

Synthesizing Concurrent Programs using Answer Set Programming

E. De Angelis¹, A. Pettorossi², M. Proietti³

¹University of Chieti-Pescara 'G. D'Annunzio'

²University of Rome 'Tor Vergata'

³CNR-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, \dots, P_k , *processes*
- ▶ s_1, \dots, s_k , *local state* variables ranging over a finite domain L
- ▶ x , a single *shared* variable ranging over a finite domain D

C : $s_1 := l_1; \dots; s_k := l_k; x := d_0; \underline{\text{do}} P_1 \parallel \dots \parallel P_i \parallel \dots \parallel P_k \underline{\text{od}}$

P_i :

$\text{true} \rightarrow \underline{\text{if}} \ gc_1 \parallel \dots \parallel gc_h \parallel \dots \parallel gc_n \ \underline{\text{fi}}$

gc_h :

$s_i = l \wedge x = d \rightarrow s_i := l'; x := d';$

where $l_1, \dots, l_k, l, l' \in L$, 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 $D = \{0, 1\}$

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

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

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

Concurrent Programs

Semantics

k -Process Concurrent Program C

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

$$s_1 := \ell_1; \dots; s_k := \ell_k; x := d_0; \underline{\text{do}} P_1 \parallel \dots \left[\begin{array}{l} \text{true} \rightarrow \underline{\text{if}} \\ \parallel s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell'_i; x := d'_0 \\ \parallel \\ \underline{\text{fi}} \end{array} \right] \dots \parallel P_k \underline{\text{od}}$$

Execution of C : Nondeterministic Interleaving of P_1, \dots, 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, \dots, P_k
- * *local state* variables s_1, \dots, s_k ranging over L
- * *shared* variable x ranging over D

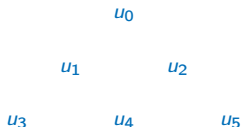
$$s_1 := \ell_1; \dots; s_k := \ell_k; x := d_0; \underline{\text{do}} P_1 \parallel \dots \left[\begin{array}{l} \text{true} \rightarrow \text{if} \\ \parallel \\ s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell'_i; x := d'_0 \\ \parallel \\ \text{fi} \\ \dots \end{array} \right] \dots \parallel P_k \underline{\text{od}}$$

Execution of C : Nondeterministic Interleaving of P_1, \dots, P_k

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

- * set of *states*

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



Concurrent Programs

Semantics

k -Process Concurrent Program C

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

$$s_1 := \ell_1; \dots; s_k := \ell_k; x := d_0; \underline{\text{do}} P_1 \parallel \dots \left[\begin{array}{l} \text{true} \rightarrow \text{if} \\ \parallel \\ s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell'_i; x := d'_0 \\ \parallel \\ \text{fi} \end{array} \right] \dots \parallel P_k \underline{\text{od}}$$

Execution of C : Nondeterministic Interleaving of P_1, \dots, P_k

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

- * set of *states*

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

- * *initial state* $S_0 = \{u_0\} \subseteq S$

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

u_1

u_2

u_3

u_4

u_5

Concurrent Programs

Semantics

k -Process Concurrent Program C

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

$$s_1 := \ell_1; \dots; s_k := \ell_k; x := d_0; \underline{\text{do}} P_1 \parallel \dots \parallel \left[\begin{array}{l} \text{true} \rightarrow \text{if} \\ \parallel \\ s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell'_i; x := d'_0 \\ \dots \\ \text{fi} \end{array} \right] \dots \parallel P_k \underline{\text{od}}$$

Execution of C : Nondeterministic Interleaving of P_1, \dots, P_k

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

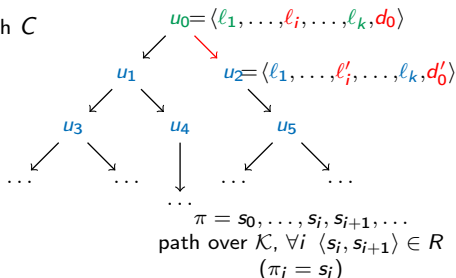
- * set of states

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

- * initial state $S_0 = \{u_0\} \subseteq S$

- * total transition relation

$$R = \{ \langle u_0, u_1 \rangle, \langle u_2, u_5 \rangle \dots \} \subseteq S \times S$$



