

# Deriving Filtering Algorithms from Constraint Checkers

Nicolas Beldiceanu(1), Mats Carlsson(2) and Thierry Petit(1)

(1) LINA FRE CNRS 2729, Ecole des Mines de Nantes,  
FR-44307 Nantes Cedex 3, France.

email: {Nicolas.Beldiceanu,Thierry.Petit}@emn.fr

(2) SICS, P.O. Box 1263, SE-164 29 Kista, Sweden.

email: Mats.Carlsson@sics.se

# CONTEXT

CONSTRAINT: **Condition** on a set of variables  
**maximum(M,[X1,X2,...,Xn])**

CHECKER: **Decision procedure** which checks if a ground instance holds or not  
**maximum(3,[3,1,3,7]) -> no**  
**maximum(7,[3,1,3,7]) -> yes**

FILTERING ALGORITHMS:

- (1) **Check feasibility** of the constraint  
**M in 1..2, X1 in 3..4, X2 in 3..4, maximum(M,[X1,X2]): infeasible**
- (2) **Eliminate values** that lead to infeasibility  
**M in 5..9, X1 in 3..4, X2 in 1..6, maximum(M,[X1,X2]): M in 5..6, X2 in 5..6**
- (3) The holy grail:  
Achieving **arc-consistency** for a constraint with the **lowest** complexity.

CHALLENGE: How to **automatically** derive a filtering algorithm from a constraint checker ?

# OVERVIEW

1. INTRODUCTION
2. TYPE OF AUTOMATON USED
3. FROM AUTOMATA TO FILTERING ALGORITHMS
4. APPLICATIONS
5. HANDLING RELAXATION FOR A COUNTER-FREE AUTOMATON
6. CONCLUSION AND PERSPECTIVES

⇒ **1. INTRODUCTION**

2. TYPE OF AUTOMATON USED

3. FROM AUTOMATA TO FILTERING ALGORITHMS

4. APPLICATIONS

5. HANDLING RELAXATION FOR A COUNTER-FREE AUTOMATON

6. CONCLUSION AND PERSPECTIVES

# INTRODUCTION

Providing **efficient filtering algorithms** is challenging since:

- There are a lot of global constraints,
- Filtering algorithms are far from obvious (4-5 each year),
- Easy to introduce errors or to forget cases.

Want to **systematically** derive **correct** filtering algorithms from **first principle** avoiding creativity.

As a first principle we select a **constraint checker for the ground case.**

# CONTRIBUTIONS

- A **model** of automaton (with counters) for writing **compact** constraint checkers,
- A **reformulation** of an automaton as a conjunction of signature and transition constraints,
- A partial **characterization** of conditions for obtaining **arc-consistency**.

# LIMITATIONS

- **Expressivity** limitation:
  - Restrict ourselves to constraints that can be checked by **scanning once** through their variables,
  - The size of the automaton has to be **bounded** by a polynomial of the number of variables.
  
- **Operational** limitation:
  - **For some constraints** for which there exists a specialized algorithm achieving arc-consistency **we don't achieve arc-consistency**.

## RELATED WORK

- **Constraint networks:**
  - **N.R.Vempaty** [**AAAI-92**],  
Solving constraint satisfaction problems using finite automata.
  - **J.Amilhastre** [**PhD-99**],  
Représentation par automate d'ensemble de solutions de problèmes de satisfaction de contraintes.
- **Arithmetic constraints:**
  - **B.Boigelot, P.Wolper** [**ICLP-02**],  
Representing arithmetic constraints with finite automata: An overview.
- **Global constraints:**
  - **G.Pesant** [**Workshop CP-03**], [**CP-04**],  
A regular language membership constraint for sequence of variables.
  - **M.Carlsson, N.Beldiceanu** [**ESOP-04**],  
From constraints to finite automata to filtering algorithms.



1. INTRODUCTION

⇒ 2. **TYPE OF AUTOMATON USED**

3. FROM AUTOMATA TO FILTERING ALGORITHMS

4. APPLICATIONS

5. HANDLING RELAXATION FOR A COUNTER-FREE AUTOMATON

6. CONCLUSION AND PERSPECTIVES

# EXAMPLE OF CONSTRAINT CHECKER

Check is achieved by scanning **once** through the variables **without** using any data structure.

```
inflexion(ninf, vars):
```

```
  ninf: domain variable
```

```
  vars: a sequence of domain variables
```

```
ninf is the number of inflexions of the sequence of variables vars
```

An inflexion is one of the following pattern:

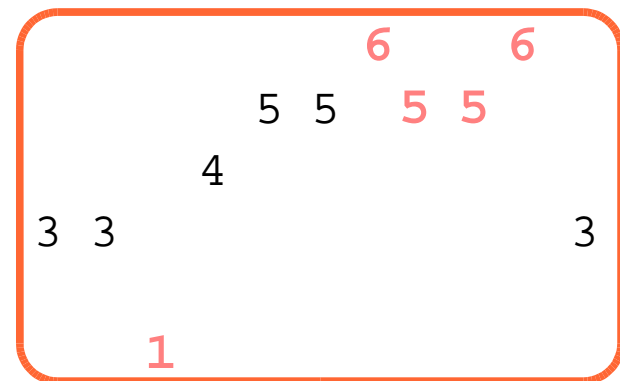
- a **strict increase followed by a strict decrease**,
- a **strict decrease followed by a strict increase**.

```
inflexion(4, [3, 3, 1, 4, 5, 5, 6, 5, 5, 6, 3])
```

holds since the sequence 3 3 1 4 5 5 6 5 5 6 3 contains the four inflexions

3 1 4, 5 6 5, 6 5 5 6 and 5 6 3:

- $3 > 1$  and  $1 < 4$ ,
- $5 < 6$  and  $6 > 5$ ,
- $6 > 5$  and  $5 = 5$  and  $5 < 6$ ,
- $5 < 6$  and  $6 > 3$ .



