS,[],0,SIZE).

size_max_seq_alldifferent0([X|Tail],Head,Set0,Size0,Size) :-
        fdset_add_element(Set0,X,Set),
        Set0\==Set,
        !,
        fdset_size(Set,Size1),
        Size2 is max(Size0,Size1),
        size_max_seq_alldifferent0(Tail,Head,Set,Size2,Size).

size_max_seq_alldifferent0(Tail,[X|Head],Set0,Size0,Size) :-
        !,
        fdset_del_element(Set0,X,Set),
        size_max_seq_alldifferent0(Tail,Head,Set,Size0,Size).

size_max_seq_alldifferent0(_52601,_52602,_52603,Size,Size).

size_max_seq_alldifferent_r(SIZE,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        collection(VARIABLES,[dvar]),
        size_max_seq_alldifferent1(VARIABLES,N,SIZES),
        eval(maximum(SIZE,SIZES)).

size_max_seq_alldifferent1([],_52602,[]).

size_max_seq_alldifferent1([AV|R],N,[[var-SIZE]|S]) :-
        SIZE in 0..N,
        eval(size_max_starting_seq_alldifferent(SIZE,[AV|R])),
        N1 is N-1,
        size_max_seq_alldifferent1(R,N1,S).
```

B.355 size_max_starting_seq_alldifferent

◊ **META-DATA:**

```
ctr_date(
    size_max_starting_seq_alldifferent,
    ['20030820','20060814','20090524','20121124']).

ctr_origin(
    size_max_starting_seq_alldifferent,
    Inspired by %c.,
    [size_max_seq_alldifferent]).

ctr_synonyms(
    size_max_starting_seq_alldifferent,
    [size_maximal_starting_sequence_alldiff,
     size_maximal_starting_sequence_alldistinct,
     size_maximal_starting_sequence_alldifferent]).

ctr_arguments(
    size_max_starting_seq_alldifferent,
    ['SIZE'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    size_max_starting_seq_alldifferent,
    ['SIZE'>=0,
     'SIZE'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    size_max_starting_seq_alldifferent,
    [size_max_starting_seq_alldifferent(
         4,
         [[var-9],
          [var-2],
          [var-4],
          [var-5],
          [var-2],
          [var-7],
          [var-4]]),
     size_max_starting_seq_alldifferent(
         7,
         [[var-9],
          [var-2],
          [var-4],
          [var-5],
```

```
          [var-1],
          [var-7],
          [var-8]]),
     size_max_starting_seq_alldifferent(
         6,
         [[var-9],
          [var-2],
          [var-4],
          [var-5],
          [var-1],
          [var-7],
          [var-9]]])]).

ctr_typical(
    size_max_starting_seq_alldifferent,
    ['SIZE'>2,
     'SIZE'<size('VARIABLES'),
     range('VARIABLES'^var)>1]).

ctr_exchangeable(
    size_max_starting_seq_alldifferent,
    [translate(['VARIABLES'^var])]).

ctr_graph(
    size_max_starting_seq_alldifferent,
    ['VARIABLES'],
    *,
    ['PATH_1'>>collection],
    [alldifferent(collection)],
    ['NARC'='SIZE'],
    []).

ctr_eval(
    size_max_starting_seq_alldifferent,
    [checker(size_max_starting_seq_alldifferent_c),
     reformulation(size_max_starting_seq_alldifferent_r)]).

ctr_pure_functional_dependency(
    size_max_starting_seq_alldifferent,
    []).

ctr_functional_dependency(
    size_max_starting_seq_alldifferent,
    1,
    [2]).
```

```
ctr_sol(size_max_starting_seq_alldifferent,2,0,2,9,[1-3,2-6]).

ctr_sol(
    size_max_starting_seq_alldifferent,
    3,
    0,
    3,
    64,
    [1-16,2-24,3-24]).

ctr_sol(
    size_max_starting_seq_alldifferent,
    4,
    0,
    4,
    625,
    [1-125,2-200,3-180,4-120]).

ctr_sol(
    size_max_starting_seq_alldifferent,
    5,
    0,
    5,
    7776,
    [1-1296,2-2160,3-2160,4-1440,5-720]).

ctr_sol(
    size_max_starting_seq_alldifferent,
    6,
    0,
    6,
    117649,
    [1-16807,2-28812,3-30870,4-23520,5-12600,6-5040]).

ctr_sol(
    size_max_starting_seq_alldifferent,
    7,
    0,
    7,
    2097152,
    [1-262144,
     2-458752,
     3-516096,
     4-430080,
     5-268800,
     6-120960,
```

```
     7-40320]).

ctr_sol(
    size_max_starting_seq_alldifferent,
    8,
    0,
    8,
    43046721,
    [1-4782969,
     2-8503056,
     3-9920232,
     4-8817984,
     5-6123600,
     6-3265920,
     7-1270080,
     8-362880]).

size_max_starting_seq_alldifferent_c(SIZE,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        size_max_starting_seq_alldifferent0(VARS,[],SIZE).

size_max_starting_seq_alldifferent0([X|Xs],Set0,Size) :-
        fdset_add_element(Set0,X,Set),
        Set0\==Set,
        !,
        size_max_starting_seq_alldifferent0(Xs,Set,Size).

size_max_starting_seq_alldifferent0(_53783,Set,Size) :-
        fdset_size(Set,Size).

size_max_starting_seq_alldifferent_r(SIZE,VARIABLES) :-
        length(VARIABLES,N),
        check_type(dvar(0,N),SIZE),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        size_max_starting_seq_alldifferent1(VARS,[],1,SUMB),
        call(SIZE#=SUMB).

size_max_starting_seq_alldifferent1([],_53781,_53782,0).

size_max_starting_seq_alldifferent1([VAR|RVARS],L,BPREV,B+SUM) :-
        size_max_starting_seq_alldifferent2(L,VAR,BPREV,CONJ),
        call(B#<=>CONJ),
```

```
        size_max_starting_seq_alldifferent1(
            RVARS,
            [VAR|L],
            B,
            SUM).

size_max_starting_seq_alldifferent2([],_53781,BPREV,BPREV).

size_max_starting_seq_alldifferent2(
    [VAR2|RVARS],
    VAR1,
    BPREV,
    VAR1#\=VAR2#/\R) :-
        size_max_starting_seq_alldifferent2(RVARS,VAR1,BPREV,R).
```

B.356 sliding_card_skip0

◇ **META-DATA:**

```
ctr_date(
    sliding_card_skip0,
    ['20000128','20030820','20040530','20060815']).

ctr_origin(sliding_card_skip0,'N.~Beldiceanu',[]).

ctr_arguments(
    sliding_card_skip0,
    ['ATLEAST'-int,
     'ATMOST'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int)]).

ctr_restrictions(
    sliding_card_skip0,
    ['ATLEAST'>=0,
     'ATLEAST'=<size('VARIABLES'),
     'ATMOST'>=0,
     'ATMOST'=<size('VARIABLES'),
     'ATMOST'>='ATLEAST',
     required('VARIABLES',var),
     required('VALUES',val),
     distinct('VALUES',val),
     'VALUES'^val=\=0]).

ctr_example(
    sliding_card_skip0,
    sliding_card_skip0(
        2,
        3,
        [[var-0],
         [var-7],
         [var-2],
         [var-9],
         [var-0],
         [var-0],
         [var-9],
         [var-4],
         [var-9]],
        [[val-7],[val-9]])).

ctr_typical(
```

```
    sliding_card_skip0,
    [size('VARIABLES')>1,
     size('VALUES')>0,
     size('VARIABLES')>size('VALUES'),
     atleast(1,'VARIABLES',0),
     'ATLEAST'>0#\/'ATMOST'<size('VARIABLES')]).

ctr_exchangeable(
    sliding_card_skip0,
    [vals(['ATLEAST'],int(>=(0)),>,dontcare,dontcare),
     vals(
         ['ATMOST'],
         int(=<(size('VARIABLES'))),
         <,
         dontcare,
         dontcare),
     items('VARIABLES',reverse),
     vals(
         ['VARIABLES'^var],
         comp_diff('VALUES'^val,=\=(0)),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    sliding_card_skip0,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2),
     'LOOP'>>collection(variables1,variables2)],
    [variables1^var=\=0,variables2^var=\=0],
    [],
    [],
    ['CC'>>[variables]],
    [among_low_up('ATLEAST','ATMOST',variables,'VALUES')]).

ctr_eval(sliding_card_skip0,[automaton(sliding_card_skip0_a)]).

sliding_card_skip0_a(FLAG,ATLEAST,ATMOST,VARIABLES,VALUES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        check_type(int(0,N),ATLEAST),
        check_type(int(0,N),ATMOST),
        ATMOST>=ATLEAST,
        collection(VALUES,[int_diff(0)]),
        get_attr1(VALUES,LIST_VALUES),
```

```
        all_different(LIST_VALUES),
        list_to_fdset(LIST_VALUES,SET_OF_VALUES),
        sliding_card_skip0_signature(
            VARIABLES,
            SIGNATURE,
            SET_OF_VALUES),
        automaton(
            SIGNATURE,
            _50685,
            SIGNATURE,
            [source(s),sink(t),sink(s)],
            [arc(s,0,s),
             arc(s,1,t,[0,L,U]),
             arc(s,2,t,[1,L,U]),
             arc(t,0,s,[C,min(L,C),max(U,C)]),
             arc(t,1,t),
             arc(t,2,t,[C+1,L,U])],
            [C,L,U],
            [ATLEAST,ATLEAST,ATMOST],
            [C1,L1,U1]),
        min(C1,L1)#>=ATLEAST#/\max(C1,U1)#=<ATMOST#<=>FLAG.

sliding_card_skip0_signature([],[],_47845).

sliding_card_skip0_signature(
    [[var-VAR]|VARs],
    [S|Ss],
    SET_OF_VALUES) :-
        VAR#\=0#<=>NZ,
        VAR in_set SET_OF_VALUES#<=>In,
        S in 0..2,
        S#=max(2*NZ+In-1,0),
        sliding_card_skip0_signature(VARs,Ss,SET_OF_VALUES).
```

B.357 sliding_distribution

◇ **META-DATA:**

```
ctr_date(
    sliding_distribution,
    ['20031008','20060815','20090524']).

ctr_origin(sliding_distribution,'\\cite{ReginPuget97}',[]).

ctr_arguments(
    sliding_distribution,
    ['SEQ'-int,
     'VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,omin-int,omax-int)]).

ctr_restrictions(
    sliding_distribution,
    ['SEQ'>0,
     'SEQ'=<size('VARIABLES'),
     required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,omin,omax]),
     distinct('VALUES',val),
     'VALUES'^omin>=0,
     'VALUES'^omax=<'SEQ',
     'VALUES'^omin=<'VALUES'^omax]).

ctr_example(
    sliding_distribution,
    sliding_distribution(
        4,
        [[var-0],
         [var-5],
         [var-0],
         [var-6],
         [var-5],
         [var-0],
         [var-0]],
        [[val-0,omin-1,omax-2],
         [val-1,omin-0,omax-4],
         [val-4,omin-0,omax-4],
         [val-5,omin-1,omax-2],
         [val-6,omin-0,omax-2]])).

ctr_typical(
```

```
    sliding_distribution,
    ['SEQ'>1,
     'SEQ'<size('VARIABLES'),
     size('VARIABLES')>size('VALUES')]).

ctr_exchangeable(
    sliding_distribution,
    [items('VARIABLES',reverse),
     vals(
         ['VARIABLES'^var],
         all(notin('VALUES'^val)),
         =,
         dontcare,
         dontcare),
     items('VALUES',all),
     vals(['VALUES'^omin],int(>=(0)),>,dontcare,dontcare),
     vals(['VALUES'^omax],int(=<('SEQ')),<,dontcare,dontcare),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    sliding_distribution,
    ['VARIABLES'],
    SEQ,
    ['PATH'>>collection],
    [global_cardinality_low_up(collection,'VALUES')],
    ['NARC'=size('VARIABLES')-'SEQ'+1],
    []).

ctr_eval(
    sliding_distribution,
    [reformulation(sliding_distribution_r)]).

ctr_contractible(
    sliding_distribution,
    ['SEQ'=1],
    VARIABLES,
    any).

ctr_contractible(sliding_distribution,[],'VARIABLES',prefix).

ctr_contractible(sliding_distribution,[],'VARIABLES',suffix).
```

```
ctr_contractible(sliding_distribution,[],'VALUES',any).

sliding_distribution_r(SEQ,VARIABLES,VALUES) :-
        integer(SEQ),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        SEQ>0,
        SEQ=<L,
        collection(VALUES,[int,int(0,L),int(0,L)]),
        length(VALUES,M),
        M>0,
        sliding_distribution1(VARIABLES,[],VALUES,SEQ).

sliding_distribution1([],_34003,_34004,_34005).

sliding_distribution1([Last|R],Seq,VALUES,SEQ) :-
        append(Seq,[Last],Sequence),
        length(Sequence,L),
        (   L>SEQ ->
            Sequence=[_34055|SeqCur],
            eval(global_cardinality_low_up(SeqCur,VALUES)),
            sliding_distribution1(R,SeqCur,VALUES,SEQ)
        ;   L=SEQ ->
            eval(global_cardinality_low_up(Sequence,VALUES)),
            sliding_distribution1(R,Sequence,VALUES,SEQ)
        ;   sliding_distribution1(R,Sequence,VALUES,SEQ)
        ).
```

B.358 sliding_sum

♢ META-DATA:

```
ctr_date(sliding_sum,['20000128','20030820','20060815']).

ctr_origin(sliding_sum,'\\index{CHIP|indexuse}CHIP',[]).

ctr_synonyms(sliding_sum,[sequence]).

ctr_arguments(
    sliding_sum,
    ['LOW'-int,
     'UP'-int,
     'SEQ'-int,
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    sliding_sum,
    ['UP'>='LOW',
     'SEQ'>0,
     'SEQ'=<size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    sliding_sum,
    sliding_sum(
        3,
        7,
        4,
        [[var-1],
         [var-4],
         [var-2],
         [var-0],
         [var-0],
         [var-3],
         [var-4]])).

ctr_typical(
    sliding_sum,
    ['LOW'>=0,
     'UP'>0,
     'SEQ'>1,
     'SEQ'<size('VARIABLES'),
     'VARIABLES'^var>=0,
     'UP'<sum('VARIABLES'^var)]).
```

```
ctr_exchangeable(sliding_sum,[items('VARIABLES',reverse)]).

ctr_graph(
    sliding_sum,
    ['VARIABLES'],
    SEQ,
    ['PATH'>>collection],
    [sum_ctr(collection,>=,'LOW'),sum_ctr(collection,=<,'UP')],
    ['NARC'=size('VARIABLES')-'SEQ'+1],
    []).

ctr_eval(sliding_sum,[reformulation(sliding_sum_r)]).

ctr_contractible(sliding_sum,['SEQ'=1],'VARIABLES',any).

ctr_contractible(sliding_sum,[],'VARIABLES',prefix).

ctr_contractible(sliding_sum,[],'VARIABLES',suffix).

sliding_sum_r(LOW,UP,SEQ,VARIABLES) :-
        integer(LOW),
        integer(UP),
        integer(SEQ),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        UP>=LOW,
        SEQ>0,
        SEQ=<L,
        sliding_sum1(VARIABLES,[],LOW,UP,SEQ).

sliding_sum1([],_35770,_35771,_35772,_35773).

sliding_sum1([Last|R],Seq,LOW,UP,SEQ) :-
        append(Seq,[Last],Sequence),
        length(Sequence,L),
        (   L>SEQ ->
            Sequence=[_35825|SeqCur],
            eval(sum_ctr(SeqCur,>=,LOW)),
            eval(sum_ctr(SeqCur,=<,UP)),
            sliding_sum1(R,SeqCur,LOW,UP,SEQ)
        ;   L=SEQ ->
            eval(sum_ctr(Sequence,>=,LOW)),
            eval(sum_ctr(Sequence,=<,UP)),
            sliding_sum1(R,Sequence,LOW,UP,SEQ)
        ;   sliding_sum1(R,Sequence,LOW,UP,SEQ)
```

```
).
```

B.359 sliding_time_window

◇ **META-DATA:**

```
ctr_date(
    sliding_time_window,
    ['20030820','20060815','20090530']).

ctr_origin(sliding_time_window,'N.~Beldiceanu',[]).

ctr_arguments(
    sliding_time_window,
    ['WINDOW_SIZE'-int,
     'LIMIT'-int,
     'TASKS'-collection(origin-dvar,duration-dvar)]).

ctr_restrictions(
    sliding_time_window,
    ['WINDOW_SIZE'>0,
     'LIMIT'>=0,
     required('TASKS',[origin,duration]),
     'TASKS'^duration>=0]).

ctr_example(
    sliding_time_window,
    sliding_time_window(
        9,
        6,
        [[origin-10,duration-3],
         [origin-5,duration-1],
         [origin-6,duration-2],
         [origin-14,duration-2],
         [origin-2,duration-2]])).

ctr_typical(
    sliding_time_window,
    ['WINDOW_SIZE'>1,
     'LIMIT'>0,
     'LIMIT'<sum('TASKS'^duration),
     size('TASKS')>1,
     'TASKS'^duration>0]).

ctr_exchangeable(
    sliding_time_window,
    [vals(['WINDOW_SIZE'],int,>,dontcare,dontcare),
     vals(['LIMIT'],int,<,dontcare,dontcare),
```

```
     items('TASKS',all),
     translate(['TASKS'^origin]),
     vals(['TASKS'^duration],int(>=(0)),>,dontcare,dontcare)]).

ctr_graph(
    sliding_time_window,
    ['TASKS'],
    2,
    ['CLIQUE'>>collection(tasks1,tasks2)],
    [tasks1^origin=<tasks2^origin,
     tasks2^origin-tasks1^origin<'WINDOW_SIZE'],
    [],
    [],
    ['SUCC'>>[source,tasks]],
    [sliding_time_window_from_start(
         WINDOW_SIZE,
         LIMIT,
         tasks,
         source^origin)]).

ctr_eval(
    sliding_time_window,
    [reformulation(sliding_time_window_r)]).

ctr_contractible(sliding_time_window,[],'TASKS',any).

ctr_application(sliding_time_window,[3]).

sliding_time_window_r(WINDOW_SIZE,LIMIT,TASKS) :-
        integer(WINDOW_SIZE),
        WINDOW_SIZE>0,
        integer(LIMIT),
        LIMIT>=0,
        collection(TASKS,[dvar,dvar_gteq(0)]),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        sliding_time_window1(
            ORIGINS,
            DURATIONS,
            1,
            ORIGINS,
            DURATIONS,
            WINDOW_SIZE,
            LIMIT).
```

B.360 sliding_time_window_from_start

◊ **META-DATA:**

```
ctr_date(
    sliding_time_window_from_start,
    ['20030820','20060815','20090530']).

ctr_origin(
    sliding_time_window_from_start,
    Used for defining %c.,
    [sliding_time_window]).

ctr_arguments(
    sliding_time_window_from_start,
    ['WINDOW_SIZE'-int,
     'LIMIT'-int,
     'TASKS'-collection(origin-dvar,duration-dvar),
     'START'-dvar]).

ctr_restrictions(
    sliding_time_window_from_start,
    ['WINDOW_SIZE'>0,
     'LIMIT'>=0,
     required('TASKS',[origin,duration]),
     'TASKS'^duration>=0]).

ctr_example(
    sliding_time_window_from_start,
    sliding_time_window_from_start(
        9,
        6,
        [[origin-10,duration-3],
         [origin-5,duration-1],
         [origin-6,duration-2]],
        5)).

ctr_typical(
    sliding_time_window_from_start,
    ['WINDOW_SIZE'>1,
     'LIMIT'>0,
     'LIMIT'<'WINDOW_SIZE',
     size('TASKS')>1,
     'TASKS'^duration>0]).

ctr_exchangeable(
```

```
    sliding_time_window_from_start,
    [vals(['WINDOW_SIZE'],int,>,dontcare,dontcare),
     vals(['LIMIT'],int,<,dontcare,dontcare),
     items('TASKS',all),
     vals(['TASKS'^duration],int(>=(0)),>,dontcare,dontcare),
     translate(['START','TASKS'^origin])]).

ctr_derived_collections(
    sliding_time_window_from_start,
    [col('S'-collection(var-dvar),[item(var-'START')])]).

ctr_graph(
    sliding_time_window_from_start,
    ['S','TASKS'],
    2,
    ['PRODUCT'>>collection(s,tasks)],
    ['TRUE'],
    ['SUM_WEIGHT_ARC'(
         max(0,
             min(s^var+'WINDOW_SIZE',
                 tasks^origin+tasks^duration)-
             max(s^var,tasks^origin)))=<
     LIMIT],
    []).

ctr_eval(
    sliding_time_window_from_start,
    [reformulation(sliding_time_window_from_start_r)]).

ctr_contractible(sliding_time_window_from_start,[],'TASKS',any).

ctr_application(sliding_time_window_from_start,[3]).

sliding_time_window_from_start_r(WINDOW_SIZE,LIMIT,TASKS,START) :-
        integer(WINDOW_SIZE),
        WINDOW_SIZE>0,
        integer(LIMIT),
        LIMIT>=0,
        collection(TASKS,[dvar,dvar_gteq(0)]),
        check_type(dvar,START),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,DURATIONS),
        sliding_time_window1(
            [START],
            [WINDOW_SIZE],
            0,
```

```
ORIGINS,
DURATIONS,
WINDOW_SIZE,
LIMIT).
```

B.361 sliding_time_window_sum

◇ META-DATA:

```
ctr_date(
    sliding_time_window_sum,
    ['20030820','20060815','20090530']).

ctr_origin(
    sliding_time_window_sum,
    Derived from %c.,
    [sliding_time_window]).

ctr_arguments(
    sliding_time_window_sum,
    ['WINDOW_SIZE'-int,
     'LIMIT'-int,
     'TASKS'-collection(origin-dvar,end-dvar,npoint-dvar)]).

ctr_restrictions(
    sliding_time_window_sum,
    ['WINDOW_SIZE'>0,
     'LIMIT'>=0,
     required('TASKS',[origin,end,npoint]),
     'TASKS'^origin=<'TASKS'^end,
     'TASKS'^npoint>=0]).

ctr_example(
    sliding_time_window_sum,
    sliding_time_window_sum(
        9,
        16,
        [[origin-10,end-13,npoint-2],
         [origin-5,end-6,npoint-3],
         [origin-6,end-8,npoint-4],
         [origin-14,end-16,npoint-5],
         [origin-2,end-4,npoint-6]])).

ctr_typical(
    sliding_time_window_sum,
    ['WINDOW_SIZE'>1,
     'LIMIT'>0,
     'LIMIT'<sum('TASKS'^npoint),
     size('TASKS')>1,
     'TASKS'^origin<'TASKS'^end,
     'TASKS'^npoint>0]).
```

```
ctr_exchangeable(
    sliding_time_window_sum,
    [vals(['WINDOW_SIZE'],int,>,dontcare,dontcare),
     vals(['LIMIT'],int,<,dontcare,dontcare),
     items('TASKS',all),
     vals(['TASKS'^npoint],int(>=(0)),>,dontcare,dontcare),
     translate(['TASKS'^origin,'TASKS'^end])]).

ctr_graph(
    sliding_time_window_sum,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin=<tasks^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    sliding_time_window_sum,
    ['TASKS'],
    2,
    ['CLIQUE'>>collection(tasks1,tasks2)],
    [tasks1^end=<tasks2^end,
     tasks2^origin-tasks1^end<'WINDOW_SIZE'-1],
    [],
    [],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^npoint)])]],
    [sum_ctr(variables,=<,'LIMIT')]).

ctr_eval(
    sliding_time_window_sum,
    [reformulation(sliding_time_window_sum_r)]).

ctr_contractible(sliding_time_window_sum,[],'TASKS',any).

ctr_application(sliding_time_window_sum,[3]).

sliding_time_window_sum_r(WINDOW_SIZE,LIMIT,TASKS) :-
        integer(WINDOW_SIZE),
        WINDOW_SIZE>0,
        integer(LIMIT),
```

```
        LIMIT>=0,
        collection(TASKS,[dvar,dvar,dvar_gteq(0)]),
        get_attr1(TASKS,ORIGINS),
        get_attr2(TASKS,ENDS),
        get_attr3(TASKS,NPOINTS),
        sliding_time_window_sum1(
            ORIGINS,
            ENDS,
            NPOINTS,
            1,
            ORIGINS,
            ENDS,
            NPOINTS,
            WINDOW_SIZE,
            LIMIT).

sliding_time_window_sum1(
    [],
    [],
    [],
    _48511,
    _48558,
    _48605,
    _48652,
    _48699,
    _48746).

sliding_time_window_sum1(
    [Oi|RO],
    [Ei|RE],
    [Pi|RP],
    I,
    ORIGINS,
    ENDS,
    NPOINTS,
    WINDOW_SIZE,
    LIMIT) :-
        Oi#=<Ei,
        sliding_time_window_sum2(
            ORIGINS,
            ENDS,
            NPOINTS,
            1,
            Oi,
            Ei,
            Pi,
```

```
            I,
            WINDOW_SIZE,
            LIMIT,
            SUM_NPOINTS),
        call(SUM_NPOINTS#=<LIMIT),
        I1 is I+1,
        sliding_time_window_sum1(
            RO,
            RE,
            RP,
            I1,
            ORIGINS,
            ENDS,
            NPOINTS,
            WINDOW_SIZE,
            LIMIT).

sliding_time_window_sum2(
    [],
    [],
    [],
    _48520,
    _48567,
    _48614,
    _48661,
    _48708,
    _48755,
    _48802,
    0) :-
        !.

sliding_time_window_sum2(
    [_48143|RO],
    [_48147|RE],
    [_48151|RP],
    J,
    Oi,
    Ei,
    Pi,
    I,
    WINDOW_SIZE,
    LIMIT,
    Pi+SUM) :-
        I=J,
        !,
        J1 is J+1,
```

```
        sliding_time_window_sum2(
            RO,
            RE,
            RP,
            J1,
            Oi,
            Ei,
            Pi,
            I,
            WINDOW_SIZE,
            LIMIT,
            SUM).

sliding_time_window_sum2(
    [_48143|RO],
    [Ej|RE],
    [_48153|RP],
    J,
    Oi,
    Ei,
    Pi,
    I,
    WINDOW_SIZE,
    LIMIT,
    SUM) :-
        I=\=J,
        fd_max(Ej,MaxEj),
        fd_min(Oi,MinOi),
        MaxEj<MinOi,
        !,
        J1 is J+1,
        sliding_time_window_sum2(
            RO,
            RE,
            RP,
            J1,
            Oi,
            Ei,
            Pi,
            I,
            WINDOW_SIZE,
            LIMIT,
            SUM).

sliding_time_window_sum2(
    [Oj|RO],
```

```
    [_48149|RE],
    [_48153|RP],
    J,
    Oi,
    Ei,
    Pi,
    I,
    WINDOW_SIZE,
    LIMIT,
    SUM) :-
        I=\=J,
        fd_min(Oj,MinOj),
        fd_max(Oi,MaxOi),
        E is MaxOi+WINDOW_SIZE-1,
        MinOj>E,
        !,
        J1 is J+1,
        sliding_time_window_sum2(
            RO,
            RE,
            RP,
            J1,
            Oi,
            Ei,
            Pi,
            I,
            WINDOW_SIZE,
            LIMIT,
            SUM).

sliding_time_window_sum2(
    [Oj|RO],
    [Ej|RE],
    [Pj|RP],
    J,
    Oi,
    Ei,
    Pi,
    I,
    WINDOW_SIZE,
    LIMIT,
    min(1,max(0,min(Oi+WINDOW_SIZE,Ej)-max(Oi,Oj)))*Pj+SUM) :-
        J1 is J+1,
        sliding_time_window_sum2(
            RO,
            RE,
```

```
RP,
J1,
Oi,
Ei,
Pi,
I,
WINDOW_SIZE,
LIMIT,
SUM).
```

B.362 smooth

◊ **META-DATA:**

```
ctr_date(smooth,['20000128','20030820','20040530','20060815']).

ctr_origin(smooth,'Derived from %c.',[change]).

ctr_arguments(
    smooth,
    ['NCHANGE'-dvar,
     'TOLERANCE'-int,
     'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    smooth,
    ['NCHANGE'>=0,
     'NCHANGE'<size('VARIABLES'),
     'TOLERANCE'>=0,
     required('VARIABLES',var)]).

ctr_example(
    smooth,
    smooth(1,2,[[var-1],[var-3],[var-4],[var-5],[var-2]])).

ctr_typical(
    smooth,
    ['TOLERANCE'>0,
     size('VARIABLES')>3,
     range('VARIABLES'^var)>1]).

ctr_typical_model(smooth,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    smooth,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_graph(
    smooth,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [abs(variables1^var-variables2^var)>'TOLERANCE'],
    ['NARC'='NCHANGE'],
    []).
```

```
ctr_eval(smooth,[checker(smooth_c),automaton(smooth_a)]).

ctr_pure_functional_dependency(smooth,[]).

ctr_functional_dependency(smooth,1,[2,3]).

ctr_contractible(smooth,['NCHANGE'=0],'VARIABLES',prefix).

ctr_contractible(smooth,['NCHANGE'=0],'VARIABLES',suffix).

ctr_contractible(
    smooth,
    ['NCHANGE'=size('VARIABLES')-1],
    VARIABLES,
    prefix).

ctr_contractible(
    smooth,
    ['NCHANGE'=size('VARIABLES')-1],
    VARIABLES,
    suffix).

smooth_a(FLAG,NCHANGE,TOLERANCE,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        N_1 is N-1,
        check_type(dvar(0,N_1),NCHANGE),
        integer(TOLERANCE),
        TOLERANCE>=0,
        smooth_signature(VARIABLES,SIGNATURE,TOLERANCE),
        automaton(
            SIGNATURE,
            _46703,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,1,s,[C+1]),arc(s,0,s)],
            [C],
            [0],
            [COUNT]),
        COUNT#=NCHANGE#<=>FLAG.

smooth_signature([],[],_44851).

smooth_signature([_44855],[],_44854) :-
        !.
```

```
smooth_signature([[var-VAR1],[var-VAR2]|VARs],[S|Ss],TOLERANCE) :-
        abs(VAR1-VAR2)#>TOLERANCE#<=>S#=1,
        smooth_signature([[var-VAR2]|VARs],Ss,TOLERANCE).

smooth_c(NCHANGE,TOLERANCE,VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        integer(NCHANGE),
        NCHANGE>=0,
        NCHANGE<N,
        integer(TOLERANCE),
        TOLERANCE>=0,
        get_attr1(VARIABLES,VARS),
        smooth_check(VARS,TOLERANCE,NCHANGE).

smooth_check([],_44853,0) :-
        !.

smooth_check([_44855],_44853,0) :-
        !.

smooth_check([VAR1,VAR2|R],TOLERANCE,NCHANGE) :-
        D is abs(VAR1-VAR2),
        (   D>TOLERANCE ->
            NCHANGE1 is NCHANGE-1
        ;   NCHANGE1 is NCHANGE
        ),
        NCHANGE1>=0,
        smooth_check([VAR2|R],TOLERANCE,NCHANGE1).
```

B.363 soft_all_equal_max_var

◇ META-DATA:

```
ctr_date(soft_all_equal_max_var,['20090926']).

ctr_origin(
    soft_all_equal_max_var,
    \cite{HebrardMarxSullivanRazgon09},
    []).

ctr_arguments(
    soft_all_equal_max_var,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    soft_all_equal_max_var,
    ['N'>=0,'N'=<size('VARIABLES'),required('VARIABLES',var)]).

ctr_example(
    soft_all_equal_max_var,
    soft_all_equal_max_var(
        1,
        [[var-5],[var-1],[var-5],[var-5]])).

ctr_typical(
    soft_all_equal_max_var,
    ['N'>0,
     'N'<size('VARIABLES'),
     'N'<size('VARIABLES')/10+2,
     size('VARIABLES')>1]).

ctr_exchangeable(
    soft_all_equal_max_var,
    [vals(['N'],int(>=(0)),>,dontcare,dontcare),
     items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    soft_all_equal_max_var,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['MAX_NSCC'=<size('VARIABLES')-'N'],
    []).
```

```
ctr_eval(
    soft_all_equal_max_var,
    [checker(soft_all_equal_max_var_c),
     reformulation(soft_all_equal_max_var_r)]).

ctr_total_relation(soft_all_equal_max_var).

ctr_sol(soft_all_equal_max_var,2,0,2,15,[0-9,1-6]).

ctr_sol(soft_all_equal_max_var,3,0,3,148,[0-64,1-60,2-24]).

ctr_sol(
    soft_all_equal_max_var,
    4,
    0,
    4,
    1905,
    [0-625,1-620,2-540,3-120]).

ctr_sol(
    soft_all_equal_max_var,
    5,
    0,
    5,
    30006,
    [0-7776,1-7770,2-7620,3-6120,4-720]).

ctr_sol(
    soft_all_equal_max_var,
    6,
    0,
    6,
    555121,
    [0-117649,1-117642,2-117390,3-113610,4-83790,5-5040]).

ctr_sol(
    soft_all_equal_max_var,
    7,
    0,
    7,
    11758048,
    [0-2097152,
     1-2097144,
     2-2096752,
     3-2088520,
```

```
     4-1992480,
     5-1345680,
     6-40320]).

ctr_sol(
    soft_all_equal_max_var,
    8,
    0,
    8,
    280310337,
    [0-43046721,
     1-43046712,
     2-43046136,
     3-43030008,
     4-42771960,
     5-40194000,
     6-24811920,
     7-362880]).

soft_all_equal_max_var_c(N,VARIABLES) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARS,L),
        (   L=0 ->
            N#=0
        ;   samsort(VARS,SVARS),
            SVARS=[V|R],
            max_nvalue_seq_size(R,1,V,1,M),
            MAX is L-M,
            N#=<MAX
        ).

soft_all_equal_max_var_r(N,VARIABLES) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        get_attr1(VARIABLES,VARS),
        get_minimum(VARS,MINVARS),
        get_maximum(VARS,MAXVARS),
        complete_card(MINVARS,MAXVARS,L,OCC,VAL_OCC),
        global_cardinality(VARS,VAL_OCC),
        MAX_OCC in 0..L,
        eval(maximum(MAX_OCC,OCC)),
        call(N#=<L-MAX_OCC).
```

B.364 soft_all_equal_min_ctr

◊ **META-DATA:**

```
ctr_date(soft_all_equal_min_ctr,['20081004']).

ctr_origin(
    soft_all_equal_min_ctr,
    \cite{HebrardSullivanRazgon08},
    []).

ctr_synonyms(
    soft_all_equal_min_ctr,
    [soft_alldiff_max_ctr,
     soft_alldifferent_max_ctr,
     soft_alldistinct_max_ctr]).

ctr_arguments(
    soft_all_equal_min_ctr,
    ['N'-int,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    soft_all_equal_min_ctr,
    ['N'>=0,
     N=<
     size('VARIABLES')*size('VARIABLES')-size('VARIABLES'),
     required('VARIABLES',var)]).

ctr_example(
    soft_all_equal_min_ctr,
    soft_all_equal_min_ctr(
        6,
        [[var-5],[var-1],[var-5],[var-5]])).

ctr_typical(
    soft_all_equal_min_ctr,
    ['N'>0,
     N<
     size('VARIABLES')*size('VARIABLES')-size('VARIABLES'),
     size('VARIABLES')>1]).

ctr_exchangeable(
    soft_all_equal_min_ctr,
    [vals(['N'],int(>=(0)),>,dontcare,dontcare),
     items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).
```

```
ctr_graph(
    soft_all_equal_min_ctr,
    ['VARIABLES'],
    2,
    ['CLIQUE'(=\=)>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NARC'>='N'],
    []).

ctr_eval(
    soft_all_equal_min_ctr,
    [checker(soft_all_equal_min_ctr_c),
     reformulation(soft_all_equal_min_ctr_r)]).

ctr_total_relation(soft_all_equal_min_ctr).

soft_all_equal_min_ctr_c(N,VARIABLES) :-
        collection(VARIABLES,[int]),
        length(VARIABLES,L),
        L2 is L*L-L,
        check_type(dvar(0,L2),N),
        (   L=0 ->
            N=0
        ;   get_attr1(VARIABLES,VARS),
            samsort(VARS,SVARS),
            SVARS=[V|R],
            soft_all_equal_min_ctr_c(R,1,V,0,NB_EQ_CTR),
            N#=<NB_EQ_CTR
        ).

soft_all_equal_min_ctr_c([],C,_42471,Sum,Res) :-
        !,
        Res is C*C-C+Sum.

soft_all_equal_min_ctr_c([V|R],C,V,Sum,Res) :-
        !,
        C1 is C+1,
        soft_all_equal_min_ctr_c(R,C1,V,Sum,Res).

soft_all_equal_min_ctr_c([V|R],C,Prev,Sum,Res) :-
        C>0,
        V=\=Prev,
        NewSum is C*C-C+Sum,
        soft_all_equal_min_ctr_c(R,1,V,NewSum,Res).
```

```
soft_all_equal_min_ctr_r(N,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        L2 is L*L-L,
        check_type(dvar(0,L2),N),
        get_attr1(VARIABLES,VARS),
        soft_all_equal_min_ctr1(VARS,TERM),
        call(N#=<TERM).

soft_all_equal_min_ctr1([],0).

soft_all_equal_min_ctr1([V|R],S+T) :-
        soft_all_equal_min_ctr2(R,V,S),
        soft_all_equal_min_ctr1(R,T).

soft_all_equal_min_ctr2([],_42467,0).

soft_all_equal_min_ctr2([U|R],V,2*B+T) :-
        B#<=>U#=V,
        soft_all_equal_min_ctr2(R,V,T).
```

B.365 soft_all_equal_min_var

◇ META-DATA:

```
ctr_date(soft_all_equal_min_var,['20090926']).

ctr_origin(
    soft_all_equal_min_var,
    \cite{HebrardMarxSullivanRazgon09},
    []).

ctr_arguments(
    soft_all_equal_min_var,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    soft_all_equal_min_var,
    ['N'>=0,required('VARIABLES',var)]).

ctr_example(
    soft_all_equal_min_var,
    soft_all_equal_min_var(
        1,
        [[var-5],[var-1],[var-5],[var-5]])).

ctr_typical(
    soft_all_equal_min_var,
    ['N'>0,
     'N'<size('VARIABLES'),
     'N'<size('VARIABLES')/10+2,
     size('VARIABLES')>1]).

ctr_exchangeable(
    soft_all_equal_min_var,
    [vals(['N'],int,<,dontcare,dontcare),
     items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    soft_all_equal_min_var,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['MAX_NSCC'>=size('VARIABLES')-'N'],
    []).
```

```
ctr_eval(
    soft_all_equal_min_var,
    [checker(soft_all_equal_min_var_c),
     reformulation(soft_all_equal_min_var_r)]).

ctr_total_relation(soft_all_equal_min_var).

ctr_sol(soft_all_equal_min_var,2,0,2,21,[0-3,1-9,2-9]).

ctr_sol(soft_all_equal_min_var,3,0,3,172,[0-4,1-40,2-64,3-64]).

ctr_sol(
    soft_all_equal_min_var,
    4,
    0,
    4,
    1845,
    [0-5,1-85,2-505,3-625,4-625]).

ctr_sol(
    soft_all_equal_min_var,
    5,
    0,
    5,
    24426,
    [0-6,1-156,2-1656,3-7056,4-7776,5-7776]).

ctr_sol(
    soft_all_equal_min_var,
    6,
    0,
    6,
    386071,
    [0-7,1-259,2-4039,3-33859,4-112609,5-117649,6-117649]).

ctr_sol(
    soft_all_equal_min_var,
    7,
    0,
    7,
    7116320,
    [0-8,
     1-400,
     2-8632,
     3-104672,
```

```
     4-751472,
     5-2056832,
     6-2097152,
     7-2097152]).

ctr_sol(
    soft_all_equal_min_var,
    8,
    0,
    8,
    150156873,
    [0-9,
     1-585,
     2-16713,
     3-274761,
     4-2852721,
     5-18234801,
     6-42683841,
     7-43046721,
     8-43046721]).

soft_all_equal_min_var_c(N,VARIABLES) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        length(VARS,L),
        (   L=0 ->
            true
        ;   samsort(VARS,SVARS),
            SVARS=[V|R],
            max_nvalue_seq_size(R,1,V,1,M),
            MAX is L-M,
            N#>=MAX
        ).

soft_all_equal_min_var_r(N,VARIABLES) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        get_attr1(VARIABLES,VARS),
        get_minimum(VARS,MINVARS),
        get_maximum(VARS,MAXVARS),
        complete_card(MINVARS,MAXVARS,L,OCC,VAL_OCC),
        global_cardinality(VARS,VAL_OCC),
        MAX_OCC in 0..L,
        eval(maximum(MAX_OCC,OCC)),
```

```
call(N#>=L-MAX_OCC).
```

B.366 soft_alldifferent_ctr

◇ **META-DATA:**

```
ctr_date(
    soft_alldifferent_ctr,
    ['20030820','20060815','20090926']).

ctr_origin(
    soft_alldifferent_ctr,
    \cite{PetitReginBessiere01},
    []).

ctr_synonyms(
    soft_alldifferent_ctr,
    [soft_alldiff_ctr,
     soft_alldistinct_ctr,
     soft_alldiff_min_ctr,
     soft_alldifferent_min_ctr,
     soft_alldistinct_min_ctr,
     soft_all_equal_max_ctr]).

ctr_arguments(
    soft_alldifferent_ctr,
    ['C'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    soft_alldifferent_ctr,
    ['C'>=0,required('VARIABLES',var)]).

ctr_example(
    soft_alldifferent_ctr,
    [soft_alldifferent_ctr(
         4,
         [[var-5],[var-1],[var-9],[var-1],[var-5],[var-5]]),
     soft_alldifferent_ctr(
         1,
         [[var-5],[var-1],[var-9],[var-1],[var-2],[var-6]]),
     soft_alldifferent_ctr(
         0,
         [[var-5],[var-1],[var-9],[var-0],[var-2],[var-6]])]).

ctr_typical(
    soft_alldifferent_ctr,
    ['C'>0,
     'C'=<size('VARIABLES')*(size('VARIABLES')-1)/2,
```

```
     size('VARIABLES')>1]).

ctr_exchangeable(
    soft_alldifferent_ctr,
    [vals(['C'],int,<,dontcare,dontcare),
     items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    soft_alldifferent_ctr,
    ['VARIABLES'],
    2,
    ['CLIQUE'(<)>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NARC'=<'C'],
    []).

ctr_eval(
    soft_alldifferent_ctr,
    [checker(soft_alldifferent_ctr_c),
     reformulation(soft_alldifferent_ctr_r)]).

ctr_contractible(soft_alldifferent_ctr,[],'VARIABLES',any).

ctr_total_relation(soft_alldifferent_ctr).

ctr_sol(soft_alldifferent_ctr,2,0,2,15,[0-6,1-9]).

ctr_sol(soft_alldifferent_ctr,3,0,3,208,[0-24,1-60,2-60,3-64]).

ctr_sol(
    soft_alldifferent_ctr,
    4,
    0,
    4,
    3625,
    [0-120,1-480,2-540,3-620,4-620,5-620,6-625]).

ctr_sol(
    soft_alldifferent_ctr,
    5,
    0,
    5,
    72576,
    [0-720,
     1-4320,
```

```
     2-6120,
     3-7320,
     4-7620,
     5-7620,
     6-7770,
     7-7770,
     8-7770,
     9-7770,
     10-7776]).

ctr_sol(
    soft_alldifferent_ctr,
    6,
    0,
    6,
    1630279,
    [0-5040,
     1-42840,
     2-80640,
     3-100590,
     4-113190,
     5-113190,
     6-116760,
     7-117390,
     8-117390,
     9-117390,
     10-117642,
     11-117642,
     12-117642,
     13-117642,
     14-117642,
     15-117649]).

ctr_sol(
    soft_alldifferent_ctr,
    7,
    0,
    7,
    40632320,
    [0-40320,
     1-463680,
     2-1169280,
     3-1580880,
     4-1933680,
     5-1968960,
     6-2051280,
```

```
      7-2086560,
      8-2086560,
      9-2088520,
      10-2095576,
      11-2096752,
      12-2096752,
      13-2096752,
      14-2096752,
      15-2097144,
      16-2097144,
      17-2097144,
      18-2097144,
      19-2097144,
      20-2097144,
      21-2097152]).

ctr_sol(
    soft_alldifferent_ctr,
    8,
    0,
    8,
    1114431777,
    [0-362880,
     1-5443200,
     2-18144000,
     3-27881280,
     4-36666000,
     5-39206160,
     6-41111280,
     7-42522480,
     8-42628320,
     9-42769440,
     10-42938784,
     11-43023456,
     12-43025976,
     13-43030008,
     14-43030008,
     15-43044120,
     16-43046136,
     17-43046136,
     18-43046136,
     19-43046136,
     20-43046136,
     21-43046712,
     22-43046712,
     23-43046712,
```

```
     24-43046712,
     25-43046712,
     26-43046712,
     27-43046712,
     28-43046721]).

soft_alldifferent_ctr_r(C,[]) :-
        !,
        check_type(dvar_gteq(0),C).

soft_alldifferent_ctr_r(C,VARIABLES) :-
        length(VARIABLES,N),
        N2 is(N*N-N)//2,
        check_type(dvar(0,N2),C),
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        soft_alldifferent_ctr1(VARS,TERM),
        call(C#>=TERM).

soft_alldifferent_ctr1([],0).

soft_alldifferent_ctr1([V|R],S+T) :-
        soft_alldifferent_ctr2(R,V,S),
        soft_alldifferent_ctr1(R,T).

soft_alldifferent_ctr2([],_83491,0).

soft_alldifferent_ctr2([U|R],V,B+T) :-
        B#<=>U#=V,
        soft_alldifferent_ctr2(R,V,T).

soft_alldifferent_ctr_c(C,[]) :-
        !,
        check_type(dvar_gteq(0),C).

soft_alldifferent_ctr_c(C,VARIABLES) :-
        length(VARIABLES,N),
        N2 is(N*N-N)//2,
        check_type(dvar(0,N2),C),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        create_pairs(VARS,KVARS),
        keysort(KVARS,SVARS),
        SVARS=[VAL-_83575|REST],
        soft_alldifferent_ctr3(REST,VAL,1,0,COST),
        C#>=COST.
```

```
soft_alldifferent_ctr3([],_83494,CPT,CUR,RES) :-
        !,
        (   CPT=1 ->
            RES is CUR
        ;   RES is CUR+CPT*(CPT-1)//2
        ).

soft_alldifferent_ctr3([V-V|R],V,CPT,CUR,RES) :-
        !,
        CPT1 is CPT+1,
        soft_alldifferent_ctr3(R,V,CPT1,CUR,RES).

soft_alldifferent_ctr3([V-V|R],_83494,CPT,CUR,RES) :-
        (   CPT=1 ->
            NEXT is CUR
        ;   NEXT is CUR+CPT*(CPT-1)//2
        ),
        soft_alldifferent_ctr3(R,V,1,NEXT,RES).
```

B.367 soft_alldifferent_var

◇ **META-DATA:**

```
ctr_date(
    soft_alldifferent_var,
    ['20030820','20060815','20090926']).

ctr_origin(
    soft_alldifferent_var,
    \cite{PetitReginBessiere01},
    []).

ctr_synonyms(
    soft_alldifferent_var,
    [soft_alldiff_var,
     soft_alldistinct_var,
     soft_alldiff_min_var,
     soft_alldifferent_min_var,
     soft_alldistinct_min_var]).

ctr_arguments(
    soft_alldifferent_var,
    ['C'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    soft_alldifferent_var,
    ['C'>=0,required('VARIABLES',var)]).

ctr_example(
    soft_alldifferent_var,
    [soft_alldifferent_var(
         3,
         [[var-5],[var-1],[var-9],[var-1],[var-5],[var-5]]),
     soft_alldifferent_var(
         1,
         [[var-5],[var-1],[var-9],[var-6],[var-5],[var-3]]),
     soft_alldifferent_var(
         0,
         [[var-8],[var-1],[var-9],[var-6],[var-5],[var-3]])]).

ctr_typical(
    soft_alldifferent_var,
    ['C'>0,
     2*'C'=<size('VARIABLES'),
     size('VARIABLES')>1,
```

```
     some_equal('VARIABLES')]).

ctr_exchangeable(
    soft_alldifferent_var,
    [vals(['C'],int,<,dontcare,dontcare),
     items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    soft_alldifferent_var,
    ['VARIABLES'],
    2,
    ['CLIQUE'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSCC'>=size('VARIABLES')-'C'],
    []).

ctr_eval(
    soft_alldifferent_var,
    [checker(soft_alldifferent_var_c),
     reformulation(soft_alldifferent_var_r)]).

ctr_contractible(soft_alldifferent_var,[],'VARIABLES',any).

ctr_total_relation(soft_alldifferent_var).

ctr_sol(soft_alldifferent_var,2,0,2,24,[0-6,1-9,2-9]).

ctr_sol(soft_alldifferent_var,3,0,3,212,[0-24,1-60,2-64,3-64]).

ctr_sol(
    soft_alldifferent_var,
    4,
    0,
    4,
    2470,
    [0-120,1-480,2-620,3-625,4-625]).

ctr_sol(
    soft_alldifferent_var,
    5,
    0,
    5,
    35682,
    [0-720,1-4320,2-7320,3-7770,4-7776,5-7776]).
```

```
ctr_sol(
    soft_alldifferent_var,
    6,
    0,
    6,
    614600,
    [0-5040,
     1-42840,
     2-97440,
     3-116340,
     4-117642,
     5-117649,
     6-117649]).

ctr_sol(
    soft_alldifferent_var,
    7,
    0,
    7,
    12286024,
    [0-40320,
     1-463680,
     2-1404480,
     3-1992480,
     4-2093616,
     5-2097144,
     6-2097152,
     7-2097152]).

ctr_sol(
    soft_alldifferent_var,
    8,
    0,
    8,
    279472266,
    [0-362880,
     1-5443200,
     2-21530880,
     3-37406880,
     4-42550704,
     5-43037568,
     6-43046712,
     7-43046721,
     8-43046721]).

soft_alldifferent_var_r(C,[]) :-
```

```
        !,
        check_type(dvar_gteq(0),C).

soft_alldifferent_var_r(C,VARIABLES) :-
        check_type(dvar_gteq(0),C),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,N),
        eval(in_interval(M,1,N)),
        eval(nvalue(M,VARIABLES)),
        C#>=N-M.

soft_alldifferent_var_c(C,[]) :-
        !,
        check_type(dvar_gteq(0),C).

soft_alldifferent_var_c(C,VARIABLES) :-
        check_type(dvar_gteq(0),C),
        collection(VARIABLES,[int]),
        length(VARIABLES,N),
        get_attr1(VARIABLES,VARS),
        sort(VARS,SVARS),
        length(SVARS,M),
        NM is N-M,
        C#>=NM.
```

B.368 soft_cumulative

◊ **META-DATA:**

```
ctr_predefined(soft_cumulative).

ctr_date(soft_cumulative,['20091121']).

ctr_origin(soft_cumulative,'Derived from %c',[cumulative]).

ctr_arguments(
    soft_cumulative,
    [TASKS-
     collection(
         origin-dvar,
         duration-dvar,
         end-dvar,
         height-dvar),
     'LIMIT'-int,
     'INTERMEDIATE_LEVEL'-int,
     'SURFACE_ON_TOP'-dvar]).

ctr_restrictions(
    soft_cumulative,
    [require_at_least(2,'TASKS',[origin,duration,end]),
     required('TASKS',height),
     'TASKS'^duration>=0,
     'TASKS'^origin=<'TASKS'^end,
     'TASKS'^height>=0,
     'LIMIT'>=0,
     'INTERMEDIATE_LEVEL'>=0,
     'INTERMEDIATE_LEVEL'=<'LIMIT',
     'SURFACE_ON_TOP'>=0]).

ctr_example(
    soft_cumulative,
    soft_cumulative(
        [[origin-1,duration-4,end-5,height-1],
         [origin-1,duration-1,end-2,height-2],
         [origin-3,duration-3,end-6,height-2]],
        3,
        2,
        3)).

ctr_typical(
    soft_cumulative,
```

```
    [size('TASKS')>1,
     range('TASKS'^origin)>1,
     range('TASKS'^duration)>1,
     range('TASKS'^end)>1,
     range('TASKS'^height)>1,
     'TASKS'^duration>0,
     'TASKS'^height>0,
     'LIMIT'<sum('TASKS'^height),
     'INTERMEDIATE_LEVEL'>0,
     'INTERMEDIATE_LEVEL'<'LIMIT',
     'SURFACE_ON_TOP'>0]).

ctr_exchangeable(
    soft_cumulative,
    [items('TASKS',all),
     translate(['TASKS'^origin,'TASKS'^end]),
     vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_application(soft_cumulative,[1]).
```

B.369 soft_same_interval_var

◇ **META-DATA:**

```
ctr_date(soft_same_interval_var,['20050507','20060815']).

