5.55 binary_tree

DESCRIPTION LINKS GRAPH

Origin Derived from `tree`.

Constraint `binary_tree(NTREES, NODES)`

Arguments

```
NTREES : dvar
NODES  : collection(index-int, succ-dvar)
```

Restrictions

```
NTREES ≥ 0
NTREES ≤ |NODES|
required(NODES, [index, succ])
NODES.index ≥ 1
NODES.index ≤ |NODES|
distinct(NODES, index)
NODES.succ ≥ 1
NODES.succ ≤ |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

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

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

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

The first `binary_tree` constraint holds since its second argument corresponds to the 2 (i.e., the first argument of the first `binary_tree` constraint) binary trees depicted by Figure 5.141.

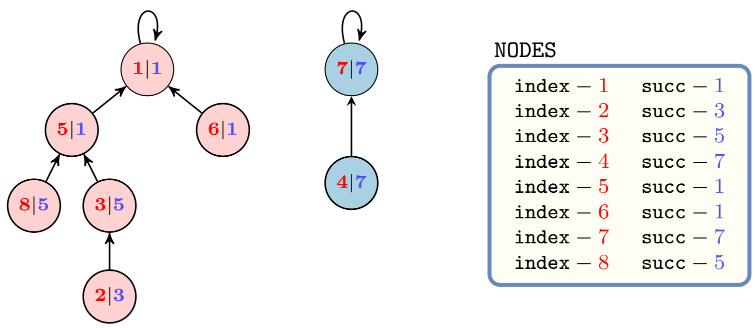


Figure 5.141: The two binary trees corresponding to the first example of the **Example** slot; each vertex contains the information `index`|`succ` where `succ` is the index of its father in the tree (by convention the father of the root is the root itself).

All solutions

Figure 5.142 gives all solutions to the following non ground instance of the `binary_tree` constraint: $\mathtt{NTREES} \in \{1,4\}$, $\mathtt{S}_1 \in [1,2]$, $\mathtt{S}_2 \in [1,3]$, $\mathtt{S}_3 \in [3,4]$, $\mathtt{S}_4 \in [3,4]$, $\mathtt{S}_5 \in [2,3]$, $\mathtt{binary\_tree}(\mathtt{NTREES}, \langle 1\ \mathtt{S}_1, 2\ \mathtt{S}_2, 3\ \mathtt{S}_3, 4\ \mathtt{S}_4, 5\ \mathtt{S}_5\rangle)$.

① $(\mathbf{4}, \langle \mathbf{1}_1, \mathbf{2}_2, \mathbf{3}_3, \mathbf{4}_4, \mathbf{2}_5\rangle)$
② $(\mathbf{4}, \langle \mathbf{1}_1, \mathbf{2}_2, \mathbf{3}_3, \mathbf{4}_4, \mathbf{3}_5\rangle)$
③ $(\mathbf{1}, \langle \mathbf{2}_1, \mathbf{3}_2, \mathbf{3}_3, \mathbf{3}_4, \mathbf{2}_5\rangle)$
④ $(\mathbf{1}, \langle \mathbf{2}_1, \mathbf{3}_2, \mathbf{4}_3, \mathbf{4}_4, \mathbf{2}_5\rangle)$
⑤ $(\mathbf{1}, \langle \mathbf{2}_1, \mathbf{3}_2, \mathbf{4}_3, \mathbf{4}_4, \mathbf{3}_5\rangle)$

Figure 5.142: All solutions corresponding to the non ground example of the `binary_tree` constraint of the **All solutions** slot; the `index` attribute is displayed as indices of the `succ` attribute and all vertices of a same tree are coloured by the same colour.

Typical

$\mathtt{NTREES} > 0$
$\mathtt{NTREES} < |\mathtt{NODES}|$
$|\mathtt{NODES}| > 2$

Symmetry

Items of `NODES` are permutable.

Arg. properties

Functional dependency: `NTREES` determined by `NODES`.

Reformulation

The `binary_tree` constraint can be expressed in term of (1) a set of $|\texttt{NODES}|^2$ reified constraints for avoiding circuit between more than one node and of (2) $|\texttt{NODES}|$ reified constraints and of one sum constraint for counting the trees and of (3) a set of $|\texttt{NODES}|^2$ reified constraints and of $|\texttt{NODES}|$ inequalities constraints for enforcing the fact that each vertex has at most two children.

