Polymorphic Systems with Arrays: Decidability and Undecidability

Ranko Lazić
University of Warwick, UK

Tom Newcomb        Bill Roscoe
University of Oxford, UK

LSV, ENS Cachan, France
27 April 2004
Context

Model checking infinite-state systems is an active research area:

- decidability,
- complexity,
- abstraction,
- semi-algorithms,
- tools, etc.

Many systems are infinite-state because they have parameters with infinite ranges, e.g. number of parallel components, data type. Seek correctness *for all* instantiations of the parameters.
Classes of infinite-state systems

*Counting abstraction* represents a system with an arbitrary number of identical parallel components as a Petri net [German & Sistla 92].

For more than rendez-vous communications, extensions of Petri nets are used, e.g.:

- for broadcasts, transfer arcs;
- for partially non-blocking rendez-vous, non-blocking arcs.
Some other abstract models related to Petri nets:
  • broadcast protocols [Emerson & Namjoshi 98];
  • multi-set rewriting specifications [Delzanno 02].
Many decidability/undecidability results, e.g.:
  • [Esparza 98]
  • [Esparza, Finkel & Mayr 99]
  • [Raskin & Van Begin 03]
  • [Lomazova & Schnoebelen 00]
  • [Delzanno 02]
UNITY-style syntax

Infinite-state systems often given using state variables, guards and assignments.

For example:

\[\text{cache} : (X \times Y) \rightarrow (Z \times \text{Enum}_3)\]

UNITY-like syntax can succinctly express systems with several parameters. Petri nets and related models either restricted to one or two dimensions (e.g. broadcast protocols, Petri nets with non-blocking arcs), or complex (e.g. nested Petri nets).

Relating the two kinds of systems non-trivial in general.
Organisation

1. Polymorphic systems with arrays: syntax, semantics.

2. Control-state reachability problems.

3. Undecidability results.

4. Decidability result.

Running example: Bully Algorithm [Garcia-Molina 82].
\[ \lambda\text{-calculus: types} \]

\[
B \ ::= \ X \mid B_1 \times \cdots \times B_n \mid B_1 + \cdots + B_{n\geq 1} \\
T \ ::= \ B \mid B \to B'
\]

\[
[X]_{\omega} = \omega[X] \\
[B_1 \times \cdots \times B_n]_{\omega} = [B_1]_{\omega} \times \cdots \times [B_n]_{\omega} \\
[B_1 + \cdots + B_n]_{\omega} = \{1\} \times [B_1]_{\omega} \cup \cdots \cup \{n\} \times [B_n]_{\omega} \\
[B \to B']_{\omega} = ([B']_{\omega})[B]_{\omega}
\]
\[ t ::= x \mid (t_1, \ldots, t_n) \mid \pi_i(t) \mid \\
\nu^B_i(t) \mid \text{case } t \text{ of } x_1.t'_1 \text{ or } \ldots \text{ or } x_n.t'_n \mid \\
\lambda x : B.t \mid t_1[t_2] \]

Well-typed term-in-context:

\[ \Omega, \Gamma \vdash t : T \]
\[ [x]_{\omega, \gamma} = \gamma[x] \]
\[ [(t_1, \ldots, t_n)]_{\omega, \gamma} = ([t_1]_{\omega, \gamma}, \ldots, [t_n]_{\omega, \gamma}) \]
\[ [[\pi_i(t)]]_{\omega, \gamma} = \pi_i([t]_{\omega, \gamma}) \]
\[ [[\nu^B_i(t)]]_{\omega, \gamma} = (i, [t]_{\omega, \gamma}) \]

\[ [\text{case } t \text{ of } x_1.t_1' \text{ or } \ldots \text{ or } x_n.t_n']_{\omega, \gamma} = \left[ t_i' \right]_{\omega, \gamma}\{x_i \mapsto v\}, \text{ where } (i, v) = \left[ t \right]_{\omega, \gamma} \]
\[ [[\lambda x : B \cdot t]]_{\omega, \gamma} = \{v \mapsto [t]_{\omega, \gamma}\{x \mapsto v\} | v \in \left[ B \right]_{\omega}\} \]
\[ [t_1[t_2]]_{\omega, \gamma} = \left[ t_1 \right]_{\omega, \gamma}\left( [t_2]_{\omega, \gamma} \right) \]
\( \lambda \)-calculus: some abbreviations

\[
\begin{align*}
Unit & = \text{empty product} \\
Bool & = Unit + Unit \\
Enum_n & = \underbrace{Unit + \cdots + Unit}_n
\end{align*}
\]
Polymorphic systems with arrays: syntax

