UNIVERSITY OF TWENTE.

Formal Methods & Tools.

SCOOP: A Tool for SymboliC Optimisations Of Probabilistic Processes

Mark Timmer

September 7, 2011

QEST 2011

The context – probabilistic model checking

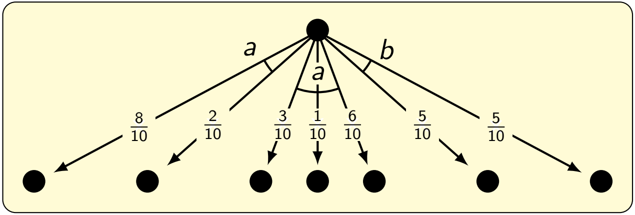
Probabilistic model checking:

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

The context – probabilistic model checking

Probabilistic model checking:

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

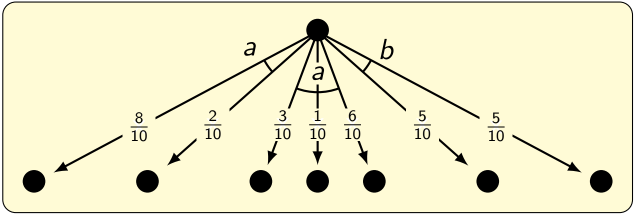


- Non-deterministically choose one of the three transitions
- Probabilistically choose the next state

The context – probabilistic model checking

Probabilistic model checking:

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

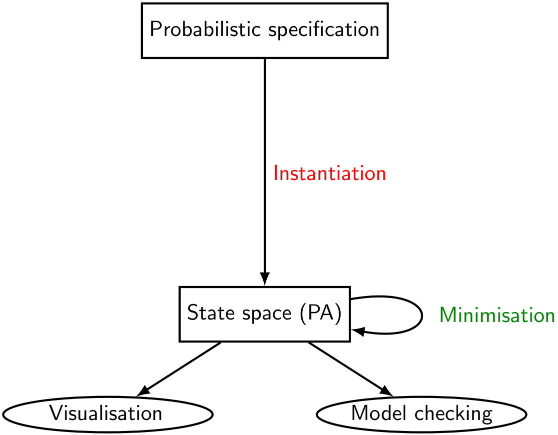


- Non-deterministically choose one of the three transitions
- Probabilistically choose the next state

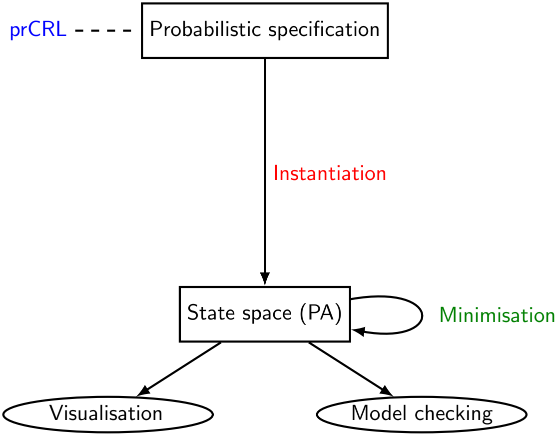
Limitations of previous approaches:

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

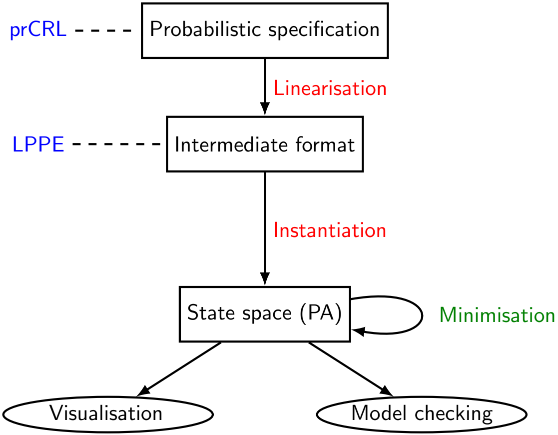
Overview of our approach



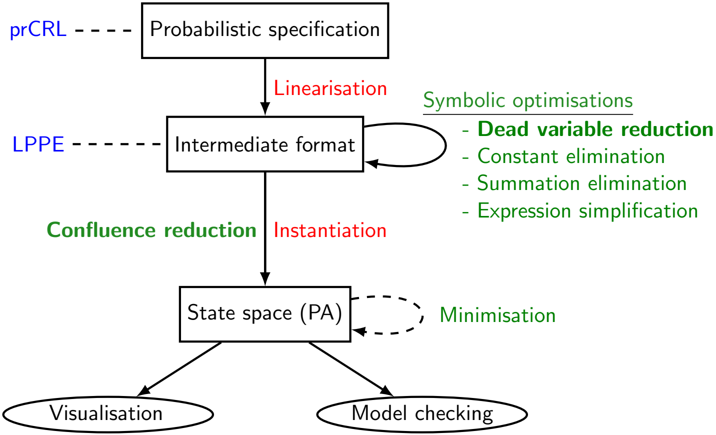
Overview of our approach



Overview of our approach



Overview of our approach



The input language: prCRL

Specification language prCRL:

- Based on μCRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata

The input language: prCRL

Specification language prCRL:

- Based on μCRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata

Basic grammar

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

The input language: prCRL

Specification language prCRL:

- Based on μCRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata

Basic grammar

$$p ::= Y(\boldsymbol{t}) \mid c \Rightarrow p \mid p + p \mid \sum_{\boldsymbol{x}:\boldsymbol{D}} p \mid a(\boldsymbol{t}) \sum\nolimits^{\bullet}_{\boldsymbol{x}:\boldsymbol{D}} f : p$$

$$X = \sum_{n:\{1,2,3\}} \mathrm{write}(n) \sum\nolimits^{\bullet}_{i:\{1,2\}} \frac{i}{3} : (i = 1 \Rightarrow X + i = 2 \Rightarrow \mathrm{beep} \cdot X)$$

The input language: prCRL

Specification language prCRL:

- Based on μCRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata

Basic grammar

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

$$X = \sum_{n:\{1,2,3\}} \mathsf{write}(n) \sum\!\!\!\bullet_{i:\{1,2\}} \tfrac{i}{3} : (i = 1 \Rightarrow X + i = 2 \Rightarrow \mathsf{beep} \cdot X)$$

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:

$$
\begin{aligned}
X(\boldsymbol{g} : \boldsymbol{G}) = & \sum_{\boldsymbol{d_1}:\boldsymbol{D_1}} c_1 \Rightarrow a_1 \sum\!\!\!\!\bullet_{\boldsymbol{e_1}:\boldsymbol{E_1}} f_1 : X(\boldsymbol{n_1}) \\
& \dots \\
& + \sum_{\boldsymbol{d_k}:\boldsymbol{D_k}} c_k \Rightarrow a_k \sum\!\!\!\!\bullet_{\boldsymbol{e_k}:\boldsymbol{E_k}} f_k : X(\boldsymbol{n_k})
\end{aligned}
$$

A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

$$
\begin{aligned}
X(\boldsymbol{g} : \boldsymbol{G}) = & \sum_{\boldsymbol{d_1} : \boldsymbol{D_1}} c_1 \Rightarrow a_1 \sum_{\bullet\, \boldsymbol{e_1} : \boldsymbol{E_1}} f_1 : X(\boldsymbol{n_1}) \\
& \dots \\
+ & \sum_{\boldsymbol{d_k} : \boldsymbol{D_k}} c_k \Rightarrow a_k \sum_{\bullet\, \boldsymbol{e_k} : \boldsymbol{E_k}} f_k : X(\boldsymbol{n_k})
\end{aligned}
$$