1. For each vertex $\texttt{NODES}[i]$ $(i \in [1, |\texttt{NODES}|])$ of the `NODES` collection we create a variable R_i that takes its value within interval $[1, |\texttt{NODES}|]$. This variable represents the *rank* of vertex $\texttt{NODES}[i]$ 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 $\texttt{NODES}[i], \texttt{NODES}[j]$ $(i, j \in [1, |\texttt{NODES}|])$ of the `NODES` collection we create a reified constraint of the form $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Rightarrow R_i < R_j$. The purpose of this constraint is to express the fact that, if there is an arc from vertex $\texttt{NODES}[i]$ to another vertex $\texttt{NODES}[j]$, then R_i should be strictly less than R_j.

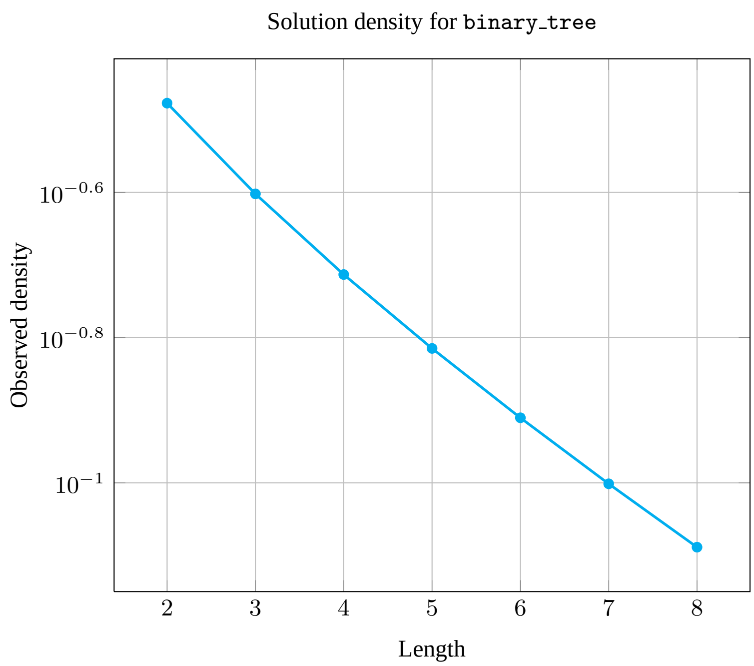
2. For each vertex $\texttt{NODES}[i]$ $(i \in [1, |\texttt{NODES}|])$ of the `NODES` collection we create a 0-1 variable B_i and state the following reified constraint $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[i].\texttt{index} \Leftrightarrow B_i$ in order to force variable B_i to be set to value 1 if and only if there is a loop on vertex $\texttt{NODES}[i]$. Finally we create a constraint $\texttt{NTREES} = B_1 + B_2 + \cdots + B_{|\texttt{NODES}|}$ for stating the fact that the number of trees is equal to the number of loops of the graph.

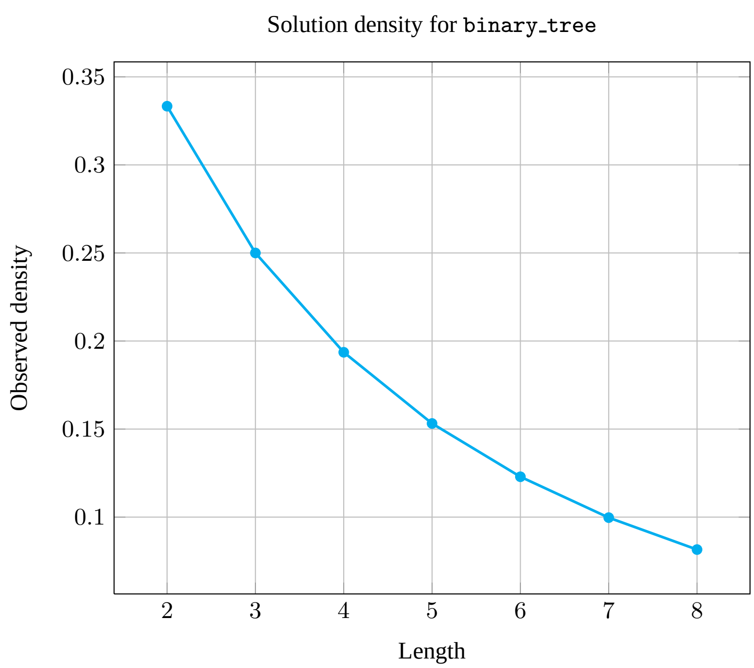
3. For each pair of vertices $\texttt{NODES}[i], \texttt{NODES}[j]$ $(i, j \in [1, |\texttt{NODES}|])$ of the `NODES` collection we create a 0-1 variable B_{ij} and state the following reified constraint $\texttt{NODES}[i].\texttt{succ} = \texttt{NODES}[j].\texttt{index} \wedge i \neq j \Leftrightarrow B_{ij}$. Variable B_{ij} is set to value 1 if and only if there is an arc from $\texttt{NODES}[i]$ to $\texttt{NODES}[j]$. Then for each vertex $\texttt{NODES}[j]$ $(j \in [1, |\texttt{NODES}|])$ we create a constraint of the form $B_{1j} + B_{2j} + \cdots + B_{|\texttt{NODES}|j} \leq 2$.