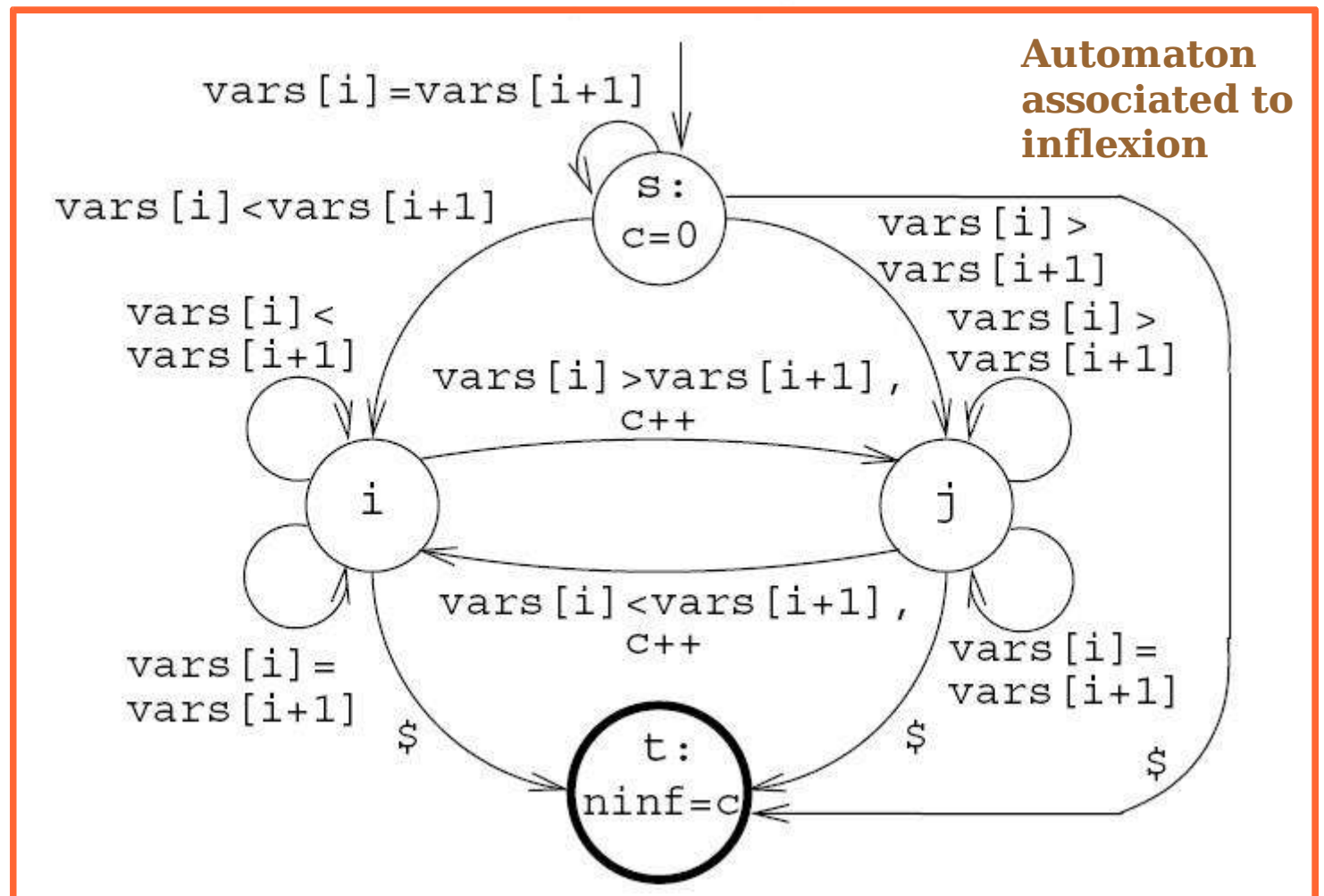
## EXAMPLE OF CONSTRAINT CHECKER (continued)

```
inflexion(ninf,vars[0..n-1]):BOOLEAN;
01 BEGIN
02   i=0; c=0;
03   WHILE i<n-1 AND vars[i]=vars[i+1] DO i++;
04   IF i<n-1 THEN less=(vars[i]<vars[i+1]);
05   WHILE i<n-1 DO
06     IF less THEN
07       IF vars[i]>vars[i+1] THEN c++; less=FALSE;
08     ELSE
09       IF vars[i]<vars[i+1] THEN c++; less=TRUE;
10     i++;
11   RETURN (ninf=c);
12 END.
```

# CONSTRAINT CHECKER

Uses a **deterministic automaton** with one single terminal state, but:

- Allows the use of **counters**.  
(value initialized in the initial state)  
(updated while triggering certain transitions)
- Final value of counters  
(e.g. value in the terminal state)  
can be **returned**.



# TRANSITIONS OF THE AUTOMATON

**Transitions** are labelled by a value in a range  $[\text{min}, \text{min}+p-1]$  or by \$.

Where do these values come from?

- To each constraint  $C$  we associate a **sequence of subsets**  $R_0, R_1, \dots, R_{m-1}$  of variables of  $C$  ( $R_0, R_1, \dots, R_{m-1}$  are called the **signature arguments** of  $C$ ),
- To the  $i$ -th subset corresponds the **signature variable**  $S_i$ ,
- The **link** between  $S_i$  and the variables of  $R_i$  is done according to  $p$  **mutually incompatible** conditions:
  - $c_1(R_i) \Leftrightarrow S_i = \text{min}$
  - $c_2(R_i) \Leftrightarrow S_i = \text{min} + 1$
  - .....
  - $c_p(R_i) \Leftrightarrow S_i = \text{min} + p - 1$

This **conjunction** is called the **signature constraint** and is denoted by  $\Psi_c(R_i, S_i)$ .

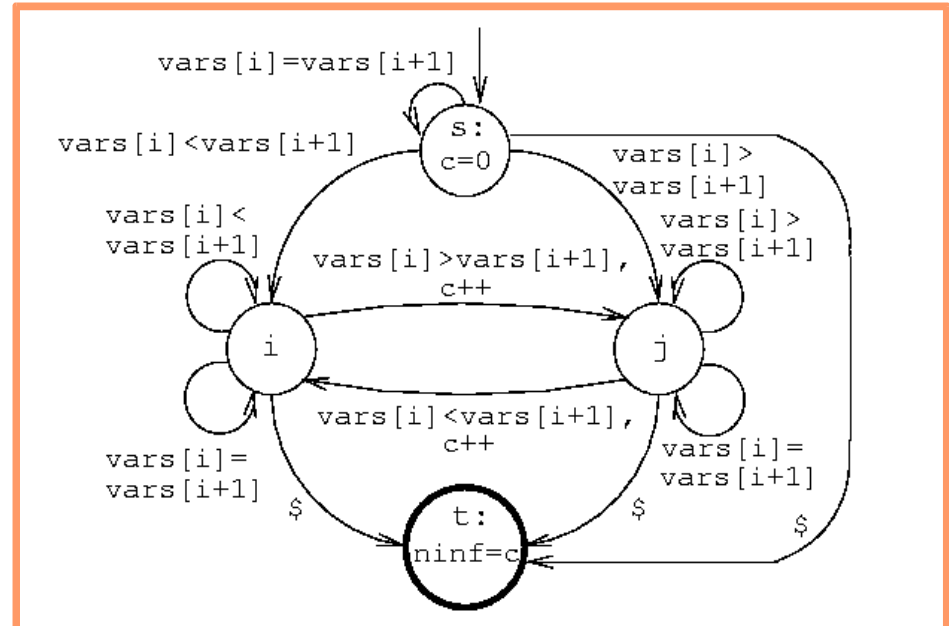
# EXAMPLE OF TRANSITIONS

**inflexion**(ninf,  $[x_0, x_1, x_2, x_3]$ )

$R_0 = \langle x_0, x_1 \rangle$   $R_1 = \langle x_1, x_2 \rangle$   $R_2 = \langle x_2, x_3 \rangle$

