Synthesizing Concurrent Programs using Answer Set Programming

E. De Angelis[1], A. Pettorossi[2], M. Proietti[3]

[1]University of Chieti-Pescara 'G. d'Annunzio'
[2]University of Rome 'Tor Vergata'
[3]CNR-IASI, Rome

CS&P 2011, Pułtusk, Poland
28th - 30th September 2011

Related work

Automated Synthesis of Concurrent Programs

- E. M. Clarke and E. A. Emerson
  Design and Synthesis of Synchronization Skeletons Using Branching Temporal Logic
  Workshop on Logic of Program, Springer-Verlag, 1982
- Z. Manna and P. Wolper
  Synthesis of Communicating Process from Temporal Specifications
  ACM TOPLAS, 1984
- ...

Answer Set Programming

- S. Heymans, D. Van Nieuwenborgh and D. Vermeir
  Synthesis from Temporal Specifications using Preferred Answer Set Programming
  LNCS no. 3701, 2005
- ...

Overview

- Concurrent Programs
  - Syntax
  - Semantics
- Specification of Concurrent Programs
  - Behavioural properties
  - Structural properties
  - Symmetric Concurrent Programs
- Automated Synthesis of Concurrent Program

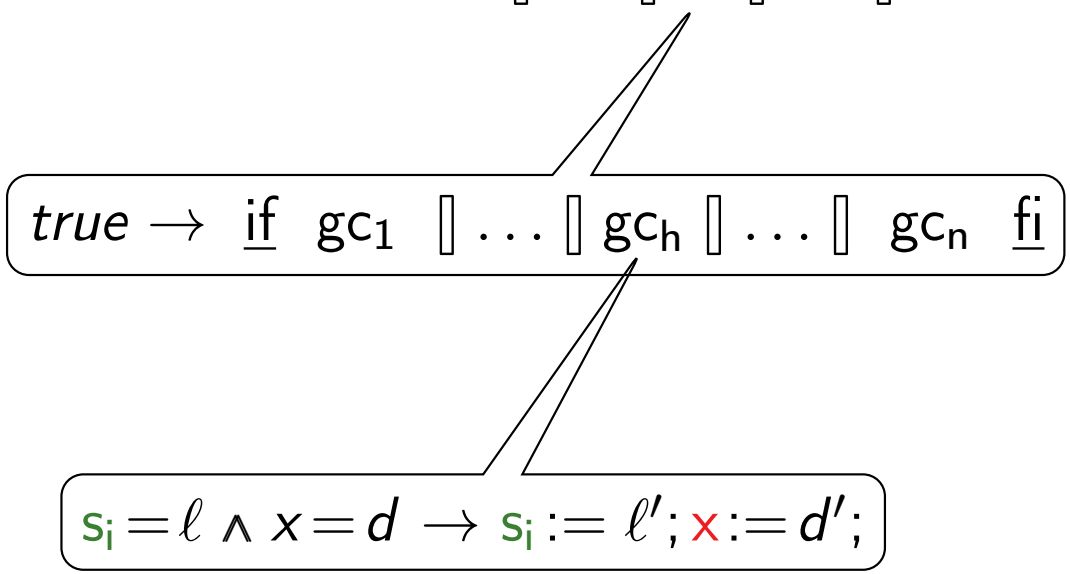
  based on Answer Set Programming
- Experimental Results

k-Process Concurrent Program

Syntax

* $P_1, \ldots, P_k$, *processes*
* $s_1, \ldots, s_k$, *local variables* ranging over a finite domain L
* x, a single *shared variable* ranging over a finite domain D

$s_1 := \ell_1; \ldots; s_k := \ell_k;\ x := d_0;\ \underline{\text{do}}\ P_1 \,[]\, \ldots \,[]\, P_i \,[]\, \ldots \,[]\, P_k\ \underline{\text{od}}$



where $\ell_1, \ldots, \ell_k, \ell, \ell' \in L$, and $d_0, d, d' \in D$.