ctr_origin(
    soft_same_interval_var,
    Derived from %c,
    [same_interval]).

ctr_synonyms(soft_same_interval_var,[soft_same_interval]).

ctr_arguments(
    soft_same_interval_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    soft_same_interval_var,
    ['C'>=0,
     'C'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'SIZE_INTERVAL'>0]).

ctr_example(
    soft_same_interval_var,
    soft_same_interval_var(
        4,
        [[var-9],[var-9],[var-9],[var-9],[var-9],[var-1]],
        [[var-9],[var-1],[var-1],[var-1],[var-1],[var-8]],
        3)).

ctr_typical(
    soft_same_interval_var,
    ['C'>0,
     size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     'SIZE_INTERVAL'>1,
     'SIZE_INTERVAL'<range('VARIABLES1'^var),
     'SIZE_INTERVAL'<range('VARIABLES2'^var)]).
```

```
ctr_exchangeable(
    soft_same_interval_var,
    [args(
         [['C'],
          ['VARIABLES1','VARIABLES2'],
          ['SIZE_INTERVAL']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    soft_same_interval_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var/'SIZE_INTERVAL'=
     variables2^var/'SIZE_INTERVAL'],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C'],
    []).

ctr_eval(
    soft_same_interval_var,
    [reformulation(soft_same_interval_var_r)]).

soft_same_interval_var_r(C,VARIABLES1,VARIABLES2,SIZE_INTERVAL) :-
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1=L2,
        check_type(dvar(0,L1),C),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
        get_attr1(VARIABLES1,VARS1),
```

```
get_attr1(VARIABLES2,VARS2),
gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
gen_collection(QUOTVARS1,var,CVARS1),
gen_collection(QUOTVARS2,var,CVARS2),
eval(soft_same_var(C,CVARS1,CVARS2)).
```

B.370 soft_same_modulo_var

◇ **META-DATA:**

```
ctr_date(soft_same_modulo_var,['20050507','20060815']).

ctr_origin(
    soft_same_modulo_var,
    Derived from %c,
    [same_modulo]).

ctr_synonyms(soft_same_modulo_var,[soft_same_modulo]).

ctr_arguments(
    soft_same_modulo_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    soft_same_modulo_var,
    ['C'>=0,
     'C'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'M'>0]).

ctr_example(
    soft_same_modulo_var,
    soft_same_modulo_var(
        4,
        [[var-9],[var-9],[var-9],[var-9],[var-9],[var-1]],
        [[var-9],[var-1],[var-1],[var-1],[var-1],[var-8]],
        3)).

ctr_typical(
    soft_same_modulo_var,
    ['C'>0,
     size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     'M'>1,
     'M'<maxval('VARIABLES1'^var),
     'M'<maxval('VARIABLES2'^var)]).
```

```
ctr_exchangeable(
    soft_same_modulo_var,
    [args([['C'],['VARIABLES1','VARIABLES2'],['M']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(['VARIABLES1'^var],mod('M'),=,dontcare,dontcare),
     vals(['VARIABLES2'^var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
    soft_same_modulo_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C'],
    []).

ctr_eval(
    soft_same_modulo_var,
    [reformulation(soft_same_modulo_var_r)]).

soft_same_modulo_var_r(C,VARIABLES1,VARIABLES2,M) :-
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1=L2,
        check_type(dvar(0,L1),C),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        integer(M),
        M>0,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        gen_remainder(VARS1,M,REMVARS1),
        gen_remainder(VARS2,M,REMVARS2),
        gen_collection(REMVARS1,var,CVARS1),
        gen_collection(REMVARS2,var,CVARS2),
        eval(soft_same_var(C,CVARS1,CVARS2)).
```

B.371 soft_same_partition_var

◊ **META-DATA:**

```
ctr_date(soft_same_partition_var,['20050507','20060816']).

ctr_origin(
    soft_same_partition_var,
    Derived from %c,
    [same_partition]).

ctr_synonyms(soft_same_partition_var,[soft_same_partition]).

ctr_types(
    soft_same_partition_var,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    soft_same_partition_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    soft_same_partition_var,
    ['C'>=0,
     'C'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2,
     size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    soft_same_partition_var,
    soft_same_partition_var(
        4,
        [[var-9],[var-9],[var-9],[var-9],[var-9],[var-1]],
        [[var-9],[var-1],[var-1],[var-1],[var-1],[var-8]],
        [[p-[[val-1],[val-2]]],
         [p-[[val-9]]],
         [p-[[val-7],[val-8]]]])).
```

```
ctr_typical(
    soft_same_partition_var,
    ['C'>0,
     size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     size('VARIABLES1')>size('PARTITIONS'),
     size('VARIABLES2')>size('PARTITIONS')]).

ctr_exchangeable(
    soft_same_partition_var,
    [args([['C'],['VARIABLES1','VARIABLES2'],['PARTITIONS']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['VARIABLES1'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    soft_same_partition_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C'],
    []).

ctr_eval(
    soft_same_partition_var,
    [reformulation(soft_same_partition_var_r)]).
```

```
soft_same_partition_var_r(C,VARIABLES1,VARIABLES2,PARTITIONS) :-
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1=L2,
        check_type(dvar(0,L1),C),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        get_col_attr1(PARTITIONS,1,PVALS),
        flattern(PVALS,VALS),
        all_different(VALS),
        length(PARTITIONS,M),
        M>1,
        length(PVALS,LPVALS),
        get_partition_var(VARS1,PVALS,PVARS1,LPVALS,0),
        get_partition_var(VARS2,PVALS,PVARS2,LPVALS,0),
        gen_collection(PVARS1,var,CVARS1),
        gen_collection(PVARS2,var,CVARS2),
        eval(soft_same_var(C,CVARS1,CVARS2)).
```

B.372 soft_same_var

◇ **META-DATA:**

```
ctr_date(soft_same_var,['20050507','20060816','20090522']).

ctr_origin(soft_same_var,'\\cite{vanHoeve05}',[]).

ctr_synonyms(soft_same_var,[soft_same]).

ctr_arguments(
    soft_same_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    soft_same_var,
    ['C'>=0,
     'C'=<size('VARIABLES1'),
     size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    soft_same_var,
    soft_same_var(
        4,
        [[var-9],[var-9],[var-9],[var-9],[var-9],[var-1]],
        [[var-9],[var-1],[var-1],[var-1],[var-1],[var-8]])).

ctr_typical(
    soft_same_var,
    ['C'>0,
     size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1]).

ctr_exchangeable(
    soft_same_var,
    [args([['C'],['VARIABLES1','VARIABLES2']]),
     items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
```

```
         =\=,
         all,
         dontcare)]).

ctr_graph(
    soft_same_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSINK_NSOURCE'=size('VARIABLES1')-'C'],
    []).

ctr_eval(soft_same_var,[reformulation(soft_same_var_r)]).

soft_same_var_r(C,VARIABLES1,VARIABLES2) :-
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1=L2,
        check_type(dvar(0,L1),C),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        eval(soft_used_by_var(C,VARIABLES1,VARIABLES2)).
```

B.373 soft_used_by_interval_var

◇ META-DATA:

```
ctr_date(soft_used_by_interval_var,['20050507','20060816']).

ctr_origin(
    soft_used_by_interval_var,
    Derived from %c.,
    [used_by_interval]).

ctr_synonyms(soft_used_by_interval_var,[soft_used_by_interval]).

ctr_arguments(
    soft_used_by_interval_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    soft_used_by_interval_var,
    ['C'>=0,
     'C'=<size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'SIZE_INTERVAL'>0]).

ctr_example(
    soft_used_by_interval_var,
    soft_used_by_interval_var(
        2,
        [[var-9],[var-1],[var-1],[var-8],[var-8]],
        [[var-9],[var-9],[var-9],[var-1]],
        3)).

ctr_typical(
    soft_used_by_interval_var,
    ['C'>0,
     size('VARIABLES1')>1,
     size('VARIABLES2')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     'SIZE_INTERVAL'>1,
     'SIZE_INTERVAL'<range('VARIABLES1'^var),
```

```
     'SIZE_INTERVAL'<range('VARIABLES2'^var)]).

ctr_exchangeable(
    soft_used_by_interval_var,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    soft_used_by_interval_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var/'SIZE_INTERVAL'=
     variables2^var/'SIZE_INTERVAL'],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C'],
    []).

ctr_eval(
    soft_used_by_interval_var,
    [reformulation(soft_used_by_interval_var_r)]).

soft_used_by_interval_var_r(
    C,
    VARIABLES1,
    VARIABLES2,
    SIZE_INTERVAL) :-
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1>=L2,
        check_type(dvar(0,L2),C),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
```

```
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
gen_collection(QUOTVARS1,var,CVARS1),
gen_collection(QUOTVARS2,var,CVARS2),
eval(soft_used_by_var(C,CVARS1,CVARS2)).
```

B.374 soft_used_by_modulo_var

◊ **META-DATA:**

```
ctr_date(soft_used_by_modulo_var,['20050507','20060816']).

ctr_origin(
    soft_used_by_modulo_var,
    Derived from %c,
    [used_by_modulo]).

ctr_synonyms(soft_used_by_modulo_var,[soft_used_by_modulo]).

ctr_arguments(
    soft_used_by_modulo_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    soft_used_by_modulo_var,
    ['C'>=0,
     'C'=<size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'M'>0]).

ctr_example(
    soft_used_by_modulo_var,
    soft_used_by_modulo_var(
        2,
        [[var-9],[var-1],[var-1],[var-8],[var-8]],
        [[var-9],[var-9],[var-9],[var-1]],
        3)).

ctr_typical(
    soft_used_by_modulo_var,
    ['C'>0,
     size('VARIABLES1')>1,
     size('VARIABLES2')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     'M'>1,
     'M'<maxval('VARIABLES1'^var),
```

```
     'M'<maxval('VARIABLES2'^var)]).

ctr_exchangeable(
    soft_used_by_modulo_var,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(['VARIABLES1'^var],mod('M'),=,dontcare,dontcare),
     vals(['VARIABLES2'^var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
    soft_used_by_modulo_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C'],
    []).

ctr_eval(
    soft_used_by_modulo_var,
    [reformulation(soft_used_by_modulo_var_r)]).

soft_used_by_modulo_var_r(C,VARIABLES1,VARIABLES2,M) :-
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1>=L2,
        check_type(dvar(0,L2),C),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        integer(M),
        M>0,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        gen_remainder(VARS1,M,REMVARS1),
        gen_remainder(VARS2,M,REMVARS2),
        gen_collection(REMVARS1,var,CVARS1),
        gen_collection(REMVARS2,var,CVARS2),
        eval(soft_used_by_var(C,CVARS1,CVARS2)).
```

B.375 soft_used_by_partition_var

◇ **META-DATA:**

```
ctr_date(soft_used_by_partition_var,['20050507','20060816']).

ctr_origin(
    soft_used_by_partition_var,
    Derived from %c.,
    [used_by_partition]).

ctr_synonyms(
    soft_used_by_partition_var,
    [soft_used_by_partition]).

ctr_types(
    soft_used_by_partition_var,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    soft_used_by_partition_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    soft_used_by_partition_var,
    ['C'>=0,
     'C'=<size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2,
     size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    soft_used_by_partition_var,
    soft_used_by_partition_var(
        2,
        [[var-9],[var-1],[var-1],[var-8],[var-8]],
        [[var-9],[var-9],[var-9],[var-1]],
        [[p-[[val-1],[val-2]]],
```

```
         [p-[[val-9]]],
         [p-[[val-7],[val-8]]]]])).

ctr_typical(
    soft_used_by_partition_var,
    ['C'>0,
     size('VARIABLES1')>1,
     size('VARIABLES2')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1,
     size('VARIABLES1')>size('PARTITIONS'),
     size('VARIABLES2')>size('PARTITIONS')]).

ctr_exchangeable(
    soft_used_by_partition_var,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['VARIABLES1'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    soft_used_by_partition_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C'],
    []).

ctr_eval(
    soft_used_by_partition_var,
```

```
    [reformulation(soft_used_by_partition_var_r)]).

soft_used_by_partition_var_r(
    C,
    VARIABLES1,
    VARIABLES2,
    PARTITIONS) :-
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1>=L2,
        check_type(dvar(0,L2),C),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        get_col_attr1(PARTITIONS,1,PVALS),
        flattern(PVALS,VALS),
        all_different(VALS),
        length(PARTITIONS,M),
        M>1,
        length(PVALS,LPVALS),
        get_partition_var(VARS1,PVALS,PVARS1,LPVALS,0),
        get_partition_var(VARS2,PVALS,PVARS2,LPVALS,0),
        gen_collection(PVARS1,var,CVARS1),
        gen_collection(PVARS2,var,CVARS2),
        eval(soft_used_by_var(C,CVARS1,CVARS2)).
```

B.376 soft_used_by_var

◇ **META-DATA:**

```
ctr_date(soft_used_by_var,['20050507','20060816']).

ctr_origin(soft_used_by_var,'Derived from %c',[used_by]).

ctr_synonyms(soft_used_by_var,[soft_used_by]).

ctr_arguments(
    soft_used_by_var,
    ['C'-dvar,
     'VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    soft_used_by_var,
    ['C'>=0,
     'C'=<size('VARIABLES2'),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    soft_used_by_var,
    soft_used_by_var(
        2,
        [[var-9],[var-1],[var-1],[var-8],[var-8]],
        [[var-9],[var-9],[var-9],[var-1]])).

ctr_typical(
    soft_used_by_var,
    ['C'>0,
     size('VARIABLES1')>1,
     size('VARIABLES2')>1,
     range('VARIABLES1'^var)>1,
     range('VARIABLES2'^var)>1]).

ctr_exchangeable(
    soft_used_by_var,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
```

```
        =\=,
        all,
        dontcare)]).

ctr_graph(
    soft_used_by_var,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSINK_NSOURCE'=size('VARIABLES2')-'C'],
    []).

ctr_eval(soft_used_by_var,[reformulation(soft_used_by_var_r)]).

soft_used_by_var_r(C,VARIABLES1,VARIABLES2) :-
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1>=L2,
        check_type(dvar(0,L2),C),
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        get_attr1(VARIABLES2,VARS2),
        get_minimum(VARS2,MINVARS2),
        get_maximum(VARS2,MAXVARS2),
        soft_used_by_var1(
            MINVARS2,
            MAXVARS2,
            L1,
            OCCS1,
            OCCS2,
            TERM),
        eval(global_cardinality(VARIABLES1,OCCS1)),
        eval(global_cardinality(VARIABLES2,OCCS2)),
        call(C#=TERM).

soft_used_by_var1(I,S,_44605,[],[],0) :-
        I>S,
        !.

soft_used_by_var1(
    I,
    S,
    MAX,
    [[val-I,noccurrence-O1]|R1],
    [[val-I,noccurrence-O2]|R2],
```

```
max(O2-O1,0)+R) :-
    I=<S,
    O1 in 0..MAX,
    O2 in 0..MAX,
    I1 is I+1,
    soft_used_by_var1(I1,S,MAX,R1,R2,R).
```

B.377 some_equal

◊ **META-DATA:**

```
ctr_date(some_equal,['20110604']).

ctr_origin(some_equal,'Derived from %c',[alldifferent]).

ctr_synonyms(
    some_equal,
    [some_eq,
     not_alldifferent,
     not_alldiff,
     not_alldistinct,
     not_distinct]).

ctr_arguments(some_equal,['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    some_equal,
    [required('VARIABLES',var),size('VARIABLES')>1]).

ctr_example(
    some_equal,
    some_equal([[var-1],[var-4],[var-1],[var-6]])).

ctr_typical(
    some_equal,
    [size('VARIABLES')>2,nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    some_equal,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,dontcare)]).

ctr_graph(
    some_equal,
    ['VARIABLES'],
    2,
    ['CLIQUE'(<)>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NARC'>0],
    []).

ctr_eval(
    some_equal,
```

```
    [checker(some_equal_c),reformulation(some_equal_r)]).

ctr_extensible(some_equal,[],'VARIABLES',any).

ctr_sol(some_equal,2,0,2,3,-).

ctr_sol(some_equal,3,0,3,40,-).

ctr_sol(some_equal,4,0,4,505,-).

ctr_sol(some_equal,5,0,5,7056,-).

ctr_sol(some_equal,6,0,6,112609,-).

ctr_sol(some_equal,7,0,7,2056832,-).

ctr_sol(some_equal,8,0,8,42683841,-).

some_equal_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        sort(VARS,S),
        length(VARS,M),
        length(S,N),
        N<M.

some_equal_r(VARIABLES) :-
        collection(VARIABLES,[dvar]),
        get_attr1(VARIABLES,VARS),
        length(VARS,M),
        M>1,
        M1 is M-1,
        N in 1..M1,
        nvalue(N,VARS).
```

B.378 sort

◇ **META-DATA:**

```
ctr_date(sort,['20030820','20060816']).

ctr_origin(sort,'\\cite{OlderSwinkelsEmden95}',[]).

ctr_synonyms(sort,[sortedness,sorted,sorting]).

ctr_arguments(
    sort,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    sort,
    [size('VARIABLES1')=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    sort,
    sort(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-1],[var-1],[var-1],[var-2],[var-5],[var-9]])).

ctr_typical(
    sort,
    [size('VARIABLES1')>1,range('VARIABLES1'^var)>1]).

ctr_exchangeable(
    sort,
    [items('VARIABLES1',all),
     translate(['VARIABLES1'^var,'VARIABLES2'^var])]).

ctr_graph(
    sort,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    [for_all('CC','NSOURCE'='NSINK'),
     'NSOURCE'=size('VARIABLES1'),
     'NSINK'=size('VARIABLES2')],
    []).
```

```
ctr_graph(
    sort,
    ['VARIABLES2'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var=<variables2^var],
    ['NARC'=size('VARIABLES2')-1],
    []).

ctr_eval(sort,[reformulation(sort_r),checker(sort_c)]).

ctr_pure_functional_dependency(sort,[]).

ctr_functional_dependency(sort,2,[1]).

sort_r(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        L1=L2,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        length(P,L1),
        domain(P,1,L1),
        sorting(VARS1,P,VARS2),
        when(ground(VARS1),once(labeling([],P))).

sort_c(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[int]),
        collection(VARIABLES2,[int]),
        length(VARIABLES1,L),
        length(VARIABLES2,L),
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        create_pairs(VARS1,PVARS1),
        create_pairs(VARS2,PVARS2),
        keysort(PVARS1,PVARS2).
```

B.379 sort_permutation

◊ **META-DATA:**

```
ctr_date(sort_permutation,['20030820','20060816','20111025']).

ctr_origin(sort_permutation,'\\cite{ZhouCP96}',[]).

ctr_usual_name(sort_permutation,sort).

ctr_synonyms(
    sort_permutation,
    [extended_sortedness,sortedness,sorted,sorting]).

ctr_arguments(
    sort_permutation,
    ['FROM'-collection(var-dvar),
     'PERMUTATION'-collection(var-dvar),
     'TO'-collection(var-dvar)]).

ctr_restrictions(
    sort_permutation,
    [size('PERMUTATION')=size('FROM'),
     size('PERMUTATION')=size('TO'),
     'PERMUTATION'^var>=1,
     'PERMUTATION'^var=<size('PERMUTATION'),
     alldifferent('PERMUTATION'),
     required('FROM',var),
     required('PERMUTATION',var),
     required('TO',var)]).

ctr_example(
    sort_permutation,
    sort_permutation(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-1],[var-6],[var-3],[var-5],[var-4],[var-2]],
        [[var-1],[var-1],[var-1],[var-2],[var-5],[var-9]])).

ctr_typical(
    sort_permutation,
    [size('FROM')>1,
     range('FROM'^var)>1,
     lex_different('FROM','TO')]).

ctr_exchangeable(
    sort_permutation,
```

```
    [translate(['FROM'^var,'TO'^var])]).

ctr_derived_collections(
    sort_permutation,
    [col('FROM_PERMUTATION'-collection(var-dvar,ind-dvar),
         [item(var-'FROM'^var,ind-'PERMUTATION'^var)])]).

ctr_graph(
    sort_permutation,
    ['FROM_PERMUTATION','TO'],
    2,
    ['PRODUCT'>>collection(from_permutation,to)],
    [from_permutation^var=to^var,from_permutation^ind=to^key],
    ['NARC'=size('PERMUTATION')],
    []).

ctr_graph(
    sort_permutation,
    ['TO'],
    2,
    ['PATH'>>collection(to1,to2)],
    [to1^var=<to2^var],
    ['NARC'=size('TO')-1],
    []).

ctr_eval(sort_permutation,[builtin(sort_permutation_b)]).

ctr_functional_dependency(sort_permutation,3,[1]).

sort_permutation_b(FROM,PERMUTATION,TO) :-
        length(FROM,F),
        length(PERMUTATION,P),
        length(TO,T),
        F=P,
        P=T,
        collection(FROM,[dvar]),
        collection(PERMUTATION,[dvar(1,P)]),
        collection(TO,[dvar]),
        get_attr1(FROM,FVARS),
        get_attr1(PERMUTATION,PVARS),
        get_attr1(TO,TVARS),
        sorting(FVARS,PVARS,TVARS).
```

B.380 stable_compatibility

◊ **META-DATA:**

```
ctr_date(stable_compatibility,['20070601']).

ctr_origin(
    stable_compatibility,
    P.~Flener, \cite{BeldiceanuFlenerLorca06},
    []).

ctr_arguments(
    stable_compatibility,
    [NODES-
     collection(index-int,father-dvar,prec-sint,inc-sint)]).

ctr_restrictions(
    stable_compatibility,
    [required('NODES',[index,father,prec,inc]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^father>=1,
     'NODES'^father=<size('NODES'),
     'NODES'^prec>=1,
     'NODES'^prec=<size('NODES'),
     'NODES'^inc>=1,
     'NODES'^inc=<size('NODES'),
     'NODES'^inc>'NODES'^index]).

ctr_example(
    stable_compatibility,
    stable_compatibility(
        [[index-1,father-4,prec-{11,12},inc-{}],
         [index-2,father-3,prec-{8,9},inc-{}],
         [index-3,father-4,prec-{2,10},inc-{}],
         [index-4,father-5,prec-{1,3},inc-{}],
         [index-5,father-7,prec-{4,13},inc-{}],
         [index-6,father-2,prec-{8,14},inc-{}],
         [index-7,father-7,prec-{6,13},inc-{}],
         [index-8,father-6,prec-{},inc-{9,10,11,12,13,14}],
         [index-9,father-2,prec-{},inc-{10,11,12,13}],
         [index-10,father-3,prec-{},inc-{11,12,13}],
         [index-11,father-1,prec-{},inc-{12,13}],
         [index-12,father-1,prec-{},inc-{13}],
         [index-13,father-5,prec-{},inc-{14}],
```

```
         [index-14,father-6,prec-{},inc-{}]])).

ctr_typical(
    stable_compatibility,
    [size('NODES')>2,range('NODES'^father)>1]).

ctr_exchangeable(stable_compatibility,[items('NODES',all)]).

ctr_graph(
    stable_compatibility,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^father=nodes2^index],
    ['MAX_NSCC'=<1,
     'NCC'=1,
     'MAX_ID'=<2,
     'PATH_FROM_TO'(index,index,prec)=1,
     'PATH_FROM_TO'(index,index,inc)=0,
     'PATH_FROM_TO'(index,inc,index)=0],
    []).

ctr_application(stable_compatibility,[1]).
```

B.381 stage_element

◇ **META-DATA:**

```
ctr_date(stage_element,['20040828','20060816']).

ctr_origin(
    stage_element,
    \index{Choco|indexuse}Choco, derived from %c.,
    [element]).

ctr_usual_name(stage_element,stage_elt).

ctr_synonyms(stage_element,[stage_elem]).

ctr_arguments(
    stage_element,
    ['ITEM'-collection(index-dvar,value-dvar),
     'TABLE'-collection(low-int,up-int,value-int)]).

ctr_restrictions(
    stage_element,
    [required('ITEM',[index,value]),
     size('ITEM')=1,
     size('TABLE')>0,
     required('TABLE',[low,up,value]),
     'TABLE'^low=<'TABLE'^up,
     increasing_seq('TABLE',[low])]).

ctr_example(
    stage_element,
    stage_element(
        [[index-5,value-6]],
        [[low-3,up-7,value-6],
         [low-8,up-8,value-8],
         [low-9,up-14,value-2],
         [low-15,up-19,value-9]])).

ctr_typical(
    stage_element,
    [size('TABLE')>1,
     range('TABLE'^value)>1,
     'TABLE'^low<'TABLE'^up]).

ctr_exchangeable(
    stage_element,
```

```
    [vals(['ITEM'^value,'TABLE'^value],int,=\=,all,dontcare)]).

ctr_graph(
    stage_element,
    ['TABLE'],
    2,
    ['PATH'>>collection(table1,table2)],
    [table1^low=<table1^up,
     table1^up+1=table2^low,
     table2^low=<table2^up],
    ['NARC'=size('TABLE')-1],
    []).

ctr_graph(
    stage_element,
    ['ITEM','TABLE'],
    2,
    ['PRODUCT'>>collection(item,table)],
    [item^index>=table^low,
     item^index=<table^up,
     item^value=table^value],
    ['NARC'=1],
    []).

ctr_eval(stage_element,[automaton(stage_element_a)]).

ctr_pure_functional_dependency(stage_element,[]).

ctr_functional_dependency(stage_element,1-2,[1-1,2]).

ctr_extensible(stage_element,[],'TABLE',suffix).

stage_element_a(FLAG,ITEM,TABLE) :-
        collection(ITEM,[dvar,dvar]),
        collection(TABLE,[int,int,int]),
        length(TABLE,N),
        N>0,
        get_attr1(TABLE,LOWS),
        get_attr2(TABLE,UPS),
        check_lesseq(LOWS,UPS),
        collection_increasing_seq(TABLE,[1]),
        ITEM=[[index-ITEM_INDEX,value-ITEM_VALUE]],
        stage_element_signature(
            TABLE,
            SIGNATURE,
            ITEM_INDEX,
```

```
            ITEM_VALUE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _49090,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

stage_element_signature([],[],_46094,_46095).

stage_element_signature(
    [[low-TABLE_LOW,up-TABLE_UP,value-TABLE_VALUE]|TABLEs],
    [S|Ss],
    ITEM_INDEX,
    ITEM_VALUE) :-
        TABLE_LOW#=<ITEM_INDEX#/\ITEM_INDEX#=<TABLE_UP#/\
        ITEM_VALUE#=TABLE_VALUE#<=>
        S,
        stage_element_signature(
            TABLEs,
            Ss,
            ITEM_INDEX,
            ITEM_VALUE).
```

B.382 stretch_circuit

◊ META-DATA:

```
ctr_date(stretch_circuit,['20030820','20060817','20090716']).

ctr_origin(stretch_circuit,'\\cite{Pesant01}',[]).

ctr_usual_name(stretch_circuit,stretch).

ctr_arguments(
    stretch_circuit,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,lmin-int,lmax-int)]).

ctr_restrictions(
    stretch_circuit,
    [size('VARIABLES')>0,
     required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,lmin,lmax]),
     distinct('VALUES',val),
     'VALUES'^lmin=<'VALUES'^lmax,
     'VALUES'^lmin=<size('VARIABLES'),
     sum('VALUES'^lmin)=<size('VARIABLES')]).

ctr_example(
    stretch_circuit,
    stretch_circuit(
        [[var-6],
         [var-6],
         [var-3],
         [var-1],
         [var-1],
         [var-1],
         [var-6],
         [var-6]],
        [[val-1,lmin-2,lmax-4],
         [val-2,lmin-2,lmax-3],
         [val-3,lmin-1,lmax-6],
         [val-6,lmin-2,lmax-4]])).

ctr_typical(
    stretch_circuit,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
```

```
     size('VARIABLES')>size('VALUES'),
     size('VALUES')>1,
     'VALUES'^lmax=<size('VARIABLES')]).

ctr_exchangeable(
    stretch_circuit,
    [items('VARIABLES',shift),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    stretch_circuit,
    ['VARIABLES'],
    2,
    foreach(
        VALUES,
        ['CIRCUIT'>>collection(variables1,variables2),
         'LOOP'>>collection(variables1,variables2)]),
    [variables1^var='VALUES'^val,variables2^var='VALUES'^val],
    [not_in('MIN_NCC',1,'VALUES'^lmin-1),
     'MAX_NCC'=<'VALUES'^lmax],
    []).

ctr_eval(stretch_circuit,[reformulation(stretch_circuit_r)]).

stretch_circuit_r(VARIABLES,VALUES) :-
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int,int,int]),
        length(VARIABLES,N),
        stretch_circuit1(VALUES,0,N,DELTA),
        prefix_length(VARIABLES,VARS_DELTA,DELTA),
        append(VARIABLES,VARS_DELTA,VARS),
        ND is N+DELTA,
        stretch_circuit2(VALUES,N,ND,VALS),
        eval(stretch_path(VARS,VALS)).

stretch_circuit1([],C,N,DELTA) :-
        DELTA is min(C,N).

stretch_circuit1([[_52393,_52395,_52400-L]|R],C,N,DELTA) :-
        M is max(L,C),
```

```
        stretch_circuit1(R,M,N,DELTA).

stretch_circuit2([],_52385,_52386,[]).

stretch_circuit2([[A,B,lmax-L]|R],N,ND,[[A,B,lmax-LL]|S]) :-
        (   L>=N ->
            LL=ND
        ;   LL=L
        ),
        stretch_circuit2(R,N,ND,S).
```

B.383 stretch_path

◇ **META-DATA:**

```
ctr_date(stretch_path,['20030820','20060817','20090712']).

ctr_origin(stretch_path,'\\cite{Pesant01}',[]).

ctr_usual_name(stretch_path,stretch).

ctr_arguments(
    stretch_path,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,lmin-int,lmax-int)]).

ctr_restrictions(
    stretch_path,
    [size('VARIABLES')>0,
     required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,lmin,lmax]),
     distinct('VALUES',val),
     'VALUES'^lmin>=0,
     'VALUES'^lmin=<'VALUES'^lmax,
     'VALUES'^lmin=<size('VARIABLES')]).

ctr_example(
    stretch_path,
    stretch_path(
        [[var-6],
         [var-6],
         [var-3],
         [var-1],
         [var-1],
         [var-1],
         [var-6],
         [var-6]],
        [[val-1,lmin-2,lmax-4],
         [val-2,lmin-2,lmax-3],
         [val-3,lmin-1,lmax-6],
         [val-6,lmin-2,lmax-2]])).

ctr_typical(
    stretch_path,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
```

```
     size('VARIABLES')>size('VALUES'),
     size('VALUES')>1,
     sum('VALUES'^lmin)=<size('VARIABLES'),
     'VALUES'^lmax=<size('VARIABLES')]).

ctr_exchangeable(
    stretch_path,
    [items('VARIABLES',reverse),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    stretch_path,
    ['VARIABLES'],
    2,
    foreach(
        VALUES,
        ['PATH'>>collection(variables1,variables2),
         'LOOP'>>collection(variables1,variables2)]),
    [variables1^var='VALUES'^val,variables2^var='VALUES'^val],
    [not_in('MIN_NCC',1,'VALUES'^lmin-1),
     'MAX_NCC'=<'VALUES'^lmax],
    []).

ctr_eval(stretch_path,[automaton(stretch_path_a)]).

stretch_path_a(FLAG,VARIABLES,VALUES) :-
        stretch_path_get_a(VARIABLES,VALUES,AUTOMATON,ALPHABET),
        automaton_bool(FLAG,ALPHABET,AUTOMATON).

stretch_path_get_a(VARIABLES,VALUES,AUTOMATON,ALPHABET) :-
        length(VARIABLES,N),
        N>0,
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int,int(0,N),int]),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        get_attr2(VALUES,LMINS),
        get_attr3(VALUES,LMAXS),
        length(VALS,M),
        M>0,
```

```
all_different(VALS),
check_lesseq(LMINS,LMAXS),
stretch_lmin(LMINS,LMINS1),
stretch_reduce_lmax(LMAXS,N,LMAXSR),
stretch_gen_states(LMINS1,LMAXSR,N,1,STATES),
stretch_gen_transitions(
    1,
    M,
    LMINS1,
    LMAXSR,
    LMINS1,
    LMAXSR,
    N,
    TRANSITIONS),
get_minimum(VARS,MINVARS),
get_maximum(VARS,MAXVARS),
sort(VALS,SVALS),
SVALS=[MINVALS|_57481],
last(SVALS,MAXVALS),
VALS_RANGE is MAXVALS-MINVALS+1,
(   VALS_RANGE=M,
    MINVALS=<MINVARS,
    MAXVARS=<MAXVALS ->
    stretch_path_simplify_transitions(
        TRANSITIONS,
        MINVALS,
        SIMPLIFIED_TRANSITIONS),
    AUTOMATON=
    automaton(
        VARS,
        _63596,
        VARS,
        STATES,
        SIMPLIFIED_TRANSITIONS,
        [],
        [],
        []),
    SIG in MINVALS..MAXVALS
;   stretch_path_signature(VARS,VALS,M,SIGNATURE),
    AUTOMATON=
    automaton(
        SIGNATURE,
        _65060,
        SIGNATURE,
        STATES,
        TRANSITIONS,
```

```
                [],
                [],
                []),
            SIG in 0..M
        ),
        union_dom_list_int([SIG],ALPHABET).

stretch_path_simplify_transitions([],_57303,[]) :-
        !.

stretch_path_simplify_transitions(
    [arc(_57308,0,_57310)|R],
    MINVALS,
    S) :-
        !,
        stretch_path_simplify_transitions(R,MINVALS,S).

stretch_path_simplify_transitions(
    [arc(Si,E,Sj)|R],
    MINVALS,
    [arc(Si,NE,Sj)|S]) :-
        NE is MINVALS+E-1,
        stretch_path_simplify_transitions(R,MINVALS,S).

stretch_path_signature([],_57300,_57301,[]).

stretch_path_signature([VAR|VARs],VALS,M,[S|Ss]) :-
        S in 0..M,
        stretch_path_signature1(VALS,VALS,VAR,1,S),
        stretch_path_signature(VARs,VALS,M,Ss).

stretch_path_signature1([],VALS,VAR,_57305,S) :-
        stretch_path_signature2(VALS,VAR,DIFF),
        call(DIFF#<=>S#=0).

stretch_path_signature1([VAL|VALs],VALS,VAR,I,S) :-
        VAR#=VAL#<=>S#=I,
        I1 is I+1,
        stretch_path_signature1(VALs,VALS,VAR,I1,S).

stretch_path_signature2([],_57300,1).

stretch_path_signature2([VAL|VALs],VAR,VAR#\=VAL#/\R) :-
        stretch_path_signature2(VALs,VAR,R).
```

B.384 stretch_path_partition

◇ **META-DATA:**

```
ctr_date(stretch_path_partition,['20091106']).

ctr_origin(
    stretch_path_partition,
    Derived from %c.,
    [stretch_path]).

ctr_synonyms(stretch_path_partition,[stretch]).

ctr_types(
    stretch_path_partition,
    ['VALUES'-collection(val-int)]).

ctr_arguments(
    stretch_path_partition,
    ['VARIABLES'-collection(var-dvar),
     'PARTLIMITS'-collection(p-'VALUES',lmin-int,lmax-int)]).

ctr_restrictions(
    stretch_path_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     size('VARIABLES')>0,
     required('VARIABLES',var),
     size('PARTLIMITS')>0,
     required('PARTLIMITS',[p,lmin,lmax]),
     'PARTLIMITS'^lmin>=0,
     'PARTLIMITS'^lmin=<'PARTLIMITS'^lmax,
     'PARTLIMITS'^lmin=<size('VARIABLES')]).

ctr_example(
    stretch_path_partition,
    stretch_path_partition(
        [[var-1],
         [var-2],
         [var-0],
         [var-0],
         [var-2],
         [var-2],
         [var-2],
         [var-0]],
```

```
        [[p-[[val-1],[val-2]],lmin-2,lmax-4],
         [p-[[val-3]],lmin-0,lmax-2]])).

ctr_typical(
    stretch_path_partition,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VARIABLES')>size('PARTLIMITS'),
     size('PARTLIMITS')>1,
     sum('PARTLIMITS'^lmin)=<size('VARIABLES'),
     'PARTLIMITS'^lmax=<size('VARIABLES')]).

ctr_exchangeable(
    stretch_path_partition,
    [items('VARIABLES',reverse),
     items('PARTLIMITS',all),
     items('PARTLIMITS'^p,all),
     vals(
         ['VARIABLES'^var,'PARLIMITS'^p^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_eval(
    stretch_path_partition,
    [reformulation(stretch_path_partition_r),
     automaton(stretch_path_partition_a)]).

stretch_path_partition_r(VARIABLES,PARTLIMITS) :-
        length(VARIABLES,N),
        N>0,
        collection(VARIABLES,[dvar]),
        collection(
            PARTLIMITS,
            [col_len_gteq(1,[int]),int(0,N),int]),
        get_attr1(VARIABLES,VARS),
        get_col_attr1(PARTLIMITS,1,PVALS),
        get_attr2(PARTLIMITS,LMINS),
        get_attr3(PARTLIMITS,LMAXS),
        length(PVALS,M),
        M>0,
        check_lesseq(LMINS,LMAXS),
        flattern(PVALS,VALS),
        all_different(VALS),
        get_partition_var(VARS,PVALS,PVARS,M),
```

```
        gen_collection(PVARS,var,PVARIABLES),
        stretch_path_partition_values(PARTLIMITS,1,VALUES),
        eval(stretch_path(PVARIABLES,VALUES)).

stretch_path_partition_values([],_31059,[]) :-
        !.

stretch_path_partition_values(
    [[_31063,lmin-LMIN,lmax-LMAX]|R],
    V,
    [[val-V,lmin-LMIN,lmax-LMAX]|S]) :-
        V1 is V+1,
        stretch_path_partition_values(R,V1,S).

stretch_path_partition_a(FLAG,VARIABLES,PARTLIMITS) :-
        stretch_path_partition_get_a(
            VARIABLES,
            PARTLIMITS,
            AUTOMATON,
            ALPHABET),
        automaton_bool(FLAG,ALPHABET,AUTOMATON).

stretch_path_partition_get_a(
    VARIABLES,
    PARTLIMITS,
    AUTOMATON,
    ALPHABET) :-
        length(VARIABLES,N),
        N>0,
        collection(VARIABLES,[dvar]),
        collection(
            PARTLIMITS,
            [col_len_gteq(1,[int]),int(0,N),int]),
        get_attr1(VARIABLES,VARS),
        get_col_attr1(PARTLIMITS,1,PVALS),
        get_attr2(PARTLIMITS,LMINS),
        get_attr3(PARTLIMITS,LMAXS),
        length(PVALS,M),
        M>0,
        check_lesseq(LMINS,LMAXS),
        flattern(PVALS,VALS),
        all_different(VALS),
        stretch_lmin(LMINS,LMINS1),
        stretch_reduce_lmax(LMAXS,N,LMAXSR),
        stretch_gen_states(LMINS1,LMAXSR,N,1,STATES),
        stretch_gen_transitions(
```

```
            1,
            M,
            LMINS1,
            LMAXSR,
            LMINS1,
            LMAXSR,
            N,
            TRANSITIONS),
        get_minimum(VARS,MINVARS),
        get_maximum(VARS,MAXVARS),
        sort(VALS,SVALS),
        SVALS=[MINVALS|_31251],
        last(SVALS,MAXVALS),
        VALS_RANGE is MAXVALS-MINVALS+1,
        (   VALS_RANGE=M,
            MINVALS=<MINVARS,
            MAXVARS=<MAXVALS ->
            COMP_VALS=[]
        ;   stretch_path_partition_complement(
                MINVARS,
                MAXVARS,
                VALS,
                COMP_VALS)
        ),
        stretch_path_partition_expand_transitions(
            TRANSITIONS,
            COMP_VALS,
            PVALS,
            EXPANDED_TRANSITIONS),
        AUTOMATON=
        automaton(
            VARS,
            _38846,
            VARS,
            STATES,
            EXPANDED_TRANSITIONS,
            [],
            [],
            []),
        append(VARS,SVALS,ALL_VALS),
        union_dom_list_int(ALL_VALS,ALPHABET).

stretch_path_partition_complement(MIN,MAX,_31060,[]) :-
        MIN>MAX,
        !.
```

```
stretch_path_partition_complement(MIN,MAX,VALS,C) :-
        member(MIN,VALS),
        !,
        MIN1 is MIN+1,
        stretch_path_partition_complement(MIN1,MAX,VALS,C).

stretch_path_partition_complement(MIN,MAX,VALS,[MIN|C]) :-
        MIN1 is MIN+1,
        stretch_path_partition_complement(MIN1,MAX,VALS,C).

stretch_path_partition_expand_transitions([],_31059,_31060,[]) :-
        !.

stretch_path_partition_expand_transitions(
    [arc(_31065,0,_31067)|R],
    [],
    PVALS,
    S) :-
        !,
        stretch_path_partition_expand_transitions(R,[],PVALS,S).

stretch_path_partition_expand_transitions(
    [arc(Si,0,Sj)|R],
    [CV|CR],
    PVALS,
    TS) :-
        !,
        stretch_path_partition_tr([CV|CR],arc(Si,0,Sj),T),
        stretch_path_partition_expand_transitions(
            R,
            [CV|CR],
            PVALS,
            S),
        append(T,S,TS).

stretch_path_partition_expand_transitions(
    [arc(Si,E,Sj)|R],
    CL,
    PVALS,
    TS) :-
        nth1(E,PVALS,VALS),
        stretch_path_partition_tr(VALS,arc(Si,E,Sj),T),
        stretch_path_partition_expand_transitions(R,CL,PVALS,S),
        append(T,S,TS).

stretch_path_partition_tr([],_31056,[]).
```

```
stretch_path_partition_tr(
    [VAL|R],
    arc(Si,E,Sj),
    [arc(Si,VAL,Sj)|S]) :-
        stretch_path_partition_tr(R,arc(Si,E,Sj),S).
```

B.385 strict_lex2

◇ **META-DATA:**

```
ctr_predefined(strict_lex2).

ctr_date(strict_lex2,['20031016','20060817']).

ctr_origin(
    strict_lex2,
    \cite{FlenerFrischHnichKiziltanMiguelPearsonWalsh02},
    []).

ctr_types(strict_lex2,['VECTOR'-collection(var-dvar)]).

ctr_arguments(strict_lex2,['MATRIX'-collection(vec-'VECTOR')]).

ctr_restrictions(
    strict_lex2,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     required('MATRIX',vec),
     same_size('MATRIX',vec)]).

ctr_example(
    strict_lex2,
    strict_lex2(
        [[vec-[[var-2],[var-2],[var-3]]],
         [vec-[[var-2],[var-3],[var-1]]]])).

ctr_typical(strict_lex2,[size('VECTOR')>1,size('MATRIX')>1]).

ctr_exchangeable(strict_lex2,[translate(['MATRIX'^vec^var])]).

ctr_eval(
    strict_lex2,
    [checker(strict_lex2_c),reformulation(strict_lex2_r)]).

strict_lex2_c(MATRIX) :-
        collection(MATRIX,[col([int])]),
        same_size(MATRIX),
        get_attr11(MATRIX,MAT),
        lex_chain_less_c1(MAT),
        transpose(MAT,TMAT),
        lex_chain_less_c1(TMAT).
```

```
strict_lex2_r(MATRIX) :-
        collection(MATRIX,[col([dvar])]),
        same_size(MATRIX),
        get_attr11(MATRIX,MAT),
        lex_chain(MAT,[op(#<)]),
        transpose(MAT,TMAT),
        lex_chain(TMAT,[op(#<)]).
```

B.386 strictly_decreasing

◇ **META-DATA:**

```
ctr_date(strictly_decreasing,['20040814','20060817']).

ctr_origin(
    strictly_decreasing,
    Derived from %c.,
    [strictly_increasing]).

ctr_arguments(
    strictly_decreasing,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    strictly_decreasing,
    [required('VARIABLES',var)]).

ctr_example(
    strictly_decreasing,
    strictly_decreasing([[var-8],[var-4],[var-3],[var-1]])).

ctr_typical(strictly_decreasing,[size('VARIABLES')>2]).

ctr_typical_model(
    strictly_decreasing,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    strictly_decreasing,
    [translate(['VARIABLES'^var])]).

ctr_graph(
    strictly_decreasing,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var>variables2^var],
    ['NARC'=size('VARIABLES')-1],
    []).

ctr_eval(
    strictly_decreasing,
    [checker(strictly_decreasing_c),
     automaton(strictly_decreasing_a)]).
```

```
ctr_contractible(strictly_decreasing,[],'VARIABLES',any).

ctr_sol(strictly_decreasing,2,0,2,3,-).

ctr_sol(strictly_decreasing,3,0,3,4,-).

ctr_sol(strictly_decreasing,4,0,4,5,-).

ctr_sol(strictly_decreasing,5,0,5,6,-).

ctr_sol(strictly_decreasing,6,0,6,7,-).

ctr_sol(strictly_decreasing,7,0,7,8,-).

ctr_sol(strictly_decreasing,8,0,8,9,-).

ctr_sol(strictly_decreasing,9,0,9,10,-).

ctr_sol(strictly_decreasing,10,0,10,11,-).

strictly_decreasing_c([[var-X],[var-Y]|_45777]) :-
        X=<Y,
        !,
        fail.

strictly_decreasing_c([]) :-
        !.

strictly_decreasing_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        strictly_decreasing_c1(VARS).

strictly_decreasing_c1([X,Y|R]) :-
        !,
        X>Y,
        strictly_decreasing_c1([Y|R]).

strictly_decreasing_c1([_45767]) :-
        !.

strictly_decreasing_c1([]).

strictly_decreasing_a(1,[]) :-
        !.
```

```
strictly_decreasing_a(0,[]) :-
        !,
        fail.

strictly_decreasing_a(FLAG,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        strictly_decreasing_signature(VARIABLES,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _46910,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

strictly_decreasing_signature([_45768],[]) :-
        !.

strictly_decreasing_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss]) :-
        S in 0..1,
        VAR1#=<VAR2#<=>S,
        strictly_decreasing_signature([[var-VAR2]|VARs],Ss).
```

B.387 strictly_increasing

◊ META-DATA:

```
ctr_date(strictly_increasing,['20040814','20060817']).

ctr_origin(strictly_increasing,'KOALOG',[]).

ctr_arguments(
    strictly_increasing,
    ['VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    strictly_increasing,
    [required('VARIABLES',var)]).

ctr_example(
    strictly_increasing,
    strictly_increasing([[var-1],[var-3],[var-6],[var-8]])).

ctr_typical(strictly_increasing,[size('VARIABLES')>2]).

ctr_typical_model(
    strictly_increasing,
    [nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    strictly_increasing,
    [translate(['VARIABLES'^var])]).

ctr_graph(
    strictly_increasing,
    ['VARIABLES'],
    2,
    ['PATH'>>collection(variables1,variables2)],
    [variables1^var<variables2^var],
    ['NARC'=size('VARIABLES')-1],
    []).

ctr_eval(
    strictly_increasing,
    [checker(strictly_increasing_c),
     automaton(strictly_increasing_a)]).

ctr_contractible(strictly_increasing,[],'VARIABLES',any).
```

```
ctr_sol(strictly_increasing,2,0,2,3,-).

ctr_sol(strictly_increasing,3,0,3,4,-).

ctr_sol(strictly_increasing,4,0,4,5,-).

ctr_sol(strictly_increasing,5,0,5,6,-).

ctr_sol(strictly_increasing,6,0,6,7,-).

ctr_sol(strictly_increasing,7,0,7,8,-).

ctr_sol(strictly_increasing,8,0,8,9,-).

ctr_sol(strictly_increasing,9,0,9,10,-).

ctr_sol(strictly_increasing,10,0,10,11,-).

strictly_increasing_c([[var-X],[var-Y]|_47069]) :-
        X>=Y,
        !,
        fail.

strictly_increasing_c([]) :-
        !.

strictly_increasing_c(VARIABLES) :-
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        strictly_increasing_c1(VARS).

strictly_increasing_c1([X,Y|R]) :-
        !,
        X<Y,
        strictly_increasing_c1([Y|R]).

strictly_increasing_c1([_47059]) :-
        !.

strictly_increasing_c1([]).

strictly_increasing_a(1,[]) :-
        !.

strictly_increasing_a(0,[]) :-
        !,
```

```
        fail.

strictly_increasing_a(FLAG,VARIABLES) :-
        collection(VARIABLES,[dvar]),
        strictly_increasing_signature(VARIABLES,SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _48202,
            SIGNATURE,
            [source(s),sink(s)],
            [arc(s,0,s)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).

strictly_increasing_signature([_47060],[]) :-
        !.

strictly_increasing_signature(
    [[var-VAR1],[var-VAR2]|VARs],
    [S|Ss]) :-
        S in 0..1,
        VAR1#>=VAR2#<=>S,
        strictly_increasing_signature([[var-VAR2]|VARs],Ss).
```

B.388 strongly_connected

◊ **META-DATA:**

```
ctr_date(strongly_connected,['20030820','20040726','20060817']).

ctr_origin(
    strongly_connected,
    \cite{AlthausBockmayrElfKasperJungerMehlhorn02},
    []).

ctr_arguments(
    strongly_connected,
    ['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    strongly_connected,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index)]).

ctr_example(
    strongly_connected,
    strongly_connected(
        [[index-1,succ-{2}],
         [index-2,succ-{3}],
         [index-3,succ-{2,5}],
         [index-4,succ-{1}],
         [index-5,succ-{4}]])).

ctr_typical(strongly_connected,[size('NODES')>2]).

ctr_exchangeable(strongly_connected,[items('NODES',all)]).

ctr_graph(
    strongly_connected,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ],
    ['MIN_NSCC'=size('NODES')],
    []).

ctr_application(strongly_connected,[1]).
```

B.389 subgraph_isomorphism

◇ META-DATA:

```
ctr_predefined(subgraph_isomorphism).

ctr_date(subgraph_isomorphism,['20090821']).

ctr_origin(subgraph_isomorphism,'\\cite{Gregor79}',[]).

ctr_arguments(
    subgraph_isomorphism,
    ['NODES_PATTERN'-collection(index-int,succ-sint),
     'NODES_TARGET'-collection(index-int,succ-svar),
     'FUNCTION'-collection(image-dvar)]).

ctr_restrictions(
    subgraph_isomorphism,
    [required('NODES_PATTERN',[index,succ]),
     'NODES_PATTERN'^index>=1,
     'NODES_PATTERN'^index=<size('NODES_PATTERN'),
     distinct('NODES_PATTERN',index),
     'NODES_PATTERN'^succ>=1,
     'NODES_PATTERN'^succ=<size('NODES_PATTERN'),
     required('NODES_TARGET',[index,succ]),
     'NODES_TARGET'^index>=1,
     'NODES_TARGET'^index=<size('NODES_TARGET'),
     distinct('NODES_TARGET',index),
     'NODES_TARGET'^succ>=1,
     'NODES_TARGET'^succ=<size('NODES_TARGET'),
     required('FUNCTION',[image]),
     'FUNCTION'^image>=1,
     'FUNCTION'^image=<size('NODES_TARGET'),
     distinct('FUNCTION',image),
     size('FUNCTION')=size('NODES_PATTERN')]).

ctr_example(
    subgraph_isomorphism,
    subgraph_isomorphism(
        [[index-1,succ-{2,4}],
         [index-2,succ-{1,3,4}],
         [index-3,succ-{}],
         [index-4,succ-{}]],
        [[index-1,succ-{}],
         [index-2,succ-{3,4,5}],
         [index-3,succ-{}],
```

```
         [index-4,succ-{2,5}],
         [index-5,succ-{}]],
        [[image-4],[image-2],[image-3],[image-5]])).

ctr_typical(
    subgraph_isomorphism,
    [size('NODES_PATTERN')>1,size('NODES_TARGET')>1]).

ctr_exchangeable(
    subgraph_isomorphism,
    [items('NODES_PATTERN',all),items('NODES_TARGET',all)]).
```

B.390 sum

◇ META-DATA:

```
ctr_date(sum,['20030820','20040726','20060817']).

ctr_origin(sum,'\\cite{Yunes02}.',[]).

ctr_synonyms(sum,[sum_pred]).

ctr_arguments(
    sum,
    ['INDEX'-dvar,
     'SETS'-collection(ind-int,set-sint),
     'CONSTANTS'-collection(cst-int),
     'S'-dvar]).

ctr_restrictions(
    sum,
    [size('SETS')>=1,
     required('SETS',[ind,set]),
     distinct('SETS',ind),
     size('CONSTANTS')>=1,
     required('CONSTANTS',cst)]).

ctr_example(
    sum,
    sum(8,
        [[ind-8,set-{2,3}],
         [ind-1,set-{3}],
         [ind-3,set-{1,4,5}],
         [ind-6,set-{2,4}]],
        [[cst-4],[cst-9],[cst-1],[cst-3],[cst-1]],
        10)).

ctr_typical(
    sum,
    [size('SETS')>1,
     size('CONSTANTS')>size('SETS'),
     range('CONSTANTS'^cst)>1]).

ctr_exchangeable(sum,[items('SETS',all)]).

