

## 5.78 counts

	DESCRIPTION	LINKS	GRAPH	AUTOMATON
<b>Origin</b>	Derived from <code>count</code> .			
<b>Constraint</b>	<code>counts(VALUEs, VARIABLEs, RELOP, LIMIT)</code>			
<b>Arguments</b>	VALUEs : <code>collection(val-int)</code> VARIABLEs : <code>collection(var-dvar)</code> RELOP : <code>atom</code> LIMIT : <code>dvar</code>			
<b>Restrictions</b>	<code>required(VALUEs, val)</code> <code>distinct(VALUEs, val)</code> <code>required(VARIABLEs, var)</code> $RELOP \in [=, \neq, <, \geq, >, \leq]$			
<b>Purpose</b>	Let $N$ be the number of variables of the VARIABLEs collection assigned to a value of the VALUEs collection. Enforce condition $N$ RELOP LIMIT to hold.			
<b>Example</b>	$\left( \begin{array}{l} \langle 1, 3, 4, 9 \rangle, \\ \text{var} - 4, \\ \text{var} - 5, \\ \langle \text{var} - 5, \\ \text{var} - 4 \rangle, =, 3 \\ \text{var} - 1, \\ \text{var} - 5 \end{array} \right)$			
	Values 1, 3, 4 and 9 of the VALUEs collection are assigned to 3 items of the VARIABLEs = $\langle 4, 5, 5, 4, 1, 5 \rangle$ collection. The counts constraint holds since this number is in fact equal (RELOP is set to =) to the last argument of the counts constraint.			
<b>Typical</b>	$ VALUEs  > 1$ $ VARIABLEs  > 1$ $range(VARIABLEs.var) > 1$ $ VARIABLEs  >  VALUEs $ $LIMIT > 0$ $LIMIT <  VARIABLEs $			
<b>Symmetries</b>	<ul style="list-style-type: none"> <li>Items of VALUEs are <a href="#">permutable</a>.</li> <li>Items of VARIABLEs are <a href="#">permutable</a>.</li> <li>An occurrence of a value of VARIABLEs.var that belongs to VALUEs.val (resp. does not belong to VALUEs.val) can be <a href="#">replaced</a> by any other value in VALUEs.val (resp. not in VALUEs.val).</li> </ul>			
<b>Usage</b>	Used in the <b>Constraint(s) on sets</b> slot for defining some constraints like <a href="#">assign_and_counts</a> .			

- Reformulation** The `count(VALUEs, VARIABLES, RELOP, LIMIT)` constraint can be expressed in term of the conjunction `among(N, VARIABLES, VALUEs)  $\wedge$  N RELOP LIMIT`.
- Used in** `assign_and_counts`.
- See also** **assignment dimension added:** `assign_and_counts` (*assignment dimension introduced*).  
**common keyword:** `among` (*value constraint, counting constraint*).  
**specialisation:** `count` (*variable  $\in$  VALUEs replaced by variable=VALUE*).
- Keywords** **characteristic of a constraint:** `automaton`, `automaton with counters`.  
**constraint network structure:** `alpha-acyclic constraint network(2)`.  
**constraint type:** `value constraint`, `counting constraint`.  
**filtering:** `arc-consistency`.  
**final graph structure:** `acyclic`, `bipartite`, `no loop`.

<b>Arc input(s)</b>	VARIABLES VALUES
<b>Arc generator</b>	<i>PRODUCT</i> $\mapsto$ <code>collection(variables, values)</code>
<b>Arc arity</b>	2
<b>Arc constraint(s)</b>	<code>variables.var = values.val</code>
<b>Graph property(ies)</b>	<b>NARC</b> RELOP LIMIT
<b>Graph class</b>	<ul style="list-style-type: none"> <li>• <b>ACYCLIC</b></li> <li>• <b>BIPARTITE</b></li> <li>• <b>NO_LOOP</b></li> </ul>

**Graph model**

Because of the arc constraint `variables.var = values.val` and since each domain variable can take at most one value, **NARC** is the number of variables taking a value in the **VALUES** collection.

Parts (A) and (B) of Figure 5.158 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.

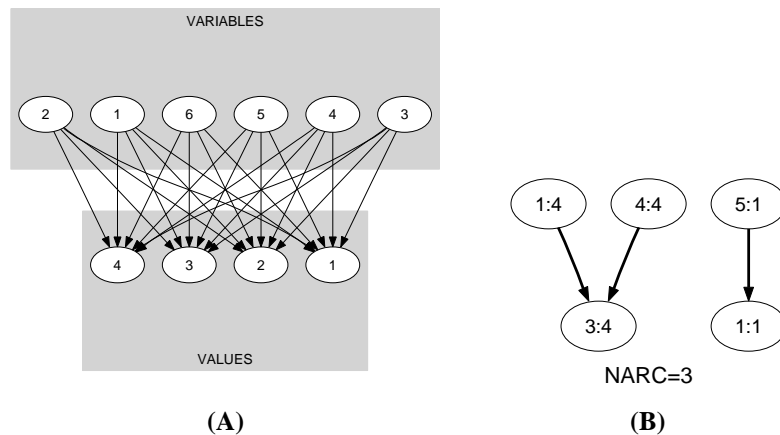


Figure 5.158: Initial and final graph of the counts constraint

**Automaton**

Figure 5.159 depicts the automaton associated with the counts constraint. To each variable  $VAR_i$  of the collection VARIABLES corresponds a 0-1 signature variable  $S_i$ . The following signature constraint links  $VAR_i$  and  $S_i$ :  $VAR_i \in VALUES \Leftrightarrow S_i$ .

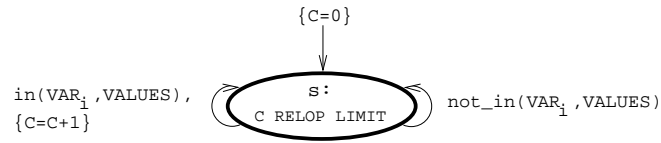


Figure 5.159: Automaton of the counts constraint

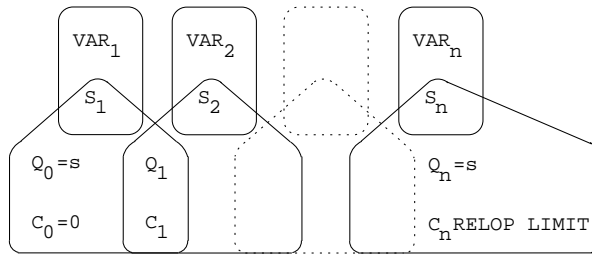


Figure 5.160: Hypergraph of the reformulation corresponding to the automaton of the counts constraint