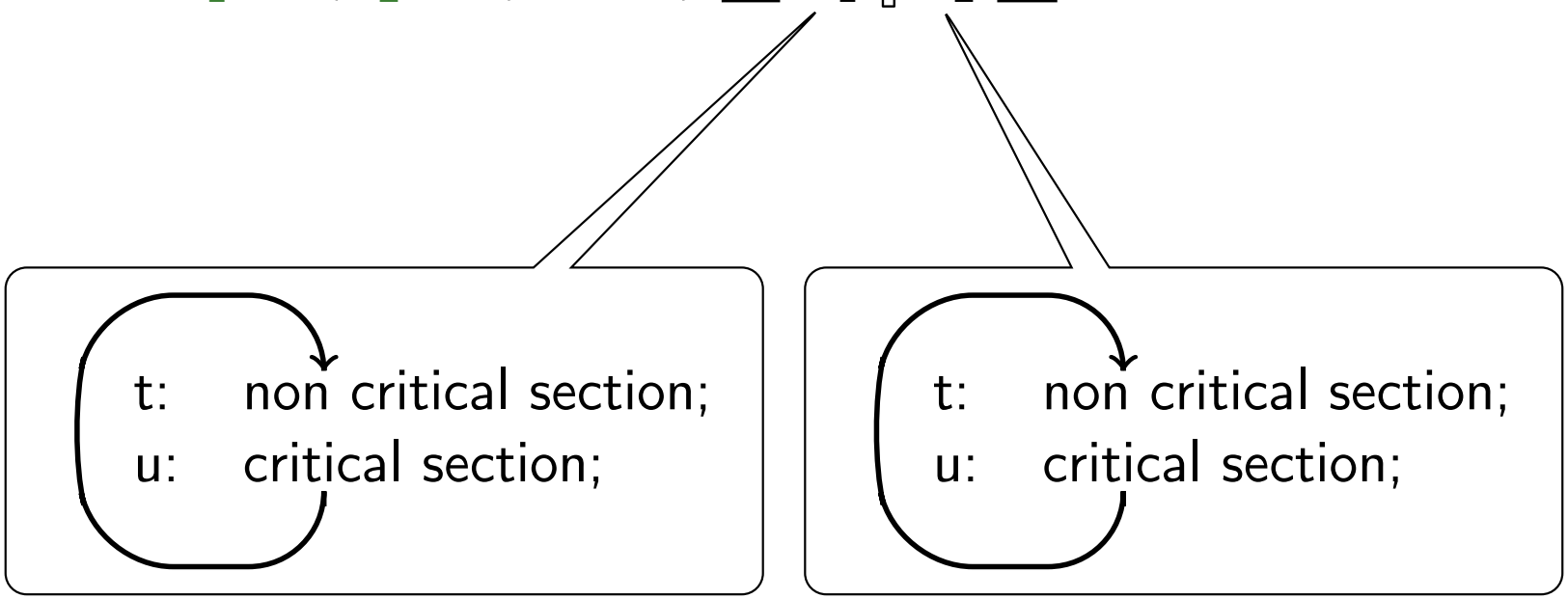
Example

Syntax

A 2-Process Concurrent Program C

- P_1, P_2, processes
- s_1, s_2, local variables ranging over $L = \{t, u\}$
- x, a single shared variable ranging over $D = \{0, 1\}$

C : $s_1 := t; s_2 := t;\ x := 0;\ \underline{\text{do}}\ P_1 \,[\!]\, P_2\ \underline{\text{od}}$



Example

Syntax

A 2-Process Concurrent Program C

- P_1, P_2, processes
- s_1, s_2, local variables ranging over $L = \{t, u\}$
- x, a single shared variable ranging over $D = \{0, 1\}$

$$C: \quad s_1 := t; s_2 := t; \; x := 0; \; \underline{\text{do}} \; P_1 \; [] \; P_2 \; \underline{\text{od}}$$

$$P_1: \; true \rightarrow \underline{\text{if}}$$
$$s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0;$$
$$[] \; s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1;$$
$$\underline{\text{fi}}$$

$$P_2: \; true \rightarrow \underline{\text{if}}$$
$$s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1;$$
$$[] \; s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0;$$
$$\underline{\text{fi}}$$

k-Process Concurrent Program

Semantics

A k-Process Concurrent Program C

* $P_1, \ldots, P_k$, processes
* $s_1, \ldots, s_k$, local variables ranging over a finite domain L
* x, a single *shared variable* ranging over a finite domain D

$$s_1 := \ell_1; \ldots; s_k := \ell_k;\ x := d_0;\ \underline{\text{do}}\ P_1 \,[]\, \ldots \,[]\, P_i \,[]\, \ldots \,[]\, P_k\ \underline{\text{od}}$$

The Kripke structure $\mathcal{K} = \langle S, S_0, R, \lambda \rangle$ associated with C:

* set of *states*: $S = L^k \times D$
* set of *initial states*: $S_0 = \{\langle \ell_1, \ldots, \ell_k, d_0 \rangle\}$
* total *transition relation*: $R \subseteq S \times S$
* total *labelling function*: $\lambda : S \to \mathcal{P}(Elem)$

Example

$$
\begin{array}{l}
s_1 := t;\ s_2 := t;\ x := 0; \\
\underline{\text{do}} \quad \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0; \\ \quad [\!] \ s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1; \\ \quad \underline{\text{fi}} \end{array} \\
\quad \| \\
\quad \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1; \\ \quad [\!] \ s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0; \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}
\end{array}
$$

Example (Cont'd)

$s_1 := t;\ s_2 := t;\ x := 0;$

$$\underline{\text{do}}\quad \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0; \\ [\!] \ s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1; \\ \quad \underline{\text{fi}} \end{array} \Bigg\| \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1; \\ [\!] \ s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0; \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}$$

$S_0 = \{\langle t, t, 0\rangle\}$

$\langle t, t, 0\rangle$

$S = \{\langle t, t, 0\rangle, \ldots\}$

$\lambda = \{\langle\langle t, t, 0\rangle, \{s_1 = t, s_2 = t, x = 0\}\rangle, \ldots\}$

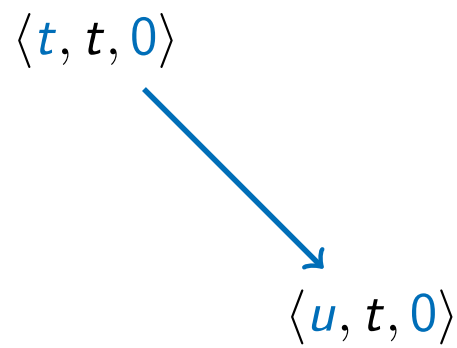
Example (Cont'd)

$$
\begin{array}{l}
s_1 := t;\ s_2 := t;\ x := 0; \\
\underline{\text{do}} \quad \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0; \\ [\!] \ s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1; \\ \quad \underline{\text{fi}} \end{array} \\
\quad \Big\| \quad \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1; \\ [\!] \ s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0; \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}
\end{array}
$$

$S_0 = \{\langle t, t, 0\rangle\}$

$S = \{\langle t, t, 0\rangle, \langle u, t, 0\rangle, \ldots\}$

$R = \{\langle\langle t, t, 0\rangle, \langle u, t, 0\rangle\rangle, \ldots$



$\lambda = \{\langle\langle t, t, 0\rangle, \{s_1 = t, s_2 = t, x = 0\}\rangle, \ldots\}$

Example (Cont'd)

