UNIVERSITY OF TWENTE.

Formal Methods & Tools.

SCOOP:
A Tool for Symbolic Optimisations
Of Probabilistic Processes

Mark Timmer
September 7, 2011

QEST 2011
Probabilistic model checking:

- Verifying *quantitative properties*,
- Using a *probabilistic* model (e.g., a probabilistic automaton)
Probabilistic model checking:

- Verifying quantitative properties,
- Using a probabilistic model (e.g., a probabilistic automaton)

Non-deterministically choose one of the three transitions

Probabilistically choose the next state
Probabilistic model checking:

- Verifying quantitative properties,
- Using a probabilistic model (e.g., a probabilistic automaton)

Non-deterministically choose one of the three transitions

Probabilistically choose the next state

Limitations of previous approaches:

- Restricted treatment of data
- Susceptible to the state space explosion problem
Overview of our approach

- Probabilistic specification
- Instantiation
- State space (PA)
- Visualisation
- Model checking
- Minimisation
Overview of our approach

- **Probabilistic specification**
  - prCRL
  - Instantiation
  - State space (PA)
    - Visualisation
    - Model checking
    - Minimisation
Overview of our approach

- prCRL
- Probabilistic specification
- Linearisation
- LPPE
- Intermediate format
- Instantiation
- State space (PA)
- Minimisation
- Visualisation
- Model checking
Overview of our approach

Probabilistic specification → Linearisation

Intermediate format

Symbolic optimisations
- Dead variable reduction
- Constant elimination
- Summation elimination
- Expression simplification

Confluence reduction
Instantiation

State space (PA)

Minimisation

Visualisation

Model checking
The input language: prCRL

Specification language prCRL:
- Based on $\mu$CRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata
The input language: prCRL

Specification language prCRL:
- Based on $\mu$CRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata

**Basic grammar**

\[
p ::= Y(t) \mid c \Rightarrow p \mid p + p \mid \sum_{x:D} p \mid a(t)\sum_{x:D} f : p
\]
The input language: prCRL

Specification language prCRL:
- Based on $\mu$CRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata

Basic grammar

$$p ::= Y(t) \mid c \Rightarrow p \mid p + p \mid \sum_{x:D} p \mid a(t)\sum_{x:D} f : p$$

$$X = \sum_{n:\{1,2,3\}} \text{write}(n) \sum_{i:\{1,2\}} \frac{i}{3} : (i = 1 \Rightarrow X + i = 2 \Rightarrow \text{beep} \cdot X)$$
The input language: prCRL

Specification language prCRL:
- Based on $\mu$CRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata

**Basic grammar**

\[
p ::= Y(t) \mid c \Rightarrow p \mid p + p \mid \sum_{x:D} p \mid a(t)\sum_{x:D} f : p
\]

\[
X = \sum_{n:\{1,2,3\}} \sum_{i:\{1\}} \frac{i}{3} : (i = 1 \Rightarrow X + i = 2 \Rightarrow \text{beep} \cdot X)
\]

For composability we introduced extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.
A linear format for prCRL: the LPPE

**LPPEs** are a subset of prCRL specifications:

\[
X(g : G) = \sum_{d_1 : D_1} c_1 \Rightarrow a_1 \sum_{e_1 : E_1} f_1 : X(n_1) \\
\ldots \\
+ \sum_{d_k : D_k} c_k \Rightarrow a_k \sum_{e_k : E_k} f_k : X(n_k)
\]

Advantages of using LPPEs instead of prCRL specifications:
- Easy state space generation
- Straight-forward parallel composition
- Symbolic optimisations enabled at the language level

Theorem
Every specification (without unguarded recursion) can be linearised to an LPPE, preserving strong probabilistic bisimulation.
A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

\[
X(g : G) = \sum_{d_1 : D_1} c_1 \Rightarrow a_1 \sum_{e_1 : E_1} f_1 : X(n_1) \\
\ldots \\
+ \sum_{d_k : D_k} c_k \Rightarrow a_k \sum_{e_k : E_k} f_k : X(n_k)
\]

Advantages of using LPPEs instead of prCRL specifications:

