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

26th CILC, Pescara, Italy
31th August - 2nd September 2011

Overview and Related Work

A bit of history

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
- P. C. Attie and E. A. Emerson
  Synthesis of Concurrent Programs with Many Similar Processes
  ACM TOPLAS, 1998

Answer Set Programming

- K. Heljanko and I. Niemelä
  Bounded LTL Model Checking with Stable Models
  Theory and Practice of Logic Programming vol.3, 2003
- S. Heymans, D. Van Nieuwenborgh and D. Vermeir
  Synthesis from Temporal Specifications using Preferred Answer Set Programming
  LNCS no. 3701, 2005

Overview and Related Work

Overview of our approach

- 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

Concurrent Programs

Syntax

k-Process Concurrent Program

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

$$C: \quad s_1 := \ell_1; \ldots; s_k := \ell_k;\ x := d_0;\ \underline{\text{do}}\ \mathrm{P}_1 \mathbin{[\!]} \ldots \mathbin{[\!]} \mathrm{P}_i \mathbin{[\!]} \ldots \mathbin{[\!]} \mathrm{P}_k\ \underline{\text{od}}$$

$$\mathrm{P}_i: \quad \boxed{\mathit{true} \rightarrow\ \underline{\text{if}}\ \ gc_1 \mathbin{[\!]} \ldots \mathbin{[\!]} gc_h \mathbin{[\!]} \ldots \mathbin{[\!]} gc_n\ \ \underline{\text{fi}}}$$

$$gc_h: \quad \boxed{s_i = \ell \wedge x = d \ \rightarrow\ s_i := \ell'; x := d';}$$

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

Concurrent Programs

Syntax: Example

A 2-Process Concurrent Program C:

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

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

$$\begin{array}{l}
P_1 : \mathit{true} \rightarrow \underline{\text{if}} \\
\quad\ \ s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0; \\
\quad [\!]\ s_1 = t \wedge x = 1 \rightarrow \mathit{skip}; \\
\quad [\!]\ s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1; \\
\qquad \underline{\text{fi}}
\end{array}$$

$$\begin{array}{l}
P_2 : \mathit{true} \rightarrow \underline{\text{if}} \\
\quad\ \ s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1; \\
\quad [\!]\ s_2 = t \wedge x = 0 \rightarrow \mathit{skip}; \\
\quad [\!]\ s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0; \\
\qquad \underline{\text{fi}}
\end{array}$$

Concurrent Programs

Semantics

k-Process Concurrent Program C

- processes $P_1, \ldots, P_k$
- *local state* variables $s_1, \ldots, s_k$ ranging over L
- *shared* variable x ranging over D

$$s_1 := \ell_1; \ldots; s_k := \ell_k; x := d_0;\ \underline{\text{do}}\ \mathrm{P}_1 \,[\!]\, \ldots \left\| \begin{array}{l} \mathit{true} \rightarrow \underline{\text{if}} \quad \begin{array}{c} \ldots \end{array} \\ \qquad\quad\;\; [\!] \;\; s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell_i'; x := d_0' \\ \qquad\quad\;\; [\!] \qquad\qquad \ldots \\ \qquad\quad\;\; \underline{\text{fi}} \end{array} \right\| \ldots \,[\!]\, \mathrm{P}_k\ \underline{\text{od}}$$

Execution of C: Nondeterministic Interleaving of $P_1, \ldots, P_k$

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

Concurrent Programs

Semantics

k-Process Concurrent Program C

- processes $P_1, \ldots, P_k$
- *local state* variables $s_1, \ldots, s_k$ ranging over L
- *shared* variable x ranging over D

$$s_1 := \ell_1; \ldots; s_k := \ell_k; x := d_0;\ \underline{\text{do}}\ \mathrm{P}_1 \ [\!]\ \ldots \left\| \begin{array}{ll} \mathit{true} \rightarrow \underline{\text{if}} & \ldots \\ \quad [\!] & s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell_i'; x := d_0' \\ \quad [\!] & \ldots \\ \quad \underline{\text{fi}} & \end{array} \right\| \ldots\ [\!]\ \mathrm{P}_k\ \underline{\text{od}}$$

