Efficient Modelling and Generation of Probabilistic Automata as well as Markov Automata

Mark Timmer
June 29, 2012

Joint work with Joost-Pieter Katoen, Jaco van de Pol, and Mariëlle Stoelinga
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)

Diagram:

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

Limitations of previous approaches:
- Susceptible to the state space explosion problem
- Restricted treatment of data
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:
- Susceptible to the state space explosion problem
- Restricted treatment of data
Combating the state space explosion

1. Specification
2. Instantiation
3. Minimisation (optimisation)
4. State space
5. State space
Combating the state space explosion

- Specification
- Instantiation
- Optimised instantiation
  - Dead variable reduction
  - Confluence reduction
- Minimisation (optimisation)
Overview of our approach

- Probabilistic specification
  - Instantiation
  - Minimisation
  - State space (PA)
    - Visualisation
    - Model checking

Efficient Modelling and Generation of Probabilistic Automata as v
Overview of our approach

Probabilistic specification → Linearisation

Intermediate format

- Linearisation
- Instantiation

State space (PA)

- Optimisation
  - Dead variable reduction
  - Confluence reduction

Visualisation → Minimisation → Model checking
Overview of our approach

- Probabilistic specification
  - prCRL
  - LPPE

- Intermediate format
  - Linearisation
  - Instantiation
  - Optimisation
    - Dead variable reduction
    - Confluence reduction

- State space (PA)
  - Minimisation

- Visualisation
- Model checking
Mimic behaviour with equal probabilities:
Contents

1 Introduction
2 A process algebra with data and probability: prCRL
3 Linearisation: from prCRL to LPPE
4 Reduction techniques
5 Modelling Markov Automata using MAPA
6 Encoding and decoding
7 Reduction techniques revisited
8 Case study
9 Conclusions and Future Work
A process algebra with data and probability: prCRL

Specification language prCRL:
- Based on $\mu$CRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata
- Minimal set of operators to facilitate formal manipulation
- Syntactic sugar easily definable
A process algebra with data and probability: prCRL

Specification language prCRL:
- Based on $\mu$CRL (so data), with additional probabilistic choice
- Semantics defined in terms of probabilistic automata
- Minimal set of operators to facilitate formal manipulation
- Syntactic sugar easily definable

The grammar of prCRL process terms

Process terms in prCRL are obtained by the following grammar:

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

Process equations and processes

A process equation is something of the form $X(g : G) = p$. 
An example specification

Sending an arbitrary natural number

\[ X(\text{active} : \text{Bool}) = \]
\[ \text{not(}\text{active}\text{)} \Rightarrow \text{ping} \cdot \sum_{b: \text{Bool}} X(b) \]
\[ + \text{active} \Rightarrow \tau \sum_{n: \mathbb{N} > 0} \frac{1}{2^n} : \left( \text{send}(n) \cdot X(\text{false}) \right) \]
An example specification

Sending an arbitrary natural number

\[ X(\text{active} : \text{Bool}) = \]
\[ \text{not(}\text{active}\text{)} \Rightarrow \text{ping} \cdot \sum_{b: \text{Bool}} X(b) \]
\[ + \text{active} \Rightarrow \tau \sum_{n: \mathbb{N} > 0} \frac{1}{2^n} : (\text{send}(n) \cdot X(\text{false})) \]
Composability using extended prCRL

For composability we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.
Composability using extended prCRL

For composability we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m')
\]
Composability using extended prCRL

For composability we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m')
\]

\[
Z = (X(1) \parallel Y(2))
\]
Composability using extended prCRL

For composability we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} \cdot X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} \cdot Y(m')
\]

\[
Z = (X(1) \parallel Y(2))
\]

\[
\gamma(\text{choose}, \text{choose}') = \text{chooseTogether}
\]
Composability using extended prCRL

For composability we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[ X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n') \]

\[ Y(m : \{1, 2\}) = \text{write}_Y(m) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m') \]

\[ Z = \partial_{\text{choose}, \text{choose}'}(X(1) \parallel Y(2)) \]

\[ \gamma(\text{choose}, \text{choose}') = \text{chooseTogether} \]
Composability using extended prCRL

For composability we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m')
\]

\[
Z = \partial_{\{\text{choose}, \text{choose}'\}}(X(1) \parallel Y(2))
\]

\[
\gamma(\text{choose}, \text{choose}') = \text{chooseTogether}
\]

\[
\text{write}_X(1) \circlearrowleft Z \circlearrowright \text{write}_Y(2)
\]
Composability using extended prCRL

For composability we introduce extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

\[
X(n : \{1, 2\}) = \text{write}_X(n) \cdot X(n) + \text{choose} \sum_{n' : \{1, 2\}} \frac{1}{2} : X(n')
\]

\[
Y(m : \{1, 2\}) = \text{write}_Y(m) \cdot Y(m) + \text{choose}' \sum_{m' : \{1, 2\}} \frac{1}{2} : Y(m')
\]

\[
Z = \partial_{\{\text{choose, choose}'\}}(X(1) \parallel Y(2))
\]

\[
\gamma(\text{choose}, \text{choose}') = \text{chooseTogether}
\]
LPPEs are a subset of prCRL specifications:

\[
X(g : G) = \sum_{d_1:D_1} c_1 \Rightarrow a_1(b_1) \sum_{e_1:E_1} f_1 : X(n_1) \\
\vdots \\
+ \sum_{d_k:D_k} c_k \Rightarrow a_k(b_k) \sum_{e_k:E_k} f_k : X(n_k)
\]
LPPEs are a subset of prCRL specifications:

$$X(g : G) = \sum_{d_1 : D_1} c_1 \Rightarrow a_1(b_1) \sum_{e_1 : E_1} f_1 : X(n_1)$$

$$+ \sum_{d_k : D_k} c_k \Rightarrow a_k(b_k) \sum_{e_k : E_k} f_k : X(n_k)$$

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:

\[
X(g : G) = \sum_{d_1 : D_1} c_1 \Rightarrow a_1(b_1) \sum_{e_1 : E_1} f_1 : X(n_1) \\
\ldots \\
+ \sum_{d_k : D_k} c_k \Rightarrow a_k(b_k) \sum_{e_k : E_k} f_k : X(n_k)
\]

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.
Linear Probabilistic Process Equations – an example

\[ X(\text{false}) \]

\[ \sum_{b: \text{Bool}} X(b) \]

\[ \text{send}(1) \]

\[ \text{ping} \]

\[ \text{send}(2) \]

\[ \tau \]

\[ 0.5 \]

\[ 0.25 \]

\[ \text{send}(1) \cdot X(\text{false}) \]

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

\[ \ldots \]

**Specification in prCRL**

\[
X(\text{active} : \text{Bool}) =
\]

\[
\text{not} (\text{active}) \Rightarrow \text{ping} \cdot \sum_{b: \text{Bool}} X(b)
\]

\[
+ \text{active} \Rightarrow \tau \sum_{n : \mathbb{N} \geq 0} \frac{1}{2^n} : \text{send}(n) \cdot X(\text{false})
\]
Linear Probabilistic Process Equations – an example

 Specification in prCRL

\[
X(\text{active} : \text{Bool}) = \\
\quad \text{not(}\text{active}\text{)} \Rightarrow \text{ping} \cdot \sum_{b: \text{Bool}} X(b) \\
\quad + \text{active} \Rightarrow \tau \sum_{n: \mathbb{N} \geq 0} \frac{1}{2^n} : \text{send}(n) \cdot X(\text{false})
\]

 Specification in LPPE

\[
X(\text{pc} : \{1..3\}, n : \mathbb{N} \geq 0) = \\
\quad + \text{pc} = 1 \Rightarrow \text{ping} \cdot X(2, 1) \\
\quad + \text{pc} = 2 \Rightarrow \text{ping} \cdot X(2, 1) \\
\quad + \text{pc} = 2 \Rightarrow \tau \sum_{n: \mathbb{N} > 0} \frac{1}{2^n} : X(3, n) \\
\quad + \text{pc} = 3 \Rightarrow \text{send}(n) \cdot X(1, 1)
\]
Linearisation: a simple example without data