- Easy state space generation
- Straight-forward parallel composition
- **Symbolic optimisations** enabled at the language level
A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

\[
X(g : G) = \sum c_1 \Rightarrow a_1 \sum f_1 : X(n_1) \\
\ldots \\
+ \sum c_k \Rightarrow a_k \sum f_k : X(n_k)
\]

Advantages of using LPPEs instead of prCRL specifications:

- Easy state space generation
- Straight-forward parallel composition
- Symbolic optimisations enabled at the language level

Theorem

Every specification (without unguarded recursion) can be linearised to an LPPE, preserving strong probabilistic bisimulation.
Optimisation techniques

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification
Optimisation techniques

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification

2. State space reduction techniques
   - Dead variable reduction
   - Confluence reduction

\[ X(id) = \text{print}(id) \cdot X(id) \]

\[ \text{init} X(Mark) \rightarrow \]
Optimisation techniques

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification

2. State space reduction techniques
   - Dead variable reduction
   - Confluence reduction

\[ X(id : Id) = \text{print}(id) \cdot X(id) \]
\[ \text{init } X(Mark) \]

\[ \rightarrow \]
\[ X = \text{print}(Mark) \cdot X \]
\[ \text{init } X \]
Optimisation techniques

1. **LPPE simplification techniques**
   - Constant elimination
   - Summation elimination
   - Expression simplification

2. **State space reduction techniques**
   - Dead variable reduction
   - Confluence reduction

\[ X = \sum_{d:\{1,2,3\}} \text{send}(d) \cdot X \]

\[ \text{init } X \]

\[ \Rightarrow \]

\[ X = \text{send}(2) \cdot X \]

\[ \text{init } X \]
Optimisation techniques

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification

2. State space reduction techniques
   - Dead variable reduction
   - Confluence reduction

\[ X = (3 = 1 + 2 \lor x > 5) \Rightarrow \text{beep} \cdot Y \]

\[ \rightarrow \]

\[ X = \text{beep} \cdot Y \]
Optimisation techniques

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification

2. State space reduction techniques
   - Dead variable reduction
   - Confluence reduction

- Deduce the control flow of an LPPE
- Examine relevance (liveness) of variables
- Reset dead variables
Optimisation techniques

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification

2. State space reduction techniques
   - Dead variable reduction
   - Confluence reduction

- Detect **confluent** internal transitions
- Give these transitions **priority**
SCOOP: Symbolic Optimisations Of Probabilistic Processes

prCRL $\xrightarrow{\text{linearise}}$ LPPE $\xrightarrow{\text{optimise}}$ LPPE $\xrightarrow{\text{instantiate}}$ AUT/PRISM

- Open source, publicly available (6640 lines of Haskell code)
- Stand-alone downloadable tool and web-based interface

<table>
<thead>
<tr>
<th>Spec.</th>
<th>States Before</th>
<th>States After</th>
</tr>
</thead>
<tbody>
<tr>
<td>leader-3-15</td>
<td>1,043,635</td>
<td>251,226,313</td>
</tr>
<tr>
<td>leader-3-18</td>
<td>2,028,181</td>
<td>428,940,116</td>
</tr>
<tr>
<td>leader-3-21</td>
<td>out of mem.</td>
<td>187,972,675</td>
</tr>
<tr>
<td>leader-3-27</td>
<td>out of mem.</td>
<td>398,170,1</td>
</tr>
<tr>
<td>leader-4-5</td>
<td>759,952</td>
<td>61,920,300</td>
</tr>
<tr>
<td>leader-4-6</td>
<td>1,648,975</td>
<td>127,579,608</td>
</tr>
<tr>
<td>leader-4-7</td>
<td>out of mem.</td>
<td>235,310,1</td>
</tr>
<tr>
<td>leader-4-8</td>
<td>out of mem.</td>
<td>400,125</td>
</tr>
<tr>
<td>leader-5-2</td>
<td>260,994</td>
<td>14,978,978</td>
</tr>
<tr>
<td>leader-5-3</td>
<td>out of mem.</td>
<td>112,559,694</td>
</tr>
</tbody>
</table>