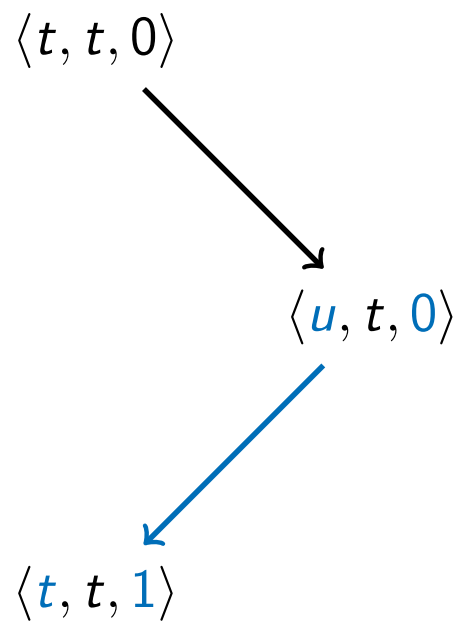
$$
\begin{array}{l}
s_1 := t;\ s_2 := t;\ x := 0; \\
\underline{\text{do}} \quad \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0; \\ \quad [\!] \ s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1; \\ \quad \underline{\text{fi}} \end{array} \\
\qquad \| \\
\begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1; \\ \quad [\!] \ s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0; \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}
\end{array}
$$

$S_0 = \{\langle t, t, 0\rangle\}$

$S = \{\langle t, t, 0\rangle, \langle u, t, 0\rangle, \langle t, t, 1\rangle, \ldots\}$

$R = \{\langle\langle t, t, 0\rangle, \langle u, t, 0\rangle\rangle, \langle\langle u, t, 0\rangle, \langle t, t, 1\rangle\rangle, \ldots$

$\lambda = \{\langle\langle t, t, 0\rangle, \{s_1 = t, s_2 = t, x = 0\}\rangle, \ldots\}$



Example (Cont'd)

$s_1 := t;\ s_2 := t;\ x := 0;$

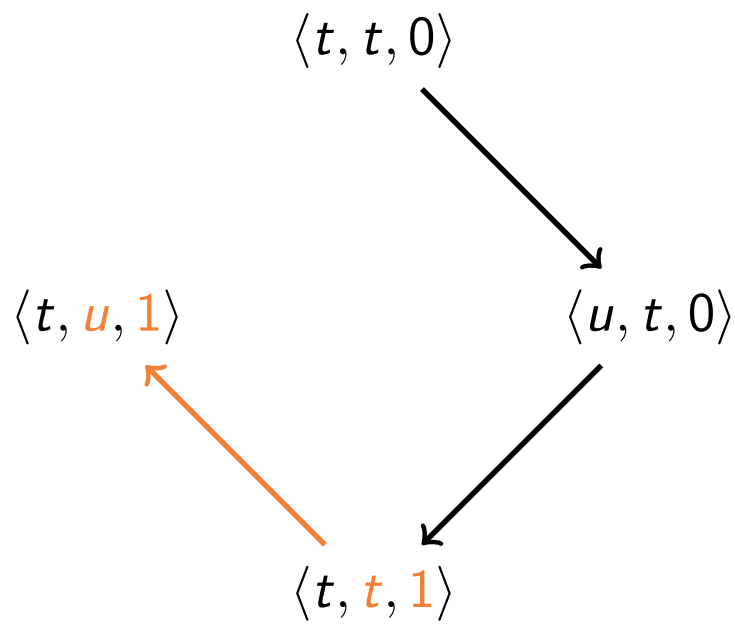
$$\underline{\text{do}}\quad \begin{array}{l} \mathit{true} \rightarrow \underline{\text{if}} \\ \quad s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0; \\ \quad [\!] \ s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1; \\ \quad \underline{\text{fi}} \end{array} \quad \| \quad \begin{array}{l} \mathit{true} \rightarrow \underline{\text{if}} \\ \quad s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1; \\ \quad [\!] \ s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0; \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}$$

$S_0 = \{\langle t, t, 0\rangle\}$

$S = \{\langle t, t, 0\rangle, \langle u, t, 0\rangle,$
$\langle t, t, 1\rangle, \langle t, u, 1\rangle, \ldots\}$

$R = \{\langle\langle t, t, 0\rangle, \langle u, t, 0\rangle\rangle,$
$\langle\langle u, t, 0\rangle, \langle t, t, 1\rangle\rangle,$
$\langle\langle t, t, 1\rangle, \langle t, u, 1\rangle\rangle, \ldots$

$\lambda = \{\langle\langle t, t, 0\rangle, \{s_1 = t, s_2 = t, x = 0\}\rangle, \ldots\}$



Example (Cont'd)

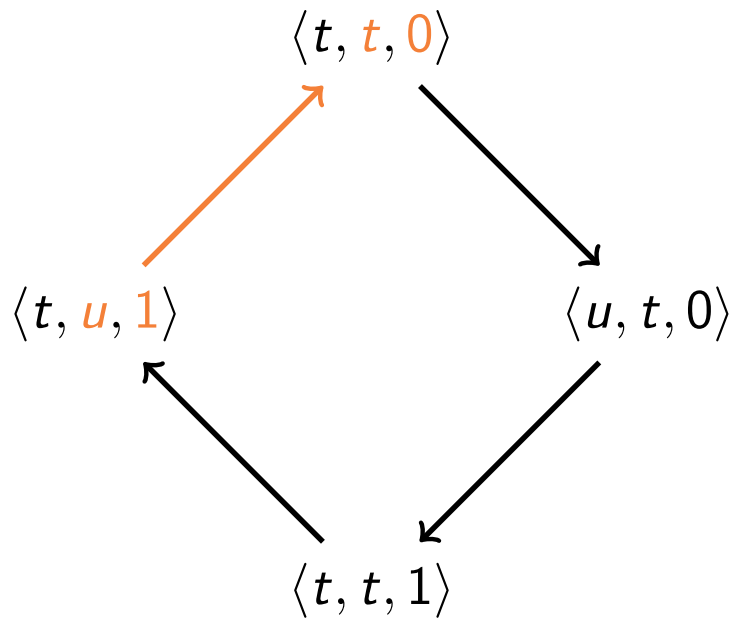
$$
\begin{array}{l}
s_1 := t;\ s_2 := t;\ x := 0; \\
\quad \underline{\text{do}} \quad \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0; \\ \quad [\!] \ s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1; \\ \quad \underline{\text{fi}} \end{array} \\
\qquad \Big\| \quad \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1; \\ \quad [\!] \ s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0; \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}
\end{array}
$$