$\Psi_{\text{inflexion}}(S_i, x_i, x_{i+1}) : (x_i > x_{i+1} \Leftrightarrow S_i = 0) \wedge (x_i = x_{i+1} \Leftrightarrow S_i = 1) \wedge (x_i < x_{i+1} \Leftrightarrow S_i = 2)$

## Automaton associated to inflexion



# DESCRIPTION OF AN AUTOMATON

An automaton A of a constraint C is defined by:

- (1) **Signature** : signature variables of C
- (2) **SignatureDomain** : range of possible values of signature variables
- (3) **SignatureArg** : signature argument of C
- (4) **SignatureArgPattern** : symbolic names for arguments of **SignatureArg**
- (5) **Counters** : `t(Counter, InitialValue, FinalVariable)`
- (6) **States** : `source(id), sink(id), node(id)`
- (7) **Transitions** : `arc(id1,label,id2), arc(id1,label,id2,counters)`

# EXAMPLE OF DESCRIPTION OF AN AUTOMATON

For **inflexion**( $n_{inf}, vars[0..n-1]$ ) we have :

- (1) **Signature** :  $S_0, S_1, \dots, S_{n-2}$
- (2) **SignatureDomain** :  $0..2,$
- (3) **SignatureArg** :  $\langle vars[0], vars[1] \rangle, \dots, \langle vars[n-2], vars[n-1] \rangle$
- (4) **SignatureArgPattern** : not used
- (5) **Counters** :  $t(c, 0, n_{inf})$
- (6) **States** :  $source(s), node(i), node(j), sink(t)$
- (7) **Transitions** :  $arc(s, 1, s), \quad arc(i, 0, j, [C+1]), arc(s, 2, i),$   
 $arc(i, \$, t), \quad arc(s, 0, j), \quad arc(j, 1, j),$   
 $arc(s, \$, t), \quad arc(j, 0, j), \quad arc(i, 1, i),$   
 $arc(j, 2, i, [C+1]), arc(i, 2, i), \quad arc(j, \$, t).$

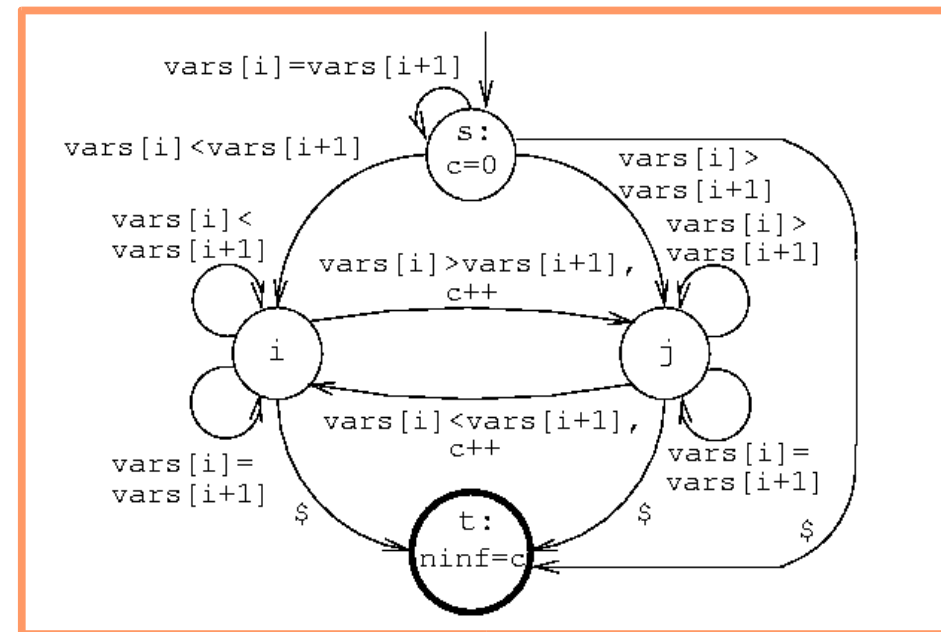


# RUNNING AN AUTOMATON ON A GROUND INSTANCE

For **inflexion**(4,[3,3,1,4,5,5,6,5,5,6,3]) :

**s,c=0** — {3=3  $\Leftrightarrow$   $S_0=1$ } — **s**  
 — {3>1  $\Leftrightarrow$   $S_1=0$ } — **j**  
 — {1<4  $\Leftrightarrow$   $S_2=2$ } — **i**  
     **[c=1]**<sup>2</sup>  
 — {4<5  $\Leftrightarrow$   $S_3=2$ } — **i**  
 — {5=5  $\Leftrightarrow$   $S_4=1$ } — **i**  
 — {5<6  $\Leftrightarrow$   $S_5=2$ } — **i**  
 — {6>5  $\Leftrightarrow$   $S_6=0$ } — **j**  
     **[c=2]**<sup>6</sup>  
 — {5=5  $\Leftrightarrow$   $S_7=1$ } — **j**  
 — {5<6  $\Leftrightarrow$   $S_8=2$ } — **i**  
     **[c=3]**<sup>8</sup>  
 — {6>3  $\Leftrightarrow$   $S_9=0$ } — **j**  
     **[c=4]**<sup>9</sup>  
 — {\$} — **t,ninf=c=4.**

Automaton associated to inflexion



1. INTRODUCTION

2. TYPE OF AUTOMATON USED

⇒ 3. **FROM AUTOMATA TO FILTERING ALGORITHMS**

4. APPLICATIONS

5. HANDLING RELAXATION FOR A COUNTER-FREE AUTOMATON

6. CONCLUSION AND PERSPECTIVES

## PRINCIPLE USED FOR FILTERING

Simulate **all potential executions** of an automaton according to the **current domain** of the variables in order to deduce **infeasible assignments**

How do we achieve this ?

By **reformulating** this as a conjunction of **signature** and **transition** constraints

## REFORMULATION

A global constraint C can be seen as the **conjunction** of **signature** and **transition** constraints :

$$\Psi_C(S_0, R_0) \quad \wedge \quad \Phi_C(\mathbf{s}, \vec{K}_0, S_0, Q_1, \vec{K}_1) \quad \wedge$$
$$\Psi_C(S_1, R_1) \quad \wedge \quad \Phi_C(Q_1, \vec{K}_1, S_1, Q_2, \vec{K}_2) \quad \wedge$$

.....

$$\Psi_C(S_{m-1}, R_{m-1}) \quad \wedge \quad \Phi_C(Q_{m-1}, \vec{K}_{m-1}, S_{m-1}, Q_m, \vec{K}_m) \quad \wedge$$
$$\Phi_C(Q_m, \vec{K}_m, \$, \mathbf{t}, \vec{K}_{m+1})$$

