5.372 sort_permutation

DESCRIPTION LINKS GRAPH

Origin [449]

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

|PERMUTATION| = |FROM|
|PERMUTATION| = |TO|
PERMUTATION.var ≥ 1
PERMUTATION.var ≤ |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.

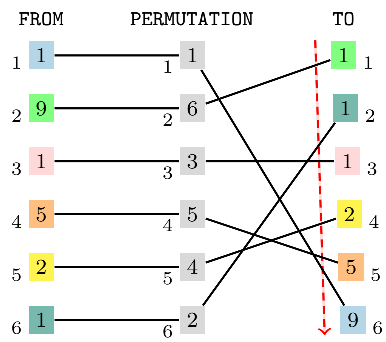


Figure 5.718: Illustration of the correspondence between the items of the FROM and the TO collections according to the permutation defined by the items of the PERMUTATION collection of the **Example** slot (note that the items of the TO collection are sorted in increasing order)

Typical

$|\texttt{FROM}| > 1$
$\texttt{range}(\texttt{FROM.var}) > 1$
$\texttt{lex\_different}(\texttt{FROM}, \texttt{TO})$

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.
- Functional dependency: PERMUTATION determined by FROM and TO.

Remark

This constraint is referenced under the name sorting in **SICStus Prolog**.

Algorithm

[449].

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 $\forall i \in [1, n] : \sum_{j=1}^{j=n} (\texttt{FROM}[j] > \texttt{TO}[i]) \geq n - i$ (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

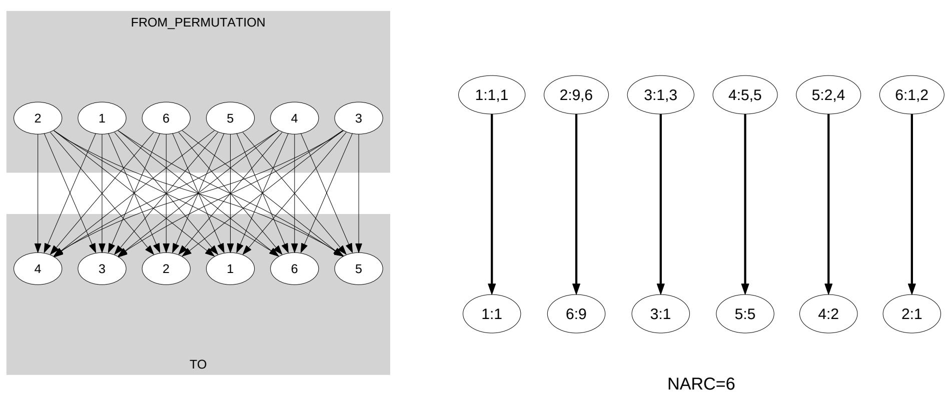
Parts (A) and (B) of Figure 5.719 respectively show the initial and final graph associated with the first graph constraint of the **Example** slot. In both graphs the source vertices correspond to the items of the derived collection FROM_PERMUTATION, while the sink vertices correspond to the items of the TO collection. Since the first graph constraint uses the **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 5.719: Initial and final graph of the `sort_permutation` constraint