Consider the following prCRL specification:

\[ X = a \cdot b \cdot c \cdot X \]
Consider the following prCRL specification:

\[ X = a \cdot b \cdot c \cdot X \]

The control flow of \( X \) is given by:

![Diagram showing the control flow of \( X \)]
Linearisation: a simple example without data

Consider the following prCRL specification:

\[ X = a \cdot b \cdot c \cdot X \]

The control flow of \( X \) is given by:

![Control flow diagram](image-url)
Linearisation: a simple example without data

Consider the following prCRL specification:

\[ X = a \cdot b \cdot c \cdot X \]

The control flow of \( X \) is given by:

The corresponding LPPE (initialised with \( pc = 1 \)):

\[
Y(pc: \{1, 2, 3\}) =
\begin{align*}
pc = 1 & \Rightarrow a \cdot Y(2) \\
+ pc = 2 & \Rightarrow b \cdot Y(3) \\
+ pc = 3 & \Rightarrow c \cdot Y(1)
\end{align*}
\]
Linearisation: a more complicated example with data

Consider the following prCRL specification:

\[ X = \sum_{d \in D} \text{get}(d) \cdot (\tau \cdot \text{loss} \cdot X + \tau \cdot \text{send}(d) \cdot X) \]
Consider the following prCRL specification:

\[ X = \sum_{d:D} \text{get}(d) \cdot (\tau \cdot \text{loss} \cdot X + \tau \cdot \text{send}(d) \cdot X) \]

Control flow:
Consider the following prCRL specification:

\[ X = \sum_{d:D} \text{get}(d) \cdot (\tau \cdot \text{loss} \cdot X + \tau \cdot \text{send}(d) \cdot X) \]

Control flow:
Linearisation: a more complicated example with data

Consider the following prCRL specification:

\[ X = \sum_{d:D} \text{get}(d) \cdot (\tau \cdot \text{loss} \cdot X + \tau \cdot \text{send}(d) \cdot X) \]

**Control flow:**

**LPPE:**

\[
Y(pc: \{1, 2, 3, 4\}, x: D) = \sum_{d:D} 
\begin{align*}
pc = 1 & \Rightarrow \text{get}(d) \cdot Y(2, d) \\
+ & \quad pc = 2 \Rightarrow \tau \cdot Y(3, x) \\
+ & \quad pc = 2 \Rightarrow \tau \cdot Y(4, x) \\
+ & \quad pc = 3 \Rightarrow \text{loss} \cdot Y(1, x) \\
+ & \quad pc = 4 \Rightarrow \text{send}(x) \cdot Y(1, x)
\end{align*}
\]
Consider the following prCRL specification:

$$X = \sum_{d:D} \text{get}(d) \cdot (\tau \cdot \text{loss} \cdot X + \tau \cdot \text{send}(d) \cdot X)$$

Control flow:

LPPE:

$$Y(pc: \{1, 2, 3, 4\}, x: D) =$$

$$\sum_{d:D} pc = 1 \Rightarrow \text{get}(d) \cdot Y(2, d)$$

$$+ \quad pc = 2 \Rightarrow \tau \cdot Y(3, x)$$

$$+ \quad pc = 2 \Rightarrow \tau \cdot Y(4, x)$$

$$+ \quad pc = 3 \Rightarrow \text{loss} \cdot Y(1, x)$$

$$+ \quad pc = 4 \Rightarrow \text{send}(x) \cdot Y(1, x)$$

Initial process: $Y(1, d_1)$. 

---

**Efficient Modelling and Generation of Probabilistic Automata as well as Markov Automata**
Linearisation: a more algorithmic approach

Consider the following prCRL specification:

\[ X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right) \]
Consider the following prCRL specification:

\[
X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\]

1. \[
X_1(d : D, e : D, f : D) = \\
\sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\]
Linearisation: a more algorithmic approach

Consider the following prCRL specification:

$$X(d : D) = \sum_{e : D} a(d+e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e+f) \cdot X(5) \right)$$

1. $$X_1(d : D, e : D, f : D) = \sum_{e : D} a(d+e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e+f) \cdot X(5) \right)$$
Consider the following prCRL specification:

\[
X(d : D) = \sum_{e: D} a(d+e) \sum_{f: D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e+f) \cdot X(5) \right)
\]

1. \(X_1(d : D, e : D, f : D) = \sum_{e: D} a(d+e) \sum_{f: D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e+f) \cdot X(5) \right)\)

2. \(X_1(d : D, e : D, f : D) = \sum_{e: D} a(d+e) \sum_{f: D} \frac{1}{|D|} : X_2(d, e, f)\)
Consider the following prCRL specification:

\[ X(d : D) = \sum_{e:D} a(d + e) \sum_{f:D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right) \]

\[
\begin{align*}
X_1(d : D, e : D, f : D) &= \sum_{e:D} a(d + e) \sum_{f:D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right) \\
X_2(d : D, e : D, f : D) &= \sum_{e:D} a(d + e) \sum_{f:D} \frac{1}{|D|} : X_2(d, e, f) \\
&= c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5)
\end{align*}
\]
Linearisation: a more algorithmic approach

Consider the following prCRL specification:

\[
X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\]

\[
\begin{align*}
1 & \quad X_1(d : D, e : D, f : D) = \\
& \quad \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\end{align*}
\]

\[
\begin{align*}
2 & \quad X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f) \\
& \quad X_2(d : D, e : D, f : D) = c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5)
\end{align*}
\]
Consider the following prCRL specification:

$$X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : (c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5))$$

1. $$X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : (c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5))$$

2. $$X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f)$$
   $$X_2(d : D, e : D, f : D) = c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5)$$

3. $$X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f)$$
Consider the following prCRL specification:

\[ X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right) \]

1. \( X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right) \)

2. \( X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f) \)
   \( X_2(d : D, e : D, f : D) = c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \)

3. \( X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f) \)
   \( X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f) \)
Linearisation: a more algorithmic approach

Consider the following prCRL specification:

\[ X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right) \]

1. \[ X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right) \]

2. \[ X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f) \]
   \[ X_2(d : D, e : D, f : D) = c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \]

3. \[ X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f) \]
   \[ X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f) \]
   \[ X_3(d : D, e : D, f : D) = c(f) \cdot X(5) \]
Consider the following prCRL specification:

\[
X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} \cdot \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\]

1. \(X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} \cdot (c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5))\)

2. \(X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} \cdot X_2(d, e, f)\)
   \(X_2(d : D, e : D, f : D) = c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5)\)

3. \(X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} \cdot X_2(d, e, f)\)
   \(X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f)\)
   \(X_3(d : D, e : D, f : D) = c(f) \cdot X(5)\)

4. \(X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} \cdot X_2(d, e, f)\)
   \(X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f)\)
Consider the following prCRL specification:

\[ X(d : D) = \sum_{e:D} a(d+e) \sum_{f:D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e+f) \cdot X(5) \right) \]

1. \[ X_1(d : D, e : D, f : D) = \sum_{e:D} a(d+e) \sum_{f:D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e+f) \cdot X(5) \right) \]
2. \[ X_1(d : D, e : D, f : D) = \sum_{e:D} a(d+e) \sum_{f:D} \frac{1}{|D|} : X_2(d, e, f) \]
   \[ X_2(d : D, e : D, f : D) = c(e) \cdot c(f) \cdot X(5) + c(e+f) \cdot X(5) \]
3. \[ X_1(d : D, e : D, f : D) = \sum_{e:D} a(d+e) \sum_{f:D} \frac{1}{|D|} : X_2(d, e, f) \]
   \[ X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e+f) \cdot X_1(5, e, f) \]
   \[ X_3(d : D, e : D, f : D) = c(f) \cdot X(5) \]
4. \[ X_1(d : D, e : D, f : D) = \sum_{e:D} a(d+e) \sum_{f:D} \frac{1}{|D|} : X_2(d, e, f) \]
   \[ X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e+f) \cdot X_1(5, e, f) \]
Consider the following prCRL specification:

\[
X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\]

1. \(X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\]

2. \(X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f)
\)
\(X_2(d : D, e : D, f : D) = c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5)
\]