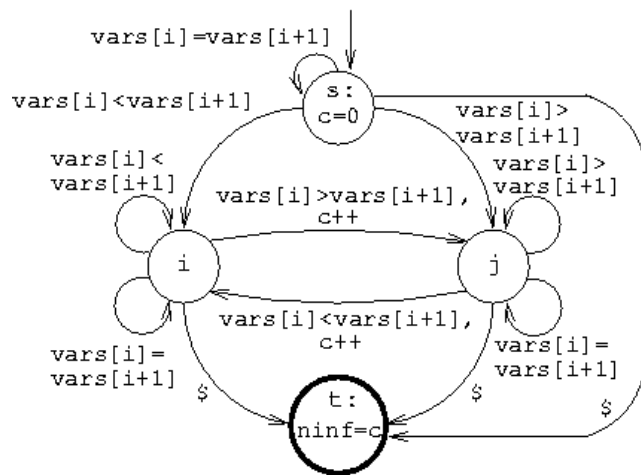
# ENCODING A TRANSITION CONSTRAINT $\Phi_C(Q_i, \vec{K}_i, S_i, Q_{i+1}, \vec{K}_{i+1})$

$$\bigvee \left( (Q_i = q_1) \wedge (S_i = s_1) \wedge (Q_{i+1} = q'_1) \wedge (\vec{K}_{i+1} = f_1(\vec{K}_i)) \right. \\
 \dots\dots\dots \\
 \left. (Q_i = q_{na}) \wedge (S_i = s_{na}) \wedge (Q_{i+1} = q'_{na}) \wedge (\vec{K}_{i+1} = f_{na}(\vec{K}_i)) \right)$$

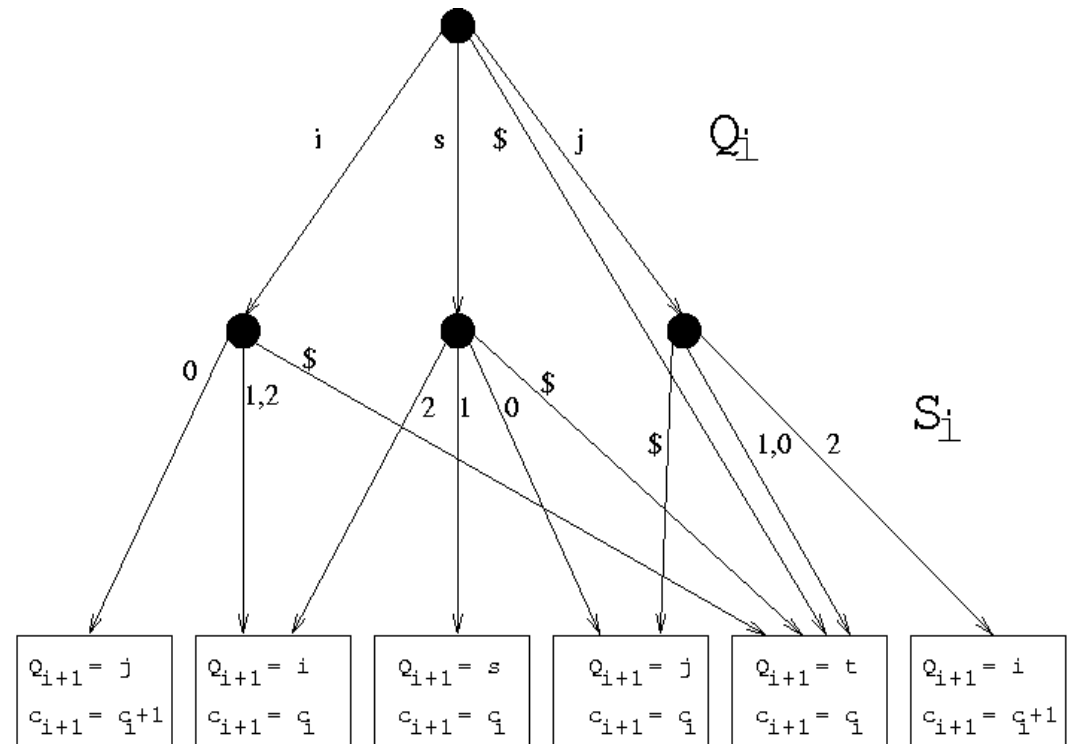
Encoded by one **case** constraint (SICStus) and possibly **element** and **arithmetic** constraints.  
 In ECLiPSe could use **generalized propagation**.

# EXAMPLE OF TRANSITION CONSTRAINT $\Phi_C(Q_i, \vec{K}_i, S_i, Q_{i+1}, \vec{K}_{i+1})$

automaton



decision tree



# CONSISTENCY

**PROPERTY:** If the constraint hypergraph associated with the reformulation is **Berge-acyclic** and **AC on each constraint**, then the full network is globally consistent [Janssen and Vilarem 88].

**FACT 1** : The **case** constraint **achieves AC**.

**FACT 2** : When no counter is used the transition constraint is encoded with **one single case** constraint.

**RESULT:** If we don't use any counter and no intersection between the signature arguments then the constraint hypergraph is Berge-acyclic.

If the constraint hypergraph is Berge-acyclic and the signature constraint achieves AC then **reformulation achieves AC**.

# PERFORMANCE

Compare the **hard coded**  $\leq_{lex}$  constraint described in [T2002-17]

vs. the **simulated**  $\leq_{lex}$  constraint on two problems:

- Finding **first solution** for the **Balanced Incomplete Block Design** [CSPlib]
- Finding **all solutions** for a **single**  $\leq_{lex}$  constraint

Problem	Built-in $\leq_{lex}$	Simulated $\leq_{lex}$
$v, b, r, k, \lambda$		
6, 50, 25, 3, 10	0.250	0.440
6, 60, 30, 3, 12	0.330	0.570
8, 14, 7, 4, 3	0.090	0.120
9, 120, 40, 4, 10	1.570	2.180
10, 90, 27, 3, 6	1.670	2.070
10, 120, 36, 3, 8	3.530	3.870
12, 88, 22, 3, 4	1.470	2.040
13, 104, 24, 3, 4	1.840	2.770
15, 70, 14, 3, 2	1.200	1.860

**Balanced Incomplete Block Design**

m	Built-in $\leq_{lex}$	Simulated $\leq_{lex}$
4	0.010	0.020
5	0.110	0.170
6	1.640	2.300
7	29.530	39.100

**single  $\leq_{lex}$  constraint**

Time in seconds (uses SICStus Prolog 3.11 on a 600MHz Pentium III)



1. INTRODUCTION

2. TYPE OF AUTOMATON USED

3. FROM AUTOMATA TO FILTERING ALGORITHMS

⇒ 4. APPLICATIONS

5. HANDLING RELAXATION FOR A COUNTER-FREE AUTOMATON

6. CONCLUSION AND PERSPECTIVES

# DESIGN OF FILTERING ALGORITHMS

<ftp://ftp.sics.se/pub/SICS-reports/Reports/SICS-T--2004-08--SE.pdf> provides an automaton for the following constraints :

- Unary** constraints : in, not\_in.
- Channeling** constraints : domain\_constraint.
- Counting** constraints : among, atleast, atmost, count.
- Sliding sequence** constraints : change, longest\_change, smooth.
- Variations around **element** : element, element\_greatereq, element\_lesseq, element\_sparse.
- Variations around **maximum** : maximum, max\_index.
- Constraints on **words** : global\_contiguity, group, group\_skip\_isolated\_item, pattern.
- Constraints on **vectors** : between,  $\leq_{\text{lex}}$ , lex\_different, differ\_from\_at\_least\_k\_pos.
- Geometrical** constraints : two\_quad\_are\_in\_contact, two\_quad\_do\_not\_overlap.
- Constraint on a **sequence** : inflexion, top, valley.
- Miscellaneous** constraints : in\_same\_partition, not\_all\_equal, sliding\_card\_skip0.

