DFA-based formulation for constraint negation

Nadjib Lazaar, Nourredine Aribi, Arnaud Gotlieb, Yahia Lebbah

ModRef Workshop
8 Oct. 2012

Motivations

- Many CP-application domains are coming to challenge modern CP languages and their global constraints tool-box:
  - Program verification [Collavizza et al.,10] [Lazaar et al.,10]
  - Reinforcement learning [Bessière et al.,07]

Motivations

- Many CP-application domains are coming to challenge modern CP languages and their global constraints tool-box:
  - Program verification [Collavizza et al.,10][Lazaar et al.,10]
  - Reinforcement learning [Bessière et al.,07]

The need: Global constraint negation!

Motivations

- Many CP-application domains are coming to challenge modern CP languages and their global constraints tool-box:
  - Program verification [Collavizza et al.,10][Lazaar et al.,10]
  - Reinforcement learning [Bessière et al.,07]

The need: Global constraint negation!

How:

- Reformulation (e.g., negation of an atLeast is an atMost)
- Decomposition and syntactic transformation:

Motivations

- Many CP-application domains are coming to challenge modern CP languages and their global constraints tool-box:
  - Program verification [Collavizza et al.,10][Lazaar et al.,10]
  - Reinforcement learning [Bessière et al.,07]

The need: Global constraint negation!

How:

- Reformulation (e.g., negation of an ATLEAST is an ATMOST)
- Decomposition and syntactic transformation:

$\text{INVERSE}(tab_1, tab_2)$ Rewrite

$$\forall i \in D_1 : \quad tab_1[tab_2[i]] = i$$
$$\forall j \in D_2 : \quad tab_2[tab_1[j]] = j$$

Motivations

- Many CP-application domains are coming to challenge modern CP languages and their global constraints tool-box:
  - Program verification [Collavizza et al.,10][Lazaar et al.,10]
  - Reinforcement learning [Bessière et al.,07]

The need: Global constraint negation!

How:

- Reformulation (e.g., negation of an ATLEAST is an ATMOST)
- Decomposition and syntactic transformation:

$\text{INVERSE}(tab_1, tab_2) \xrightarrow{\text{Rewrite}} \boxed{\begin{array}{ll} \forall i \in D_1 : & tab_1[tab_2[i]] = i \\ \forall j \in D_2 : & tab_2[tab_1[j]] = j \end{array}} \xrightarrow{\text{Negation}} \boxed{\begin{array}{ll} \exists i \in D_1 : & tab_1[tab_2[i]] \neq i \\ \exists j \in D_2 : & tab_2[tab_1[j]] \neq j \end{array}}$

Motivations

- More and more generic constraint representations and reformulations are proposed:
  - Deterministic finite automaton (DFA) [Pesant,04]
  - Multivalued Decision Diagram (MDD) [Andersen et al.,07][Hoda et al.,10]
  - Grammar [Quimper et al., 06][Katsirelos et al.,09]
  - Berge-acyclic graph [Beldiceanu et al., catalog11]
  - …

Motivations

- More and more generic constraint representations and reformulations are proposed:
  - Deterministic finite automaton (DFA) [Pesant,04]
  - Multivalued Decision Diagram (MDD) [Andersen et al.,07][Hoda et al.,10]
  - Grammar [Quimper et al., 06][Katsirelos et al.,09]
  - Berge-acyclic graph [Beldiceanu et al., catalog11]
  - …

The need

- Generic global constraint negation

Motivations

- More and more generic constraint representations and reformulations are proposed:
  - **Deterministic finite automaton (DFA)** [Pesant,04]
  - Multivalued Decision Diagram (MDD) [Andersen et al.,07][Hoda et al.,10]
  - Grammar [Quimper et al., 06][Katsirelos et al.,09]
  - Berge-acyclic graph [Beldiceanu et al., catalog11]
  - …

The need

- Generic global constraint negation

This paper

- **DFA-based global constraint negation**

REGULAR constraint

- DFAs accept precisely regular languages

- $\text{REGULAR}(x_1 \ldots x_n, \mathcal{A})$ holds iff:
  - $x_1 \ldots x_n$ is a string accepted by the deterministic finite automaton $\mathcal{A}$

- Many global constraints are instances of REGULAR
  - AMONG, CONTIGUITY, LEX, PRECEDENCE …

- 3-stages for the consistency algorithm ($\mathcal{O}(nd|E|)$)
  - Forward step
  - Backward step
  - Maintaining step

Negation on DFA-based GC

Let $\mathcal{A} = (X, E, \Psi, \Sigma, \delta, e_0, F)$ a DFA of a given constraint C:

- X a sequence of finite-domain variables
- E a finite set of states ei
- Ψ a finite set of labeled states:(*source*, *node*, *sink* and *final* states)
- Σ a finite alphabet
- δ a transition function
- e_0 the start state
- F a subset of final states

Negation on DFA-based GC

Property [hopcroft et al.,06]: The **complement** of a regular language is regular