ctr_graph(
    sum,
    ['SETS','CONSTANTS'],
```

```
    2,
    ['PRODUCT'>>collection(sets,constants)],
    ['INDEX'=sets^ind,constants^key in_set sets^set],
    ['SUM'('CONSTANTS',cst)='S'],
    []).

ctr_functional_dependency(sum,4,[1,2,3]).
```

B.391 sum_ctr

◇ META-DATA:

```
ctr_date(sum_ctr,['20030820','20040807','20060817']).

ctr_origin(sum_ctr,'Arithmetic constraint.',[]).

ctr_synonyms(sum_ctr,[constant_sum,sum,linear,scalar_product]).

ctr_arguments(
    sum_ctr,
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'VAR'-dvar]).

ctr_restrictions(
    sum_ctr,
    [required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(sum_ctr,sum_ctr([[var-1],[var-1],[var-4]],=,6)).

ctr_typical(
    sum_ctr,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(sum_ctr,[items('VARIABLES',all)]).

ctr_graph(
    sum_ctr,
    ['VARIABLES'],
    1,
    ['SELF'>>collection(variables)],
    ['TRUE'],
    ['CTR'('SUM'('VARIABLES',var),'VAR')],
    []).

ctr_eval(sum_ctr,[checker(sum_ctr_c),reformulation(sum_ctr_r)]).

ctr_pure_functional_dependency(sum_ctr,[in_list('CTR',[=])]).

ctr_contractible(
    sum_ctr,
    [in_list('CTR',[<,=<]),minval('VARIABLES'^var)>=0],
    VARIABLES,
```

```
    any).

ctr_contractible(
    sum_ctr,
    [in_list('CTR',[>=,>]),maxval('VARIABLES'^var)=<0],
    VARIABLES,
    any).

ctr_extensible(
    sum_ctr,
    [in_list('CTR',[>=,>]),minval('VARIABLES'^var)>=0],
    VARIABLES,
    any).

ctr_extensible(
    sum_ctr,
    [in_list('CTR',[<,=<]),maxval('VARIABLES'^var)=<0],
    VARIABLES,
    any).

ctr_aggregate(sum_ctr,[],[union,id,+]).

ctr_cond_imply(
    sum_ctr,
    sum_squares_ctr,
    ['VARIABLES'^var>=0,'VARIABLES'^var=<1],
    ['VARIABLES'^var>=0,'VARIABLES'^var=<1],
    id).

ctr_cond_imply(
    sum_ctr,
    sum_cubes_ctr,
    ['VARIABLES'^var>= -1,'VARIABLES'^var=<1],
    ['VARIABLES'^var>= -1,'VARIABLES'^var=<1],
    id).

ctr_cond_imply(
    sum_ctr,
    sum_powers5_ctr,
    ['VARIABLES'^var>= -1,'VARIABLES'^var=<1],
    ['VARIABLES'^var>= -1,'VARIABLES'^var=<1],
    id).

ctr_cond_imply(
    sum_ctr,
    increasing_sum,
```

```
    [in_list('CTR',[=]),increasing('VARIABLES')],
    [],
    ['VARIABLES','VAR']).

sum_ctr_r(VARIABLES,CTR,VAR) :-
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_var(VARS,SUM),
        call_term_relop_value(SUM,CTR,VAR).

sum_ctr_c(VARIABLES,=,VAR) :-
        !,
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        sumlist(VARS,VAR).

sum_ctr_c(VARIABLES,=\=,VAR) :-
        !,
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        sumlist(VARS,SUM),
        SUM=\=VAR.

sum_ctr_c(VARIABLES,<,VAR) :-
        !,
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        sumlist(VARS,SUM),
        SUM<VAR.

sum_ctr_c(VARIABLES,>=,VAR) :-
        !,
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        sumlist(VARS,SUM),
        SUM>=VAR.

sum_ctr_c(VARIABLES,>,VAR) :-
        !,
        collection(VARIABLES,[int]),
```

```
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        sumlist(VARS,SUM),
        SUM>VAR.

sum_ctr_c(VARIABLES,=<,VAR) :-
        collection(VARIABLES,[int]),
        check_type(int,VAR),
        get_attr1(VARIABLES,VARS),
        sumlist(VARS,SUM),
        SUM=<VAR.
```

B.392 sum_cubes_ctr

◇ META-DATA:

```
ctr_predefined(sum_cubes_ctr).

ctr_date(sum_cubes_ctr,['20111111']).

ctr_origin(sum_cubes_ctr,'Arithmetic constraint.',[]).

ctr_synonyms(
    sum_cubes_ctr,
    [sum_cubes,sum_of_cubes,sum_of_cubes_ctr]).

ctr_arguments(
    sum_cubes_ctr,
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'VAR'-dvar]).

ctr_restrictions(
    sum_cubes_ctr,
    [required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    sum_cubes_ctr,
    sum_cubes_ctr([[var-1],[var-2],[var-2]],=,17)).

ctr_typical(
    sum_cubes_ctr,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(sum_cubes_ctr,[items('VARIABLES',all)]).

ctr_eval(sum_cubes_ctr,[reformulation(sum_cubes_ctr_r)]).

ctr_pure_functional_dependency(
    sum_cubes_ctr,
    [in_list('CTR',[=])]).

ctr_contractible(
    sum_cubes_ctr,
    [in_list('CTR',[<,=<]),minval('VARIABLES'^var)>=0],
    VARIABLES,
    any).
```

```
ctr_contractible(
    sum_cubes_ctr,
    [in_list('CTR',[>=,>]),maxval('VARIABLES'^var)=<0],
    VARIABLES,
    any).

ctr_extensible(
    sum_cubes_ctr,
    [in_list('CTR',[>=,>]),minval('VARIABLES'^var)>=0],
    VARIABLES,
    any).

ctr_extensible(
    sum_cubes_ctr,
    [in_list('CTR',[<,=<]),maxval('VARIABLES'^var)=<0],
    VARIABLES,
    any).

ctr_aggregate(sum_cubes_ctr,[],[union,id,+]).

sum_cubes_ctr_r(VARIABLES,CTR,VAR) :-
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_cubes_var(VARS,SUM_CUBES),
        call_term_relop_value(SUM_CUBES,CTR,VAR).
```

B.393 sum_free

◇ META-DATA:

```
ctr_predefined(sum_free).

ctr_date(sum_free,['20061003']).

ctr_origin(sum_free,'\\cite{HoeveSabharwal07}',[]).

ctr_arguments(sum_free,['S'-svar]).

ctr_example(sum_free,sum_free({1,3,5,9})).
```

B.394 sum_of_increments

◇ **META-DATA:**

```
ctr_predefined(sum_of_increments).

ctr_date(sum_of_increments,['20111105']).

ctr_origin(sum_of_increments,'\\cite{Brand09}',[]).

ctr_synonyms(
    sum_of_increments,
    [increments_sum,incr_sum,sum_incr,sum_increments]).

ctr_arguments(
    sum_of_increments,
    ['VARIABLES'-collection(var-dvar),'LIMIT'-dvar]).

ctr_restrictions(
    sum_of_increments,
    [required('VARIABLES',var),'VARIABLES'^var>=0,'LIMIT'>=0]).

ctr_example(
    sum_of_increments,
    [sum_of_increments(
         [[var-4],[var-4],[var-3],[var-4],[var-6]],
         7)]).

ctr_typical(
    sum_of_increments,
    [size('VARIABLES')>2,
     range('VARIABLES'^var)>1,
     maxval('VARIABLES'^var)>0,
     'LIMIT'>0,
     'LIMIT'=<size('VARIABLES')*range('VARIABLES'^var)/2]).

ctr_exchangeable(
    sum_of_increments,
    [translate(['VARIABLES'^var,'LIMIT']),
     items('VARIABLES',reverse),
     vals(['LIMIT'],int,<,dontcare,dontcare)]).

ctr_eval(
    sum_of_increments,
    [reformulation(sum_of_increments_r)]).
```

```
ctr_contractible(sum_of_increments,[],'VARIABLES',prefix).

ctr_contractible(sum_of_increments,[],'VARIABLES',suffix).

ctr_sol(sum_of_increments,2,0,2,14,[0-1,1-4,2-9]).

ctr_sol(sum_of_increments,3,0,3,145,[0-1,1-7,2-23,3-54,4-60]).

ctr_sol(
    sum_of_increments,
    4,
    0,
    4,
    2875,
    [0-1,1-11,2-51,3-156,4-375,5-485,6-563,7-608,8-625]).

ctr_sol(
    sum_of_increments,
    5,
    0,
    5,
    51415,
    [0-1,
     1-16,
     2-101,
     3-396,
     4-1167,
     5-2848,
     6-4263,
     7-5568,
     8-6616,
     9-7314,
     10-7650,
     11-7720,
     12-7755]).

ctr_sol(
    sum_of_increments,
    6,
    0,
    6,
    1210104,
    [0-1,
     1-22,
     2-183,
     3-904,
```

```
      4-3235,
      5-9318,
      6-22981,
      7-38836,
      8-56703,
      9-74658,
      10-90639,
      11-102875,
      12-110425,
      13-113827,
      14-115857,
      15-116942,
      16-117437,
      17-117612,
      18-117649]).

ctr_sol(
     sum_of_increments,
     7,
     0,
     7,
     28573741,
     [0-1,
      1-29,
      2-309,
      3-1891,
      4-8135,
      5-27483,
      6-77947,
      7-193742,
      8-359880,
      9-578511,
      10-837441,
      11-1115687,
      12-1386029,
      13-1619993,
      14-1795694,
      15-1908968,
      16-1988222,
      17-2039616,
      18-2069933,
      19-2085763,
      20-2092817,
      21-2095436,
      22-2096360,
      23-2096822,
```

```
     24-2097032]).

ctr_sol(
    sum_of_increments,
    8,
    0,
    8,
    801944469,
    [0-1,
     1-37,
     2-493,
     3-3679,
     4-18835,
     5-74143,
     6-240751,
     7-675244,
     8-1688427,
     9-3369015,
     10-5865915,
     11-9220695,
     12-13354545,
     13-18051195,
     14-22965651,
     15-27670800,
     16-31755573,
     17-34989993,
     18-37574073,
     19-39526569,
     20-40912205,
     21-41827847,
     22-42386387,
     23-42700112,
     24-42865683,
     25-42953199,
     26-43002171,
     27-43027581,
     28-43039551,
     29-43044507,
     30-43046215,
     31-43046656,
     32-43046721]).

sum_of_increments_r([],_66971) :-
        !.

sum_of_increments_r(VARIABLES,LIMIT) :-
```

```
        collection(VARIABLES,[dvar_gteq(0)]),
        check_type(dvar_gteq(0),LIMIT),
        get_attr1(VARIABLES,VARS),
        fd_max(LIMIT,MaxL),
        sum_of_increments_r1([0|VARS],MaxL,SUM),
        call(SUM#=<LIMIT).

sum_of_increments_r1([_66973],_66971,0) :-
        !.

sum_of_increments_r1([V1,V2|R],MaxL,S2+S) :-
        S2 in 0..MaxL,
        V2-V1#=<S2,
        sum_of_increments_r1([V2|R],MaxL,S).
```

B.395 sum_of_weights_of_distinct_values

◇ META-DATA:

```
ctr_date(
    sum_of_weights_of_distinct_values,
    ['20030820','20040726','20060817']).

ctr_origin(
    sum_of_weights_of_distinct_values,
    \cite{BeldiceanuCarlssonThiel02},
    []).

ctr_synonyms(sum_of_weights_of_distinct_values,[swdv]).

ctr_arguments(
    sum_of_weights_of_distinct_values,
    ['VARIABLES'-collection(var-dvar),
     'VALUES'-collection(val-int,weight-int),
     'COST'-dvar]).

ctr_restrictions(
    sum_of_weights_of_distinct_values,
    [required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,weight]),
     'VALUES'^weight>=0,
     distinct('VALUES',val),
     in_attr('VARIABLES',var,'VALUES',val),
     'COST'>=0]).

ctr_example(
    sum_of_weights_of_distinct_values,
    sum_of_weights_of_distinct_values(
        [[var-1],[var-6],[var-1]],
        [[val-1,weight-5],[val-2,weight-3],[val-6,weight-7]],
        12)).

ctr_typical(
    sum_of_weights_of_distinct_values,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     size('VALUES')>1,
     range('VALUES'^weight)>1,
     maxval('VALUES'^weight)>0]).
```

```
ctr_exchangeable(
    sum_of_weights_of_distinct_values,
    [items('VARIABLES',all),
     vals(['VARIABLES'^var],int,=\=,all,in),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    sum_of_weights_of_distinct_values,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=values^val],
    ['NSOURCE'=size('VARIABLES'),
     'SUM'('VALUES',weight)='COST'],
    []).

ctr_eval(
    sum_of_weights_of_distinct_values,
    [checker(sum_of_weights_of_distinct_values_c),
     reformulation(sum_of_weights_of_distinct_values_r)]).

ctr_pure_functional_dependency(
    sum_of_weights_of_distinct_values,
    []).

ctr_functional_dependency(
    sum_of_weights_of_distinct_values,
    3,
    [1,2]).

sum_of_weights_of_distinct_values_r(VARIABLES,VALUES,COST) :-
        collection(VARIABLES,[dvar]),
        collection(VALUES,[int,int_gteq(0)]),
        check_type(dvar_gteq(0),COST),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        get_attr2(VALUES,WEIGHTS),
        all_different(VALS),
        (   VARS=[] ->
            COST#=0
```

```
        ;   sum_of_weights_of_distinct_values1(VARS,VALS),
            sum_of_weights_of_distinct_values3(
                VALS,
                WEIGHTS,
                VARS,
                TERM),
            call(COST#=TERM)
        ).

sum_of_weights_of_distinct_values1([],_44987).

sum_of_weights_of_distinct_values1([VAR|RVAR],VALS) :-
        sum_of_weights_of_distinct_values2(VALS,VAR,OR_TERM),
        call(OR_TERM),
        sum_of_weights_of_distinct_values1(RVAR,VALS).

sum_of_weights_of_distinct_values2([],_44987,0).

sum_of_weights_of_distinct_values2(
    [VAL|RVAL],
    VAR,
    VAR#=VAL#\/TERM) :-
        sum_of_weights_of_distinct_values2(RVAL,VAR,TERM).

sum_of_weights_of_distinct_values3([],[],_44988,0).

sum_of_weights_of_distinct_values3(
    [VAL|RVAL],
    [WEIGHT|RWEIGHT],
    VARS,
    WEIGHT*B+TERM) :-
        sum_of_weights_of_distinct_values4(VARS,VAL,OR_TERM),
        call(B#<=>OR_TERM),
        sum_of_weights_of_distinct_values3(
            RVAL,
            RWEIGHT,
            VARS,
            TERM).

sum_of_weights_of_distinct_values4([],_44987,0).

sum_of_weights_of_distinct_values4(
    [VAR|RVAR],
    VAL,
    VAL#=VAR#\/TERM) :-
        sum_of_weights_of_distinct_values4(RVAR,VAL,TERM).
```

```
sum_of_weights_of_distinct_values_c(VARIABLES,VALUES,COST) :-
        collection(VARIABLES,[int]),
        collection(VALUES,[int,int_gteq(0)]),
        integer(COST),
        COST>=0,
        get_attr1(VARIABLES,VARS),
        samsort(VARS,SVARS),
        get_attr12(VALUES,VAL_WEIGHT),
        keysort(VAL_WEIGHT,SVAL_WEIGHT),
        sum_of_weights_of_distinct_values_inc(SVAL_WEIGHT),
        sum_of_weights_of_distinct_values_check(
            SVARS,
            SVAL_WEIGHT,
            COST).

sum_of_weights_of_distinct_values_inc([]) :-
        !.

sum_of_weights_of_distinct_values_inc([_44990]) :-
        !.

sum_of_weights_of_distinct_values_inc(
    [Val1-_44994,Val2-Weight2|R]) :-
        Val1<Val2,
        sum_of_weights_of_distinct_values_inc([Val2-Weight2|R]).

sum_of_weights_of_distinct_values_check([],_44990,0) :-
        !.

sum_of_weights_of_distinct_values_check(
    [Val|R],
    [Val-Weight|S],
    Cost) :-
        !,
        Cost1 is Cost-Weight,
        sum_of_weights_of_distinct_values_check(
            R,
            [Val-0|S],
            Cost1).

sum_of_weights_of_distinct_values_check(
    [Var|R],
    [Val-_Weight|S],
    Cost) :-
        Var>Val,
```

```
sum_of_weights_of_distinct_values_check([Var|R],S,Cost).
```

B.396 sum_powers4_ctr

◇ **META-DATA:**

```
ctr_predefined(sum_powers4_ctr).

ctr_date(sum_powers4_ctr,['20120403']).

ctr_origin(sum_powers4_ctr,'Arithmetic constraint.',[]).

ctr_synonyms(
    sum_powers4_ctr,
    [sum_powers4,sum_of_powers4,sum_of_powers4_ctr]).

ctr_arguments(
    sum_powers4_ctr,
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'VAR'-dvar]).

ctr_restrictions(
    sum_powers4_ctr,
    [required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    sum_powers4_ctr,
    sum_powers4_ctr([[var-1],[var-1],[var-2]],=,18)).

ctr_typical(
    sum_powers4_ctr,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(sum_powers4_ctr,[items('VARIABLES',all)]).

ctr_eval(sum_powers4_ctr,[reformulation(sum_powers4_ctr_r)]).

ctr_pure_functional_dependency(
    sum_powers4_ctr,
    [in_list('CTR',[=])]).

ctr_contractible(
    sum_powers4_ctr,
    [in_list('CTR',[<,=<])],
    VARIABLES,
    any).
```

```
ctr_extensible(
    sum_powers4_ctr,
    [in_list('CTR',[>=,>])],
    VARIABLES,
    any).

ctr_aggregate(sum_powers4_ctr,[],[union,id,+]).

sum_powers4_ctr_r(VARIABLES,CTR,VAR) :-
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_powers4_var(VARS,SUM_POWERS4),
        call_term_relop_value(SUM_POWERS4,CTR,VAR).
```

B.397 sum_powers5_ctr

◇ **META-DATA:**

```
ctr_predefined(sum_powers5_ctr).

ctr_date(sum_powers5_ctr,['20120403']).

ctr_origin(sum_powers5_ctr,'Arithmetic constraint.',[]).

ctr_synonyms(
    sum_powers5_ctr,
    [sum_powers5,sum_of_powers5,sum_of_powers5_ctr]).

ctr_arguments(
    sum_powers5_ctr,
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'VAR'-dvar]).

ctr_restrictions(
    sum_powers5_ctr,
    [required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    sum_powers5_ctr,
    sum_powers5_ctr([[var-1],[var-1],[var-2]],=,34)).

ctr_typical(
    sum_powers5_ctr,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(sum_powers5_ctr,[items('VARIABLES',all)]).

ctr_eval(sum_powers5_ctr,[reformulation(sum_powers5_ctr_r)]).

ctr_pure_functional_dependency(
    sum_powers5_ctr,
    [in_list('CTR',[=])]).

ctr_contractible(
    sum_powers5_ctr,
    [in_list('CTR',[<,=<]),minval('VARIABLES'^var)>=0],
    VARIABLES,
    any).
```

```
ctr_contractible(
    sum_powers5_ctr,
    [in_list('CTR',[>=,>]),maxval('VARIABLES'^var)=<0],
    VARIABLES,
    any).

ctr_extensible(
    sum_powers5_ctr,
    [in_list('CTR',[>=,>]),minval('VARIABLES'^var)>=0],
    VARIABLES,
    any).

ctr_extensible(
    sum_powers5_ctr,
    [in_list('CTR',[<,=<]),maxval('VARIABLES'^var)=<0],
    VARIABLES,
    any).

ctr_aggregate(sum_powers5_ctr,[],[union,id,+]).

sum_powers5_ctr_r(VARIABLES,CTR,VAR) :-
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_powers5_var(VARS,SUM_POWERS5),
        call_term_relop_value(SUM_POWERS5,CTR,VAR).
```

B.398 sum_powers6_ctr

◇ META-DATA:

```
ctr_predefined(sum_powers6_ctr).

ctr_date(sum_powers6_ctr,['20120403']).

ctr_origin(sum_powers6_ctr,'Arithmetic constraint.',[]).

ctr_synonyms(
    sum_powers6_ctr,
    [sum_powers6,sum_of_powers6,sum_of_powers6_ctr]).

ctr_arguments(
    sum_powers6_ctr,
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'VAR'-dvar]).

ctr_restrictions(
    sum_powers6_ctr,
    [required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    sum_powers6_ctr,
    sum_powers6_ctr([[var-1],[var-1],[var-2]],=,66)).

ctr_typical(
    sum_powers6_ctr,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(sum_powers6_ctr,[items('VARIABLES',all)]).

ctr_eval(sum_powers6_ctr,[reformulation(sum_powers6_ctr_r)]).

ctr_pure_functional_dependency(
    sum_powers6_ctr,
    [in_list('CTR',[=])]).

ctr_contractible(
    sum_powers6_ctr,
    [in_list('CTR',[<,=<])],
    VARIABLES,
    any).
```

```
ctr_extensible(
    sum_powers6_ctr,
    [in_list('CTR',[>=,>])],
    VARIABLES,
    any).

ctr_aggregate(sum_powers6_ctr,[],[union,id,+]).

sum_powers6_ctr_r(VARIABLES,CTR,VAR) :-
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_powers6_var(VARS,SUM_POWERS6),
        call_term_relop_value(SUM_POWERS6,CTR,VAR).
```

B.399 sum_set

◇ **META-DATA:**

```
ctr_date(sum_set,['20031001','20060818']).

ctr_origin(sum_set,'H.~Cambazard',[]).

ctr_arguments(
    sum_set,
    ['SV'-svar,
     'VALUES'-collection(val-int,coef-int),
     'CTR'-atom,
     'VAR'-dvar]).

ctr_restrictions(
    sum_set,
    [required('VALUES',[val,coef]),
     distinct('VALUES',val),
     'VALUES'^coef>=0,
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    sum_set,
    sum_set(
        {2,3,6},
        [[val-2,coef-7],
         [val-9,coef-1],
         [val-5,coef-7],
         [val-6,coef-2]],
        =,
        9)).

ctr_typical(
    sum_set,
    [size('VALUES')>1,
     'VALUES'^coef>0,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(sum_set,[items('VALUES',all)]).

ctr_graph(
    sum_set,
    ['VALUES'],
    1,
    ['SELF'>>collection(values)],
```

```
[values^val in_set 'SV'],
['CTR'('SUM'('VALUES',coef),'VAR')],
[]).
```

B.400 sum_squares_ctr

◇ **META-DATA:**

```
ctr_predefined(sum_squares_ctr).

ctr_date(sum_squares_ctr,['20110612']).

ctr_origin(sum_squares_ctr,'Arithmetic constraint.',[]).

ctr_synonyms(
    sum_squares_ctr,
    [sum_squares,sum_of_squares,sum_of_squares_ctr]).

ctr_arguments(
    sum_squares_ctr,
    ['VARIABLES'-collection(var-dvar),'CTR'-atom,'VAR'-dvar]).

ctr_restrictions(
    sum_squares_ctr,
    [required('VARIABLES',var),
     in_list('CTR',[=,=\=,<,>=,>,=<])]).

ctr_example(
    sum_squares_ctr,
    sum_squares_ctr([[var-1],[var-1],[var-4]],=,18)).

ctr_typical(
    sum_squares_ctr,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     in_list('CTR',[=,<,>=,>,=<])]).

ctr_exchangeable(sum_squares_ctr,[items('VARIABLES',all)]).

ctr_eval(
    sum_squares_ctr,
    [checker(sum_squares_ctr_c),
     reformulation(sum_squares_ctr_r)]).

ctr_pure_functional_dependency(
    sum_squares_ctr,
    [in_list('CTR',[=])]).

ctr_contractible(
    sum_squares_ctr,
```

```
    [in_list('CTR',[<,=<])],
    VARIABLES,
    any).

ctr_extensible(
    sum_squares_ctr,
    [in_list('CTR',[>=,>])],
    VARIABLES,
    any).

ctr_aggregate(sum_squares_ctr,[],[union,id,+]).

ctr_cond_imply(
    sum_squares_ctr,
    sum_powers4_ctr,
    ['VARIABLES'^var>= -1,'VARIABLES'^var=<1],
    ['VARIABLES'^var>= -1,'VARIABLES'^var=<1],
    id).

ctr_cond_imply(
    sum_squares_ctr,
    sum_powers6_ctr,
    ['VARIABLES'^var>= -1,'VARIABLES'^var=<1],
    ['VARIABLES'^var>= -1,'VARIABLES'^var=<1],
    id).

sum_squares_ctr_r(VARIABLES,CTR,VAR) :-
        collection(VARIABLES,[dvar]),
        memberchk(CTR,[=,=\=,<,>=,>,=<]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_squares_var(VARS,SUM_SQUARES),
        call_term_relop_value(SUM_SQUARES,CTR,VAR).

sum_squares_ctr_c(VARIABLES,=,VAR) :-
        collection(VARIABLES,[int]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_squares_int(VARS,0,VAR).

sum_squares_ctr_c(VARIABLES,=\=,VAR) :-
        collection(VARIABLES,[int]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_squares_int(VARS,0,SUM),
        SUM=\=VAR.
```

```
sum_squares_ctr_c(VARIABLES,<,VAR) :-
        collection(VARIABLES,[int]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_squares_int(VARS,0,SUM),
        SUM<VAR.

sum_squares_ctr_c(VARIABLES,>=,VAR) :-
        collection(VARIABLES,[int]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_squares_int(VARS,0,SUM),
        SUM>=VAR.

sum_squares_ctr_c(VARIABLES,>,VAR) :-
        collection(VARIABLES,[int]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_squares_int(VARS,0,SUM),
        SUM>VAR.

sum_squares_ctr_c(VARIABLES,=<,VAR) :-
        collection(VARIABLES,[int]),
        check_type(dvar,VAR),
        get_attr1(VARIABLES,VARS),
        build_sum_squares_int(VARS,0,SUM),
        SUM=<VAR.
```

B.401 symmetric

◇ META-DATA:

```
ctr_date(symmetric,['20060930']).

ctr_origin(symmetric,'\\cite{Dooms06}',[]).

ctr_arguments(
    symmetric,
    ['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    symmetric,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index)]).

ctr_example(
    symmetric,
    symmetric(
        [[index-1,succ-{1,2,3}],
         [index-2,succ-{1,3}],
         [index-3,succ-{1,2}],
         [index-4,succ-{5,6}],
         [index-5,succ-{4}],
         [index-6,succ-{4}]])).

ctr_typical(symmetric,[size('NODES')>2]).

ctr_exchangeable(symmetric,[items('NODES',all)]).

ctr_graph(
    symmetric,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ],
    [],
    ['SYMMETRIC']).
```

B.402 symmetric_alldifferent

◇ **META-DATA:**

```
ctr_date(
    symmetric_alldifferent,
    ['20000128','20030820','20060818']).

ctr_origin(symmetric_alldifferent,'\\cite{Regin99}',[]).

ctr_synonyms(
    symmetric_alldifferent,
    [symmetric_alldiff,
     symmetric_alldistinct,
     symm_alldifferent,
     symm_alldiff,
     symm_alldistinct,
     one_factor,
     two_cycle]).

ctr_arguments(
    symmetric_alldifferent,
    ['NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    symmetric_alldifferent,
    [size('NODES')mod 2=0,
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    symmetric_alldifferent,
    symmetric_alldifferent(
        [[index-1,succ-3],
         [index-2,succ-4],
         [index-3,succ-1],
         [index-4,succ-2]])).

ctr_typical(symmetric_alldifferent,[size('NODES')>=4]).

ctr_exchangeable(symmetric_alldifferent,[items('NODES',all)]).
```

```
ctr_graph(
    symmetric_alldifferent,
    ['NODES'],
    2,
    ['CLIQUE'(=\=)>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index,nodes2^succ=nodes1^index],
    ['NARC'=size('NODES')],
    []).

ctr_eval(
    symmetric_alldifferent,
    [reformulation(symmetric_alldifferent_r1),
     reformulation(symmetric_alldifferent_r2),
     checker(symmetric_alldifferent_c)]).

ctr_cond_imply(
    symmetric_alldifferent,
    balance_cycle,
    [],
    ['BALANCE'=0],
    [none,'NODES']).

ctr_cond_imply(
    symmetric_alldifferent,
    cycle,
    [],
    [2*'NCYCLE'=size('NODES')],
    [none,'NODES']).

ctr_cond_imply(
    symmetric_alldifferent,
    permutation,
    [],
    [],
    index_to_col).

ctr_sol(symmetric_alldifferent,2,0,2,1,-).

ctr_sol(symmetric_alldifferent,3,0,3,0,[]).

ctr_sol(symmetric_alldifferent,4,0,4,3,-).

ctr_sol(symmetric_alldifferent,5,0,5,0,[]).

ctr_sol(symmetric_alldifferent,6,0,6,15,-).
```

```
ctr_sol(symmetric_alldifferent,7,0,7,0,[]).

ctr_sol(symmetric_alldifferent,8,0,8,105,-).

ctr_sol(symmetric_alldifferent,9,0,9,0,[]).

ctr_sol(symmetric_alldifferent,10,0,10,945,-).

symmetric_alldifferent_r1(NODES) :-
        symmetric_alldifferent_r1a(NODES,INODES),
        eval(inverse(INODES)).

symmetric_alldifferent_r1a([],[]).

symmetric_alldifferent_r1a(
    [[index-INDEX,succ-SUCC]|R],
    [[index-INDEX,succ-SUCC,pred-SUCC]|S]) :-
        SUCC#\=INDEX,
        symmetric_alldifferent_r1a(R,S).

symmetric_alldifferent_r2([]) :-
        !.

symmetric_alldifferent_r2(NODES) :-
        length(NODES,N),
        symmetric_alldifferent0(NODES,SNODES),
        length(SNODES,N),
        collection(SNODES,[int(1,N),dvar(1,N)]),
        get_attr1(SNODES,INDEXES),
        get_attr2(SNODES,SUCCS),
        all_different(INDEXES),
        derangement1(SUCCS,INDEXES),
        symmetric_alldifferent1(SUCCS,1,SUCCS).

symmetric_alldifferent_c([]) :-
        !.

symmetric_alldifferent_c(NODES) :-
        length(NODES,N),
        symmetric_alldifferent0(NODES,SNODES),
        length(SNODES,N),
        collection(SNODES,[int(1,N),int(1,N)]),
        get_attr1(SNODES,INDEXES),
        get_attr2(SNODES,SUCCS),
        sort(INDEXES,SINDEXES),
        length(SINDEXES,N),
```

```
        sym_pairs(INDEXES,SUCCS,PAIRS),
        keysort(PAIRS,SPAIRS),
        symmetric_alldifferent_check(SPAIRS).

sym_pairs([],[],[]) :-
        !.

sym_pairs([I|R],[J|S],[I-a(J),J-b(I)|T]) :-
        sym_pairs(R,S,T).
```

B.403 symmetric_alldifferent_except_0

◇ **META-DATA:**

```
ctr_predefined(symmetric_alldifferent_except_0).

ctr_date(symmetric_alldifferent_except_0,['20120208']).

ctr_origin(
    symmetric_alldifferent_except_0,
    Derived from %c,
    [symmetric_alldifferent]).

ctr_synonyms(
    symmetric_alldifferent_except_0,
    [symmetric_alldiff_except_0,
     symmetric_alldistinct_except_0,
     symm_alldifferent_except_0,
     symm_alldiff_except_0,
     symm_alldistinct_except_0]).

ctr_arguments(
    symmetric_alldifferent_except_0,
    ['NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    symmetric_alldifferent_except_0,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=0,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    symmetric_alldifferent_except_0,
    [symmetric_alldifferent_except_0(
         [[index-1,succ-3],
          [index-2,succ-0],
          [index-3,succ-1],
          [index-4,succ-0]])]).

ctr_typical(
    symmetric_alldifferent_except_0,
    [size('NODES')>=4,
     minval('NODES'^succ)=0,
```

```
     maxval('NODES'^succ)>0]).

ctr_exchangeable(
    symmetric_alldifferent_except_0,
    [items('NODES',all)]).

ctr_eval(
    symmetric_alldifferent_except_0,
    [checker(symmetric_alldifferent_except_0_c),
     reformulation(symmetric_alldifferent_except_0_r),
     density(symmetric_alldifferent_except_0_d)]).

ctr_cond_imply(
    symmetric_alldifferent_except_0,
    alldifferent_except_0,
    [],
    [],
    index_to_col).

ctr_sol(symmetric_alldifferent_except_0,2,0,2,2,-).

ctr_sol(symmetric_alldifferent_except_0,3,0,3,4,-).

ctr_sol(symmetric_alldifferent_except_0,4,0,4,10,-).

ctr_sol(symmetric_alldifferent_except_0,5,0,5,26,-).

ctr_sol(symmetric_alldifferent_except_0,6,0,6,76,-).

ctr_sol(symmetric_alldifferent_except_0,7,0,7,232,-).

ctr_sol(symmetric_alldifferent_except_0,8,0,8,764,-).

symmetric_alldifferent_except_0_r([]) :-
        !.

symmetric_alldifferent_except_0_r(NODES) :-
        symmetric_alldifferent0(NODES,SNODES),
        length(SNODES,N),
        collection(SNODES,[int(1,N),dvar(0,N)]),
        get_attr1(SNODES,INDEXES),
        get_attr2(SNODES,SUCCS),
        all_different(INDEXES),
        derangement1(SUCCS,INDEXES),
        symmetric_alldifferent1(SUCCS,1,SUCCS).
```

```
symmetric_alldifferent_except_0_c([]) :-
        !.

symmetric_alldifferent_except_0_c(NODES) :-
        length(NODES,N),
        symmetric_alldifferent0(NODES,SNODES),
        length(SNODES,N),
        collection(SNODES,[int(1,N),int(0,N)]),
        get_attr1(SNODES,INDEXES),
        get_attr2(SNODES,SUCCS),
        sort(INDEXES,SINDEXES),
        length(SINDEXES,N),
        sym_pairs_skip_zeros(INDEXES,SUCCS,PAIRS),
        keysort(PAIRS,SPAIRS),
        symmetric_alldifferent_check(SPAIRS).

sym_pairs_skip_zeros([],[],[]) :-
        !.

sym_pairs_skip_zeros([_I|R],[0|S],T) :-
        !,
        sym_pairs_skip_zeros(R,S,T).

sym_pairs_skip_zeros([I|R],[J|S],[I-a(J),J-b(I)|T]) :-
        sym_pairs_skip_zeros(R,S,T).

symmetric_alldifferent_except_0_d(Density,NODES) :-
        get_attr2(NODES,SUCCS),
        sort(SUCCS,SORTED),
        length(SUCCS,N),
        length(SORTED,S),
        Density is S/N.
```

B.404 symmetric_alldifferent_loop

◇ META-DATA:

```
ctr_date(symmetric_alldifferent_loop,['20120221']).

ctr_origin(
    symmetric_alldifferent_loop,
    Derived from %c,
    [symmetric_alldifferent]).

ctr_synonyms(
    symmetric_alldifferent_loop,
    [symmetric_alldiff_loop,
     symmetric_alldistinct_loop,
     symm_alldifferent_loop,
     symm_alldiff_loop,
     symm_alldistinct_loop]).

ctr_arguments(
    symmetric_alldifferent_loop,
    ['NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    symmetric_alldifferent_loop,
    [required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    symmetric_alldifferent_loop,
    [symmetric_alldifferent_loop(
         [[index-1,succ-1],
          [index-2,succ-4],
          [index-3,succ-3],
          [index-4,succ-2]])]).

ctr_typical(symmetric_alldifferent_loop,[size('NODES')>=4]).

ctr_exchangeable(
    symmetric_alldifferent_loop,
    [items('NODES',all)]).
```

```
ctr_graph(
    symmetric_alldifferent_loop,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index,nodes2^succ=nodes1^index],
    ['NARC'=size('NODES')],
    []).

ctr_eval(
    symmetric_alldifferent_loop,
    [checker(symmetric_alldifferent_loop_c),
     reformulation(symmetric_alldifferent_loop_r1),
     reformulation(symmetric_alldifferent_loop_r2)]).

ctr_cond_imply(
    symmetric_alldifferent_loop,
    permutation,
    [],
    [],
    index_to_col).

ctr_sol(symmetric_alldifferent_loop,2,0,2,2,-).

ctr_sol(symmetric_alldifferent_loop,3,0,3,4,-).

ctr_sol(symmetric_alldifferent_loop,4,0,4,10,-).

ctr_sol(symmetric_alldifferent_loop,5,0,5,26,-).

ctr_sol(symmetric_alldifferent_loop,6,0,6,76,-).

ctr_sol(symmetric_alldifferent_loop,7,0,7,232,-).

ctr_sol(symmetric_alldifferent_loop,8,0,8,764,-).

ctr_sol(symmetric_alldifferent_loop,9,0,9,2620,-).

ctr_sol(symmetric_alldifferent_loop,10,0,10,9496,-).

symmetric_alldifferent_loop_r1(NODES) :-
        symmetric_alldifferent_loop_r1a(NODES,INODES),
        eval(inverse(INODES)).

symmetric_alldifferent_loop_r1a([],[]).
```

```
symmetric_alldifferent_loop_r1a(
    [[index-INDEX,succ-SUCC]|R],
    [[index-INDEX,succ-SUCC,pred-SUCC]|S]) :-
        symmetric_alldifferent_loop_r1a(R,S).

symmetric_alldifferent_loop_r2([]) :-
        !.

symmetric_alldifferent_loop_r2(NODES) :-
        symmetric_alldifferent0(NODES,SNODES),
        length(SNODES,N),
        collection(SNODES,[int(1,N),dvar(1,N)]),
        get_attr1(SNODES,INDEXES),
        get_attr2(SNODES,SUCCS),
        all_different(INDEXES),
        symmetric_alldifferent1(SUCCS,1,SUCCS).

symmetric_alldifferent_loop_c([]) :-
        !.

symmetric_alldifferent_loop_c(NODES) :-
        length(NODES,N),
        symmetric_alldifferent0(NODES,SNODES),
        length(SNODES,N),
        collection(SNODES,[int(1,N),int(1,N)]),
        get_attr1(SNODES,INDEXES),
        get_attr2(SNODES,SUCCS),
        sort(INDEXES,SINDEXES),
        length(SINDEXES,N),
        sym_pairs_skip_loops(INDEXES,SUCCS,PAIRS),
        keysort(PAIRS,SPAIRS),
        symmetric_alldifferent_check(SPAIRS).

sym_pairs_skip_loops([],[],[]) :-
        !.

sym_pairs_skip_loops([I|R],[I|S],T) :-
        !,
        sym_pairs_skip_loops(R,S,T).

sym_pairs_skip_loops([I|R],[J|S],[I-a(J),J-b(I)|T]) :-
        sym_pairs_skip_loops(R,S,T).
```

B.405 symmetric_cardinality

◇ **META-DATA:**

```
ctr_date(symmetric_cardinality,['20040530','20060818']).

ctr_origin(
    symmetric_cardinality,
    Derived from %c by W.~Kocjan.,
    [global_cardinality]).

ctr_arguments(
    symmetric_cardinality,
    ['VARS'-collection(idvar-int,var-svar,l-int,u-int),
     'VALS'-collection(idval-int,val-svar,l-int,u-int)]).

ctr_restrictions(
    symmetric_cardinality,
    [required('VARS',[idvar,var,l,u]),
     size('VARS')>=1,
     'VARS'^idvar>=1,
     'VARS'^idvar=<size('VARS'),
     distinct('VARS',idvar),
     'VARS'^l>=0,
     'VARS'^l=<'VARS'^u,
     'VARS'^u=<size('VALS'),
     required('VALS',[idval,val,l,u]),
     size('VALS')>=1,
     'VALS'^idval>=1,
     'VALS'^idval=<size('VALS'),
     distinct('VALS',idval),
     'VALS'^l>=0,
     'VALS'^l=<'VALS'^u,
     'VALS'^u=<size('VARS')]).

ctr_example(
    symmetric_cardinality,
    symmetric_cardinality(
        [[idvar-1,var-{3},l-0,u-1],
         [idvar-2,var-{1},l-1,u-2],
         [idvar-3,var-{1,2},l-1,u-2],
         [idvar-4,var-{1,3},l-2,u-3]],
        [[idval-1,val-{2,3,4},l-3,u-4],
         [idval-2,val-{3},l-1,u-1],
         [idval-3,val-{1,4},l-1,u-2],
         [idval-4,val-{},l-0,u-1]])).
```

```
ctr_typical(
    symmetric_cardinality,
    [size('VARS')>1,size('VALS')>1]).

ctr_exchangeable(
    symmetric_cardinality,
    [items('VARS',all),items('VALS',all)]).

ctr_graph(
    symmetric_cardinality,
    ['VARS','VALS'],
    2,
    ['PRODUCT'>>collection(vars,vals)],
    [vars^idvar in_set vals^val#<=>vals^idval in_set vars^var,
     vars^l=<card_set(vars^var),
     vars^u>=card_set(vars^var),
     vals^l=<card_set(vals^val),
     vals^u>=card_set(vals^val)],
    ['NARC'=size('VARS')*size('VALS')],
    []).
```

B.406 symmetric_gcc

◊ **META-DATA:**

```
ctr_date(symmetric_gcc,['20030820','20040530','20060818']).

ctr_origin(
    symmetric_gcc,
    Derived from %c by W.~Kocjan.,
    [global_cardinality]).

ctr_synonyms(symmetric_gcc,[sgcc]).

ctr_arguments(
    symmetric_gcc,
    ['VARS'-collection(idvar-int,var-svar,nocc-dvar),
     'VALS'-collection(idval-int,val-svar,nocc-dvar)]).

ctr_restrictions(
    symmetric_gcc,
    [required('VARS',[idvar,var,nocc]),
     size('VARS')>=1,
     'VARS'^idvar>=1,
     'VARS'^idvar=<size('VARS'),
     distinct('VARS',idvar),
     'VARS'^nocc>=0,
     'VARS'^nocc=<size('VALS'),
     required('VALS',[idval,val,nocc]),
     size('VALS')>=1,
     'VALS'^idval>=1,
     'VALS'^idval=<size('VALS'),
     distinct('VALS',idval),
     'VALS'^nocc>=0,
     'VALS'^nocc=<size('VARS')]).

ctr_example(
    symmetric_gcc,
    symmetric_gcc(
        [[idvar-1,var-{3},nocc-1],
         [idvar-2,var-{1},nocc-1],
         [idvar-3,var-{1,2},nocc-2],
         [idvar-4,var-{1,3},nocc-2]],
        [[idval-1,val-{2,3,4},nocc-3],
         [idval-2,val-{3},nocc-1],
         [idval-3,val-{1,4},nocc-2],
         [idval-4,val-{},nocc-0]])).
```

```
ctr_typical(symmetric_gcc,[size('VARS')>1,size('VALS')>1]).

ctr_exchangeable(
    symmetric_gcc,
    [items('VARS',all),items('VALS',all)]).

ctr_graph(
    symmetric_gcc,
    ['VARS','VALS'],
    2,
    ['PRODUCT'>>collection(vars,vals)],
    [vars^idvar in_set vals^val#<=>vals^idval in_set vars^var,
     vars^nocc=card_set(vars^var),
     vals^nocc=card_set(vals^val)],
    ['NARC'=size('VARS')*size('VALS')],
    []).
```

B.407 tasks_intersection

◇ **META-DATA:**

```
ctr_predefined(tasks_intersection).

ctr_date(tasks_intersection,['20140511']).

ctr_origin(
    tasks_intersection,
    Inspired by video summarization.,
    []).

ctr_synonyms(
    tasks_intersection,
    [intersection_between_sequences_of_tasks,
     intersection_between_intervals,
     intersection_between_tasks_chains]).

ctr_arguments(
    tasks_intersection,
    ['INTERSECTION'-dvar,
     'TASKS1'-collection(origin-dvar,duration-dvar,end-dvar),
     'TASKS2'-collection(origin-dvar,duration-dvar,end-dvar)]).

ctr_restrictions(
    tasks_intersection,
    ['INTERSECTION'>=0,
     require_at_least(2,'TASKS1',[origin,duration,end]),
     require_at_least(2,'TASKS2',[origin,duration,end]),
     'TASKS1'^duration>=0,
     'TASKS2'^duration>=0,
     'TASKS1'^origin=<'TASKS1'^end,
     'TASKS2'^origin=<'TASKS2'^end,
     'INTERSECTION'=<sum('TASKS1'^duration),
     'INTERSECTION'=<sum('TASKS2'^duration)]).

ctr_example(
    tasks_intersection,
    tasks_intersection(
        3,
        [[origin-2,duration-2,end-4],
         [origin-7,duration-2,end-9],
         [origin-9,duration-0,end-9]],
        [[origin-1,duration-3,end-4],
         [origin-5,duration-1,end-6],
```

```
         [origin-8,duration-2,end-10]])).

ctr_typical(
    tasks_intersection,
    ['INTERSECTION'>0,
     size('TASKS1')>1,
     size('TASKS2')>1,
     range('TASKS1'^duration)>1,
     range('TASKS2'^duration)>1]).

ctr_eval(
    tasks_intersection,
    [reformulation(tasks_intersection_r)]).

ctr_pure_functional_dependency(tasks_intersection,[]).

ctr_functional_dependency(tasks_intersection,1,[2,3]).

tasks_intersection_r(INTERSECTION,TASKS1,TASKS2) :-
        check_type(dvar_gteq(0),INTERSECTION),
        collection(TASKS1,[dvar,dvar_gteq(0),dvar]),
        collection(TASKS2,[dvar,dvar_gteq(0),dvar]),
        tasks_intersection1(TASKS1),
        tasks_intersection1(TASKS2),
        tasks_intersection2(TASKS1,TASKS2,INTER),
        call(INTERSECTION#=INTER).

tasks_intersection1([[origin-O,duration-D,end-E]]) :-
        !,
        E#=O+D.

tasks_intersection1(
    [[origin-O1,duration-D1,end-E1],
     [origin-O2,duration-D2,end-E2]|
     R]) :-
        E1#=O1+D1,
        E1#=<O2,
        tasks_intersection1([[origin-O2,duration-D2,end-E2]|R]).

tasks_intersection2([],_28816,0) :-
        !.

tasks_intersection2([TASK|R],TASKS2,I+S) :-
        tasks_intersection3(TASKS2,TASK,INTER),
        call(I#=INTER),
        tasks_intersection2(R,TASKS2,S).
```

```
tasks_intersection3([],_28816,0) :-
        !.

tasks_intersection3(
    [[origin-O2,duration-_28831,end-E2]|R],
    [origin-O1,duration-D1,end-E1],
    INTER+S) :-
        INTER#=max(min(E1,E2)-max(O1,O2),0),
        tasks_intersection3(R,[origin-O1,duration-D1,end-E1],S).
```

B.408 temporal_path

◇ META-DATA:

```
ctr_date(
    temporal_path,
    ['20000128','20030820','20060818','20090511']).

ctr_origin(temporal_path,'ILOG',[]).

ctr_arguments(
    temporal_path,
    ['NPATH'-dvar,
     NODES-
     collection(index-int,succ-dvar,start-dvar,end-dvar)]).

ctr_restrictions(
    temporal_path,
    ['NPATH'>=1,
     'NPATH'=<size('NODES'),
     required('NODES',[index,succ,start,end]),
     size('NODES')>0,
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES'),
     'NODES'^start=<'NODES'^end]).

ctr_example(
    temporal_path,
    temporal_path(
        2,
        [[index-1,succ-2,start-0,end-1],
         [index-2,succ-6,start-3,end-5],
         [index-3,succ-4,start-0,end-3],
         [index-4,succ-5,start-4,end-6],
         [index-5,succ-7,start-7,end-8],
         [index-6,succ-6,start-7,end-9],
         [index-7,succ-7,start-9,end-10]])).

ctr_typical(
    temporal_path,
    ['NPATH'<size('NODES'),
     size('NODES')>1,
     'NODES'^start<'NODES'^end]).
```

```
ctr_exchangeable(
    temporal_path,
    [items('NODES',all),
     translate(['NODES'^start,'NODES'^end])]).

ctr_graph(
    temporal_path,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index,
     nodes1^succ=nodes1^index#\/nodes1^end=<nodes2^start,
     nodes1^start=<nodes1^end,
     nodes2^start=<nodes2^end],
    ['MAX_ID'=<1,'NCC'='NPATH','NVERTEX'=size('NODES')],
    []).

ctr_eval(temporal_path,[reformulation(temporal_path_r)]).

ctr_functional_dependency(temporal_path,1,[2]).

ctr_application(temporal_path,[2]).

temporal_path_r(NPATH,NODES) :-
        temporal_path0(NODES,SNODES),
        length(SNODES,N),
        N>0,
        check_type(dvar(1,N),NPATH),
        collection(SNODES,[int(1,N),dvar(1,N),dvar,dvar]),
        get_attr1(SNODES,INDEXES),
        get_attr2(SNODES,SUCCS),
        get_attr3(SNODES,STARTS),
        get_attr4(SNODES,ENDS),
        all_different(INDEXES),
        ori_end(STARTS,ENDS),
        temporal_path1(INDEXES,SUCCS,TNODES),
        eval(path(NPATH,TNODES)),
        temporal_path2(SUCCS,ENDS,[],STARTS).

temporal_path0(NODES,SNODES) :-
        temporal_path0a(NODES,L),
        sort(L,S),
        temporal_path0a(SNODES,S),
        !.
```

```
temporal_path0a([],[]).

temporal_path0a(
    [[index-INDEX,succ-SUCC,start-START,end-END]|R],
    [INDEX-(SUCC,START,END)|T]) :-
        temporal_path0a(R,T).

temporal_path1([],[],[]).

temporal_path1(
    [INDEX|RINDEX],
    [SUCC|RSUCC],
    [[index-INDEX,succ-SUCC]|RNODES]) :-
        temporal_path1(RINDEX,RSUCC,RNODES).

temporal_path2([],[],_54121,_54122).

temporal_path2(
    [SUCCi|RSUCC],
    [ENDi|REND],
    PREV_STARTS,
    [_STARTi|RSTART]) :-
        append(PREV_STARTS,[ENDi],NEW_PREV_STARTS),
        append(NEW_PREV_STARTS,RSTART,TABLE),
        element(SUCCi,TABLE,START_SUCCi),
        ENDi#=<START_SUCCi,
        temporal_path2(RSUCC,REND,NEW_PREV_STARTS,RSTART).
```

B.409 tour

◇ **META-DATA:**

```
ctr_date(tour,['20030820','20060819']).

ctr_origin(
    tour,
    \cite{AlthausBockmayrElfKasperJungerMehlhorn02},
    []).

ctr_synonyms(tour,[atour,cycle]).

ctr_arguments(tour,['NODES'-collection(index-int,succ-svar)]).

ctr_restrictions(
    tour,
    [size('NODES')>=3,
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index)]).

ctr_example(
    tour,
    tour(
        [[index-1,succ-{2,4}],
         [index-2,succ-{1,3}],
         [index-3,succ-{2,4}],
         [index-4,succ-{1,3}]])).

ctr_exchangeable(tour,[items('NODES',all)]).

ctr_graph(
    tour,
    ['NODES'],
    2,
    ['CLIQUE'(=\=)>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ#<=>
     nodes1^index in_set nodes2^succ],
    ['NARC'=size('NODES')*size('NODES')-size('NODES')],
    []).

ctr_graph(
    tour,
    ['NODES'],
```

```
    2,
    ['CLIQUE'(=\=)>>collection(nodes1,nodes2)],
    [nodes2^index in_set nodes1^succ],
    ['MIN_NSCC'=size('NODES'),
     'MIN_ID'=2,
     'MAX_ID'=2,
     'MIN_OD'=2,
     'MAX_OD'=2],
    []).

ctr_application(tour,[1]).
```

B.410 track

◊ **META-DATA:**

```
ctr_date(track,['20030820','20060819','20090510']).

ctr_origin(track,'\\cite{Marte01}',[]).

ctr_arguments(
    track,
    ['NTRAIL'-int,
     'TASKS'-collection(trail-int,origin-dvar,end-dvar)]).

ctr_restrictions(
    track,
    ['NTRAIL'>0,
     'NTRAIL'=<size('TASKS'),
     size('TASKS')>0,
     required('TASKS',[trail,origin,end]),
     'TASKS'^origin=<'TASKS'^end]).

ctr_example(
    track,
    track(
        2,
        [[trail-1,origin-1,end-2],
         [trail-2,origin-1,end-2],
         [trail-1,origin-2,end-4],
         [trail-2,origin-2,end-3],
         [trail-2,origin-3,end-4]])).

ctr_typical(
    track,
    ['NTRAIL'<size('TASKS'),
     size('TASKS')>1,
     range('TASKS'^trail)>1,
     'TASKS'^origin<'TASKS'^end]).

ctr_exchangeable(
    track,
    [items('TASKS',all),
     vals(['TASKS'^trail],int,=\=,all,dontcare),
     translate(['TASKS'^origin,'TASKS'^end])]).