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:

$$
\begin{aligned}
X(\boldsymbol{g} : \boldsymbol{G}) = & \sum_{\boldsymbol{d_1} : \boldsymbol{D_1}} c_1 \Rightarrow a_1 \sum_{\bullet\, \boldsymbol{e_1} : \boldsymbol{E_1}} f_1 : X(\boldsymbol{n_1}) \\
& \dots \\
+ & \sum_{\boldsymbol{d_k} : \boldsymbol{D_k}} c_k \Rightarrow a_k \sum_{\bullet\, \boldsymbol{e_k} : \boldsymbol{E_k}} f_k : X(\boldsymbol{n_k})
\end{aligned}
$$

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

Optimisation techniques

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification
2. State space reduction techniques
   - Dead variable reduction
   - Confluence reduction

$X(id : Id) = print(id) \cdot X(id)$

init $X(Mark)$

$\rightarrow$

$X = print(Mark) \cdot X$

init X

Optimisation techniques

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification
2. State space reduction techniques
   - Dead variable reduction
   - Confluence reduction

$X = \sum_{d:\{1,2,3\}} d = 2 \Rightarrow send(d) \cdot X$

init X

$\rightarrow$

$X = send(2) \cdot X$

init X

Optimisation techniques

1. LPPE simplification techniques
    - Constant elimination
    - Summation elimination
    - Expression simplification
2. State space reduction techniques
    - Dead variable reduction
    - Confluence reduction

$X = (3 = 1 + 2 \vee x > 5) \Rightarrow \mathit{beep} \cdot Y$ $\rightarrow$ $X = \mathit{beep} \cdot Y$

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

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

SCOOP

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

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

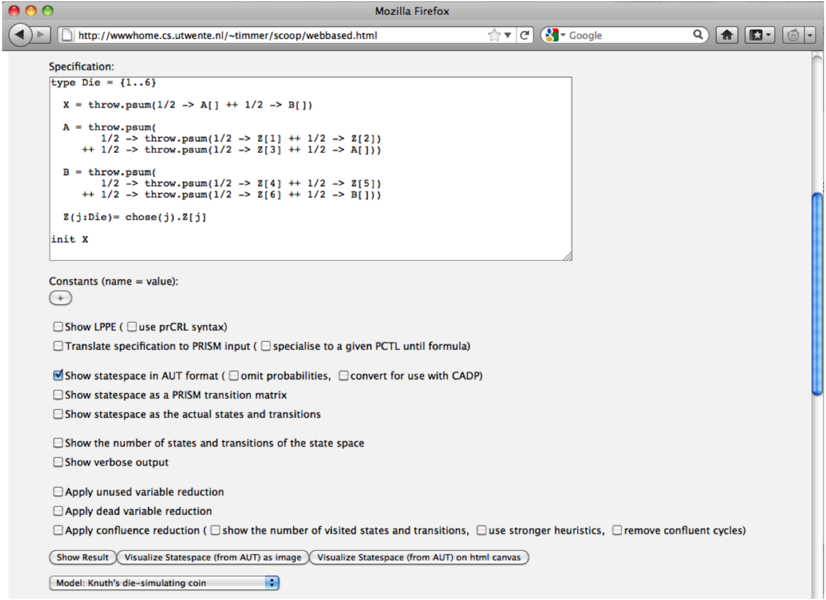
SCOOP

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

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

Spec.	Original States	Reduced States	Visited States	Running time (sec) Before	After
leader-3-15	1,043,635	68,926	251,226	313.35	65.96
leader-3-18	2,028,181	118,675	428,940	1161.58	124.74
leader-3-21	out of mem.	187,972	675,225	–	205.90
leader-3-27	out of mem.	398,170	1,418,220	–	497.94
leader-4-5	759,952	61,920	300,569	322.62	75.14
leader-4-6	1,648,975	127,579	608,799	1073.16	155.74
leader-4-7	out of mem.	235,310	1,108,391	–	291.25
leader-4-8	out of mem,	400,125	1,865,627	–	1069.56
leader-5-2	260,994	14,978	97,006	155.37	29.40
leader-5-3	out of mem.	112,559	694,182	–	213.10

Screenshot


Mozilla Firefox
http://wwwhome.cs.utwente.nl/~timmer/scoop/webbased.html
Google
Specification:
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)= chose(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

Questions?

For more information, and the tool itself, go to

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