Synthesizing Concurrent Programs using Answer Set Programming

Emanuele De Angelis[1,3]
deangelis@sci.unich.it
www.sci.unich.it/∼deangelis

joint work with:
Alberto Pettorossi[2,3] and Maurizio Proietti[3]

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

LIAFA, Paris, France
13th February 2012

Concurrent Programs Synthesis

Concurrent programs

finite set of processes

which interact by using *communication protocols*

to guarantee a *desired behaviour* of concurrent programs

Communications protocols may be

- hard to design,
- hard to *prove* correct

Concurrent Programs Synthesis

Concurrent programs

finite set of processes

which interact by using *communication protocols*

to guarantee a *desired behaviour* of concurrent programs

Communications protocols may be

- ~~hard to design~~,
- hard to *prove* correct

Synthesis

by providing

- a formal specification

Concurrent Programs Synthesis

Concurrent programs

finite set of processes

which interact by using *communication protocols*

to guarantee a *desired behaviour* of concurrent programs

Communications protocols may be

- ~~hard to design~~,
- ~~hard to *prove* correct~~

Synthesis

by providing

- a formal specification

we automatically *derive* a concurrent program which is

- correct by construction

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)
- Answer Set Programming
- Synthesis of Concurrent Program

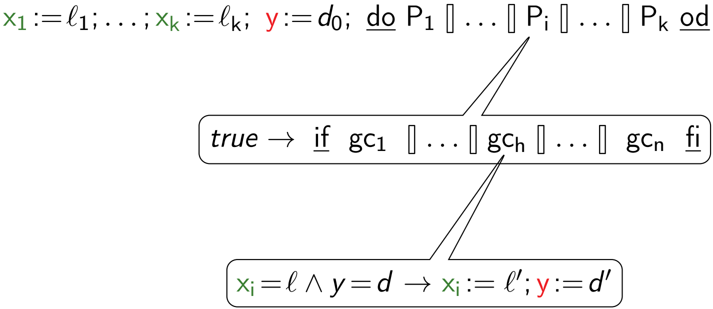
  (based on Answer Set Programming)
- Experimental Results

Concurrent Programs: Syntax

k-Process Concurrent Program

Syntax

- $\mathrm{P}_1, \ldots, \mathrm{P}_\mathrm{k}$, *processes*
- x_1,…,x_k, *local variables* ranging over a finite domain L
- y, a single *shared variable* ranging over a finite domain D



$$\mathrm{x}_1 := \ell_1; \ldots ; \mathrm{x}_\mathrm{k} := \ell_\mathrm{k};\ \mathrm{y} := d_0;\ \underline{\mathrm{do}}\ \mathrm{P}_1 \talloblong \ldots \talloblong \mathrm{P}_\mathrm{i} \talloblong \ldots \talloblong \mathrm{P}_\mathrm{k}\ \underline{\mathrm{od}}$$

$$\boxed{\mathit{true} \rightarrow\ \underline{\mathrm{if}}\ \ \mathrm{gc}_1\ \talloblong \ldots \talloblong \mathrm{gc}_\mathrm{h} \talloblong \ldots \talloblong\ \mathrm{gc}_\mathrm{n}\ \ \underline{\mathrm{fi}}}$$

$$\boxed{\mathrm{x}_\mathrm{i} = \ell \wedge y = d \rightarrow \mathrm{x}_\mathrm{i} := \ell'; \mathrm{y} := d'}$$

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

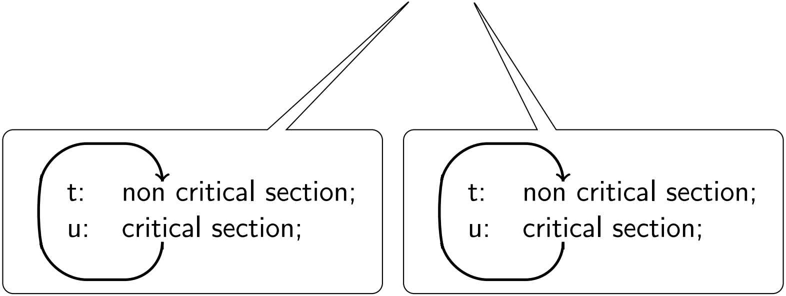
Example

Syntax

A 2-Process Concurrent Program C

- P_1, P_2, processes
- x_1, x_2, local variables ranging over $L = \{t, u\}$
- y, a single shared variable ranging over $D = \{0, 1\}$

$$C : \quad \mathsf{x}_1 := t; \mathsf{x}_2 := t;\ \mathsf{y} := 0;\ \underline{\mathsf{do}}\ \mathsf{P}_1 \ [\!]\ \mathsf{P}_2\ \underline{\mathsf{od}}$$



t: non critical section;
u: critical section;

t: non critical section;
u: critical section;

Example

Syntax

A 2-Process Concurrent Program C

- P_1, P_2, processes
- x_1, x_2, local variables ranging over $L = \{t, u\}$
- y, a single shared variable ranging over $D = \{0, 1\}$

$$C: \quad \mathrm{x}_1 := t; \mathrm{x}_2 := t; \; \mathrm{y} := 0; \; \underline{\mathrm{do}}\ \mathrm{P}_1 \;[\!]\; \mathrm{P}_2\ \underline{\mathrm{od}}$$

$$\begin{aligned} \mathrm{P}_1 :\ & \mathit{true} \rightarrow \underline{\mathrm{if}} \\ & \quad \mathrm{x}_1 = t \wedge \mathrm{y} = 0 \rightarrow \mathrm{x}_1 := u; \mathrm{y} := 0 \\ & \;[\!]\; \mathrm{x}_1 = u \wedge \mathrm{y} = 0 \rightarrow \mathrm{x}_1 := t; \mathrm{y} := 1 \\ & \qquad \underline{\mathrm{fi}} \end{aligned}$$

$$\begin{aligned} \mathrm{P}_2 :\ & \mathit{true} \rightarrow \underline{\mathrm{if}} \\ & \quad \mathrm{x}_2 = t \wedge \mathrm{y} = 1 \rightarrow \mathrm{x}_2 := u; \mathrm{y} := 1 \\ & \;[\!]\; \mathrm{x}_2 = u \wedge \mathrm{y} = 1 \rightarrow \mathrm{x}_2 := t; \mathrm{y} := 0 \\ & \qquad \underline{\mathrm{fi}} \end{aligned}$$