Counting

Length (n)	2	3	4	5	6	7	8
Solutions	3	16	121	1191	14461	209098	3510921

Number of solutions for `binary_tree`: domains $0..n$

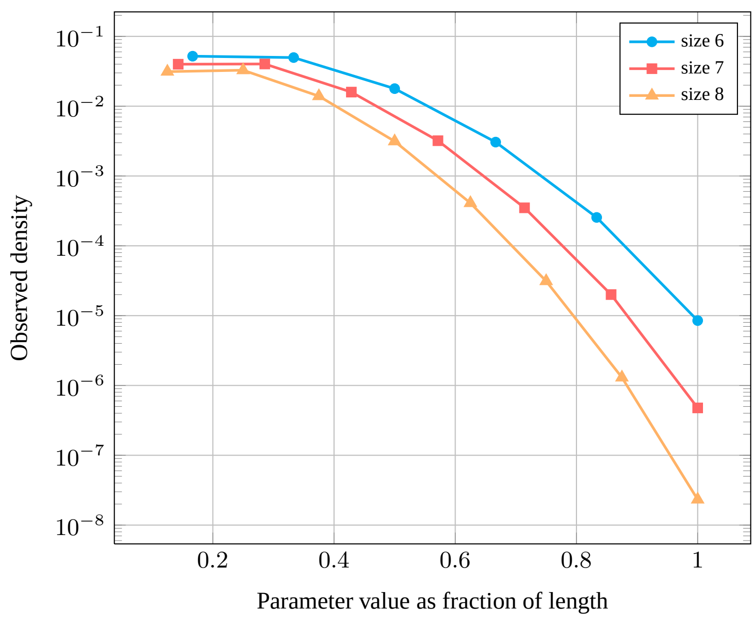

Solution density for binary_tree
$10^{-0.6}$
$10^{-0.8}$
10^{-1}
Observed density
2
3
4
5
6
7
8
Length

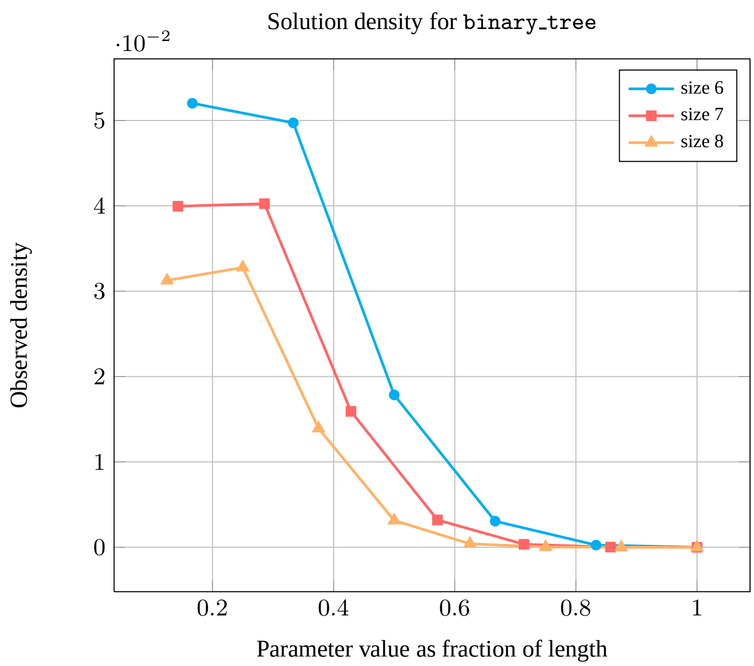

Solution density for binary_tree
0.35
0.3
0.25
0.2
0.15
0.1
Observed density
2
3
4
5
6
7
8
Length

Length (n)		2	3	4	5	6	7	8
Total		3	16	121	1191	14461	209098	3510921
Parameter value	1	2	9	60	540	6120	83790	1345680
	2	1	6	48	480	5850	84420	1411200
	3	-	1	12	150	2100	33390	599760
	4	-	-	1	20	360	6720	135240
	5	-	-	-	1	30	735	17640
	6	-	-	-	-	1	42	1344
	7	-	-	-	-	-	1	56
	8	-	-	-	-	-	-	1

Solution count for `binary_tree`: domains $0..n$

Solution density for `binary_tree`





See also

generalisation: `tree` *(at most two childrens replaced by no restriction on maximum number of childrens)*.