Concurrent Programs

Semantics

k -Process Concurrent Program C

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

$$s_1 := \ell_1; \dots; s_k := \ell_k; x := d_0; \underline{\text{do}} P_1 \parallel \dots \parallel \left[\begin{array}{l} \text{true} \rightarrow \text{if} \\ \parallel \\ s_i = \ell_i \wedge x = d_0 \rightarrow s_i := \ell'_i; x := d'_0 \\ \parallel \\ \text{fi} \\ \dots \end{array} \right] \dots \parallel P_k \underline{\text{od}}$$

Execution of C : Nondeterministic Interleaving of P_1, \dots, P_k

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

- * set of states

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

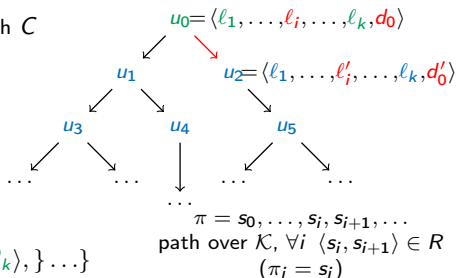
- * initial state $S_0 = \{u_0\} \subseteq S$

- * total transition relation

$$R = \{ \langle u_0, u_1 \rangle, \langle u_2, u_5 \rangle \dots \} \subseteq S \times S$$

- * labelling function

$$\lambda = \{ \langle u_0, \{s_1 = \ell_1, \dots, s_i = \ell_i, \dots, s_k = \ell_k\} \rangle, \dots \}$$



Example

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

$\underline{\text{do}}$	$\underline{\text{true}} \rightarrow \underline{\text{if}}$ $s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0;$ $\parallel s_1 = t \wedge x = 1 \rightarrow \text{skip};$ $\parallel s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1;$ $\underline{\text{fi}}$	\parallel	$\underline{\text{true}} \rightarrow \underline{\text{if}}$ $s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1;$ $\parallel s_2 = t \wedge x = 0 \rightarrow \text{skip};$ $\parallel s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0;$ $\underline{\text{fi}}$	$\underline{\text{od}}$
-------------------------	---	-------------	---	-------------------------

Example (Cont'd)

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

<u>do</u>	$true \rightarrow$ <u>if</u> $s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0;$ $\parallel s_1 = t \wedge x = 1 \rightarrow skip;$ $\parallel s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1;$ <u>fi</u>		$true \rightarrow$ <u>if</u> $s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1;$ $\parallel s_2 = t \wedge x = 0 \rightarrow skip;$ $\parallel s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0;$ <u>fi</u>	<u>od</u>
-----------	---	--	---	-----------

$\langle t, t, 0 \rangle$

Example (Cont'd)

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

<u>do</u>	$true \rightarrow$ <u>if</u> $s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0;$ $\square s_1 = t \wedge x = 1 \rightarrow skip;$ $\square s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1;$ <u>fi</u>		$true \rightarrow$ <u>if</u> $s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1;$ $\square s_2 = t \wedge x = 0 \rightarrow skip;$ $\square s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0;$ <u>fi</u>	<u>od</u>
-----------	---	--	---	-----------

$\langle t, t, 0 \rangle$

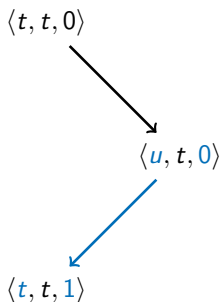


$\langle u, t, 0 \rangle$

Example (Cont'd)

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

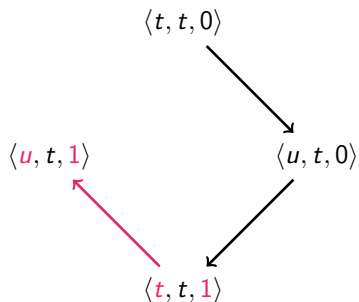
<u>do</u>	$true \rightarrow$ <u>if</u> $s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0;$ $\parallel s_1 = t \wedge x = 1 \rightarrow skip;$ $\parallel s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1;$ <u>fi</u>		$true \rightarrow$ <u>if</u> $s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1;$ $\parallel s_2 = t \wedge x = 0 \rightarrow skip;$ $\parallel s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0;$ <u>fi</u>	<u>od</u>
-----------	---	--	---	-----------



Example (Cont'd)