Concurrent Programs: Semantics

k-Process Concurrent Program

Semantics

A k-Process Concurrent Program C

- $\mathsf{P}_1, \ldots, \mathsf{P}_\mathsf{k}$, processes
- $\mathsf{x}_1,\ldots,\mathsf{x}_\mathsf{k}$, local variables ranging over a finite domain L
- y, a single *shared variable* ranging over a finite domain D

$$\mathsf{x}_1 := \ell_1; \ldots ; \mathsf{x}_\mathsf{k} := \ell_\mathsf{k};\ \mathsf{y} := d_0;\ \underline{\mathsf{do}}\ \mathsf{P}_1 \,[\!]\, \ldots \,[\!]\, \mathsf{P}_\mathsf{i} \,[\!]\, \ldots \,[\!]\, \mathsf{P}_\mathsf{k}\ \underline{\mathsf{od}}$$

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

$$
\begin{array}{l}
\mathrm{x}_1 := t;\ \ \mathrm{x}_2 := t;\ \ \mathrm{y} := 0 \\
\underline{\mathrm{do}} \quad
\begin{array}{l}
\quad \mathit{true} \rightarrow \underline{\mathrm{if}} \\
\qquad\quad \mathrm{x}_1 = t \wedge \mathrm{y} = 0 \rightarrow \mathrm{x}_1 := u; \mathrm{y} := 0 \\
\qquad [\!]\ \mathrm{x}_1 = u \wedge \mathrm{y} = 0 \rightarrow \mathrm{x}_1 := t; \mathrm{y} := 1 \\
\qquad\qquad\quad \underline{\mathrm{fi}}
\end{array}
\left\|
\begin{array}{l}
\mathit{true} \rightarrow \underline{\mathrm{if}} \\
\qquad \mathrm{x}_2 = t \wedge \mathrm{y} = 1 \rightarrow \mathrm{x}_2 := u; \mathrm{y} := 1 \\
\quad [\!]\ \mathrm{x}_2 = u \wedge \mathrm{y} = 1 \rightarrow \mathrm{x}_2 := t; \mathrm{y} := 0 \\
\qquad\qquad \underline{\mathrm{fi}}
\end{array}
\right.
\quad \underline{\mathrm{od}}
\end{array}
$$

Example (Cont'd)

$$\mathsf{x}_1 := t;\; \mathsf{x}_2 := t;\; \mathsf{y} := 0$$

$$\underline{\text{do}} \quad \begin{array}{l} \mathit{true} \to \underline{\text{if}} \\ \quad \mathsf{x}_1 = t \wedge \mathsf{y} = 0 \to \mathsf{x}_1 := u; \mathsf{y} := 0 \\ [\,]\; \mathsf{x}_1 = u \wedge \mathsf{y} = 0 \to \mathsf{x}_1 := t; \mathsf{y} := 1 \\ \quad \underline{\text{fi}} \end{array} \;\Bigg\|\; \begin{array}{l} \mathit{true} \to \underline{\text{if}} \\ \quad \mathsf{x}_2 = t \wedge \mathsf{y} = 1 \to \mathsf{x}_2 := u; \mathsf{y} := 1 \\ [\,]\; \mathsf{x}_2 = u \wedge \mathsf{y} = 1 \to \mathsf{x}_2 := t; \mathsf{y} := 0 \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}$$

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

$\langle t, t, 0\rangle$

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

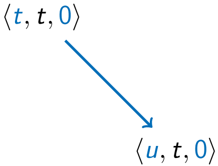
$$\lambda = \{\langle\langle t, t, 0\rangle, \{\mathsf{x}_1 = t, \mathsf{x}_2 = t, \mathsf{y} = 0\}\rangle, \ldots\}$$

Example (Cont'd)

$$x_1 := t;\ \ x_2 := t;\ \ y := 0$$

$$\underline{\mathrm{do}} \quad \begin{array}{l} \quad \mathit{true} \rightarrow \underline{\mathrm{if}} \\ \qquad \underline{x_1 = t \wedge y = 0 \rightarrow x_1 := u; y := 0} \\ \quad [\,]\ x_1 = u \wedge y = 0 \rightarrow x_1 := t; y := 1 \\ \qquad \underline{\mathrm{fi}} \end{array} \Bigg\| \begin{array}{l} \mathit{true} \rightarrow \underline{\mathrm{if}} \\ \qquad x_2 = t \wedge y = 1 \rightarrow x_2 := u; y := 1 \\ \quad [\,]\ x_2 = u \wedge y = 1 \rightarrow x_2 := t; y := 0 \\ \qquad \underline{\mathrm{fi}} \end{array} \quad \underline{\mathrm{od}}$$

$$S_0 = \{\langle t, t, 0\rangle\}$$
$$S = \{\langle t, t, 0\rangle, \langle u, t, 0\rangle, \ldots\}$$



$\langle t, t, 0\rangle$

$\langle u, t, 0\rangle$

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

$$\lambda = \{\langle\langle t, t, 0\rangle, \{x_1 = t, x_2 = t, y = 0\}\rangle, \ldots\}$$

Example (Cont'd)