A PSA is a 5-tuple \((\Omega, \Gamma, \Theta, R, I)\):

**parameters:** \((\Omega, \Gamma)\) is a signature;

**state variables:** \(\Theta\) is disjoint from \(\Gamma\), and \((\Omega, \Gamma\Theta)\) is a signature;

**instructions:** each \(\rho \in R\) is of the form

\[
\Phi : c \cdot \{x_1 := t_1, \ldots, x_k := t_k\}
\]

**instantiations:** \(I\) is a set of instantiations of \((\Omega, \Gamma)\).
Example array operations

reset:

\[ a := \lambda x : B \cdot t \]

copy:

\[ a := a' \]

map:

\[ a := \lambda x : B \cdot t[(a'_1[x], \ldots, a'_n[x])] \]
multiple partial assign:

\[ a := \lambda x : B \cdot \text{if } d_1 \text{ then } t_1 \text{ elseif } \cdots \text{ d}_n \text{ then } t_n \text{ else } a[x] \]

abbreviated as

\[ a[x : d_1; \cdots; d_n] := t_1; \cdots; t_n \]

write:

\[ a[x : x = t_1; \cdots; x = t_n] := t'_1; \cdots; t'_n \]

abbreviated as

\[ a[t_1; \cdots; t_n] := t'_1; \cdots; t'_n \]
cross-section:

\[ a[x : (\pi_1(x) = t)] := t' \]

choose:

\[ \langle a' : B \rightarrow B' \rangle : true \cdot \{a := a'\} \]
Polymorphic systems with arrays: semantics

The semantics of a PSA \((\Omega, \Gamma, \Theta, R, I)\) is the transition system \((S, \rightarrow)\):

**states:**

\[ S = \{ (\omega, \gamma, \theta) \mid (\omega, \gamma) \in I \land \theta \in []\Theta\omega \} \]

**transitions:** \((\omega, \gamma, \theta) \rightarrow (\omega', \gamma', \theta')\) iff \(\omega' = \omega, \gamma' = \gamma\), and there exists \(\rho \in R\) which can produce \(\theta'\) from \(\theta\).
More precisely, as $\rho$ is of the form

$$\Phi : c \cdot \{x_1 := t_1, \ldots, x_k := t_k\}$$

there exists $\phi \in \llbracket \Phi \rrbracket_\omega$ such that $\llbracket c \rrbracket_{\omega, \gamma \theta \phi} = tt$, and:

- $\theta'[x_i] = \llbracket t_i \rrbracket_{\omega, \gamma \theta \phi}$ for each $i$;

- $\theta'[x'] = \theta[x']$ for all $x' \not\in \{x_1, \ldots, x_k\}$. 
Example: Bully Algorithm

[Garcia-Molina 82]

Leader election in a distributed system.

Process identifiers linearly ordered.
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>→*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Run</td>
<td>Run</td>
<td>Run</td>
<td>Coord</td>
<td>fail (4)</td>
</tr>
<tr>
<td>Run</td>
<td>Run</td>
<td>Run</td>
<td>Run</td>
<td>Fld</td>
<td>tock*</td>
</tr>
<tr>
<td>Elect</td>
<td>Elect</td>
<td>Elect</td>
<td>Fld</td>
<td></td>
<td>signal (2)</td>
</tr>
<tr>
<td>Await</td>
<td>Elect</td>
<td>Elect</td>
<td>Fld</td>
<td></td>
<td>tock* signal (3)</td>
</tr>
<tr>
<td>Await</td>
<td>Await</td>
<td>Elect</td>
<td>Fld</td>
<td></td>
<td>tock* fail (3)</td>
</tr>
<tr>
<td>Await</td>
<td>Await</td>
<td>Fld</td>
<td>Fld</td>
<td></td>
<td>tock*</td>
</tr>
<tr>
<td>Coord</td>
<td>Elect</td>
<td>Fld</td>
<td>Fld</td>
<td></td>
<td>signal (2) tock*</td>
</tr>
<tr>
<td>Await</td>
<td>Coord</td>
<td>Fld</td>
<td>Fld</td>
<td></td>
<td>signal (2)</td>
</tr>
<tr>
<td>Run</td>
<td>Coord</td>
<td>Fld</td>
<td>Fld</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Signature:

\( ([X], \langle \leq_X : X \times X \rightarrow \text{Bool} \rangle) \)

Instantiations:

\( (X \mapsto \widehat{k} = \{1, \ldots, k\}, \leq_X \mapsto \leq_{\widehat{k}}) \)

State variables:

\[
a : X \rightarrow (\{\text{Elect, Coord, Await, Run, Fld}\} \times \\
\{1, \ldots, T_S\} \times \{1, \ldots, \max\{T_E, T_A, T_R\}\})
\]

Some abbreviations: \( a[t].m, a[t].c, a[t].c' \).
Instructions:

tock \langle \rangle : true.
\quad a := \lambda x : X.
\quad \text{if } a[x].m \neq \text{Fld} \land a[x].c = T_S \text{ then } (\text{Fld}, 1, 1)
\quad \text{elseif } a[x].m = \text{Elect} \land a[x].c' = T_E \text{ then } (\text{Coord}, a[x].c + 1, 1)
\quad \text{elseif } a[x].m = \text{Await} \land a[x].c' = T_A \text{ then } (\text{Elect}, a[x].c + 1, 1)
\quad \text{elseif } a[x].m = \text{Run} \land a[x].c' = T_R \text{ then } (\text{Elect}, a[x].c + 1, 1)
\quad \text{elseif } a[x].m \neq \text{Fld} \land a[x].m \neq \text{Coord}
\quad \quad \text{then } (a[x].m, a[x].c + 1, a[x].c' + 1)
\quad \text{elseif } a[x].m = \text{Coord} \text{ then } (a[x].m, a[x].c + 1, a[x].c')
\quad \text{else } a[x]
\textbf{signal} \langle x : X \rangle : a[x].m \neq Fld \\
\hspace{1em} \vdash \lambda x' : X. \: \begin{align*}
&\text{if } x' = x \text{ then } (a[x].m, 1, a[x].c') \\
&\quad \text{elseif } x' < x \land a[x].m \neq \text{Coord} \land \\
&\hspace{1em} a[x'].m \in \{ \text{Elect, Coord} \} \text{ then } (\text{Await}, a[x'].c, 1) \\
&\quad \text{elseif } x' < x \land a[x].m = \text{Coord} \land a[x'].m \neq Fld \\
&\hspace{1em} \text{then } (\text{Run}, a[x'].c, 1) \\
&\quad \text{else } a[x']
\end{align*}

\textbf{fail} \hspace{1em} \langle x : X \rangle : a[x].m \neq Fld \cdot a[x] : = (Fld, 1, 1)

\textbf{revive} \hspace{1em} \langle x : X \rangle : a[x].m = Fld \cdot a[x] : = (\text{Elect}, 1, 1)
Initialised control-state reachability

Suppose we have a PSA \((\Omega, \Gamma, \Theta, R, I)\) with:

- a state variable \(b : \text{Enum}_n\),
- \(i, j \in \{1, \ldots, n\}\), and
- for each array state variable \(a : B \rightarrow B'\), a term \(\Omega, \Gamma \Theta_{bas} \vdash t_a : B'\).

To decide whether there exists a sequence of transitions from a state satisfying

\[ b = e_i \land \bigwedge_{a : B \rightarrow B' \in \Theta} \forall x : B \cdot a[x] = t_a \]

to a state satisfying

\[ b = e_j \]
Uninitialised control-state reachability

Suppose we have a PSA \((\Omega, \Gamma, \Theta, R, I)\) with:

- a state variable \(b : Enum_n\), and
- \(i, j \in \{1, \ldots, n\}\).

To decide whether there exists a sequence of transitions from a state satisfying

\[ b = e_i \]

to a state satisfying

\[ b = e_j \]
Example: Bully Algorithm

- There are never two distinct processes in Coord mode.

We add a state variable $b : \{0, 1\}$, and an instruction

$$\langle x : X, x' : X \rangle : x \neq x' \land a[x].m = \text{Coord} \land a[x'].m = \text{Coord} \cdot b := 1$$

The check is whether, from a state in which

$$b = 0 \land \forall x : X \cdot a[x] = (\text{Elect}, 1, 1)$$

the system can reach a state in which

$$b = 1$$
A process cannot continuously be Run since receiving a signal from a Coord until receiving a signal from a Coord whose identifier is smaller than that of the previous one.