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

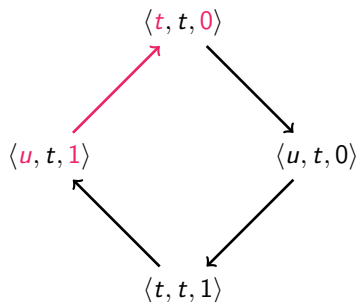
<u>do</u>	$true \rightarrow$ <u>if</u> $s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0;$ $\parallel s_1 = t \wedge x = 1 \rightarrow skip;$ $\parallel s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1;$ <u>fi</u>		$true \rightarrow$ <u>if</u> $s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1;$ $\parallel s_2 = t \wedge x = 0 \rightarrow skip;$ $\parallel s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0;$ <u>fi</u>	<u>od</u>
-----------	---	--	---	-----------



Example (Cont'd)

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

<u>do</u>	$true \rightarrow$ <u>if</u> $s_1 = t \wedge x = 0 \rightarrow s_1 := u; x := 0;$ $\square s_1 = t \wedge x = 1 \rightarrow skip;$ $\square s_1 = u \wedge x = 0 \rightarrow s_1 := t; x := 1;$ <u>fi</u>		$true \rightarrow$ <u>if</u> $s_2 = t \wedge x = 1 \rightarrow s_2 := u; x := 1;$ $\square s_2 = t \wedge x = 0 \rightarrow skip;$ $\square s_2 = u \wedge x = 1 \rightarrow s_2 := t; x := 0;$ <u>fi</u>	<u>od</u>
-----------	---	--	---	-----------



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: $\varphi ::= b \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid EX\varphi \mid EG\varphi \mid E[\varphi_1 U \varphi_2]$
 $AG\varphi \equiv \neg EF\neg\varphi$, etc.

- ▶ Semantics:
$$\left. \begin{array}{l} \text{Kripke structure } \mathcal{K} \\ \text{state } s \\ \text{CTL formula } \varphi \end{array} \right\} \mathcal{K}, s \models \varphi$$

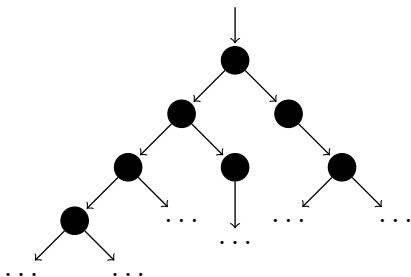
recursively defined as follows:

- $\mathcal{K}, s \models b$ 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 $\mathcal{K}, s \models \varphi_2$
- $\mathcal{K}, s \models EX\varphi$ iff there exists $\langle s, t \rangle \in R$ such that $\mathcal{K}, t \models \varphi$
- $\mathcal{K}, s \models E[\varphi_1 U \varphi_2]$ iff there exists a path $\pi = \langle s, s_1, \dots \rangle$ in \mathcal{K} and $i \geq 0$ such that $\mathcal{K}, \pi_i \models \varphi_2$ and for all $0 \leq j < i$, $\mathcal{K}, \pi_j \models \varphi_1$
- $\mathcal{K}, s \models EG\varphi$ iff there exists a path π such that $\pi_0 = s$ and for all $i \geq 0$, $\mathcal{K}, \pi_i \models \varphi$

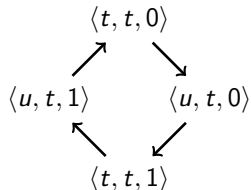
Specification: Behavioural Properties

Computation Tree Logic

AG p



2-Process Concurrent Program for mutual exclusion: $AG \neg(s_1 = u \wedge s_2 = u)$



$\mathcal{K}, \langle t, t, 0 \rangle \models AG \neg(s_1 = u \wedge s_2 = u)$

$\langle t, t, 0 \rangle \rightarrow \langle u, t, 0 \rangle \rightarrow \langle t, t, 1 \rangle \rightarrow \langle u, t, 1 \rangle$

$\langle u, t, 1 \rangle \leftarrow \langle t, t, 1 \rangle \leftarrow \langle u, t, 0 \rangle \leftarrow \langle t, t, 0 \rangle$

\downarrow
...