3. \(X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f)
\)
\(X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f)
\)
\(X_3(d : D, e : D, f : D) = c(f) \cdot X(5)
\]

4. \(X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f) \right)
\)
\(X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f)
\)
\(X_3(d : D, e : D, f : D) = c(f) \cdot X_1(5, e, f)
\)
Linearisation: a more algorithmic approach

Consider the following prCRL specification:

\[
X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} \cdot \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\]

\[
\begin{align*}
4 \quad X_1(d : D, e : D, f : D) &= \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} \cdot X_2(d, e, f) \\
X_2(d : D, e : D, f : D) &= c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f) \\
X_3(d : D, e : D, f : D) &= c(f) \cdot X_1(5, e, f)
\end{align*}
\]
Linearisation: a more algorithmic approach

Consider the following prCRL specification:

\[
X(d : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : \left( c(e) \cdot c(f) \cdot X(5) + c(e + f) \cdot X(5) \right)
\]

4 \quad X_1(d : D, e : D, f : D) = \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X_2(d, e, f)

X_2(d : D, e : D, f : D) = c(e) \cdot X_3(d, e, f) + c(e + f) \cdot X_1(5, e, f)

X_3(d : D, e : D, f : D) = c(f) \cdot X_1(5, e, f)

\[
X(pc : \{1, 2, 3\}, d : D, e : D, f : D) =
\]

\[
pc = 1 \Rightarrow \sum_{e : D} a(d + e) \sum_{f : D} \frac{1}{|D|} : X(2, d, e, f)
\]

\[
+ pc = 2 \Rightarrow c(e) \cdot X(3, d, e, f)
\]

\[
+ pc = 2 \Rightarrow c(e + f) \cdot X(1, 5, e, f)
\]

\[
+ pc = 3 \Rightarrow c(f) \cdot X(1, 5, e, f)
\]
Linearisation

In general, we always linearise in two steps:

1. Transform the specification to intermediate regular form (IRF) (every process is a summation of single-action terms)
2. Merge all processes into one big process by introducing a program counter

In the first step, global parameters are introduced to remember the values of bound variables.
Reductions techniques for LPPEs

1. LPPE simplification techniques
   - Constant elimination
   - Summation elimination
   - Expression simplification
Reductions techniques for LPPEs

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

2. State space reduction techniques
   - Dead variable reduction
   - Confluence reduction
Reductions techniques for LPPEs

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

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

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

\[
X = \text{print}(Mark) \cdot X
\]
\[\text{init } X\]
Reductions techniques for LPPEs

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

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

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

\[ \text{init } X \]

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

\[ \text{init } X \]
Reductions techniques for LPPEs

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

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

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

\[ X = \text{beep} \cdot Y \]
Reductions techniques for LPPEs

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
Reductions techniques for LPPEs

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**
What you heard so far

- We developed the process algebra prCRL, incorporating both data and probability;
- We defined a normal form for prCRL, the LPPE; starting point for symbolic optimisations and easy state space generation;
- We provided a linearisation algorithm to transform prCRL specifications to LPPEs, proved it correct and implemented it;
- We developed several reduction techniques for LPPEs that preserve strong/branching probabilistic bisimulation.
Contents

1 Introduction
2 A process algebra with data and probability: prCRL
3 Linearisation: from prCRL to LPPE
4 Reduction techniques
5 Modelling Markov Automata using MAPA
6 Encoding and decoding
7 Reduction techniques revisited
8 Case study
9 Conclusions and Future Work
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

LTSs
DTMCs
CTMCs

Observed limitations:
- No easy process-algebraic modelling language with data
- Susceptible to the state space explosion problem
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Probabilistic Automata (PAs)
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Interactive Markov Chains (IMCs)
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Markov Automata (MAs)
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Markov Automata (MAs)

Observed limitations:
- No easy process-algebraic modelling language with data
- Susceptible to the state space explosion problem
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Markov Automata (MAs)
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Markov Automata (MAs)
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Markov Automata (MAs)
The overall goal: efficient and expressive modelling

Specifying systems with
- Nondeterminism
- Probability
- Stochastic timing

Markov Automata (MAs)

Observed limitations:
- No easy process-algebraic modelling language with data
- Susceptible to the state space explosion problem
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Markov Automata (MAs)
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Stochastic timing

Markov Automata (MAs)

Observed limitations:
- No easy process-algebraic modelling language with data
- Susceptible to the state space explosion problem
The overall goal: efficient and expressive modelling

Specifying systems with

- Nondeterminism
- Probability
- Markov Automata (MAs)
- Stochastic timing

Observed limitations:

- No easy process-algebraic modelling language with data
- Susceptible to the state space explosion problem
Approach: extending and reusing

\[ PA \rightarrow MA \]
Approach: extending and reusing

\[
\begin{align*}
\text{PA} & \rightarrow \text{MA} \\
\text{prCRL} & \rightarrow \text{MAPA} \quad \text{(Markov Automata Process Algebra)}
\end{align*}
\]
Approach: extending and reusing

\[
\begin{array}{c|c|c}
PA & \rightarrow & MA \\
\hline
prCRL & \rightarrow & MAPA \quad \text{(Markov Automata Process Algebra)} \\
LPPE & \rightarrow & MLPPE \quad \text{(Markovian LPPE)} \\
\end{array}
\]
Approach: extending and reusing

PA $\rightarrow$ MA

prCRL $\rightarrow$ MAPA (Markov Automata Process Algebra)

LPPE $\rightarrow$ MLPPE (Markovian LPPE)
Strong bisimulation for Markov automata

Mimic interactive behaviour:
Strong bisimulation for Markov automata

Mimic interactive behaviour:

Mimic Markovian behaviour:
Strong bisimulation for Markov automata

Mimic interactive behaviour:

Mimic Markovian behaviour:

(If a state enables a $\tau$-transition, all rates are ignored.)
A process algebra with data for MAs: MAPA

Specification language MAPA:

- Based on prCRL: data and probabilistic choice
- Additional feature: Markovian rates
- Semantics defined in terms of Markov automata
- Minimal set of operators to facilitate formal manipulation
- Syntactic sugar easily definable
A process algebra with data for MAs: MAPA

Specification language MAPA:
- Based on prCRL: data and probabilistic choice
- Additional feature: Markovian rates
- Semantics defined in terms of Markov automata
- Minimal set of operators to facilitate formal manipulation
- Syntactic sugar easily definable

The grammar of MAPA

Process terms in MAPA are obtained by the following grammar:

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

There are 10 types of jobs
The type of job that arrives is chosen nondeterministically
Service time depends on job type (hence, we need queues)

The specification of the stations:
\[
\text{type} \colon \text{Jobs} = \{1, \ldots, 10\}
\]
\[
\text{Station} (i \colon \{1, 2\}, q \colon \text{Queue}) = \text{notFull} (q) \Rightarrow (2^i)
\]
\[
\sum_{j \colon \text{Jobs}} \text{arrive} (j)
\]
\[
\text{Station} (i, \text{enqueue} (q, j)) + \text{notEmpty} (q) \Rightarrow \text{deliver} (i, \text{head} (q))
\]
\[
\sum_k \in \{1, 9\} k = 1 \Rightarrow \text{Station} (i, q) + k = 9 \Rightarrow \text{Station} (i, \text{tail} (q))
\]
There are 10 types of jobs

The type of job that arrives is chosen nondeterministically

Service time depends on job type (hence, we need queues)

The specification of the stations:

\[
\text{Jobs} = \{1, \ldots, 10\}
\]

\[
\text{Station}(i:\{1, 2\}, q:\text{Queue}) = \text{notFull}(q) \Rightarrow (2i).
\]

\[
\sum_{j:\text{Jobs}} \text{arrive}(j) \cdot \text{Station}(i, \text{enqueue}(q, j)) + \text{notEmpty}(q) \Rightarrow \text{deliver}(i, \text{head}(q))
\]

\[
\sum_{k\in\{1, 9\}} k10: k = 1 \Rightarrow \text{Station}(i, q) + k = 9 \Rightarrow \text{Station}(i, \text{tail}(q))
\]
An example specification

- There are **10 types of jobs**
- The type of job that arrives is chosen **nondeterministically**
- Service time depends on job type (hence, we need **queues**)