Negation on DFA-based GC

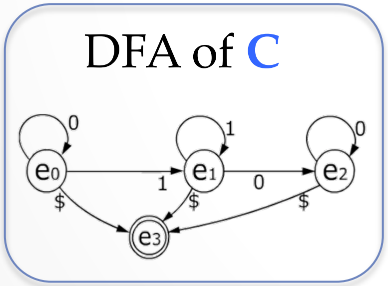
Property [hopcroft et al.,06]: The **complement** of a regular language is regular

Proposition: The **complement** of a DFA of a given constraint C represents the DFA of the negated form notC

Negation on DFA-based GC

Property [hopcroft et al.,06]: The **complement** of a regular language is regular

Proposition: The **complement** of a DFA of a given constraint C represents the DFA of the negated form notC



Negation on DFA-based GC

Property [hopcroft et al.,06]: The **complement** of a regular language is regular

Proposition: The **complement** of a DFA of a given constraint C represents the DFA of the negated form notC

Negation on DFA-based GC

We get the **complement** of a global constraint DFA's
By combining the 3 following operators:

Negation on DFA-based GC

We get the **complement** of a global constraint DFA's
By combining the 3 following operators:

- Complete automaton operator C(A)

The complete automaton of $\mathcal{A} = (X, E, \Psi, \Sigma, \delta, e_0, F)$ is $\mathcal{C}(\mathcal{A}) =$ $(X, E', \Psi', \Sigma, \delta', e_0, F)$ s.t.:

- $E' = E \cup \{e_k : \exists s \in \Sigma, e_i \in E \; s.t. \; (e_i, s, e_k) \notin \delta\}$
- $\Psi' = \Psi \cup \{sink(e_k) : e_k \in E' \backslash E\}$
- $\delta' = \delta \cup \{(e_i, s, e_k) : \exists s \in \Sigma, e_i \in E', e_k \notin E\}$

Negation on DFA-based GC

We get the **complement** of a global constraint DFA's
By combining the 3 following operators:

- Complete automaton operator C(A)
- Swap-state operator S(A)

A swap state on $\mathcal{A} = (X, E, \Psi, \Sigma, \delta, e_0, F)$ is $\mathcal{S}(\mathcal{A}) = (X, E, \psi', \Sigma, \delta, e_0, F')$ s.t.:

$$\forall e_i, e_j \in E : \; final(e_i), sink(e_j) \in \Psi \; \Rightarrow sink(e_i), final(e_j) \in \Psi'$$

Negation on DFA-based GC

We get the **complement** of a global constraint DFA's
By combining the 3 following operators:

- Complete automaton operator C(A)
- Swap-state operator S(A)
- Clean-up operator U(A)

A clean-up on $\mathcal{A} = (X, E, \Psi, \Sigma, \delta, e_0, F)$ is $\mathcal{U}(\mathcal{A}) = (X, E', \Psi', \Sigma, \delta', e_0, F)$ s.t.:

- $E' = E \backslash \{e_k : sink(e_k) \in \Psi\}$
- $\Psi' = \Psi \backslash \{sink(e_k) : e_k \in E\}$
- $\delta' = \delta \setminus \{(e_i, s, e_k) \in E \times \Sigma \times E \ : \ sink(e_k) \in \Psi\}$

Negation on DFA-based GC

We get the **complement** of a global constraint DFA's
By combining the 3 following operators:

- Complete automaton operator C(A)
- Swap-state operator S(A)
- Clean-up operator U(A)

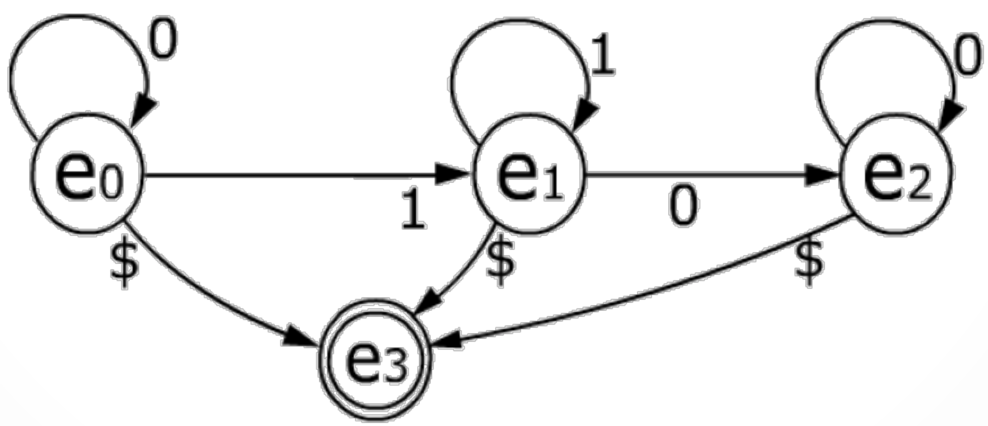
$\bar{\mathcal{A}}$ is the complement of $\mathcal{A}$ s.t.:

$$\boxed{\bar{\mathcal{A}} = \mathcal{U}(\mathcal{S}(\mathcal{C}(\mathcal{A})))}$$

Example (GLOBAL_CONTIGUITY)

- GLOBAL-CONTIGUITY($x_1 \ldots x_n$), This constraint enforce all variables $x_1 \ldots x_n$ to be assigned value 0 or 1, and all variables assigned to value 1 appear contiguously.