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. $f : D \rightarrow D$, 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, \dots, P_k in C satisfies T , and
2. C satisfies f

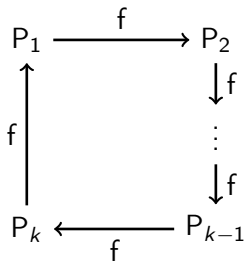
Symmetric Concurrent Programs

k -generating function:

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

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

of order k



$$f^k = id$$

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: $f^2 = id$

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

$\quad \quad \quad \square \quad s_1 = t \wedge x = 1 \rightarrow s_1 := t ; x := 1 ;$

$\quad \quad \quad \square \quad s_1 = u \wedge x = 0 \rightarrow s_1 := t ; x := 1 ; \underline{fi}$

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: $f^2 = id$

$true \rightarrow \underline{if} \ s_1 = t \wedge x = 0 \rightarrow s_1 := u ; x := 0 ;$
 $\swarrow \quad \quad \quad \searrow$
 $f \quad \quad \quad f$
 $true \rightarrow \underline{if} \ s_2 = t \wedge x = 1 \rightarrow s_2 := u ; x := 1 ;$

$\square \ s_1 = t \wedge x = 1 \rightarrow s_1 := t ; x := 1 ;$

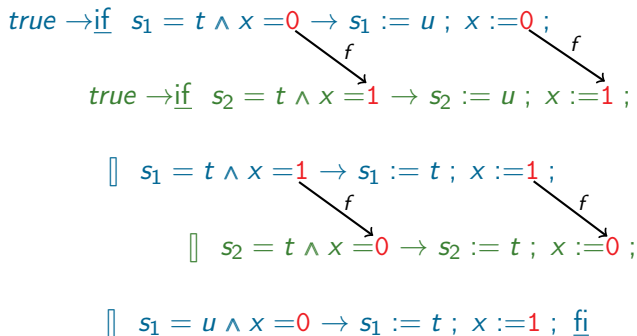
$\square \ s_1 = u \wedge x = 0 \rightarrow s_1 := t ; x := 1 ; \underline{fi}$

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: $f^2 = id$

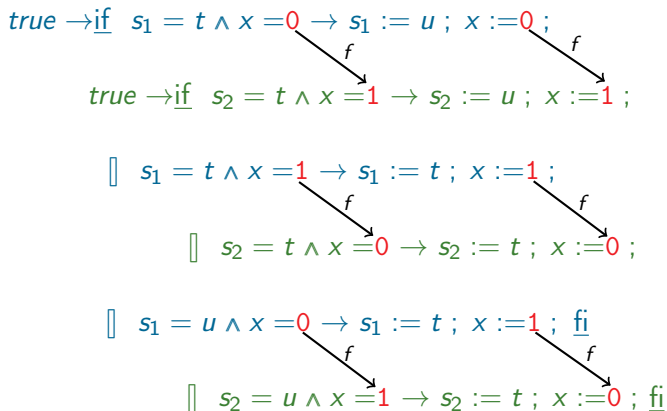


Example

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: $f^2 = id$



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

$$s_1 := l_1; s_2 := l_2; x := d_0;$$
$$\begin{array}{l} \underline{\text{do}} \\ \quad \text{true} \rightarrow \underline{\text{if}} \\ \quad \quad s_1 = ? \wedge x = ? \rightarrow s_1 := ?; x := ?; \\ \quad \quad \dots \\ \quad \quad s_1 = ? \wedge x = ? \rightarrow s_1 := ?; x := ?; \\ \quad \quad \underline{\text{fi}} \\ \underline{\text{od}} \end{array} \left\| \begin{array}{l} \text{true} \rightarrow \underline{\text{if}} \\ \quad s_2 = ? \wedge x = ? \rightarrow s_2 := ?; x := ?; \\ \quad \dots \\ \quad s_2 = ? \wedge x = ? \rightarrow s_2 := ?; x := ?; \\ \quad \underline{\text{fi}} \\ \underline{\text{od}} \end{array} \right.$$

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 P_1, \dots, 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. $\leftarrow \text{notsat}(s_0, \varphi)$
2. $\text{sat}(U, F) \leftarrow \text{elem}(F, U)$
3. $\text{sat}(U, \text{not}(F)) \leftarrow \text{not sat}(U, F)$
4. $\text{sat}(U, \text{and}(F_1, F_2)) \leftarrow \text{sat}(U, F_1) \wedge \text{sat}(U, F_2)$
5. $\text{sat}(U, \text{ex}(F)) \leftarrow \text{tr}(U, V) \wedge \text{sat}(V, F)$
6. $\text{sat}(U, \text{eu}(F_1, F_2)) \leftarrow \text{sat}(U, F_2)$
7. $\text{sat}(U, \text{eu}(F_1, F_2)) \leftarrow \text{sat}(U, F_1) \wedge \text{tr}(U, V) \wedge \text{sat}(V, \text{eu}(F_1, F_2))$
8. $\text{sat}(U, \text{eg}(F)) \leftarrow \text{satpath}(U, V, F) \wedge \text{satpath}(V, V, F)$
9. $\text{satpath}(U, V, F) \leftarrow \text{sat}(U, F) \wedge \text{tr}(U, V) \wedge \text{sat}(V, F)$
10. $\text{satpath}(U, Z, F) \leftarrow \text{sat}(U, F) \wedge \text{tr}(U, V) \wedge \text{satpath}(V, Z, F)$

