5.200 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

$$\left(\left\langle \begin{array}{llll} \mathtt{index}-1 & \mathtt{succ}-2 & \mathtt{pred}-2, \\ \mathtt{index}-2 & \mathtt{succ}-1 & \mathtt{pred}-1, \\ \mathtt{index}-3 & \mathtt{succ}-5 & \mathtt{pred}-4, \\ \mathtt{index}-4 & \mathtt{succ}-3 & \mathtt{pred}-5, \\ \mathtt{index}-5 & \mathtt{succ}-4 & \mathtt{pred}-3 \end{array} \right\rangle\right)$$

The inverse constraint holds since:

- NODES[1].succ $= 2 \Leftrightarrow$ NODES[2].pred $= 1$,
- NODES[2].succ $= 1 \Leftrightarrow$ NODES[1].pred $= 2$,
- NODES[3].succ $= 5 \Leftrightarrow$ NODES[5].pred $= 3$,
- NODES[4].succ $= 3 \Leftrightarrow$ NODES[3].pred $= 4$,
- NODES[5].succ $= 4 \Leftrightarrow$ NODES[4].pred $= 5$.

Typical |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 for instance the case 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 heuristics 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 [129, 130]. 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 [148] 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.

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	$CLIQUE \mapsto$ 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 5.465 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.

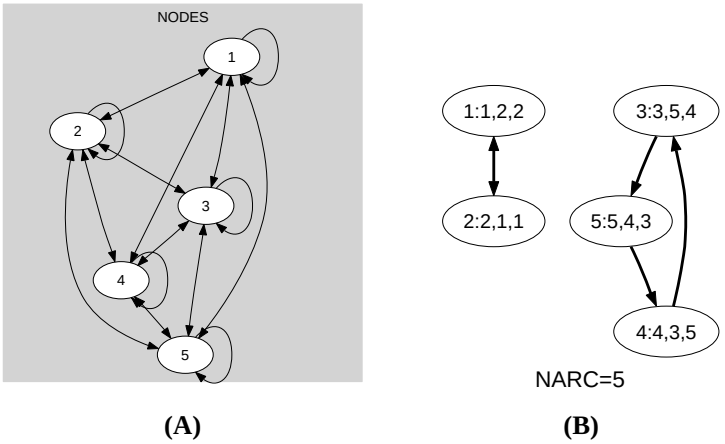

Figure 5.465: Initial and final graph of the `inverse` constraint

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 $\mathbf{NARC} = |\texttt{NODES}|$ to $\mathbf{NARC} \geq |\texttt{NODES}|$ and simplify $\overline{\underline{\mathbf{NARC}}}$ to $\overline{\mathbf{NARC}}$.

Automaton

Figure 5.466 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.

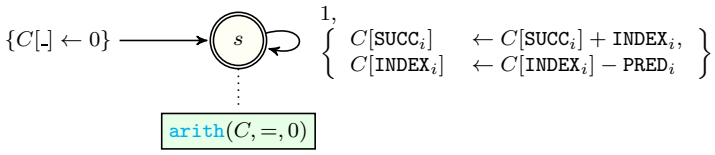

Figure 5.466: Automaton of the `inverse` constraint