There is never a Coord process and a Run process with a greater identifier.

We add a state variable $b : \{0, 1\}$, and an instruction

$$\langle x : X, x' : X \rangle : x < x' \land a[x].m = \text{Coord} \land a[x'].m = \text{Run} \cdot b := 1$$

The check is as in the first example.
Theorem 1

Initialised CSR is undecidable for each of the following classes of PSAs:

$X \times X$-to-$\text{Bool}$  \bullet  signature is ($\{X\}, \langle \equiv_X : X \times X \rightarrow \text{Bool} \rangle$);

\bullet  only one array state variable, of type $X \times X \rightarrow \text{Bool}$;

\bullet  no instruction parameters which are arrays, and each array assignment is a write;

\bullet  instantiations are ($X \mapsto \widehat{k}, =_X \mapsto =_{\widehat{k}}$).
$X \times Y$-to-$\text{Bool}$  • signature is

$\langle \{X, Y\}, \langle \equiv_X : X \times X \to \text{Bool}, \equiv_Y : Y \times Y \to \text{Bool} \rangle \rangle$

• only one array state variable, of type $X \times Y \to \text{Bool}$;

• no instruction parameters which are arrays, and each array assignment is a write;

• instantiations are

$$
(X \mapsto \hat{k}, Y \mapsto \hat{l},
\equiv_X \mapsto \equiv_{\hat{k}}, \equiv_Y \mapsto \equiv_{\hat{l}})
$$
\(X\to Y, Z\) • signature is

\(\langle \equiv_X: X \times X \to \text{Bool}, \equiv_Y: Y \times Y \to \text{Bool}, \equiv_Z: Z \times Z \to \text{Bool} \rangle\)

• only two array state variables, of types \(X \to Y\) and \(X \to Z\);

• no instruction parameters which are arrays, and each array assignment is a write;

• instantiations are

\(\langle X \mapsto \hat{k}, Y \mapsto \hat{l}, Z \mapsto \hat{m}, \equiv_X \mapsto \equiv_{\hat{k}}, \equiv_Y \mapsto \equiv_{\hat{l}}, \equiv_Z \mapsto \equiv_{\hat{m}} \rangle\)
$X, \leq_{to} Y$  • signature is

\[
\{ X, Y \}, \langle \leq_X: X \times X \to \text{Bool}, =_Y: Y \times Y \to \text{Bool} \rangle
\]

• only one array state variable, of type $X \to Y$;

• no instruction parameters which are arrays, and each array assignment is a write;

• instantiations are

\[
(X \mapsto \hat{k}, Y \mapsto \hat{l}, \\
\leq_X \mapsto \leq_{\hat{k}}, =_Y \mapsto =_{\hat{l}})
\]
Corollary

For classes of PSAs obtained by extending the classes above to allow resets of arrays, uninitialised CSR is undecidable.
Proof of Theorem 1

By reducing from location reachability for two-counter machines: given a 2CM and a location $L_j$, to decide whether a configuration with location $L_j$ is reachable from $(L_1, 0, 0)$. 
$X \times X$-to-Bool

\[ c_1 = 3 \]
\[ c_2 = 1 \]

<table>
<thead>
<tr>
<th></th>
<th>$x_1'$</th>
<th>$x_2'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$t$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$t$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$t$</td>
</tr>
<tr>
<td>$x_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
$X \times Y \xrightarrow{\text{to-Bool}}$

\[
c_1 = 2 \\
c_2 = 1
\]

<table>
<thead>
<tr>
<th></th>
<th>$y_1$</th>
<th>$y'_1$</th>
<th>$y_2$</th>
<th>$y'_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$t$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td></td>
<td>$t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t$</td>
<td>$t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x'_1$</td>
<td>$t$</td>
<td>$t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x'_2$</td>
<td></td>
<td></td>
<td>$t$</td>
<td>$t$</td>
</tr>
</tbody>
</table>
**X-to-Y, Z**