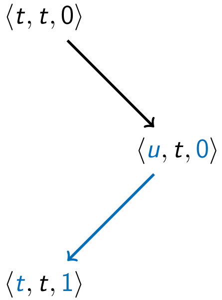
$$x_1 := t;\ \ x_2 := t;\ \ y := 0$$

$$\underline{\text{do}} \quad \begin{array}{l||l} \mathit{true} \rightarrow \underline{\text{if}} & \mathit{true} \rightarrow \underline{\text{if}} \\ \quad x_1 = t \wedge y = 0 \rightarrow x_1 := u; y := 0 & \quad x_2 = t \wedge y = 1 \rightarrow x_2 := u; y := 1 \\ [\!]\ \underline{x_1 = u \wedge y = 0 \rightarrow x_1 := t; y := 1} & [\!]\ x_2 = u \wedge y = 1 \rightarrow x_2 := t; y := 0 \\ \quad \underline{\text{fi}} & \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}$$

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

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

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



$$\lambda = \{\langle\langle t, t, 0\rangle, \{x_1 = t, x_2 = t, y = 0\}\rangle, \ldots\}$$

Example (Cont'd)

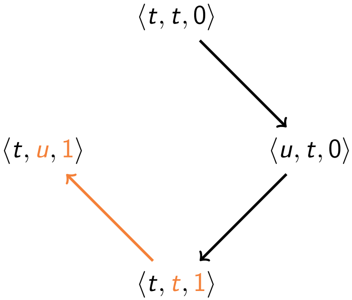
$$x_1 := t;\ \ x_2 := t;\ \ y := 0$$

$$\underline{\text{do}} \quad \begin{array}{l} \quad true \rightarrow \underline{\text{if}} \\ \qquad x_1 = t \wedge y = 0 \rightarrow x_1 := u; y := 0 \\ \quad [\!] \; x_1 = u \wedge y = 0 \rightarrow x_1 := t; y := 1 \\ \qquad \underline{\text{fi}} \end{array} \;\Bigg\|\; \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \quad \underline{x_2 = t \wedge y = 1 \rightarrow x_2 := u; y := 1} \\ [\!] \; x_2 = u \wedge y = 1 \rightarrow x_2 := t; y := 0 \\ \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}$$

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

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

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



$$\lambda = \{\langle\langle t, t, 0\rangle, \{x_1 = t, x_2 = t, y = 0\}\rangle, \ldots\}$$

Example (Cont'd)

$$x_1 := t;\ \ x_2 := t;\ \ y := 0$$

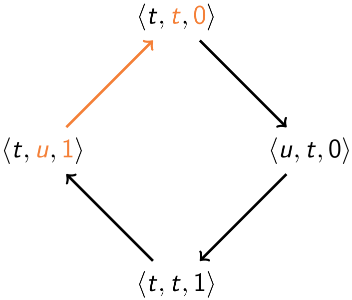
$$\underline{\text{do}} \quad \begin{array}{l} \mathit{true} \rightarrow \underline{\text{if}} \\ \qquad x_1 = t \land y = 0 \rightarrow x_1 := u; y := 0 \\ \quad [\,]\ x_1 = u \land y = 0 \rightarrow x_1 := t; y := 1 \\ \qquad \underline{\text{fi}} \end{array} \ \Big\| \ \begin{array}{l} \mathit{true} \rightarrow \underline{\text{if}} \\ \qquad x_2 = t \land y = 1 \rightarrow x_2 := u; y := 1 \\ \quad [\,]\ \underline{x_2 = u \land y = 1 \rightarrow x_2 := t; y := 0} \\ \qquad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}$$

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

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

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

$$\lambda = \{\langle\langle t, t, 0\rangle, \{x_1 = t, x_2 = t, y = 0\}\rangle, \ldots\}$$



Specification: Behavioural Properties

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**,....

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:

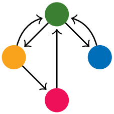
  $$\begin{array}{lll}
  \mathcal{K}, s \models b & \text{iff} & b \in \lambda(s)\\
  \mathcal{K}, s \models \neg\,\varphi & \text{iff} & \mathcal{K}, s \models \varphi \text{ does not hold}\\
  \mathcal{K}, s \models \varphi_1 \wedge \varphi_2 & \text{iff} & \mathcal{K}, s \models \varphi_1 \text{ and } \mathcal{K}, s \models \varphi_2\\
  \mathcal{K}, s \models \mathsf{EX}\,\varphi & \text{iff} & \text{there exists } \langle s, t\rangle \in R \text{ such that } \mathcal{K}, t \models \varphi\\
  \mathcal{K}, s \models \mathsf{EG}\,\varphi & \text{iff} & \text{there exists a path } \pi \text{ such that}\\
   & & \pi_0 = s \text{ and for all } i \geq 0,\ \ \mathcal{K}, \pi_i \models \varphi\\
  \mathcal{K}, s \models \mathsf{E}\,[\varphi_1\,\mathsf{U}\,\varphi_2] & \text{iff} & \text{there exists a path } \pi = \langle s, x_1, \ldots\rangle \text{ in } \mathcal{K} \text{ and } i \geq 0\\
   & & \text{such that } \mathcal{K}, \pi_i \models \varphi_2\\
   & & \text{and for all } 0 \leq j < i,\ \ \mathcal{K}, \pi_j \models \varphi_1
  \end{array}$$

Specification: Behavioural Properties

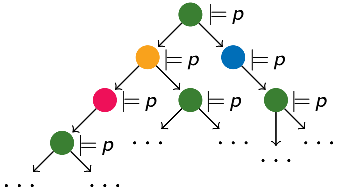
Computation Tree Logic

k-Process Concurrent Program satisfying p

M :

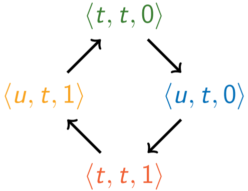


$$M, ● \models \mathrm{AG}\, p$$

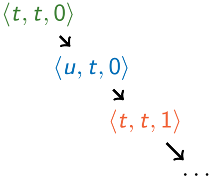


2-Process Concurrent Program satisfying mutual exclusion

K :



$$K, \langle t, t, 0\rangle \models \mathrm{AG}\, \neg(x_1 = u \wedge x_2 = u)$$



Specification: Behavioural Properties

A k-Process concurrent program satisfies a CTL formula φ
iff
the associated Kripke structure satisfies φ.