Logic program based synthesis

Encoding behavioural properties (Cont'd)

$$\begin{aligned} 11.1 \quad & tr(s(S_1, \dots, S_k, X), s(S'_1, \dots, S'_k, X')) \leftarrow \\ & \quad reachable(s(S_1, \dots, S_k, X)) \wedge \\ & \quad gc(1, S_1, X, S'_1, X') \wedge \langle S_1, X \rangle \neq \langle S'_1, X' \rangle \end{aligned}$$

...

$$\begin{aligned} 11.k \quad & tr(s(S_1, \dots, S_k, X), s(S'_1, \dots, S'_k, X')) \leftarrow \\ & \quad reachable(s(S_1, \dots, S_k, X)) \wedge \\ & \quad gc(k, S_k, X, S'_k, X') \wedge \langle S_k, X \rangle \neq \langle S'_k, X' \rangle \end{aligned}$$

$$12. \quad \leftarrow \text{not } out(S) \wedge reachable(S)$$

$$13. \quad out(S) \leftarrow tr(S, Z)$$

$$14. \quad reachable(s_0) \leftarrow$$

$$15. \quad reachable(S) \leftarrow tr(Z, S)$$

Logic program based synthesis

Encoding structural Properties

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

We encode Σ as follows:

- 1.1 $\bigvee_{\langle S', X' \rangle \in \text{Next}(\langle S_1, X \rangle)} gc(1, S_1, X, S', X') \leftarrow \text{reachable}(S_1, \dots, S_k, X)$
- 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 $\leftarrow gc(2, S, X, S', X') \wedge \text{not } ps(2, S, X)$
- 2.3 $ps(2, S_2, X) \leftarrow \text{reachable}(S_1, S_2, \dots, S_k, X)$
- ...
- k.1 $gc(k, S, f(X), S', f(X')) \leftarrow gc(k-1, S, X, S', X')$
- k.2 $\leftarrow gc(k, S, X, S', X') \wedge \text{not } ps(k, S, X)$
- k.3 $ps(k, S_k, X) \leftarrow \text{reachable}(S_1, S_2, \dots, S_k, X)$

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

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,

$$(s_1 := \ell_0; \dots; s_k := \ell_0; x := d_0; \underline{\text{do}} P_1 \parallel \dots \parallel P_k \underline{\text{od}}) \models \varphi$$

iff there exists an answer set M in $\text{ans}(\Pi)$ such that

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

$$(s_i = \ell \wedge x = d \rightarrow s_i := \ell'; x := d') \text{ is in } P_i \text{ iff } M \models \text{gc}(i, \ell, d, \ell', d')$$

Experimental results

Examples

- ▶ *Mutual Exclusion* (ME): it is not the case that process P_i is in its critical section ($s_i = u$), and process P_j is in its critical section ($s_j = u$) at the same time: for all i, j in $\{1, \dots, k\}$, with $i \neq j$,

$$AG \neg (s_i = u \wedge s_j = u)$$

- ▶ *Starvation Freedom* (SF): if a process is waiting to enter the critical section ($s_i = w$), then after a finite amount of time, it will execute its critical section ($s_i = u$): for all i in $\{1, \dots, k\}$,

$$AG (s_i = w \rightarrow AF s_i = u)$$

- ▶ *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 $\{1, \dots, k\}$,

$$AG ((s_i = w \wedge s_j = u) \rightarrow AF (s_j = t \wedge A[\neg(s_j = u) \cup s_i = u]))$$

- ▶ *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 $\{1, \dots, k\}$,

$$AG ((s_i = w \wedge \bigwedge_{j \in \{1, \dots, k\} \setminus \{i\}} s_j = t) \rightarrow EX s_i = u)$$

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

$$f_1 = \{\langle 0, 1 \rangle, \langle 1, 0 \rangle\}$$

$$f_2 = \{\langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 2, 2 \rangle\}$$

$$f_3 = \{\langle 0, 1 \rangle, \langle 1, 2 \rangle, \langle 2, 0 \rangle\}$$

$$T = \{\langle t, w \rangle, \langle w, w \rangle, \langle w, u \rangle, \langle u, t \rangle\}$$