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x'_1$</th>
<th>$x'_2$</th>
<th>$x_2$</th>
<th>$y_1^1$</th>
<th>$y_1^2$</th>
<th>$y_2^1$</th>
<th>$y_2^2$</th>
<th>$z_1^1$</th>
<th>$z_1^2$</th>
<th>$z_2^1$</th>
<th>$z_2^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1^1$</td>
<td>$y_2^1$</td>
<td>$y_1^2$</td>
<td>$y_2^2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z_1^1$</td>
<td>$z_2^1$</td>
<td>$z_1^2$</td>
<td>$z_2^2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ c_1 = 2 \]
\[ c_2 = 1 \]
\[ x, \leq \text{to} - Y \]

\[
\begin{align*}
c_1 &= 3 \\
c_2 &= 1
\end{align*}
\]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( y_1 )</th>
<th>( y_1 )</th>
<th>( y_1 )</th>
<th>( y_1 )</th>
<th>( y_1 )</th>
<th>( x'_1 )</th>
<th>( x''_1 )</th>
<th>( x_2 )</th>
<th>( x'_2 )</th>
<th>( x''_2 )</th>
<th>( y_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_1 )</td>
<td>( y_2 )</td>
<td>( y_1 )</td>
<td>( y_3 )</td>
<td>( y_2 )</td>
<td>( y_4 )</td>
<td>( y_1 )</td>
<td>( y_1 )</td>
<td>( y_2 )</td>
<td>( y_2 )</td>
<td>( y_1 )</td>
<td></td>
</tr>
</tbody>
</table>
Theorem 2

Initialised and uninitialised CSR problems are decidable for the class \(X,\leq\)-to-Enum of PSAs:

- signature is \((\{X\}, \langle \leq_X: X \times X \to \text{Bool} \rangle)\);

- the type of any array state variable, and of any array instruction parameter, is of the form \(X \to \text{Enum}_m\);

- instantiations are \((X \mapsto \tilde{k}, \leq_X \mapsto \leq_{\tilde{k}})\).
Proof of Theorem 2

By reducing to whether a monadic MSR(NC) specification can reach the upward closure of a finite set of constrained configurations.

The latter problem is decidable, and a decision procedure has been implemented [Delzanno 02].
We have an instance of the initialised or uninitialised CSR problem, which is for a PSA \((\Omega, \Gamma, \Theta, R, I)\) in the class \(X, \leq\)-to-Enum.

By properties of the \(\lambda\)-calculus, we can assume:

- \(\Theta\) is of the form
  \[
  \langle b : \text{Enum}_n, x_1 : X, \ldots, x_l : X, a : X \rightarrow \text{Enum}_m \rangle
  \]

- the parameters of any \(\rho \in R\) are of the form
  \[
  \langle x_{l+1} : X, \ldots, x_{l+l'} : X, a' : X \rightarrow \text{Enum}_{m'} \rangle
  \]
We construct a monadic MSR(NC) specification \((\mathcal{P}, \text{NC}, \mathcal{I}, \mathcal{R})\). Let \(\mathcal{P}\) consist of:

- nullary predicate symbols \(z, nz, b_1, \ldots, b_n\);

- unary predicate symbols \(x_1, \ldots, x_i\);

- unary predicate symbols \(a_{i,j}'\) for \(i \in \{1, \ldots, m\}, j \in \{0, 1, \ldots, m'\}\).

\(\text{NC}\) is the system of name constraints:

\[
\varphi ::= \text{false} \mid \text{true} \mid x = x' \mid x < x' \mid \varphi \land \varphi'
\]

\(\text{NC}\) constraints are interpreted over the integers \(\mathbb{Z}\).
For any state \((\omega, \gamma, \theta)\) of the PSA \((\Omega, \Gamma, \Theta, R, I)\), where \(\omega = \{X \mapsto \hat{k}\}\) and \(\gamma = \{\leq X \mapsto \leq \hat{k}\}\), let
\[
F(\omega, \gamma, \theta) = z \mid b_{\theta[b]} \mid x_1(\theta[x_1]) \mid \cdots \mid x_l(\theta[x_l]) \mid aa'_{\theta[a](1), 0(1)} \mid \cdots \mid aa'_{\theta[a](k), 0(k)}
\]

The MSR(NC) specification \((\mathcal{P}, \text{NC}, \mathcal{I}, \mathcal{R})\) can reach a configuration \(\mathcal{M}\) with \(z \in \mathcal{M}\) from \(F(\omega, \gamma, \theta)\) if and only if \(\mathcal{M} = F(\omega, \gamma, \theta')\) for some state \((\omega, \gamma, \theta')\) reachable from \((\omega, \gamma, \theta)\).