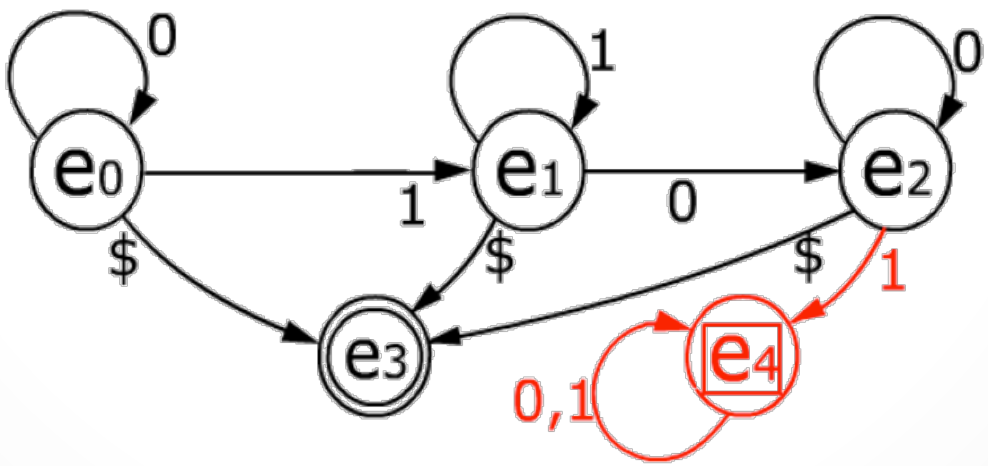
GLOBAL-CONTIGUITY DFA:



Example (GLOBAL_CONTIGUITY)

- GLOBAL-CONTIGUITY($x_1 \ldots x_n$), This constraint enforce all variables $x_1 \ldots x_n$ to be assigned value 0 or 1, and all variables assigned to value 1 appear contiguously.

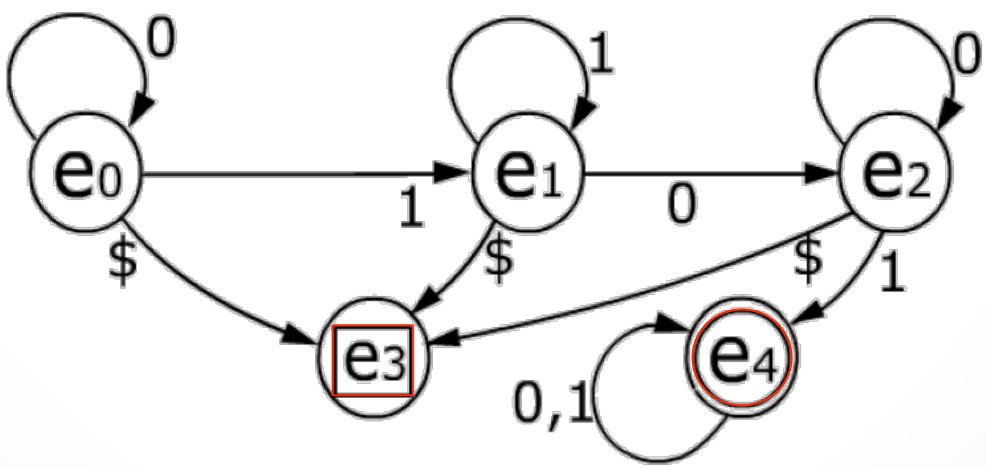
1- Complete:



Example (GLOBAL_CONTIGUITY)

- **GLOBAL-CONTIGUITY($x_1 \ldots x_n$),** This constraint enforce all variables $x_1 \ldots x_n$ to be assigned value 0 or 1, and all variables assigned to value 1 appear contiguously.

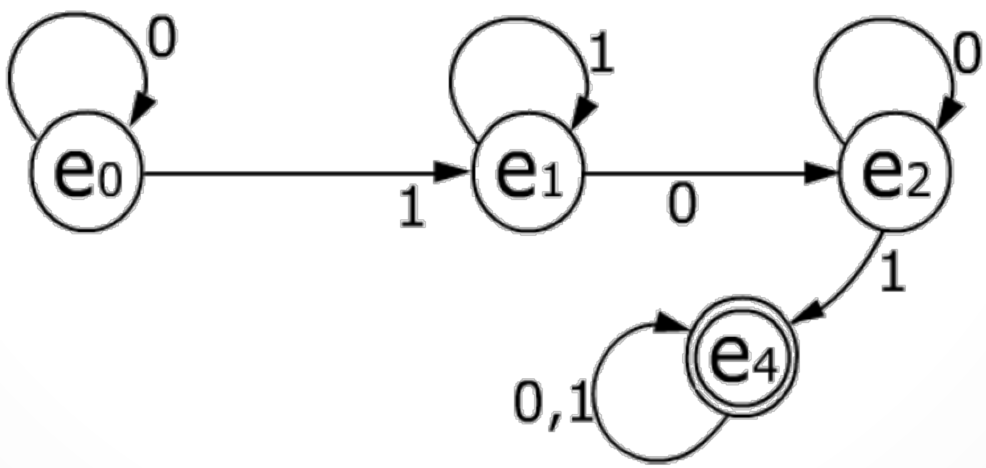
2- Swap-state:



Example (GLOBAL_CONTIGUITY)

- GLOBAL-CONTIGUITY($x_1 \ldots x_n$), This constraint enforce all variables $x_1 \ldots x_n$ to be assigned value 0 or 1, and all variables assigned to value 1 appear contiguously.