There are 10 types of jobs
The type of job that arrives is chosen nondeterministically
Service time depends on job type (hence, we need queues)
An example specification

There are 10 types of jobs

The type of job that arrives is chosen nondeterministically

Service time depends on job type (hence, we need queues)

The specification of the stations:

\[
\text{type } \text{Jobs} = \{1, \ldots, 10\}
\]

\[
\text{Station}(i : \{1, 2\}, q : \text{Queue})
\]

\[
= \text{notFull}(q) \quad \Rightarrow (2i) \cdot \sum_{j:\text{Jobs}} \text{arrive}(j).\text{Station}(i, \text{enqueue}(q, j))
\]
An example specification

- There are 10 types of jobs
- The type of job that arrives is chosen nondeterministically
- Service time depends on job type (hence, we need queues)

The specification of the stations:

\[
\text{type } \text{Jobs} = \{1, \ldots, 10\}
\]

\[
\text{Station}(i : \{1, 2\}, q : \text{Queue})
\]

\[
= \text{notFull}(q) \implies (2i) \cdot \sum_{j:\text{Jobs}} \text{arrive}(j).\text{Station}(i, \text{enqueue}(q, j))
\]

\[
+ \text{notEmpty}(q) \implies \text{deliver}(i, \text{head}(q)) \sum_{k \in \{1, 9\}} \frac{k}{10} : k = 1 \implies \text{Station}(i, q)
\]

\[
+ k = 9 \implies \text{Station}(i, \text{tail}(q))
\]
An example specification

There are 10 types of jobs

The type of job that arrives is chosen nondeterministically

Service time depends on job type (hence, we need queues)

The specification of the stations:

**type Jobs = \{1, \ldots, 10\}**

**Station\(i: \{1, 2\}, q : \text{Queue}\)**

\[= \text{notFull}(q) \Rightarrow (2i) \cdot \sum_{j: \text{Jobs}} \text{arrive}(j) \cdot \text{Station}(i, \text{enqueue}(q, j))\]

\[+ \text{notEmpty}(q) \Rightarrow \text{deliver}(i, \text{head}(q))\left( \frac{1}{10} : \text{Station}(i, q) \oplus \frac{9}{10} : \text{Station}(i, \text{tail}(q)) \right)\]
Derivation-based operational semantics

\[
\begin{align*}
\text{MarkovPrefix} & \quad \frac{\lambda \cdot p}{(\lambda) \cdot p & \rightarrow \lambda \cdot p} \\
\text{SumLeft} & \quad \frac{p \rightarrow^a p'}{p + q & \rightarrow^a p'}
\end{align*}
\]
Derivation-based operational semantics

\[
\begin{align*}
\text{MarkovPrefix} & : \quad \lambda \cdot p \xrightarrow{\lambda} p \\
\text{SumLeft} & : \quad p + q \xrightarrow{a} p'
\end{align*}
\]

\[X = (3) \cdot (5) \cdot (2) \cdot X\]
Derivation-based operational semantics

\[
\text{MarkovPrefix} \quad \frac{\lambda}{(\lambda) \cdot p} \xrightarrow{\lambda} p \\
\text{SumLeft} \quad \frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'}
\]

\[X = (3) \cdot (5) \cdot (2) \cdot X\]
Derivation-based operational semantics

\[ \text{MarkovPrefix} \quad (\lambda) \cdot p \xrightarrow{\lambda} p \]

\[ \text{SumLeft} \quad \frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \]

\[ X = (3) \cdot (5) \cdot (2) \cdot X \]

\[ X = (3) \cdot (5) \cdot X + c \cdot X \]
Derivation-based operational semantics

\[ \text{MarkovPrefix} \quad (\lambda) \cdot p \xrightarrow{\lambda} p \]

\[ \text{SumLeft} \quad p \xrightarrow{a} p' \]

\[ X = (3) \cdot (5) \cdot (2) \cdot X \]

\[ X = (3) \cdot (5) \cdot X + c \cdot X \]
**Derivation-based operational semantics**

\[
\begin{align*}
\text{MarkovPrefix} & \quad (\lambda) \cdot p \xrightarrow{\lambda} p \\
\text{SumLeft} & \quad p \xrightarrow{a} p' \\
\text{SumRight} & \quad p + q \xrightarrow{a} p'
\end{align*}
\]
Derivation-based operational semantics

\[
\begin{align*}
\text{MarkovPrefix} & \quad \frac{\lambda}{(\lambda) \cdot p \xrightarrow{\lambda} p} \\
\text{SumLeft} & \quad \frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'}
\end{align*}
\]

\[X = (3) \cdot (5) \cdot X + (3) \cdot (5) \cdot X\]
Derivation-based operational semantics

\[ \text{MARKOVPREFIX} \quad \frac{\cdot}{(\lambda) \cdot p} \xrightarrow{\lambda} p \]

\[ \text{SUMLEFT} \quad \frac{p}{p + q} \xrightarrow{a} p' \]

\[ X = (3) \cdot (5) \cdot X + (3) \cdot (5) \cdot X \]

This is not right!
Derivation-based operational semantics

\[
\text{MarkovPrefix} \quad (\lambda) \cdot p \xrightarrow{\lambda} p
\]

\[
\text{SUM\!LEFT} \quad p \xrightarrow{a} p', \quad p + q \xrightarrow{a} p'
\]

\[
X = (3) \cdot (5) \cdot X + (3) \cdot (5) \cdot X
\]

This is not right!

As a solution, we look at derivations:
Derivation-based operational semantics

\[ \text{MARKOV PREFIX} \quad (\lambda) \cdot p \xrightarrow{\lambda}_{MP} p \]

\[ \text{SUM LEFT} \quad p \xrightarrow{a}_{D} p' \]

\[ p + q \xrightarrow{a}_{SL+D} p' \]

\[ X = (3) \cdot (5) \cdot X + (3) \cdot (5) \cdot X \]

This is not right!

As a solution, we look at derivations:
This is not right!

As a solution, we look at derivations:

\[
X \xrightarrow{3} \langle SL, MP \rangle (5) \cdot X \\
X \xrightarrow{3} \langle SR, MP \rangle (5) \cdot X
\]

Hence, the total rate from \( X \) to \( (5) \cdot X \) is \( 3 + 3 = 6 \).
Derivation-based operational semantics

\[
\text{MarkovPrefix} \quad \frac{(\lambda) \cdot p \xrightarrow{\lambda} MP \quad p}{p \xrightarrow{a \times D} p'}
\]

\[
\text{SUMLEFT} \quad \frac{p \xrightarrow{a \times SL + D} p'}{p + q \xrightarrow{a} SL + D \quad p'}
\]

\[X = (3) \cdot (5) \cdot X + (3) \cdot (5) \cdot X\]

This is not right!

As a solution, we look at derivations:

\[X \xrightarrow{3} \langle SL, MP \rangle (5) \cdot X\]

\[X \xrightarrow{3} \langle SR, MP \rangle (5) \cdot X\]

Hence, the total rate from \(X\) to \((5) \cdot X\) is \(3 + 3 = 6\).
MLPPEs

We defined a special format for MAPA, the MLPPE:

\[
X(g : G) = \sum_{i \in I} \sum_{d_i : D_i} c_i \Rightarrow a_i(b_i) \sum_{e_i : E_i} f_i : X(n_i)
\]

\[
+ \sum_{j \in J} \sum_{d_j : D_j} c_j \Rightarrow (\lambda_j) \cdot X(n_j)
\]
We defined a special format for MAPA, the **MLPPE**:

\[
X(g : G) = \sum_{i \in I} \sum_{d_i : D_i} c_i \Rightarrow a_i(b_i) \sum_{f_i : X(n_i)} f_i \\
+ \sum_{j \in J} \sum_{d_j : D_j} c_j \Rightarrow (\lambda_j) \cdot X(n_j)
\]

Advantages of using MLPPEs instead of MAPA specifications:

- Easy **state space generation**
- Straight-forward **parallel composition**
- **Symbolic optimisations** enabled at the language level
Basic idea: encode a rate $\lambda$ as action rate($\lambda$).

