5.229 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

$(\langle 5, 2, 7, 1\rangle, \langle 5, 2, 6, 2\rangle)$

The lex_greater constraint holds since VECTOR1 $= \langle 5, 2, 7, 1\rangle$ is lexicographically strictly greater than VECTOR2 $= \langle 5, 2, 6, 2\rangle$.

Typical

|VECTOR1| > 1
∨ (|VECTOR1| < 5, nval([VECTOR1.var, VECTOR2.var]) < 2 ∗ |VECTOR1|)
∨ (maxval([VECTOR1.var, VECTOR2.var]) ≤ 1, 2 ∗ |VECTOR1|−max_nvalue([VECTOR1.var, VECTOR2.var]) > 2)

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 [174].

Algorithm

The first filtering algorithm maintaining arc-consistency for this constraint was presented in [173]. A second filtering algorithm maintaining arc-consistency and detecting entailment in a more eager way, was given in [96]. This second algorithm was derived from a deterministic finite automata. A third filtering algorithm extending the algorithm presented in [173] detecting entailment is given in the PhD thesis of Z. Kızıltan [239, page 95]. The previous thesis [239, 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 [175] in [176].

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 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)

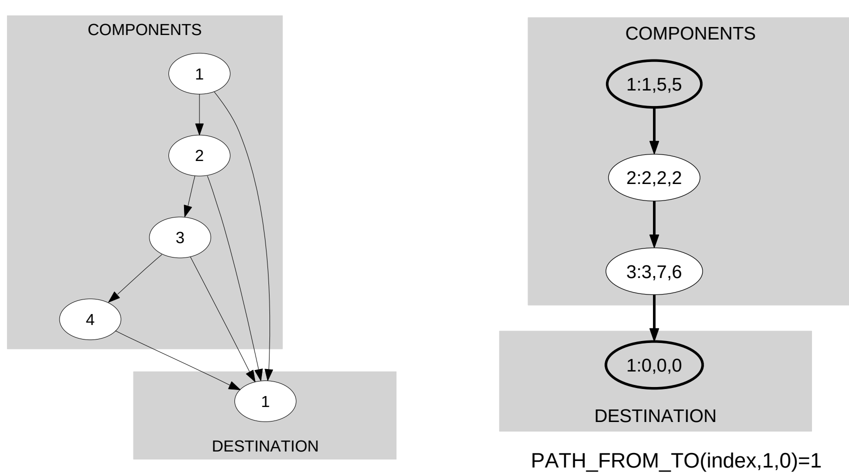
$$\bigvee\left(\begin{array}{l}\texttt{item2.index} > 0 \wedge \texttt{item1.x} = \texttt{item1.y},\\ \texttt{item2.index} = 0 \wedge \texttt{item1.x} > \texttt{item1.y}\end{array}\right)$$

Graph property(ies) **PATH_FROM_TO**$(\texttt{index}, 1, 0) = 1$

Graph model

Parts (A) and (B) of Figure 5.488 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 5.488: 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 $\texttt{item}_1.x > \texttt{item}_2.y$.
- 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_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 **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 5.489 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: $(\texttt{VAR1}_i < \texttt{VAR2}_i \Leftrightarrow S_i = 1) \wedge (\texttt{VAR1}_i = \texttt{VAR2}_i \Leftrightarrow S_i = 2) \wedge (\texttt{VAR1}_i > \texttt{VAR2}_i \Leftrightarrow S_i = 3)$.

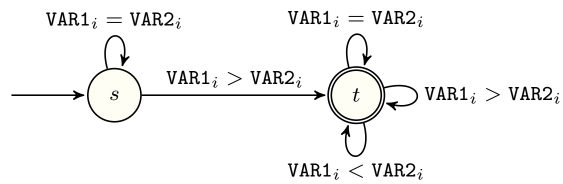


Figure 5.489: Automaton of the `lex_greater` constraint

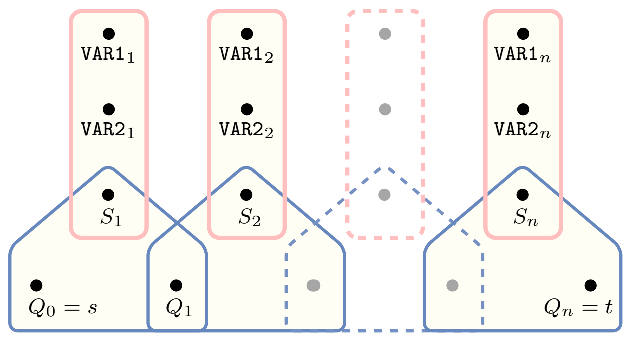


Figure 5.490: Hypergraph of the reformulation corresponding to the automaton of the `lex_greater` constraint