Execution of C: Nondeterministic Interleaving of $P_1, \ldots, P_k$

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

- set of *states*

  $S = \{u_0, u_1, \ldots, u_n\} = L^k \times D$

u_0

u_1 u_2

u_3 u_4 u_5

Concurrent Programs

Semantics

k-Process Concurrent Program C

* processes $P_1, \ldots, P_k$
* *local state* variables $s_1, \ldots, s_k$ ranging over L
* *shared* variable x ranging over D

$$s_1 := \ell_1; \ldots; s_k := \ell_k; x := d_0;\ \underline{\text{do}}\ \mathrm{P}_1 \ [\!]\ \ldots \left\| \begin{array}{l} \mathit{true} \rightarrow \underline{\text{if}} \quad \ldots \\ \qquad\quad [\!]\ s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell_i'; x := d_0' \\ \qquad\quad [\!] \quad \ldots \\ \qquad\quad \underline{\text{fi}} \end{array} \right\| \ldots\ [\!]\ \mathrm{P}_k\ \underline{\text{od}}$$

Execution of C: Nondeterministic Interleaving of $P_1, \ldots, P_k$

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

* set of *states*
  $S = \{u_0, u_1, \ldots, u_n\} = L^k \times D$
* *initial state* $S_0 = \{u_0\} \subseteq S$

$u_0 = \langle \ell_1, \ldots, \ell_i, \ldots, \ell_k, d_0 \rangle$

u_1 u_2

u_3 u_4 u_5

Concurrent Programs

Semantics

k-Process Concurrent Program C

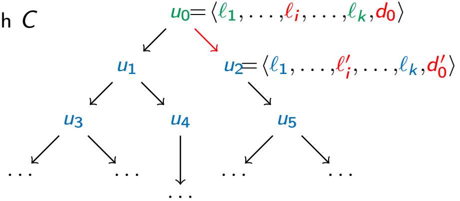
- processes $P_1, \ldots, P_k$
- *local state* variables $s_1, \ldots, s_k$ ranging over L
- *shared* variable x ranging over D

$$s_1 := \ell_1; \ldots; s_k := \ell_k; x := d_0;\ \underline{\text{do}}\ \mathrm{P}_1\ [\!]\ \ldots \left\| \begin{array}{l} \mathit{true} \to \underline{\text{if}} \\ \qquad\qquad \ldots \\ \quad [\!]\ s_i = \ell_i \wedge x = d_0 \to s_i := \ell'_i; x := d'_0 \\ \qquad\qquad \ldots \\ \quad [\!] \\ \quad \underline{\text{fi}} \end{array} \right\| \ldots\ [\!]\ \mathrm{P}_k\ \underline{\text{od}}$$

Execution of C: Nondeterministic Interleaving of $P_1, \ldots, P_k$

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

- set of *states* $S = \{u_0, u_1, \ldots, u_n\} = L^k \times D$
- *initial state* $S_0 = \{u_0\} \subseteq S$
- total *transition* relation $R = \{\langle u_0, u_1 \rangle, \langle u_2, u_5 \rangle \ldots\} \subseteq S \times S$



$\pi = s_0, \ldots, s_i, s_{i+1}, \ldots$
path over $\mathcal{K}$, $\forall i\ \langle s_i, s_{i+1} \rangle \in R$
$(\pi_i = s_i)$

Concurrent Programs

Semantics

k-Process Concurrent Program C

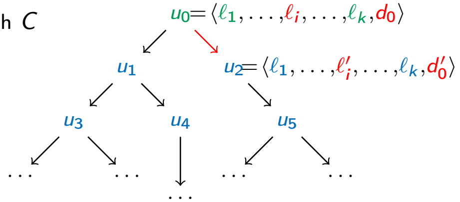
- processes $P_1, \ldots, P_k$
- *local state* variables $s_1, \ldots, s_k$ ranging over L
- *shared* variable x ranging over D

$$s_1 := \ell_1; \ldots; s_k := \ell_k; x := d_0;\ \underline{\text{do}}\ \mathrm{P}_1 \,[\!]\, \ldots \left\| \begin{array}{l} true \rightarrow \underline{\text{if}} \\ \qquad\qquad [\!]\ \ldots \\ \qquad\qquad [\!]\ s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell'_i; x := d'_0 \\ \qquad\qquad [\!]\ \ldots \\ \qquad\quad \underline{\text{fi}} \end{array} \right\| \ldots \,[\!]\, \mathrm{P}_k\ \underline{\text{od}}$$