Specification: Structural Properties

Specification: Structural Properties

Symmetric Program Structure

- global property of a concurrent program C

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

  an element $d_0 \in D$
- pattern of execution of each process $\mathsf{P_i}$

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

  an element $\ell_0 \in L$

$$\sigma = \langle f, d_0, T, \ell_0 \rangle$$

Specification: Structural Properties

Symmetric Program Structure

k-generating function f is

either the identity function id

$$\begin{tikzcd}
P \arrow[r, "id"] & P \arrow[r, "id"] & \cdots \arrow[r, "id"] & P \arrow[r, "id"] & P \arrow[llll, bend left, "id"]
\end{tikzcd}$$

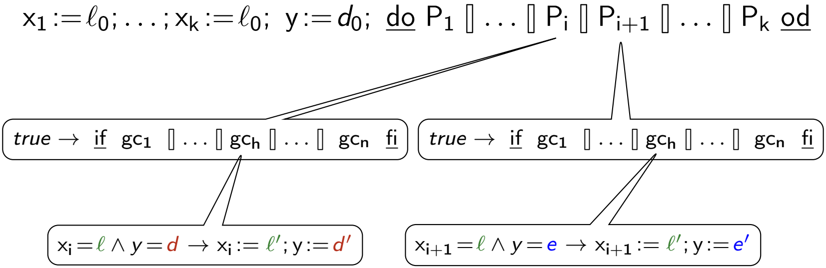
(Dijkstra's semaphore)

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

$$\begin{tikzcd}
P_1 \arrow[r, "f"] & P_2 \arrow[r, "f"] & \cdots \arrow[r, "f"] & P_{k-1} \arrow[r, "f"] & P_k \arrow[llll, bend left, "f"]
\end{tikzcd}$$

(Peterson's algorithm)

Symmetric Concurrent Programs



$$x_1 := \ell_0; \ldots ; x_k := \ell_0;\; y := d_0;\; \underline{\mathsf{do}}\ P_1 \,[\!]\, \ldots \,[\!]\, P_i \,[\!]\, P_{i+1} \,[\!]\, \ldots \,[\!]\, P_k\ \underline{\mathsf{od}}$$

$$\mathit{true} \rightarrow\ \underline{\mathsf{if}}\ \ gc_1\ [\!]\ \ldots\ [\!]\ gc_h\ [\!]\ \ldots\ [\!]\ \ gc_n\ \ \underline{\mathsf{fi}}$$

$$\mathit{true} \rightarrow\ \underline{\mathsf{if}}\ \ gc_1\ [\!]\ \ldots\ [\!]\ gc_h\ [\!]\ \ldots\ [\!]\ \ gc_n\ \ \underline{\mathsf{fi}}$$

$$x_i = \ell \wedge y = d \rightarrow x_i := \ell'; y := d'$$

$$x_{i+1} = \ell \wedge y = e \rightarrow x_{i+1} := \ell'; y := e'$$

A k-Process concurrent program satisfies $\langle f, d_0, T, \ell_0 \rangle$

iff

(i) for all i, for all h, $\langle \ell, \ell' \rangle \in T$, and

(ii) $f(d) = e$ and $f(d') = e'$

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric program structure σ :

$$f\colon \begin{tikzcd} 0 \arrow[dr] & 0 \\ 1 \arrow[ur] & 1 \end{tikzcd} \quad d_0 = 0 \quad T : \begin{tikzcd} t \arrow[loop above] \arrow[r, shift left] & u \arrow[l, shift left] \end{tikzcd} \quad \ell_0 = t$$

$$\mathit{true} \rightarrow \underline{\text{if}} \quad x_1 = t \wedge y = 0 \rightarrow x_1 := u \ ; \ y := 0$$

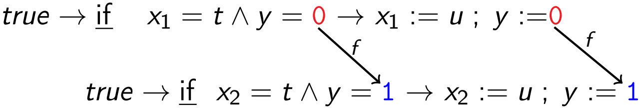
$$\talloblong \quad x_1 = t \wedge y = 1 \rightarrow x_1 := t \ ; \ y := 1$$

$$\talloblong \quad x_1 = u \wedge y = 0 \rightarrow x_1 := t \ ; \ y := 1 \ \underline{\text{fi}}$$

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric program structure σ : f : $$\begin{tikzcd} 0 \arrow[dr] & 0 \\ 1 \arrow[ur] & 1 \end{tikzcd}$$ $d_0 = 0$ T : $$\begin{tikzcd} t \arrow[loop above] \arrow[r, shift left] & u \arrow[l, shift left] \end{tikzcd}$$ $\ell_0 = t$



$$true \rightarrow \underline{\text{if}} \quad x_1 = t \wedge y = 0 \rightarrow x_1 := u \ ; \ y := 0$$

$$true \rightarrow \underline{\text{if}} \quad x_2 = t \wedge y = 1 \rightarrow x_2 := u \ ; \ y := 1$$

$$[] \quad x_1 = t \wedge y = 1 \rightarrow x_1 := t \ ; \ y := 1$$

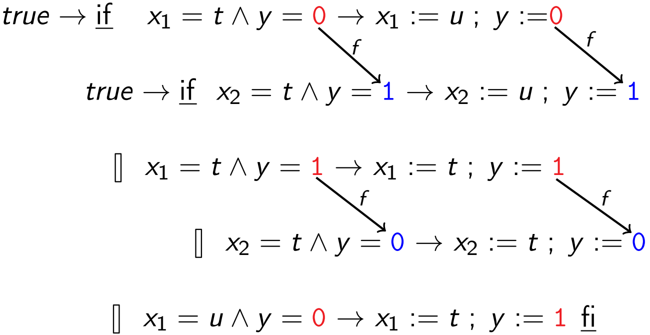
$$[] \quad x_1 = u \wedge y = 0 \rightarrow x_1 := t \ ; \ y := 1 \ \underline{\text{fi}}$$