Problem: Bisimulation-preserving reductions on prCRL might change MAPA behaviour ($\lambda \cdot p + (\lambda \cdot p) \not\approx \lambda \cdot p$).
Encoding into prCRL

**Basic idea:** encode a rate $\lambda$ as action rate($\lambda$).

**Problem:** Bisimulation-preserving reductions on prCRL might change MAPA behaviour ($\lambda p + \lambda p \approx \lambda \cdot p$).

Efficient Modelling and Generation of Probabilistic Automata as well as Markov Automata

UNIVERSITY OF TWENTE. Efficient Modelling and Generation of Probabilistic Automata as well as Markov Automata

June 29, 2012 28 / 37
Encoding into prCRL

Basic idea: encode a rate \( \lambda \) as action rate(\( \lambda \)).

Problem: Bisimulation-preserving reductions on prCRL might change MAPA behaviour.

\[
\lambda \cdot p + \lambda \cdot p \not\approx MA \approx PA (\lambda) \Rightarrow rate(\lambda) \cdot p
\]
Basic idea: encode a rate $\lambda$ as action rate $\lambda$. 

Problem: Bisimulation-preserving reductions on prCRL might change MAPA behaviour $\lambda \cdot p + \lambda \cdot p \Rightarrow \text{rate}(\lambda) \cdot p + \text{rate}(\lambda) \cdot p \nless \approx \text{MAPA} \approx \text{PA}(\lambda) \Rightarrow \text{rate}(\lambda) \cdot p$.
Basic idea: encode a rate $\lambda$ as action $rate(\lambda)$.
Encoding into prCRL

- MAPA
- prCRL
- MLPPE
- LPPE

Basic idea: encode a rate $\lambda$ as action $\text{rate}(\lambda)$.

Problem:

Bisimulation-preserving reductions on prCRL might change MAPA behaviour.
Encoding into prCRL

Basic idea: encode a rate $\lambda$ as action $\text{rate}(\lambda)$.

Problem:

Bisimulation-preserving reductions on prCRL might change MAPA behaviour

$$(\lambda) \cdot p + (\lambda) \cdot p$$
Encoding into prCRL

Basic idea: encode a rate $\lambda$ as action $\text{rate}(\lambda)$.

Problem:

Bisimulation-preserving reductions on prCRL might change MAPA behaviour

$$(\lambda) \cdot p + (\lambda) \cdot p \Rightarrow \text{rate}(\lambda) \cdot p + \text{rate}(\lambda) \cdot p$$
Basic idea: encode a rate $\lambda$ as action $\text{rate}(\lambda)$.

Problem:

Bisimulation-preserving reductions on prCRL might change MAPA behaviour

\[(\lambda) \cdot p + (\lambda) \cdot p \Rightarrow \text{rate}(\lambda) \cdot p + \text{rate}(\lambda) \cdot p\]

\[\approx_{\text{PA}}\]

\[\text{rate}(\lambda) \cdot p\]
Encoding into prCRL

Basic idea: encode a rate $\lambda$ as action $\text{rate}(\lambda)$.

Problem:

Bisimulation-preserving reductions on prCRL might change MAPA behaviour

$$(\lambda) \cdot p + (\lambda) \cdot p \Rightarrow \text{rate}(\lambda) \cdot p + \text{rate}(\lambda) \cdot p$$

$\approx_{\text{PA}}$

$$(\lambda) \cdot p \Leftrightarrow \text{rate}(\lambda) \cdot p$$
Encoding into \textit{prCRL}

Basic idea: encode a rate $\lambda$ as action $\text{rate}(\lambda)$.

Problem:

Bisimulation-preserving reductions on \textit{prCRL} might change MAPA behaviour

\[
(\lambda) \cdot p + (\lambda) \cdot p \Rightarrow \text{rate}(\lambda) \cdot p + \text{rate}(\lambda) \cdot p
\]

$\not\approx_{\text{MA}}$ $\approx_{\text{PA}}$

\[
(\lambda) \cdot p \Leftarrow \text{rate}(\lambda) \cdot p
\]
Possible solution: encode a rate $\lambda$ as action rate$_i(\lambda)$.
Encoding into prCRL

Possible solution: encode a rate $\lambda$ as action rate $i(\lambda)$.

Problem:
This still doesn’t work...
**Encoding into prCRL**

Possible solution: encode a rate $\lambda$ as action rate $i(\lambda)$.

Problem:

This still doesn't work...

$$(\lambda) \cdot p + (\lambda) \cdot p$$
Encoding into prCRL

Possible solution: encode a rate $\lambda$ as action rate,$_i(\lambda)$.

Problem:
This still doesn’t work... 

\[(\lambda) \cdot p + (\lambda) \cdot p \Rightarrow rate_1(\lambda) \cdot p + rate_2(\lambda) \cdot p\]
Encoding into prCRL

Possible solution: encode a rate $\lambda$ as action $\text{rate}_i(\lambda)$.

Problem:
This still doesn’t work…

$$(\lambda) \cdot p + (\lambda) \cdot p \Rightarrow \text{rate}_1(\lambda) \cdot p + \text{rate}_2(\lambda) \cdot p$$

$\approx_{\text{PA}}$

$$\text{rate}_1(\lambda) \cdot p + \text{rate}_2(\lambda) \cdot p + \text{rate}_2(\lambda) \cdot p$$
Encoding into prCRL

Possible solution: encode a rate $\lambda$ as action $\text{rate}_i(\lambda)$.

Problem:
This still doesn't work...

\[ (\lambda) \cdot p + (\lambda) \cdot p \Rightarrow \text{rate}_1(\lambda) \cdot p + \text{rate}_2(\lambda) \cdot p \]
\[ \approx_{\text{PA}} \]

\[ (\lambda) \cdot p + (\lambda) \cdot p + (\lambda) \cdot p \Leftarrow \text{rate}_1(\lambda) \cdot p + \text{rate}_2(\lambda) \cdot p + \text{rate}_2(\lambda) \cdot p \]
Encoding into prCRL

Possible solution: encode a rate $\lambda$ as action $rate_i(\lambda)$.

Problem:
This still doesn’t work...

$$
(\lambda) \cdot p + (\lambda) \cdot p \Rightarrow rate_1(\lambda) \cdot p + rate_2(\lambda) \cdot p
$$

$\not\approx_{MA}$ $\approx_{PA}$

$$
(\lambda) \cdot p + (\lambda) \cdot p + (\lambda) \cdot p \iff rate_1(\lambda) \cdot p + rate_2(\lambda) \cdot p + rate_2(\lambda) \cdot p
$$
Encoding into prCRL

Possible solution: encode a rate $\lambda$ as action rate$_i(\lambda)$.

Problem:
This still doesn’t work...

\[
(\lambda) \cdot p + (\lambda) \cdot p \Rightarrow rate_1(\lambda) \cdot p + rate_2(\lambda) \cdot p
\]

$\not\approx_{MA}$  $\approx_{PA}$

\[
(\lambda) \cdot p + (\lambda) \cdot p + (\lambda) \cdot p \Leftarrow rate_1(\lambda) \cdot p + rate_2(\lambda) \cdot p + rate_2(\lambda) \cdot p
\]

Stronger equivalence on prCRL specifications needed!
Two prCRL terms are **derivation-preserving bisimulation** if

- There is a **strong bisimulation** relation $R$ containing them.
Derivation-preserving bisimulation

Two prCRL terms are derivation-preserving bisimulation if

- There is a strong bisimulation relation $R$ containing them
- Every bisimilar pair $(p, p')$ has the same number of $\text{rate}(\lambda)$ derivations to every equivalence class $[r]_R$. 