# AVAILABLE AUTOMATA

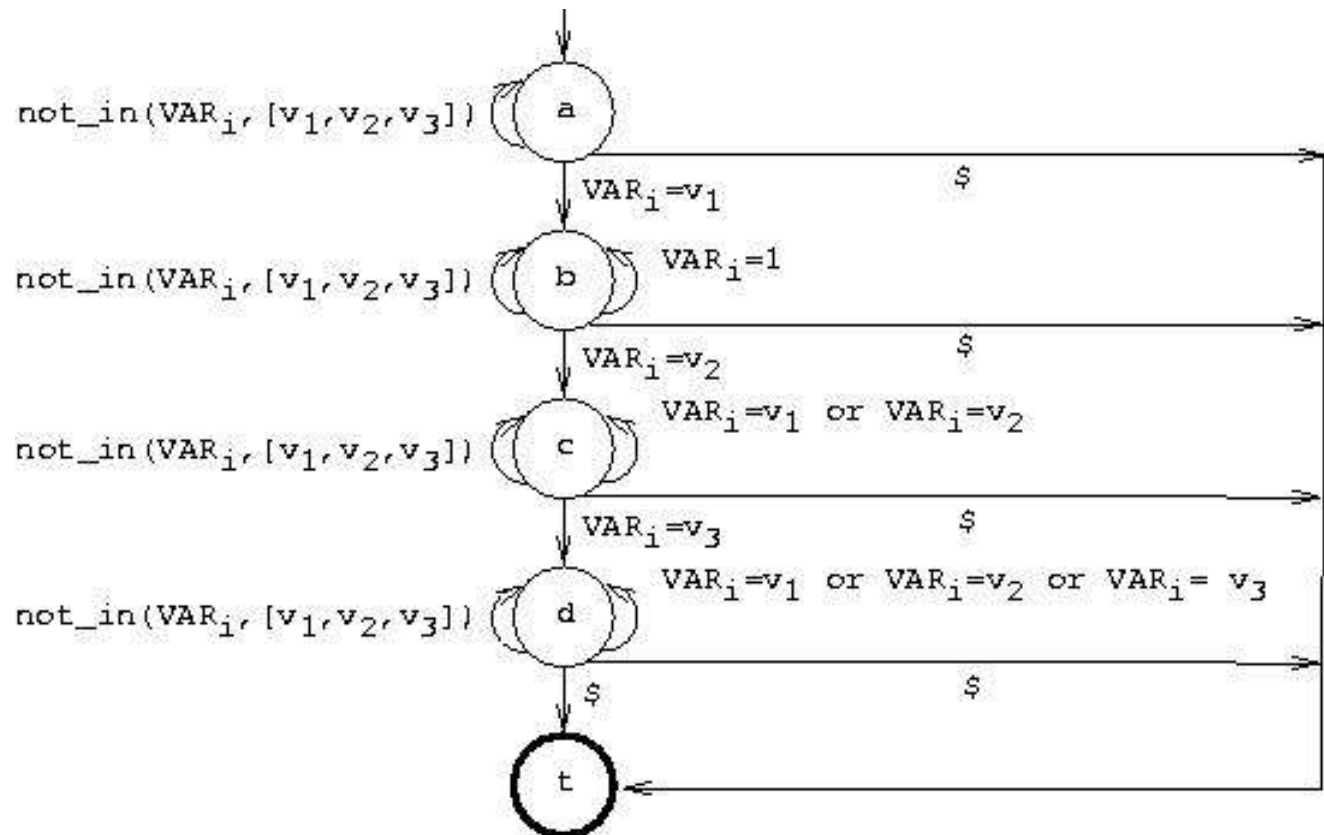
alldifferent  
alldifferent\_except\_0  
alldifferent\_interval  
alldifferent\_modulo  
alldifferent\_on\_intersection  
alldifferent\_same\_value  
among  
among\_diff\_0  
among\_interval  
among\_low\_up  
among\_modulo  
arith  
arith\_or  
arith\_sliding  
assign\_and\_counts  
atleast  
atmost  
balance  
balance\_interval  
balance\_modulo  
bin\_packing  
cardinality\_atleast  
cardinality\_atmost  
change  
change\_continuity  
change\_pair  
circular\_change  
count  
counts  
cumulative  
cyclic\_change  
cyclic\_change\_joker  
decreasing  
deepest\_valley  
differ\_from\_at\_least\_k\_pos  
disjoint  
distance\_change  
domain\_constraint  
elem  
element  
element\_greatereq  
element\_lesseq  
element\_matrix  
element\_sparse  
exactly  
global\_cardinality  
global\_contiguity  
group  
group\_skip\_isolated\_item  
highest\_peak  
in  
in\_same\_partition  
increasing  
inflexion  
**int\_value\_precede**  
interval\_and\_count  
interval\_and\_sum  
inverse  
ith\_pos\_different\_from\_0  
lex\_between  
lex\_different  
lex\_greater  
lex\_greatereq  
lex\_less  
lex\_lesseq  
longest\_change  
max\_index  
max\_nvalue  
maximum  
min\_index  
min\_n  
min\_nvalue  
minimum  
minimum\_except\_0  
minimum\_greater\_than  
next\_element  
no\_peak  
no\_valley  
not\_all\_equal  
not\_in  
nvalue  
peak  
same  
sequence\_folding  
sliding\_card\_skip0  
smooth  
stage\_element  
strictly\_decreasing  
strictly\_increasing  
two\_orth\_are\_in\_contact  
two\_orth\_do\_not\_overlap  
used\_by  
valley

# INTEGER VALUE PRECEDENCE (all-pairs)

[CP 2004]

Yat Chiu Lee, Jimmy H.M. Lee

```
int_value_precede([v1, v2, ..., vm], [VAR1, VAR2, ..., VARn])
```



Automaton for  $m=3$

# FILTERING ALGORITHM FOR A CONJUNCTION OF GLOBAL CONSTRAINTS

Given **two global constraints** and their corresponding automata and signature constraint we have to:

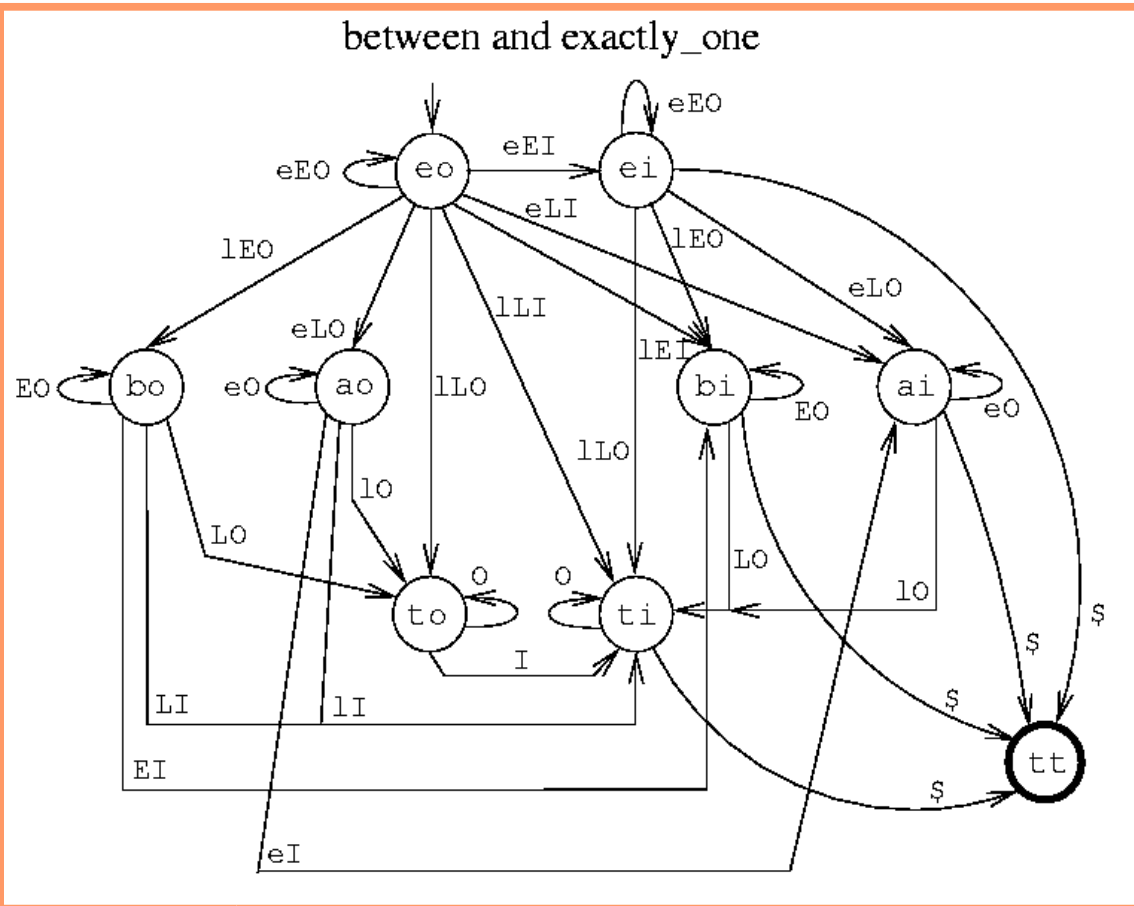
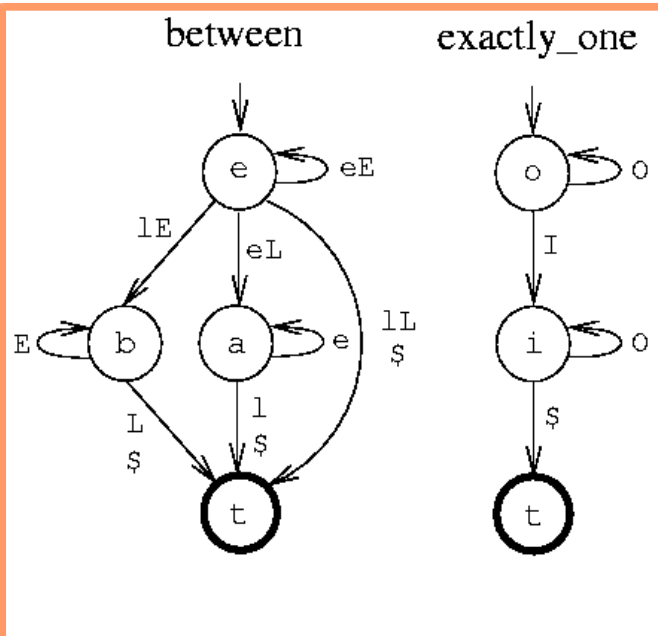
- Compute the **new signature constraint** associated to their conjunction,
- Compute the **product of the two automata**.

## between( $\vec{a}, \vec{x}, \vec{b}$ ) and exactly\_one( $\vec{x}, \text{values}$ )

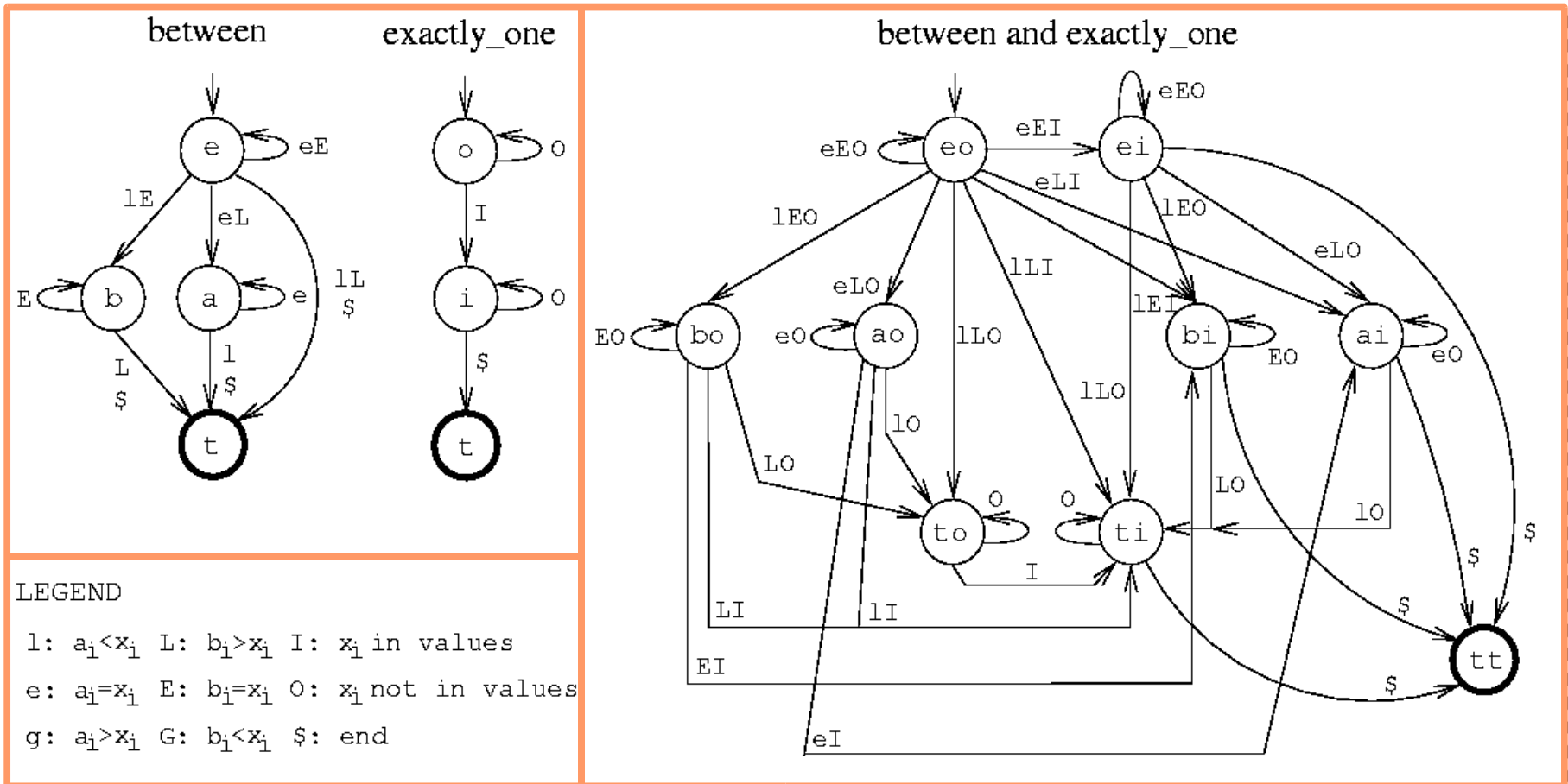
Transition constraint for the conjunction **combines** the following set of conditions:  
{ $a_i < x_i$ ,  $a_i = x_i$ ,  $a_i > x_i$ }, { $b_i > x_i$ ,  $b_i = x_i$ ,  $b_i < x_i$ }, { $x_i \in \text{values}$ ,  $x_i \notin \text{values}$ }.