Execution of C: Nondeterministic Interleaving of $P_1, \ldots, P_k$

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

- set of *states*
  $S = \{u_0, u_1, \ldots, u_n\} = L^k \times D$
- *initial state* $S_0 = \{u_0\} \subseteq S$
- total *transition* relation
  $R = \{\langle u_0, u_1 \rangle, \langle u_2, u_5 \rangle \ldots\} \subseteq S \times S$
- *labelling* function
  $\lambda = \{\langle u_0, \{s_1 = \ell_1, \ldots, s_i = \ell_i, \ldots, s_k = \ell_k\rangle, \} \ldots\}$



$\pi = s_0, \ldots, s_i, s_{i+1}, \ldots$
path over $\mathcal{K}$, $\forall i\ \langle s_i, s_{i+1} \rangle \in R$
$(\pi_i = s_i)$

Example

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

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

Example (Cont'd)

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

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

$$\langle t, t, 0 \rangle$$

Example (Cont'd)

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

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

$$
\begin{tikzcd}
\langle t, t, 0 \rangle \arrow[dr] & \\
& \langle u, t, 0 \rangle
\end{tikzcd}
$$

Example (Cont'd)

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

$$\underline{\mathrm{do}}\quad \begin{array}{l} \mathit{true} \rightarrow \underline{\mathrm{if}} \\ \quad \mathsf{s}_1 = t \wedge x = 0 \rightarrow \mathsf{s}_1 := u;\, x := 0; \\ \ [\!]\ \mathsf{s}_1 = t \wedge x = 1 \rightarrow \mathit{skip}; \\ \ [\!]\ \mathsf{s}_1 = u \wedge x = 0 \rightarrow \mathsf{s}_1 := t;\, x := 1; \\ \qquad \underline{\mathrm{fi}} \end{array} \ \Big\|\ \begin{array}{l} \mathit{true} \rightarrow \underline{\mathrm{if}} \\ \quad \mathsf{s}_2 = t \wedge x = 1 \rightarrow \mathsf{s}_2 := u;\, x := 1; \\ \ [\!]\ \mathsf{s}_2 = t \wedge x = 0 \rightarrow \mathit{skip}; \\ \ [\!]\ \mathsf{s}_2 = u \wedge x = 1 \rightarrow \mathsf{s}_2 := t;\, x := 0; \\ \qquad \underline{\mathrm{fi}} \end{array} \quad \underline{\mathrm{od}}$$

$$\begin{tikzcd} \langle t, t, 0\rangle \arrow[dr] & \\ & \langle u, t, 0\rangle \arrow[dl] \\ \langle t, t, 1\rangle & \end{tikzcd}$$

Example (Cont'd)

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

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

$$
\begin{tikzcd}
 & \langle t, t, 0 \rangle \arrow[dr] & \\
\langle t, u, 1 \rangle & & \langle u, t, 0 \rangle \arrow[dl] \\
 & \langle t, t, 1 \rangle \arrow[ul] &
\end{tikzcd}
$$

Example (Cont'd)

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

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

$$\begin{tikzcd}
 & \langle t, t, 0 \rangle \arrow[dr] & \\
\langle u, t, 1 \rangle \arrow[ur] & & \langle u, t, 0 \rangle \arrow[dl] \\
 & \langle t, t, 1 \rangle \arrow[ul] &
\end{tikzcd}$$

Specification: Behavioural Properties

Time dependant behavioural properties of Concurrent Programs:

- liveness, something 'good' will eventually happen
- safety, something 'bad' will never happen

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: $\varphi ::= b \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathsf{EX}\,\varphi \mid \mathsf{EG}\,\varphi \mid \mathsf{E}[\varphi_1\,\mathsf{U}\,\varphi_2]$
  $\mathsf{AG}\varphi \equiv \neg\mathsf{EF}\neg\,\varphi$, etc.
- Semantics:
  $$\left.\begin{array}{r}\text{Kripke structure } \mathcal{K}\\ \text{state } s\\ \text{CTL formula } \varphi\end{array}\right\}\quad \mathcal{K}, s \models \varphi$$

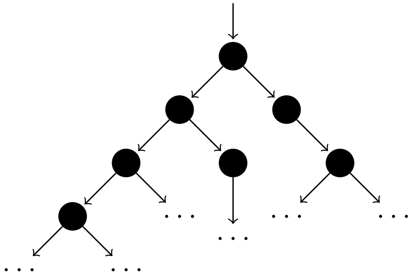
  recursively defined as follows:

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

Specification: Behavioural Properties

Computation Tree Logic

AG p



2-Process Concurrent Program for mutual exclusion: $\mathsf{AG}\,\neg(\mathsf{s}_1 = u \wedge \mathsf{s}_2 = u)$

$$\begin{tikzcd}
 & \langle t, t, 0\rangle \arrow[dr] & \\
\langle u, t, 1\rangle \arrow[ur] & & \langle u, t, 0\rangle \arrow[dl] \\
 & \langle t, t, 1\rangle \arrow[ul] &
\end{tikzcd}$$

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

$$\begin{tikzcd}
\langle t, t, 0\rangle \arrow[r] & \langle u, t, 0\rangle \arrow[r] & \langle t, t, 1\rangle \arrow[r] & \langle u, t, 1\rangle \arrow[d] \\
\langle u, t, 1\rangle \arrow[d] & \langle t, t, 1\rangle \arrow[l] & \langle u, t, 0\rangle \arrow[l] & \langle t, t, 0\rangle \arrow[l] \\
\cdots & & &
\end{tikzcd}$$

Specification: Structural Properties

Symmetric Program Structure

- Intra-process properties
  - e.g. $T \subseteq L \times L$, local transition relation
    - pattern of execution of each process
    - local property of each process
- Inter-process properties
  - e.g. $f : D \to D$, k-generating function
    - permutation (bijection) on D (domain of the shared variable)
    - pattern shared by processes $P_i's$
    - global property of 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:

* $f = id$
* f is a generator of a cyclic group

$$G_f = \{id, f, f^2, \ldots, f^{k-1}\}$$

of order k

$$\begin{tikzcd}
P_1 \arrow[r, "f"] & P_2 \arrow[d, "f"] \\
 & \vdots \arrow[d, "f"] \\
P_k \arrow[uu, "f"] & P_{k-1} \arrow[l, "f"]
\end{tikzcd}$$

$$f^k = id$$

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric Program Structure Σ

- $T = \{\langle t, u\rangle, \langle u, t\rangle, \langle t, t\rangle\}$
- $f = \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$ of order 2: $f^2 = id$

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

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

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

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric Program Structure Σ

- $T = \{\langle t, u\rangle, \langle u, t\rangle, \langle t, t\rangle\}$
- $f = \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$ of order 2: $f^2 = id$

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

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

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

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

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric Program Structure Σ

- $T = \{\langle t, u\rangle, \langle u, t\rangle, \langle t, t\rangle\}$
- $f = \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$ of order 2: $f^2 = id$

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

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

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

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

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

Example

A 2-Process Symmetric Concurrent Program for Mutual Exclusion

Symmetric Program Structure Σ

- $T = \{\langle t, u\rangle, \langle u, t\rangle, \langle t, t\rangle\}$
- $f = \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$ of order 2: $f^2 = id$

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

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

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

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

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

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

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

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

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

Synthesis Problem

The synthesis problem of a k-process symmetric concurrent program C starting from:

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

consists in finding a k-Process Symmetric Concurrent Program C such that

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

We guess a process P_i whose local structure satisfies T and we generate a set of processes $\mathrm{P}_1, \ldots, \mathrm{P}_k$ using f such that the Kripke structure associated with C satisfies φ

ASP-based Synthesis Procedure

Answer set program Π is the union of:

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

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

- answer set program Π encodes a synthesis problem,
- answer sets encode solutions to the problem

Logic program based synthesis

Encoding behavioural properties

φ, a CTL formula representing a behavioural property

We encode φ and the CTL satisfaction relation as a logic program:

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

Logic program based synthesis

Encoding behavioural properties (Cont'd)

11.1 $tr(s(S_1, \ldots, S_k, X), s(S'_1, \ldots, S'_k, X')) \leftarrow$

$reachable(s(S_1, \ldots, S_k, X)) \wedge$

$gc(1, S_1, X, S'_1, X') \wedge \langle S_1, X \rangle \neq \langle S'_1, X' \rangle$

$\ldots$

11.k $tr(s(S_1, \ldots, S_k, X), s(S'_1, \ldots, S'_k, X')) \leftarrow$

$reachable(s(S_1, \ldots, S_k, X)) \wedge$

$gc(k, S_k, X, S'_k, X') \wedge \langle S_k, X \rangle \neq \langle S'_k, X' \rangle$

12. $\leftarrow \mathtt{not}\ out(S) \wedge reachable(S)$
13. $out(S) \leftarrow tr(S, Z)$
14. $reachable(s_0) \leftarrow$
15. $reachable(S) \leftarrow tr(Z, S)$

Logic program based synthesis

Encoding structural Properties

$$\Sigma = \langle f, T \rangle, \text{ a symmetric program structure}$$

We encode Σ as follows:

1.1 $\bigvee_{\langle S', X' \rangle \in Next(\langle S_1, X \rangle)} gc(1, S_1, X, S', X') \leftarrow reachable(S_1, \ldots, S_k, X)$

1.2 $\leftarrow gc(1, S, X, S', X') \wedge gc(1, S, X, S'', X'') \wedge \langle S', X' \rangle \neq \langle S'', X'' \rangle$

2.1 $gc(2, S, f(X), S', f(X')) \leftarrow gc(1, S, X, S', X')$

2.2 $\leftarrow gc(2, S, X, S', X') \wedge \texttt{not } ps(2, S, X)$

2.3 $ps(2, S_2, X) \leftarrow reachable(S_1, S_2, \ldots, S_k, X)$

$\ldots$

k.1 $gc(k, S, f(X), S', f(X')) \leftarrow gc(k-1, S, X, S', X')$

k.2 $\leftarrow gc(k, S, X, S', X') \wedge \texttt{not } ps(k, S, X)$

k.3 $ps(k, S_k, X) \leftarrow reachable(S_1, S_2, \ldots, S_k, X)$

where $Next(\ell, d) = \{\langle \ell', d' \rangle \mid \ell \mapsto \ell' \in T \wedge d' \in D\}$.

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,

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

iff there exists an answer set M in $ans(\Pi)$ such that
$\forall i \in \{1, \ldots, k\}$, $\forall \ell, \ell' \in L$, $\forall d, d' \in D$,
$\big(\mathsf{s}_i = \ell \wedge x = d \to \mathsf{s}_i := \ell'; x := d'\big)$ *is in P_i iff* $M \models \mathrm{gc}(i, \ell, d, \ell', d')$

Experimental results

Examples

- *Mutual Exclusion* (ME): it is not the case that process P_i is in its critical section ($\mathtt{s}_i = \mathtt{u}$), and process P_j is in its critical section ($\mathtt{s}_j = \mathtt{u}$) at the same time: for all i, j in $\{1, \ldots, k\}$, with $i \neq j$,
$$\mathsf{AG}\,\neg(\mathtt{s}_i = \mathtt{u} \land \mathtt{s}_j = \mathtt{u})$$
- *Starvation Freedom* (SF): if a process is waiting to enter the critical section ($\mathtt{s}_i = \mathtt{w}$), then after a finite amount of time, it will execute its critical section ($\mathtt{s}_i = \mathtt{u}$): for all i in $\{1, \ldots, k\}$,
$$\mathsf{AG}\,(\mathtt{s}_i = \mathtt{w} \rightarrow \mathsf{AF}\,\mathtt{s}_i = \mathtt{u})$$
- *Bounded Overtaking* (BO): while process P_i is in its waiting section, any other process P_j exits from its critical section at most once: for all i, j in $\{1, \ldots, k\}$,
$$\mathsf{AG}\,((\mathtt{s}_i = \mathtt{w} \land \mathtt{s}_j = \mathtt{u}) \rightarrow \mathsf{AF}\,(\mathtt{s}_j = \mathtt{t} \land \mathsf{A}[\neg(\mathtt{s}_j = \mathtt{u})\,\mathsf{U}\,\mathtt{s}_i = \mathtt{u}]))$$
- *Maximal Reactivity* (MR): if process P_i is waiting to execute the critical section and all other processes are executing their noncritical sections, then in the next state P_i will enter its critical section: for all i in $\{1, \ldots, k\}$,
$$\mathsf{AG}\,((\mathtt{s}_i = \mathtt{w} \land \bigwedge_{j \in \{1,\ldots,k\} \setminus \{i\}} \mathtt{s}_j = \mathtt{t}) \rightarrow \mathsf{EX}\,\mathtt{s}_i = \mathtt{u})$$

Experimental results

Synthesis of k-Process Symmetric Concurrent Programs for the mutual exclusion problem satisfying some Behavioural properties in { ME, SF, BO, MR}.

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

$f1 = \{\langle 0,1\rangle, \langle 1,0\rangle\}$
$f2 = \{\langle 0,1\rangle, \langle 1,0\rangle, \langle 2,2\rangle\}$
$f3 = \{\langle 0,1\rangle, \langle 1,2\rangle, \langle 2,0\rangle\}$
$T = \{\langle t,w\rangle, \langle w,w\rangle, \langle w,u\rangle, \langle u,t\rangle\}$