$$S_0 = \{\langle t, t, 0\rangle\}$$

$$S = \{\langle t, t, 0\rangle, \langle u, t, 0\rangle, \langle t, t, 1\rangle, \langle t, u, 1\rangle, \ldots\}$$

$$R = \{\langle\langle t, t, 0\rangle, \langle u, t, 0\rangle\rangle, \langle\langle u, t, 0\rangle, \langle t, t, 1\rangle\rangle, \langle\langle t, t, 1\rangle, \langle t, u, 1\rangle\rangle, \langle\langle t, u, 1\rangle, \langle t, t, 0\rangle\rangle\}$$

$$\lambda = \{\langle\langle t, t, 0\rangle, \{s_1 = t, s_2 = t, x = 0\}\rangle, \ldots\}$$



Specification: Behavioural Properties

Time dependant behavioural properties of Concurrent Programs:

- safety
- liveness

Specified in a Temporal Logic, i.e., Computation Tree Logic (CTL):

- path quantifiers: for all paths **A**, for some paths **E**
- temporal operators: eventually **F**, globally **G**, next **X**,....

> A k-Process Concurrent Program C satisfies a behavioural property φ iff
> the Kripke structure $\mathcal{K}$ associated with C satisfies φ.

Specification: Behavioural Properties

Computation Tree Logic

- Syntax: $\varphi ::= b \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{EX}\,\varphi \mid \mathsf{EG}\,\varphi \mid \mathsf{E}[\varphi_1\,\mathsf{U}\,\varphi_2]$

  $\mathsf{AG}\varphi \equiv \neg\mathsf{EF}\neg\,\varphi$, etc.

- Semantics:

$$\left.\begin{array}{r}\text{Kripke structure } \mathcal{K}\\ \text{state } s\\ \text{formula } \varphi\end{array}\right\} \quad \mathcal{K}, s \models \varphi$$

recursively defined as follows:

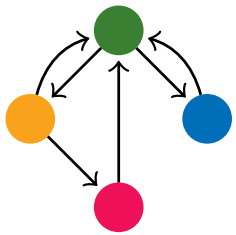
$\mathcal{K}, s \models b$	iff	$b \in \lambda(s)$
$\mathcal{K}, s \models \neg\,\varphi$	iff	$\mathcal{K}, s \models \varphi$ does not hold
$\mathcal{K}, s \models \varphi_1 \wedge \varphi_2$	iff	$\mathcal{K}, s \models \varphi_1$ and $\mathcal{K}, s \models \varphi_2$
$\mathcal{K}, s \models \mathsf{EX}\,\varphi$	iff	there exists $\langle s, t\rangle \in R$ such that $\mathcal{K}, t \models \varphi$
$\mathcal{K}, s \models \mathsf{EG}\,\varphi$	iff	there exists a path π such that $\pi_0 = s$ and for all $i \geq 0$, $\mathcal{K}, \pi_i \models \varphi$
$\mathcal{K}, s \models \mathsf{E}\,[\varphi_1\,\mathsf{U}\,\varphi_2]$	iff	there exists a path $\pi = \langle s, s_1, \ldots\rangle$ in $\mathcal{K}$ and $i \geq 0$ such that $\mathcal{K}, \pi_i \models \varphi_2$ and for all $0 \leq j < i$, $\mathcal{K}, \pi_j \models \varphi_1$

Specification: Behavioural Properties

Computation Tree Logic

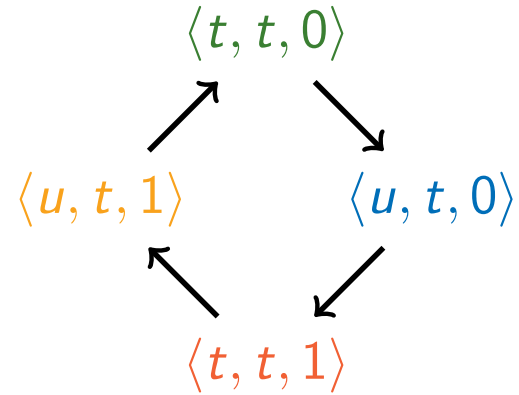
k-Process Concurrent Program satisfying p

M :



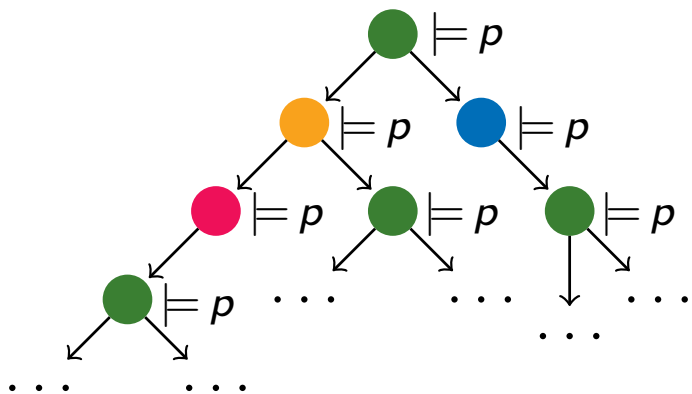
2-Process Concurrent Program satisfying mutual exclusion