implied by: `path`.

implies: `tree`.

implies (items to collection): `atleast_nvector`.

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.

modelling: functional dependency.

Arc input(s)	NODES
Arc generator	$CLIQUE \mapsto$ collection(nodes1, nodes2)
Arc arity	2
Arc constraint(s)	nodes1.succ = nodes2.index
Graph property(ies)	• **MAX_NSCC** ≤ 1 • **NCC** = NTREES • **MAX_ID** ≤ 2
Graph class	ONE_SUCC

Graph model

We use the same graph constraint as for the tree constraint, except that we add the graph property **MAX_ID** ≤ 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 5.143 respectively show the initial and final graph associated with the first example of 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 NTREES = **NCC** = 2 and since **MAX_ID** = 2.

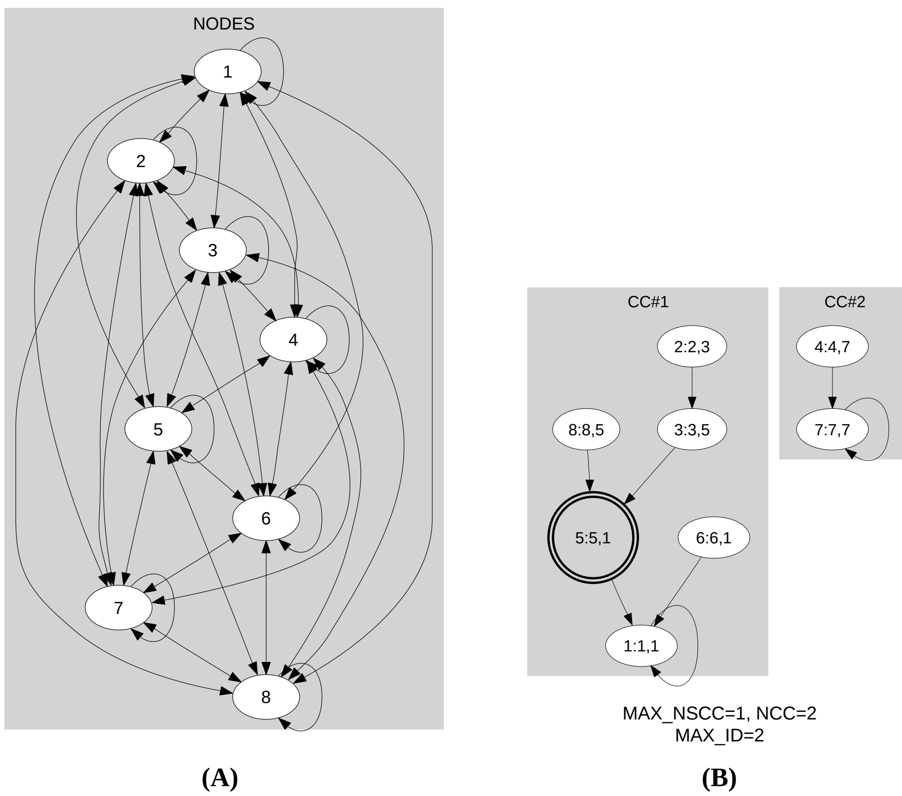


Figure 5.143: Initial and final graph of the `binary_tree` constraint