**Proposition**

Derivation-preserving bisimulation is a congruence for prCRL.
Two prCRL terms are derivation-preserving bisimulation if

- There is a **strong bisimulation** relation $R$ containing them
- Every bisimilar pair $(p, p')$ has the **same number of** $rate(\lambda)$ derivations to every equivalence class $[r]_R$.

\[
rate(\lambda) \cdot a \cdot X + rate(\lambda) \cdot a \cdot X \\
\]

\[
\begin{array}{c}
a \quad rate(\lambda) \quad (2x) \\
a \cdot X \\
\end{array} \quad \not\approx_{dp} \quad \begin{array}{c}
a \quad rate(\lambda) \quad (1x) \\
a \cdot X \\
\end{array}
\]
Derivation-preserving bisimulation

Two prCRL terms are *derivation-preserving bisimulation* if

- There is a **strong bisimulation** relation $R$ containing them
- Every bisimilar pair $(p, p')$ has the **same number of rate($\lambda$)** derivations to every equivalence class $[r]_R$.

\[
\begin{align*}
\text{rate}(\lambda) \cdot a \cdot X &+ \text{rate}(\lambda) \cdot a \cdot X \\
\text{rate}(\lambda) \cdot a \cdot X &+ \text{rate}(\lambda) \cdot (a \cdot X + a \cdot X) \\
(1x) \text{rate}(\lambda) &+ (1x) \text{rate}(\lambda) \\
\end{align*}
\]

\[
\begin{align*}
a \cdot X &\approx_{dp} a \cdot X \\
\end{align*}
\]
Derivation-preserving bisimulation

Two prCRL terms are derivation-preserving bisimulation if

- There is a strong bisimulation relation \( R \) containing them
- Every bisimilar pair \((p, p')\) has the same number of \( \text{rate}(\lambda) \) derivations to every equivalence class \([r]_R\).

**Proposition**

Derivation-preserving bisimulation is a congruence for prCRL.
Derivation-preserving bisimulation: important results

Theorem

Given a derivation-preserving prCRL transformation \( f \),

\[
\text{decode}(f(\text{encode}(M))) \approx M
\]

for every MAPA specification \( M \).
Derivation-preserving bisimulation: important results

**Theorem**

Given a derivation-preserving prCRL transformation \( f \),

\[
\text{decode}(f(\text{encode}(M))) \approx M
\]

for every MAPA specification \( M \).

This enables many techniques from the PA world to be **generalised trivially** to the MA world!
Derivation-preserving bisimulation: important results

**Theorem**

*Given a derivation-preserving prCRL transformation \( f \),

\[
\text{decode}(f(\text{encode}(M))) \approx M
\]

*for every MAPA specification \( M \).*

This enables many techniques from the PA world to be generalised trivially to the MA world!

**Corollary**

*The linearisation procedure of prCRL can be reused for MAPA.*
Generalising existing reduction techniques

Existing reduction techniques that preserve derivations:

- Constant elimination
- Expression simplification
- Dead variable reduction
Generalising existing reduction techniques

Existing reduction techniques that preserve derivations:

- **Constant elimination**
- **Expression simplification**
- **Dead variable reduction**
Generalising existing reduction techniques

Existing reduction techniques that preserve derivations:

- Constant elimination
- Expression simplification
- Dead variable reduction

\[
\text{deadVarRed} = \text{decode} \circ \text{deadVarRedOld} \circ \text{encode}
\]
New reduction techniques for MAPA:

- Maximal progress reduction
- Summation elimination
Novel reduction techniques

New reduction techniques for MAPA:

- **Maximal progress reduction**
- **Summation elimination**

\[ X = \tau \cdot X + (5) \cdot X \rightarrow X = \tau \cdot X \]
Novel reduction techniques

New reduction techniques for MAPA:

- Maximal progress reduction
- Summation elimination

\[ X = \sum_{d: \{1,2,3\}} d = 2 \implies \text{send}(d) \cdot X \]
\[ Y = \sum_{d: \{1,2,3\}} (5) \cdot Y \]
\[ \rightarrow \]

\[ X = \text{send}(2) \cdot X \]
\[ Y = (15) \cdot Y \]
Implementation and Case Study

Implementation in SCOOP:

- Programmed in Haskell
- Stand-alone and web-based interface
- Linearisation, optimisation, state space generation

<table>
<thead>
<tr>
<th>Spec.</th>
<th>States</th>
<th>Trans.</th>
<th>MLPPE</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue-3-5</td>
<td>316,058</td>
<td>581,892</td>
<td>15 / 335</td>
<td>87.4</td>
</tr>
<tr>
<td>queue-3-6</td>
<td>218,714</td>
<td>484,548</td>
<td>8 / 224</td>
<td>20.7</td>
</tr>
<tr>
<td>queue-3-6'</td>
<td>76%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-5-2</td>
<td>27,659</td>
<td>47,130</td>
<td>15 / 335</td>
<td>4.3</td>
</tr>
<tr>
<td>queue-5-3</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>235.8</td>
</tr>
<tr>
<td>queue-5-3'</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-25-1</td>
<td>3,330</td>
<td>5,256</td>
<td>15 / 335</td>
<td>0.5</td>
</tr>
<tr>
<td>queue-100-1</td>
<td>50,805</td>
<td>81,006</td>
<td>15 / 335</td>
<td>8.9</td>
</tr>
<tr>
<td>mutex-3-2</td>
<td>17,352</td>
<td>40,200</td>
<td>27 / 3,540</td>
<td>12.3</td>
</tr>
<tr>
<td>mutex-3-4</td>
<td>129,112</td>
<td>320,136</td>
<td>27 / 3,540</td>
<td>95.8</td>
</tr>
<tr>
<td>mutex-3-6</td>
<td>425,528</td>
<td>1,137,048</td>
<td>27 / 3,540</td>
<td>330.8</td>
</tr>
<tr>
<td>mutex-4-1</td>
<td>27,701</td>
<td>80,516</td>
<td>36 / 5,872</td>
<td>33.0</td>
</tr>
<tr>
<td>mutex-4-2</td>
<td>360,768</td>
<td>1,035,584</td>
<td>36 / 5,872</td>
<td>435.9</td>
</tr>
<tr>
<td>mutex-4-3</td>
<td>1,711,141</td>
<td>5,015,692</td>
<td>36 / 5,872</td>
<td>2,108.0</td>
</tr>
<tr>
<td>mutex-5-1</td>
<td>294,882</td>
<td>1,051,775</td>
<td>45 / 8,780</td>
<td>549.7</td>
</tr>
<tr>
<td>mutex-5-2</td>
<td>218,714</td>
<td>841,750</td>
<td>20 / 5,430</td>
<td>216.6</td>
</tr>
<tr>
<td>(mutex-25-1)</td>
<td>100%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(mutex-100-1)</td>
<td>26%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table: State space generation using SCOOP.
Implementation in SCOOP:

Programmed in Haskell

Stand-alone and web-based interface

Linearisation, optimisation, state space generation

<table>
<thead>
<tr>
<th>Spec.</th>
<th>States</th>
<th>Trans.</th>
<th>MLPPE</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue-3-5</td>
<td>316,058</td>
<td>581,892</td>
<td>15 / 335</td>
<td>87.4</td>
</tr>
<tr>
<td>76%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-3-6</td>
<td>1,005,699</td>
<td>1,874,138</td>
<td>15 / 335</td>
<td>323.3</td>
</tr>
<tr>
<td>67%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-3-6'</td>
<td>1,005,699</td>
<td>1,874,138</td>
<td>15 / 335</td>
<td>319.5</td>
</tr>
<tr>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-5-2</td>
<td>27,659</td>
<td>47,130</td>
<td>15 / 335</td>
<td>4.3</td>
</tr>
<tr>
<td>56%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-5-3</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>235.8</td>
</tr>
<tr>
<td>64%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-5-3'</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>233.2</td>
</tr>
<tr>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-25-1</td>
<td>3,330</td>
<td>5,256</td>
<td>15 / 335</td>
<td>0.5</td>
</tr>
<tr>
<td>20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue-100-1</td>
<td>50,805</td>
<td>81,006</td>
<td>15 / 335</td>
<td>8.9</td>
</tr>
<tr>
<td>26%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-3-2</td>
<td>17,352</td>
<td>40,200</td>
<td>27 / 3,540</td>
<td>12.3</td>
</tr>
<tr>
<td>63%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-3-4</td>
<td>129,112</td>
<td>320,136</td>
<td>27 / 3,540</td>
<td>95.8</td>
</tr>
<tr>
<td>68%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-3-6</td>
<td>425,528</td>
<td>1,137,048</td>
<td>27 / 3,540</td>
<td>330.8</td>
</tr>
<tr>
<td>70%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-4-1</td>
<td>27,701</td>
<td>80,516</td>
<td>36 / 5,872</td>
<td>33.0</td>
</tr>
<tr>
<td>59%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-4-2</td>
<td>360,768</td>
<td>1,035,584</td>
<td>36 / 5,872</td>
<td>435.9</td>
</tr>
<tr>
<td>67%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-4-3</td>
<td>1,711,141</td>
<td>5,015,692</td>
<td>36 / 5,872</td>
<td>2,108.0</td>
</tr>
<tr>
<td>69%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-5-1</td>
<td>294,882</td>
<td>1,051,775</td>
<td>45 / 8,780</td>
<td>549.7</td>
</tr>
<tr>
<td>61%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-5-2</td>
<td>27,659</td>
<td>47,130</td>
<td>15 / 335</td>
<td>4.3</td>
</tr>
<tr>
<td>56%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-5-3</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>235.8</td>
</tr>
<tr>
<td>64%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mutex-5-3'</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>233.2</td>
</tr>
<tr>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table: State space generation using SCOOP.
**Implementation and Case Study**