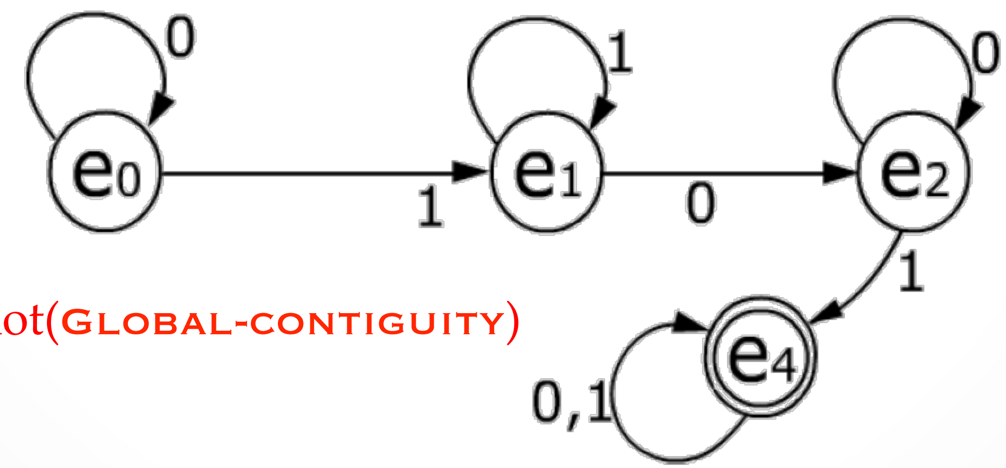
3- Clean-up:



Example (Global_contiguity)

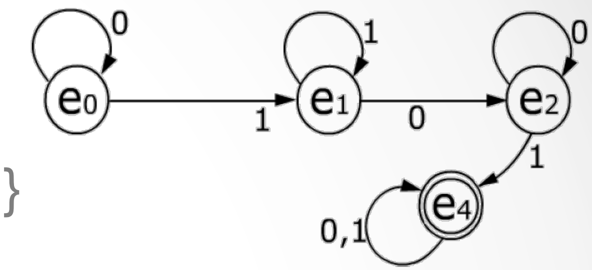
- Global-contiguity($x_1 \ldots x_n$), This constraint enforce all variables $x_1 \ldots x_n$ to be assigned value 0 or 1, and all variables assigned to value 1 appear contiguously.

3- Clean-up:

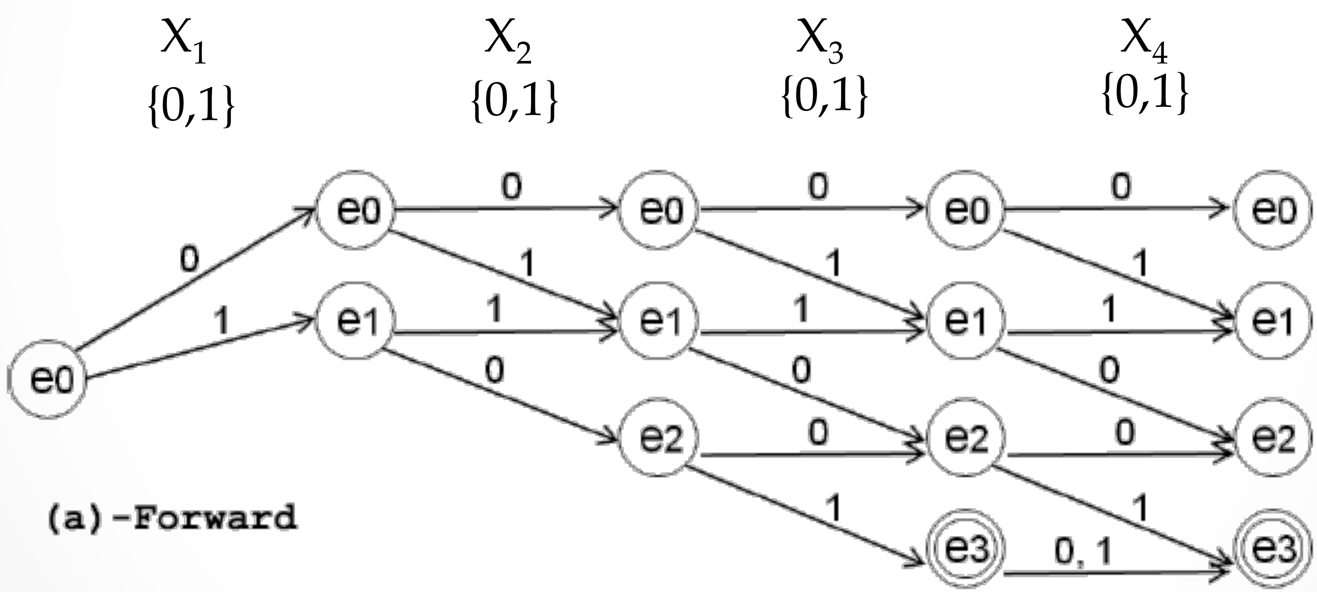


DFA of not(Global-contiguity)