ctr_derived_collections(
    track,
```

```
    [col(TIME_POINTS-
         collection(origin-dvar,end-dvar,point-dvar),
         [item(
              origin-'TASKS'^origin,
              end-'TASKS'^end,
              point-'TASKS'^origin),
          item(
              origin-'TASKS'^origin,
              end-'TASKS'^end,
              point-'TASKS'^end-1)])]).

ctr_graph(
    track,
    ['TASKS'],
    1,
    ['SELF'>>collection(tasks)],
    [tasks^origin=<tasks^end],
    ['NARC'=size('TASKS')],
    []).

ctr_graph(
    track,
    ['TIME_POINTS','TASKS'],
    2,
    ['PRODUCT'>>collection(time_points,tasks)],
    [time_points^end>time_points^origin,
     tasks^origin=<time_points^point,
     time_points^point<tasks^end],
    [],
    [],
    [SUCC>>
     [source,
      variables-
      col('VARIABLES'-collection(var-dvar),
          [item(var-'TASKS'^trail)])]],
    [nvalue('NTRAIL',variables)]).

ctr_eval(track,[reformulation(track_r)]).

ctr_application(track,[2]).

track_r(NTRAIL,TASKS) :-
        length(TASKS,N),
        check_type(dvar(1,N),NTRAIL),
        collection(TASKS,[int(1,N),dvar,dvar]),
        get_attr1(TASKS,TRAILS),
```

```
        get_attr2(TASKS,ORIGINS),
        get_attr3(TASKS,ENDS),
        ori_end(ORIGINS,ENDS),
        track1(
            ORIGINS,
            ENDS,
            TRAILS,
            1,
            ORIGINS,
            ENDS,
            TRAILS,
            NTRAIL),
        track3(
            ORIGINS,
            ENDS,
            TRAILS,
            1,
            ORIGINS,
            ENDS,
            TRAILS,
            NTRAIL).

track1([],[],[],_60504,_60505,_60506,_60507,_60508).

track1([Oi|RO],[Ei|RE],[Ti|TC],I,ORIGINS,ENDS,TRAILS,NTRAIL) :-
        track2(ORIGINS,ENDS,TRAILS,1,I,Oi,Ei,Ti,COLi),
        nvalue(NTRAIL,COLi),
        I1 is I+1,
        track1(RO,RE,TC,I1,ORIGINS,ENDS,TRAILS,NTRAIL).

track2([],[],[],_60504,_60505,_60506,_60507,_60508,[]).

track2([_60513|RO],[_60517|RE],[_60521|RT],J,I,Oi,Ei,Ti,[Ti|R]) :-
        I=J,
        !,
        J1 is J+1,
        track2(RO,RE,RT,J1,I,Oi,Ei,Ti,R).

track2([Oj|RO],[Ej|RE],[Tj|RT],J,I,Oi,Ei,Ti,[Tij|R]) :-
        I=\=J,
        K in 1..2,
        Min is min(Ti,Tj),
        Max is max(Ti,Tj),
        Tij in Min..Max,
        element(K,[Ti,Tj],Tij),
        Oj#=<Oi#/\Ej#>Oi#/\Tij#=Tj#\/
```

```
        (Oj#>Oi#\/Ej#=<Oi)#/\Tij#=Ti,
        J1 is J+1,
        track2(RO,RE,RT,J1,I,Oi,Ei,Ti,R).

track3([],[],[],_60504,_60505,_60506,_60507,_60508).

track3([Oi|RO],[Ei|RE],[Ti|TC],I,ORIGINS,ENDS,TRAILS,NTRAIL) :-
        track4(ORIGINS,ENDS,TRAILS,1,I,Oi,Ei,Ti,COLi),
        nvalue(NTRAIL,COLi),
        I1 is I+1,
        track3(RO,RE,TC,I1,ORIGINS,ENDS,TRAILS,NTRAIL).

track4([],[],[],_60504,_60505,_60506,_60507,_60508,[]).

track4([_60513|RO],[_60517|RE],[_60521|RT],J,I,Oi,Ei,Ti,[Ti|R]) :-
        I=J,
        !,
        J1 is J+1,
        track4(RO,RE,RT,J1,I,Oi,Ei,Ti,R).

track4([Oj|RO],[Ej|RE],[Tj|RT],J,I,Oi,Ei,Ti,[Tij|R]) :-
        I=\=J,
        K in 1..2,
        Min is min(Ti,Tj),
        Max is max(Ti,Tj),
        Tij in Min..Max,
        element(K,[Ti,Tj],Tij),
        Oj#=<Ei-1#/\Ej#>Ei-1#/\Tij#=Tj#\/
        (Oj#>Ei-1#\/Ej#=<Ei-1)#/\Tij#=Ti,
        J1 is J+1,
        track4(RO,RE,RT,J1,I,Oi,Ei,Ti,R).
```

B.411 tree

◊ **META-DATA:**

```
ctr_date(tree,['20000128','20030820','20060819']).

ctr_origin(tree,'N.~Beldiceanu',[]).

ctr_arguments(
    tree,
    ['NTREES'-dvar,'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    tree,
    ['NTREES'>=1,
     'NTREES'=<size('NODES'),
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    tree,
    [tree(
         2,
         [[index-1,succ-1],
          [index-2,succ-5],
          [index-3,succ-5],
          [index-4,succ-7],
          [index-5,succ-1],
          [index-6,succ-1],
          [index-7,succ-7],
          [index-8,succ-5]]),
     tree(
         8,
         [[index-1,succ-1],
          [index-2,succ-2],
          [index-3,succ-3],
          [index-4,succ-4],
          [index-5,succ-5],
          [index-6,succ-6],
          [index-7,succ-7],
          [index-8,succ-8]]),
     tree(
```

```
          7,
          [[index-1,succ-6],
           [index-2,succ-2],
           [index-3,succ-3],
           [index-4,succ-4],
           [index-5,succ-5],
           [index-6,succ-6],
           [index-7,succ-7],
           [index-8,succ-8]])]).

ctr_typical(tree,['NTREES'<size('NODES'),size('NODES')>2]).

ctr_exchangeable(tree,[items('NODES',all)]).

ctr_graph(
    tree,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1,'NCC'='NTREES'],
    []).

ctr_eval(tree,[reformulation(tree_r)]).

ctr_functional_dependency(tree,1,[2]).

ctr_application(tree,[2]).

ctr_sol(tree,2,0,2,3,[1-2,2-1]).

ctr_sol(tree,3,0,3,16,[1-9,2-6,3-1]).

ctr_sol(tree,4,0,4,125,[1-64,2-48,3-12,4-1]).

ctr_sol(tree,5,0,5,1296,[1-625,2-500,3-150,4-20,5-1]).

ctr_sol(tree,6,0,6,16807,[1-7776,2-6480,3-2160,4-360,5-30,6-1]).

ctr_sol(
    tree,
    7,
    0,
    7,
    262144,
    [1-117649,2-100842,3-36015,4-6860,5-735,6-42,7-1]).
```

```
ctr_sol(
    tree,
    8,
    0,
    8,
    4782969,
    [1-2097152,
     2-1835008,
     3-688128,
     4-143360,
     5-17920,
     6-1344,
     7-56,
     8-1]).

tree_r(NTREES,NODES) :-
        length(NODES,N),
        check_type(dvar(1,N),NTREES),
        collection(NODES,[int(1,N),dvar(1,N)]),
        get_attr1(NODES,INDEXES),
        get_attr2(NODES,SUCCS),
        all_different(INDEXES),
        length(RANKS,N),
        domain(RANKS,1,N),
        tree1(SUCCS,RANKS,INDEXES,RANKS,INDEXES,Term),
        call(NTREES#=Term).

tree1([],[],_82590,_82591,_82592,0).

tree1([S|U],[R|P],[I|K],RANKS,INDEXES,B+T) :-
        S#=I#<=>B,
        tree2(S,R,I,RANKS,INDEXES),
        tree1(U,P,K,RANKS,INDEXES,T).

tree2(_82591,_82592,_82593,_82594,[]) :-
        !.

tree2(S_I,R_I,I,[R_J|P],[J|K]) :-
        S_I#=J#/\I#\=J#=>R_I#>R_J,
        tree2(S_I,R_I,I,P,K).
```

B.412 tree_range

◇ META-DATA:

```
ctr_date(
    tree_range,
    ['20030820','20040727','20060819','20090923']).

ctr_origin(tree_range,'Derived from %c.',[tree]).

ctr_arguments(
    tree_range,
    ['NTREES'-dvar,
     'R'-dvar,
     'NODES'-collection(index-int,succ-dvar)]).

ctr_restrictions(
    tree_range,
    ['NTREES'>=0,
     'R'>=0,
     'R'<size('NODES'),
     size('NODES')>0,
     required('NODES',[index,succ]),
     'NODES'^index>=1,
     'NODES'^index=<size('NODES'),
     distinct('NODES',index),
     'NODES'^succ>=1,
     'NODES'^succ=<size('NODES')]).

ctr_example(
    tree_range,
    tree_range(
        2,
        1,
        [[index-1,succ-1],
         [index-2,succ-5],
         [index-3,succ-5],
         [index-4,succ-7],
         [index-5,succ-1],
         [index-6,succ-1],
         [index-7,succ-7],
         [index-8,succ-5]])).

ctr_typical(
    tree_range,
    ['NTREES'<size('NODES'),size('NODES')>2]).
```

```
ctr_exchangeable(tree_range,[items('NODES',all)]).

ctr_graph(
    tree_range,
    ['NODES'],
    2,
    ['CLIQUE'>>collection(nodes1,nodes2)],
    [nodes1^succ=nodes2^index],
    ['MAX_NSCC'=<1,'NCC'='NTREES','RANGE_DRG'='R'],
    []).

ctr_eval(tree_range,[reformulation(tree_range_r)]).

ctr_functional_dependency(tree_range,1,[3]).

ctr_functional_dependency(tree_range,2,[3]).

ctr_application(tree_range,[3]).

tree_range_r(NTREES,R,NODES) :-
        tree_range0(NODES,SNODES),
        length(SNODES,N),
        N>0,
        N1 is N-1,
        check_type(dvar(1,N),NTREES),
        check_type(dvar(0,N1),R),
        collection(SNODES,[int(1,N),dvar(1,N)]),
        get_attr1(SNODES,INDEXES),
        get_attr2(SNODES,SUCCS),
        all_different(INDEXES),
        eval(tree(NTREES,SNODES)),
        tree_range1(
            INDEXES,
            SUCCS,
            DISTS1,
            DISTS2,
            OCCS1,
            OCCS2,
            SUCCS1,
            LS,
            OLS),
        eval(domain(DISTS1,0,N)),
        tree_range2(INDEXES,SUCCS,N,[],DISTS2),
        eval(domain(OCCS1,0,N)),
        eval(global_cardinality(SUCCS1,OCCS2)),
```

```
        eval(domain(LS,0,1)),
        tree_range3(OLS),
        eval(in_interval(MIN,0,N)),
        eval(open_minimum(MIN,OLS)),
        eval(in_interval(MAX,0,N)),
        eval(maximum(MAX,DISTS1)),
        eval(
            scalar_product(
                [[coeff-1,var-MAX],[coeff- -1,var-MIN]],
                =,
                R)).

tree_range0(NODES,SNODES) :-
        tree_range0a(NODES,L),
        sort(L,S),
        tree_range0a(SNODES,S),
        !.

tree_range0a([],[]).

tree_range0a([[index-I,succ-S]|R],[I-S|T]) :-
        tree_range0a(R,T).

tree_range1([],[],[],[],[],[],[],[],[]).

tree_range1(
    [I|RI],
    [S|RS],
    [[var-V]|RV1],
    [[value-V]|RV2],
    [[var-O]|RO],
    [[val-I,noccurrence-O]|RIO],
    [[var-S]|RSS],
    [[var-L]|RL],
    [[var-O,bool-L]|ROL]) :-
        tree_range1(RI,RS,RV1,RV2,RO,RIO,RSS,RL,ROL).

tree_range2([],[],_54501,_54502,_54503).

tree_range2(
    [_IND|RIND],
    [SUCC|RSUCC],
    N,
    DISTS_BEFORE,
    DISTS_AFTER) :-
        append(DISTS_BEFORE,[[value-0]],TD),
```

```
        DISTS_AFTER=[[value-D]|RDISTS_AFTER],
        append(TD,RDISTS_AFTER,TABLE),
        eval(in_interval(DS,0,N)),
        eval(element(SUCC,TABLE,DS)),
        eval(
            scalar_product(
                [[coeff-1,var-D],[coeff- -1,var-DS]],
                =,
                1)),
        append(DISTS_BEFORE,[[value-D]],DISTS_BEFORE1),
        tree_range2(RIND,RSUCC,N,DISTS_BEFORE1,RDISTS_AFTER).

tree_range3([]).

tree_range3([[var-O,bool-L]|ROL]) :-
        L#<=>O#>0,
        tree_range3(ROL).
```

B.413 tree_resource

◇ META-DATA:

```
ctr_date(tree_resource,['20030820','20060819']).

ctr_origin(tree_resource,'Derived from %c.',[tree]).

ctr_arguments(
    tree_resource,
    ['RESOURCE'-collection(id-int,nb_task-dvar),
     'TASK'-collection(id-int,father-dvar,resource-dvar)]).

ctr_restrictions(
    tree_resource,
    [size('RESOURCE')>0,
     required('RESOURCE',[id,nb_task]),
     'RESOURCE'^id>=1,
     'RESOURCE'^id=<size('RESOURCE'),
     distinct('RESOURCE',id),
     'RESOURCE'^nb_task>=0,
     'RESOURCE'^nb_task=<size('TASK'),
     required('TASK',[id,father,resource]),
     'TASK'^id>size('RESOURCE'),
     'TASK'^id=<size('RESOURCE')+size('TASK'),
     distinct('TASK',id),
     'TASK'^father>=1,
     'TASK'^father=<size('RESOURCE')+size('TASK'),
     'TASK'^resource>=1,
     'TASK'^resource=<size('RESOURCE')]).

ctr_example(
    tree_resource,
    tree_resource(
        [[id-1,nb_task-4],[id-2,nb_task-0],[id-3,nb_task-1]],
        [[id-4,father-8,resource-1],
         [id-5,father-3,resource-3],
         [id-6,father-8,resource-1],
         [id-7,father-1,resource-1],
         [id-8,father-1,resource-1]])).

ctr_typical(
    tree_resource,
    [size('RESOURCE')>0,size('TASK')>size('RESOURCE')]).

ctr_exchangeable(
```

```
    tree_resource,
    [items('RESOURCE',all),items('TASK',all)]).

ctr_derived_collections(
    tree_resource,
    [col(RESOURCE_TASK-
         collection(index-int,succ-dvar,name-dvar),
         [item(
              index-'RESOURCE'^id,
              succ-'RESOURCE'^id,
              name-'RESOURCE'^id),
          item(
              index-'TASK'^id,
              succ-'TASK'^father,
              name-'TASK'^resource)])]).

ctr_graph(
    tree_resource,
    ['RESOURCE_TASK'],
    2,
    ['CLIQUE'>>collection(resource_task1,resource_task2)],
    [resource_task1^succ=resource_task2^index,
     resource_task1^name=resource_task2^name],
    ['MAX_NSCC'=<1,
     'NCC'=size('RESOURCE'),
     'NVERTEX'=size('RESOURCE')+size('TASK')],
    []).

ctr_graph(
    tree_resource,
    ['RESOURCE_TASK'],
    2,
    foreach(
        RESOURCE,
        ['CLIQUE'>>collection(resource_task1,resource_task2)]),
    [resource_task1^succ=resource_task2^index,
     resource_task1^name=resource_task2^name,
     resource_task1^name='RESOURCE'^id],
    ['NVERTEX'='RESOURCE'^nb_task+1],
    []).

ctr_eval(tree_resource,[reformulation(tree_resource_r)]).

ctr_application(tree_resource,[2]).

tree_resource_r(RESOURCE,TASK) :-
```

```
        length(RESOURCE,R),
        length(TASK,T),
        R>0,
        collection(RESOURCE,[int(1,R),dvar(0,T)]),
        get_attr1(RESOURCE,RIDS),
        get_attr2(RESOURCE,RNBTASKS),
        all_different(RIDS),
        R1 is R+1,
        RT is R+T,
        collection(TASK,[int(R1,RT),dvar(1,RT),dvar(1,R)]),
        get_attr1(TASK,TIDS),
        get_attr2(TASK,TFATHERS),
        get_attr3(TASK,TRESOURCES),
        all_different(TIDS),
        tree_resource1(RIDS,CNODES1),
        tree_resource2(TIDS,TFATHERS,CNODES2),
        append(CNODES1,CNODES2,NODES),
        eval(tree(R,NODES)),
        tree_resource3(TIDS,TRESOURCES,TIR),
        sort(TIR,STIR),
        tree_resource4(1,R,INC),
        append(INC,STIR,TAB),
        tree_resource5(TAB,TABR),
        tree_resource6(TFATHERS,TRESOURCES,TABR),
        tree_resource7(STIR,GCVARS),
        tree_resource8(RIDS,RNBTASKS,GCVALS),
        eval(global_cardinality(GCVARS,GCVALS)).

tree_resource1([],[]).

tree_resource1([I|R],[[index-I,succ-I]|S]) :-
        tree_resource1(R,S).

tree_resource2([],[],[]).

tree_resource2([I|R],[F|S],[[index-I,succ-F]|T]) :-
        tree_resource2(R,S,T).

tree_resource3([],[],[]).

tree_resource3([I|RI],[R|RR],[I-R|S]) :-
        tree_resource3(RI,RR,S).

tree_resource4(I,R,[]) :-
        I>R,
        !.
```

```
tree_resource4(I,R,[I-I|S]) :-
        I=<R,
        I1 is I+1,
        tree_resource4(I1,R,S).

tree_resource5([],[]).

tree_resource5([_60800-R|S],[[value-R]|T]) :-
        tree_resource5(S,T).

tree_resource6([],[],_60794).

tree_resource6([Fi|RF],[Ri|RR],TABR) :-
        eval(element(Fi,TABR,Ri)),
        tree_resource6(RF,RR,TABR).

tree_resource7([],[]).

tree_resource7([_60800-V|R],[[var-V]|S]) :-
        tree_resource7(R,S).

tree_resource8([],[],[]).

tree_resource8([V|R],[O|S],[[val-V,noccurrence-O]|T]) :-
        tree_resource8(R,S,T).
```

B.414 twin

◇ META-DATA:

```
ctr_predefined(twin).

ctr_date(twin,['20111129']).

ctr_origin(
    twin,
    Pairs of variables related by hiden %c constraints sharing the same tabl
    [element]).

ctr_arguments(twin,['PAIRS'-collection(x-dvar,y-dvar)]).

ctr_restrictions(
    twin,
    [required('PAIRS',x),required('PAIRS',y),size('PAIRS')>0]).

ctr_example(
    twin,
    twin(
        [[x-1,y-8],
         [x-9,y-6],
         [x-1,y-8],
         [x-5,y-0],
         [x-6,y-7],
         [x-9,y-6]])).

ctr_typical(
    twin,
    [size('PAIRS')>1,
     size('PAIRS')>nval('PAIRS'^x),
     size('PAIRS')>nval('PAIRS'^y),
     nval('PAIRS'^x)>1,
     nval('PAIRS'^y)>1,
     nval('PAIRS'^x)=nval('PAIRS'^y),
     nval('PAIRS'^x)<size('PAIRS'),
     nval('PAIRS'^y)<size('PAIRS')]).

ctr_eval(twin,[checker(twin_c),reformulation(twin_r)]).

ctr_contractible(twin,[],'PAIRS',any).

twin_c(PAIRS) :-
        collection(PAIRS,[int,int]),
```

```
        length(PAIRS,N),
        N>0,
        get_attr12(PAIRS,P12),
        sort(P12,S12),
        twin1(S12),
        get_attr21(PAIRS,P21),
        sort(P21,S21),
        twin1(S21).

twin1([]) :-
        !.

twin1([_26043]) :-
        !.

twin1([X1-_26047,X2-Y|R]) :-
        X1\==X2,
        twin1([X2-Y|R]).

twin_r(PAIRS) :-
        collection(PAIRS,[dvar,dvar]),
        length(PAIRS,N),
        N>0,
        get_attr1(PAIRS,XS),
        get_attr2(PAIRS,YS),
        get_min_list_dvar(XS,_26090,MinX),
        get_min_list_dvar(YS,_26099,MinY),
        get_max_list_dvar(YS,_26108,MaxY),
        RangeY is MaxY-MinY+1,
        twin1(XS,YS,MinX,MinY,RangeY,XYS),
        NPAIRS in 1..N,
        nvalue(NPAIRS,XS),
        nvalue(NPAIRS,YS),
        nvalue(NPAIRS,XYS).

twin1([],[],_26044,_26045,_26046,[]) :-
        !.

twin1([X|RX],[Y|RY],MinX,MinY,RangeY,[XY|RXY]) :-
        XY#=RangeY*(X-MinX)+(Y-MinY),
        twin1(RX,RY,MinX,MinY,RangeY,RXY).
```

B.415 two_layer_edge_crossing

◇ **META-DATA:**

```
ctr_date(two_layer_edge_crossing,['20030820','20060819']).

ctr_origin(
    two_layer_edge_crossing,
    Inspired by \cite{HararySchwenk72}.,
    []).

ctr_arguments(
    two_layer_edge_crossing,
    ['NCROSS'-dvar,
     'VERTICES_LAYER1'-collection(id-int,pos-dvar),
     'VERTICES_LAYER2'-collection(id-int,pos-dvar),
     'EDGES'-collection(id-int,vertex1-int,vertex2-int)]).

ctr_restrictions(
    two_layer_edge_crossing,
    ['NCROSS'>=0,
     required('VERTICES_LAYER1',[id,pos]),
     'VERTICES_LAYER1'^id>=1,
     'VERTICES_LAYER1'^id=<size('VERTICES_LAYER1'),
     distinct('VERTICES_LAYER1',id),
     distinct('VERTICES_LAYER1',pos),
     required('VERTICES_LAYER2',[id,pos]),
     'VERTICES_LAYER2'^id>=1,
     'VERTICES_LAYER2'^id=<size('VERTICES_LAYER2'),
     distinct('VERTICES_LAYER2',id),
     distinct('VERTICES_LAYER2',pos),
     required('EDGES',[id,vertex1,vertex2]),
     'EDGES'^id>=1,
     'EDGES'^id=<size('EDGES'),
     distinct('EDGES',id),
     'EDGES'^vertex1>=1,
     'EDGES'^vertex1=<size('VERTICES_LAYER1'),
     'EDGES'^vertex2>=1,
     'EDGES'^vertex2=<size('VERTICES_LAYER2')]).

ctr_example(
    two_layer_edge_crossing,
    two_layer_edge_crossing(
        2,
        [[id-1,pos-1],[id-2,pos-2]],
        [[id-1,pos-3],[id-2,pos-1],[id-3,pos-2]],
```

```
        [[id-1,vertex1-2,vertex2-2],
         [id-2,vertex1-2,vertex2-3],
         [id-3,vertex1-1,vertex2-1]])).

ctr_typical(
    two_layer_edge_crossing,
    [size('VERTICES_LAYER1')>1,
     size('VERTICES_LAYER2')>1,
     size('EDGES')>=size('VERTICES_LAYER1'),
     size('EDGES')>=size('VERTICES_LAYER2')]).

ctr_exchangeable(
    two_layer_edge_crossing,
    [args(
         [['NCROSS'],
          ['VERTICES_LAYER1','VERTICES_LAYER2'],
          ['EDGES']]),
     items('VERTICES_LAYER1',all),
     items('VERTICES_LAYER2',all)]).

ctr_derived_collections(
    two_layer_edge_crossing,
    [col(EDGES_EXTREMITIES-
         collection(layer1-dvar,layer2-dvar),
         [item(
              layer1-
              'EDGES'^vertex1('VERTICES_LAYER1',pos,id),
              layer2-
              'EDGES'^vertex2('VERTICES_LAYER2',pos,id))])]).

ctr_graph(
    two_layer_edge_crossing,
    ['EDGES_EXTREMITIES'],
    2,
    ['CLIQUE'(<)>>
     collection(edges_extremities1,edges_extremities2)],
    [edges_extremities1^layer1<
     edges_extremities2^layer1#/\
     edges_extremities1^layer2>edges_extremities2^layer2#\/
     edges_extremities1^layer1>edges_extremities2^layer1#/\
     edges_extremities1^layer2<edges_extremities2^layer2],
    ['NARC'='NCROSS'],
    []).

ctr_pure_functional_dependency(two_layer_edge_crossing,[]).
```

```
ctr_functional_dependency(two_layer_edge_crossing,1,[2,3,4]).
```

B.416 two_orth_are_in_contact

◊ **META-DATA:**

```
ctr_date(
    two_orth_are_in_contact,
    ['20030820','20040530','20060819']).

ctr_origin(
    two_orth_are_in_contact,
    \cite{Roach84}, used for defining %c.,
    [orths_are_connected]).

ctr_types(
    two_orth_are_in_contact,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    two_orth_are_in_contact,
    ['ORTHOTOPE1'-'ORTHOTOPE','ORTHOTOPE2'-'ORTHOTOPE']).

ctr_restrictions(
    two_orth_are_in_contact,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2')]).

ctr_example(
    two_orth_are_in_contact,
    two_orth_are_in_contact(
        [[ori-1,siz-3,end-4],[ori-5,siz-2,end-7]],
        [[ori-3,siz-2,end-5],[ori-2,siz-3,end-5]])).

ctr_typical(two_orth_are_in_contact,[size('ORTHOTOPE')>1]).

ctr_exchangeable(
    two_orth_are_in_contact,
    [args([['ORTHOTOPE1','ORTHOTOPE2']]),
     items_sync('ORTHOTOPE1','ORTHOTOPE2',all)]).

ctr_graph(
    two_orth_are_in_contact,
```

```
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT'(=)>>collection(orthotope1,orthotope2)],
    [orthotope1^end>orthotope2^ori,
     orthotope2^end>orthotope1^ori],
    ['NARC'=size('ORTHOTOPE1')-1],
    []).

ctr_graph(
    two_orth_are_in_contact,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT'(=)>>collection(orthotope1,orthotope2)],
    [max(0,
         max(orthotope1^ori,orthotope2^ori)-
         min(orthotope1^end,orthotope2^end))=
     0],
    ['NARC'=size('ORTHOTOPE1')],
    []).

ctr_eval(
    two_orth_are_in_contact,
    [automaton(two_orth_are_in_contact_a)]).

two_orth_are_in_contact_a(FLAG,ORTHOTOPE1,ORTHOTOPE2) :-
        length(ORTHOTOPE1,D1),
        length(ORTHOTOPE2,D2),
        D1>0,
        D2>0,
        D1=D2,
        collection(ORTHOTOPE1,[dvar,dvar_gteq(1),dvar]),
        collection(ORTHOTOPE2,[dvar,dvar_gteq(1),dvar]),
        get_attr1(ORTHOTOPE1,ORIS1),
        get_attr3(ORTHOTOPE1,ENDS1),
        check_lesseq(ORIS1,ENDS1),
        get_attr1(ORTHOTOPE2,ORIS2),
        get_attr3(ORTHOTOPE2,ENDS2),
        check_lesseq(ORIS2,ENDS2),
        eval(orth_link_ori_siz_end(ORTHOTOPE1)),
        eval(orth_link_ori_siz_end(ORTHOTOPE2)),
        two_orth_are_in_contact_signature(
            ORTHOTOPE1,
            ORTHOTOPE2,
            SIGNATURE),
        AUTOMATON=
        automaton(
```

```
            SIGNATURE,
            _49040,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,0,s),arc(s,1,t),arc(t,0,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1,2],AUTOMATON).

two_orth_are_in_contact_signature([],[],[]).

two_orth_are_in_contact_signature(
    [[ori-ORI1,siz-SIZ1,end-END1]|Q1],
    [[ori-ORI2,siz-SIZ2,end-END2]|Q2],
    [S|Ss]) :-
        S in 0..2,
        SIZ1#>0#/\SIZ2#>0#/\END1#>ORI2#/\END2#>ORI1#<=>S#=0,
        SIZ1#>0#/\SIZ2#>0#/\(END1#=ORI2#\/END2#=ORI1)#<=>S#=1,
        two_orth_are_in_contact_signature(Q1,Q2,Ss).
```

B.417 two_orth_column

♢ META-DATA:

```
ctr_date(two_orth_column,['20030820']).

ctr_origin(
    two_orth_column,
    Used for defining %c.,
    [diffn_column]).

ctr_types(
    two_orth_column,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    two_orth_column,
    ['ORTHOTOPE1'-'ORTHOTOPE',
     'ORTHOTOPE2'-'ORTHOTOPE',
     'DIM'-int]).

ctr_restrictions(
    two_orth_column,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2'),
     'DIM'>0,
     'DIM'=<size('ORTHOTOPE1')]).

ctr_example(
    two_orth_column,
    two_orth_column(
        [[ori-1,siz-3,end-4],[ori-1,siz-1,end-2]],
        [[ori-4,siz-2,end-6],[ori-1,siz-3,end-4]],
        1)).

ctr_typical(two_orth_column,[size('ORTHOTOPE')>1]).

ctr_exchangeable(
    two_orth_column,
    [args([['ORTHOTOPE1','ORTHOTOPE2'],['DIM']])]).
```

```
ctr_graph(
    two_orth_column,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT'(=)>>collection(orthotope1,orthotope2)],
    [orthotope1^key='DIM'#/\
     orthotope1^ori<orthotope2^end#/\
     orthotope2^ori<orthotope1^end#/\
     orthotope1^siz>0#/\
     orthotope2^siz>0#=>
     min(orthotope1^end,orthotope2^end)-
     max(orthotope1^ori,orthotope2^ori)=
     orthotope1^siz#/\
     orthotope1^siz=orthotope2^siz],
    ['NARC'=1],
    []).

ctr_eval(two_orth_column,[reformulation(two_orth_column_r)]).

two_orth_column_r(ORTHOTOPE1,ORTHOTOPE2,DIM) :-
        collection(ORTHOTOPE1,[dvar,dvar_gteq(0),dvar]),
        collection(ORTHOTOPE2,[dvar,dvar_gteq(0),dvar]),
        length(ORTHOTOPE1,DIM1),
        length(ORTHOTOPE2,DIM2),
        DIM1=DIM2,
        check_type(int(1,DIM1),DIM),
        get_attr1(ORTHOTOPE1,ORIS1),
        nth1(DIM,ORIS1,O1),
        get_attr2(ORTHOTOPE1,SIZS1),
        nth1(DIM,SIZS1,S1),
        get_attr3(ORTHOTOPE1,ENDS1),
        nth1(DIM,ENDS1,E1),
        get_attr1(ORTHOTOPE2,ORIS2),
        nth1(DIM,ORIS2,O2),
        get_attr2(ORTHOTOPE2,SIZS2),
        nth1(DIM,SIZS2,S2),
        get_attr3(ORTHOTOPE2,ENDS2),
        nth1(DIM,ENDS2,E2),
        O1#<E2#/\O2#<E1#/\S1#>0#/\S2#>0#=>
        min(E1,E2)-max(O1,O2)#=S1#/\S1#=S2.
```

B.418 two_orth_do_not_overlap

◊ **META-DATA:**

```
ctr_date(
    two_orth_do_not_overlap,
    ['20030820','20040530','20060819']).

ctr_origin(
    two_orth_do_not_overlap,
    Used for defining %c.,
    [diffn]).

ctr_types(
    two_orth_do_not_overlap,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    two_orth_do_not_overlap,
    ['ORTHOTOPE1'-'ORTHOTOPE','ORTHOTOPE2'-'ORTHOTOPE']).

ctr_restrictions(
    two_orth_do_not_overlap,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2')]).

ctr_example(
    two_orth_do_not_overlap,
    two_orth_do_not_overlap(
        [[ori-2,siz-2,end-4],[ori-1,siz-3,end-4]],
        [[ori-4,siz-4,end-8],[ori-3,siz-3,end-6]])).

ctr_typical(two_orth_do_not_overlap,[size('ORTHOTOPE')>1]).

ctr_exchangeable(
    two_orth_do_not_overlap,
    [args([['ORTHOTOPE1','ORTHOTOPE2']]),
     items_sync('ORTHOTOPE1','ORTHOTOPE2',all),
     vals(['ORTHOTOPE1'^siz],int(>=(0)),>,dontcare,dontcare),
     vals(['ORTHOTOPE2'^siz],int(>=(0)),>,dontcare,dontcare)]).
```

```
ctr_graph(
    two_orth_do_not_overlap,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['SYMMETRIC_PRODUCT'(=)>>
     collection(orthotope1,orthotope2)],
    [orthotope1^end=<orthotope2^ori#\/orthotope1^siz=0],
    ['NARC'>=1],
    ['BIPARTITE','NO_LOOP']).

ctr_eval(
    two_orth_do_not_overlap,
    [automaton(two_orth_do_not_overlap_a)]).

two_orth_do_not_overlap_a(FLAG,ORTHOTOPE1,ORTHOTOPE2) :-
        length(ORTHOTOPE1,D1),
        length(ORTHOTOPE2,D2),
        D1>0,
        D2>0,
        D1=D2,
        collection(ORTHOTOPE1,[dvar,dvar_gteq(0),dvar]),
        collection(ORTHOTOPE2,[dvar,dvar_gteq(0),dvar]),
        get_attr1(ORTHOTOPE1,ORIS1),
        get_attr3(ORTHOTOPE1,ENDS1),
        check_lesseq(ORIS1,ENDS1),
        get_attr1(ORTHOTOPE2,ORIS2),
        get_attr3(ORTHOTOPE2,ENDS2),
        check_lesseq(ORIS2,ENDS2),
        eval(orth_link_ori_siz_end(ORTHOTOPE1)),
        eval(orth_link_ori_siz_end(ORTHOTOPE2)),
        two_orth_do_not_overlap_signature(
            ORTHOTOPE1,
            ORTHOTOPE2,
            SIGNATURE),
        AUTOMATON=
        automaton(
            SIGNATURE,
            _46582,
            SIGNATURE,
            [source(s),sink(t)],
            [arc(s,1,s),arc(s,0,t),arc(t,0,t),arc(t,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

```
two_orth_do_not_overlap_signature([],[],[]).

two_orth_do_not_overlap_signature(
    [[ori-ORI1,siz-SIZ1,end-END1]|Q1],
    [[ori-ORI2,siz-SIZ2,end-END2]|Q2],
    [S|Ss]) :-
        SIZ1#>0#/\SIZ2#>0#/\END1#>ORI2#/\END2#>ORI1#<=>S,
        two_orth_do_not_overlap_signature(Q1,Q2,Ss).
```

B.419 two_orth_include

◇ **META-DATA:**

```
ctr_date(two_orth_include,['20030820','20090524']).

ctr_origin(
    two_orth_include,
    Used for defining %c.,
    [diffn_include]).

ctr_types(
    two_orth_include,
    ['ORTHOTOPE'-collection(ori-dvar,siz-dvar,end-dvar)]).

ctr_arguments(
    two_orth_include,
    ['ORTHOTOPE1'-'ORTHOTOPE',
     'ORTHOTOPE2'-'ORTHOTOPE',
     'DIM'-int]).

ctr_restrictions(
    two_orth_include,
    [size('ORTHOTOPE')>0,
     require_at_least(2,'ORTHOTOPE',[ori,siz,end]),
     'ORTHOTOPE'^siz>=0,
     'ORTHOTOPE'^ori=<'ORTHOTOPE'^end,
     size('ORTHOTOPE1')=size('ORTHOTOPE2'),
     orth_link_ori_siz_end('ORTHOTOPE1'),
     orth_link_ori_siz_end('ORTHOTOPE2'),
     'DIM'>0,
     'DIM'=<size('ORTHOTOPE1')]).

ctr_example(
    two_orth_include,
    two_orth_include(
        [[ori-1,siz-3,end-4],[ori-1,siz-1,end-2]],
        [[ori-1,siz-2,end-3],[ori-2,siz-3,end-5]],
        1)).

ctr_typical(two_orth_include,[size('ORTHOTOPE')>1]).

ctr_exchangeable(
    two_orth_include,
    [args([['ORTHOTOPE1','ORTHOTOPE2'],['DIM']])]).
```

```
ctr_graph(
    two_orth_include,
    ['ORTHOTOPE1','ORTHOTOPE2'],
    2,
    ['PRODUCT'(=)>>collection(orthotope1,orthotope2)],
    [orthotope1^key='DIM'#/\
     orthotope1^ori<orthotope2^end#/\
     orthotope2^ori<orthotope1^end#/\
     orthotope1^siz>0#/\
     orthotope2^siz>0#=>
     min(orthotope1^end,orthotope2^end)-
     max(orthotope1^ori,orthotope2^ori)=
     min(orthotope1^siz,orthotope2^siz)],
    ['NARC'=1],
    []).

ctr_eval(two_orth_include,[reformulation(two_orth_include_r)]).

two_orth_include_r(ORTHOTOPE1,ORTHOTOPE2,DIM) :-
        collection(ORTHOTOPE1,[dvar,dvar_gteq(0),dvar]),
        collection(ORTHOTOPE2,[dvar,dvar_gteq(0),dvar]),
        length(ORTHOTOPE1,DIM1),
        length(ORTHOTOPE2,DIM2),
        DIM1=DIM2,
        check_type(int(1,DIM1),DIM),
        get_attr1(ORTHOTOPE1,ORIS1),
        nth1(DIM,ORIS1,O1),
        get_attr2(ORTHOTOPE1,SIZS1),
        nth1(DIM,SIZS1,S1),
        get_attr3(ORTHOTOPE1,ENDS1),
        nth1(DIM,ENDS1,E1),
        get_attr1(ORTHOTOPE2,ORIS2),
        nth1(DIM,ORIS2,O2),
        get_attr2(ORTHOTOPE2,SIZS2),
        nth1(DIM,SIZS2,S2),
        get_attr3(ORTHOTOPE2,ENDS2),
        nth1(DIM,ENDS2,E2),
        O1#<E2#/\O2#<E1#/\S1#>0#/\S2#>0#=>
        min(E1,E2)-max(O1,O2)#=min(S1,S2).
```

B.420 used_by

◊ **META-DATA:**

```
ctr_date(used_by,['20000128','20030820','20040530','20060820']).

ctr_origin(used_by,'N.~Beldiceanu',[]).

ctr_arguments(
    used_by,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    used_by,
    [size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    used_by,
    used_by(
        [[var-1],[var-9],[var-1],[var-5],[var-2],[var-1]],
        [[var-1],[var-1],[var-2],[var-5]])).

ctr_typical(
    used_by,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1]).

ctr_exchangeable(
    used_by,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
         =\=,
         all,
         dontcare)]).

ctr_graph(
    used_by,
    ['VARIABLES1','VARIABLES2'],
```

```
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    [for_all('CC','NSOURCE'>='NSINK'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_eval(used_by,[reformulation(used_by_r)]).

ctr_contractible(used_by,[],'VARIABLES2',any).

ctr_extensible(used_by,[],'VARIABLES1',any).

ctr_aggregate(used_by,[],[union,union]).

used_by_r(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1>=N2,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        used_by_reified(VARS2,VARS1,VARS2).
```

B.421 used_by_interval

◇ **META-DATA:**

```
ctr_date(used_by_interval,['20030820','20060820']).

ctr_origin(used_by_interval,'Derived from %c.',[used_by]).

ctr_arguments(
    used_by_interval,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'SIZE_INTERVAL'-int]).

ctr_restrictions(
    used_by_interval,
    [size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'SIZE_INTERVAL'>0]).

ctr_example(
    used_by_interval,
    used_by_interval(
        [[var-1],[var-9],[var-1],[var-8],[var-6],[var-2]],
        [[var-1],[var-0],[var-7],[var-7]],
        3)).

ctr_typical(
    used_by_interval,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1,
     'SIZE_INTERVAL'>1,
     'SIZE_INTERVAL'<range('VARIABLES1'^var),
     'SIZE_INTERVAL'<range('VARIABLES2'^var)]).

ctr_exchangeable(
    used_by_interval,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var],
         intervals('SIZE_INTERVAL'),
         =,
```

```
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'^var],
         intervals('SIZE_INTERVAL'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    used_by_interval,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var/'SIZE_INTERVAL'=
     variables2^var/'SIZE_INTERVAL'],
    [for_all('CC','NSOURCE'>='NSINK'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_eval(used_by_interval,[reformulation(used_by_interval_r)]).

ctr_contractible(used_by_interval,[],'VARIABLES2',any).

ctr_extensible(used_by_interval,[],'VARIABLES1',any).

ctr_aggregate(used_by_interval,[],[union,union,id]).

used_by_interval_r(VARIABLES1,VARIABLES2,SIZE_INTERVAL) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1>=N2,
        integer(SIZE_INTERVAL),
        SIZE_INTERVAL>0,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        gen_quotient(VARS1,SIZE_INTERVAL,QUOTVARS1),
        gen_quotient(VARS2,SIZE_INTERVAL,QUOTVARS2),
        used_by_reified(QUOTVARS2,QUOTVARS1,QUOTVARS2).
```

B.422 used_by_modulo

◇ **META-DATA:**

```
ctr_date(used_by_modulo,['20030820','20060820']).

ctr_origin(used_by_modulo,'Derived from %c.',[used_by]).

ctr_arguments(
    used_by_modulo,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'M'-int]).

ctr_restrictions(
    used_by_modulo,
    [size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     'M'>0]).

ctr_example(
    used_by_modulo,
    used_by_modulo(
        [[var-1],[var-9],[var-4],[var-5],[var-2],[var-1]],
        [[var-7],[var-1],[var-2],[var-5]],
        3)).

ctr_typical(
    used_by_modulo,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1,
     'M'>1,
     'M'<maxval('VARIABLES1'^var),
     'M'<maxval('VARIABLES2'^var)]).

ctr_exchangeable(
    used_by_modulo,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(['VARIABLES1'^var],mod('M'),=,dontcare,dontcare),
     vals(['VARIABLES2'^var],mod('M'),=,dontcare,dontcare)]).

ctr_graph(
```

```
    used_by_modulo,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var mod 'M'=variables2^var mod 'M'],
    [for_all('CC','NSOURCE'>='NSINK'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_eval(used_by_modulo,[reformulation(used_by_modulo_r)]).

ctr_contractible(used_by_modulo,[],'VARIABLES2',any).

ctr_extensible(used_by_modulo,[],'VARIABLES1',any).

ctr_aggregate(used_by_modulo,[],[union,union,id]).

used_by_modulo_r(VARIABLES1,VARIABLES2,M) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,N1),
        length(VARIABLES2,N2),
        N1>=N2,
        integer(M),
        M>0,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        gen_remainder(VARS1,M,REMVARS1),
        gen_remainder(VARS2,M,REMVARS2),
        used_by_reified(REMVARS2,REMVARS1,REMVARS2).
```

B.423 used_by_partition

◊ **META-DATA:**

```
ctr_date(used_by_partition,['20030820','20060820']).

ctr_origin(used_by_partition,'Derived from %c.',[used_by]).

ctr_types(used_by_partition,['VALUES'-collection(val-int)]).

ctr_arguments(
    used_by_partition,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar),
     'PARTITIONS'-collection(p-'VALUES')]).

ctr_restrictions(
    used_by_partition,
    [size('VALUES')>=1,
     required('VALUES',val),
     distinct('VALUES',val),
     size('VARIABLES1')>=size('VARIABLES2'),
     required('VARIABLES1',var),
     required('VARIABLES2',var),
     required('PARTITIONS',p),
     size('PARTITIONS')>=2]).

ctr_example(
    used_by_partition,
    used_by_partition(
        [[var-1],[var-9],[var-1],[var-6],[var-2],[var-3]],
        [[var-1],[var-3],[var-6],[var-6]],
        [[p-[[val-1],[val-3]]],
         [p-[[val-4]]],
         [p-[[val-2],[val-6]]]])).

ctr_typical(
    used_by_partition,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1,
     size('VARIABLES1')>size('PARTITIONS'),
     size('VARIABLES2')>size('PARTITIONS')]).

ctr_exchangeable(
```

```
    used_by_partition,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     items('PARTITIONS',all),
     items('PARTITIONS'^p,all),
     vals(
         ['VARIABLES1'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare),
     vals(
         ['VARIABLES2'^var],
         part('PARTITIONS'),
         =,
         dontcare,
         dontcare)]).

ctr_graph(
    used_by_partition,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [in_same_partition(
         variables1^var,
         variables2^var,
         PARTITIONS)],
    [for_all('CC','NSOURCE'>='NSINK'),
     'NSINK'=size('VARIABLES2')],
    []).

ctr_eval(
    used_by_partition,
    [reformulation(used_by_partition_r)]).

ctr_aggregate(used_by_partition,[],[union,union,id]).

ctr_contractible(used_by_partition,[],'VARIABLES2',any).

ctr_extensible(used_by_partition,[],'VARIABLES1',any).

used_by_partition_r(VARIABLES1,VARIABLES2,PARTITIONS) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        collection(PARTITIONS,[col_len_gteq(1,[int])]),
        length(VARIABLES1,N1),
```

```
length(VARIABLES2,N2),
N1>=N2,
get_attr1(VARIABLES1,VARS1),
get_attr1(VARIABLES2,VARS2),
get_col_attr1(PARTITIONS,1,PVALS),
flattern(PVALS,VALS),
all_different(VALS),
length(PARTITIONS,P),
P>1,
length(PVALS,LPVALS),
LPVALS1 is LPVALS+1,
get_partition_var(VARS1,PVALS,PVARS1,LPVALS1,0),
get_partition_var(VARS2,PVALS,PVARS2,LPVALS1,0),
used_by_reified(PVARS2,PVARS1,PVARS2).
```

B.424 uses

♢ META-DATA:

```
ctr_date(uses,['20050917','20060820']).

ctr_origin(
    uses,
    \cite{BessiereHebrardHnichKiziltanWalsh05IJCAI},
    []).

ctr_arguments(
    uses,
    ['VARIABLES1'-collection(var-dvar),
     'VARIABLES2'-collection(var-dvar)]).

ctr_restrictions(
    uses,
    [min(1,size('VARIABLES1'))>=min(1,size('VARIABLES2')),
     required('VARIABLES1',var),
     required('VARIABLES2',var)]).

ctr_example(
    uses,
    uses(
        [[var-3],[var-3],[var-4],[var-6]],
        [[var-3],[var-4],[var-4],[var-4],[var-4]])).

ctr_typical(
    uses,
    [size('VARIABLES1')>1,
     range('VARIABLES1'^var)>1,
     size('VARIABLES2')>1,
     range('VARIABLES2'^var)>1,
     size('VARIABLES1')=<size('VARIABLES2')]).

ctr_exchangeable(
    uses,
    [items('VARIABLES1',all),
     items('VARIABLES2',all),
     vals(
         ['VARIABLES1'^var,'VARIABLES2'^var],
         int,
         =\=,
         all,
         dontcare)]).
```

```
ctr_graph(
    uses,
    ['VARIABLES1','VARIABLES2'],
    2,
    ['PRODUCT'>>collection(variables1,variables2)],
    [variables1^var=variables2^var],
    ['NSINK'=size('VARIABLES2')],
    ['ACYCLIC','BIPARTITE','NO_LOOP']).

ctr_eval(uses,[reformulation(uses_r)]).

ctr_contractible(uses,[],'VARIABLES2',any).

ctr_extensible(uses,[],'VARIABLES1',any).

ctr_aggregate(uses,[],[union,union]).

uses_r(VARIABLES1,VARIABLES2) :-
        collection(VARIABLES1,[dvar]),
        collection(VARIABLES2,[dvar]),
        length(VARIABLES1,L1),
        length(VARIABLES2,L2),
        M1 is min(1,L1),
        M2 is min(1,L2),
        M1>=M2,
        get_attr1(VARIABLES1,VARS1),
        get_attr1(VARIABLES2,VARS2),
        uses1(VARS2,VARS1).

uses1([],_47176).

uses1([VAR2|R],VARS1) :-
        uses2(VARS1,VAR2,TERM),
        call(TERM),
        uses1(R,VARS1).

uses2([],_47176,0).

uses2([VAR1|R],VAR2,VAR2#=VAR1#\/S) :-
        uses2(R,VAR2,S).
```

B.425 valley

◇ META-DATA:

```
ctr_date(valley,['20040530']).

ctr_origin(valley,'Derived from %c.',[inflexion]).

ctr_arguments(
    valley,
    ['N'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    valley,
    ['N'>=0,
     2*'N'=<max(size('VARIABLES')-1,0),
     required('VARIABLES',var)]).

ctr_example(
    valley,
    [valley(
         1,
         [[var-1],
          [var-1],
          [var-4],
          [var-8],
          [var-8],
          [var-2],
          [var-7],
          [var-1]]),
     valley(
         0,
         [[var-1],
          [var-1],
          [var-4],
          [var-5],
          [var-8],
          [var-8],
          [var-4],
          [var-1]]),
     valley(
         4,
         [[var-1],
          [var-0],
          [var-4],
          [var-0],
```

```
          [var-8],
          [var-2],
          [var-4],
          [var-1],
          [var-2]])]).

ctr_typical(
    valley,
    [size('VARIABLES')>2,range('VARIABLES'^var)>1]).

ctr_typical_model(valley,[nval('VARIABLES'^var)>2]).

ctr_exchangeable(
    valley,
    [items('VARIABLES',reverse),translate(['VARIABLES'^var])]).

ctr_eval(
    valley,
    [checker(valley_c),
     automaton(valley_a),
     automaton_with_signature(valley_a_s)]).

ctr_pure_functional_dependency(valley,[]).

ctr_functional_dependency(valley,1,[2]).

ctr_contractible(valley,['N'=0],'VARIABLES',any).

ctr_cond_imply(
    valley,
    atleast_nvalue,
    ['N'>0],
    ['NVAL'=2],
    [none,'VARIABLES']).

ctr_cond_imply(
    valley,
    inflexion,
    [],
    ['N'=peak('VARIABLES'^var)+valley('VARIABLES'^var)],
    [none,'VARIABLES']).

ctr_sol(valley,2,0,2,9,[0-9]).

ctr_sol(valley,3,0,3,64,[0-50,1-14]).
```

```
ctr_sol(valley,4,0,4,625,[0-295,1-330]).

ctr_sol(valley,5,0,5,7776,[0-1792,1-5313,2-671]).

ctr_sol(valley,6,0,6,117649,[0-11088,1-73528,2-33033]).

ctr_sol(
    valley,
    7,
    0,
    7,
    2097152,
    [0-69498,1-944430,2-1010922,3-72302]).

ctr_sol(
    valley,
    8,
    0,
    8,
    43046721,
    [0-439791,1-11654622,2-24895038,3-6057270]).

valley_c(N,VARIABLES) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[int]),
        get_attr1(VARIABLES,VARS),
        valley_c(VARS,s,0,N).

valley_c([V1,V2|R],s,C,N) :-
        V1=<V2,
        !,
        valley_c([V2|R],s,C,N).

valley_c([_V1,V2|R],s,C,N) :-
        !,
        valley_c([V2|R],u,C,N).

valley_c([V1,V2|R],u,C,N) :-
        V1>=V2,
        !,
        valley_c([V2|R],u,C,N).

valley_c([_V1,V2|R],u,C,N) :-
        !,
        C1 is C+1,
        valley_c([V2|R],s,C1,N).
```

```
valley_c([_53210],_53207,N,N) :-
        !.

valley_c([],_53204,N,N).

valley_counters_check([V1,V2|R],s,C,[C|S]) :-
        V1=<V2,
        !,
        valley_counters_check([V2|R],s,C,S).

valley_counters_check([_V1,V2|R],s,C,[C|S]) :-
        !,
        valley_counters_check([V2|R],u,C,S).

valley_counters_check([V1,V2|R],u,C,[C|S]) :-
        V1>=V2,
        !,
        valley_counters_check([V2|R],u,C,S).

valley_counters_check([_V1,V2|R],u,C,[C1|S]) :-
        !,
        C1 is C+1,
        valley_counters_check([V2|R],s,C1,S).

valley_counters_check([V|R],init,C,[0|S]) :-
        !,
        valley_counters_check([V|R],s,C,S).

valley_counters_check([_53207],_53204,_53205,[]).

ctr_automaton_signature(
    valley,
    valley_a,
    pair_signature(2,signature)).

valley_a(FLAG,N,VARIABLES) :-
        pair_signature(VARIABLES,SIGNATURE),
        valley_a_s(FLAG,N,VARIABLES,SIGNATURE).

valley_a_s(FLAG,N,VARIABLES,SIGNATURE) :-
        check_type(dvar_gteq(0),N),
        collection(VARIABLES,[dvar]),
        length(VARIABLES,L),
        MAX is max(L-1,0),
        2*N#=<MAX,
```

```
automaton(
    SIGNATURE,
    _54835,
    SIGNATURE,
    [source(s),sink(u),sink(s)],
    [arc(s,0,s),
     arc(s,1,s),
     arc(s,2,u),
     arc(u,0,s,[C+1]),
     arc(u,1,u),
     arc(u,2,u)],
    [C],
    [0],
    [COUNT]),
COUNT#=N#<=>FLAG.
```