Programmed in Haskell

<table>
<thead>
<tr>
<th>Original States</th>
<th>Transitions</th>
<th>MLPPE Time</th>
<th>Reduced States</th>
<th>Transitions</th>
<th>MLPPE Time</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue-3-5</td>
<td>316,058</td>
<td>581,892</td>
<td>15 / 335</td>
<td>218,714</td>
<td>484,548</td>
<td>76%</td>
</tr>
<tr>
<td>queue-3-6</td>
<td>1,005,699</td>
<td>1,874,138</td>
<td>15 / 335</td>
<td>670,294</td>
<td>1,538,733</td>
<td>80%</td>
</tr>
<tr>
<td>queue-3-6'</td>
<td>1,005,699</td>
<td>1,874,138</td>
<td>15 / 335</td>
<td>74</td>
<td>108</td>
<td>100%</td>
</tr>
<tr>
<td>queue-5-2</td>
<td>27,659</td>
<td>47,130</td>
<td>15 / 335</td>
<td>23,690</td>
<td>43,161</td>
<td>56%</td>
</tr>
<tr>
<td>queue-5-3</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>926,746</td>
<td>1,851,312</td>
<td>64%</td>
</tr>
<tr>
<td>queue-5-3'</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>170</td>
<td>256</td>
<td>100%</td>
</tr>
<tr>
<td>queue-25-1</td>
<td>3,330</td>
<td>5,256</td>
<td>15 / 335</td>
<td>3,330</td>
<td>5,256</td>
<td>20%</td>
</tr>
<tr>
<td>queue-100-1</td>
<td>50,805</td>
<td>81,006</td>
<td>15 / 335</td>
<td>50,805</td>
<td>81,006</td>
<td>26%</td>
</tr>
<tr>
<td>mutex-3-2</td>
<td>17,352</td>
<td>40,200</td>
<td>27 / 3,540</td>
<td>10,560</td>
<td>25,392</td>
<td>63%</td>
</tr>
<tr>
<td>mutex-3-4</td>
<td>129,112</td>
<td>320,136</td>
<td>27 / 3,540</td>
<td>70,744</td>
<td>169,128</td>
<td>68%</td>
</tr>
<tr>
<td>mutex-3-6</td>
<td>425,528</td>
<td>1,137,048</td>
<td>27 / 3,540</td>
<td>224,000</td>
<td>534,624</td>
<td>70%</td>
</tr>
<tr>
<td>mutex-4-1</td>
<td>27,701</td>
<td>80,516</td>
<td>36 / 5,872</td>
<td>20,025</td>
<td>62,876</td>
<td>59%</td>
</tr>
<tr>
<td>mutex-4-2</td>
<td>360,768</td>
<td>1,035,584</td>
<td>36 / 5,872</td>
<td>218,624</td>
<td>671,328</td>
<td>67%</td>
</tr>
<tr>
<td>mutex-4-3</td>
<td>1,711,141</td>
<td>5,015,692</td>
<td>36 / 5,872</td>
<td>958,921</td>
<td>2,923,300</td>
<td>69%</td>
</tr>
<tr>
<td>mutex-5-1</td>
<td>294,882</td>
<td>1,051,775</td>
<td>45 / 8,780</td>
<td>218,717</td>
<td>841,750</td>
<td>61%</td>
</tr>
<tr>
<td>mutex-5-2</td>
<td>27,701</td>
<td>80,516</td>
<td>36 / 5,872</td>
<td>20,025</td>
<td>62,876</td>
<td>59%</td>
</tr>
<tr>
<td>mutex-5-3</td>
<td>1,711,141</td>
<td>5,015,692</td>
<td>36 / 5,872</td>
<td>958,921</td>
<td>2,923,300</td>
<td>69%</td>
</tr>
</tbody>
</table>

**Table:** State space generation using SCOOP.
X =

(T => tau . X[])

Initial state: X

Powered by puptol
X =

(T => tau . X[])

Initial state: X

Powered by puptol
Implementation and Case Study

Implementation in SCOOP:
- Programmed in Haskell
- Stand-alone and web-based interface
- Linearisation, optimisation, state space generation

<table>
<thead>
<tr>
<th>Spec.</th>
<th>Original</th>
<th>Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Trans.</td>
</tr>
<tr>
<td>queue-3-5</td>
<td>316,058</td>
<td>581,892</td>
</tr>
<tr>
<td>queue-3-6</td>
<td>1,005,699</td>
<td>1,874,138</td>
</tr>
<tr>
<td>queue-3-6'</td>
<td>1,005,699</td>
<td>1,874,138</td>
</tr>
<tr>
<td>queue-5-2</td>
<td>27,659</td>
<td>47,130</td>
</tr>
<tr>
<td>queue-5-3</td>
<td>1,191,738</td>
<td>2,116,304</td>
</tr>
<tr>
<td>queue-5-3'</td>
<td>1,191,738</td>
<td>2,116,304</td>
</tr>
<tr>
<td>queue-25-1</td>
<td>3,330</td>
<td>5,256</td>
</tr>
<tr>
<td>queue-100-1</td>
<td>50,805</td>
<td>81,006</td>
</tr>
<tr>
<td>mutex-3-2</td>
<td>17,352</td>
<td>40,200</td>
</tr>
<tr>
<td>mutex-3-4</td>
<td>129,112</td>
<td>320,136</td>
</tr>
<tr>
<td>mutex-3-6</td>
<td>425,528</td>
<td>1,137,048</td>
</tr>
<tr>
<td>mutex-4-1</td>
<td>27,701</td>
<td>80,516</td>
</tr>
<tr>
<td>mutex-4-2</td>
<td>360,768</td>
<td>1,035,584</td>
</tr>
<tr>
<td>mutex-4-3</td>
<td>1,711,141</td>
<td>5,015,692</td>
</tr>
<tr>
<td>mutex-5-1</td>
<td>294,882</td>
<td>1,051,775</td>
</tr>
</tbody>
</table>

Table: State space generation using SCOOP.
Implementation and Case Study

Implementation in SCOOP:
- Programmed in Haskell
- Stand-alone and web-based interface
- Linearisation, optimisation, state space generation