<b>0</b> if $a_i < x_i \wedge b_i > x_i \wedge x_i \notin \text{values}$ ,	<b>9</b> if $a_i < x_i \wedge b_i > x_i \wedge x_i \in \text{values}$ ,
<b>1</b> if $a_i < x_i \wedge b_i = x_i \wedge x_i \notin \text{values}$ ,	<b>10</b> if $a_i < x_i \wedge b_i = x_i \wedge x_i \in \text{values}$ ,
<b>2</b> if $a_i < x_i \wedge b_i < x_i \wedge x_i \notin \text{values}$ ,	<b>11</b> if $a_i < x_i \wedge b_i < x_i \wedge x_i \in \text{values}$ ,
<b>3</b> if $a_i = x_i \wedge b_i > x_i \wedge x_i \notin \text{values}$ ,	<b>12</b> if $a_i = x_i \wedge b_i > x_i \wedge x_i \in \text{values}$ ,
<b>4</b> if $a_i = x_i \wedge b_i = x_i \wedge x_i \notin \text{values}$ ,	<b>13</b> if $a_i = x_i \wedge b_i = x_i \wedge x_i \in \text{values}$ ,
<b>5</b> if $a_i = x_i \wedge b_i > x_i \wedge x_i \notin \text{values}$ ,	<b>14</b> if $a_i = x_i \wedge b_i > x_i \wedge x_i \in \text{values}$ ,
<b>6</b> if $a_i > x_i \wedge b_i > x_i \wedge x_i \notin \text{values}$ ,	<b>15</b> if $a_i > x_i \wedge b_i > x_i \wedge x_i \in \text{values}$ ,
<b>7</b> if $a_i > x_i \wedge b_i = x_i \wedge x_i \notin \text{values}$ ,	<b>16</b> if $a_i > x_i \wedge b_i = x_i \wedge x_i \in \text{values}$ ,
<b>8</b> if $a_i > x_i \wedge b_i < x_i \wedge x_i \notin \text{values}$ ,	<b>17</b> if $a_i > x_i \wedge b_i < x_i \wedge x_i \in \text{values}$ .

# between( $\vec{a}, \vec{x}, \vec{b}$ ) and exactly\_one( $\vec{x}, \text{values}$ ) (continued)



LEGEND  
 l:  $a_i < x_i$  L:  $b_i > x_i$  I:  $x_i$  in values  
 e:  $a_i = x_i$  E:  $b_i = x_i$  O:  $x_i$  not in values  
 g:  $a_i > x_i$  G:  $b_i < x_i$  \$: end



## EXAMPLE OF PRUNING

$u \in \{0,1\}, v \in \{0,3\}, w \in \{0,1,2,3\}$

between( $\langle 0,3,1 \rangle, \langle u,v,w \rangle, \langle 1,0,2 \rangle$ ) and exactly\_one( $\langle u,v,w \rangle, \{0\}$ )

$(u=1, v=0, w=0)$ : unique solution such that  $w=0$

Finding out that  $w \neq 0$  requires to **reason globally on both constraints**:

After two transitions, the automaton will be either in state **ai** or in state **bi**. In either state, a 0 must already have been seen, and so there is no support for  $w=0$ .



1. INTRODUCTION

2. TYPE OF AUTOMATON USED

3. FROM AUTOMATA TO FILTERING ALGORITHMS

4. APPLICATIONS

⇒ 5. **HANDLING RELAXATION FOR A COUNTER-FREE AUTOMATON**

6. CONCLUSION AND PERSPECTIVES

# VIOLATION COST OF A CONSTRAINT

**Minimum** number of subsets of its signature argument for which it is necessary to change at least one variable in order to get back to a solution.

## EXAMPLE

**global\_contiguity(vars[0..m-1]):**

At most one sequence of consecutive 1 in a sequence of 0-1 variables.

$V_0 \in \{0,1\}, V_1 \in \{1\}, V_2 \in \{1\}, V_3 \in \{0\}, V_4 \in \{1\}, V_5 \in \{0,1\}, V_6 \in \{1\},$

$\text{global\_contiguity}([V_0, V_1, V_2, V_3, V_4, V_5, V_6])$  **is violated** (since 2 sequences of 1).