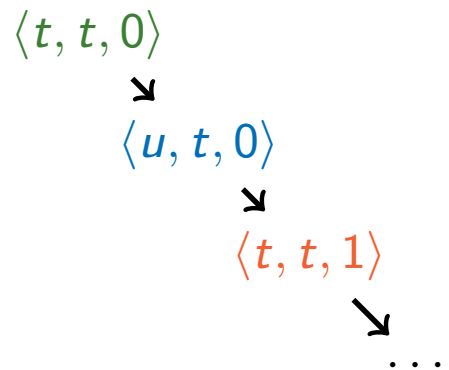
K :



$M, \bullet \models \mathsf{AG}\, p$

$K, \langle t, t, 0\rangle \models \mathsf{AG}\, \neg(\mathsf{s}_1 = u \wedge \mathsf{s}_2 = u)$





Specification: Structural Properties

Symmetric Program Structure

- local transition relation $T \subseteq L \times L$

  pattern of execution of each process P_i

- k-generating function $f: D \to D$ (permutation on the domain of x)

  global property of a concurrent program C

T and f define a Symmetric Program Structure $\Sigma = \langle f, T \rangle$

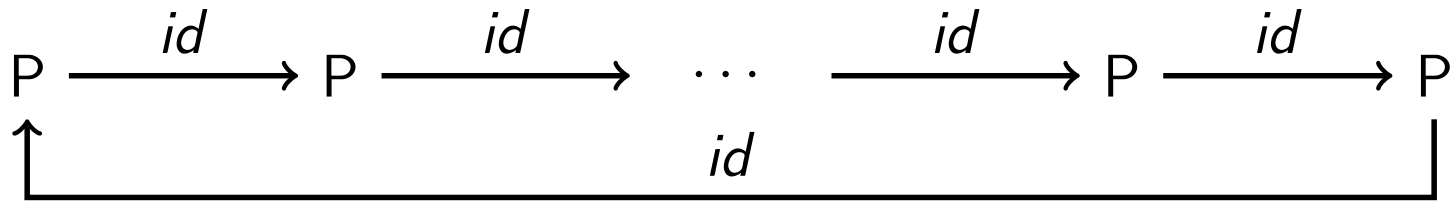
A k-Process Concurrent Program C satisfies Σ *iff*

1. each process $P_1, \ldots, P_k$ in C satisfies T, and
2. C satisfies f

Symmetric Concurrent Programs

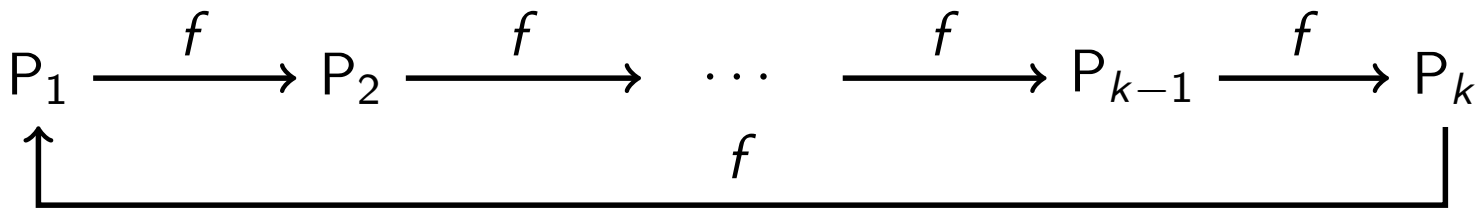
k-generating function f is

either the identity function id



(Dijkstra's semaphore)

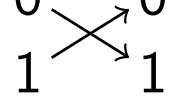
or a generator of a cyclic group $\{id, f, \ldots, f^{k-1}\}$ of order k



(Peterson's algorithm)

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric program structure Σ: T :  f: 

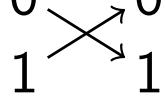
$$true \rightarrow \underline{\text{if}} \;\; s_1 = t \wedge x = 0 \rightarrow s_1 := u \; ; \; x := 0 \; ;$$

$$[] \;\; s_1 = t \wedge x = 1 \rightarrow s_1 := t \; ; \; x := 1 \; ;$$

$$[] \;\; s_1 = u \wedge x = 0 \rightarrow s_1 := t \; ; \; x := 1 \; ; \; \underline{\text{fi}}$$

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric program structure Σ: T :  f: 

$$true \rightarrow \underline{\text{if}} \;\; s_1 = t \wedge x = 0 \rightarrow s_1 := u \;;\; x := 0 \;;$$

$$true \rightarrow \underline{\text{if}} \;\; s_2 = t \wedge x = 1 \rightarrow s_2 := u \;;\; x := 1 \;;$$

$$[] \;\; s_1 = t \wedge x = 1 \rightarrow s_1 := t \;;\; x := 1 \;;$$

$$[] \;\; s_1 = u \wedge x = 0 \rightarrow s_1 := t \;;\; x := 1 \;;\; \underline{\text{fi}}$$

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric program structure Σ: $T : t \rightleftarrows u$ $\circlearrowright$ $\quad f: \begin{matrix} 0 & 0 \\ 1 & 1 \end{matrix}$ (0 ↔ 0, 1 ↔ 1)

$$true \rightarrow \underline{\text{if}} \;\; s_1 = t \wedge x = 0 \rightarrow s_1 := u \;;\; x := 0 \;;$$

$$true \rightarrow \underline{\text{if}} \;\; s_2 = t \wedge x = 1 \rightarrow s_2 := u \;;\; x := 1 \;;$$

$$[] \;\; s_1 = t \wedge x = 1 \rightarrow s_1 := t \;;\; x := 1 \;;$$

$$[] \;\; s_2 = t \wedge x = 0 \rightarrow s_2 := t \;;\; x := 0 \;;$$

$$[] \;\; s_1 = u \wedge x = 0 \rightarrow s_1 := t \;;\; x := 1 \;;\; \underline{\text{fi}}$$