B.426 vec_eq_tuple

◇ **META-DATA:**

```
ctr_date(vec_eq_tuple,['20030820','20060820']).

ctr_origin(vec_eq_tuple,'Used for defining %c.',[in_relation]).

ctr_arguments(
    vec_eq_tuple,
    ['VARIABLES'-collection(var-dvar),
     'TUPLE'-collection(val-int)]).

ctr_restrictions(
    vec_eq_tuple,
    [required('VARIABLES',var),
     required('TUPLE',val),
     size('VARIABLES')=size('TUPLE')]).

ctr_example(
    vec_eq_tuple,
    vec_eq_tuple(
        [[var-5],[var-3],[var-3]],
        [[val-5],[val-3],[val-3]])).

ctr_typical(
    vec_eq_tuple,
    [size('VARIABLES')>1,
     range('VARIABLES'^var)>1,
     range('TUPLE'^val)>1]).

ctr_exchangeable(
    vec_eq_tuple,
    [args([['VARIABLES','TUPLE']]),
     items_sync('VARIABLES','TUPLE',all)]).

ctr_graph(
    vec_eq_tuple,
    ['VARIABLES','TUPLE'],
    2,
    ['PRODUCT'(=)>>collection(variables,tuple)],
    [variables^var=tuple^val],
    ['NARC'=size('VARIABLES')],
    []).

ctr_eval(vec_eq_tuple,[reformulation(vec_eq_tuple_r)]).
```

```
ctr_contractible(vec_eq_tuple,[],['VARIABLES','TUPLE'],any).

vec_eq_tuple_r(VARIABLES,TUPLE) :-
        collection(VARIABLES,[dvar]),
        collection(TUPLE,[int]),
        length(VARIABLES,N),
        length(TUPLE,M),
        N=M,
        get_attr1(VARIABLES,VARS),
        get_attr1(TUPLE,VALS),
        vec_eq_tuple1(VARS,VALS).

vec_eq_tuple1([],[]).

vec_eq_tuple1([VAR|R],[VAL|S]) :-
        VAR#=VAL,
        vec_eq_tuple1(R,S).
```

B.427 visible

◇ **META-DATA:**

```
ctr_predefined(visible).

ctr_date(visible,['20071013']).

ctr_origin(
    visible,
    Extension of \emph{accessibility} parameter of %c.,
    [diffn]).

ctr_types(
    visible,
    ['VARIABLES'-collection(v-dvar),
     'INTEGERS'-collection(v-int),
     'POSITIVES'-collection(v-int),
     'DIMDIR'-collection(dim-int,dir-int)]).

ctr_arguments(
    visible,
    ['K'-int,
     'DIMS'-sint,
     'FROM'-'DIMDIR',
     OBJECTS-
     collection(
         oid-int,
         sid-dvar,
         x-'VARIABLES',
         start-dvar,
         duration-dvar,
         end-dvar),
     SBOXES-
     collection(
         sid-int,
         t-'INTEGERS',
         l-'POSITIVES',
         f-'DIMDIR')]).

ctr_restrictions(
    visible,
    [size('VARIABLES')>=1,
     size('INTEGERS')>=1,
     size('POSITIVES')>=1,
     required('VARIABLES',v),
```

```
     size('VARIABLES')='K',
     required('INTEGERS',v),
     size('INTEGERS')='K',
     required('POSITIVES',v),
     size('POSITIVES')='K',
     'POSITIVES'^v>0,
     required('DIMDIR',[dim,dir]),
     size('DIMDIR')>0,
     size('DIMDIR')=<'K'+'K',
     distinct('DIMDIR',[]),
     'DIMDIR'^dim>=0,
     'DIMDIR'^dim<'K',
     'DIMDIR'^dir>=0,
     'DIMDIR'^dir=<1,
     'K'>=0,
     'DIMS'>=0,
     'DIMS'<'K',
     distinct('OBJECTS',oid),
     required('OBJECTS',[oid,sid,x]),
     require_at_least(2,'OBJECTS',[start,duration,end]),
     'OBJECTS'^oid>=1,
     'OBJECTS'^oid=<size('OBJECTS'),
     'OBJECTS'^sid>=1,
     'OBJECTS'^sid=<size('SBOXES'),
     'OBJECTS'^duration>=0,
     size('SBOXES')>=1,
     required('SBOXES',[sid,t,l]),
     'SBOXES'^sid>=1,
     'SBOXES'^sid=<size('SBOXES'),
     do_not_overlap('SBOXES')]).

ctr_example(
    visible,
    [visible(
         2,
         {0,1},
         [[dim-0,dir-1]],
         [[oid-1,
           sid-1,
           x-[[v-1],[v-2]],
           start-8,
           duration-8,
           end-16],
          [oid-2,
           sid-2,
           x-[[v-4],[v-2]],
```

```
      start-1,
      duration-15,
      end-16]],
    [[sid-1,
      t-[[v-0],[v-0]],
      l-[[v-1],[v-2]],
      f-[[dim-0,dir-1]]],
     [sid-2,
      t-[[v-0],[v-0]],
      l-[[v-2],[v-3]],
      f-[[dim-0,dir-1]]]]),
visible(
    2,
    {0,1},
    [[dim-0,dir-1]],
    [[oid-1,
      sid-1,
      x-[[v-1],[v-2]],
      start-1,
      duration-8,
      end-9],
     [oid-2,
      sid-2,
      x-[[v-4],[v-2]],
      start-1,
      duration-15,
      end-16]],
    [[sid-1,
      t-[[v-0],[v-0]],
      l-[[v-1],[v-2]],
      f-[[dim-0,dir-1]]],
     [sid-2,
      t-[[v-0],[v-0]],
      l-[[v-2],[v-3]],
      f-[[dim-0,dir-1]]]]),
visible(
    2,
    {0,1},
    [[dim-0,dir-1]],
    [[oid-1,
      sid-1,
      x-[[v-1],[v-1]],
      start-1,
      duration-15,
      end-16],
     [oid-2,
```

```
       sid-2,
       x-[[v-2],[v-2]],
       start-6,
       duration-6,
       end-12]],
     [[sid-1,
       t-[[v-0],[v-0]],
       l-[[v-1],[v-2]],
       f-[[dim-0,dir-1]]],
      [sid-2,
       t-[[v-0],[v-0]],
       l-[[v-2],[v-3]],
       f-[[dim-0,dir-1]]]]),
visible(
     2,
     {0,1},
     [[dim-0,dir-1]],
     [[oid-1,
       sid-1,
       x-[[v-4],[v-1]],
       start-1,
       duration-8,
       end-9],
      [oid-2,
       sid-2,
       x-[[v-1],[v-2]],
       start-1,
       duration-15,
       end-16]],
     [[sid-1,
       t-[[v-0],[v-0]],
       l-[[v-1],[v-2]],
       f-[[dim-0,dir-1]]],
      [sid-2,
       t-[[v-0],[v-0]],
       l-[[v-2],[v-3]],
       f-[[dim-0,dir-1]]]]),
visible(
     2,
     {0},
     [[dim-0,dir-1]],
     [[oid-1,
       sid-1,
       x-[[v-2],[v-1]],
       start-1,
       duration-8,
```

```
           end-9],
          [oid-2,
           sid-2,
           x-[[v-4],[v-3]],
           start-1,
           duration-15,
           end-16]],
         [[sid-1,
           t-[[v-0],[v-0]],
           l-[[v-1],[v-2]],
           f-[[dim-0,dir-1]]],
          [sid-2,
           t-[[v-0],[v-0]],
           l-[[v-2],[v-2]],
           f-[[dim-0,dir-1]]]])]).

ctr_typical(visible,[size('OBJECTS')>1]).

ctr_exchangeable(
    visible,
    [items('OBJECTS',all),items('SBOXES',all)]).

ctr_application(visible,[4]).
```

B.428 weighted_partial_alldiff

◇ META-DATA:

```
ctr_date(
    weighted_partial_alldiff,
    ['20040814','20060820','20090503']).

ctr_origin(
    weighted_partial_alldiff,
    \cite[page 71]{Thiel04},
    []).

ctr_synonyms(
    weighted_partial_alldiff,
    [weighted_partial_alldifferent,
     weighted_partial_alldistinct,
     wpa]).

ctr_arguments(
    weighted_partial_alldiff,
    ['VARIABLES'-collection(var-dvar),
     'UNDEFINED'-int,
     'VALUES'-collection(val-int,weight-int),
     'COST'-dvar]).

ctr_restrictions(
    weighted_partial_alldiff,
    [required('VARIABLES',var),
     size('VALUES')>0,
     required('VALUES',[val,weight]),
     in_attr('VARIABLES',var,'VALUES',val),
     distinct('VALUES',val)]).

ctr_example(
    weighted_partial_alldiff,
    weighted_partial_alldiff(
        [[var-4],[var-0],[var-1],[var-2],[var-0],[var-0]],
        0,
        [[val-0,weight-0],
         [val-1,weight-2],
         [val-2,weight- -1],
         [val-4,weight-7],
         [val-5,weight- -8],
         [val-6,weight-2]],
        8)).
```

```
ctr_typical(
    weighted_partial_alldiff,
    [size('VARIABLES')>0,
     atleast(1,'VARIABLES','UNDEFINED'),
     size('VARIABLES')=<size('VALUES')+2]).

ctr_exchangeable(
    weighted_partial_alldiff,
    [items('VARIABLES',all),
     items('VALUES',all),
     vals(
         ['VARIABLES'^var,'VALUES'^val],
         int(=\=('UNDEFINED')),
         =\=,
         all,
         dontcare)]).

ctr_graph(
    weighted_partial_alldiff,
    ['VARIABLES','VALUES'],
    2,
    ['PRODUCT'>>collection(variables,values)],
    [variables^var=\='UNDEFINED',variables^var=values^val],
    ['MAX_ID'=<1,'SUM'('VALUES',weight)='COST'],
    []).

ctr_eval(
    weighted_partial_alldiff,
    [reformulation(weighted_partial_alldiff_r)]).

ctr_functional_dependency(weighted_partial_alldiff,4,[1,3]).

weighted_partial_alldiff_r(VARIABLES,UNDEFINED,VALUES,COST) :-
        collection(VARIABLES,[dvar]),
        integer(UNDEFINED),
        collection(VALUES,[int,int]),
        length(VALUES,N),
        N>0,
        check_type(dvar,COST),
        get_attr1(VARIABLES,VARS),
        get_attr1(VALUES,VALS),
        get_attr2(VALUES,WEIGHTS),
        all_different(VALS),
        get_proj1(VALUES,CVALS),
        weighted_partial_alldiff0(VALS,WEIGHTS,UNDEFINED),
```

```
        weighted_partial_alldiff1(VARS,CVALS),
        weighted_partial_alldiff2(VARS,UNDEFINED),
        weighted_partial_alldiff4(VALS,WEIGHTS,VARS,TERM),
        call(COST#=TERM).

weighted_partial_alldiff0(
    [UNDEFINED|_54795],
    [0|_54799],
    UNDEFINED) :-
        !.

weighted_partial_alldiff0([_V|R],[_W|S],UNDEFINED) :-
        weighted_partial_alldiff0(R,S,UNDEFINED).

weighted_partial_alldiff1([],_54789).

weighted_partial_alldiff1([VAR|R],VALUES) :-
        eval(VAR in VALUES),
        weighted_partial_alldiff1(R,VALUES).

weighted_partial_alldiff2([],_54789).

weighted_partial_alldiff2([_54793],_54792) :-
        !.

weighted_partial_alldiff2([VAR|R],UNDEFINED) :-
        weighted_partial_alldiff3(R,VAR,UNDEFINED),
        weighted_partial_alldiff2(R,UNDEFINED).

weighted_partial_alldiff3([],_54789,_54790).

weighted_partial_alldiff3([UAR|R],VAR,UNDEFINED) :-
        UAR#\=VAR#\/UAR#=UNDEFINED,
        weighted_partial_alldiff3(R,VAR,UNDEFINED).

weighted_partial_alldiff4([],[],_54790,0).

weighted_partial_alldiff4([VAL|R],[WEIGHT|S],VARS,WEIGHT*B+T) :-
        weighted_partial_alldiff5(VARS,VAL,WEIGHT,TERM),
        call(B#<=>TERM),
        weighted_partial_alldiff4(R,S,VARS,T).

weighted_partial_alldiff5([],_54789,_54790,0).

weighted_partial_alldiff5([VAR|R],VAL,WEIGHT,VAR#=VAL#\/T) :-
        weighted_partial_alldiff5(R,VAL,WEIGHT,T).
```

B.429 xor

◇ META-DATA:

```
ctr_date(xor,['20051226']).

ctr_origin(xor,'Logic',[]).

ctr_synonyms(xor,[odd,rel]).

ctr_arguments(
    xor,
    ['VAR'-dvar,'VARIABLES'-collection(var-dvar)]).

ctr_restrictions(
    xor,
    ['VAR'>=0,
     'VAR'=<1,
     size('VARIABLES')=2,
     required('VARIABLES',var),
     'VARIABLES'^var>=0,
     'VARIABLES'^var=<1]).

ctr_example(
    xor,
    [xor(0,[[var-0],[var-0]]),
     xor(1,[[var-0],[var-1]]),
     xor(1,[[var-1],[var-0]]),
     xor(0,[[var-1],[var-1]])]).

ctr_exchangeable(xor,[items('VARIABLES',all)]).

ctr_eval(xor,[automaton(xor_a)]).

ctr_pure_functional_dependency(xor,[]).

ctr_functional_dependency(xor,1,[2]).

ctr_sol(xor,2,0,2,4,[0-2,1-2]).

ctr_sol(xor,3,0,3,0,[]).

ctr_sol(xor,4,0,4,0,[]).

ctr_sol(xor,5,0,5,0,[]).
```

```
ctr_sol(xor,6,0,6,0,[]).

ctr_sol(xor,7,0,7,0,[]).

ctr_sol(xor,8,0,8,0,[]).

xor_a(FLAG,VAR,VARIABLES) :-
        check_type(dvar(0,1),VAR),
        collection(VARIABLES,[dvar(0,1)]),
        length(VARIABLES,2),
        get_attr1(VARIABLES,LIST),
        append([VAR],LIST,LIST_VARIABLES),
        AUTOMATON=
        automaton(
            LIST_VARIABLES,
            _42654,
            LIST_VARIABLES,
            [source(s),sink(t)],
            [arc(s,0,i),
             arc(s,1,j),
             arc(i,0,k),
             arc(i,1,l),
             arc(j,0,l),
             arc(j,1,k),
             arc(k,0,t),
             arc(l,1,t)],
            [],
            [],
            []),
        automaton_bool(FLAG,[0,1],AUTOMATON).
```

B.430 zero_or_not_zero

◇ META-DATA:

```
ctr_predefined(zero_or_not_zero).

ctr_date(zero_or_not_zero,['20120515']).

ctr_origin(zero_or_not_zero,'Arithmetic.',[]).

ctr_synonyms(
    zero_or_not_zero,
    [zeros_or_not_zeros,not_zero_or_zero,not_zeros_or_zeros]).

ctr_arguments(zero_or_not_zero,['VAR1'-dvar,'VAR2'-dvar]).

ctr_example(zero_or_not_zero,zero_or_not_zero(1,8)).

ctr_exchangeable(zero_or_not_zero,[args([['VAR1','VAR2']])]).

ctr_eval(
    zero_or_not_zero,
    [checker(zero_or_not_zero_c),
     reformulation(zero_or_not_zero_r)]).

zero_or_not_zero_c(0,0) :-
        !.

zero_or_not_zero_c(VAR1,VAR2) :-
        check_type(int,VAR1),
        check_type(int,VAR2),
        VAR1=\=0,
        VAR2=\=0.

zero_or_not_zero_r(VAR1,VAR2) :-
        check_type(dvar,VAR1),
        check_type(dvar,VAR2),
        VAR1#=0#/\VAR2#=0#\/VAR1#\=0#/\VAR2#\=0.
```

B.431 zero_or_not_zero_vectors

◇ **META-DATA:**

```
ctr_predefined(zero_or_not_zero_vectors).

ctr_date(zero_or_not_zero_vectors,['20120516']).

ctr_origin(zero_or_not_zero_vectors,'Tournament scheduling',[]).

ctr_synonyms(
    zero_or_not_zero_vectors,
    [zeros_or_not_zeros_vectors,
     not_zero_or_zero_vectors,
     not_zeros_or_zeros_vectors]).

ctr_types(
    zero_or_not_zero_vectors,
    ['VECTOR'-collection(var-dvar)]).

ctr_arguments(
    zero_or_not_zero_vectors,
    ['VECTORS'-collection(vec-'VECTOR')]).

ctr_restrictions(
    zero_or_not_zero_vectors,
    [size('VECTOR')>=1,
     required('VECTOR',var),
     size('VECTORS')>=1,
     required('VECTORS',vec),
     same_size('VECTORS',vec)]).

ctr_example(
    zero_or_not_zero_vectors,
    zero_or_not_zero_vectors(
        [[vec-[[var-5],[var-6]]],
         [vec-[[var-5],[var-6]]],
         [vec-[[var-0],[var-0]]],
         [vec-[[var-9],[var-3]]],
         [vec-[[var-0],[var-0]]]])).

ctr_typical(
    zero_or_not_zero_vectors,
    [size('VECTOR')>1,size('VECTORS')>1]).

ctr_eval(
```

```
    zero_or_not_zero_vectors,
    [checker(zero_or_not_zero_vectors_c),
     reformulation(zero_or_not_zero_vectors_r),
     density(zero_or_not_zero_vectors_d)]).

ctr_contractible(zero_or_not_zero_vectors,[],'VECTORS',any).

zero_or_not_zero_vectors_c(VECTORS) :-
        VECTORS=[[vec-[[var-_24288]|_24283]]|_24276],
        collection(VECTORS,[col([int])]),
        same_size(VECTORS),
        zero_or_not_zero_vectors_c(VECTORS,1).

zero_or_not_zero_vectors_c([],0) :-
        !.

zero_or_not_zero_vectors_c([[vec-V]|R],F) :-
        (   V=[[var-0]|W] ->
            NextF=0,
            zero_vector(W)
        ;   NextF=F,
            not_zero_vector(W)
        ),
        zero_or_not_zero_vectors_c(R,NextF).

zero_or_not_zero_vectors_r(VECTORS) :-
        VECTORS=[[vec-[[var-_24288]|_24283]]|_24276],
        collection(VECTORS,[col([dvar])]),
        zero_or_not_zero_vectors_r1(VECTORS,AtleastOneZero),
        call(AtleastOneZero).

zero_or_not_zero_vectors_r1([],0) :-
        !.

zero_or_not_zero_vectors_r1([[vec-V]|R],Zero#\/S) :-
        zero_or_not_zero_vectors_r2(V,Zero,NotZero),
        call(Zero#\/NotZero),
        zero_or_not_zero_vectors_r1(R,S).

zero_or_not_zero_vectors_r2([],1,1) :-
        !.

zero_or_not_zero_vectors_r2([[var-V]|R],V#=0#/\S,V#\=0#/\T) :-
        zero_or_not_zero_vectors_r2(R,S,T).

zero_or_not_zero_vectors_d(Density,VECTORS) :-
```

```
count_zeros_in_vectors(VECTORS,0,ZEROS),
VECTORS=[[vec-[V|_24295]]|_24288],
length(VECTORS,N),
length(V,M),
A is N*M,
Density is(A-ZEROS)/A.
```