Example (Global_contiguity)

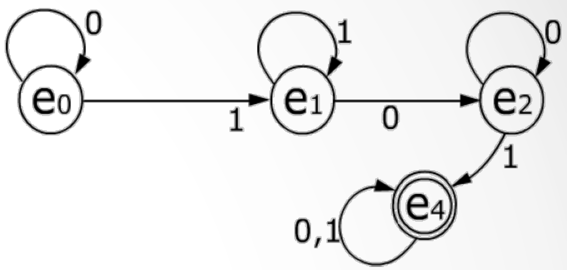


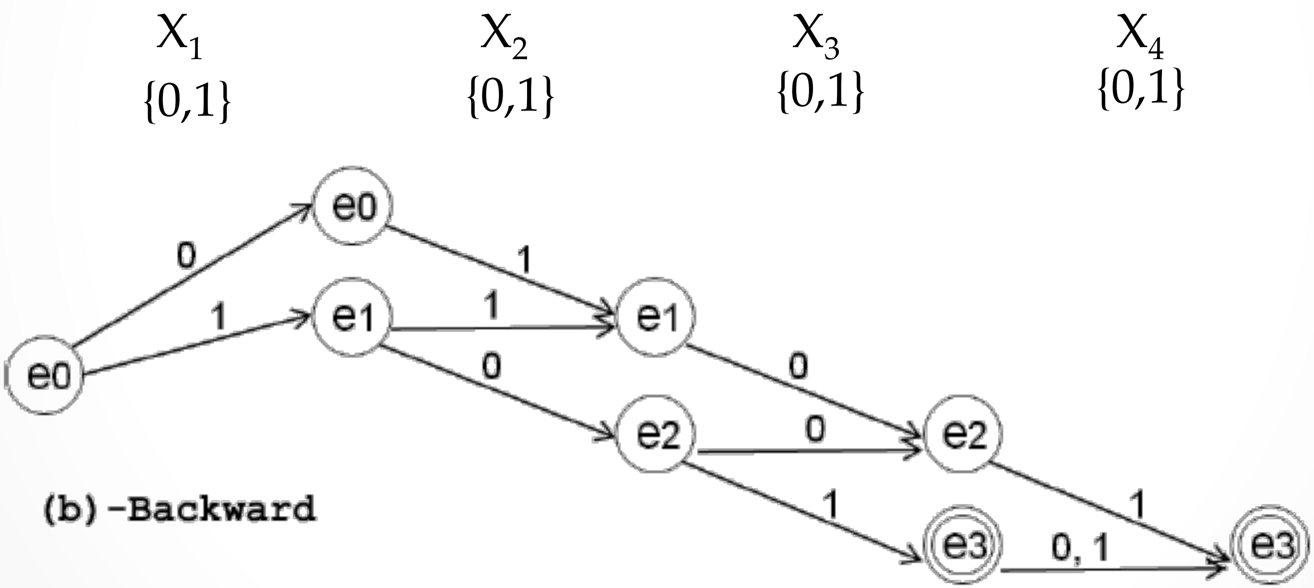
- Filtering using **regular:**
  - Let x_1 to x_4 sequence of variables in {0,1}



Example (Global_contiguity)