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric program structure Σ: $T : t \rightleftarrows u$ $\circlearrowleft$ $\quad f: \begin{matrix} 0 & 0 \\ 1 & 1 \end{matrix}$ (crossed)

$$true \rightarrow \underline{\text{if}} \;\; s_1 = t \wedge x = 0 \rightarrow s_1 := u \;;\; x := 0 \;;$$

$$true \rightarrow \underline{\text{if}} \;\; s_2 = t \wedge x = 1 \rightarrow s_2 := u \;;\; x := 1 \;;$$

$$[] \;\; s_1 = t \wedge x = 1 \rightarrow s_1 := t \;;\; x := 1 \;;$$

$$[] \;\; s_2 = t \wedge x = 0 \rightarrow s_2 := t \;;\; x := 0 \;;$$

$$[] \;\; s_1 = u \wedge x = 0 \rightarrow s_1 := t \;;\; x := 1 \;;\; \underline{\text{fi}}$$

$$[] \;\; s_2 = u \wedge x = 1 \rightarrow s_2 := t \;;\; x := 0 \;;\; \underline{\text{fi}}$$

Synthesis of a Concurrent Programs

- automatically derive a k-concurrent program from a given specification:
  - Behavioural Properties
  - Structural Properties
- derive a program correct by construction

$$s_1 := \ell_1;\ s_2 := \ell_2;\ x := d_0;$$

$$\underline{\text{do}} \quad \begin{array}{l} \mathit{true} \rightarrow \underline{\text{if}} \\ \quad s_1 = ? \wedge x = ? \rightarrow s_1 := ?; x := ?; \\ [] \quad \ldots \\ [] \; s_1 = ? \wedge x = ? \rightarrow s_1 := ?; x := ?; \\ \quad \underline{\text{fi}} \end{array} \;\Big\|\; \begin{array}{l} \mathit{true} \rightarrow \underline{\text{if}} \\ \quad s_2 = ? \wedge x = ? \rightarrow s_2 := ?; x := ?; \\ [] \quad \ldots \\ [] \; s_2 = ? \wedge x = ? \rightarrow s_2 := ?; x := ?; \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}$$

Synthesis Problem

Given:

1. a CTL formula φ (Behavioural Property),
2. a Symmetric Program Structure $\Sigma = \langle f, T \rangle$ (Structural Property)

we look for a k-process concurrent program C such that

* for all $i > 0$ $f(P_i) = P_{(i \bmod k)+1}$, and
* C satisfies φ

> We guess P_1 satisfying T and we generate the set $P_2, \ldots, P_k$ using f such that the Kripke structure associated with C satisfies φ

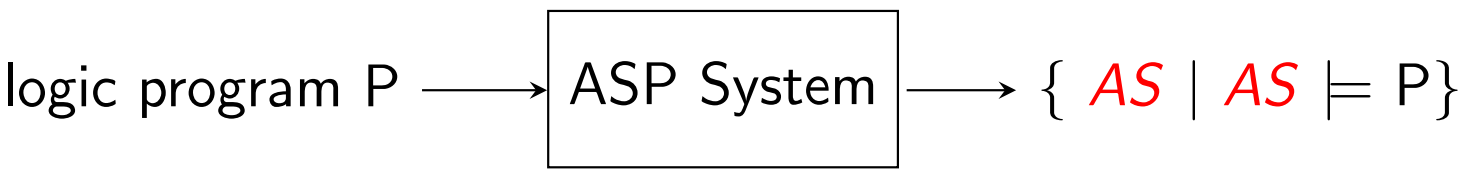
Answer Set Programming (ASP)

- knowledge representation and reasoning tool

  logic program $\Rightarrow$ encoding of a problem

  Answer Set $\Rightarrow$ solution of a problem

- guess & check
  - guess: rules for generate candidate solutions
  - check: rules for eliminate invalid candidates



ASP-based Synthesis Procedure

We reduce the derivation of a k-Process Concurrent Program to an answer set computation

$$\boxed{\varphi + \Sigma}$$

ASP-based Synthesis Procedure

We reduce the derivation of a k-Process Concurrent Program to an answer set computation

Encoding: a logic program Π encodes a *synthesis problem*.

Π is the union of:

- Π_φ, encoding Behavioural Properties φ
- Π_Σ, encoding Structural Properties Σ

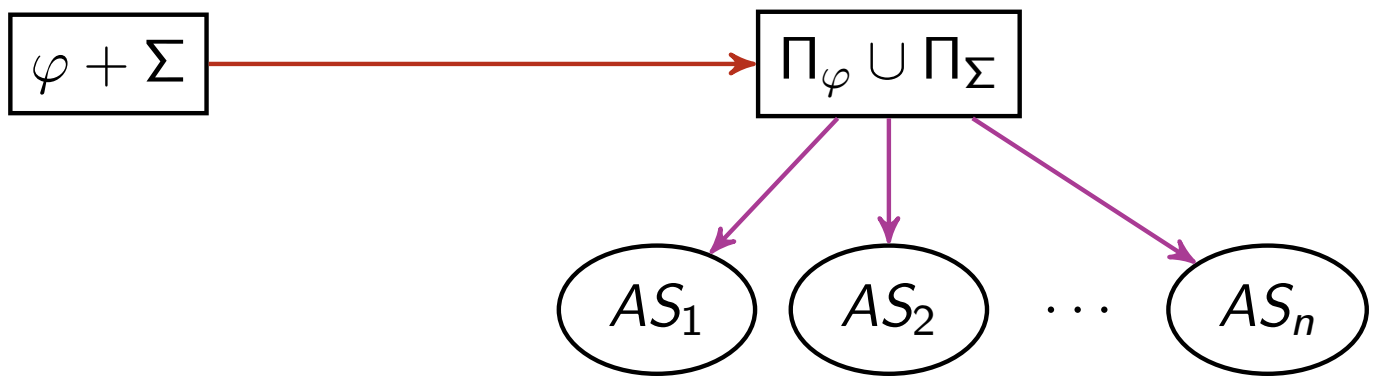
ASP-based Synthesis Procedure

We reduce the derivation of a k-Process Concurrent Program to an answer set computation

Encoding: a logic program Π encodes a *synthesis problem*.