B.432 Utilities

```
:- use_module(library(lists)).
:- use_module(library(ordsets)).
:- use_module(library(clpfd)).
:- use_module(library(plunit)).
:- use_module(library(trees)).
:- use_module(library(samsort)).

%% to use when everything is not necessarly ground
eval(Ctr) :-
    Ctr =..[Name|Args],
    ctr_eval(Name, Methods),
    (   member(      builtin(Pred), Methods) -> Goal =..[Pred|Args]
    ;   member(    automaton(Pred), Methods) -> Goal =..[Pred,1|Args]
    ;   member(     automata(Pred), Methods) -> Goal =..[Pred|Args]
        % defined by a conjunction of automata, no negation available
    ;   member(reformulation(Pred), Methods) -> Goal =..[Pred|Args]
    ;   member(        logic(Pred), Methods) -> Goal =..[Pred|Args]
    ), !,
    call(Goal).

%% to use when everything is not necessarly ground and when
%% want to share similar signature variables between constraints
%% that have an automaton
eval_with_signature(Ctr) :-
    Ctr =..[Name|Args],
    ctr_eval(Name, Methods),
    (   member(automaton_with_signature(Pred), Methods) ->
        Goal =..[Pred|Args]
    ;   write(no_signature_version(Ctr)),nl,
        abort
    ), !,
    call(Goal).

%% to use when everything is ground: call first a checker
%% if it exist (since normally faster), and then builtin, ...
checker(Ctr) :-
    Ctr =..[Name|Args],
    ctr_eval(Name, Methods),
    (   member(      checker(Pred), Methods) -> Goal =..[Pred|Args]
    ;   member(      builtin(Pred), Methods) -> Goal =..[Pred|Args]
    ;   member(    automaton(Pred), Methods) -> Goal =..[Pred,1|Args]
    ;   member(     automata(Pred), Methods) -> Goal =..[Pred|Args]
        % defined by a conjunction of automata, no negation available
    ;   member(reformulation(Pred), Methods) -> Goal =..[Pred|Args]
```

```
    ;   member(        logic(Pred), Methods) -> Goal =..[Pred|Args]
    ), !,
    statistics(runtime,[Start|_]),
    (   call(Goal) ->
        statistics(runtime,[End|_]),
        Time is End-Start,
        increment(Pred,Time)
    ;   statistics(runtime,[End|_]),
        Time is End-Start,
        increment_fail(Pred,Time),
        fail
    ).


:-dynamic(counter/5).

increment(Pred,Time):-
    (   retract(counter(Pred,N,Old,Fail,FailTime)) ->
        N1 is N+1,
        New is Old+Time,
        asserta(counter(Pred,N1,New,Fail,FailTime))
    ;   asserta(counter(Pred,1,Time,0,0))
    ).

increment_fail(Pred,Time):-
    (   retract(counter(Pred,Succ,SuccTime,N,Old)) ->
        N1 is N+1,
        New is Old+Time,
        asserta(counter(Pred,Succ,SuccTime,N1,New))
    ;   asserta(counter(Pred,0,0,1,Time))
    ).

counter_reset:-
    retract(counter(_,_,_,_,_)),
    fail.
counter_reset.

counter_list(L):-
    findall(counter(A,B,C,D,E),counter(A,B,C,D,E),L).

%% used in checker.pl to check the examples using the evaluator
%% if it exist
eval_or_check(Ctr) :-
    Ctr =..[Name|Args],
    ctr_eval(Name, Methods),
    (   member(      builtin(Pred), Methods) -> Goal =..[Pred|Args]
```

```
    ;   member(    automaton(Pred), Methods) -> Goal =..[Pred,1|Args]
    ;   member(     automata(Pred), Methods) -> Goal =..[Pred|Args]
        % defined by a conjunction of automata, no negation available
    ;   member(reformulation(Pred), Methods) -> Goal =..[Pred|Args]
    ;   member(        logic(Pred), Methods) -> Goal =..[Pred|Args]
    ;   member(      checker(Pred), Methods) -> Goal =..[Pred|Args]
    ), !,
    call(Goal).

%% used to compute the density of a ground satisfied instance
density(Ctr, Density) :-
    Ctr =..[Name|Args],
    ctr_eval(Name, Methods),
    memberchk(density(Pred), Methods),
    Goal =..[Pred,Density|Args],
    call(Goal).

%% to use to evaluate the negation of a constraint, use:
%%  . reified automaton or
%%  . reified constraint for pure functional dependency or
%%  . existing constraint of the catalog with exactly same arguments
neg_eval(Ctr) :-
    Ctr =..[Name|Args],
    ctr_pure_functional_dependency(Name), !,
    NegCtr =..[Name,0|Args],
    reified_ctr_pure_functional_dependency(NegCtr), !.
neg_eval(Ctr) :-
    Ctr =..[Name|Args],
    ctr_eval(Name, Methods),
    (member(automaton(Pred), Methods) -> Goal =..[Pred,0|Args]), !,
    call(Goal), !.
neg_eval(Ctr) :-
    Ctr =..[Name|Args],
    ctr_see_also(Name, Links),
    member(link(negation, NegName, _, _), Links), !,
    NegCtr =..[NegName|Args],
    eval(NegCtr), !.

%% reified version for constraints that can be described
%% in term of pure functional dependency
%% (reified variables put before arguments)
reified_ctr_pure_functional_dependency(Ctr) :-
    Ctr =..[Name,Bool|Args],
    ctr_pure_functional_dependency(Name),
    ctr_arguments(Name, ListArgsCtr),
    findall(F, ctr_functional_dependency(Name,F,_), LF),
```

```
    sort(LF, SLF),
    length(Args, NArgs),
    build_args_ctr(1, NArgs, Args, ListArgsCtr, SLF, NewArgs, AndExpr),
    NewCtr =..[Name|NewArgs],
    eval(NewCtr),
    call(AndExpr #<=> Bool).

build_args_ctr(I, N, [], [], [], [], 1) :-
    I > N, !.
build_args_ctr(I, N, [Arg|RArg], [ArgType|RArgType],
               [F|RF], [Var|R], Var#=Arg #/\ S) :-
    I =< N,
    I = F,
    ArgType = _-dvar, !,
    Var in -1000000..1000000,
    I1 is I+1,
    build_args_ctr(I1, N, RArg, RArgType, RF, R, S).
build_args_ctr(I, N, [Arg|RArg], [_|RArgType], LF, [Arg|R], S) :-
    I =< N, !,
    I1 is I+1,
    build_args_ctr(I1, N, RArg, RArgType, LF, R, S).

%% depending on the flag, call positive automaton,
%% or computes negative automaton and auxiliary constraints
%% and call them
automaton_bool(1, _ALPHABET, POS_AUTOMATON) :- !,
    call(POS_AUTOMATON).
automaton_bool(0, ALPHABET, POS_AUTOMATON) :-
    negaut(POS_AUTOMATON, ALPHABET, NEG_AUTOMATON, NEG_AUXILIARY),
    call(NEG_AUTOMATON),
    call(NEG_AUXILIARY).

%%--------------------------------------------------------------
%% An utility for negating an automaton
%% (WARNING: only valid if everything expressed with ONE SINGLE automaton)
%%
%% negaut(+PosAutomaton, +Alphabet, -NegAutomaton, -AuxConstraint)
%% PosAutomaton: automaton/8 constraint
%% Alphabet: list of atom
%% NegAutomaton: automaton/8 constraint
%% AuxConstraint: constraint
%%
%% Semantics:
%% - (NegAutomaton, AuxConstraint) expresses the negation of PosAutomaton.
%%
%% Synopsis:
```

```
%% - If necessary, add a nonsink state 'fail', and:
%%   * for every letter A of the alphabet: add an arc from 'fail' over
%%     A to 'fail';
%%   * for every state S and letter A of the alphabet, if there is no
%%     outgoing arc from S over A, add an arc from S over A to 'fail'.
%%
%% - If the automaton is counter-free, compute NegAutomaton by swapping
%%   sinks and nonsinks.  AuxConstraint is 'true'.
%%
%% - Otherwise with counters [C1,...,Cn]:
%%   * Suppose that the final counter values are [V1,...,Vn].
%%   * Add a first counter C0 so that C0=0 iff the original automaton
%%     stops in a sink state.
%%   * Convert arcs as follows:
%%       if S2 is sink:
%%        arc(S1,A,S2) --> arc(S1,A,S2,[0,C1,...,Cn])
%%       if S2 is nonsink:
%%        arc(S1,A,S2) --> arc(S1,A,S2,[1,C1,...,Cn])
%%       if S2 is sink:
%%        arc(S1,A,S2,[Y1,...,Yn]) --> arc(S1,A,S2,[0,Y1,...,Yn])
%%       if S2 is nonsink:
%%        arc(S1,A,S2,[Y1,...,Yn]) --> arc(S1,A,S2,[1,Y1,...,Yn])
%%   * The counters for arcs with conditions are augmented similarly.
%%   * For every arc with a condition:
%%        arc(S1,A,_,(P1 -> Q1 ; ... ; Pm -> Qm))
%%     such that (P1 #\/ ... #\/ Pm) could be false, add an arc:
%%        arc(S1,A,fail,((#\P1 #/\ ... #/\ #\Pm) -> [1,C1,...,Cn])
%%   * Conpute NegAutomaton by making all states sinks.
%%   * Let the final counter values of NegAutomaton be [X0,X1,...,Xn].
%%   * AuxConstraint is (X0 #= 1 #\/ X1 #\= V1 #\/ ... #\/ Xn #\= Vn).

negaut(PosAut, Alphabet1, NegAut, Aux) :-
    PosAut = automaton(Args, Arg, Signature,
                       PosSourcesSinks, PosArcs,
                       Counters, Initial, Final),
    NegAut = automaton(Args, Arg, Signature,
                       NegSourcesSinks, NegArcs,
                       NegCounters, NegInitial, NegFinal),
    (   foreach(SS1, PosSourcesSinks),
        fromto(Sources1,Sources1b,Sources1c,[]),
        fromto(Sinks1,Sinks1b,Sinks1c,[])
    do  (   SS1 = source(SS2)
        ->  Sources1b = [SS2|Sources1c], Sinks1b = Sinks1c
        ;   SS1 = sink(SS2)
        ->  Sinks1b = [SS2|Sinks1c], Sources1b = Sources1c
        )
```

```
    ),
    (   foreach(Arc,PosArcs),
        fromto(States1,[S1,S2|States1c],States1c,[])
    do  (   Arc = arc(S1,_,S2) -> true
        ;   Arc = arc(S1,_,S2,_)
        )
    ),
    sort(Alphabet1, Alphabet2),
    sort(Sources1, Sources2),
    sort(Sinks1, Sinks2),
    sort(States1, States2),
    (   foreach(P,Final),
        foreach(N,NegFinalT),
        foreach(N #\= P,NeqsT)
    do  true
    ),
    (   Counters==[] ->
        NegCounters = [],
        NegInitial = [],
        NegFinal = [],
        Aux = true,
        negaut_simple(PosArcs, NegSourcesSinks, NegArcs,
                      Sources2, Sinks2, States2, Alphabet2)
    ;   NegCounters = [_|Counters],
        NegInitial = [0|Initial],
        NegFinal = [FlagT|NegFinalT],
        Neqs = [FlagT #= 1|NeqsT],
        orify(Neqs, Aux),
        negaut_counters(PosArcs, NegSourcesSinks, NegArcs,
                        Sources2, Sinks2, States2, Alphabet2, Counters)
    ).

negaut_simple(PosArcs, NegSourcesSinks, NegArcs,
          Sources1, Sinks1, States1, Alphabet) :-
    ord_subtract(States1, Sinks1, Sinks2),
    (   foreach(S1,Sources1),
        foreach(source(S1),Sources2)
    do  true
    ),
    (   foreach(S2,Sinks2),
        foreach(sink(S2),Sinks3)
    do  true
    ),
    (   foreach(arc(S3,K,_),PosArcs),
        foreach(S3-K,KL1)
    do  true
```

```
    ),
    keysort(KL1, KL2),
    keyclumped(KL2, KL3),
    (   foreach(S4-Set1,KL3),
        fromto(NegArcs4,NegArcs5,NegArcs7,NegArcs8),
        param(Alphabet)
    do  ord_subtract(Alphabet, Set1, CSet1),
        (   foreach(C,CSet1),
            fromto(NegArcs5,[arc(S4,C,fail)|NegArcs6],NegArcs6,NegArcs7),
            param(S4)
        do  true
        )
    ),
    (   NegArcs4 == NegArcs8 ->
        NegArcs8 = [],
        append(Sources2, Sinks3, NegSourcesSinks)
    ;   (   foreach(A,Alphabet),
            foreach(arc(fail,A,fail),NegArcs8)
        do  true
        ),
        append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
    ),
    append(NegArcs4, PosArcs, NegArcs).

negaut_counters(PosArcs1, NegSourcesSinks, NegArcs,
            Sources1, Sinks1, States1, Alphabet, Counters) :-
    (   foreach(S1,Sources1),
        foreach(source(S1),Sources2)
    do  true
    ),
    (   foreach(S2,States1),
        foreach(sink(S2),Sinks3)
    do  true
    ),
    (   foreach(Arc1,PosArcs1),
        foreach(Arc2,PosArcs2),
        fromto(NegArcs1,NegArcs2,NegArcs3,NegArcs4),
        foreach(S3-K,KL1),
        param(Sinks1,Counters)
    do  Arc1 =.. [arc,S3,K|_],
        (ord_member(S3, Sinks1) -> F=0 ; F=1),
        augment_arc(Arc1, F, Counters, Arc2, NegArcs2, NegArcs3)
    ),
    keysort(KL1, KL2),
    keyclumped(KL2, KL3),
    (   foreach(S4-Set1,KL3),
```

```
        fromto(NegArcs4,NegArcs5,NegArcs7,NegArcs8),
        param(Alphabet,Counters)
    do  ord_subtract(Alphabet, Set1, CSet1),
        (   foreach(C,CSet1),
            fromto(NegArcs5,[arc(S4,C,fail,[1|Counters])|NegArcs6],
                   NegArcs6,NegArcs7),
            param(S4,Counters)
        do  true
        )
    ),
    (   NegArcs1 == NegArcs8 ->
        NegArcs8 = [],
        append(Sources2, Sinks3, NegSourcesSinks)
    ;   (   foreach(A,Alphabet),
            foreach(arc(fail,A,fail),NegArcs8)
        do  true
        ),
        append(Sources2, [sink(fail)|Sinks3], NegSourcesSinks)
    ),
    append(NegArcs4, PosArcs2, NegArcs).

augment_arc(arc(S1,K,S2), F, Ctrs, arc(S1,K,S2,[F|Ctrs])) --> [].
augment_arc(arc(S1,K,S2,(P1->Q1 ; P2->Q2)), F, _,
            arc(S1,K,S2,(P1->[F|Q1] ; P2->[F|Q2]))) --> !,
    [].  % assume (P1;P2) is entailed
augment_arc(arc(S1,K,S2,(P1->Q1)), F, Ctrs,
            arc(S1,K,S2,(P1->[F|Q1]))) --> !,
    {neg_arith(P1, P2)},
    [arc(S1,K,fail,(P2->[1|Ctrs]))].
augment_arc(arc(S1,K,S2,Ctrs), F, _,
            arc(S1,K,S2,[F|Ctrs])) --> [].

neg_arith(X #= Y, X #\= Y).
neg_arith(X #\= Y, X #= Y).
neg_arith(X #< Y, X #>= Y).
neg_arith(X #=< Y, X #> Y).
neg_arith(X #> Y, X #=< Y).
neg_arith(X #>= Y, X #< Y).

orify([], true).
orify([X|L], Disj) :- orify(L, X, Disj).

orify([], X, X).
orify([Y|L], X, (X #\/ Disj)) :- orify(L, Y, Disj).
%%-------------------------------------------------------------
```

```
union_dom_list_int(Dvars, Union) :-
    (   foreach(V,Dvars),
        foreach(S,Sets)
    do  fd_set(V, S)
    ),
    fdset_union(Sets, U),
    fdset_to_list(U, Union).

union_dom_set([], []).
union_dom_set([V|R], S) :-
    fd_set(V, SetValuesOfV),
    union_dom_set(R, Set),
    fdset_union(SetValuesOfV, Set, S).

same_size([]).
same_size([[_-L]|R]) :-
    length(L, N),
    same_size(R, N).

same_size([], _).
same_size([[_-L]|R], N) :-
    length(L, N),
    same_size(R, N).

create_pairs([], []) :- !.
create_pairs([V|R], [V-V|S]) :-
    create_pairs(R,S).

gen_pairs([], []) :- !.
gen_pairs([_], []) :- !.
gen_pairs([V1,V2|R], [V1-V2|S]) :-
    gen_pairs([V2|R], S).

sort_collection(COL, ATTR, SORTED_COL) :-
    build_key_collection(COL, ATTR, KEY_COL),
    keysort(KEY_COL, SORTED_KEY_COL),
    remove_key_from_collection(SORTED_KEY_COL, SORTED_COL).

build_key_collection([], _, []).
build_key_collection([ITEM|RCOL], ATTR, [KEY-ITEM|R]) :-
    extract_attr_value(ITEM, ATTR, KEY),
    build_key_collection(RCOL, ATTR, R).

extract_attr_value([ATTR-VALUE|_], ATTR, VALUE) :- !.
extract_attr_value([_|RITEM], ATTR, VALUE) :-
    extract_attr_value(RITEM, ATTR, VALUE).
```

```
remove_key_from_collection([], []).
remove_key_from_collection([_-ITEM|R], [ITEM|S]) :-
    remove_key_from_collection(R, S).

remove_key_from_col([], []).
remove_key_from_col([[_-ITEM]|R], [ITEM|S]) :-
    remove_key_from_col(R, S).

list_dvar_range([], 0) :- !.
list_dvar_range([X|Y], R) :-
    get_minimum([X|Y], Minimum),
    get_maximum([X|Y], Maximum),
    Min in Minimum..Maximum,
    Max in Minimum..Maximum,
    minimum(Min, [X|Y]),
    maximum(Max, [X|Y]),
    R #= Max-Min+1.

collection_distinct([], _).
collection_distinct([ITEM|R], ATTR) :-
    nth1(ATTR, ITEM, _-A),
    get_attr1(A, L),
    all_different(L),
    collection_distinct(R, ATTR).

collection_increasing_seq(COL, ATTRS) :-
    collection_increasing_seq1(COL, ATTRS, A),
    lex_chain(A, [op(#<)]).

collection_increasing_seq1([], _, []).
collection_increasing_seq1([ITEM|R], ATTRS, [ITEM_ATTRS|S]) :-
    collection_increasing_seq2(ATTRS, ITEM, ITEM_ATTRS),
    collection_increasing_seq1(R, ATTRS, S).

collection_increasing_seq2([], _, []).
collection_increasing_seq2([ATTR|R], ITEM, [A|S]) :-
    nth1(ATTR, ITEM, _-A),
    collection_increasing_seq2(R, ITEM, S).

collection([], _) :- !.
collection([Item|R], Types) :-
    check_item(Types, Item),
    collection(R, Types).

create_collection([], _, []).
```

```
create_collection([V|R], ATTR, [[ATTR-V]|S]) :-
    create_collection(R, ATTR, S).

create_collection([], [], _, _, []).
create_collection([V1|R1], [V2|R2], ATTR1, ATTR2, [[ATTR1-V1,ATTR2-V2]|S]) :
    create_collection(R1, R2, ATTR1, ATTR2, S).

create_collection([], _, _, []).
create_collection([[_-L]|R], ATTR1, ATTR2, [[ATTR1-C]|S]) :-
    get_attr1(L, A),
    create_collection(A, ATTR2, C),
    create_collection(R, ATTR1, ATTR2, S).

check_item([], []) :- !.
check_item([T|S], [_-V|R]) :-
    check_type(T, V),
    check_item(S, R).

check_type(atom, V) :-
    atom(V), !.
check_type(atom(L), V) :-
    atom(V),
    member(V, L), !.
check_type(int, V) :-
    integer(V), !.
check_type(int_gteq(VAL), V) :-
    integer(V),
    V >= VAL, !.
check_type(int_diff(VAL), V) :-
    integer(V),
    V =\= VAL, !.
check_type(dvar, V) :-
    integer(V), !.
check_type(dvar, V) :-
    fd_var(V), !.
check_type(fdvar, V) :-
    var(V), !.
check_type(fdvar, V) :-
    integer(V), !.
check_type(fdvar, V) :-
    fd_var(V), !.
check_type(dvar_gteq(VAL), V) :-
    integer(V),
    V >= VAL, !.
check_type(dvar_gteq(VAL), V) :-
    fd_var(V),
```

```
    V #>= VAL, !.
check_type(int(Low,Up), V) :-
    integer(V),
    V >= Low,
    V =< Up, !.
check_type(dvar(Low,Up), V) :-
    integer(V),
    V >= Low,
    V =< Up, !.
check_type(dvar(Low,Up), V) :-
    fd_var(V),
    V #>= Low,
    V #=< Up, !.
check_type(fdvar(Low,Up), V) :-
    integer(V),
    V >= Low,
    V =< Up, !.
check_type(fdvar(Low,Up), V) :-
    fd_var(V),
    V #>= Low,
    V #=< Up, !.
check_type(fdvar(Low,Up), V) :-
    var(V),
    V #>= Low,
    V #=< Up, !.
check_type(col(Types), C) :-
    collection(C, Types), !.
check_type(col(Len,Types), C) :-
    length(C, Len),
    collection(C, Types), !.
check_type(col_len_gteq(Len,Types), C) :-
    length(C, L),
    L >= Len,
    collection(C, Types), !.
check_type(non_empty_col(Types), C) :-
    length(C, L),
    L > 0,
    collection(C, Types), !.

check_type(sint, _V) :-         % TODO
    !.
check_type(svar, _V) :-         % TODO
    !.

get_col_attr1([], _, []).
get_col_attr1([[_-C|_]|R], 1, [D|S]) :- !,
```

```
    get_attr1(C, D),
    get_col_attr1(R, 1, S).
get_col_attr1([[_-C|_]|R], 2, [D|S]) :- !,
    get_attr2(C, D),
    get_col_attr1(R, 2, S).
get_col_attr1([[_-C|_]|R], 3, [D|S]) :-
    get_attr3(C, D),
    get_col_attr1(R, 3, S).

get_col_attr2([], _, []).
get_col_attr2([[_,_-C|_]|R], 1, [D|S]) :- !,
    get_attr1(C, D),
    get_col_attr2(R, 1, S).
get_col_attr2([[_,_-C|_]|R], 2, [D|S]) :- !,
    get_attr2(C, D),
    get_col_attr2(R, 2, S).
get_col_attr2([[_,_-C|_]|R], 3, [D|S]) :-
    get_attr3(C, D),
    get_col_attr2(R, 3, S).

get_col_attr3([], _, []).
get_col_attr3([[_,_,_-C|_]|R], 1, [D|S]) :- !,
    get_attr1(C, D),
    get_col_attr3(R, 1, S).
get_col_attr3([[_,_,_-C|_]|R], 2, [D|S]) :- !,
    get_attr2(C, D),
    get_col_attr3(R, 2, S).
get_col_attr3([[_,_,_-C|_]|R], 3, [D|S]) :-
    get_attr3(C, D),
    get_col_attr3(R, 3, S).

get_attr12([], []).
get_attr12([[_-V1,_-V2|_]|R], [V1-V2|S]) :-
    get_attr12(R, S).

get_attr21([], []).
get_attr21([[_-V1,_-V2|_]|R], [V2-V1|S]) :-
    get_attr21(R, S).

get_attr12_sum([], []).
get_attr12_sum([[_-V1,_-V2|_]|R], [V|S]) :-
    V is V1+V2,
    get_attr12_sum(R, S).

get_attr12_diff20([], []).
get_attr12_diff20([[_,_-0|_]|R], S) :- !,
```

```
    get_attr12_diff20(R, S).
get_attr12_diff20([[_-V1,_-V2|_]|R], [V1-V2|S]) :-
    get_attr12_diff20(R, S).

get_attr12_diff20_end([], []).
get_attr12_diff20_end([[_,_-0|_]|R], S) :- !,
    get_attr12_diff20_end(R, S).
get_attr12_diff20_end([[_-V1,_-V2|_]|R], [V12-V|S]) :-
    V12 is V1+V2,
    V is -V2,
    get_attr12_diff20_end(R, S).

get_kattr1([], _, []).
get_kattr1([[_-V|_]|R], K, [V-K|S]) :-
    K1 is K+1,
    get_kattr1(R, K1, S).

get_attr1([], []).
get_attr1([[_-V|_]|R], [V|S]) :-
    get_attr1(R, S).

get_attr2([], []).
get_attr2([[_,_-V|_]|R], [V|S]) :-
    get_attr2(R, S).

get_attr3([], []).
get_attr3([[_,_,_-V|_]|R], [V|S]) :-
    get_attr3(R, S).

get_attr4([], []).
get_attr4([[_,_,_,_-V|_]|R], [V|S]) :-
    get_attr4(R, S).

get_attr5([], []).
get_attr5([[_,_,_,_,_-V|_]|R], [V|S]) :-
    get_attr5(R, S).

get_attr6([], []).
get_attr6([[_,_,_,_,_,_-V|_]|R], [V|S]) :-
    get_attr6(R, S).

get_attr7([], []).
get_attr7([[_,_,_,_,_,_,_-V|_]|R], [V|S]) :-
    get_attr7(R, S).

get_attr8([], []).
```

```
get_attr8([[_,_,_,_,_,_,_,_-V|_]|R], [V|S]) :-
    get_attr8(R, S).

get_attr11([], []).
get_attr11([[_-V|_]|R], [U|S]) :-
    get_attr1(V, U),
    get_attr11(R, S).

get_proj1([], []).
get_proj1([[A-V|_]|R], [[A-V]|S]) :-
    get_proj1(R, S).

get_minimum([], 0).
get_minimum([V|R], M) :-
    fd_min(V, Min),
    get_minimum1(R, Min, M).

get_minimum1([], Min, Min).
get_minimum1([V|R], Min, M) :-
    fd_min(V, MinV),
    MinV < Min, !,
    get_minimum1(R, MinV, M).
get_minimum1([_|R], Min, M) :-
    get_minimum1(R, Min, M).

get_maximum([], 0).
get_maximum([V|R], M) :-
    fd_max(V, Max),
    get_maximum1(R, Max, M).

get_maximum1([], Max, Max).
get_maximum1([V|R], Max, M) :-
    fd_max(V, MaxV),
    MaxV > Max, !,
    get_maximum1(R, MaxV, M).
get_maximum1([_|R], Max, M) :-
    get_maximum1(R, Max, M).

gen_collection([], _, []).
gen_collection([V|R], ATTR, [[ATTR-V]|S]) :-
    gen_collection(R, ATTR, S).

gen_varcst([], [], []).
gen_varcst([V|R], [C|S], [VC|T]) :-
    VC #= V+C,
    gen_varcst(R, S, T).
```

```
gen_quotient([], _, []).
gen_quotient([V|R], Size, [Q|T]) :-
    Size1 is Size-1,
    Remainder in 0.. Size1,
    V #= Size*Q+ Remainder,
    gen_quotient(R, Size, T).

gen_quotient_fix([], _, []).
gen_quotient_fix([V|R], Size, [Q|T]) :-
    (   V >= 0 ->
        Q is V // Size
    ;   Q is -((-1-V) // Size) - 1
    ),
    gen_quotient_fix(R, Size, T).

gen_remainder([], _, []).
gen_remainder([V|R], M, [Remainder |T]) :-
    M1 is M-1,
    Remainder in 0.. M1,
    V #= M*_+ Remainder,
    gen_remainder(R, M, T).

flattern([], []).
flattern([L|R], S) :-
    flattern(R, T),
    append(L, T, S).

get_partition_var([], _, [], _).
get_partition_var([V|R], PVALS, [P|S], MAX) :-
    P in 0..MAX,
    gen_part_var(PVALS, 1, V, P),
    get_partition_var(R, PVALS, S, MAX).

get_partition_var([], _, [], _, _).
get_partition_var([V|R], PVALS, [P|S], MAX, DIFF) :-
    P in 1..MAX,
    P #\= DIFF,
    gen_part_var(PVALS, 1, V, P),
    get_partition_var(R, PVALS, S, MAX, DIFF).

gen_part_var([], _, _, _).
gen_part_var([L|R], N, V, P) :-
    gen_part_var1(L, N, V, P, Vdiff),
    call(Vdiff #=> P #\= N),
    N1 is N+1,
```

```
    gen_part_var(R, N1, V, P).

gen_part_var1([], _, _, _, 1).
gen_part_var1([U|R], N, V, P, V #\= U #/\ S) :-
    V #= U #=> P #= N,
    gen_part_var1(R, N, V, P, S).

set_to_list({}, []).
set_to_list({S}, L) :-
    set_to_list(S, L, []).

set_to_list((X,Y)) --> !,
    set_to_list(X),
    set_to_list(Y).
set_to_list(X) --> [X].

list_to_set([], {}).
list_to_set([H|T], {S}) :-
    list_to_set(T, H, S).

list_to_set([], X, X).
list_to_set([H|T], X, (X,S)) :-
    list_to_set(T, H, S).

count_var_notin_values([], _, 0) :- !.
count_var_notin_values([VAR|RVAR], SORTED_VALS, NOUT) :-
    fd_set(VAR, S),
    fdset_to_list(S, L),
    (ord_intersect(L, SORTED_VALS) -> I=0 ; I=1),
    count_var_notin_values(RVAR, SORTED_VALS, N),
    NOUT is N+I.

complete_card(MIN, MAX, _, [], []) :-
    MIN > MAX, !.
complete_card(MIN, MAX, L, [[var-OCC]|R], [MIN-OCC|S]) :-
    MIN =< MAX,
    OCC in 0..L,
    MIN1 is MIN+1,
    complete_card(MIN1, MAX, L, R, S).

complete_card(MIN, MIN, NVARS, VALS, NOCCS, [V-N]) :- !,
    complete_card1(MIN, VALS, NOCCS, V_N),
    (V_N=[] -> V=MIN, N in 0..NVARS ; V_N=V-N).
complete_card(MIN, MAX, NVARS, VALS, NOCCS, [V-N|R]) :-
    MIN < MAX,
    complete_card1(MIN, VALS, NOCCS, V_N),
```

```
    (V_N=[] -> V=MIN, N in 0..NVARS ; V_N=V-N),
    MIN1 is MIN + 1,
    complete_card(MIN1, MAX, NVARS, VALS, NOCCS, R).

complete_card1(_, [], [], []) :- !.
complete_card1(MIN, [MIN|_], [NOCC|_], MIN-NOCC) :- !.
complete_card1(MIN, [VAL|R], [_NOCC|S], MN) :-
    MIN =\= VAL,
    complete_card1(MIN, R, S, MN).

complete_card_low_up(MIN, MIN, NVARS, VALS, OMINS, OMAXS, [V-N]) :- !,
    complete_card_low_up1(MIN, VALS, OMINS, OMAXS, V_N),
    (V_N=[] -> V=MIN, N in 0..NVARS ; V_N=V-N).
complete_card_low_up(MIN, MAX, NVARS, VALS, OMINS, OMAXS, [V-N|R]) :-
    MIN < MAX,
    complete_card_low_up1(MIN, VALS, OMINS, OMAXS, V_N),
    (V_N=[] -> V=MIN, N in 0..NVARS ; V_N=V-N),
    MIN1 is MIN + 1,
    complete_card_low_up(MIN1, MAX, NVARS, VALS, OMINS, OMAXS, R).

complete_card_low_up1(_, [], [], [], []) :- !.
complete_card_low_up1(MIN, [MIN|_], [OMIN|_], [OMAX|_], MIN-NOCC) :- !,
    NOCC in OMIN..OMAX.
complete_card_low_up1(MIN, [VAL|R], [_|S], [_|T], MN) :-
    MIN =\= VAL,
    complete_card_low_up1(MIN, R, S, T, MN).

complete_card_consec(LOW, UP, ATMOST, NVAR, [LOW-N|R]) :-
    LOW < UP, !,
    N in 0..ATMOST,
    LOW1 is LOW+1,
    complete_card_consec(LOW1, UP, ATMOST, NVAR, R).
complete_card_consec(LOW, LOW, _, NVAR, [LOW-N]) :-
    N in 0..NVAR.

build_or_var_in_values([], _, true).
build_or_var_in_values([U], V, (V#=U)) :- !.
build_or_var_in_values([U1,U2|R], V, (V#=U1) #\/ S) :-
    build_or_var_in_values([U2|R], V, S).

call_term_relop_value(TERM,   =, VALUE) :- !,
    call(TERM #= VALUE).
call_term_relop_value(TERM, =\=, VALUE) :- !,
    call(TERM #\= VALUE).
call_term_relop_value(TERM,   <, VALUE) :- !,
    call(TERM #< VALUE).
```

```
call_term_relop_value(TERM,  >=, VALUE) :- !,
    call(TERM #>= VALUE).
call_term_relop_value(TERM,   >, VALUE) :- !,
    call(TERM #> VALUE).
call_term_relop_value(TERM,  =<, VALUE) :-
    call(TERM #=< VALUE).

gen_matrix_bool(MINBINS, MAXBINS, _, []) :-
    MINBINS > MAXBINS, !.
gen_matrix_bool(MINBINS, MAXBINS, BINS, [LINE|RLINES]) :-
    MINBINS =< MAXBINS,
    gen_matrix_bool1(BINS, MINBINS, LINE),
    MINBINS1 is MINBINS+1,
    gen_matrix_bool(MINBINS1, MAXBINS, BINS, RLINES).

gen_matrix_bool1([], _, []).
gen_matrix_bool1([BIN|RBINS], IDBIN, [B|R]) :-
    BIN #= IDBIN #<=> B,
    gen_matrix_bool1(RBINS, IDBIN, R).

common1([], _, [], 0).
common1([V|R], VARS2, [LINE|S], SB+T) :-
    common2(VARS2, V, LINE, SUM),
    call(SUM #> 0 #<=> SB),
    common1(R, VARS2, S, T).

common2([], _, [], 0).
common2([U|R], V, [B|S], B+T) :-
    U #= V #<=> B,
    common2(R, V, S, T).

gen_cum_tasks([], [], [], [], _, []).
gen_cum_tasks([O|RO], [D|RD], [E|RE], [H|RH],
          T, [task(O,D,E,H,T)|R]) :-
    T1 is T+1,
    gen_cum_tasks(RO, RD, RE, RH, T1, R).

k_ary_tree([], _, _, _).
k_ary_tree([J|R], INDEXES, SUCCS, K) :-
    k_ary_tree1(INDEXES, SUCCS, J, Term),
    call(Term #=< K),
    k_ary_tree(R, INDEXES, SUCCS, K).

k_ary_tree1([], [], _, 0).
k_ary_tree1([I|S], [S_I|R], J, B_IJ+T) :-
    S_I #= J #/\ I #\= J #<=> B_IJ,
```

```
    k_ary_tree1(S, R, J, T).

ori_dur_end([], [], []).
ori_dur_end([O|RO], [D|RD], [E|RE]) :-
    O + D #= E,
    ori_dur_end(RO, RD, RE).

ori_end([], []).
ori_end([O|RO], [E|RE]) :-
    O #=< E,
    ori_end(RO, RE).

link_index_to_attribute([], [], _, _).
link_index_to_attribute([ID|RID], [ATT|RATT], Vi, Ai) :-
    Vi #= ID #<=> Ai #= ATT,
    link_index_to_attribute(RID, RATT, Vi, Ai).

get_sliding_prod([], _, []).
get_sliding_prod([V|R], P, [P|S]) :-
    Q is V*P,
    get_sliding_prod(R, Q, S).

get_min_list_list_dvar([], []).
get_min_list_list_dvar([L|R], [Min|S]) :-
    get_min_list_dvar(L, _, Min),
    get_min_list_list_dvar(R, S).

get_min_list_dvar([], Min, Min).
get_min_list_dvar([V|R], Cur, Min) :-
    fd_min(V, Vmin),
    (integer(Cur) -> Next is min(Cur,Vmin) ; Next = Vmin),
    get_min_list_dvar(R, Next, Min).

get_max_list_list_dvar([], []).
get_max_list_list_dvar([L|R], [Min|S]) :-
    get_max_list_dvar(L, _, Min),
    get_max_list_list_dvar(R, S).

get_max_list_dvar([], Max, Max).
get_max_list_dvar([V|R], Cur, Max) :-
    fd_max(V, Vmax),
    (integer(Cur) -> Next is max(Cur,Vmax) ; Next = Vmax),
    get_max_list_dvar(R, Next, Max).

get_ranges([], [], []).
get_ranges([A|R], [B|S], [C|T]) :-
```

```
    C is B-A+1,
    get_ranges(R, S, T).

create_matrix(N, Inf, Sup, MB) :-
    length(MB, N),
    create_matrix1(MB, N, Inf, Sup).

create_matrix1([], _, _, _).
create_matrix1([L|R], N, Inf, Sup) :-
    length(L, N),
    domain(L, Inf, Sup),
    create_matrix1(R, N, Inf, Sup).

count_relop(=  , NIN, LIMIT, FLAG) :- NIN #=  LIMIT #<=> FLAG.
count_relop(=\=, NIN, LIMIT, FLAG) :- NIN #\= LIMIT #<=> FLAG.
count_relop(<  , NIN, LIMIT, FLAG) :- NIN #<  LIMIT #<=> FLAG.
count_relop(>= , NIN, LIMIT, FLAG) :- NIN #>= LIMIT #<=> FLAG.
count_relop(>  , NIN, LIMIT, FLAG) :- NIN #>  LIMIT #<=> FLAG.
count_relop(=< , NIN, LIMIT, FLAG) :- NIN #=< LIMIT #<=> FLAG.

used_by_reified([], _, _).
used_by_reified([V|R], VARS1, VARS2) :-
    used_by_reified1(VARS1, V, Term1),
    used_by_reified1(VARS2, V, Term2),
    call(Term1 #>= Term2),
    used_by_reified(R, VARS1, VARS2).

used_by_reified1([], _, 0).
used_by_reified1([U|R], V,  B+T) :-
    U #= V #<=> B,
    used_by_reified1(R, V, T).

remove_duplicates([], []).
remove_duplicates([X|R], S) :-
    member(X, R), !,
    remove_duplicates(R, S).
remove_duplicates([X|R], [X|S]) :-
    remove_duplicates(R, S).

gcc_no_loop1([], _, 0).
gcc_no_loop1([VAR|RVAR], J, BJ+S) :-
    BJ #<=> VAR #= J,
    J1 is J+1,
    gcc_no_loop1(RVAR, J1, S).

gcc_no_loop2(J, N, _, [], _, 0) :-
```

```
    J > N, !.
gcc_no_loop2(J, N, I, [VAR|RVAR], VAL, BIJ+S) :-
    J =< N,
    J =\= I, !,
    BIJ #<=> VAR #= VAL,
    J1 is J+1,
    gcc_no_loop2(J1, N, I, RVAR, VAL, S).
gcc_no_loop2(J, N, I, [_|RVAR], VAL, 0+S) :-
    J =< N,
    J = I,
    J1 is J+1,
    gcc_no_loop2(J1, N, I, RVAR, VAL, S).

%% cond_lex/5 is used in order to state automata associated to constraints
%% cond_lex_greatereq, cond_lex_greater, cond_lex_lesseq and cond_lex_less.
%% cond_lex/3 is used in order to state the automaton associated to
%% constraint cond_lex_cost.
cond_lex(VECTOR1, VECTOR2, PREFERENCE_TABLE, O1, O2) :-
    cond_lex_signature(VECTOR1, VECT1),
    cond_lex_signature(VECTOR2, VECT2),
                             % from each item extract a tuple of values
                             % and add key at the end
    gen_tuples(PREFERENCE_TABLE, 1, T1),
                             % sort in lexicographic order
    sort(T1, T2),
                             % to each tuple of value add state variables
    gen_tuples_var(T2, T3),
                             % get arity of the tuples
    T1 = [T|_], functor(T, _, N),
    retractall(num_state(_)),
                             % initial state number minus 1
    assert(num_state(0)),
                             % fix the states variables
    gen_state(1, N, T3),
                             % get last state
    num_state(LastS),
                             % generate the list of states of the automaton
    gen_states(0, LastS, States),
                             % generate the list of transitions of the automaton
    gen_transitions(1, N, T3, Transitions),
                             % get number of tuples of preference table
    length(PREFERENCE_TABLE, NbTuples),
                             % O1 indicates position of tuple for VECTOR1
                             % O2 indicates position of tuple for VECTOR2
    domain([O1,O2], 1, NbTuples),
                             % build signature variables for automaton for O1
```

```
    append(VECT1,[O1],VECTOR_O1),
                                % build signature variables for automaton fo
    append(VECT2,[O2],VECTOR_O2),
                                % state automaton that computes O1
    automaton(VECTOR_O1, _,
              VECTOR_O1,
              States,
              Transitions,
              [], [], []),
                                % state automaton that computes O2
    automaton(VECTOR_O2, _,
              VECTOR_O2,
              States,
              Transitions,
              [], [], []).

cond_lex(VECTOR, PREFERENCE_TABLE, O) :-
    cond_lex_signature(VECTOR, VECT),
                                % from each item extract a tuple of values
                                % and add key at the end
    gen_tuples(PREFERENCE_TABLE, 1, T1),
                                % sort in lexicographic order
    sort(T1, T2),
                                % to each tuple, add state variables
    gen_tuples_var(T2, T3),
                                % get arity of the tuples
    T1 = [T|_], functor(T, _, N),
    retractall(num_state(_)),
                                % initial state number minus 1
    assert(num_state(0)),
                                % fix the states variables
    gen_state(1, N, T3),
                                % get last state
    num_state(LastS),
                                % generate states of the automaton
    gen_states(0, LastS, States),
                                % generate transitions of the automaton
    gen_transitions(1, N, T3, Transitions),
                                % get number of tuples of preference table
    length(PREFERENCE_TABLE, NbTuples),
                                % O is position of tuple for VECTOR
    domain([O], 1, NbTuples),
                                % build signature for automaton computing O
    append(VECT,[O],VECTOR_O),
                                % state automaton that computes O
    automaton(VECTOR_O, _,
```

```
               VECTOR_O,
               States,
               Transitions,
               [], [], []).

cond_lex_signature([], []).
cond_lex_signature([[var-VAR]|R], [VAR|S]) :-
    cond_lex_signature(R, S).

gen_tuples([], _, []).
gen_tuples([[_-X]|Y], I, [U|V]) :-
    gen_tuple(X, I, U),
    J is I + 1,
    gen_tuples(Y, J, V).

gen_tuple(X, I, U) :-
    gen_tup(X, Y),
    append(Y, [I], Y1),
    append([t], Y1, Z),
    U =.. Z.

gen_tup([], []).
gen_tup([[_-I]|R], [I|S]) :-
    gen_tup(R, S).

gen_tuples_var([], []).
gen_tuples_var([A|B], [C|D]) :-
    A =.. LA,
    LA = [TA|RA],
    add_var_to_list_elem(RA, RC),
    LC = [TA|RC],
    C =.. LC,
    gen_tuples_var(B, D).

add_var_to_list_elem([], []).
add_var_to_list_elem([A|RA], [A-_|R]) :-
    add_var_to_list_elem(RA, R).

gen_state(I, N, L) :-
    I < N, !,
    gen_state1(L, [], I, 1, 1),
    J is I + 1,
    gen_state(J, N, L).
gen_state(N, N, L) :-
    gen_state1(L, [], N, 1, 0).
```

```
gen_state1([], _, _, _, _) :- !.
gen_state1([F|R], [], I, Inc, Inc1) :- !,
    arg(I, F, FI),
    FI = _-S1,
    num_state(S),
    S1 is S + Inc,
    retract(num_state(S)),
    assert(num_state(S1)),
    gen_state1(R, F, I, Inc1, Inc1).
gen_state1([F|R], P, I, Inc, Inc1) :-
    arg(I, F, FI),
    arg(I, P, PI),
    FI = VI-SI1,
    PI = UI-_,
    UI =\= VI, !,
    num_state(SI),
    SI1 is SI + Inc,
    retract(num_state(SI)),
    assert(num_state(SI1)),
    gen_state1(R, F, I, Inc1, Inc1).
gen_state1([F|R], P, I, Inc, Inc1) :-
    I > 1,
    J is I - 1,
    arg(J, F, FJ),
    arg(J, P, PJ),
    FJ = _-SJ,
    PJ = _-RJ,
    SJ =\= RJ, !,
    arg(I, F, FI),
    FI = _-SI1,
    num_state(SI),
    SI1 is SI + Inc,
    retract(num_state(SI)),
    assert(num_state(SI1)),
    gen_state1(R, F, I, Inc1, Inc1).
gen_state1([F|R], _, I, _, Inc1) :-
    arg(I, F, FI),
    FI = _-SI,
    num_state(SI),
    gen_state1(R, F, I, Inc1, Inc1).

gen_states(0, J, [source(0)|R]) :- !,
    gen_states(1, J, R).
gen_states(I, J, R /*[node(I)|R]*/) :-
    I > 0,
    I < J, !,
```

```
    I1 is I + 1,
    gen_states(I1, J, R).
gen_states(J, J, [sink(J)]) :-
    J > 0.

gen_transitions(I, N, L, T) :-
    I =< N, !,
    gen_transitions1(L, [], I, T1),
    J is I + 1,
    gen_transitions(J, N, L, T2),
    append(T1, T2, T).
gen_transitions(I, N, _, []) :-
    I > N.

gen_transitions1([F|R], [], 1, [arc(0,V1,S1)|Rarc]) :- !,
    arg(1, F, F1),
    F1 = V1-S1,
    gen_transitions1(R, F, 1, Rarc).
gen_transitions1([F|R], [], I, [arc(SJ,VI,SI)|Rarc]) :-
    I > 1, !,
    J is I - 1,
    arg(J, F, FJ),
    arg(I, F, FI),
    FJ = _-SJ,
    FI = VI-SI,
    gen_transitions1(R, F, I, Rarc).
gen_transitions1([F|R], P, I, [arc(SJ,VI,SI)|Rarc]) :-
    arg(I, F, FI),
    arg(I, P, PI),
    FI = VI-SI,
    PI = UI-RI,
    (SI =\= RI -> true ; VI=\=UI), !,
    (I=1 -> SJ=0 ; J is I-1, arg(J,F,FJ), FJ=_-SJ),
    gen_transitions1(R, F, I, Rarc).
gen_transitions1([F|R], _, I, Rarc) :- !,
    gen_transitions1(R, F, I, Rarc).
gen_transitions1([], _, _, []).

sliding_time_window1([], [], _, _, _, _, _).
sliding_time_window1([Oi|RO], [Di|RD], I, ORIGINS, DURATIONS,
                     WINDOW_SIZE, LIMIT) :-
    sliding_time_window2(ORIGINS, DURATIONS, 1, Oi, Di, I,
                         WINDOW_SIZE, LIMIT, SUM_INTER),
    call(SUM_INTER #=< LIMIT),
    I1 is I+1,
    sliding_time_window1(RO, RD, I1, ORIGINS, DURATIONS,
```

```
                         WINDOW_SIZE, LIMIT).

sliding_time_window2([], [], _, _, _, _, _, _, 0) :- !.
sliding_time_window2([_|RO], [_|RD], J, Oi, Di, I,
                     WINDOW_SIZE, LIMIT, min(Di,WINDOW_SIZE)+SUM) :-
    I = J, !,
    J1 is J+1,
    sliding_time_window2(RO, RD, J1, Oi, Di, I,
                         WINDOW_SIZE, LIMIT, SUM).
sliding_time_window2([Oj|RO], [Dj|RD], J, Oi, Di, I,
                     WINDOW_SIZE, LIMIT, SUM) :-
    I =\= J,
    fd_max(Oj, MaxOj),
    fd_max(Dj, MaxDj),
    fd_min(Oi, MinOi),
    MaxEj is MaxOj+MaxDj,
    MaxEj < MinOi, !,
    J1 is J+1,
    sliding_time_window2(RO, RD, J1, Oi, Di, I,
                         WINDOW_SIZE, LIMIT, SUM).
sliding_time_window2([Oj|RO], [_Dj|RD], J, Oi, Di, I,
                     WINDOW_SIZE, LIMIT, SUM) :-
    I =\= J,
    fd_min(Oj,MinOj),
    fd_max(Oi,MaxOi),
    E is MaxOi+WINDOW_SIZE-1,
    MinOj > E, !,
    J1 is J+1,
    sliding_time_window2(RO, RD, J1, Oi, Di, I,
                         WINDOW_SIZE, LIMIT, SUM).
sliding_time_window2([Oj|RO], [Dj|RD], J, Oi, Di, I, WINDOW_SIZE, LIMIT,
                     max(0,min(Oi+WINDOW_SIZE,Oj+Dj)-max(Oi,Oj))+SUM) :-
    J1 is J+1,
    sliding_time_window2(RO, RD, J1, Oi, Di, I,
                         WINDOW_SIZE, LIMIT, SUM).

gen_automaton_state(ATOM, I, J, STATE) :-
    number_codes(I, ICODE), atom_codes(IATOM, ICODE),
    number_codes(J, JCODE), atom_codes(JATOM, JCODE),
    atom_concat(ATOM, '_', SH),
    atom_concat(SH, IATOM, SI),
    atom_concat(SI, '_', SJ),
    atom_concat(SJ, JATOM, STATE).

check_lesseq([], []).
check_lesseq([U|R], [V|S]) :-
```

```
    U #=< V,
    check_lesseq(R, S).

get_sum([], 0).
get_sum([V|R], S) :-
    get_sum(R, T),
    S is V+T.

build_sum_squares_int([V|R], CUR, RES) :- !,
    NEXT is V*V+CUR,
    build_sum_squares_int(R, NEXT, RES).
build_sum_squares_int([], RES, RES).

build_sum_var([], 0).
build_sum_var([V|R], V+S) :-
    build_sum_var(R, S).

build_sum_squares_var([], 0).
build_sum_squares_var([V|R], V*V+S) :-
    build_sum_squares_var(R, S).

build_sum_cubes_var([], 0).
build_sum_cubes_var([V|R], V*V*V+S) :-
    build_sum_cubes_var(R, S).

build_sum_powers4_var([], 0).
build_sum_powers4_var([V|R], V*V*V*V+S) :-
    build_sum_powers4_var(R, S).

build_sum_powers5_var([], 0).
build_sum_powers5_var([V|R], V*V*V*V*V+S) :-
    build_sum_powers5_var(R, S).

build_sum_powers6_var([], 0).
build_sum_powers6_var([V|R], V*V*V*V*V*V+S) :-
    build_sum_powers6_var(R, S).

build_prod_var([], 1).
build_prod_var([V|R], V*S) :-
    build_prod_var(R, S).

build_sliding_sums([], _, []).
build_sliding_sums([V|R], P, [PV|S]) :-
    PV #= P+V,
    build_sliding_sums(R, PV, S).
```

```
period1(0, _, []) :- !.
period1(P, L, [R|S]) :-
    P > 0,
    period2(L, 0, P, R),
    P1 is P-1,
    period1(P1, L, S).

period2([], _, 0, []) :- !.
period2([], I, P, [[]|R]) :-
    P > 0,
    P1 is P-1,
    period2([], I, P1, R).
period2([X|Y], I, P, R) :-
    I1 is (I+1) mod P,
    period2(Y, I1, P, S),
    period3(X, I, S, R).

period3(X, 0, [U|V], [W|V]) :- !,
    append([X], U, W).
period3(X, I, [U|V], [U|W]) :-
    I > 0,
    I1 is I-1,
    period3(X, I1, V, W).

period4([], _, _, []).
period4([L|LL], Z, CTR, [B|S]) :-
    period5(L, Z, CTR, R),
    call(R #<=> B),
    period4(LL, Z, CTR, S).

period5([], _, _, 1).
period5([L|R], Z, CTR, T #/\ S) :-
    period6(L, Z, CTR, T),
    period5(R, Z, CTR, S).

period6([], _, _, 1) :- !.
period6([_], _, _, 1) :- !.
period6([X,Y|R], 1, =, X#=Y #/\ S) :- !,
    period6([Y|R], 1, =, S).
period6([X,Y|R], 1, =\=, X#\=Y #/\ S) :- !,
    period6([Y|R], 1, =\=, S).
period6([X,Y|R], 1, <, X#<Y #/\ S) :- !,
    period6([Y|R], 1, <, S).
period6([X,Y|R], 1, >=, X#>=Y #/\ S) :- !,
    period6([Y|R], 1, >=, S).
period6([X,Y|R], 1, >, X#>Y #/\ S) :- !,
```

```
    period6([Y|R], 1, >, S).
period6([X,Y|R], 1, =<, X#=<Y #/\ S) :- !,
    period6([Y|R], 1, =<, S).
period6([X,Y|R], 0, =, (X#=0 #\/ Y#=0 #\/ X#=Y) #/\ S) :- !,
    period6([Y|R], 0, =, S).
period6([X,Y|R], 0, =\=, (X#=0 #\/ Y#=0 #\/ X#\=Y) #/\ S) :- !,
    period6([Y|R], 0, =\=, S).
period6([X,Y|R], 0, <, (X#=0 #\/ Y#=0 #\/ X#<Y) #/\ S) :- !,
    period6([Y|R], 0, <, S).
period6([X,Y|R], 0, >=, (X#=0 #\/ Y#=0 #\/ X#>=Y) #/\ S) :- !,
    period6([Y|R], 0, >=, S).
period6([X,Y|R], 0, >, (X#=0 #\/ Y#=0 #\/ X#>Y) #/\ S) :- !,
    period6([Y|R], 0, >, S).
period6([X,Y|R], 0, =<, (X#=0 #\/ Y#=0 #\/ X#=<Y) #/\ S) :- !,
    period6([Y|R], 0, =<, S).
period6([X,Y|R], 2, CTRS, Term #/\ S) :- !,
    build_vectors_compare(X, Y, CTRS, Term),
    period6([Y|R], 2, CTRS, S).

period7([], _, _, _, 0).
period7([B|R], I, P, N, (N #/\ B #/\ P#=I) #\/ S) :- !,
    I1 is I+1,
    period7(R, I1, P, N #/\ #\B, S).

build_vectors_compare([], [], [], 1) :- !.
build_vectors_compare([X|RX], [Y|RY], [=|RCTR], X#=Y #/\ R) :-
    build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [=\=|RCTR], X#\=Y #/\ R) :-
    build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [<|RCTR], X#<Y #/\ R) :-
    build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [>=|RCTR], X#>=Y #/\ R) :-
    build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [>|RCTR], X#>Y #/\ R) :-
    build_vectors_compare(RX, RY, RCTR, R).
build_vectors_compare([X|RX], [Y|RY], [=<|RCTR], X#=<Y #/\ R) :-
    build_vectors_compare(RX, RY, RCTR, R).

build_vectors_compare_change([], [], [], 0) :- !.
build_vectors_compare_change([X|RX], [Y|RY], [=|RCTR],  X#=Y #\/ R) :- !,
    build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [=\=|RCTR], X#\=Y #\/ R) :- !,
    build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [<|RCTR],  X#<Y #\/ R) :- !,
    build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [>=|RCTR],  X#>=Y #\/ R) :- !,
```

```
    build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [>|RCTR],  X#>Y #\/ R) :- !,
    build_vectors_compare_change(RX, RY, RCTR, R).
build_vectors_compare_change([X|RX], [Y|RY], [=<|RCTR],  X#=<Y #\/ R) :- !,
    build_vectors_compare_change(RX, RY, RCTR, R).

geost_dims(D, D, [D]) :- !.
geost_dims(D, K, [D|R]) :-
    D < K,
    D1 is D+1,
    geost_dims(D1, K, R).

geost1([], [], [], []).
geost1([OID|R], [SID|S], [X|T], [object(OID,SID,X)|U]) :-
    geost1(R, S, T, U).

geost2([], [], [], []).
geost2([SID|R], [T|S], [L|U], [sbox(SID,T,L)|V]) :-
    geost2(R, S, U, V).

bin_packing1([], _, []).
bin_packing1([[_-B,_-W]|R], I, [task(B,1,B1,W,I)|RT]) :-
    I1 is I+1,
    B1 #= B+1,
    bin_packing1(R, I1, RT).

nvector_common(NVEC, VECTORS) :-
    vectors_convert_to_vars(VECTORS, VARS),
    nvalue(NVEC, VARS).

vectors_convert_to_vars(VECTORS, VARS) :-
    get_col_attr1(VECTORS, 1, VECTS),
    transpose(VECTS, TVECTS),
    get_min_list_list_dvar(TVECTS, MINS),
    get_max_list_list_dvar(TVECTS, MAXS),
    get_ranges(MINS, MAXS, RANGES),
    reverse(RANGES, RRANGES),
    get_sliding_prod(RRANGES, 1, PRODS),
    reverse(MINS, RMINS),
    nvector_common1(VECTS, RMINS, PRODS, VARS).

nvector_common1([], _, _, []).
nvector_common1([VECT|R], RMINS, PRODS, [V|S]) :-
    reverse(VECT, RVECT),
    nvector_common2(RVECT, RMINS, PRODS, Term),
    call(V #= Term),
```

```
    nvector_common1(R, RMINS, PRODS, S).

nvector_common2([], _, _, 0).
nvector_common2([V|R], [MIN|S], [PROD|T], PROD*V-Q+E) :-
    Q is PROD*MIN,
    nvector_common2(R, S, T, E).

stretch_lmin([], []) :- !.
stretch_lmin([0|R], [1|S]) :- !,
    stretch_lmin(R, S).
stretch_lmin([L|R], [L|S]) :-
    L > 0,
    stretch_lmin(R, S).

stretch_reduce_lmax([], _, []).
stretch_reduce_lmax([L|R], N, [M|S]) :-
    M is min(L,N),
    stretch_reduce_lmax(R, N, S).

stretch_gen_states([], [], _, _, [sink(s),source(s)]).
stretch_gen_states([LMIN|LMINs], [LMAX|LMAXs], NVAR, I, STATES) :-
    LMIN =< LMAX,
    (   LMIN =< 1, LMAX >= NVAR -> STATES1 = []
    ;   stretch_gen_states1(LMIN, LMAX, I, STATES1)
    ),
    I1 is I+1,
    stretch_gen_states(LMINs, LMAXs, NVAR, I1, STATES2),
    append(STATES1, STATES2, STATES).

stretch_gen_states1(LCUR, LMAX, _, []) :-
    LCUR > LMAX, !.
stretch_gen_states1(LCUR, LMAX, I, [sink(S)|R]) :-
    LCUR =< LMAX,
    gen_automaton_state('s',I,LCUR,S),
    LCUR1 is LCUR+1,
    stretch_gen_states1(LCUR1, LMAX, I, R).

stretch_gen_transitions(I, M, [], [], _, _, _, [arc(s,0,s)]) :-
    I > M, !.
stretch_gen_transitions(I, M, [LMIN|LMINs], [LMAX|LMAXs],
                        LLMIN, LLMAX, NVAR, TRANSITIONS) :-
    I =< M,
    (   LMIN =< 1, LMAX >= NVAR ->
        T0  = [arc(s,I,s)],
        LT1 = [],
        LT2 = []
```

```
    ;   gen_automaton_state('s',I,1,S_I_1),
        T0 = [arc(s,I,S_I_1)],
        stretch_gen_transitions1(1, LMAX, LMIN, I, M, LLMIN, LLMAX, NVAR, LT
        LMAX1 is LMAX-1,
        stretch_gen_transitions2(1, LMAX1, I, M, LT2)
    ),
    I1 is I+1,
    stretch_gen_transitions(I1, M, LMINs, LMAXs, LLMIN, LLMAX, NVAR, LT0),
    append(T0, LT0, T1),
    append(T1, LT1, T2),
    append(T2, LT2, TRANSITIONS).

stretch_gen_transitions1(J, LMAX, _, _, _, _, _, _, []) :-
    J > LMAX, !.
stretch_gen_transitions1(J, LMAX, LMIN, I, M, LLMIN,
                         LLMAX, NVAR, TRANSITIONS) :-
    J =< LMAX,
    gen_automaton_state('s',I,J,S_I_J),
    (   J >= LMIN
    ->  stretch_gen_transitions11(1, M, I, J, LLMIN, LLMAX, NVAR, LT1)
    ;   LT1 = []
    ),
    J1 is J+1,
    stretch_gen_transitions1(J1, LMAX, LMIN, I, M, LLMIN, LLMAX, NVAR, LT0),
    (J >= LMIN -> append([arc(S_I_J,0,s)], LT0, T1) ; T1 = LT0),
    append(T1, LT1, TRANSITIONS).


stretch_gen_transitions11(K, M, _, _, _, _, _, []) :-
    K > M, !.
stretch_gen_transitions11(K, M, I, J, [_|LMINs], [_|LMAXs], NVAR, R) :-
    K = I, !,
    K1 is K+1,
    stretch_gen_transitions11(K1, M, I, J, LMINs, LMAXs, NVAR, R).
stretch_gen_transitions11(K, M, I, J, [LMIN|LMINs], [LMAX|LMAXs], NVAR,
                          [arc(S_I_J,K,S_K_1)|R]) :-
    K =< M,
    K =\= I,
    gen_automaton_state('s',I,J,S_I_J),
    (   LMIN =< 1, LMAX >= NVAR ->
        S_K_1 = 's'
    ;   gen_automaton_state('s',K,1,S_K_1)
    ),
    K1 is K+1,
    stretch_gen_transitions11(K1, M, I, J, LMINs, LMAXs, NVAR, R).
```

```
stretch_gen_transitions2(J, LMAX, _, _, []) :-
    J > LMAX, !.
stretch_gen_transitions2(J, LMAX, I, M, [arc(S_I_J,I,S_I_J1)|R]) :-
    J =< LMAX,
    gen_automaton_state('s',I,J,S_I_J),
    J1 is J+1,
    gen_automaton_state('s',I,J1,S_I_J1),
    stretch_gen_transitions2(J1, LMAX, I, M, R).

symmetric_alldifferent0(NODES, SNODES) :-
    symmetric_alldifferent0a(NODES, L),
    sort(L, S),
    symmetric_alldifferent0a(SNODES, S).

symmetric_alldifferent0a([], []).
symmetric_alldifferent0a([[index-INDEX,succ-SUCC]|R], [INDEX-SUCC|S]) :-
    symmetric_alldifferent0a(R, S).

symmetric_alldifferent1([], _, _).
symmetric_alldifferent1([Si|RS], I, SUCCS) :-
    symmetric_alldifferent2(SUCCS, 1, Si, I),
    I1 is I+1,
    symmetric_alldifferent1(RS, I1, SUCCS).

symmetric_alldifferent2([], _, _, _).
symmetric_alldifferent2([Sj|RS], J, Si, I) :-
    Si #= J #<=> Sj #= I,
    J1 is J+1,
    symmetric_alldifferent2(RS, J1, Si, I).

symmetric_alldifferent_check([]) :- !.
symmetric_alldifferent_check([I-a(J),I-b(J)|R]) :- !,
    I \== J,
    symmetric_alldifferent_check(R).
symmetric_alldifferent_check([I-b(J),I-a(J)|R]) :-
    I \== J,
    symmetric_alldifferent_check(R).

derangement1([], []).
derangement1([S|R], [I|T]) :-
    S #\= I,
    derangement1(R, T).

derangement1_fix([], []).
derangement1_fix([S|R], [I|T]) :-
    S =\= I,
```

```
    derangement1_fix(R, T).

incomparablec(U, V) :-
    length(U, N),
    length(V, N),
    N > 1,
    create_pairs(U, PU),
    create_pairs(V, PV),
    keysort(PU, SU),
    keysort(PV, SV),
    incomparablec1(SU, SV),
    incomparablec1(SV, SU).

incomparablec1([U-_|_], [V-_|_]) :-
    U > V, !.
incomparablec1([_|R], [_|S]) :-
    incomparablec1(R, S).

differ_from_k_pos([], [], 0) :- !.
differ_from_k_pos([[_-V1]|R1], [[_-V2]|R2], B+R) :-
    V1 #\= V2 #<=> B,
    differ_from_k_pos(R1, R2, R).

create_vectors_vars([], _, _, _, []) :- !.
create_vectors_vars([VEC|R], MINS, MAXS, MAX_VAL1, [V|S]) :-
    create_vector_var(VEC, MINS, MAXS, 1, Term),
    V in 0..MAX_VAL1,
    call(V #= Term),
    create_vectors_vars(R, MINS, MAXS, MAX_VAL1, S).

create_vector_var([], [], [], _, 0) :- !.
create_vector_var([V|R], [Min|RMin], [Max|RMax], P, P*(V-Min)+T) :-
    NewP is P*(Max-Min+1),
    create_vector_var(R, RMin, RMax, NewP, T).

create_occ_vars(Val, LastVal, _, [], []) :-
    Val > LastVal, !.
create_occ_vars(Val, LastVal, MAX, [Val-Occ|R], [Occ|S]) :-
    Val =< LastVal,
    Occ in 0..MAX,
    NextVal is Val+1,
    create_occ_vars(NextVal, LastVal, MAX, R, S).

create_nocc_vars(Val, LastVal, _, [], []) :-
    Val > LastVal, !.
create_nocc_vars(Val, LastVal, MAX, [[val-Val,noccurrence-Occ]|R], [[var-Occ
```

```
    Val =< LastVal,
    Occ in 0..MAX,
    NextVal is Val+1,
    create_nocc_vars(NextVal, LastVal, MAX, R, S).

get_max_val_vec_vars([], [], M, M) :- !.
get_max_val_vec_vars([Min|RMin], [Max|RMax], C, M) :-
    NewC is (Max-Min+1)*C,
    get_max_val_vec_vars(RMin, RMax, NewC, M).

get_min_max_vectors([], _, _, MIN, MAX, MIN, MAX) :- !.
get_min_max_vectors([V|R], Flag, K, MIN, MAX, RES_MIN, RES_MAX) :-
    get_min_max_vector(V, Flag, MIN, MAX, NEW_MIN, NEW_MAX),
    get_min_max_vectors(R, 1, K, NEW_MIN, NEW_MAX, RES_MIN, RES_MAX).

get_min_max_vector([], _, [], [], [], []) :- !.
get_min_max_vector([V|R], Flag, [Min|RMin], [Max|RMax],
                   [MIN|RMIN], [MAX|RMAX]) :-
    fd_min(V, MinV),
    fd_max(V, MaxV),
    (Flag = 1 -> MIN is min(Min, MinV) ; MIN is MinV),
    (Flag = 1 -> MAX is max(Max, MaxV) ; MAX is MaxV),
    get_min_max_vector(R, Flag, RMin, RMax, RMIN, RMAX).

get_max_occ_tuples_of_values([], _, M, M) :- !.
get_max_occ_tuples_of_values([T|R], Limit, CurMax, M) :-
    get_max_occ_tuples_of_values1(R, T, NewR, 1, Count),
    NewCurMax is max(CurMax, Count),
    get_max_occ_tuples_of_values(NewR, Limit, NewCurMax, M).

get_max_occ_tuples_of_values1([], _, [], M, M) :- !.
get_max_occ_tuples_of_values1([T|R], T, NewR, CurMax, M) :- !,
    NewCurMax is CurMax+1,
    get_max_occ_tuples_of_values1(R, T, NewR, NewCurMax, M).
get_max_occ_tuples_of_values1(L, _, L, M, M).

generate_subtuples([], _, _, []) :- !.
generate_subtuples([Tuple|R], K, SortFlag, Result) :-
    (   SortFlag = 1
    ->  Tuple=[Atom-T],
        sort(T, ST),
        length(T, N),
        length(ST, N),
        STuple=[Atom-ST]
    ;   STuple = Tuple
    ),
```

```
    gen_subtuples(STuple, K, SubTuples1),
    generate_subtuples(R, K, SortFlag, SubTuples2),
    append(SubTuples1, SubTuples2, Result).

gen_subtuples([_-Tuple], K, Result) :-
    length(Tuple, Len),
    findall(SubTuples, gen_sub_tuples(Tuple, Len, K, SubTuples), Result).

gen_sub_tuples(_, _, 0, []).
gen_sub_tuples([[_-V]|R], Len, K, [V|S]) :-
    K > 0,
    Len > K,
    K1 is K-1,
    Len1 is Len-1,
    gen_sub_tuples(R, Len1, K1, S).
gen_sub_tuples([_|R], Len, K, S) :-
    K > 0,
    Len > K,
    Len1 is Len-1,
    gen_sub_tuples(R, Len1, K, S).
gen_sub_tuples(L, Len, Len, RES) :-
    Len > 0,
    remove_key_from_col(L, RES).

lex_chain_lesseq_c1([]) :- !.
lex_chain_lesseq_c1([_]) :- !.
lex_chain_lesseq_c1([VECT1,VECT2|R]) :-
    lex_lesseq_c1(VECT1, VECT2),
    lex_chain_lesseq_c1([VECT2|R]).

lex_lesseq_c1([], []) :- !.
lex_lesseq_c1([V|R], [V|S]) :- !,
    lex_lesseq_c1(R, S).
lex_lesseq_c1([V1|_], [V2|_]) :-
    V1 < V2.

lex_chain_less_c1([]) :- !.
lex_chain_less_c1([_]) :- !.
lex_chain_less_c1([VECT1,VECT2|R]) :-
    lex_less_c1(VECT1, VECT2),
    lex_chain_less_c1([VECT2|R]).

lex_less_c1([V|R], [V|S]) :- !,
    lex_less_c1(R, S).
lex_less_c1([V1|_], [V2|_]) :-
    V1 < V2.
```

```
lex_alldifferent_density(0, []) :- !.
lex_alldifferent_density(Density, VECTORS) :-
    length(VECTORS, Needed),
    VECTORS = [[vec-VECTOR]|_],
    length(VECTOR, C),
    length(LMin, C),
    length(LMax, C),
    get_min_max_vectors_components(VECTORS, LMin, LMax, Min, Max),
    compute_product_availabel(Min, Max, 1, Available),
    (all_vectors_sorted(VECTORS) -> Div = 2 ; Div = 1),
    Density is Needed/(Available/Div).

get_min_max_vectors_components([], Min, Max, Min, Max) :- !.
get_min_max_vectors_components([[_-V]|R], LMin, LMax, Min, Max) :-
    get_min_max_vector_component(V, LMin, LMax, NMin, NMax),
    get_min_max_vectors_components(R, NMin, NMax, Min, Max).

get_min_max_vector_component([], [], [], [], []) :- !.
get_min_max_vector_component([[_-I]|R], [Min|S], [Max|T],
                             [NewMin|U], [NewMax|V]) :-
    (var(Min) -> NewMin is I ; NewMin is min(I,Min)),
    (var(Max) -> NewMax is I ; NewMax is max(I,Max)),
    get_min_max_vector_component(R, S, T, U, V).

compute_product_availabel([], [], A, A) :- !.
compute_product_availabel([Min|R], [Max|S], Cur, Res) :-
    NewCur is Cur*(Max-Min+1),
    compute_product_availabel(R, S, NewCur, Res).

all_vectors_sorted([]) :- !.
all_vectors_sorted([[vec-V]|R]) :-
    vector_sorted(V),
    all_vectors_sorted(R).

vector_sorted([]) :- !.
vector_sorted([_]) :- !.
vector_sorted([[var-C],[var-D]|R]) :-
    C =< D,
    vector_sorted([[var-D]|R]).

zero_vector([]) :- !.
zero_vector([[var-0]|R]) :-
    zero_vector(R).

not_zero_vector([]) :- !.
```

```
not_zero_vector([[var-V]|R]) :-
    V =\= 0,
    not_zero_vector(R).

count_zeros_in_vectors([], Zeros, Zeros) :- !.
count_zeros_in_vectors([[vec-V]|R], Cur, Res) :-
    count_zeros_in_vector(V, Cur, Next),
    count_zeros_in_vectors(R, Next, Res).

count_zeros_in_vector([], Zeros, Zeros) :- !.
count_zeros_in_vector([[var-V]|R], Cur, Res) :-
    (V = 0 -> Next is Cur+1 ; Next is Cur),
    count_zeros_in_vector(R, Next, Res).

group_convert([], [], [], _).
group_convert([V|R], [B|S], [NB|T], VALS) :-
    (   memberchk(V, VALS) ->
        B = 1, NB = 0
    ;   B = 0, NB = 1
    ),
    group_convert(R, S, T, VALS).

nvisible_from_start(s, [Vi|R], _, _, N) :- !,
    nvisible_from_start(t, R, Vi, 1, N).
nvisible_from_start(t, [Vi|R], M, C, N) :-
    M < Vi, !,
    C1 is C+1,
    nvisible_from_start(t, R, Vi, C1, N).
nvisible_from_start(t, [Vi|R], M, C, N) :-
    M >= Vi, !,
    nvisible_from_start(t, R, M, C, N).
nvisible_from_start(_, [], _, N, N).

length_first_eq_sequence([V,V|R], C, LEN) :- !,
    C1 is C+1,
    length_first_eq_sequence([V|R], C1, LEN).
length_first_eq_sequence(_, LEN, LEN).

change_eq_c([V,V|R], C, NCHANGE) :- !,
    C1 is C+1,
    change_eq_c([V|R], C1, NCHANGE).
change_eq_c([_,V|R], C, NCHANGE) :- !,
    change_eq_c([V|R], C, NCHANGE).
change_eq_c(_, NCHANGE, NCHANGE).

change_neq_c([V,V|R], C, NCHANGE) :- !,
```

```
    change_neq_c([V|R], C, NCHANGE).
change_neq_c([_,V|R], C, NCHANGE) :- !,
    C1 is C+1,
    change_neq_c([V|R], C1, NCHANGE).
change_neq_c(_, NCHANGE, NCHANGE).

change_lt_c([V1,V2|R], C, NCHANGE) :-
    V1 < V2, !,
    C1 is C+1,
    change_lt_c([V2|R], C1, NCHANGE).
change_lt_c([_,V|R], C, NCHANGE) :- !,
    change_lt_c([V|R], C, NCHANGE).
change_lt_c(_, NCHANGE, NCHANGE).

change_geq_c([V1,V2|R], C, NCHANGE) :-
    V1 >= V2, !,
    C1 is C+1,
    change_geq_c([V2|R], C1, NCHANGE).
change_geq_c([_,V|R], C, NCHANGE) :- !,
    change_geq_c([V|R], C, NCHANGE).
change_geq_c(_, NCHANGE, NCHANGE).

change_gt_c([V1,V2|R], C, NCHANGE) :-
    V1 > V2, !,
    C1 is C+1,
    change_gt_c([V2|R], C1, NCHANGE).
change_gt_c([_,V|R], C, NCHANGE) :- !,
    change_gt_c([V|R], C, NCHANGE).
change_gt_c(_, NCHANGE, NCHANGE).

change_leq_c([V1,V2|R], C, NCHANGE) :-
    V1 =< V2, !,
    C1 is C+1,
    change_leq_c([V2|R], C1, NCHANGE).
change_leq_c([_,V|R], C, NCHANGE) :- !,
    change_leq_c([V|R], C, NCHANGE).
change_leq_c(_, NCHANGE, NCHANGE).

max_nvalue_seq_size([], C, _, Best, Res) :- !,
    Res is max(C,Best).
max_nvalue_seq_size([V|R], C, V, Best, Res) :- !,
    C1 is C+1,
    max_nvalue_seq_size(R, C1, V, Best, Res).
max_nvalue_seq_size([V|R], C, Prev, Best, Res) :-
    C > 0,
    V =\= Prev,
```

```
    NewBest is max(C,Best),
    max_nvalue_seq_size(R, 1, V, NewBest, Res).

pair_signature([], []) :- !.
pair_signature([_], []) :- !.
pair_signature([[var-VAR1],[var-VAR2]|VARs], [S|Ss]) :-
    S in 0..2,
    VAR1 #< VAR2 #<=> S #= 0,
    VAR1 #= VAR2 #<=> S #= 1,
    VAR1 #> VAR2 #<=> S #= 2,
    pair_signature([[var-VAR2]|VARs], Ss).

pair_first_signature([], []) :- !.
pair_first_signature([_], []) :- !.
pair_first_signature([[var-VAR1],[var-VAR2]|VARs], [VAR1|Rs]) :-
    pair_first_signature([[var-VAR2]|VARs], Rs).

pair_first_last_signature([[var-LastVAR]], [], LastVAR) :- !.
pair_first_last_signature([[var-VAR1],[var-VAR2]|VARs],
                          [VAR1|Rs], LastVAR) :-
    pair_first_last_signature([[var-VAR2]|VARs], Rs, LastVAR).

pair_index_signature([], _, []) :- !.
pair_index_signature([_], _, []) :- !.
pair_index_signature([[var-_VAR1],[var-VAR2]|VARs], I, [I|Is]) :-
    I1 is I+1,
    pair_index_signature([[var-VAR2]|VARs], I1, Is).

difference_decreasing_slope_signature([_], []) :- !.
difference_decreasing_slope_signature([[var-VAR1],[var-VAR2]|VARs],
                                      [DIFFERENCE|RD]) :-
    VAR1 #= DIFFERENCE + VAR2,
    difference_decreasing_slope_signature([[var-VAR2]|VARs], RD).

difference_increasing_slope_signature([_], []) :- !.
difference_increasing_slope_signature([[var-VAR1],[var-VAR2]|VARs],
                                      [DIFFERENCE|RD]) :-
    VAR2 #= DIFFERENCE + VAR1,
    difference_increasing_slope_signature([[var-VAR2]|VARs], RD).

balance1([], _, [], [], []) :- !.
balance1([I|V], N, [[val-I,noccurrence-O]|R], [[var-O]|T], [[var-O1]|U]) :-
    O in 0..N,
    O #= 0 #=> O1 #= N,
    O #> 0 #=> O1 #= O,
    balance1(V, N, R, T, U).
```

```
balance2([], _, _, []) :- !.
balance2([V|R], N, VARS, [[var-O]|S]) :-
    balance3(VARS, V, Term),
    O in 1..N,
    call(O #= Term),
    balance2(R, N, VARS, S).

balance3([], _, 0) :- !.
balance3([U|R], V, B+S) :-
    U #= V #<=> B,
    balance3(R, V, S).

increasing_values([]) :- !.
increasing_values([_]) :- !.
increasing_values([U,V|R]) :-
    U < V,
    increasing_values([V|R]).

remove_value_from_vars([], _) :- !.
remove_value_from_vars([VAR|R], VAL) :-
    VAR #\= VAL,
    remove_value_from_vars(R, VAL).
```

Appendix C

Systems Correspondence Tables

C.1 From the Catalog to Choco

Catalog	Choco
ABS_VALUE	ABS
ALL_EQUAL	ATMOSTNVALUE
ALLDIFFERENT	ALLDIFFERENT
AMONG	AMONG
AMONG_VAR	AMONG
AND	REIFIEDAND
ARITH	EQ NEQ GEQ GT LEQ LT
ATLEAST	OCCURENCEMIN
ATMOST	OCCURENCEMAX
ATMOST_NVALUE	ATMOSTNVALUE
BIN_PACKING	PACK
BIN_PACKING_CAPA	PACK
CLAUSE_AND	REIFIEDAND CLAUSE
CLAUSE_OR	REIFIEDOR CLAUSE
COUNT	OCCURENCE
CUMULATIVE	CUMULATIVEMAX
CUMULATIVE_TWO_D	GEOST
DECREASING	INCREASINGNVALUE
DIFFN	GEOST
DISJOINT_TASKS	DISJOINT
DISJUNCTIVE	DISJUNCTIVE
DISTANCE	DISTANCEEQ
DOMAIN	MEMBER
DOMAIN_CONSTRAINT	DOMAINCHANNELING
ELEM	NTH
ELEMENT	NTH
ELEMENT_MATRIX	NTH
EQ	EQ
EQ_SET	EQ
EQUIVALENT	IFONLYIF
EXACTLY	OCCURENCE
GEOST	GEOST
GEOST_TIME	GEOST
GEQ	GEQ
GLOBAL_CARDINALITY	GLOBALCARDINALITY
GLOBAL_CARDINALITY_LOW_UP	GLOBALCARDINALITY
GT	GT
IMPLY	REIFIEDLEFTIMP
IN	MEMBER
IN_INTERVAL	MEMBER

Catalog	Choco
IN_RELATION	FEASPAIRAC INFEASPAIRAC RELATIONPAIRAC FEASTUPLEAC INFEASTUPLEAC RELATIONTUPLEAC
IN_SET	MEMBER
INCREASING	INCREASINGNVALUE
INCREASING_NVALUE	INCREASINGNVALUE
INVERSE	INVERSECHANNELING
INVERSE_OFFSET	INVERSECHANNELING
INVERSE_SET	INVERSESET
LEQ	LEQ
LEX_BETWEEN	LEXCHAINEQ
LEX_CHAIN_LESS	LEXCHAIN
LEX_CHAIN_LESSEQ	LEXCHAINEQ
LEX_GREATER	LEX
LEX_GREATEREQ	LEXEQ
LEX_LESS	LEX
LEX_LESSEQ	LEXEQ
LEX_LESSEQ_ALLPERM	LEXIMIN
LT	LT
MAXIMUM	MAX
MINIMUM	MIN
NAND	CLAUSE
NEQ	NEQ
NOR	REIFIEDXNOR
NOT_IN	NOTMEMBER
OR	REIFIEDOR
PROPER_FOREST	TREE
SCALAR_PRODUCT	EQUATION
SORT	SORTING
STRETCH_PATH	STRETCHPATH
STRICTLY_DECREASING	INCREASINGNVALUE
STRICTLY_INCREASING	INCREASINGNVALUE
SUM_CTR	EQUATION
TREE	TREE
XOR	REIFIEDXOR

C.2 From the Catalog to Gecode

Catalog	Gecode
ABS_VALUE	ABS
ALL_EQUAL	REL
ALLDIFFERENT	LINEAR
ALLDIFFERENT_CST	LINEAR
AMONG	COUNT
AMONG_SEQ	SEQUENCE
AMONG_VAR	COUNT
AND	REL
ARITH	REL
ATLEAST	COUNT ATLEAST
ATMOST	COUNT ATMOST
BIN_PACKING	BINPACKING
BIN_PACKING_CAPA	BINPACKING
CIRCUIT	CIRCUIT
CLAUSE_AND	CLAUSE
CLAUSE_OR	CLAUSE
COUNT	COUNT
COUNTS	COUNT
CUMULATIVE	CUMULATIVE
CUMULATIVES	CUMULATIVES
DECREASING	REL
DIFFN	NOOVERLAP
DISJUNCTIVE	UNARY
DOMAIN	DOM
DOMAIN_CONSTRAINT	CHANNEL
ELEM	ELEMENT
ELEMENT	ELEMENT
ELEMENT_MATRIX	ELEMENT
EQ	REL
EQ_SET	REL
EQUIVALENT	REL
EXACTLY	COUNT EXACTLY
GEQ	REL
GLOBAL_CARDINALITY	COUNT
GT	REL
IMPLY	REL
IN	REL DOM
IN_INTERVAL	DOM
IN_INTERVALS	DOM
IN_RELATION	EXTENSIONAL
IN_SET	REL DOM
INCREASING	REL
INT_VALUE_PRECEDE	PRECEDE

Catalog	Gecode
INT_VALUE_PRECEDE_CHAIN	PRECEDE
INVERSE	CHANNEL
INVERSE_OFFSET	CHANNEL
LEQ	REL
LEX_GREATER	REL
LEX_GREATEREQ	REL
LEX_LESS	REL
LEX_LESSEQ	REL
LINK_SET_TO_BOOLEANS	CHANNEL
LT	REL
MAXIMUM	MAX
MINIMUM	MIN
NAND	CLAUSE
NEQ	REL
NOR	CLAUSE
NOT_ALL_EQUAL	REL
NOT_IN	REL
NVALUE	NVALUES
NVALUES	NVALUES
OR	REL
ROOTS	ROOTS
SCALAR_PRODUCT	LINEAR
SET_VALUE_PRECEDE	PRECEDE
SORT	SORTED
SORT_PERMUTATION	SORTED
STRICTLY_DECREASING	REL
STRICTLY_INCREASING	REL
SUM_CTR	LINEAR
SUM_SET	WEIGHTS
XOR	REL

C.3 From the Catalog to JaCoP

Catalog	JaCoP
ALLDIFFERENT	ALLDIFFERENT ALLDIFF ALLDISTINCT
AMONG	AMONG
AMONG_SEQ	SEQUENCE
AMONG_VAR	AMONGVAR
AND	ANDBOOL
ATLEAST	COUNT
ATMOST	COUNT
CIRCUIT	CIRCUIT
COUNT	COUNT
CUMULATIVE	CUMULATIVE
DIFFN	DIFF2 DIFF DISJOINT DISJOINTCONDITIONAL
DISTANCE	DISTANCE DISTANCE2
ELEM	ELEMENT
ELEMENT	ELEMENT
EQ	XEQY
EQUIVALENT	EQBOOL
EXACTLY	COUNT
GEOST	GEOST
GEOST_TIME	GEOST
GEQ	XGTEQY
GLOBAL_CARDINALITY	GCC
GT	XGTY
IMPLY	IFTHENBOOL
IN	IN
IN_INTERVAL	IN
IN_INTERVALS	IN
IN_RELATION	EXTENSIONALSUPPORTVA EXTENSIONALSUPPORTMDD EXTENSIONALSUPPORTSTR
LEQ	XLTEQY
LT	XLTY
MAXIMUM	MAX
MINIMUM	MIN
OR	ORBOOL
POWER	XEXPYEQZ
SCALAR_PRODUCT	SUMWEIGHT
STRETCH_PATH	STRETCH
XOR	XORBOOL

C.4 From the Catalog to MiniZinc

Catalog	**MiniZinc**
ALL_EQUAL	ALL_EQUAL
ALLDIFFERENT	ALL_DIFFERENT
ALLDIFFERENT_EXCEPT_0	ALL_DIFFERENT
AMONG	AMONG
ATLEAST	AT_LEAST
ATMOST	AT_MOST
ATMOST1	AT_MOST1
BIN_PACKING	BIN_PACKING
BIN_PACKING_CAPA	BIN_PACKING_CAPA
CIRCUIT	CIRCUIT
COUNT	COUNT_EQ COUNT_GEQ COUNT_GT COUNT_LEQ COUNT_LT COUNT_NEQ
CUMULATIVE	CUMULATIVE
DECREASING	DECREASING
DIFFN	DIFFN
DISJOINT	DISJOINT
ELEMENT	ELEMENT
EXACTLY	EXACTLY
GLOBAL_CARDINALITY	GLOBAL_CARDINALITY
GLOBAL_CARDINALITY_LOW_UP	GLOBAL_CARDINALITY_LOW_UP
IN	MEMBER
IN_RELATION	TABLE
INCREASING	INCREASING
INT_VALUE_PRECEDE	VALUE_PRECEDE
INT_VALUE_PRECEDE_CHAIN	VALUE_PRECEDE_CHAIN
INVERSE	INVERSE
INVERSE_SET	INVERSE_SET
LEX2	LEX2
LEX_GREATER	LEX_GREATER
LEX_GREATEREQ	LEX_GREATEREQ
LEX_LESS	LEX_LESS
LEX_LESSEQ	LEX_LESSEQ
LINK_SET_TO_BOOLEANS	LINK_SET_TO_BOOLEANS
MAXIMUM	MAXIMUM
MINIMUM	MINIMUM
NVALUE	NVALUE
ROOTS	ROOTS
SLIDING_SUM	SLIDING_SUM
SORT	SORT
STRICT_LEX2	STRICT_LEX2
SUM	SUM_PRED