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric program structure σ:

$$f\colon \begin{tikzcd} 0 \arrow[dr] & 0 \\ 1 \arrow[ur] & 1 \end{tikzcd} \qquad d_0 = 0 \qquad T : \begin{tikzcd} t \arrow[loop above] \arrow[r, shift left] & u \arrow[l, shift left] \end{tikzcd} \qquad \ell_0 = t$$



$true \to \underline{\text{if}}\quad x_1 = t \wedge y = 0 \to x_1 := u\ ;\ y := 0$

$true \to \underline{\text{if}}\quad x_2 = t \wedge y = 1 \to x_2 := u\ ;\ y := 1$

$[\!]\quad x_1 = t \wedge y = 1 \to x_1 := t\ ;\ y := 1$

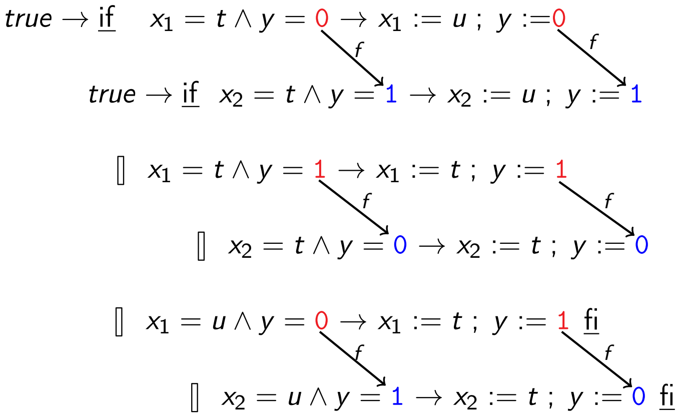
$[\!]\quad x_2 = t \wedge y = 0 \to x_2 := t\ ;\ y := 0$

$[\!]\quad x_1 = u \wedge y = 0 \to x_1 := t\ ;\ y := 1\ \underline{\text{fi}}$

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric program structure σ : f: $$\begin{tikzcd} 0 \arrow[dr] & 0 \\ 1 \arrow[ur] & 1 \end{tikzcd}$$ $d_0 = 0$ T : $$\begin{tikzcd} t \arrow[loop above] \arrow[r, shift left] & u \arrow[l, shift left] \end{tikzcd}$$ $\ell_0 = t$



$$true \rightarrow \underline{\text{if}} \quad x_1 = t \wedge y = 0 \rightarrow x_1 := u \; ; \; y := 0$$

$$true \rightarrow \underline{\text{if}} \quad x_2 = t \wedge y = 1 \rightarrow x_2 := u \; ; \; y := 1$$

$$[\!] \quad x_1 = t \wedge y = 1 \rightarrow x_1 := t \; ; \; y := 1$$

$$[\!] \quad x_2 = t \wedge y = 0 \rightarrow x_2 := t \; ; \; y := 0$$

$$[\!] \quad x_1 = u \wedge y = 0 \rightarrow x_1 := t \; ; \; y := 1 \; \underline{\text{fi}}$$

$$[\!] \quad x_2 = u \wedge y = 1 \rightarrow x_2 := t \; ; \; y := 0 \; \underline{\text{fi}}$$

Synthesis of a Concurrent Programs

Synthesis of a Concurrent Programs

Automatically derive a k-process concurrent program

$$x_1 := \ell_0;\; x_2 := \ell_0;\; y := d_0;$$

$$\underline{\text{do}} \quad \begin{array}{ll} & \mathit{true} \rightarrow \underline{\text{if}} \\ & x_1 = ? \wedge y = ? \rightarrow x_1 := ?; y := ? \\ [\!] & \quad\quad \cdots \\ \vdots & \quad\quad \vdots \\ [\!] & \quad\quad \cdots \\ [\!] & x_1 = ? \wedge y = ? \rightarrow x_1 := ?; y := ? \\ & \quad \underline{\text{fi}} \end{array} \;\Bigg\|\; \begin{array}{ll} & \mathit{true} \rightarrow \underline{\text{if}} \\ & x_2 = ? \wedge y = ? \rightarrow x_2 := ?; y := ? \\ [\!] & \quad\quad \cdots \\ \vdots & \quad\quad \vdots \\ [\!] & \quad\quad \cdots \\ [\!] & x_2 = ? \wedge y = ? \rightarrow x_2 := ?; y := ? \\ & \quad \underline{\text{fi}} \end{array} \quad \underline{\text{od}}$$

from a given formal specification consisting of

- Behavioural Properties
- Structural Properties

Synthesis Problem

Given:

1. a CTL formula φ (Behavioural Property),
2. a Program Structure $\sigma = \langle f, d_0, T, \ell_0 \rangle$ (Structural Property)

we look for a k-process concurrent program

$$\mathrm{x_1} := \ell_0; \ldots ; \mathrm{x_k} := \ell_0;\ \mathrm{y} := \mathrm{d_0}; \quad \underline{\mathrm{do}}\ \mathrm{P_1} \,[\!]\, \ldots \,[\!]\, \mathrm{P_i} \,[\!]\, \ldots \,[\!]\, \mathrm{P_k}\ \underline{\mathrm{od}}$$

such that C satisfies:

- $*$ σ (for all $i > 0$, $\langle \ell_i, \ell'_i \rangle \in T$, $f(P_i) = P_{(i\ mod\ k)+1}$) and
- $*$ φ ($\mathcal{K}, s_0 \vDash \varphi$)

Synthesis procedure:

1. to guess $\mathrm{P_1}$ satisfying T and
2. to generate the set $\mathrm{P_2}, \ldots, \mathrm{P}_k$ using f

such that the Kripke structure associated with C satisfies φ

Answer Set Programming

Logic Programming

Answer Set Programming

Declarative paradigm for solving combinatorial search problems

logic program $\Rightarrow$ encoding of a problem
answer set $\Rightarrow$ solution of a problem