- Filtering using regular:
  - Let x_1 to x_4 sequence of variables in {0,1}

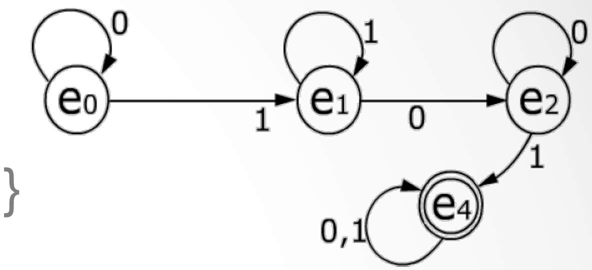




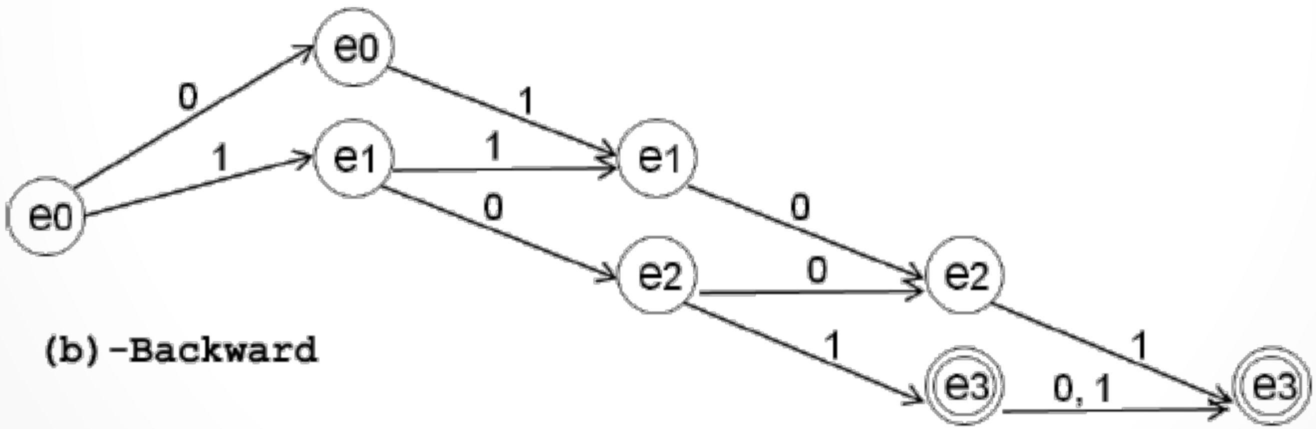
(b)-Backward

Example (Global_contiguity)

- Filtering using regular:
  - Let x_1 to x_4 sequence of variables in {0,1}

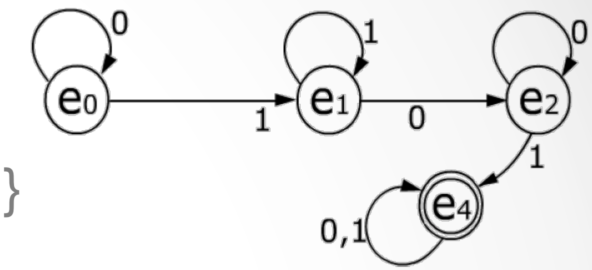


X_1	X_2	X_3	X_4
{0,1}	{0,1}	$\{\cancel{0},1\}$	{0,1}



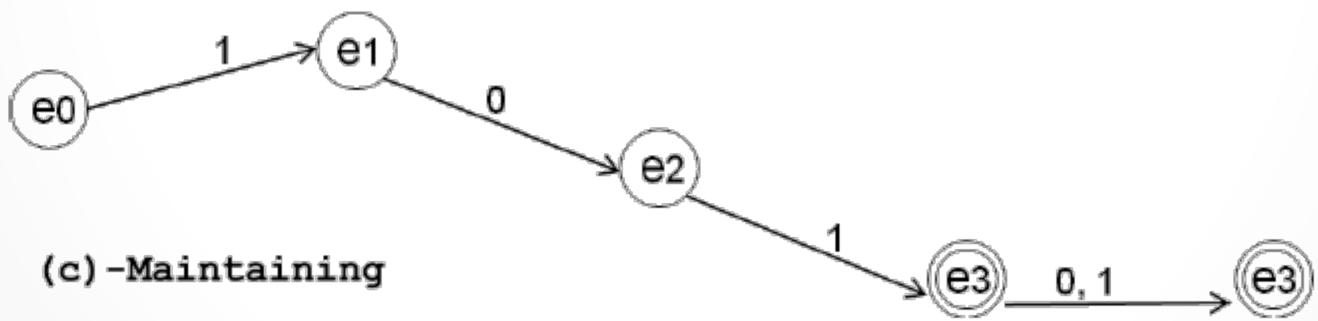
(b)-Backward

Example (GLOBAL_CONTIGUITY)



- Filtering using REGULAR:
  - Let x_1 to x_4 sequence of variables in {0,1}

X_1	X_2	X_3	X_4
{1}	{0}	{1}	{0,1}



Experimental validation

- GLOBAL_CONTIGUITY

Decompisition of GLOBAL_CONTIGUITY in primitive constraints gives:

$$\texttt{global\_contiguity}(\texttt{x}) \equiv \forall i, j \in 1..n : i < j \; s.t. \\ (x_i = 1) \wedge (x_j = 0) \Rightarrow (\forall k \in j+1..n : x_k = 0)$$

Syntactic negation:

$$\neg\texttt{global\_contiguity}(\texttt{x}) \equiv \exists i, j \in 1..n : i < j \; s.t. \\ (x_i = 1) \wedge (x_j = 0) \wedge (\exists k \in j+1..n : x_k = 1)$$