C.5 From the Catalog to SICStus

Catalog	**SICStus**
ALLDIFFERENT	ALL_DIFFERENT ALL_DISTINCT
AND	#/\
ARITH	#< #=< #> #>= #= #\=
ATLEAST	COUNT
ATMOST	COUNT
CIRCUIT	CIRCUIT
COUNT	COUNT
CUMULATIVE	CUMULATIVE
CUMULATIVES	CUMULATIVES
DOMAIN	DOMAIN
DOMAIN_CONSTRAINT	IN IN_SET
ELEM	ELEMENT
ELEMENT	ELEMENT
EQ	#=
EQUIVALENT	#<=>
EXACTLY	COUNT
GEOST	GEOST
GEQ	#>=
GLOBAL_CARDINALITY	GLOBAL_CARDINALITY
GLOBAL_CARDINALITY_WITH_COSTS	GLOBAL_CARDINALITY
GT	#>
IMPLY	#=>
IN	IN IN_SET
IN_INTERVAL	IN
IN_INTERVALS	IN
IN_RELATION	CASE RELATION TABLE
INVERSE	ASSIGNMENT
LEQ	#=<
LEX_BETWEEN	LEX_CHAIN
LEX_CHAIN_LESS	LEX_CHAIN
LEX_CHAIN_LESSEQ	LEX_CHAIN
LEX_GREATER	LEX_CHAIN
LEX_GREATEREQ	LEX_CHAIN
LEX_LESS	LEX_CHAIN
LEX_LESSEQ	LEX_CHAIN
LT	#<
MAXIMUM	MAXIMUM
MINIMUM	MINIMUM

Catalog	SICStus
MINIMUM_WEIGHT_ALLDIFFERENT	ALL_DIFFERENT ALL_DISTINCT
NAND	#/\
NEQ	#\=
NOR	#\/
NVALUE	NVALUE
OR	#\/
SCALAR_PRODUCT	SCALAR_PRODUCT
SORT	SORTING
SORT_PERMUTATION	SORTING
SUM_CTR	SCALAR_PRODUCT
XOR	#\

C.6 From Choco to the Catalog

Choco	Catalog
ABS	ABS_VALUE
ALLDIFFERENT	ALLDIFFERENT
AMONG	AMONG AMONG_VAR
ATMOSTNVALUE	ALL_EQUAL ATMOST_NVALUE
CLAUSE	CLAUSE_AND CLAUSE_OR NAND
CUMULATIVEMAX	CUMULATIVE
DISJOINT	DISJOINT_TASKS
DISJUNCTIVE	DISJUNCTIVE
DISTANCEEQ	DISTANCE
DOMAINCHANNELING	DOMAIN_CONSTRAINT
EQ	ARITH EQ EQ_SET
EQUATION	SCALAR_PRODUCT SUM_CTR
FEASPAIRAC	IN_RELATION
FEASTUPLEAC	IN_RELATION
GEOST	CUMULATIVE_TWO_D DIFFN GEOST GEOST_TIME
GEQ	ARITH GEQ
GLOBALCARDINALITY	GLOBAL_CARDINALITY GLOBAL_CARDINALITY_LOW_UP
GT	ARITH GT
IFONLYIF	EQUIVALENT
INCREASINGNVALUE	DECREASING INCREASING INCREASING_NVALUE STRICTLY_DECREASING STRICTLY_INCREASING
INFEASPAIRAC	IN_RELATION
INFEASTUPLEAC	IN_RELATION
INVERSECHANNELING	INVERSE INVERSE_OFFSET
INVERSESET	INVERSE_SET
LEQ	ARITH LEQ
LEX	LEX_GREATER LEX_LESS
LEXCHAIN	LEX_CHAIN_LESS

Choco	**Catalog**
LEXCHAINEQ	LEX_BETWEEN LEX_CHAIN_LESSEQ
LEXEQ	LEX_GREATEREQ LEX_LESSEQ
LEXIMIN	LEX_LESSEQ_ALLPERM
LT	ARITH LT
MAX	MAXIMUM
MEMBER	DOMAIN IN IN_INTERVAL IN_SET
MIN	MINIMUM
NEQ	ARITH NEQ
NOTMEMBER	NOT_IN
NTH	ELEM ELEMENT ELEMENT_MATRIX
OCCURENCE	COUNT EXACTLY
OCCURENCEMAX	ATMOST
OCCURENCEMIN	ATLEAST
PACK	BIN_PACKING BIN_PACKING_CAPA
REIFIEDAND	AND CLAUSE_AND
REIFIEDLEFTIMP	IMPLY
REIFIEDOR	CLAUSE_OR OR
REIFIEDXNOR	NOR
REIFIEDXOR	XOR
RELATIONPAIRAC	IN_RELATION
RELATIONTUPLEAC	IN_RELATION
SORTING	SORT
STRETCHPATH	STRETCH_PATH
TREE	PROPER_FOREST TREE

C.7 From Gecode to the Catalog

Gecode	**Catalog**
ABS	ABS_VALUE
ATLEAST	ATLEAST
ATMOST	ATMOST
BINPACKING	BIN_PACKING BIN_PACKING_CAPA
CHANNEL	DOMAIN_CONSTRAINT INVERSE INVERSE_OFFSET LINK_SET_TO_BOOLEANS
CIRCUIT	CIRCUIT
CLAUSE	CLAUSE_AND CLAUSE_OR NAND NOR
COUNT	AMONG AMONG_VAR ATLEAST ATMOST COUNT COUNTS EXACTLY GLOBAL_CARDINALITY
CUMULATIVE	CUMULATIVE
CUMULATIVES	CUMULATIVES
DOM	DOMAIN IN IN_INTERVAL IN_INTERVALS IN_SET
ELEMENT	ELEM ELEMENT ELEMENT_MATRIX
EXACTLY	EXACTLY
EXTENSIONAL	IN_RELATION
LINEAR	ALLDIFFERENT ALLDIFFERENT_CST SCALAR_PRODUCT SUM_CTR
MAX	MAXIMUM
MIN	MINIMUM
NOOVERLAP	DIFFN
NVALUES	NVALUE NVALUES
PRECEDE	INT_VALUE_PRECEDE INT_VALUE_PRECEDE_CHAIN SET_VALUE_PRECEDE

Gecode	Catalog
REL	ALL_EQUAL
	AND
	ARITH
	DECREASING
	EQ
	EQ_SET
	EQUIVALENT
	GEQ
	GT
	IMPLY
	IN
	IN_SET
	INCREASING
	LEQ
	LEX_GREATER
	LEX_GREATEREQ
	LEX_LESS
	LEX_LESSEQ
	LT
	NEQ
	NOT_ALL_EQUAL
	NOT_IN
	OR
	STRICTLY_DECREASING
	STRICTLY_INCREASING
	XOR
ROOTS	ROOTS
SEQUENCE	AMONG_SEQ
SORTED	SORT
	SORT_PERMUTATION
UNARY	DISJUNCTIVE
WEIGHTS	SUM_SET

C.8 From JaCoP to the Catalog

JaCoP	Catalog
ALLDIFF	ALLDIFFERENT
ALLDIFFERENT	ALLDIFFERENT
ALLDISTINCT	ALLDIFFERENT
AMONG	AMONG
AMONGVAR	AMONG_VAR
ANDBOOL	AND
CIRCUIT	CIRCUIT
COUNT	ATLEAST ATMOST COUNT EXACTLY
CUMULATIVE	CUMULATIVE
DIFF	DIFFN
DIFF2	DIFFN
DISJOINT	DIFFN
DISJOINTCONDITIONAL	DIFFN
DISTANCE	DISTANCE
DISTANCE2	DISTANCE
ELEMENT	ELEM ELEMENT
EQBOOL	EQUIVALENT
EXTENSIONALSUPPORTMDD	IN_RELATION
EXTENSIONALSUPPORTSTR	IN_RELATION
EXTENSIONALSUPPORTVA	IN_RELATION
GCC	GLOBAL_CARDINALITY
GEOST	GEOST GEOST_TIME
IFTHENBOOL	IMPLY
IN	IN IN_INTERVAL IN_INTERVALS
MAX	MAXIMUM
MIN	MINIMUM
ORBOOL	OR
SEQUENCE	AMONG_SEQ
STRETCH	STRETCH_PATH
SUMWEIGHT	SCALAR_PRODUCT
XEQY	EQ
XEXPYEQZ	POWER
XGTEQY	GEQ
XGTY	GT
XLTEQY	LEQ
XLTY	LT
XORBOOL	XOR

C.9 From MiniZinc to the Catalog

MiniZinc	**Catalog**
ALL_DIFFERENT	ALLDIFFERENT
ALL_DIFFERENT	ALLDIFFERENT_EXCEPT_0
ALL_EQUAL	ALL_EQUAL
AMONG	AMONG
AT_LEAST	ATLEAST
AT_MOST	ATMOST
AT_MOST1	ATMOST1
BIN_PACKING	BIN_PACKING
BIN_PACKING_CAPA	BIN_PACKING_CAPA
CIRCUIT	CIRCUIT
COUNT_EQ	COUNT
COUNT_GEQ	COUNT
COUNT_GT	COUNT
COUNT_LEQ	COUNT
COUNT_LT	COUNT
COUNT_NEQ	COUNT
CUMULATIVE	CUMULATIVE
DECREASING	DECREASING
DIFFN	DIFFN
DISJOINT	DISJOINT
ELEMENT	ELEMENT
EXACTLY	EXACTLY
GLOBAL_CARDINALITY	GLOBAL_CARDINALITY
GLOBAL_CARDINALITY_LOW_UP	GLOBAL_CARDINALITY_LOW_UP
INCREASING	INCREASING
INVERSE	INVERSE
INVERSE_SET	INVERSE_SET
LEX2	LEX2
LEX_GREATER	LEX_GREATER
LEX_GREATEREQ	LEX_GREATEREQ
LEX_LESS	LEX_LESS
LEX_LESSEQ	LEX_LESSEQ
LINK_SET_TO_BOOLEANS	LINK_SET_TO_BOOLEANS
MAXIMUM	MAXIMUM
MEMBER	IN
MINIMUM	MINIMUM
NVALUE	NVALUE
ROOTS	ROOTS
SLIDING_SUM	SLIDING_SUM
SORT	SORT
STRICT_LEX2	STRICT_LEX2
SUM_PRED	SUM
TABLE	IN_RELATION
VALUE_PRECEDE	INT_VALUE_PRECEDE
VALUE_PRECEDE_CHAIN	INT_VALUE_PRECEDE_CHAIN

C.10 From SICStus to the Catalog

SICStus	Catalog
#/\	AND NAND
#<	ARITH LT
#<=>	EQUIVALENT
#=	ARITH EQ
#=<	ARITH LEQ
#=>	IMPLY
#>	ARITH GT
#>=	ARITH GEQ
#\	XOR
#\/	NOR OR
#\=	ARITH NEQ
ALL_DIFFERENT	ALLDIFFERENT MINIMUM_WEIGHT_ALLDIFFERENT
ALL_DISTINCT	ALLDIFFERENT MINIMUM_WEIGHT_ALLDIFFERENT
ASSIGNMENT	INVERSE
CASE	IN_RELATION
CIRCUIT	CIRCUIT
COUNT	ATLEAST ATMOST COUNT EXACTLY
CUMULATIVE	CUMULATIVE
CUMULATIVES	CUMULATIVES
DOMAIN	DOMAIN
ELEMENT	ELEMENT
ELEMENT	ELEM
GEOST	GEOST
GLOBAL_CARDINALITY	GLOBAL_CARDINALITY GLOBAL_CARDINALITY_WITH_COSTS
IN	DOMAIN_CONSTRAINT IN IN_INTERVAL IN_INTERVALS
IN_SET	DOMAIN_CONSTRAINT IN

SICStus	**Catalog**
LEX_CHAIN	LEX_BETWEEN LEX_CHAIN_LESS LEX_CHAIN_LESSEQ LEX_GREATER LEX_GREATEREQ LEX_LESS LEX_LESSEQ
MAXIMUM	MAXIMUM
MINIMUM	MINIMUM
NVALUE	NVALUE
RELATION	IN_RELATION
SCALAR_PRODUCT	SCALAR_PRODUCT SUM_CTR
SORTING	SORT SORT_PERMUTATION
TABLE	IN_RELATION

List of exercises

Bibliography

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in order to Solve Complex Scheduling and Placement Problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993. 207, 282, 371, 779, 982

[2] M. Ågren, M. Carlsson, N. Beldiceanu, M. Sbihi, C. Truchet, and S. Zampelli. Six Ways of Integrating Symmetries within Non-Overlapping Constraints. In W.-J. van Hoeve and J. N. Hooker, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *LNCS*, pages 11–25, Pittsburgh, PA, USA, 2009. Springer-Verlag. 325

[3] M. Ågren, M. Carlsson, N. Beldiceanu, M. Sbihi, C. Truchet, and S. Zampelli. Six Ways of Integrating Symmetries within Non-Overlapping Constraints. Technical Report T2009-01, Swedish Institute of Computer Science, 2009. 1293, 1601, 1605, 1609

[4] A. V. Aho, Y. Sagiv, T. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestor with an application to the optimization of relational expressions. *SIAM Journal of Computing*, 10(3):405–421, 1981. 2285

[5] S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of Constraint Systems. In *Compugraphics*, pages 83–92, Alvor, Algarve, Portugal, 1993. 549

[6] E. Althaus, A. Bockmayr, M. Elf, T. Kasper, M. Jünger, and K. Mehlhorn. SCIL—Symbolic Constraints in Integer Linear Programming. In *10th European Symposium on Algorithms (ESA'02)*, volume 2461 of *LNCS*, pages 75–87. Springer-Verlag, September 2002. 857, 2034, 2322, 2396

[7] R. Alvarez-Valdes, F. Parreno, and J. M. Tamarit. A tabu search algorithm for the pallet loading problem. *OR Spectrum*, 27(1):43–61, 2005. 323

[8] J. Amilhastre. *Représentation par automate d'ensemble de solutions de problèmes de satisfaction de contraintes*. PhD thesis, Montpellier II University, France, 1999. In French. ii

[9] G. Appa, D. Magos, and I. Mourtos. LP Relaxations of Multiple *alldifferent* Predicates. In J.-C. Régin and M. Rueher, editors, *First International Conference on Integration of AI and OR Techniques in Constraint Programming for*

Combinatorial Optimization Problems (CPAIOR'04), volume 3011 of *LNCS*, pages 364–369, Nice, France, April 2004. Springer-Verlag. 1527

[10] G. Appa, D. Magos, and L. Mourtos. On the system of two *alldifferent* predicates. *Information Processing Letters*, 94(3):99–105, 2005. 1527

[11] K. R. Apt and M. G. Wallace. *Constraint Logic Programming using ECLiPSe*. Cambridge University Press, New York, NY, USA, 2007. 109, 478

[12] K. Artiouchine and P. Baptiste. Inter-Distance Constraint: An Extension of the All-Different Constraint for Scheduling Equal Length Jobs. In P van Beek, editor, *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS*, pages 62–76. Springer-Verlag, 2005. 541, 1143

[13] K. Artiouchine and P. Baptiste. Arc-B-Consistency of the Inter-Distance Constraint. *Constraints (Special Issue on Global Constraints)*, 12(1):3–19, 2007. 541

[14] S. Asaf, H. Eran, Y. Richter, D. P. Connors, D. L. Gresh, J. Ortega, and M. J. Mcinnis. Applying Constraint Programming to Identification and Assignment of Service Professionals. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP'2010)*, volume 6308 of *LNCS*, pages 24–37, St Andrews, Scotland, 2010. Springer-Verlag. 1523

[15] A. S. Asratian, T. M. J. Denley, and R. Häggkvist. *Bipartite Graphs and their Applications*. Cambridge University Press, 1998. 547

[16] F. Bacchus. GAC Via Unit Propagation. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 133–147. Springer-Verlag, 2007. 350

[17] P. Baptiste and S. Demassey. Tight LP Bounds for Resource Constrained Project Scheduling. *OR Spectrum*, 26:251–262, 2004. 986

[18] P. Baptiste and C. Le Pape. Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems. *Constraints*, 5(1–2):119–139, 2000. 986

[19] P. Baptiste, C. Le Pape, and L. Peridy. Global Constraints for Partial CSPs: A Case-Study of Resource and Due Date Constraints. In M. J. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming (CP'98)*, volume 1520 of *LNCS*, pages 87–101. Springer-Verlag, 1998. 1143

[20] F. W. Barnes. Packing the maximum number of $m \times n$ tiles in a $p \times q$ rectangle. *Discrete Mathematics*, 26:93–100, 1979. 323

[21] N. Barnier and P. Brisset. Graph Coloring for Air Traffic Flow Management. In *4th Int. Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'02)*, pages 133–147, Le Croisic, France, 2002. 1523

[22] N. Barnier, P. Brisset, and T. Rivière. Slot Allocation with Constraint Programming: Models and Results. In *In Proc. of the Fourth International Air Traffic Management R&D Seminar ATM*, 2001. 2269

[23] R. Barták. Dynamic Global Constraints in Backtracking Based Environments. *Annals of Operations Research*, 118(1–4):101–119, 2003. 319

[24] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Englewood Cliffs, NJ, 1999. 2430

[25] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *JACM*, 30:479–513, 1983. 195

[26] N. Beldiceanu. Parallel Machine Scheduling with Calendar Rules. In *6th International Workshop on Project Management and Scheduling (PMS'98)*, Istanbul, Turkey, 1998. 202

[27] N. Beldiceanu. Global Constraints as Graph Properties on a Structured Network of Elementary Constraints of the Same Type. In R. Dechter, editor, *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS*, pages 52–66. Springer-Verlag, 2000. Preprint available as SICS Tech Report T2000-01, `soda.swedish-ict.se/2284/2/SICS-T--2000-01--SE.pdf`. ii, 88, 408

[28] N. Beldiceanu. Global Constraints as Graph Properties on Structured Network of Elementary Constraints of the Same Type. Technical Report T2000-01, Swedish Institute of Computer Science, 2000. i, iii, 408, 800

[29] N. Beldiceanu. Pruning for the *minimum* Constraint Family and for the *number of distinct values* Constraint Family. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 211–224. Springer-Verlag, 2001. Preprint available as SICS Tech Report T2000-10, `http://soda.swedish-ict.se/2291/2/SICS-T--2000-10--SE.pdf`. 88, 711, 1131, 1686, 1711, 1750, 1799, 1841, 1848, 1861, 1907

[30] N. Beldiceanu. Deriving Filtering Algorithms from Constraints Checkers. Slides presented at the Principles and Practice of Constraint Programming (CP'2004) Conference, September 2004. 1481

[31] N. Beldiceanu, É. Bourreau, P. Chan, and D. Rivreau. Partial Search Strategy in CHIP. In *2nd international conference on metaheuristics - MIC97*, pages 1–8. INRIA & PRiSM-Versailles, July 1997. 365

[32] N. Beldiceanu and M. Carlsson. Revisiting the *cardinality* Operator and Introducing the *cardinality-path* Constraint Family. In P. Codognet, editor, *Int. Conf. on Logic Programming (ICLP'2001)*, volume 2237 of *LNCS*, pages 59–73. Springer-Verlag, 2001. Preprint available as SICS Tech Report T2000-11A, `http://soda.swedish-ict.se/2294/2/SICS-T--2000-11A--SE.pdf`. 651, 823, 851, 2088, 2172, 2191

[33] N. Beldiceanu and M. Carlsson. Sweep as a Generic Pruning Technique Applied to Constraint Relaxation. In *CP'01 3rd Workshop on Soft Constraints (Soft'01)*, pages 43–55, Paphos, Cyprus, 2001. 381

[34] N. Beldiceanu and M. Carlsson. Sweep as a Generic Pruning Technique Applied to the Non-Overlapping Rectangles Constraints. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 377–391. Springer-Verlag, 2001. Preprint available as SICS Tech Report T2001-13, http://soda.swedish-ict.se/2384/2/SICS-T--2001-13--SE.pdf. 380, 383, 1099

[35] N. Beldiceanu and M. Carlsson. A New Multi-Resource *cumulatives* Constraint with Negative Heights. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 63–79. Springer-Verlag, 2002. Preprint available as SICS Tech Report T2001-11, http://soda.swedish-ict.se/2381/2/SICS-T--2001-11--SE.pdf. 1016, 1019

[36] N. Beldiceanu, M. Carlsson, P. Flener, and J. Pearson. On the Reification of Global Constraints. Technical Report T2012-02, Swedish Institute of Computer Science, 2012. 344

[37] N. Beldiceanu, M. Carlsson, P. Flener, and J. Pearson. On the reification of global constraints. *Constraints*, 18(1):1–6, 2013. 11

[38] N. Beldiceanu, M. Carlsson, P. Flener, M. A. F. Rodríguez, and J. Pearson. Linking Prefixes and Suffixes for Constraints Encoded Using Automata with Accumulators. In B. O'Sullivan, editor, *Principles and Practice of Constraint Programming (CP'2014)*, volume 8656 of *LNCS*, pages 142–157, Lyon, France, 2014. Springer-Verlag. 274

[39] N. Beldiceanu, M. Carlsson, and T. Petit. Deriving Filtering Algorithms from Constraint Checkers. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 107–122. Springer-Verlag, 2004. Preprint available as SICS Tech Report T2004-08, soda.swedish-ict.se/2346/1/SICS-T--2004-08--SE.pdf. ii, 6, 88, 192, 359

[40] N. Beldiceanu, M. Carlsson, T. Petit, and J.-C. Régin. An $O(n \log n)$ Bound Consistency Algorithm for the Conjunction of an *alldifferent* and an Inequality between a Sum of Variables and a Constant, and its Generalization. In L. De Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, editors, *ECAI 2012, 20th European Conference on Artificial Intelligence, Including Prestigious Applications of Artificial Intelligence (PAIS-2012), Proceedings*, pages 145–150. IOS Press, August 2012. 548

[41] N. Beldiceanu, M. Carlsson, and E. Poder. New Filtering for the *cumulative* Constraint in the Context of Non-Overlapping Rectangles. In L. Perron and M. A. Trick, editors, *International Conference on Integration of AI and OR*

Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'08), volume 5015 of *LNCS*, pages 21–35, Paris, France, May 2008. Springer-Verlag. 230, 282, 371, 986

[42] N. Beldiceanu, M. Carlsson, E. Poder, R. Sadek, and C. Truchet. A Generic Geometrical Constraint Kernel in Space and Time for Handling Polymorphic *k*-Dimensional Objects. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 180–194. Springer-Verlag, 2007. 326, 371, 382, 1148, 1152, 1292, 1297, 1587, 1872

[43] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global Constraint Catalog. Technical Report T2005-08, Swedish Institute of Computer Science, 2005. Available at http://soda.swedish-ict.se/2366/1/SICS-T--2005-08--SE.pdf. i

[44] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global Constraint Catalog, 2nd Edition. Technical Report T2010-07, Swedish Institute of Computer Science, 2010. Available at http://soda.swedish-ict.se/4041/1/T2010_07.pdf. 723

[45] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global Constraint Catalog, 2nd Edition (revision a). Technical Report T2012-03, Swedish Institute of Computer Science, 2012. Available at http://soda.swedish-ict.se/5195/1/T2012-03.pdf. 724

[46] N. Beldiceanu, M. Carlsson, and S. Thiel. Cost-Filtering Algorithms for the two Sides of the *sum of weights of distinct values* Constraint. Technical Report T2002-14, Swedish Institute of Computer Science, 2002. 711, 1131, 1841, 1848, 1861, 1907, 2346

[47] N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994. 88, 620, 640, 650, 1026, 1085, 1096, 1098, 2363, 2391

[48] N. Beldiceanu, P. Flener, and X. Lorca. The *tree* Constraint. In R. Barták and M. Milano, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, volume 3524 of *LNCS*, pages 64–78, Prague, Czech Republic, May 2005. Springer-Verlag. 2408

[49] N. Beldiceanu, P. Flener, and X. Lorca. Combining tree partitioning, precedence, incomparability, and degree constraints, with an application to phylogenetic and ordered-path problems. Technical Report Technical Report 2006-020, Department of Information Technology, Uppsala University, Sweden, 2006. Available at http://www.it.uu.se/research/publications/reports/2006-020. 2282

[50] N. Beldiceanu, Q. Guo, and S. Thiel. Non-overlapping Constraints between Convex Polytopes. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 392–407. Springer-Verlag, 2001. Preprint available as SICS Tech Report T2001-12, http://soda.swedish-ict.se/2382/2/SICS-T--2001-12--SE.pdf. 207, 1099

[51] N. Beldiceanu, F. Hermenier, X. Lorca, and T. Petit. The *increasing_nvalue* Constraint. In A. Lodi, M. Milano, and P. Toth, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*, volume 6140 of *LNCS*, pages 25–39, Bologna, Italy, 2010. Springer-Verlag. 1433

[52] N. Beldiceanu, I. Katriel, and X. Lorca. Undirected Forest Constraints. In J. C. Beck and B. Smith, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'06)*, volume 3990 of *LNCS*, pages 29–43, Cork, Ireland, May/June 2006. Springer-Verlag. 2080, 2081

[53] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering Algorithms for the *same* and *usedby* Constraints. Technical Report 2004/1/001, MPI, 2004. 2101, 2111, 2449

[54] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering Algorithms for the *same* Constraint. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'04)*, volume 3011 of *LNCS*, pages 65–79. Springer-Verlag, 2004. 5, 2099, 2101, 2111

[55] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering Algorithms for the *same* and *usedby* Constraints. *Archives of Control Sciences, Special Issue on constraint programming for decision and control*, 2005. To appear. 2101, 2449

[56] N. Beldiceanu, I. Katriel, and S. Thiel. GCC-like Restrictions on the *same* Constraint. In B. Faltings, A. Petcu, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints, Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004; Lausanne, Switzerland, June 23-25, 2004, Revised Selected and Invited Papers*, volume 3419 of *LNCS*, pages 1–11. Springer-Verlag, 2005. 2107, 2111

[57] N. Beldiceanu and T. Petit. Cost Evaluation of Soft Global Constraints. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3011 of *LNCS*, pages 80–95. Springer-Verlag, 2004. 5

[58] N. Beldiceanu, T. Petit, and G. Rochart. Bounds of parameters for global constraints. *RAIRO Operations Research*, 40(4):327–353, 2006. 942

[59] N. Beldiceanu and E. Poder. Cumulated Profiles of Minimum and Maximum Resource Utilisation. In *Ninth Int. Conf. on Project Management and Scheduling*, Nancy, France, April 2004. 110

[60] N. Beldiceanu and E. Poder. The *period* Constraint. In B. Demoen, editor, *Int. Conf. on Logic Programming (ICLP'2004)*, LNCS. Springer-Verlag, 2004. 2051, 2053

[61] N. Beldiceanu and H. Simonis. A Constraint Seeker: Finding and Ranking Global Constraints from Examples. In J. H.M. Lee, editor, *Principles and Practice of Constraint Programming (CP'2011)*, volume 6876 of *LNCS*, pages 12–25, Perugia, Italy, 2011. Springer-Verlag. i, 12

[62] N. Beldiceanu and H. Simonis. A Model Seeker: Extracting Global Constraint Models from Positive Examples. In M. Milano, editor, *Principles and Practice of Constraint Programming (CP'2012)*, volume 7514 of *LNCS*, pages 141–157, Quebec City, Canada, 2012. Springer-Verlag. i

[63] C. Berge. *Graphes*. Dunod, 1970. In French. 5, 68, 549

[64] C. Berge. *Hypergraphes, Combinatoire des ensembles finis*. Gauthier-Villars, 1987. In French. 61, 195

[65] C. Bessière. Constraint Propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3. Elsevier, 2006. 169, 170, 201

[66] C. Bessière, R. Coletta, and T. Petit. Acquiring Parameters of Implied Global Constraints. In P. van Beek, editor, *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS*, pages 747–751. Springer-Verlag, 2005. 5

[67] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. *Among*, *common* and *disjoint* Constraints. In M. Carlsson, F. Fages, B. Hnich, and F. Rossi, editors, *Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP 2005)*, pages 223–235, June 2005. 621

[68] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. Filtering Algorithms for the *nvalue* Constraint. In R. Barták and M. Milano, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05)*, volume 3524 of *LNCS*, pages 79–93, Prague, Czech Republic, May 2005. Springer-Verlag. 440, 693, 710, 711, 1907

[69] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. The *range* and *roots* Constraints: Specifying Counting and Occurrence Problems. In *19th Int. Joint Conf. on Artificial Intelligence (IJCAI-05)*, pages 60–65, 2005. 110, 2094, 2466, 2467

[70] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. *Among*, *common* and *disjoint* Constraints. In B. Hnich, M. Carlsson, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints, Joint ERCIM/Colognet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP05; Uppsala, Sweden, June 2005; Revised Selected and Invited Papers*, volume 3978 of *LNAI*, pages 28–43. Springer-Verlag, 2006. 621

[71] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. The Range Constraint: Algorithms and Implementation. In J. C. Beck and B. Smith, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'06)*, volume 3990 of *LNCS*, pages 59–73, Cork, Ireland, May/June 2006. Springer-Verlag. 110

[72] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. The Roots Constraint. In F. Benhamou, editor, *Principles and Practice of Constraint Programming (CP'2006)*, volume 4204 of *LNCS*, pages 75–90, Nantes, France, 2006. Springer-Verlag. 2094

[73] C. Bessière, E. Hebrard, B. Hnich, Z. Kızıltan, and T. Walsh. Range and Roots: Two Common Patterns for Specifying and Propagating Counting and Occurrence Constraints. *Artificial Intelligence*, 173(11):1054–1078, 2009. 2094

[74] C. Bessière, E. Hebrard, B. Hnich, and T. Walsh. *Disjoint*, *partition* and *intersection* Constraints for Set and Multiset Variables. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 138–152. Springer-Verlag, 2004. 709

[75] C. Bessière, E. Hebrard, B. Hnich, and T. Walsh. The Complexity of Global Constraints. In D. L. McGuinness and G. Ferguson, editors, *19th National Conference on Artificial Intelligence (AAAI'04)*, pages 112–117. AAAI Press, 2004. 710, 1906

[76] C. Bessière, E. Hebrard, B. Hnich, and T. Walsh. The Tractability of Global Constraints. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 716–720. Springer-Verlag, 2004. 903, 1306

[77] C. Bessière, E. Hebrard, G. Katsirelos, Z. Kızıltan, É. Picard-Cantin, C.-G. Quimper, and T. Walsh. The balance Constraint Family. In B. O'Sullivan, editor, *Principles and Practice of Constraint Programming (CP'2014)*, volume 8656 of *LNCS*, pages 174–189, Lyon, France, 2014. Springer-Verlag. 494, 495, 496, 724

[78] C. Bessière, G. Katsirelos, N. Narodytska, C.-G. Quimper, and T. Walsh. Decompositions of *all different*, *global cardinality* and related constraints. In *21th Int. Joint Conf. on Artificial Intelligence (IJCAI-09)*, 2009. 551, 1319

[79] C. Bessière, G. Katsirelos, N. Narodytska, C.-G. Quimper, and T. Walsh. Decomposition of the *nvalue* Constraint. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP'2010)*, volume 6308 of *LNCS*, pages 114–128, St Andrews, Scotland, 2010. Springer-Verlag. 1907

[80] C. Bessière, G. Katsirelos, N. Narodytska, C.-G. Quimper, and T. Walsh. Propagating Conjunctions of *alldifferent* Constraints. In *25th National Conference on Artificial Intelligence (AAAI'10)*, pages 1–6. AAAI Press, July 2010. 201, 1527

[81] C. Bessière, G. Katsirelos, N. Narodytska, and T. Walsh. Circuit Complexity and Decompositions of Global Constraints. In *21th Int. Joint Conf. on Artificial Intelligence (IJCAI-09)*, pages 412–418, 2009. 551

[82] C. Bessière, N. Narodytska, C.-G. Quimper, and T. Walsh. The *alldifferent* Constraint with Precedences. In T. Achterberg and J. C. Beck, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'11)*, volume 6697 of *LNCS*, pages 36–52, Berlin, Germany, 2011. Springer-Verlag. 548

[83] O. R. Bininda-Emonds, J. L. Gittleman, and M. A. Steel. The (super)tree of life: Procedures, problems, and prospects. *Annual Reviews of Ecological Systems*, 33:265–289, 2002. 2284

[84] M. Biró. Object-oriented interaction in resource constrained scheduling. *Information Processing Letters*, 36(2):65–67, 1990. 1100

[85] N. Bleuzen-Guernalec and A. Colmerauer. Narrowing a Block of Sortings in Quadratic Time. In G. Smolka, editor, *Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *LNCS*, pages 2–16. Springer-Verlag, 1997. 549, 2269

[86] N. Bleuzen-Guernalec and A. Colmerauer. Optimal Narrowing of a Block of Sortings in Optimal Time. *Constraints*, 5(1/2):85–118, 2000. 2269

[87] M. Bodirsky and M. Kutz. Determining the Consistency of Partial Tree Descriptions. *Artificial Intelligence*, 171(2–3):185–196, 2007. 2285

[88] M. Bohlin. Design and Implementation of a Graph-Based Constraint Model for Local Search. Licentiate Thesis 27, Mälardalen University, 2004. 5

[89] J. Du Boisberranger, D. Gardy, X. Lorca, and C. Truchet. When is it worthwhile to propagate a constraint? A probabilistic analysis of *AllDifferent*. In M. E. Nebel and W. Szpankowski, editors, *Proceedings of the 10th Meeting on Analytic Algorithmics and Combinatorics, ANALCO 2013*, pages 80–90. SIAM, 2013. 243

[90] M. Bomhoff. *Bipartite graphs and the decomposition of systems of equations*. PhD thesis, Twente University, Netherlands, 2013. 549

[91] L. Bordeaux and É. Monfroy. Extraction de programmes à partir de spécifications en logique existentielle du second ordre par compilation de connaissances. In *Programmation en logique avec contraintes, JFPLC 2003*, Revue des sciences et technologies de l'information, pages 189–202. Lavoisier, June 2003. In French. 5

[92] S. Bourdais, P. Galinier, and G. Pesant. HIBISCUS: A Constraint Programming Application to Staff Scheduling in Health Care. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 153–167. Springer-Verlag, 2003. 88, 1306, 1409, 2038

[93] É. Bourreau. *Traitement de contraintes sur les graphes en programmation par contraintes*. PhD thesis, Paris 13 University, France, March 1999. In French. 202, 1028, 1043, 1051

[94] C. J. Bouwkamp and A. J. W. Duijvestijn. Catalogue of Simple Perfect Squared Squares of orders 21 through 25. Technical Report EUT Report 92-WSK-03, Eindhoven University of Technology, Faculty of Mathematics and Computing Science, The Netherlands, November 1992. 370, 1099

[95] S. Brand. The Sum-of-Increments Constraint in the Consecutive-Ones Matrix Decomposition Problem. In S. Y. Shin and S. Ossowski, editors, *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*, pages 1417–1418, Honolulu, Hawaii, USA, 2009. 2340, 2341

[96] S. Brand, N. Narodytska, C.-G. Quimper, P. J. Stuckey, and T. Walsh. Encodings of the *sequence* Constraint. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 210–224. Springer-Verlag, 2007. 651

[97] C. Bron and J. Kerbosch. Finding All Cliques of an Undirected Graph. *Communications of the ACM*, 16(9):575–577, September 1973. 299, 541, 548, 882, 1143

[98] H. Cambazard and B. O'Sullivan. Propagating the Bin Packing Constraint Using Linear Programming. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP'2010)*, volume 6308 of *LNCS*, pages 129–136, St Andrews, Scotland, 2010. Springer-Verlag. 779

[99] A. Caprara, A. Lodi, S. Martello, and M. Monaci. Packing into the smallest square: Worst-case analysis of lower bounds. *Discrete Optimization*, 3:317–326, 2006. 365

[100] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982. 1142

[101] J. Carlier and E. Pinson. Une méthode arborescente pour optimiser la durée d'un job-shop. Technical Report ISSN/0294-2755, Institut de Mathématiques Appliquées, UCO, Angers, 1988. Les Cahiers de L'I.M.A., In French. 1144

[102] J. Carlier and E. Pinson. A Practical Use of Jackson's Preemptive Schedule for Solving the job-shop Problem. *Annals of Operations Research*, 26:269–287, 1990. 1144

[103] M. Carlsson. Filtering for the *case* constraint. Talk given at Advanced School on Global Constraints. Samos, Greece, 2006. 110

[104] M. Carlsson and N. Beldiceanu. Arc-Consistency for a *chain of lexicographic ordering* Constraints. Technical Report T2002-18, Swedish Institute of Computer Science, 2002. 1592, 1593, 1597, 1601, 1604, 1605, 1608, 1609

[105] M. Carlsson and N. Beldiceanu. Revisiting the *lexicographic ordering* Constraint. Technical Report T2002-17, Swedish Institute of Computer Science, 2002. 1621, 1627, 1633, 1639

[106] M. Carlsson and N. Beldiceanu. Dispensation Order Generation for Pyrosequencing. In Yi-Ping Phoebe Chen, editor, *Proceedings of Proceedings of the Second Asia Pacific Bioinformatics Conference (APBC2004)*, volume 29, pages 327–332, Dunedin, New Zealand, January 2004. Australian Computer Society. 1856

[107] M. Carlsson, N. Beldiceanu, and J. Martin. A Geometric Constraint over k-Dimensional Objects and Shapes Subject to Business Rules. In P. J. Stuckey, editor, *Principles and Practice of Constraint Programming (CP'2008)*, volume 5202 of *LNCS*, pages 220–234. Springer-Verlag, 2008. ii, 2, 202, 293, 382, 476

[108] M. Carlsson et al. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, 3.10 edition, January 2003. Available at `http://www.sics.se/sicstus/`. 110, 962, 1098, 1195

[109] M. Carlsson, G. Ottosson, and B. Carlson. An Open-Ended Finite Domain Constraint Solver. In H. Glaser, P. Hartel, and H. Kuchen, editors, *Programming Languages: Implementations, Logics, and Programming (PLILP'97)*, volume 1292 of *LNCS*, pages 191–206, Southampton, 1997. Springer-Verlag. 109, 113

[110] J. M. V. Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659, 1999. 779

[111] J. M. V. Carvalho. LP models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002. 779

[112] Y. Caseau and F. Laburthe. Cumulative Scheduling with Task Intervals. In *Joint International Conference and Symposium on Logic Programming (JICSLP'96)*. MIT Press, 1996. 986

[113] Y. Caseau and F. Laburthe. Solving Small TSPs with Constraints. In Lee Naish, editor, *Fourteenth International Conference on Logic Programming (ICLP'97)*, pages 316–330. MIT Press, 1997. 110

[114] A. Cayley. A theorem on trees. *Quart. J. Math.*, 23:376–378, 1889. 2080

[115] G. Chabert and N. Beldiceanu. Sweeping with Continous Domains. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP'2010)*, volume 6308 of *LNCS*, pages 137–151, St Andrews, Scotland, 2010. Springer-Verlag. 380

[116] G. Chabert and S. Demassey. The Conjunction of Interval Among Constraints. In N. Beldiceanu, N. Jussien, and É. Pinson, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'12)*, volume 7298 of *LNCS*, pages 113–128, Nantes, France, 2012. Springer-Verlag. 621

[117] G. Chabert, L. Jaulin, and X. Lorca. A Constraint on the Number of Distinct Vectors with Application to Localization. In I. P. Gent, editor, *Principles and Practice of Constraint Programming (CP'2009)*, volume 5732 of *LNCS*, pages 196–210. Springer-Verlag, 2009. 360, 1933

[118] G. Chabert and X. Lorca. On the Clique Partition of Rectangle Graphs. Technical Report 09-03-INFO, Mines Nantes, 2009. 1933

[119] W. Chang. A Remark on the Definition of Costas Arrays. *Proceedings of the IEEE*, 75(4):522–523, April 1987. 228

[120] P. Charman. *Gestion des contraintes géométriques pour l'aide à l'aménagement spatial*. PhD thesis, École Nationale des Ponts et Chaussées, France, November 1995. In French. 257

[121] A. M. Cheadle, W. Harvey, A. J. Sadler, J. Schimpf, K. Shen, and M. G. Wallace. ECLiPSe: An Introduction. Technical Report 03-1, IC-Parc, Imperial College London, 2003. 109

[122] K. C. K. Cheng and R. H. C. Yap. Ad-hoc Global Constraints for Life. In P. van Beek, editor, *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS*, pages 182–195. Springer-Verlag, 2005. 110

[123] K. C. K. Cheng and R. H. C. Yap. Applying Ad-hoc Global Constraints with the *case* Constraint to Still-Life. *Constraints*, 11(2–3):91–114, 2006. 110

[124] G. Chu, M. J. García de la Banda, and P. J. Stuckey. Automatically Exploiting Subproblem Equivalence in Constraint Programming. In A. Lodi, M. Milano, and P. Toth, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*, volume 6140 of *LNCS*, pages 71–86, Bologna, Italy, 2010. Springer-Verlag. 5

[125] V. Chvátal. Tough graphs and hamiltonian circuits. *Discrete Mathematics*, 5:215–228, 1973. 857

[126] F. Clautiaux, J. Carlier, and A. Moukrim. A New Exact Method for the Orthogonal Packing Problem. *European Journal of Operational Research*, 183(3):1196–1211, 2007. 393

[127] A. De Clercq. *Ordonnancement cumulatif avec dépassements de capacités: Contrainte globale et décompositions*. PhD thesis, Mines Nantes, France, 2012. In French. 2230

[128] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing - an updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithms Design for Computer System Design*, pages 49–106. Springer-Verlag, New York, 1984. 779

[129] D. Cohen, P. Jeavons, C. Jefferson, K. E. Petrie, and B. Smith. Symmetry Definitions for Constraint Satisfaction Problems. *Constraints*, 11(2-3):115–137, 2006. 27, 160

[130] Mozart Consortium. The Mozart Programming System, version 1.3.2, 2006. Available at `http://www.mozart-oz.org`. 109, 1172

[131] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, fifteenth edition, 1990. 978, 980

[132] M.-C. Costa. Persistency in maximum cardinality bipartite matchings. *Operation Research Letters*, 15:143–149, 1994. 549

[133] COSYTEC. *CHIP Reference Manual*, release 5.1 edition, 1997. 110, 2229

[134] M.-C. Coté, B. Gendron, and L.-M. Rousseau. Modeling the Regular Constraint with Integer Programming. In P. Van Hentenryck and L. Wolsey, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'07)*, volume 4150 of *LNCS*, pages 29–43, Brussels, Belgium, May 2007. Springer-Verlag. ii, 110, 291

[135] X. Cousin. *Application of Constraint Logic Programming on Timetable Problem*. PhD thesis, Rennes I University, France, June 1993. In French. 1486

[136] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977. 162

[137] H. T. Croft, K. J. Falconer, and R. K. Guy. *Unsolved Problems in Geometry*. Springer-Verlag, New York, Berlin, 1991. 370

[138] R. Cymer. Dulmage-Mendelsohn canonical decomposition as a generic pruning technique. *Constraints*, 17(3):234–272, 2012. 549, 583, 959, 1500, 2101, 2223, 2245, 2261, 2379, 2449

[139] R. Cymer. *Applications of Matching Theory in Constraint Programming*. PhD thesis, Hannover University, Germany, 2013. 549, 959, 1500, 2101, 2223, 2245, 2261, 2363, 2369, 2373, 2379, 2396, 2449

[140] R. Cymer. Gallai-Edmonds decomposition as a pruning technique. *Central European Journal of Operations Research*, 2013. 2034, 2081, 2363, 2369, 2373, 2396

[141] M. Dawande, J. Kalagnanam, and J. Sethuraman. Variable Sized Bin Packing with Color Constraints. Technical Report RC21350, IBM Research Report, 1998. 891

[142] R. Dechter and J. Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, 34(1):1–38, 1987. 48

[143] R. Dechter and J. Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence*, 34:1–38, 1988. 279

[144] R. Dechter and J. Pearl. Tree Clustering for Constraint Networks. *Artificial Intelligence*, 38(3):353–366, 1989. 167

[145] S. Demassey, G. Pesant, and L.-M. Rousseau. A *cost-regular* based hybrid column generation approach. *Constraints*, 11(4), 2006. Special issue following CPAIOR'05. 110

[146] T. Denmat, A. Gotlieb, and M. Ducassé. An Abstract Interpretation Model Based Combinator for Modelling While Loops in Constraint Programming. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 241–255. Springer-Verlag, 2007. 1288, 2070

[147] N. Deo. Note on Hopcroft and Tarjan's Planarity Algorithm. *Journal of the Association for Computing Machinery*, 23(1):74–75, January 1976. 857

[148] F. Diedrich, K. Jansen, U. M. Schwarz, and D. Trystram. A Survey on Approximation Algorithms for Scheduling with Machine Unavailability. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics*, volume 5515 of *LNCS*, pages 50–64. Springer-Verlag, 2009. 358

[149] M. Dincbas and H. Simonis. APACHE - A Constraint Based, Automated Stand Allocation System. In *Advanced Software Technology in Air Transport (ASTAIR'91)*, pages 267–282, London, UK, 1991. Royal Aeronautical Society. 1523

[150] M. Dincbas, P. Van Hentenryck, H. Simonis, T. Graf A. Aggoun, and F. Berthier. The Constraint Logic Programming Language CHIP. In *Int. Conf. on Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, Japan, 1988. 109

[151] G. Dooms. *The CP(Graph) Computation Domain in Constraint Programming*. PhD thesis, University Catholic of Louvain, Belgium, September 2006. 15, 17, 109, 796, 797, 944, 1062, 2322, 2360

[152] G. Dooms and I. Katriel. The *minimum spanning tree* Constraint. In F. Benhamou, editor, *Principles and Practice of Constraint Programming (CP'2006)*, volume 4204 of *LNCS*, pages 152–166, Nantes, France, 2006. Springer-Verlag. 110, 2081, 2408

[153] K. A. Dowsland. The three-dimensional pallet chart: An analysis of the factors affecting the set of feasible layouts for a class of two-dimensional packing problems. *Journal of the Operational Research Society*, 35:895–905, 1984. 323

[154] K. A. Dowsland. Determining an upper bound for a class of rectangular packing problems. *Computers & operations research*, 12(22):201–205, 1985. 323

[155] H. E. Dudeney. *The Canterbury Puzzles*. Thomas Nelson & Sons, New York, 1919. 5

[156] A. J. W. Duijvestijn. Simple Perfect Squared Square of Lowest Order. *Journal of Combinatorial Theory*, Series B 25:555–558, 1978. 370

[157] A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10(4):517–534, 1958. 549, 583, 959, 1500, 2379

[158] B. O'Sullivan E. Hebrard, D. Marx and I. Razgon. Constraints of Difference and Equality: A Complete Taxonomic Characterisation. In I. P. Gent, editor, *Principles and Practice of Constraint Programming (CP'2009)*, volume 5732 of *LNCS*, pages 424–438. Springer-Verlag, 2009. 2196, 2202, 2206, 2215, 2223

[159] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. 2081, 2369, 2374

[160] K. M. Elbassioni, M. Kutz I. Katriel, and M. Mahajan. On Systems of *alldifferent* Constraints. Unpublished, Max-Planck-Institut für Informatik, 2005. 1522, 1536, 1552

[161] K. M. Elbassioni, I. Katriel, M. Kutz, and M. Mahajan. Simultaneous Matchings. In Xiaotie Deng and Ding-Zhu Du, editors, *Algorithms and Computation, 16th International Symposium, (ISAAC 2005)*, volume 3827 of *LNCS*, pages 106–115. Springer-Verlag, 2005. 1526

[162] D. Eppstein and D. Strash. Listing All Maximal Cliques in Large Sparse Real-World Graphs. In P. M. Pardalos and S. Rebennack, editors, *10th International Symposium on Experimental Algorithms (SEA 2011)*, volume 6630 of *LNCS*, pages 364–375, Kolimpari, Chania, Crete, Greece, 2011. Springer-Verlag. 548

[163] J. Erschler and P. Lopez. Energy-Based Approach for Task Scheduling under Time and Resources Constraints. In *2nd International Workshop on Project Management and Scheduling*, pages 115–121, Compiégne, France, June 1990. 986

[164] L. Euler. Solution d'une question curieuse qui ne parait soumise à aucune analyse. *Mém. Acad. Sci. Berlin*, 15:310–337, 1759. 5

[165] F. Fages and A. Lal. A Global Constraint for Cutset Problems. In *5th Int. Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'03)*, Montréal, Canada, 2003. 1022

[166] J.-G. Fages. *Exploitation de structures de graphe en programmation par contraintes*. PhD thesis, Mines Nantes, France, 2014. In French. 1507

[167] J.-G. Fages and X. Lorca. Revisiting the *tree* Constraint. In J. H.M. Lee, editor, *Principles and Practice of Constraint Programming (CP'2011)*, volume 6876 of *LNCS*, pages 271–285, Perugia, Italy, 2011. Springer-Verlag. 2408

[168] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *JACM*, 30:514–550, July 1983. 167

[169] H. Fahimi and C.-G. Quimper. Linear-Time Filtering Algorithms for the Disjunctive Constraint. In C. E. Brodley and P. Stone, editors, *28th National Conference on Artificial Intelligence (AAAI'14)*, pages 2637–2643. AAAI Press, July 2014. 1144

[170] T. Fahle. Cost Based Filtering vs. Upper Bounds for Maximum Clique. In *4th Int. Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'02)*, Le Croisic, France, 2002. 882

[171] B.-J. Falkowski and L. Schmitz. A note on the queens' problem. *Information Processing Letters*, 23(1):39–46, 1986. 314

[172] B. Faltings and S. Macho-Gonzalez. Open Constraint Satisfaction. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 356–370. Springer-Verlag, 2002. 319

[173] B. Faltings and S. Macho-Gonzalez. Open Constraint Programming. *Artificial Intelligence*, 161(1–2):181–208, 2005. 319

[174] O. Favaron, M. Mahéo, and J.-F. Saclé. On the residue of a graph. *J. Graph Theory*, 15:39–64, 1991. 440

[175] S. P. Fekete, J. Schepers, and J. van der Veen. An Exact Algorithm for Higher-Dimensional Orthogonal Packing. *Operations Research*, 55(3):569–587, 2007. 350

[176] P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. In DingZhu Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization, Supplement Vol. A*, pages 209–258. Kluwer Academic Publishers, 1999. 306

[177] A. Fink and S. Voss. Applications of modern heuristic search methods to pattern sequencing problems. *Computers and Operations Research*, 26:17–34, 1999. 326, 995

[178] C. Flamm, I. L. Hofacker, and P. F. Stadler. RNA in silico: The computational biology of RNA secondary structures. *Adv. Complex Syst.*, 2:5–90, 1999. 2136

[179] P. Flener, A. M. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, J. Pearson, and T. Walsh. Breaking Row and Column Symmetries in Matrix Models. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 462–476. Springer-Verlag, 2002. 616, 1584, 1644, 2308

[180] P. Flener, J. Pearson, M. Sellmann, P. Van Hentenryck, and M. Ågren. Dynamic structural symmetry breaking for constraint satisfaction problems. *Constraints*, 14(4):506–538, 2009. 27, 160

[181] F. Focacci. *Solving Combinatorial Optimization Problems in Constraint Programming*. PhD thesis, Ferrara University, Italy, 2001. 1122

[182] F. Focacci, A. Lodi, and M. Milano. Cost-Based Domain Filtering. In *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 189–203. Springer-Verlag, 1999. 1820

[183] E. C. Freuder. A Sufficient Condition for Backtrack-Free Search. *J. ACM*, 29(1):24–32, 1982. 279

[184] A. M. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh. Global Constraints for Lexicographic Orderings. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 93–108. Springer-Verlag, 2002. 88, 1621, 1627, 1633, 1639

[185] A. M. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh. Multiset Ordering Constraints. In *18th Int. Joint Conf. on Artificial Intelligence (IJCAI-2003)*, 2003. 1620, 1627, 1632, 1639

[186] T. Frühwirth. Theory and Practice of Constraint Handling Rules, Special Issue on Constraint Logic Programming. *Journal of Logic Programming*, 37(1–3):95–138, October 1998. 1621, 1627, 1633, 1639