$V_0 \in \{0,1\}, V_1 \in \{1\}, V_2 \in \{1\}, V_3 \in \{0\}, V_4 \in \{1\}, V_5 \in \{0,1\}, V_6 \in \{1\}, \text{cost} \in \{0, \mathbf{1}\},$

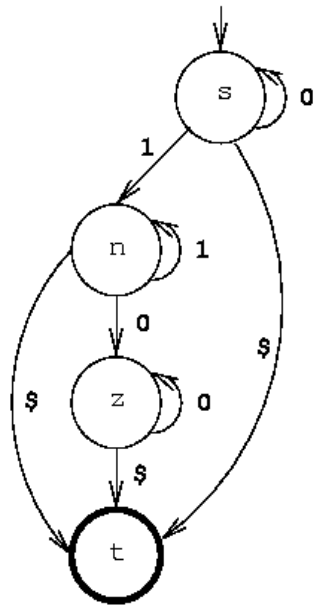
$\text{soft\_global\_contiguity}([V_0, V_1, V_2, V_3, V_4, V_5, V_6], \text{cost})$

**min(cost)=1** (since becomes feasible if turn  $V_3$  to 1)

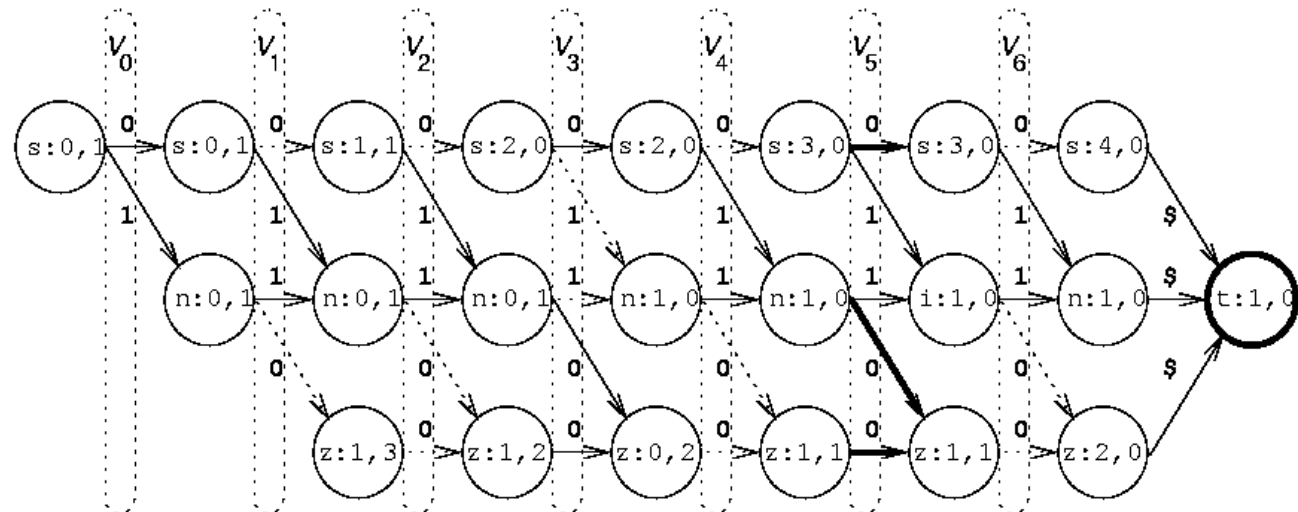
**$V_5 \neq 0$**  (since at most one variable can be changed)

# EVALUATING THE MINIMUM NUMBER OF VIOLATIONS

- STEP1:** Construct graph of all potential executions of the automaton  
**STEP2:** Compute the path from source to sink with minimum number of violations (use a topological sort)



(A) Automaton for global\_contiguity



(B) Graph of potential executions of the automaton of global\_contiguity according to  $V_0, V_1, V_2, V_3, V_4, V_5, V_6$

# FILTERING ACCORDING TO THE MAXIMUM NUMBER OF VIOLATIONS

## NOTATION:

**before** $[n_k]$  : minimum number of infeasible arcs on all paths from **source** to  $n_k$ ,

**after**  $[n_k]$  : minimum number of infeasible arcs on all paths from  $n_k$  to **sink**,

**Min** : minimum violation cost from the **source** to the **sink**,

**Min** $_1^i$  : minimum violation cost according to the hypothesis that  $S_1=i$ ,

$A_1^i$  : set of arcs, labeled by  $i$ , for which the origin has a rank of  $1$ .

## OBSERVATIONS:

$\min(\mathbf{before}[a] + \mathbf{after}[b])$  is the minimum violation under the hypothesis that  $S_1$  remains assigned to  $i$ .  $a \rightarrow b$  in  $A_1^i$ .

If previous cost is greater than **Min** then no path from source to sink which uses an arc of  $A_1^i$  and which has a cost of **Min**.

## RESULT:

$$\mathbf{Min}_1^i = \min(\min(\mathbf{before}[a] + \mathbf{after}[b]), \mathbf{Min} + 1)$$
$$a \rightarrow b \in A_1^i$$

1. INTRODUCTION
2. TYPE OF AUTOMATON USED
3. FROM AUTOMATA TO FILTERING ALGORITHMS
4. APPLICATIONS
5. HANDLING RELAXATION FOR A COUNTER-FREE AUTOMATON
- ⇒ 6. **CONCLUSION AND PERSPECTIVES**

# CONCLUSION

**RESULT:** Derive **automatically** a filtering algorithm from a constraint checker.

**CONSEQUENCE:** **Correctness** of the filtering algorithm relies **only** on the correctness of the checker.

## **METHOD:**

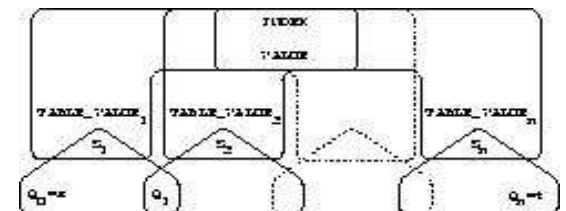
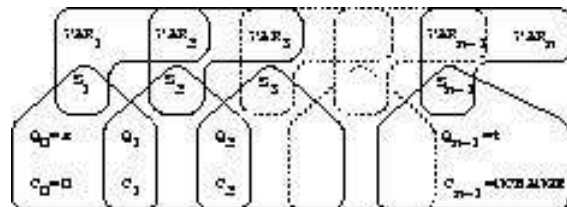
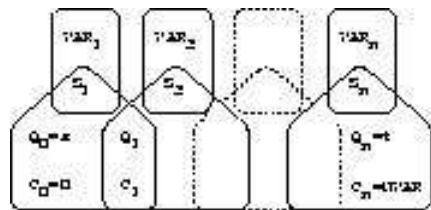
- Automaton **accepting solutions** of the constraint,
- Encode all potential executions of the automaton as a **conjunction of signature and transition constraints**.

**PROPERTY:** Achieves **arc-consistency** under some restrictions.

**An other way (versus graph properties) to describe some constraint.**

# PERSPECTIVES

- **Extend** the automaton,
- **Characterize** other properties where we currently achieve arc-consistency,
- Make the link with other topics such as **explanation**, **model checking**,
- **Enhance** the propagation for typical **recurring hypergraphs**:



## OPPORTUNITY:

An opportunity for more interaction between global constraints and non-binary constraint networks (**structured** constraint networks).

## **MORE INFORMATION**

Technical report describing the method and giving 40 automata available at:

<http://www.sics.se/library>

<ftp://ftp.sics.se/pub/SICS-reports/Reports/SICS-T--2004-08--SE.pdf>

Definition of constraints:

<http://www.sics.se/library>

<ftp://ftp.sics.se/pub/SICS-reports/Reports/SICS-T--2000-01--SE.pdf>

updated (draft):

<http://www.sics.se/isl/cps/>