- logic programs are set of Prolog like rules
  - ∗ with extensions: disjunctive rules, cardinality constraints, etc.

$$\mathrm{a}_1 \vee \ldots \vee \mathrm{a}_k \leftarrow \mathrm{b}_1, \ldots, \mathrm{b}_m, \text{not } \mathrm{c}_1, \ldots, \text{not } \mathrm{c}_n$$

  - ∗ generate & test programming methodology

generate: rules for generating candidate solutions
test: rules for eliminating invalid candidates

Example: Hamiltonian cycle

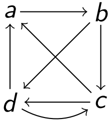
arc(a,b)
arc(b,c)
arc(b,d)
arc(c,a)
arc(c,d)
arc(d,a)
arc(d,c)

node(a)
node(b)
node(c)
node(d)

$\text{in(X,Y)} \vee \text{out(X,Y)} \leftarrow \text{arc(X,Y)}$
$\text{reached(X)} \leftarrow \text{in(X,Y)}$

$\leftarrow \text{in(X,Y)} \wedge \text{in(X,Z)} \wedge \text{Y} \neq \text{Z}$
$\leftarrow \text{in(X,Y)} \wedge \text{in(Z,Y)} \wedge \text{X} \neq \text{Z}$
$\leftarrow \text{node(X)} \wedge \text{not reached(Y)}$

$\Rightarrow$



Logic Programming

Answer Set Programming

Declarative paradigm for solving combinatorial search problems

logic program ⇒ encoding of a problem
answer set ⇒ solution of a problem

- answer sets are set of ground atoms which can be derived from a program

$$\text{logic program P} \longrightarrow \boxed{\text{ASP System}} \longrightarrow \{\, AS \mid AS \models \mathrm{P}\}$$

Example: Hamiltonian cycle

arc(a,b)
arc(b,c)
arc(b,d)
arc(c,a)
arc(c,d)
arc(d,a)
arc(d,c)

node(a)
node(b)
node(c)
node(d)

$\mathrm{in}(X,Y) \lor \mathrm{out}(X,Y) \leftarrow \mathrm{arc}(X,Y)$
$\mathrm{reached}(X) \leftarrow \mathrm{in}(X,Y)$

$\leftarrow \mathrm{in}(X,Y) \land \mathrm{in}(X,Z) \land Y \neq Z$
$\leftarrow \mathrm{in}(X,Y) \land \mathrm{in}(Z,Y) \land X \neq Z$
$\leftarrow \mathrm{node}(X) \land \mathrm{not\ reached}(Y)$

$$\longrightarrow \boxed{\text{ASP System}} \longrightarrow$$

{
{in(a,b), in(b,d),
in(a,b), in(b,d)},

{in(a,b), in(b,c),
in(c,d), in(d,a) }
}

Logic Programming

Answer Set Programming

Declarative paradigm for solving combinatorial search problems

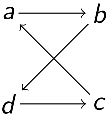
logic program ⇒ encoding of a problem
answer set ⇒ solution of a problem

- answer sets are set of ground atoms which can be derived from a program

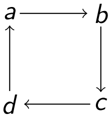
$$\text{logic program P} \longrightarrow \boxed{\text{ASP System}} \longrightarrow \{\, AS \mid AS \models \text{P}\}$$

Example: Hamiltonian cycle

{in(a,b), in(b,d),
in(a,b), in(b,d)}, ⇒



{in(a,b), in(b,c),
in(c,d), in(b,a)}, ⇒



ASP-based Synthesis Procedure

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

$$\boxed{\varphi + \sigma} \longrightarrow \boxed{\Pi_\varphi \cup \Pi_\sigma}$$

Π_σ: Program encoding structural properties

1.1 $\mathit{enabled}(1, X_1, Y) \vee \mathit{disabled}(1, X_1, Y) \leftarrow \mathit{reachable}(\langle X_1, \ldots, X_k, Y\rangle)$

1.2 $\mathit{gc}(1, X, Y, X_1, Y_1) \vee \ldots \vee \mathit{gc}(1, X, Y, X_m, Y_m) \leftarrow \mathit{enabled}(1, X, Y) \wedge$
$\mathit{candidates}(X, Y, [\langle X_1, Y_1\rangle, \ldots, \langle X_m, Y_m\rangle])$

2.1 $\mathit{enabled}(P, X, Y) \leftarrow \mathit{gc}(P, X, Y, X', Y')$

2.2.1 $\mathit{gc}(2, X, Z, X', Z') \leftarrow \mathit{gc}(1, X, Y, X', Y') \wedge \mathit{perm}(Y, Z) \wedge \mathit{perm}(Y', Z')$

$$\vdots$$

2.2.k $\mathit{gc}(k, X, Z, X', Z') \leftarrow \mathit{gc}(k-1, X, Y, X', Y') \wedge \mathit{perm}(Y, Z) \wedge \mathit{perm}(Y', Z')$

3.1 $\mathit{reachable}(s_0) \leftarrow \mathit{init}(s_0)$

3.2 $\mathit{reachable}(\langle X_1, \ldots, X_k, Y\rangle) \leftarrow \mathit{tr}(\langle X_1', \ldots, X_k', Y'\rangle, \langle X_1, \ldots, X_k, Y\rangle)$

4.1 $\mathit{tr}(\langle X_1, \ldots, X_k, Y\rangle, \langle X_1', \ldots, X_k, Y'\rangle) \leftarrow \mathit{reachable}(\langle X_1, \ldots, X_k, Y\rangle) \wedge \mathit{gc}(1, X_1, Y, X_1', Y')$

$$\vdots$$

4.k $\mathit{tr}(\langle X_1, \ldots, X_k, Y\rangle, \langle X_1, \ldots, X_k', Y'\rangle) \leftarrow \mathit{reachable}(\langle X_1, \ldots, X_k, Y\rangle) \wedge \mathit{gc}(k, X_k, Y, X_k', Y')$