[187] T. Frühwirth. Complete Propagation Rules for Lexicographic Order Constraints over Arbitrary Domains. In B. Hnich, M. Carlsson, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints, Joint ERCIM/Colognet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP05; Uppsala, Sweden, June 2005; Revised Selected and Invited Papers*, volume 3978 of *LNAI*, pages 14–28. Springer-Verlag, 2006. 1621, 1627, 1633, 1639

[188] A. G. Frutos, Q. Liu, A. J. Thiel, A. M. W. Sanner, A. E. Condon, L. M. Smith, and R. M. Corn. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research*, 25:4748–4757, 1997. 498, 1088

[189] D. Gale. A theorem on flows in networks. *Pacific J. Math.*, 7:1073–1082, 1957. 2100

[190] T. Gallai. Kritische graphen II. *Publ. Math. Inst. Hungar. Acad. Sci.*, 8:373–395, 1963. 2081, 2369, 2374

[191] I. Gambini. A Method for Cutting Squares into Distinct Squares. *Discrete Applied Mathematics*, 98(1–2):65–80, 1999. 1099

[192] I. Gambini. *Quant aux carrés carrelés*. PhD thesis, Aix-Marseille II University, France, December 1999. In French. 1099

[193] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Softw., Pract. Exper.*, 30(11):1203–1233, 2000. 114, 477

[194] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H.Freeman and co., San Francisco, 1979. 1172

[195] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4:312–316, 1983. 2429

[196] M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2):256–269, 1981. 541

[197] A. Gargani and P. Refalo. An Efficient Model and Strategy for the Steel Mill Slab Design Problem. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 77–89. Springer-Verlag, 2007. 891

[198] H. Gehring, K. Menschner, and M. Meyer. A computer-based heuristic for packing pooled shipment containers. *European Journal of Operational Research*, 44:277–288, 1990. 1100

[199] I. P. Gent, C. Jefferson, and I. Miguel. Minion: A Fast Scalable Constraint Solver. In *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, pages 98–102. IOS Press, 2006. 109

[200] I. P. Gent, I. Miguel, and P. Nightingale. Generalised arc consistency for the Alldifferent constraint: An empirical survey. *Artificial Intelligence*, 172(18):1973–2000, 2008. 549

[201] I. P. Gent and P. Nightingale. A new encoding of the *alldifferent* constraint into SAT. In A. Frisch and I. Miguel, editors, *CP'2004 Third International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 95–110, Toronto, Canada, September 2004. 350

[202] I. P. Gent, P. Prosser, B. M. Smith, and W. Wei. Supertree Construction with Constraint Programming. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 837–841. Springer-Verlag, 2003. 2285

[203] L. Georgiadis. *Linear-Time Algorithms for Dominators and Related Problems*. PhD thesis, Princeton University, USA, 2005. 1172

[204] M. L. Ginsberg and W. D. Harvey. Limited Discrepancy Search. In C. S. Mellish, editor, *14th Int. Joint Conf. on Artificial Intelligence (IJCAI-95)*, volume 1, pages 607–615. Morgan Kaufmann, 1995. 291, 1122

[205] F. Glover. Maximum matchings in convex bipartite graphs. *Naval Res. Logist. Quart.*, 14:313–316, 1967. 199, 549

[206] S. W. Golomb. *Polyominoes*. Scribners, New York, 1965. 2066

[207] S. W. Golomb. How to Number a Graph. In R. C. Read, editor, *Graph Theory and Computing*, pages 23–37. Academic Press, New York, 1972. 1344

[208] M. Gondran and M. Minoux. *Graphs and Algorithms*. Wiley, New York, 2nd revised edition, 1984. 68

[209] S. Grandcolas and C. Pinto. A SAT Encoding for Multi-dimensional Packing Problems. In A. Lodi, M. Milano, and P. Toth, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*, volume 6140 of *LNCS*, pages 141–146, Bologna, Italy, 2010. Springer-Verlag. 350

[210] E. Ya. Grinberg. Plane homogeneous graphs of degree three without hamiltonian circuits. *Latv. Mat. Ezhegodnik*, 4:51–58, 1968. In Russian. 857

[211] P. Hansen. Degrés et nombre de stabilité d'un graphe. *Cahiers du Centre d'Études de Recherche Opérationnelle*, 17:213–220, 1975. 440

[212] F. Harary and A. J. Schwenk. A new crossing number for bipartite graphs. *Utilitas Math.*, 1:203–209, 1972. 2428, 2430

[213] G. H. Hardy and E. M. Wright. *Theory of numbers*. Oxford Univ. Press, Oxford, 1975. 1184

[214] W. Harvey and J. Schimpf. Bounds Consistency Techniques for Long Linear Constraints. In *CP'02 Workshop on Techniques foR Implementing Constraint programming Systems (TRICS)*, pages 39–46, 2002. 2135

[215] E. Hebrard, B. Hnich, B. O'Sullivan, and T. Walsh. Finding diverse and similar solutions in constraint programming. In M. Veloso and S. Kambhampati, editors, *20th National Conference on Artificial Intelligence (AAAI'05)*, pages 372–377. AAAI Press, July 2005. 239

[216] E. Hebrard, B. O'Sullivan, and I. Razgon. A Soft Constraint of Equality: Complexity and Approximability. In P. J. Stuckey, editor, *Principles and Practice of Constraint Programming (CP'2008)*, volume 5202 of *LNCS*, pages 358–371. Springer-Verlag, 2008. 2202

[217] S. Heinz, T. Schlechte, R. Stephan, and M. Winkler. Solving steel mill slab design problems. *Constraints*, 17(1):39–50, 2012. 891

[218] P. Hell and A. Rosa. Graph decomposition, handcuffed prisoners and balanced *p*-designs. *Discrete Mathematic*, 2(3):229–252, 1972. 1703

[219] L. Hellsten. Consistency propagation for *stretch* constraints. Master's thesis, Waterloo University, 2004. 823, 2191, 2293, 2297

[220] L. Hellsten, G. Pesant, and P. van Beek. A Domain Consistency Algorithm for the *stretch* Constraint. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 290–304. Springer-Verlag, 2004. 2293, 2297

[221] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic Second-Order Logic in Practice. In E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS'95, Aarhus, Denmark*, volume 1019 of *LNCS*, pages 89–110. Springer-Verlag, May 1995. 5

[222] M. Henz, T. Müller, and S. Thiel. Global Constraints for Round Robin Tournament Scheduling. *European Journal of Operations Research*, 153(1):92–101, Feb 2004. 2363

[223] B. M. Hernández. *The Systematic Generation of Channelled Models in Constraint Satisfaction*. PhD thesis, University of York, York, YO10 5DD, UK, Department of Computer Science, 2007. 205

[224] S. Hoda, W.-J. van Hoeve, and J. N. Hooker. A Systematic Approach to MDD-Based Constraint Programming. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP'2010)*, volume 6308 of *LNCS*, pages 266–280, St Andrews, Scotland, 2010. Springer-Verlag. 5

[225] J. N. Hooker. Good and Bad Futures for Constraint Programming (and Operations Research). *Constraint Programming Letters*, 1:21–32, November 2007. i

[226] J. N. Hooker. *Integrated Methods for Optimization*. Springer Science + Business Media, LLC, New York, 2007. 291, 548, 549, 857, 1143, 1319

[227] J. N. Hooker and H. Yan. A Relaxation for the *cumulative* Constraint. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 686–690. Springer-Verlag, 2002. Available at `http://ba.gsia.cmu.edu/jnh/papers.html`. 5, 986

[228] J. Hopcroft and R. E. Tarjan. Efficient Planarity Testing. *Journal of the Association for Computing Machinery*, 21(4):549–568, October 1974. 857

[229] E. Hopper. *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. PhD thesis, Cardiff University, UK, 2000. 375

[230] S. Hougardy. Personal communication, June 8, 2012. 325

[231] F. Le Huédé, M. Grabisch, C. Labreuche, and P. Savéant. Integration and propagation of a multi-criteria decision making model in constraint programming. *Journal of Heuristics*, 12(4–5):329–346, September 2006. 110, 408

[232] M. Hujter. On the dynamic storage allocation problem. Technical report, Manuscript, Computer and Automation Institute, Hungarian Acad. Sci., 1990. 1100

[233] H. Isermann. Obere Schranken für die Lösung des zweidimensionalen Packproblems auf der Basis struktureller Identitäten. In Fandel and Gehring, editors, *Operations Research – Beiträge zur quantitativen Wirtschaftforschung*, pages 341–348. Springer-Verlag, 1991. 323

[234] G. F. Italiano, L. Laura, and F. Santaroni. Finding Strong Bridges and Strong Articulation Points in Linear Time. In W. Wu and O. Daescu, editors, *Combinatorial Optimization and Applications - 4th International Conference, COCOA 2010, Proceedings, Part I*, volume 6508 of *LNCS*, pages 157–169, Kailua-Kona, Hawaii, USA, 2010. Springer-Verlag. 376, 377, 2408

[235] J. Jackson. Rational Amusements for Winter Evenings. Longman, London, 1821. 1986

[236] J. R. Jackson. Scheduling a Production Line to Minimize Maximum Tardiness. Technical Report 43, University of California, Los Angeles, 1955. Management Science Research Project. 1144

[237] L. Jaulin. Localization of an Underwater Robot using Interval Constraint Programming. In F. Benhamou, editor, *Principles and Practice of Constraint Programming (CP'2006)*, volume 4204 of *LNCS*, pages 244–255, Nantes, France, 2006. Springer-Verlag. 360

[238] L. R. Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962. 887

[239] N. Jussien and V. Barichard. The PALM system: explanation-based constraint programming. In *CP'00 Workshop on Techniques foR Implementing Constraint programming Systems (TRICS)*, pages 118–133, 2000. 550

[240] L. V. Kalè. An Almost Perfect Heuristic for the N-queens Problem. *Information Processing Letters*, 34(4):173–178, April 1990. 314

[241] R. Kameugne, L. P. Fotso, J. Scott, and Y. Ngo-Kateu. A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints. In J. H.M. Lee, editor, *Principles and Practice of Constraint Programming (CP'2011)*, volume 6876 of *LNCS*, pages 478–492, Perugia, Italy, 2011. Springer-Verlag. 986

[242] I. Katriel. *Constraints and Changes*. PhD thesis, Saarlandes University, Germany, 2005. 1307, 2101, 2449

[243] I. Katriel. Expected-Case Analysis for Delayed Filtering. In J. C. Beck and B. Smith, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'06)*, volume 3990 of *LNCS*, pages 119–125, Cork, Ireland, May/June 2006. Springer-Verlag. 243

[244] I. Katriel and S. Thiel. Fast Bound Consistency for the *global cardinality* Constraint. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 437–451. Springer-Verlag, 2003. 5, 1307

[245] I. Katriel and S. Thiel. Complete Bound Consistency for the Global Cardinality Constraint. *Constraints*, 10(3), 2005. 1307

[246] I. Katriel and P. Van Hentenryck. Randomized Filtering Algorithms. Technical Report CS-06-09, Department of Computer Science, Brown University, 2006. 243

[247] L. G. Kaya and J. N. Hooker. A Filter for the *circuit* Constraint. In F. Benhamou, editor, *Principles and Practice of Constraint Programming (CP'2006)*, volume 4204 of *LNCS*, pages 706–710, Nantes, France, 2006. Springer-Verlag. 857

[248] R. Keber. Stauraumprobleme bei Stückguttransporten. Technical Report Heft 17, Wissenschaftliche Berichte des Institutes für Fördertechnik der Universität Karlsruhe, 1985. 323

[249] T. P. Kirkman. On a Problem in Combinatorics. *Cambridge and Dublin Math. J.*, 2:191–204, 1847. 5

[250] Z. Kızıltan. *Symmetry Breaking Ordering Constraints*. PhD thesis, Uppsala University, Sweden, March 2004. 1621, 1627, 1633, 1639

[251] Z. Kızıltan and T. Walsh. Constraint Programming with Multisets. In *Second International Workshop on Symmetry on Constraint Satisfaction Problems (SymCon-02)*, 2002. held along CP-2002. 2100

[252] W. Kocjan and P. Kreuger. Filtering Methods for *symmetric cardinality* Constraint. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'04)*, volume 3011 of *LNCS*, pages 200–208. Springer-Verlag, 2004. 2379, 2383

[253] R. E. Korf. Optimal Rectangle Packing: New Results. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, pages 142–149, 2004. 365

[254] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quaterly*, 2(1–2):83–97, March–June 1955. 283, 1821

[255] M. Labbé, G. Laporte, and I. Rodríguez-Martín. Path, tree and cycle location. In *Fleet Management and Logistics*, pages 187–204. Kluwer Academic Publishers, 1998. 1046, 1047, 2391

[256] F. Laburthe. Choco: Implementing a CP Kernel. In *CP'00 Workshop on Techniques foR Implementing Constraint programming Systems (TRICS)*, 2000. 109, 113

[257] M. Z. Lagerkvist and G. Pesant. Modeling Irregular Shape Placement Problems with Regular Constraints. In *First Workshop on Bin Packing and Placement Constraints BPPC'08, associated to CPAIOR08*, 2008. 326

[258] M. Z. Lagerkvist and C. Schulte. Propagator Groups. In I. P. Gent, editor, *Principles and Practice of Constraint Programming (CP'2009)*, volume 5732 of *LNCS*, pages 524–538. Springer-Verlag, 2009. 196

[259] A. Lahrichi. *Ordonnancements: la notion de partie obligatoire et son application aux problèmes cumulatifs*. PhD thesis, Paris 6 University, France, May 1979. In French. 207

[260] A. Lahrichi. Ordonnancements: la notion de partie obligatoire et son application aux problèmes cumulatifs. *RAIRO-Recherche Opérationnelle*, 16(3):241–262, 1982. In French. 207

[261] A. Lahrichi. Scheduling: the Notions of Hump, Compulsory Parts and their Use in Cumulative Problems. *C.R. Acad. Sci., Paris*, 294:209–211, February 1982. 207, 985, 997, 1007, 1143

[262] A. Lahrichi and M. Gondran. Théorie des parties obligatoires et découpes à deux dimensions. Technical Report HI/4762-02, Électricité de France (EDF), January 1984. In French. 207

[263] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah. Some Applications of the Generalized Travelling Salesman Problem. *J. of the Operational Research Society*, 47:1461–1467, 1996. 864, 865

[264] F. Lardeux, E. Monfroy, and F. Saubion. Interleaved *alldifferent* Constraints: CSP vs. SAT Approaches. In D. Dochev, M. Pistore, and P. Traverso, editors, *Artificial Intelligence: Methodology, Systems, and Applications, 13th International Conference, AIMSA 2008*, volume 5253 of *LNCS*, pages 380–384. Springer-Verlag, 2008. 1527

[265] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4):403–422, August 2002. 2325

[266] J.-L. Laurière. *Un langage et un programme pour énoncer et résoudre des problèmes combinatoires*. Thèse de doctorat d'état, Paris 6 University, France, May 1976. In French. 548

[267] J.-L. Laurière. A Language and a Program for Stating and Solving Combinatorial Problems. *Artificial Intelligence*, 10(1):29–127, 1978. ii, 108, 546, 548, 856, 1028

[268] Y. C. Law. *Using Constraints to Break Value Symmetries in Constraint Satisfaction Problems*. PhD thesis, The Chinese University of Hong Kong, September 2005. 286, 2142

[269] Y. C. Law and J. H.M. Lee. Global Constraints for Integer and Set Value Precedence. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 362–376. Springer-Verlag, 2004. 1476, 1480, 2142

[270] M. Leconte. A Bounds-Based Reduction Scheme for Constraints of Difference. In *CP'96, Second International Workshop on Constraint-based Reasoning*, pages 19–28, Key West, FL, USA, 1996. 549, 550

[271] H. Levy and D. W. Low. A contraction algorithm for finding small cycle cutsets. *J. of Algorithms*, 9:470–493, 1988. 1022

[272] K. Li and K.-H. Cheng. On Three-Dimensional Packing. *SIAM Journal on Computing*, 19(5):847–867, 1990. 1100

[273] C. C. Lindner and A. Rosa. Topics on steiner systems. *Annals of Discrete Mathematic*, 7, 1980. 1703

[274] A. Linhares and H. H. Yanasse. Connections between cutting-pattern sequencing, VLSI design and flexible machines. *Computers and Operations Research*, 29:1759–1772, 2002. 995

[275] E. L. Lloyd, M. L. Soffa, and C. C. Wang. On locating minimum feedback vertex sets. *J. of Computer and System Science*, 37:292–311, 1988. 1022

[276] H. C. R. Lock. An Implementation of the *cumulatives* Constraint. Technical report, University of Karlsruhe, 1996. Working paper. 986

[277] A. Lopez-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. A Fast and Simple Algorithm for Bounds Consistency of the *alldifferent* Constraint. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'2003)*, pages 245–250, 2003. 549, 550

[278] A. Lubiw. Doubly Lexical Orderings of Matrices. In *Proceedings of the 17th Annual Association for Computing Machinery Symposium on Theory of Computing (STOC-85)*, pages 396–404. ACM Press, 1985. 1584

[279] A. Lubiw. Doubly Lexical Orderings of Matrices. *SIAM Journal on Computing*, 16(5):854–879, October 1987. 1584

[280] É. Lucas. *Récréations mathématiques*, volume 1-2. Gauthier-Villars, 1882. 5, 506

[281] R. Maculet. *Représentation des connaissances spatiales (algèbre de Manhattan et raisonnement spatial avec contraintes*. PhD thesis, Paris VI University, France, 1991. In French. 257, 258

[282] M. J. Maher. Analysis of a *global contiguity* Constraint. In *Workshop on Rule-Based Constraint Reasoning and Programming*, 2002. held along CP-2002. 88, 1338, 1339

[283] M. J. Maher. Open Contractible Global Constraints. In *21th Int. Joint Conf. on Artificial Intelligence (IJCAI-09)*, pages 578–583, 2009. 219, 319

[284] M. J. Maher, N. Narodytska, C.-G. Quimper, and T. Walsh. Flow-Based Propagators for the *sequence* and Related Global Constraints. In P. J. Stuckey, editor, *Principles and Practice of Constraint Programming (CP'2008)*, volume 5202 of *LNCS*, pages 159–174. Springer-Verlag, 2008. 651, 2172

[285] M. Marte. A Global Constraint for Parallelizing the Execution of Task Sets in Non-Preemptive Scheduling. In *CP'2001 Doctoral Programme*, 2001. 2400

[286] S. Martello and P. Toth. *Knapsack problems. Algorithms and Computer Implementations*. Interscience Series in Discrete Mathematics and Optimization. Wiley, 1990. 779

[287] P. Martin and D. B. Shmoys. A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3-5, 1996, Proceedings*, volume 1084 of *LNCS*, pages 389–403. Springer-Verlag, 1996. 1523

[288] J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphism. *Inf. Sci.*, 19(3):229–250, 1979. 1352, 2324

[289] B. Medjdoub. *Méthodes de conception fonctionnelle en architecture: une approche CAO basée sur des contraintes: ARCHiPLAN*. PhD thesis, École Centrale de Paris, France, May 1996. In French. 257

[290] B. Medjdoub and B. Yannou. Separating topology and geometry in space planning. *Computer-aided design*, 32(1):39–61, 2000. 257

[291] K. Mehlhorn. Constraint Programming and Graph Algorithms. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *27th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, volume 1853 of *LNCS*, pages 571–575. Springer-Verlag, 2000. 5

[292] K. Mehlhorn and S. Thiel. Faster Algorithms for Bound-Consistency of the *sortedness* and the *alldifferent* Constraint. In *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS*, pages 306–319. Springer-Verlag, 2000. 5, 549, 550, 2269

[293] J. Menana and S. Demassey. Sequencing and Counting with the *multicost-regular* Constraint. In W.-J. van Hoeve and J. N. Hooker, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *LNCS*, pages 178–192, Pittsburgh, PA, USA, 2009. Springer-Verlag. 110

[294] L. Mercier and P. Van Hentenryck. Edge finding for cumulative scheduling. *Informs Journal of Computing*, 20:143–153, 2008. 986

[295] J.-P. Métivier. *Relaxation de contraintes globales : Mise en oeuvre et Application*. PhD thesis, Caen University, France, 2010. In French. 110

[296] S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction Problems. In *8th National Conference on Artificial Intelligence (AAAI-90)*, pages 25–32. AAAI Press, 1990. 2158

[297] M. Mohri. Weighted automata algorithms. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science, pages 213–254. Springer, 2009. 348

[298] J.-N. Monette, Y. Deville, and P. Dupont. A Position-Based Propagator for the Open-Shop Problem. In P. Van Hentenryck and L. Wolsey, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'07)*, volume 4150 of *LNCS*, pages 186–199, Brussels, Belgium, May 2007. Springer-Verlag. 1126, 1127

[299] M. Müller-Hannemann, W. Stille, and K. Weihe. Evaluating the Bin-Packing Constraint, Part i: Overview of the Algorithmic Approach. Technical report, TU Darmstadt, 2003. 779

[300] M. Müller-Hannemann, W. Stille, and K. Weihe. Evaluating the Bin-Packing Constraint, Part ii: An Adaptive Rounding Problem. Technical report, TU Darmstadt, 2003. 779

[301] M. Müller-Hannemann, W. Stille, and K. Weihe. Evaluating the Bin-Packing Constraint, Part iii: Joint Evaluation with Concave Constraints. Technical report, TU Darmstadt, 2003. 779

[302] M. Müller-Hannemann, W. Stille, and K. Weihe. Patterns of Usage for Global Constraints: A Case Study Based on the Bin-Packing Constraint. Technical report, TU Darmstadt, 2003. 779

[303] J. Nelissen. *Neue Ansätze zur Lösung des Palettenbeladungsproblems*. PhD thesis, RWTH Aachen, Germany, 1994. In German. 323, 1099, 1100

[304] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. MiniZinc: Towards a Standard CP Modelling Language. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 529–543. Springer-Verlag, 2007. 476

[305] M. P. Ng and N. C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69:19–31, 1996. 2284, 2285

[306] R. Nieuwenhuis. SAT Modulo Theories: Getting the Best of SAT and Global Constraint Filtering. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP'2010)*, volume 6308 of *LNCS*, pages 1–2, St Andrews, Scotland, 2010. Springer-Verlag. 5

[307] Organizing Committee of the Third International Competition of CSP Solvers. XML representation of constraint networks format XCSP 2.1. Available at http://www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf, 2008. 476

[308] W. J. Older, G. M. Swinkels, and M. H. van Emden. Getting to the Real Problem: Experience with BNR Prolog in OR. In *3rd Int. Conf. on the Practical Application of Prolog (PAP'95)*, pages 465–478. Alinmead Software Ltd., 1995. 2100, 2268, 2269

[309] A. Oplobedu, J. Marcovitch, and Y. Tourbier. CHARME: Un langage industriel de programmation par contraintes, illustré par une application chez Renault. In *Proceedings of the Ninth International Workshop on Expert Systems and their Applications: General Conference*, volume 1, pages 55–70, 1989. 109, 1304

[310] P. Östergård and W. Weakley. Values of domination numbers of the queen's graph. *The Electronic Journal of Combinatorics*, 8(1), 2001. 247

[311] G. Ottosson and E. S. Thorsteinsson. Linear Relaxations and Reduced-cost Based Propagation of Continuous Variable Subscripts. In *Second International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, March 2000. 1218

[312] G. Ottosson, E. S. Thorsteinsson, and J. N. Hooker. Mixed Global Constraints and Inference in Hybrid IP-CLP Solvers. In *CP'99 Post-Conference Workshop on Large-Scale Combinatorial Optimization and Constraints*, pages 57–78, 1999. 1206, 1210

[313] P. Ouellet and C.-G. Quimper. The Multi-Inter-Distance Constraint. In *22th Int. Joint Conf. on Artificial Intelligence (IJCAI-11)*, 2011. 1830, 1831

[314] F. Pachet and P. Roy. Automatic Generation of Music Programs. In *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 331–345. Springer-Verlag, 1999. 823, 1904, 1906

[315] L. Péridy and D. Rivreau. An $O(n \log n)$ Stable Algorithm for Immediate Selections Adjustments. In G. Kendall, E. K. Burke, S. Petrovic, and M. Gendreau, editors, *Multidisciplinary Scheduling: Theory and Applications, 1st International Conference Selected Papers*, pages 205–222. Springer-Verlag, 2005. 1144

[316] G. Pesant. A Filtering Algorithm for the *stretch* Constraint. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 183–195. Springer-Verlag, 2001. 823, 1433, 2191, 2292, 2293, 2296, 2297

[317] G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 482–495. Springer-Verlag, 2004. ii, 110, 192, 827, 2039, 2194

[318] G. Pesant and J.-C. Régin. SPREAD: A Balancing Constraint Based on Statistics. In P. van Beek, editor, *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS*, pages 460–474. Springer-Verlag, 2005. 110, 194

[319] G. Pesant and P. Soriano. An Optimal Strategy for the Constrained Cycle Cover Problem. CRT Pub. 98-45, CRT, Montréal, December 1998. 2363

[320] J. Petersen. Die Theorie der Regularen Graphs. *Acta Mathematica*, 15:193–220, 1891. 549

[321] T. Petit, N. Beldiceanu, and X. Lorca. A Generalized Arc-Consistency Algorithm for a Class of Counting Constraints. In *22th Int. Joint Conf. on Artificial Intelligence (IJCAI-11)*, 2011. 823, 1433, 2191

[322] T. Petit and E. Poder. The Soft Cumulative Constraint. *CoRR*, abs/0907.0939, 2009. 2230

[323] T. Petit and J.-C. Régin. The Ordered Global Cardinality Constraint. Technical Report 09-07-INFO, Mines Nantes, 2009. 2000, 2001

[324] T. Petit, J.-C. Régin, and N. Beldiceanu. A $\theta(n)$ Bound-Consistency Algorithm for the *increasing sum* Constraint. In J. H.M. Lee, editor, *Principles and Practice of Constraint Programming (CP'2011)*, volume 6876 of *LNCS*, pages 721–728, Perugia, Italy, 2011. Springer-Verlag. 1450, 1451

[325] T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithms for Over-Constrained Problems. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP'2001)*, volume 2239 of *LNCS*, pages 451–463. Springer-Verlag, 2001. 5, 2214, 2215, 2222, 2223

[326] C. E. Pfefferkorn. A Heuristic Problem Solving Design System for Equipment or Furniture Layouts. *Communications of the ACM*, 18(5):286–297, May 1975. 257

[327] J. Pitrat. MALICE, notre collègue. In *Colloque Métaconnaissance de Berder*, pages 4–19, September 2001. In French. iii, 1306

[328] J. Pitrat. A Step toward an Artificial Artificial Intelligence Scientist. Technical report, Paris VI University, France, 2008. iii, 158, 294, 1306

[329] J. Pitrat. *Artificial Beings*. Wiley InterScience, April 2009. iii

[330] E. Poder. *Programmation par contraintes et ordonnancement de tâches avec consommation variable de ressource*. PhD thesis, Blaise-Pascal - Clermont II University, France, October 2002. In French. 207

[331] E. Poder, N. Beldiceanu, and E. Sanlaville. Computing a Lower Approximation of the Compulsory Part of a Task with Varying Duration and Varying Resource Consumption. *European Journal of Operational Research*, 153:239–254, 2004. 110, 207, 208

[332] C. Pralet and G. Verfaillie. Slice Encoding for Constraint-based Planning. In I. P. Gent, editor, *Principles and Practice of Constraint Programming (CP'2009)*, volume 5732 of *LNCS*, pages 669–683. Springer-Verlag, 2009. 359

[333] F. P. Preparata and M. I. Shamos. *Computational Geometry. An Introduction.* Springer-Verlag, New York, Berlin, 1985. Corrected and Expanded Second Printing. 379

[334] J.-F. Puget. A C++ Implementation of CLP. In *Second Singapore International Conference on Intelligent Systems (SPICIS)*, pages 256–261, Singapore, November 1994. 109

[335] J.-F. Puget. A Fast Algorithm for the Bound Consistency of *alldiff* Constraints. In *15th National Conference on Artificial Intelligence (AAAI-98)*, pages 359–366. AAAI Press, 1998. 549, 550

[336] J.-F. Puget. Using Constraint Programming to Compute Symmetries. In *Third International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon-03)*, 2003. held along CP-2003. 2325

[337] J.-F. Puget. Automatic Detection of Variable and Value Symmetries. In P. van Beek, editor, *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS*, pages 477–489. Springer-Verlag, 2005. 2325

[338] J.-F. Puget. Symmetry Breaking Revisited. *Constraints*, 10(1):23–46, 2005. 2325

[339] L. O. Quesada. *Solving Constrained Graph Problems using Reachability Constraints based on Transitive Closure and Dominators*. PhD thesis, University catholic of Louvain, Belgium, November 2006. 1172

[340] L. O. Quesada, P. Van Roy, and Y. Deville. Reachability: a constrained path propagator implemented as a multi-agent system. In *CLEI2005*, 2005. 1172

[341] L. O. Quesada, P. Van Roy, Y. Deville, and R. Collet. Using Dominators for Solving Constrained Path Problems. In P. Van Hentenryck, editor, *Practical Aspects of Declarative Languages, 8th International Symposium*, volume 3819 of *LNCS*, pages 73–87, Charleston, SC, USA, January 2006. Springer-Verlag. 1170

[342] C. Quimper. Enforcing domain consistency on the extended global cardinality constraint is NP-hard. Technical Report TR CS-2003-39, School of Computer Science, University of Waterloo, 2003. 1305

[343] C.-G. Quimper, A. López-Ortiz, and G. Pesant. A Quadratic Propagator for the Inter-Distance Constraint. In *21th National Conference on Artificial Intelligence (AAAI'06)*. AAAI Press, 2006. 541

[344] C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. Improved Algorithms for the *global cardinality* Constraint. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 542–556. Springer-Verlag, 2004. 5, 1307

[345] C.-G. Quimper, P. van Beek, A. López-Ortiz, A. Golynski, and S. B. Sadjad. An Efficient Bounds Consistency Algorithm for the *global cardinality* Constraint. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 600–614. Springer-Verlag, 2003. 1307

[346] C.-G. Quimper and T. Walsh. Beyond Finite Domains: The All Different and Global Cardinality Constraints. In P. van Beek, editor, *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS*, pages 812–816. Springer-Verlag, 2005. 110, 570, 1587

[347] C.-G. Quimper and T. Walsh. Beyond Finite Domains: the All Different and Global Cardinality Constraints. In F. Azevedo, C. Gervet, and E. Pontelli, editors, *Constraint Programming Beyond Finite Integer Domains (CP'2005)*, pages 5–17, October 2005. 110, 570, 1587

[348] C.-G. Quimper and T. Walsh. The All Different and Global Cardinality Constraints on Set, Multiset and Tuple Variables. In B. Hnich, M. Carlsson, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints, Joint ERCIM/Colognet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP05; Uppsala, Sweden, June 2005; Revised Selected and Invited Papers*, volume 3978 of *LNAI*, pages 1–13. Springer-Verlag, 2006. 570, 1587

[349] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In B. Nebel, C. Rich, and W. R. Swartout, editors, *Proc. of 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 165–176. Morgan Kaufmann, 1992. 339, 954, 955, 970, 971, 974, 975, 1134, 1135, 1252, 1253, 1472, 1473, 1722, 1723, 2022, 2023

[350] P. Refalo. Linear Formulation of Constraint Programming Models and Hybrid Solvers. In *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS*. Springer-Verlag, 2000. 1176

[351] J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSP. In *12th National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, 1994. 5, 8, 88, 549, 550

[352] J.-C. Régin. *Développement d'outils algorithmiques pour l'Intelligence Artificielle*. PhD thesis, Montpellier II University, France, 1995. In French. 692, 693, 1906, 2325

[353] J.-C. Régin. Generalized Arc Consistency for *global cardinality* Constraint. In *14th National Conference on Artificial Intelligence (AAAI-96)*, pages 209–215, 1996. 5, 811, 815, 1305, 1307, 1319, 1323, 1327, 1952, 1965, 1969

[354] J.-C. Régin. The Global *minimum distance* Constraint. Technical report, ILOG, 1997. 540

[355] J.-C. Régin. Arc Consistency for Global Cardinality Constraints with Costs. In J. Jaffar, editor, *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 390–404. Springer-Verlag, 1999. 1330, 1332

[356] J.-C. Régin. The Symmetric *alldiff* Constraint. In *16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, pages 420–425, 1999. 5, 2362, 2363

[357] J.-C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7(3–4):387–405, 2002. 1332

[358] J.-C. Régin. Solving the Maximum Clique Problem with Constraint Programming. In *Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'03)*, pages 166–179, Montréal, Canada, May 2003. 882

[359] J.-C. Régin. Using Constraint Programming to Solve the Maximum Clique Problem. In F. Rossi, editor, *Principles and Practice of Constraint Programming (CP'2003)*, volume 2833 of *LNCS*, pages 634–648. Springer-Verlag, 2003. 882

[360] J.-C. Régin. Simpler and Incremental Consistency Checking and Arc Consistency Filtering Algorithms for the *weighted spanning tree* Constraint. In L. Perron and M. A. Trick, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'08)*, volume 5015 of *LNCS*, pages 233–247, Paris, France, 2008. Springer-Verlag. 110, 2081, 2408

[361] J.-C. Régin and C. Gomes. The *cardinality matrix* Constraint. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 572–587. Springer-Verlag, 2004. 887, 1306

[362] J.-C. Régin and J.-F. Puget. A Filtering Algorithm for Global Sequencing Constraints. In G. Smolka, editor, *Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *LNCS*, pages 32–46. Springer-Verlag, 1997. 2168

[363] J.-C. Régin, L.-M. Rousseau, M. Rueher, and W.-J. van Hoeve. The Weighted Spanning Tree Constraint Revisited. In A. Lodi, M. Milano, and P. Toth, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*, volume 6140 of *LNCS*, pages 287–291, Bologna, Italy, 2010. Springer-Verlag. 2081, 2408

[364] J.-C. Régin and M. Rueher. A Global Constraint Combining a *sum* Constraint and Binary Inequalities. In *IJCAI-99 Workshop on Non Binary Constraints*, 1999. 5

[365] J.-C. Régin and M. Rueher. A Global Constraint Combining a Sum Constraint and Difference Constraints. In R. Dechter, editor, *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS*, pages 384–395. Springer-Verlag, 2000. 110, 2332

[366] J.-C. Régin and M. Rueher. *inequality-sum*: A Global Constraint Capturing the Objective Function. *RAIRO Operations Research*, 39(2):123–139, 2005. 110, 2332

[367] C. Ribeiro and M. A. Carravilla. A Global Constraint for Nesting Problems. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'04)*, volume 3011 of *LNCS*, pages 256–270. Springer-Verlag, 2004. 1099

[368] Y. Richter, A. Freund, and Y. Naveh. Generalizing *alldifferent*: the *somedifferent* Constraint. In F. Benhamou, editor, *Principles and Practice of Constraint Programming (CP'2006)*, volume 4204 of *LNCS*, pages 468–483, Nantes, France, 2006. Springer-Verlag. 1527

[369] J. A. Roach. The Rectangle Placement Language. Technical Report 08903, Laboratory for Computer Science Research, Hill Center for the Mathematical Sciences, Busch Campus, Rutgers University, New Brunswick, New Jersey, September 1985. 2432

[370] G. Rochart. *Explications et programmation par contraintes avancée*. PhD thesis, Nantes University, France, 2005. In French. 550

[371] G. Rochart and N. Jussien. Explanations for global constraints: instrumenting the *stretch* constraint. Technical Report 03-01-INFO, Mines Nantes, 2003. 5, 2293

[372] F. Rochon du Verdier. *Résolution de problèmes d'aménagement spatial fondée sur la satisfaction de contraintes. Validation sur l'implantation d'équipements électroniques hyperfréquences*. PhD thesis, Lyon I University, France, July 1992. In French. 1099

[373] L. Roditty. A faster and simpler fully dynamic transitive closure algorithm. In *SODA'03: Proceedings of the fourteenth anuual ACM-SIAM symposium on Discrete algorithms*, pages 404–412. Springer-Verlag, 2003. 1172

[374] A. Rosa and C. Huang. Complete Classification of Solutions to the Problem of 9 Prisoners. In *Proceedings of the 25th Summer Meeting of Canadiant Mathematical Congress*, 1971. 1699

[375] B. Roy. Regard historique sur la place de la recherche opérationnelle et de l'aide à la décision en France. *Mathématiques et sciences humaines*, 175:25–40, 2006. In French, available at `http://msh.revues.org/3570`. 548

[376] A. J. Sadler and C. Gervet. Global reasoning on sets. In *CP'00 Workshop on Modelling and Problem Formulation (FORMUL'01)*, 2001. 708, 709

[377] H. R. D. Saidy and M. T. Taghavi-Fard. Study of Scheduling Problems with Machine Availability Constraint. *Journal of Industrial and Systems Engineering*, 1(4):360–383, 2008. 358

[378] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley, 1989. 1007, 1099

[379] P. Schaus, Y. Deville, and P. Dupont. Bound-Consistent Deviation. Slides presented at the Principles and Practice of Constraint Programming (CP'2007) Conference, 2007. 193

[380] P. Schaus, Y. Deville, and P. Dupont. Bound-Consistent Deviation Constraint. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 620–634. Springer-Verlag, 2007. 110, 194

[381] P. Schaus, Y. Deville, P. Dupont, and J.-C. Régin. Simplification and extension of *spread*. In *3th Workshop on Constraint Propagation and Implementation*, 2006. 110, 194

[382] P. Schaus, Y. Deville, P. Dupont, and J.-C. Régin. The *deviation* Constraint. In P. Van Hentenryck and L. Wolsey, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'07)*, volume 4150 of *LNCS*, pages 260–274, Brussels, Belgium, May 2007. Springer-Verlag. 110, 194

[383] P. Schaus, P. Van Hentenryck, and A. Zanarini. Revisiting the Soft Global Cardinality Constraint. In A. Lodi, M. Milano, and P. Toth, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*, volume 6140 of *LNCS*, pages 307–312, Bologna, Italy, 2010. Springer-Verlag. 110

[384] G. Scheithauer and J. Terno. The G4-Heuristic for the Pallet Loading Problem. *European Journal of Operational Research*, 46:511–522, 1996. 323

[385] C. Schulte, M. Lagerkvist, and G. Tack. Gecode, 2006. Available at `http://www.gecode.org`. 109, 113, 1127, 1172

[386] A. Schutt and A. Wolf. A New $O(n^2 \log n)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP'2010)*, volume 6308 of *LNCS*, pages 445–459, St Andrews, Scotland, 2010. Springer-Verlag. 986

[387] R. Sedgewick and O. Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley, 1996. 1668

[388] M. Sellmann. An arc consistency algorithm for the *minimum weight all different* constraint. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 744–749. Springer-Verlag, 2002. 1821

[389] M. Sellmann and S. Kadioglu. Dichotomic Search Protocols for Constrained Optimization. In P. J. Stuckey, editor, *Principles and Practice of Constraint Programming (CP'2008)*, volume 5202 of *LNCS*, pages 251–265. Springer-Verlag, 2008. 279

[390] P. Shaw. A Constraint for Bin Packing. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 648–662. Springer-Verlag, 2004. 779

[391] J. B. Shearer. Golomb rulers. Available at `http://www.research.ibm.com/people/s/shearer/grule.html`. 1344

[392] J. A. Shufet and H. J. Berliner. Generating hamiltonian circuits without backtracking from errors. *Theoretical Computer Science*, 132:347–375, 1994. 857

[393] H. Simonis. Channel Routing Seen as a Constraint Problem. Technical Report TR-LP-51, ECRC, 1990. 942

[394] H. Simonis. Building Industrial Application with Constraint Programming. In H. Common, C. Marché, and R. Treinen, editors, *Constraints in Computational Logics - Theory and Applications, International Summer School, CCL'99 Gif sur Yvette September 5-8, 1999, France, Revised Lectures*, pages 271–309. Springer-Verlag, 2001. 1523

[395] H. Simonis. Sudoku as a Constraint Problem. In B. Hnich, P. Prosser, and B. Smith, editors, *Modelling and Reformulating Constraint Satisfaction Problems, Fourth International Workshop, Sitges (Barcelona), Spain*, pages 13–27, 2005. 1523

[396] H. Simonis. Personal communication, June 2013. 1306

[397] H. Simonis, A. Aggoun, N. Beldiceanu, and É. Bourreau. Complex Constraint Abstraction: Global Constraint Visualization. In P. Deransart, M. V. Hermenegildo, and J. Małuszyński, editors, *Analysis and Vizualisation Tools for Constraint Programming*, volume 1870 of *LNCS*, pages 299–317. Springer-Verlag, 2000. 5

[398] H. Simonis, É. Bourreau, and N. Beldiceanu. A Note on Perfect Square Placement. Technical report, Cosytec, 1999. 371

[399] H. Simonis and T. Cornelissens. Modelling Producer/Consumer Constraints. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming (CP'95)*, volume 976 of *LNCS*, pages 449–462. Springer-Verlag, 1995. 333

[400] H. Simonis, P. Davern, J. Feldman, D. Mehta, L. Quesada, and M. Carlsson. A Generic Visualization Platform for CP. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP'2010)*, volume 6308 of *LNCS*, pages 460–474, St Andrews, Scotland, 2010. Springer-Verlag. 5

[401] H. Simonis and B. O'Sullivan. Search Strategies for Rectangle Packing. In P. J. Stuckey, editor, *Principles and Practice of Constraint Programming (CP'2008)*, volume 5202 of *LNCS*, pages 52–66. Springer-Verlag, 2008. 283, 365

[402] S. Skiena. *Implementing Discrete Mathematics. Combinatoric and Graph Theory with Mathematica*. Addison-Wesley, 1990. 61

[403] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. Published electronically at `http://oeis.org`, 2010. 549, 1028, 2408

[404] B. M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb Ruler Problem. In *IJCAI-99 Workshop on Non Binary Constraints*, 1999. 1345

[405] G. Smolka. Constraints in Oz. *ACM Computing Surveys*, 28(4), 1996. 109

[406] S. Sorlin and C. Solnon. A Parametric Filtering Algorithm for the Graph Isomorphism Problem. *Constraints*, 13(4), December 2008. 1353

[407] R. P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, New York, 2001. ii

[408] R. P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, New York, 2nd edition, 2011. ii

[409] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992. 2284, 2285

[410] P. J. Stuckey, M. J. García de la Banda, M. J. Maher, K. Marriott, J. K. Slaney, Z. Somogyi, M. G. Wallace, and T. Walsh. The G12 Project: Mapping Solver Independent Models to Efficient Solutions. In P. van Beek, editor, *Principles and Practice of Constraint Programming (CP'2005)*, volume 3709 of *LNCS*, pages 13–16. Springer-Verlag, 2005. 476

[411] S. Subbarayan. Integrating CSP Decomposition Techniques and BDDs for Compiling Configuration Problems. In R. Barták and M. Milano, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'05)*, volume 3524 of *LNCS*, pages 351–365, Prague, Czech Republic, May/June 2005. Springer-Verlag. 209

[412] T. Szczygiel. CLP Approaches to 2D Angle Placements. In *Proceedings of the 6th Annual Workshop of the ERCIM Working Group on Constraints*, Prague, Czech Republic, June 2001. Available at `http://arxiv.org/html/cs/0110012`. 1097

[413] R. Szymanek. *Constraint-Driven Design Space Exploration for Memory-Dominated Embedded Systems*. PhD thesis, Lund University, Sweden, June 2004. 1097

[414] R. Szymanek and K. Kuchcinski. A Constructive Algorithm for Memory-Aware Task Assignment and Scheduling. In *Proceedings of the Ninth International Symposium on Hardware/Software Codesign*, Copenhagen, 2001. 1097

[415] T. Tantau. *The* TikZ *&* PGF *Packages: Manual for version 2.10-cvs*, 2.10 edition, 2012. 114

[416] PLATON team. *Eclair*. Thales R & T, Orsay, France, v8.0 edition, 2003. Technical report 61 364. 109, 823

[417] S. Thiel. *Efficient Algorithms for Constraint Propagation and for Processing Tree Descriptions*. PhD thesis, Saarlandes University, Germany, 2004. 8, 1098, 2492, 2493

[418] C. Tong. Toward and engineering science of knowledge-based design. *Artificial Intelligence in Engineering*, 2(3):133–166, July 1987. 257

[419] M. A. Trick. A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. *Annals of Operations Research*, 118(1–4):73–84, 2003. 250, 779

[420] M. A. Trick. Integer and Constraint Programming Approaches for Round Robin Tournament Scheduling. In E. K. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2002, Gent, Belgium, August 21-23, 2002, Selected Revised Papers*, volume 2740 of *LNCS*, pages 63–77. Springer-Verlag, 2003. 2363

[421] P. Turán. On an Extremal Problem in Graph Theory. *Mat. Fiz. Lapok*, 48:436–452, 1941. In Hungarian. 440

[422] N. Ueda and T. Nagao. NP-completeness Results for NONOGRAM via Parsimonious Reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, May 1996. 294

[423] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, January 1976. 2325

[424] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979. 12, 549

[425] M. R. C. van Dongen, C. Lecoutre, and O. Roussel. Results of the Second CSP Solver Competition. In M. R. C. van Dongen, C. Lecoutre, and O. Roussel, editors, *Proceedings of the Second International CSP Solver Competition*, pages 1–10, 2008. 476

[426] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989. 1305

[427] P. Van Hentenryck. Scheduling and Packing in the Constraint Language cc(FD). In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994. 371

[428] P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999. 476

[429] P. Van Hentenryck and J.-P. Carillon. Generality vs. Specificity: an Experience with AI and OR Techniques. In *National Conference on Artificial Intelligence (AAAI-88)*, 1988. 88, 1194

[430] P. Van Hentenryck and Y. Deville. The *cardinality* Operator: a New Logical Connective in Constraint Logic Programming. In *Int. Conf. on Logic Programming (ICLP'91)*. MIT Press, 1991. 381

[431] P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, Implementation and Evaluation of the Constraint Language cc(FD). In A. Podelski, editor, *Constraints: Basics and Trends*, volume 910 of *LNCS*, pages 293–316. Springer-Verlag, 1995. 213

[432] W.-J. van Hoeve. The *alldifferent* Constraint: A Survey. In *Proceedings of the 6th Annual Workshop of the ERCIM Working Group on Constraints*, Prague, Czech Republic, June 2001. Available at `http://arxiv.org/html/cs/0110012`. 549

[433] W.-J. van Hoeve. A Hyper-Arc Consistency Algorithm for the *soft alldifferent* Constraint. In M. G. Wallace, editor, *Principles and Practice of Constraint Programming (CP'2004)*, volume 3258 of *LNCS*, pages 679–689. Springer-Verlag, 2004. 5, 2215

[434] W.-J. van Hoeve. *Operations Research Techniques in Constraint Programming*. PhD thesis, Amsterdam University, CWI, Netherlands, 2005. 1122, 2101, 2244, 2245

[435] W.-J. van Hoeve, G. Pesant, and L.-M. Rousseau. On Global Warming (Softening Global Constraints). In *Workshop on Soft Constraints*, Toronto, Canada, September 2004. 5, 110, 1307

[436] W.-J. van Hoeve, G. Pesant, and L.-M. Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, 12(4–5):347–373, 2006. 110

[437] W.-J. van Hoeve, G. Pesant, L.-M. Rousseau, and A. Sabharwal. Revisiting the *sequence* Constraint. In F. Benhamou, editor, *Principles and Practice of Constraint Programming (CP'2006)*, volume 4204 of *LNCS*, pages 620–634, Nantes, France, 2006. Springer-Verlag. 651

[438] W.-J. van Hoeve and J.-C. Régin. Open Constraints in a Closed World. In J. C. Beck and B. Smith, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'06)*, volume 3990 of *LNCS*, pages 244–257, Cork, Ireland, May/June 2006. Springer-Verlag. 319, 1952, 1964, 1965, 1968, 1969

[439] W.-J. van Hoeve and A. Sabharwal. Two Set-Constraints for Modeling and Efficiency. In *Sixth International Workshop on Constraint Modelling and Reformulation (ModRef'07), Providence, USA*, 2007. 708, 709, 2338

[440] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 1992. 370

[441] A. Vellino. Costas Arrays. Technical Report 90079, Computing Research Laboratory, Bell-Northern Research, 1990. 228

[442] N. R. Vempaty. Solving Constraint Satisfaction Problems using Finite State Automata. In *National Conference on Artificial Intelligence (AAAI-92)*, pages 453–458. AAAI Press, 1992. ii

[443] P. Vilím. $O(n \log n)$ Filtering Algorithms for Unary Resource Constraint. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'04)*, volume 3011 of *LNCS*, pages 335–347. Springer-Verlag, 2004. 328, 1144

[444] P. Vilím. *Global Constraints in Scheduling*. PhD thesis, Charles University, Prague, Department of Theoretical Computer Science and Mathematical Logic, Czech Republic, 2007. 328

[445] P. Vilím. Edge Finding Filtering Algorithm for Discrete Cumulative Resources in $O(kn \log n)$. In I. P. Gent, editor, *Principles and Practice of Constraint Programming (CP'2009)*, volume 5732 of *LNCS*, pages 802–816. Springer-Verlag, 2009. 328, 986

[446] P. Vilím. Max energy filtering algorithm for discrete cumulative resources. In W.-J. van Hoeve and J. N. Hooker, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *LNCS*, pages 294–308, Pittsburgh, PA, USA, 2009. Springer-Verlag. 328, 329, 986

[447] H. Voß. *PSTricks - Grafik für TeX und LaTeX*. Lehmanns Media, Berlin, 2007. Vierte Auflage. 114

[448] R. J. Wallace and N. Wilson. Conditional Lexicographic Orders in Constraint Satisfaction Problems. In J. C. Beck and B. Smith, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'06)*, volume 3990 of *LNCS*, pages 258–272, Cork, Ireland, May 2006. Springer-Verlag. 920, 921, 924, 928, 932, 936

[449] T. Walsh. Symmetry Breaking using Value Precedence. In *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, pages 168–172. IOS Press, 2006. 1481

[450] T. Walsh. Breaking Value Symmetry. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 880–888. Springer-Verlag, 2007. 1481

[451] J. J. Watkins. *Across the Board: The Mathematics of Chessboard Problems*. Princeton University Press, Princeton and Oxford, 2004. 547

[452] H. P. Williams and H. Yan. Representations of the *alldifferent* Predicate of Constraint Satisfaction in Integer Programming. *INFORMS Journal on Computing*, 13(2):96–103, 2001. 549

[453] A. Wolf. Better Propagation for Non-preemptive Single-Resource Constraint Problems. In B. Faltings, A. Petcu, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints, Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004; Lausanne, Switzerland, June 23-25, 2004, Revised Selected and Invited Papers*, volume 3419 of *LNCS*, pages 201–215. Springer-Verlag, 2005. 1144

[454] J. Würtz and T. Müller. Constructive Disjunction Revisited. In G. Görz and S. Hölldobler, editors, *20th German Annual Conference on Artificial Intelligence*, volume 1137 of *LNAI*, pages 377–386, Dresden, Germany, 1996. Springer-Verlag. 213

[455] Tallys H. Yunes. On the *sum* Constraint: Relaxation and Applications. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 80–92. Springer-Verlag, 2002. 2328, 2329

[456] S. Zampelli, Y. Deville, C. Solnon, S. Sorlin, and P. Dupont. Filtering for Subgraph Isomorphism. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 728–742. Springer-Verlag, 2007. 2325

[457] A. Zanarini, M. Milano, and G. Pesant. Improved Algorithm for the Soft Global Cardinality Constraint. In J. C. Beck and B. Smith, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'06)*, volume 3990 of *LNCS*, pages 288–299, Cork, Ireland, May 2006. Springer-Verlag. 110

[458] A. Zanarini and G. Pesant. Solution Counting Algorithms for Constraint-Centered Search Heuristics. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP'2007)*, volume 4741 of *LNCS*, pages 743–757. Springer-Verlag, 2007. 549

[459] A. Zanarini and G. Pesant. More Robust Counting-Based Heuristics with Alldifferent Constraints. In A. Lodi, M. Milano, and P. Toth, editors, *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*, volume 6140 of *LNCS*, pages 354–368, Bologna, Italy, 2010. Springer-Verlag. 12, 549

[460] J. Zhou. A Constraint Pprogram for Sloving the Job-Shop Pproblem. In E. C. Freuder, editor, *Principles and Practice of Constraint Programming (CP'1996)*, volume 1118 of *LNCS*, pages 510–524. Springer-Verlag, 1996. 2276

[461] J. Zhou. A Permutation-Based Approach for Solving the Job-Shop Problem. *Constraints*, 2(2):185–213, 1997. 2269, 2277

[462] N.-F. Zhou. Channel Routing with Constraint Logic Programming and Delay. In *9th Int. Conf. on Industrial Applications of AI*, pages 217–231. Gordon and Breach Science Publishers, 1996. 942

[463] W. Zimmermann and S. Cunningham. Editors' introduction: What is mathematical visualization? In *Visualization in teaching and learning mathematics*, pages 1–8. Mathematical Association of America, 1991. 5

Index

1...

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z