Experimental validation

- GLOBAL_CONTIGUITY

var	syntactic transformations based negation				DFA − based negation			
	T	M	P	N	T	M	P	N
200	53.13	24.39	65 111	396	2.09	0.42	366	397
300	186.07	77.90	116 535	496	3.31	0.77	468	497
400	509.93	179.22	170 209	596	4.81	1.559	566	597
500	1 082.35	344.86	244 235	696	6.69	2.20	668	697
600	1 936.68	589.79	315 320	796	8.77	2.96	766	797
700	3 125.01	930.34	411 935	896	11.43	4.80	868	897
800	4 735.71	1 381.68	500 407	996	14.19	6.03	966	997
900	6 760.44	1 960.24	619 627	1 096	17.27	7.28	1 068	1 097
1 000	19 407.02	2 681.01	725 507	1196	22.88	8.53	1 166	1 197
1 100	—	OOM	—	—	24.17	9 925	1 268	1 297
1 200	—	OOM	—	—	28.67	14 214	1 366	1 397
1 300	—	OOM	—	—	32.78	16 330	1 468	1 497
1 400	—	OOM	—	—	37.58	18 766	1 566	1 597
1 500	—	OOM	—	—	42.69	21 266	1 668	1 697
1 600	—	OOM	—	—	47.83	23 702	1 766	1 797
1 700	—	OOM	—	—	53.34	26 276	1 868	1 897
1 800	—	OOM	—	—	59.32	28 712	1 966	1 997
1 900	—	OOM	—	—	65.09	31 212	2 068	2 097
11 000	—	OOM	—	—	1 923.53	896 958	11 166	11 197

T : time(ms), M : memory(MB), P : propagations, N : nodes, OOM : Out − Of − Memory

Intel Core2Duo CPU, Q6600 of 2.4Ghz with 3Gb of RAM

Experimental validation

- Global_contiguity

var	syntactic transformations based negation				DFA — based negation			
	T	M	P	N	T	M	P	N
200	53.13	24.39	65 111	396	2.09	0.42	366	397
300	186.07	77.90	116 535	496	3.31	0.77	468	497
400	509.93	179.22	170 209	596	4.81	1.559	566	597
500	1 082.35	344.86	244 235	696	6.69	2.20	668	697
600	1 936.68	589.79	315 320			796	66	797
700	3 125.01	930.34	411 935	896	11.43	4.80	868	897
800	4 735.71	1 381.68	500 407	996	14.19	6.03	966	997
900	6 760.44	1 960.24	619 627	1 096	17.27	7.28	1 068	1 097
1 000	19 407.02	2 681.01	725 507	1196	22.88	8.53	1 166	1 197
1 100	—	OOM	—	—	24.17	9 925	1 268	1 297
1 200	—	OOM	—	—	28.67	14 214	1 366	1 397
1 300	—	OOM	—	—	32.78	16 330	1 468	1 497
1 400	—	OOM	—	—	37.58	18 766	1 566	1 597
1 500	—	OOM	—	—	42.69	21 266	1 668	1 697
1 600	—	OOM	—	—	47.83	23 702	1 766	1 797
1 700	—	OOM	—	—	53.34	26 276	1 868	1 897
1 800	—	OOM	—	—	59.32	28 712	1 966	1 997
1 900	—	OOM	—	[illegible]	65.09	31 212	2 068	2 097
11 000	—	OOM	—	—	1 923.53	896 958	11 166	11 197

T : time(ms), M : memory(MB), P : propagations, N : nodes, OOM : Out — Of — Memory

Intel Core2Duo CPU, Q6600 of 2.4Ghz with 3Gb of RAM

Experimental validation

- $\leq_{\text{LEX}}$

Decompisition of $\leq_{\text{LEX}}$ in primitive constraints gives:

$$\boldsymbol{x} \leq_{lex} \boldsymbol{y} \equiv (n = 0) \vee (x_0 < y_0) \vee$$

$$(x_0 = y_0 \wedge < x_1, ..., x_{n-1} > \ \leq_{\mathbf{lex}} \ < y_1, ..., y_{n-1} >)$$

Syntactic negation:

$$\neg(\boldsymbol{x} \leq_{lex} \boldsymbol{y}) \equiv ((n = 1) \wedge (x_0 > y_0)) \vee ((n > 1) \wedge ((x_0 > y_0) \vee$$

$$(< x_1, ..., x_{n-1} > \ \neg\leq_{\mathbf{lex}} \ < y_1, ..., y_{n-1} >)))$$

with a reformulation-based negation:

$$\neg(\boldsymbol{x} \leq_{lex} \boldsymbol{y}) \equiv \boldsymbol{x} >_{lex} \boldsymbol{y}$$