5. $\leftarrow \mathit{reachable}(\langle X_1, \ldots, X_k, Y\rangle) \wedge \mathtt{not}\ \mathit{enabled}(1, X_1, Y) \wedge \ldots \wedge \mathtt{not}\ \mathit{enabled}(k, X_k, Y)$

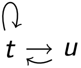
Π_φ: Program encoding behavioural properties

1. $\leftarrow \texttt{not}\ sat(s_0, \varphi) \wedge init(s_0)$
2. $sat(S, F) \leftarrow elem(F, S)$
3. $sat(S, not(F)) \leftarrow \texttt{not}\ sat(S, F)$
4. $sat(S, and(F_1, F_2)) \leftarrow sat(S, F_1) \wedge sat(S, F_2)$
5. $sat(S, ex(F)) \leftarrow tr(S, T) \wedge sat(T, F)$
6. $sat(S, eu(F_1, F_2)) \leftarrow sat(S, F_2)$
7. $sat(S, eu(F_1, F_2)) \leftarrow sat(S, F_1) \wedge tr(S, T) \wedge sat(T, eu(F_1, F_2))$
8. $sat(S, eg(F)) \leftarrow satpath(S, T, F) \wedge satpath(T, T, F)$
9. $satpath(S, T, F) \leftarrow sat(S, F) \wedge tr(S, T) \wedge sat(T, F)$
10. $satpath(S, V, F) \leftarrow sat(S, F) \wedge tr(S, T) \wedge satpath(T, V, F)$

Example

Encoding

Symmetric program structure σ: f: $\begin{tikzcd}[row sep=small] 0 \arrow[dr] & 0 \\ 1 \arrow[ur] & 1 \end{tikzcd}$ $d_0 = 0$ T :



$\ell_0 = t$

Behavioural property: $\varphi = \mathrm{AG}\,\neg(x_1 = u \wedge x_2 = u)$

$$\Pi = \{$$

$$\leftarrow not\ sat(s(t,t,0), n(ef(a(ep(x1,u), ep(x2,u))))).$$

$$sat(s(t,t,0), n(ef(a(ep(x1,u), ep(x2,u))))) \leftarrow$$
$$not\ sat(s(t,t,0), ef(a(ep(x1,u), ep(x2,u)))).$$

$$\ldots$$

$$sat(s(t,t,0), a(ap(s1,u), ap(s2,u))) \leftarrow$$
$$sat(s(t,t,0), ap(s1,u)), sat(s(t,t,0), ep(x2,u)).$$

$$\ldots$$

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

$$\ldots$$

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

$$\}$$

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

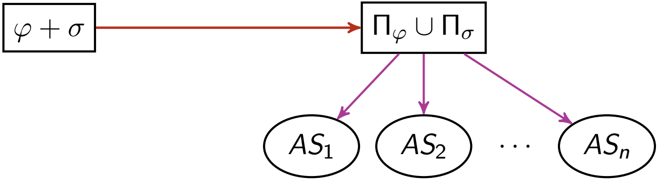
$$\boxed{\varphi + \sigma} \longrightarrow \boxed{\Pi_\varphi \cup \Pi_\sigma}$$

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

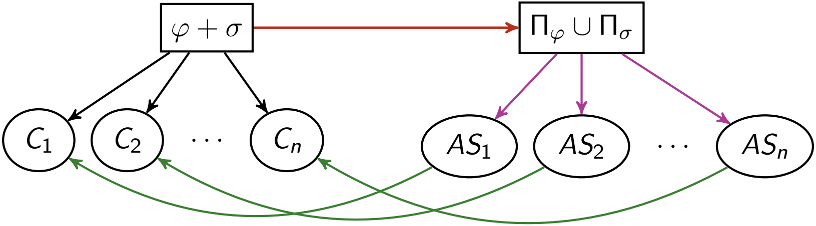


ASP-based Synthesis Procedure

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



Example

Decoding

$$K, s_0 \models AG\, \neg(x_1 = u \wedge x_2 = u)$$

$$\downarrow$$

$s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1$

$$\downarrow$$

$\{\ \ldots,$ sat(s(t,t,0),n(ef(n(n(a(ep(x_1,u),ep(x_2,u))))))), gc(1,u,0,t,1), reachable(s(u,t,0)), $\ldots$, tr(s(u,t,0),s(t,t,1)), tr(s(t,u,1),s(t,t,0)), tr(s(t,t,0),s(u,t,0)), tr(s(t,t,1),s(t,u,1)), $\ldots$, s0(t,t,0) $\}\ \in ans(\Pi)$

$$\uparrow$$

$\langle\langle t, t, 0\rangle, \langle u, t, 0\rangle\rangle \in R$

$s_0 \in S$

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* $\sigma = \langle f, d_0, T, \ell_0 \rangle$.

Then,

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

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

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

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

Experimental results

Experimental results

Examples

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

$$\mathrm{AG}\,\neg(x_i = u \land x_j = u)$$

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

$$\mathrm{AG}\,((x_i = t \rightarrow \mathrm{EX}\,x_i = w) \;\land\; (x_i = w \rightarrow \mathrm{AF}\,x_i = u))$$

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 $\{1, \ldots, k\}$,

$$\mathrm{AG}\,\neg\left[x_j = \mathtt{u} \land \mathrm{E}\left[x_i = \mathtt{w}\,\mathrm{U}\left(x_j = \mathtt{w} \land \mathrm{E}\left[x_i = \mathtt{w}\,\mathrm{U}\,(x_i = \mathtt{w} \land x_j = \mathtt{u})\right]\right)\right]\right]$$