Π is the union of:

- Π_φ, encoding Behavioural Properties φ
- Π_Σ, encoding Structural Properties Σ



ASP-based Synthesis Procedure

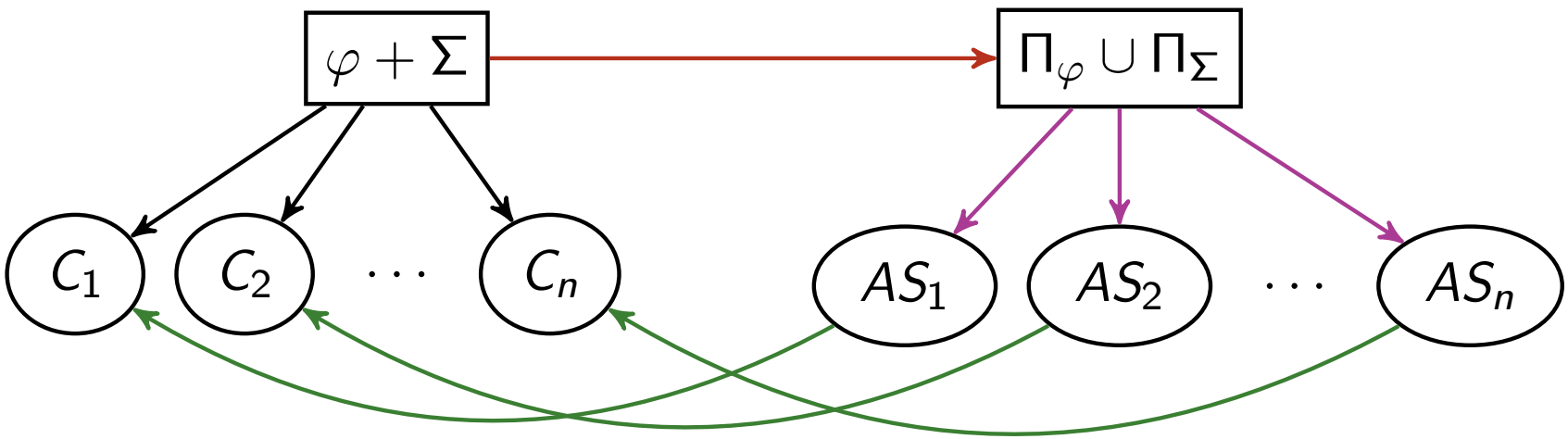
We reduce the derivation of a k-Process Concurrent Program to an answer set computation

Encoding: a logic program Π encodes a *synthesis problem*.

Π is the union of:

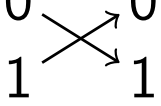
- Π_φ, encoding Behavioural Properties φ
- Π_Σ, encoding Structural Properties Σ

Decoding: answer sets $ans(\Pi) = \{AS_1, AS_2, ..., AS_n\}$ encode *solutions* of the synthesis problem



Example

Encoding

Symmetric program structure Σ: $T : t \rightleftarrows u$ f: 

Behavioural property: $\varphi = \mathsf{AG}\,\neg(\mathsf{s}_1 = u \wedge \mathsf{s}_2 = u)$

$\Pi = \{$

$\leftarrow not\ sat(s(t,t,0), n(ef(a(ap(s1,u), ap(s2,u))))).$

$sat(s(t,t,0), n(ef(a(ap(s1,u), ap(s2,u))))) \leftarrow$
$not\ sat(s(t,t,0), ef(a(ap(s1,u), ap(s2,u)))).$

$\ldots$

$sat(s(t,t,0), a(ap(s1,u), ap(s2,u))) \leftarrow$
$sat(s(t,t,0), ap(s1,u)), sat(s(t,t,0), ap(s2,u)).$

