5.334 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 5.663 illustrates this correspondence.

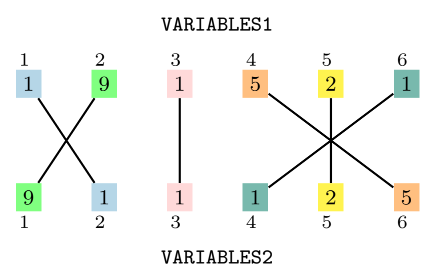


Figure 5.663: Illustration of the correspondence between the items of the VARIABLES1 and of the VARIABLES2 collections of the **Example** slot

All solutions Figure 5.664 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 5.664: 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 [48]. 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 [178]. 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 5.665. 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 5.665. Note that the `same_and_global_cardinality` constraint allows to model the matrix reconstruction problem without the additional `alldifferent` constraint.

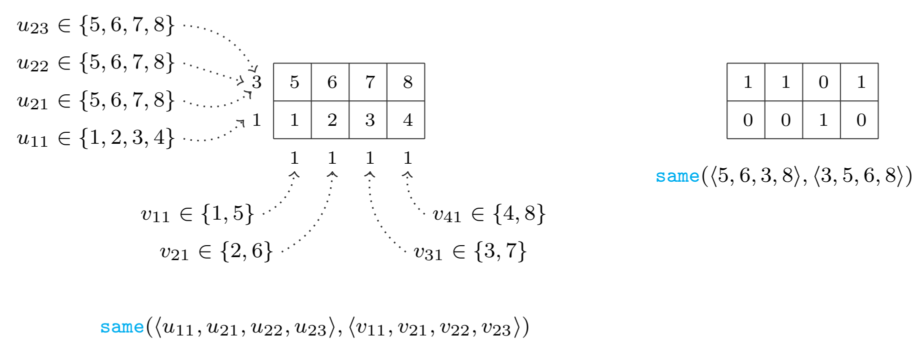


Figure 5.665: Modelling the 0-1 matrix reconstruction problem with the `same` constraint (variable u_{11} corresponds to the position of value 1 in the first row, variables u_{21}, u_{22}, u_{23} correspond to the position of value 1 in the second row, and variables v_{11}, v_{21}, v_{31}, v_{41} respectively 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 [297]. 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 [240], 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 instance, the `same` constraint

$$\texttt{same}\left(\begin{array}{l}\langle\ \texttt{var}-x_1, \texttt{var}-x_2\ \rangle,\\ \langle\ \texttt{var}-y_1, \texttt{var}-y_2\ \rangle,\end{array}\right)$$

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:

$$\text{global\_cardinality}\left(\begin{array}{l}\langle\ \texttt{var}-x_1, \texttt{var}-x_2\ \rangle,\\ \left\langle\begin{array}{ll}\texttt{val}-1 & \texttt{noccurrence}-c_1,\\ \texttt{val}-2 & \texttt{noccurrence}-c_2,\\ \texttt{val}-3 & \texttt{noccurrence}-c_3,\\ \texttt{val}-4 & \texttt{noccurrence}-c_4\end{array}\right\rangle\end{array}\right)$$

$$\text{global\_cardinality}\left(\begin{array}{l}\langle\ \texttt{var}-y_1, \texttt{var}-y_2\ \rangle,\\ \left\langle\begin{array}{ll}\texttt{val}-1 & \texttt{noccurrence}-c_1,\\ \texttt{val}-2 & \texttt{noccurrence}-c_2,\\ \texttt{val}-3 & \texttt{noccurrence}-c_3,\\ \texttt{val}-4 & \texttt{noccurrence}-c_4\end{array}\right\rangle\end{array}\right)$$

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 for instance the case 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 [423, 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 [47, 48, 49, 231], 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 [129, 130] 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`|

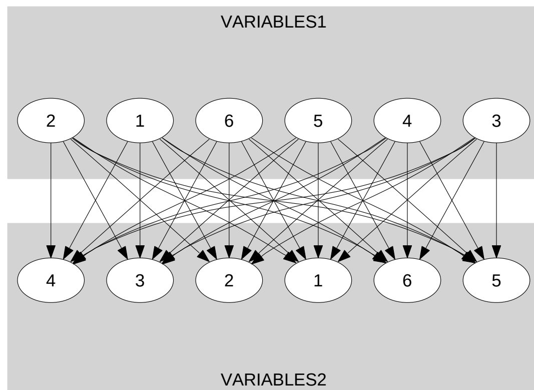
Graph model Parts (A) and (B) of Figure 5.666 respectively show the initial and final graph associated with the **Example** slot. Since we use the **NSOURCE** and **NSINK** graph properties, the source and sink vertices of the final graph are stressed with a double circle. Since there is a constraint 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`|.

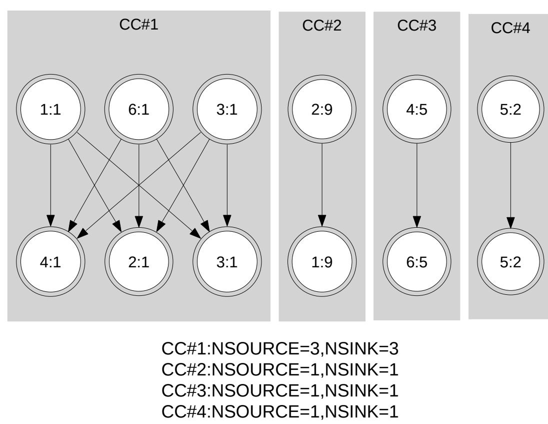
Signature Since the initial graph contains only sources and sinks, and since isolated vertices are eliminated from the final graph, we make the following observations:

- 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 $\underline{\overline{\mathbf{NSOURCE}}}$ 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 5.666: 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 $S_{i+|\mathtt{VARIABLES1}|}$ that is equal to 1.

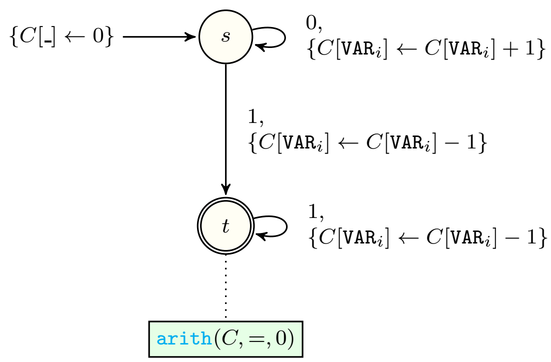


Figure 5.667: Automaton of the same constraint