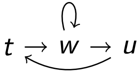
MR *Maximal Reactivity* (MR): if a process is waiting and all others are thinking then in the next state it will enter in use
for all i in $\{1, \ldots, k\}$,

$$\mathrm{AG}\,((x_i = w \land \bigwedge_{j \in \{1,\ldots,k\} \setminus \{i\}} x_j = t) \rightarrow \mathrm{EX}\,x_i = u)$$

Experimental results

Synthesis of *k*-process concurrent programs

T :



$$\begin{tikzcd}[row sep=small] id\colon 0 \arrow[r] & 0 \\ 1 \arrow[r] & 1 \end{tikzcd}
\qquad
\begin{tikzcd}[row sep=small] f_1\colon 0 \arrow[dr] & 0 \\ 1 \arrow[ur] & 1 \end{tikzcd}
\qquad
\begin{tikzcd}[row sep=small] f_2\colon 0 \arrow[dr] & 0 \\ 1 \arrow[ur] & 1 \\ 2 \arrow[r] & 2 \end{tikzcd}
\qquad
\begin{tikzcd}[row sep=small] f_3\colon 0 \arrow[dr] & 0 \\ 1 \arrow[dr] & 1 \\ 2 \arrow[uur] & 2 \end{tikzcd}$$

Program	Satisfied Properties	$\lvert D\rvert$	f	$\lvert ans(\Pi)\rvert$	Time (sec)
mutex for 2 processes	*ME*	2	id	10	0.011
	ME	2	f_1	10	0.012
	ME, *SF*	2	f_1	2	0.032
	ME, *SF*, *BO*	2	f_1	2	0.045
	ME, *SF*, *BO*, *MR*	3	f_2	2	0.139
mutex for 3 processes	*ME*	2	id	9	0.036
	ME	2	f_1	14	0.036
	ME, *SF*	3	f_3	6	3.487
	ME, *SF*, *BO*	3	f_3	4	4.323

A 2-process protocol satisfying:

- Mutual Exclusion
- Progression with Starvation Freedom
- Bounded Overtaking
- Maximal Reactivity

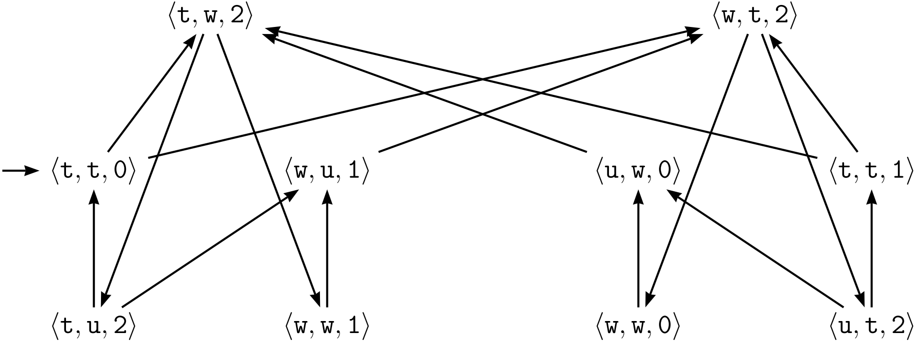
$x_1 := t; x_2 := t; y := 0$

```
$P_1$ :  $true \rightarrow$ if
         $x_1 = t \wedge y = 0 \rightarrow x_1 := w;\ y := 2;$
    ▯    $x_1 = t \wedge y = 1 \rightarrow x_1 := w;\ y := 2;$
    ▯    $x_1 = t \wedge y = 2 \rightarrow x_1 := w;\ y := 1;$
    ▯    $x_1 = w \wedge y = 0 \rightarrow x_1 := u;\ y := 0;$
    ▯    $x_1 = w \wedge y = 2 \rightarrow x_1 := u;\ y := 2;$
    ▯    $x_1 = u \wedge y = 2 \rightarrow x_1 := t;\ y := 1;$
    ▯    $x_1 = u \wedge y = 0 \rightarrow x_1 := t;\ y := 2;$
fi
```

```
$P_2$ :  $true \rightarrow$ if
         $x_2 = t \wedge y = 0 \rightarrow x_2 := w;\ y := 2;$
    ▯    $x_2 = t \wedge y = 1 \rightarrow x_2 := w;\ y := 2;$
    ▯    $x_2 = t \wedge y = 2 \rightarrow x_2 := w;\ y := 0;$
    ▯    $x_2 = w \wedge y = 1 \rightarrow x_2 := u;\ y := 1;$
    ▯    $x_2 = w \wedge y = 2 \rightarrow x_2 := u;\ y := 2;$
    ▯    $x_2 = u \wedge y = 2 \rightarrow x_2 := t;\ y := 0;$
    ▯    $x_2 = u \wedge y = 1 \rightarrow x_2 := t;\ y := 2;$
fi
```

A 2-process protocol satisfying:

- Mutual Exclusion
- Progression with Starvation Freedom
- Bounded Overtaking
- Maximal Reactivity



Complexity of the synthesis procedure

Theorem

For any number $k > 1$ *of processes, for any symmetric program structure* σ *over* $\mathcal{L}$ *and* $\mathcal{D}$*, and for any CTL formula* φ*, an answer set of the logic program* $\Pi_\varphi \cup \Pi_\sigma$ *can be computed in*

(i) exponential time w.r.t. k*,*

(ii) linear time w.r.t. $|\varphi|$*, and*

(iii) nondeterministic polynomial time w.r.t. $|\mathcal{L}|$ *and w.r.t.* $|\mathcal{D}|$*.*

Conclusions

- reduction of the design of a concurrent program to the design of its formal specification
- fully declarative solution (independent of the ASP solver)
- future work:
  - exploit CTL formulas symmetries,
  - exploit Kripke structures symmetries,
  - reduce the atomicity of transitions,
  - ...

Conclusions

- reduction of the design of a concurrent program to the design of its formal specification
- fully declarative solution (independent of the ASP solver)
- future work:
  - exploit CTL formulas symmetries,
  - exploit Kripke structures symmetries,
  - reduce the atomicity of transitions,
  - . . .

Thank you!