$\ldots$

$gc(1,X,0,Y,0) \vee gc(1,X,0,Y,1) \vee \ldots \leftarrow reachable(X,\_,0).$

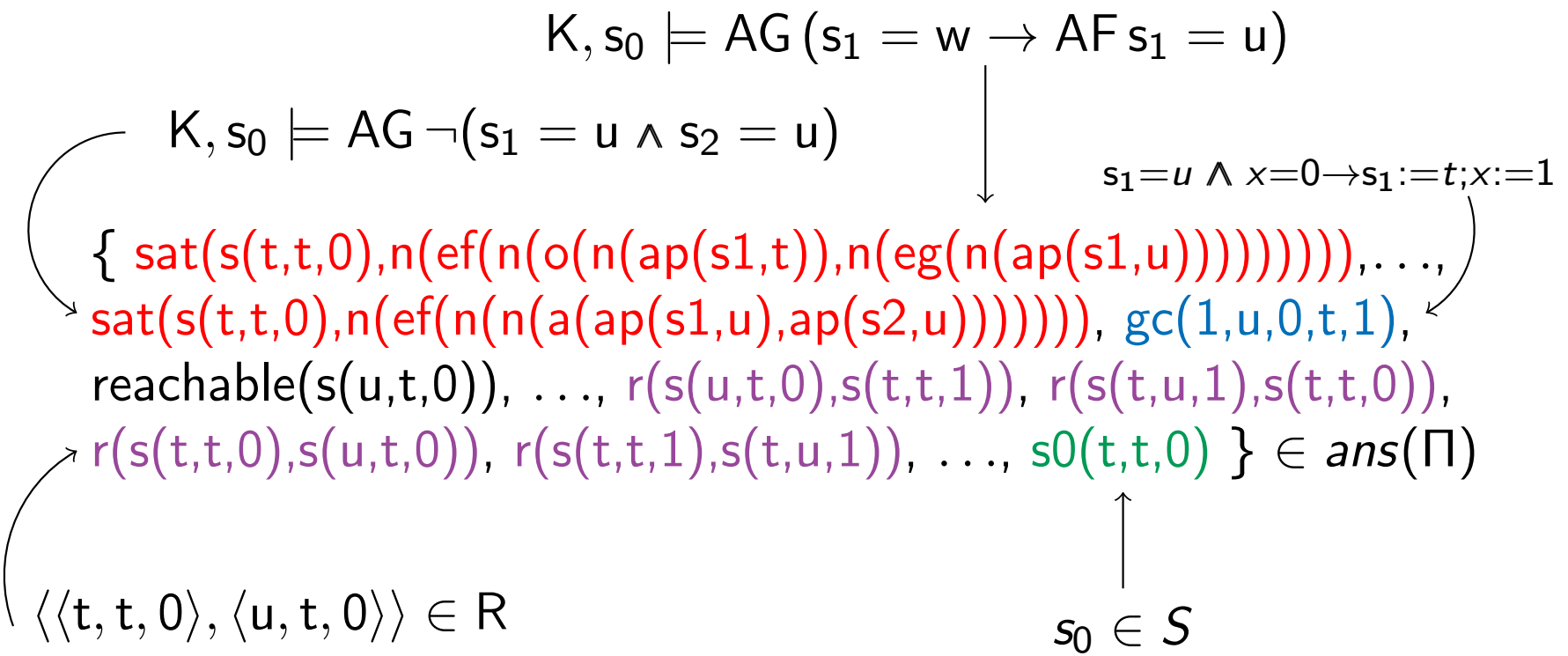
$\ldots$

$gc(2,u,f(1),t,f(0)) \leftarrow gc(1,u,0,t,1).$

$\}$

Example

Decoding



Correctness of Synthesis Procedure

Theorem (Correctness of Synthesis)

Let $\Pi = \Pi_\varphi \cup \Pi_\Sigma$. *be the logic program obtained from:*

1. *a CTL formula* φ, *and*
2. *a symmetric program structure* $\Sigma = \langle f, T \rangle$.

Then,

$$\left(\mathsf{s}_1 := \ell_0; \ldots; \mathsf{s}_k := \ell_0;\ x := d_0;\ \underline{\text{do}}\ P_1 \,[\!]\, \ldots \,[\!]\, P_k\ \underline{\text{od}}\right)\ \models\ \varphi$$

iff there exists an answer set AS *in* $ans(\Pi)$ *such that*

$\forall i \in \{1, \ldots, k\}$, $\forall \ell, \ell' \in L$, $\forall d, d' \in D$,

$\left(\mathsf{s}_i = \ell \wedge x = d \rightarrow \mathsf{s}_i := \ell'; x := d'\right)$ *is in* P_i *iff* $AS \models \mathsf{gc}(i, \ell, d, \ell', d')$

Experimental results

Examples

ME *Mutual Exclusion*: no two processes are in *use*
for all i, j in $\{1, \ldots, k\}$, with $i \neq j$,

$$\mathsf{AG}\,\neg(\mathsf{s}_i = u \wedge \mathsf{s}_j = u)$$

SF *Starvation Freedom*: if a process is waiting then it will enter in use
for all i in $\{1, \ldots, k\}$,

$$\mathsf{AG}\,(\mathsf{s}_i = w \rightarrow \mathsf{AF}\,\mathsf{s}_i = u)$$

BO *Bounded Overtaking*: while a process is waiting, any other process can exits from its critical section at most once
for all i, j in $\{1, \ldots, k\}$,

$$\mathsf{AG}\,((\mathsf{s}_i = w \wedge \mathsf{s}_j = u) \rightarrow \mathsf{AF}\,(\mathsf{s}_j = t \wedge \mathsf{A}[\neg(\mathsf{s}_j = u)\,\mathsf{U}\,\mathsf{s}_i = u]))$$

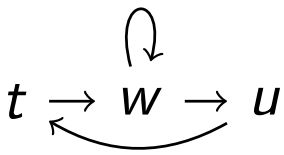
MR *Maximal Reactivity* (MR): if a process is waiting and all others are thinking then in the next state it will enter in use
for all i in $\{1, \ldots, k\}$,

$$\mathsf{AG}\,((\mathsf{s}_i = w \wedge \bigwedge_{j \in \{1,\ldots,k\} \setminus \{i\}} \mathsf{s}_j = t) \rightarrow \mathsf{EX}\,\mathsf{s}_i = u)$$

Experimental results

Synthesis of k-process concurrent programs

T :



id: $0 \longrightarrow 0$, $1 \longrightarrow 1$

f_1: $0 \longrightarrow 1$, $1 \longrightarrow 0$

f_2: $0 \longrightarrow 1$, $1 \longrightarrow 0$, $2 \longrightarrow 2$

f_3: $0 \longrightarrow 1$, $1 \longrightarrow 2$, $2 \longrightarrow 0$

Program	Satisfied Properties	$\lvert D\rvert$	f	$\lvert ans(\Pi)\rvert$	Time (sec)
mutex for 2 processes	*ME*	2	*id*	6	0.07
	ME	2	f_1	7	0.70
	ME, *SF*	2	f_1	3	0.71
	ME, *SF*, *BO*	2	f_1	3	1.44
	ME, *SF*, *BO*, *MR*	3	f_2	2	11.70
mutex for 3 processes	*ME*	2	*id*	5	0.95
	ME	2	f_1	10	0.87
	ME, *SF*	3	f_3	8	152.00
	ME, *SF*, *BO*	3	f_3	8	1700.00

Conclusions

- reduction of the design of a concurrent program to the design of its formal specification
- fully declarative solution (independent of the ASP solver)
- state space explosion problem
  - exploit CTL formulas symmetries,
  - exploit Kripke structures symmetries,
  - ...