Experimental validation

- $\leq_{\mathrm{LEX}}$

var	syntactic transformations				$>_{lex}$				DFA			
	T	M	P	N	T	M	P	N	T	M	P	N
200	7.00	1.67	1 944	400	6.60	2.03	2 341	400	4.27	0.82	302	400
300	13.24	3.50	2 708	500	12.07	4.69	3 311	500	7.07	2.10	402	500
400	21.45	6.42	3 544	600	19.09	7.64	4 341	600	11.16	3.00	502	600
500	30.86	9.56	4 308	700	27.65	11.87	5 311	700	15.28	4.32	602	700
600	43.32	13.35	5 144	800	38.13	16.03	6341	800	20.30	7.75	702	800
700	56.77	17.38	5 908	900	49.67	20.90	7 311	900	26.28	9.00	802	900
800	71.89	22.00	6 744	1 000	62.72	26.73	8 341	1 000	32.62	11.41	902	1 000
900	90.12	27.12	7 508	1 100	77.12	33.59	9 311	1 100	39.70	15.03	1 002	1 100
10^3	107.97	33.02	8 344	1 200	92.54	40.50	10 341	1200	47.38	16.57	1 102	1 200
2.10^3	402.09	123.06	16 344	2 200	334.72	153.91	20 341	2 200	161.96	65.46	2 102	2 200
3.10^3	889.68	270.51	24 344	3 200	731.38	352.69	30 341	3 200	344.10	147.19	3 102	3 200
4.10^3	1 591.25	475.89	32 344	4 200	1 300.39	625.27	40 341	4 200	597.54	255.35	4 102	4 200
5.10^3	2 527.08	738.53	40 344	5 200	2 059.30	970.92	50 341	5 200	915.20	395.56	5 102	5 200
6.10^3	3 758.49	1 059.63	48 344	6 200	3 010.67	1 388.53	60 341	6 200	1 310.84	567.67	6 102	6 200
7.10^3	5 194.56	1 440.67	56 344	7 200	4 115.96	1 879.97	70 341	7 200	1 772.84	770.78	7 102	7 200
8.10^3	6 951.67	1 880.27	64 344	8 200	5 681.13	2 446.10	80 341	8 200	2 309.79	1 004.37	8 102	8 200

T : time(ms), M : memory(MB), P : propagations, N : nodes, OOM : Out — Of — Memory

Intel Core2Duo CPU, Q6600 of 2.4Ghz with 3Gb of RAM

Experimental validation

- $\preceq_{\text{LEX}}$

var	syntactic transformations				$>_{lex}$				DFA			
	T	M	P	N	T	M	P	N	T	M	P	N
200	7.00	1.67	1 944	400	6.60	2.03	2 341	400	4.27	0.82	302	400
300	13.24	3.50	2 708	500	12.07	4.69	3 311	500	7.07	2.10	402	500
400	21.45	6.42	3 544	600	19.09	7.64	4 341	600	11.16	3.00	502	600
500	30.86	9.56	4 308	700	27.65	11.87	5 311	700	15.28	4.32	602	700
600	43.32	13.35	5 144	800	38.13	16.03	6341	800	20.30	7.75	702	800
700	56.77	17.38	5 908	900	49.67	20.90	7 311	900	26.28	9.00	802	900
800	71.89	22.00	6 744	1 000	62.72	26.73	8 341	1 000	32.62	11.41	902	1 000
900	90.12	27.12	7 508	1 100	77.12	33.59	9 311	1 100	39.70	15.03	1 002	1 100
10^3	107.97	33.02	8 344	1 200	92.54	40.50	10 341	1200	47.38	16.57	1 102	1 200
2.10^3	402.09	123.06	16 344	2 200	334.72	153.91	20 341	2 200	161.96	65.46	2 102	2 200
3.10^3	889.68	270.51	24 344	3 200	731.38	352.69	30 341	3 200	344.10	147.19	3 102	3 200
4.10^3	1 591.25	475.89	32 344	4 200	1 300.39	625.27	40 341	4 200	597.54	255.35	4 102	4 200
5.10^3	2 527.08	738.53	40 344	5 200	2 059.30	970.92	50 341	5 200	915.20	395.56	5 102	5 200
6.10^3	3 758.49	1 059.63	48 344	6 200	3 010.67	1 388.53	60 341	6 200	1 310.84	567.67	6 102	6 200
7.10^3	5 104.56	1 440.67	56 344	7 200	4 115.06	1 870.97	70 341	7 200	1 772.84	770.78	7 102	7 200
[illegible]	6 951.67	1 880.27	[illegible]	[illegible]	5 681.13	2 446.10	[illegible]	[illegible]	2 309.79	1 004.37	[illegible]	8 200

T : time(ms), M : memory(MB), P : propagations, N : nodes, OOM : Out − Of − Memory

Intel Core2Duo CPU, Q6600 of 2.4Ghz with 3Gb of RAM

Conclusions and perspectives

- A preliminary approach to an automatic negation on DFA-based constraints
  - DFA operators (complete, swap-state, clean-up)
  - A Negation for free by exploiting REGULAR consistency algorithm

Conclusions and perspectives

- A preliminary approach to an automatic negation on DFA-based constraints
  - DFA operators (complete, swap-state, clean-up)
  - A Negation for free by exploiting REGULAR consistency algorithm

- Extend the approach on other generic representation (MDD, GRAMMAR,…)

Conclusions and perspectives

- A preliminary approach to an automatic negation on DFA-based constraints
  - DFA operators (complete, swap-state, clean-up)
  - A Negation for free by exploiting REGULAR consistency algorithm

- Extend the approach on other generic representation (MDD, GRAMMAR,…)
- Generic global constraint combinaisons (i.e., conjunction, disjunction)

Conclusions and perspectives

- A preliminary approach to an automatic negation on DFA-based constraints
    - DFA operators (complete, swap-state, clean-up)
    - A Negation for free by exploiting REGULAR consistency algorithm

- Extend the approach on other generic representation (MDD, GRAMMAR,…)
- Generic global constraint combinaisons (i.e., conjunction, disjunction)

Thank you