UNIVERSITY OF TWENTE.
SCOOP: Symbolic Optimisations Of Probabilistic Processes

- Open source, publicly available (6640 lines of Haskell code)
- Stand-alone downloadable tool and web-based interface
### SCOOP: Symbolic Optimisations Of Probabilistic Processes

- Open source, publicly available (6640 lines of Haskell code)
- Stand-alone downloadable tool and web-based interface

<table>
<thead>
<tr>
<th>Spec.</th>
<th>Original States</th>
<th>Reduced States</th>
<th>Visited States</th>
<th>Running time (sec) Before</th>
<th>Running time (sec) After</th>
</tr>
</thead>
<tbody>
<tr>
<td>leader-3-15</td>
<td>1,043,635</td>
<td>68,926</td>
<td>251,226</td>
<td>313.35</td>
<td>65.96</td>
</tr>
<tr>
<td>leader-3-18</td>
<td>2,028,181</td>
<td>118,675</td>
<td>428,940</td>
<td>1161.58</td>
<td>124.74</td>
</tr>
<tr>
<td>leader-3-21</td>
<td>out of mem.</td>
<td>187,972</td>
<td>675,225</td>
<td>—</td>
<td>205.90</td>
</tr>
<tr>
<td>leader-3-27</td>
<td>out of mem.</td>
<td>398,170</td>
<td>1,418,220</td>
<td>—</td>
<td>497.94</td>
</tr>
<tr>
<td>leader-4-5</td>
<td>759,952</td>
<td>61,920</td>
<td>300,569</td>
<td>322.62</td>
<td>75.14</td>
</tr>
<tr>
<td>leader-4-6</td>
<td>1,648,975</td>
<td>127,579</td>
<td>608,799</td>
<td>1073.16</td>
<td>155.74</td>
</tr>
<tr>
<td>leader-4-7</td>
<td>out of mem.</td>
<td>235,310</td>
<td>1,108,391</td>
<td>—</td>
<td>291.25</td>
</tr>
<tr>
<td>leader-4-8</td>
<td>out of mem.</td>
<td>400,125</td>
<td>1,865,627</td>
<td>—</td>
<td>1069.56</td>
</tr>
<tr>
<td>leader-5-2</td>
<td>260,994</td>
<td>14,978</td>
<td>97,006</td>
<td>155.37</td>
<td>29.40</td>
</tr>
<tr>
<td>leader-5-3</td>
<td>out of mem.</td>
<td>112,559</td>
<td>694,182</td>
<td>—</td>
<td>213.10</td>
</tr>
</tbody>
</table>
```
type Die = {1..6}
  X = throw.psum(1/2 -> A[] ++ 1/2 -> B[])
  A = throw.psum(
      1/2 -> throw.psum(1/2 -> Z[1] ++ 1/2 -> Z[2])
      ++ 1/2 -> throw.psum(1/2 -> Z[3] ++ 1/2 -> A[]))
  B = throw.psum(
      1/2 -> throw.psum(1/2 -> Z[4] ++ 1/2 -> Z[5])
      ++ 1/2 -> throw.psum(1/2 -> Z[6] ++ 1/2 -> B[]))
Z(j;Die) = choose(j).Z[j]
init X
```

Constants (name = value):
- +

- Show LPPE (☐ use prCRL syntax)
- Translate specification to PRISM input (☐ specialise to a given PCTL until formula)
- Show statespace in AUT format (☐ omit probabilities, ☐ convert for use with CADP)
- Show statespace as a PRISM transition matrix
- Show statespace as the actual states and transitions
- Show the number of states and transitions of the state space
- Show verbose output
- Apply unused variable reduction
- Apply dead variable reduction
- Apply confluence reduction (☐ show the number of visited states and transitions, ☐ use stronger heuristics, ☐ remove confluent cycles)

Show Result  Visualize Statespace (from AUT) as image  Visualize Statespace (from AUT) on html canvas
Model: Knuth's die-simulating coin
Questions?

For more information, and the tool itself, go to

http://wwwhome.cs.utwente.nl/~timmer/scoop/