

5.43 binary_tree

	DESCRIPTION	LINKS	GRAPH
Origin	Derived from <code>tree</code> .		
Constraint	<code>binary_tree(NTREES, NODES)</code>		
Arguments	NTREES : <code>dvar</code> NODES : <code>collection(index-int, succ-dvar)</code>		
Restrictions	$NTREES \geq 0$ $NTREES \leq NODES $ <code>required(NODES, [index, succ])</code> $NODES.index \geq 1$ $NODES.index \leq NODES $ <code>distinct(NODES, index)</code> $NODES.succ \geq 1$ $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 root.

Example $2, \left(\begin{array}{l} \text{index} - 1 \quad \text{succ} - 1, \\ \text{index} - 2 \quad \text{succ} - 3, \\ \text{index} - 3 \quad \text{succ} - 5, \\ \text{index} - 4 \quad \text{succ} - 7, \\ \text{index} - 5 \quad \text{succ} - 1, \\ \text{index} - 6 \quad \text{succ} - 1, \\ \text{index} - 7 \quad \text{succ} - 7, \\ \text{index} - 8 \quad \text{succ} - 5 \end{array} \right)$

The `binary_tree` constraint holds since its second argument corresponds to the 2 (i.e., the first argument of the `binary_tree` constraint) binary trees depicted by Figure 5.80.

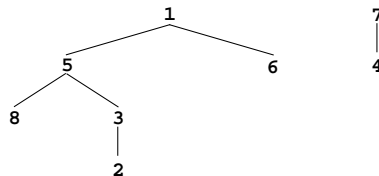


Figure 5.80: The two binary trees corresponding to the **Example** slot

Typical

```

NTREES > 0
NTREES < |NODES|
|NODES| > 2

```

Symmetry

Items of NODES are [permutable](#).

Reformulation

The `binary_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 and of (3) a set of $|\text{NODES}|^2$ reified constraints and of $|\text{NODES}|$ inequalities constraints for enforcing the fact that each vertex has at most two children.

1. For each vertex $\text{NODES}[i]$ ($i \in [1, |\text{NODES}|]$) 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 $\text{NODES}[i]$ 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 $\text{NODES}[i], \text{NODES}[j]$ ($i, j \in [1, |\text{NODES}|]$) of the NODES collection we create a reified constraint of the form $\text{NODES}[i].\text{succ} = \text{NODES}[j].\text{index} \wedge i \neq j \Rightarrow R_i < R_j$. The purpose of this constraint is to express the fact that, if there is an arc from vertex $\text{NODES}[i]$ to another vertex $\text{NODES}[j]$, then R_i should be strictly less than R_j .
2. For each vertex $\text{NODES}[i]$ ($i \in [1, |\text{NODES}|]$) of the NODES collection we create a 0-1 variable B_i and state the following reified constraint $\text{NODES}[i].\text{succ} = \text{NODES}[i].\text{index} \Leftrightarrow B_i$ in order to force variable B_i to be set to value 1 if and only if there is a loop on vertex $\text{NODES}[i]$. Finally we create a constraint $\text{NTREES} = B_1 + B_2 + \dots + B_{|\text{NODES}|}$ 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 $\text{NODES}[i], \text{NODES}[j]$ ($i, j \in [1, |\text{NODES}|]$) of the NODES collection we create a 0-1 variable B_{ij} and state the following reified constraint $\text{NODES}[i].\text{succ} = \text{NODES}[j].\text{index} \wedge i \neq j \Leftrightarrow B_{ij}$. Variable B_{ij} is set to value 1 if and only if there is an arc from $\text{NODES}[i]$ to $\text{NODES}[j]$. Then for each vertex $\text{NODES}[j]$ ($j \in [1, |\text{NODES}|]$) we create a constraint of the form $B_{1j} + B_{2j} + \dots + B_{|\text{NODES}|j} \leq 2$.

See also

[generalisation: tree](#) (at most two childrens replaced by no restriction on maximum number of childrens).

[implied by: path](#).

[implies: tree](#).

[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](#).

Arc input(s)	NODES
Arc generator	<i>CLIQUE</i> \mapsto <code>collection(nodes1, nodes2)</code>
Arc arity	2
Arc constraint(s)	<code>nodes1.succ = nodes2.index</code>
Graph property(ies)	<ul style="list-style-type: none"> • <u>MAX_NSCC</u> \leq 1 • <u>NCC</u> = NTREES • <u>MAX_ID</u> \leq 2
Graph class	<u>ONE_SUCC</u>

Graph model

We use the same graph constraint as for the `tree` constraint, except that we add the graph property MAX_ID \leq 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 5.81 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 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 $\text{NTREES} = \text{NCC} = 2$ and since MAX_ID = 2.

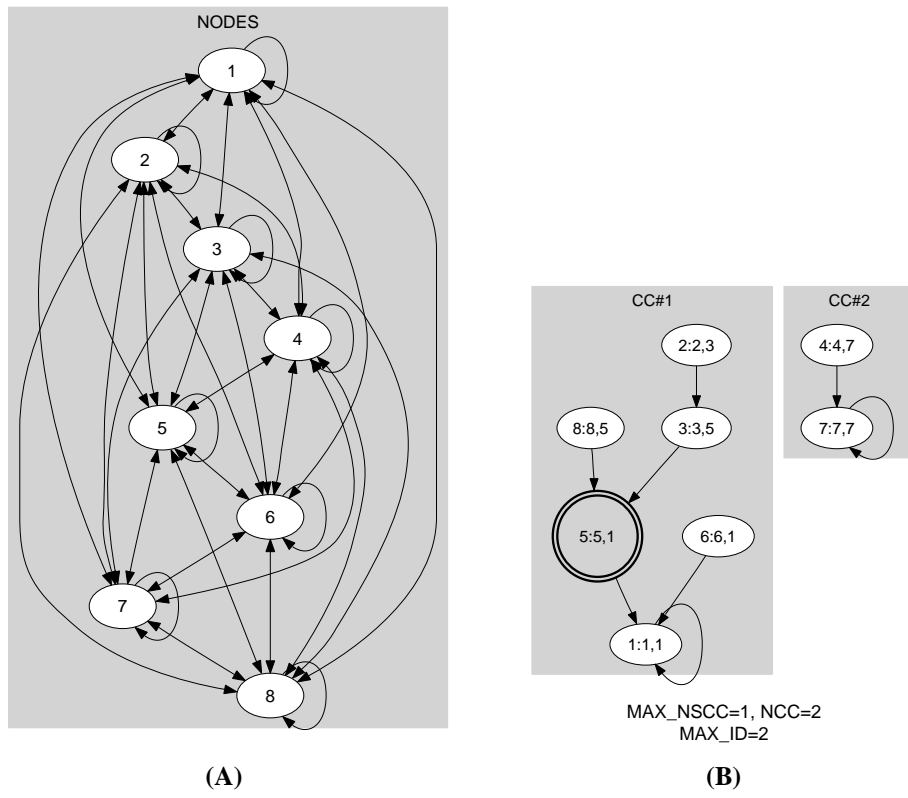


Figure 5.81: Initial and final graph of the binary_tree constraint