<table>
<thead>
<tr>
<th>Spec.</th>
<th>Original</th>
<th>Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Trans.</td>
</tr>
<tr>
<td>queue-3-5</td>
<td>316,058</td>
<td>581,892</td>
</tr>
<tr>
<td>queue-3-6</td>
<td>1,005,699</td>
<td>1,874,138</td>
</tr>
<tr>
<td>queue-3-6’</td>
<td>1,005,699</td>
<td>1,874,138</td>
</tr>
<tr>
<td>queue-5-2</td>
<td>27,659</td>
<td>47,130</td>
</tr>
<tr>
<td>queue-5-3</td>
<td>1,191,738</td>
<td>2,116,304</td>
</tr>
<tr>
<td>queue-5-3’</td>
<td>1,191,738</td>
<td>2,116,304</td>
</tr>
<tr>
<td>queue-25-1</td>
<td>3,330</td>
<td>5,256</td>
</tr>
<tr>
<td>queue-100-1</td>
<td>50,805</td>
<td>81,006</td>
</tr>
<tr>
<td>mutex-3-2</td>
<td>17,352</td>
<td>40,200</td>
</tr>
<tr>
<td>mutex-3-4</td>
<td>129,112</td>
<td>320,136</td>
</tr>
<tr>
<td>mutex-3-6</td>
<td>425,528</td>
<td>1,137,048</td>
</tr>
<tr>
<td>mutex-4-1</td>
<td>27,701</td>
<td>80,516</td>
</tr>
<tr>
<td>mutex-4-2</td>
<td>360,768</td>
<td>1,035,584</td>
</tr>
<tr>
<td>mutex-4-3</td>
<td>1,711,141</td>
<td>5,015,692</td>
</tr>
<tr>
<td>mutex-5-1</td>
<td>294,882</td>
<td>1,051,775</td>
</tr>
</tbody>
</table>

Table: State space generation using SCOOP.
Implementation and Case Study

Implementation in SCOOP:

- Programmed in Haskell
- Stand-alone and web-based interface
- Linearisation, optimisation, state space generation

<table>
<thead>
<tr>
<th>Spec.</th>
<th>Original States</th>
<th>Original Trans.</th>
<th>Original MLPPE</th>
<th>Original Time</th>
<th>Reduced States</th>
<th>Reduced Trans.</th>
<th>Reduced MLPPE</th>
<th>Reduced Time</th>
<th>Red.</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue-3-5</td>
<td>316,058</td>
<td>581,192</td>
<td>15 / 335</td>
<td>87.4</td>
<td>218,714</td>
<td>484,548</td>
<td>8 / 224</td>
<td>20.7</td>
<td>76%</td>
</tr>
<tr>
<td>queue-3-6</td>
<td>1,005,699</td>
<td>1,874,138</td>
<td>15 / 335</td>
<td>323.3</td>
<td>670,294</td>
<td>1,538,733</td>
<td>8 / 224</td>
<td>64.7</td>
<td>80%</td>
</tr>
<tr>
<td>queue-3-6'</td>
<td>1,005,699</td>
<td>1,874,138</td>
<td>15 / 335</td>
<td>319.5</td>
<td>74</td>
<td>108</td>
<td>5 / 170</td>
<td>0.0</td>
<td>100%</td>
</tr>
<tr>
<td>queue-5-2</td>
<td>27,659</td>
<td>47,130</td>
<td>15 / 335</td>
<td>4.3</td>
<td>23,690</td>
<td>43,161</td>
<td>8 / 224</td>
<td>1.9</td>
<td>56%</td>
</tr>
<tr>
<td>queue-5-3</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>235.8</td>
<td>926,746</td>
<td>1,851,312</td>
<td>8 / 224</td>
<td>84.2</td>
<td>64%</td>
</tr>
<tr>
<td>queue-5-3'</td>
<td>1,191,738</td>
<td>2,116,304</td>
<td>15 / 335</td>
<td>233.2</td>
<td>170</td>
<td>256</td>
<td>5 / 170</td>
<td>0.0</td>
<td>100%</td>
</tr>
<tr>
<td>queue-25-1</td>
<td>3,330</td>
<td>5,256</td>
<td>15 / 335</td>
<td>0.5</td>
<td>3,330</td>
<td>5,256</td>
<td>8 / 224</td>
<td>0.4</td>
<td>20%</td>
</tr>
<tr>
<td>queue-100-1</td>
<td>50,805</td>
<td>81,006</td>
<td>15 / 335</td>
<td>8.9</td>
<td>50,805</td>
<td>81,006</td>
<td>8 / 224</td>
<td>6.6</td>
<td>26%</td>
</tr>
<tr>
<td>mutex-3-2</td>
<td>17,352</td>
<td>40,200</td>
<td>27 / 3,540</td>
<td>12.3</td>
<td>10,560</td>
<td>25,392</td>
<td>12 / 2,190</td>
<td>4.6</td>
<td>63%</td>
</tr>
<tr>
<td>mutex-3-4</td>
<td>129,112</td>
<td>320,136</td>
<td>27 / 3,540</td>
<td>95.8</td>
<td>70,744</td>
<td>169,128</td>
<td>12 / 2,190</td>
<td>30.3</td>
<td>68%</td>
</tr>
<tr>
<td>mutex-3-6</td>
<td>425,528</td>
<td>1,137,048</td>
<td>27 / 3,540</td>
<td>330.8</td>
<td>224,000</td>
<td>534,624</td>
<td>12 / 2,190</td>
<td>99.0</td>
<td>70%</td>
</tr>
<tr>
<td>mutex-4-1</td>
<td>27,701</td>
<td>80,516</td>
<td>36 / 5,872</td>
<td>33.0</td>
<td>20,025</td>
<td>62,876</td>
<td>16 / 3,632</td>
<td>13.5</td>
<td>59%</td>
</tr>
<tr>
<td>mutex-4-2</td>
<td>360,768</td>
<td>1,035,584</td>
<td>36 / 5,872</td>
<td>435.9</td>
<td>218,624</td>
<td>671,328</td>
<td>16 / 3,632</td>
<td>145.5</td>
<td>67%</td>
</tr>
<tr>
<td>mutex-4-3</td>
<td>1,711,141</td>
<td>5,015,692</td>
<td>36 / 5,872</td>
<td>2,108.0</td>
<td>958,921</td>
<td>2,923,300</td>
<td>16 / 3,632</td>
<td>644.3</td>
<td>69%</td>
</tr>
<tr>
<td>mutex-5-1</td>
<td>294,882</td>
<td>1,051,775</td>
<td>45 / 8,780</td>
<td>549.7</td>
<td>218,717</td>
<td>841,750</td>
<td>20 / 5,430</td>
<td>216.6</td>
<td>61%</td>
</tr>
</tbody>
</table>

Table: State space generation using SCOOP.
GSPN analysis

GSPN (PNML)

reach P1 = 1 & P5 = 2
**GSPN analysis**

GSPN (PNML)

reach \( P1 = 1 \) & \( P5 = 2 \)

MAPA

\[
P2 >= 1 \Rightarrow T2 \\
\text{GSPN}[P2--, P4++] \\
+ P5 >= 1 \Rightarrow (4.0) \\
\text{GSPN}[P2++, P5--] \\
+ ... \\
\text{init GSPN}(1,1,1,0,1)
\]

GEMMA

\[
\text{reach } P1 = 1 \text{ & } P5 = 2
\]
GSPN analysis

GSPN (PNML)

reach P1 = 1 & P5 = 2

GEMMA


P2 >= 1 => T2.


GSPN[P2++, P5--] + ...

init GSPN(1,1,1,0,1)

MAPA

reach P1 = 1 & P5 = 2

SCOOOP

MA

#GOALS S4

UNIVERSITY OF TWENTE.

Efficient Modelling and Generation of Probabilistic Automata as well as Markov Automata

June 29, 2012 35 / 37
GSPN analysis

- **Introduction**
- **prCRL**
- **Linearisation**
- **Reductions**
- **MAPA**
- **Encoding and decoding**
- **Reductions**
- **Case study**
- **Conclusions**

GSPN (PNML)

```
reach P1 = 1 & P5 = 2
```

GEMMA

```
GSPN(P1:N,P2:N,P3:N,
P4:N,P5:N) =
P2 >= 1 => T2 .
  GSPN[P2--, P4++]
+ P5 >= 1 => (4.0) .
  GSPN[P2++, P5--]
+ ...
init GSPN(1,1,1,0,1)
reach P1 = 1 & P5 = 2
```

MAPA

```
#GOALS S2
```

SCOOP (optimised)

```
#GOALS S2
```

MA

Efficient Modelling and Generation of Probabilistic Automata as well as Markov Automata

June 29, 2012 35 / 37
GSPN analysis

GSPN (PNML)

reach P1 = 1 & P5 = 2

MAPA


P2 >= 1 => T2 .
GSPN[P2--, P4++]
+ P5 >= 1 => (4.0) .
GSPN[P2++, P5--]
+ ...

init GSPN(1,1,1,0,1)

reach P1 = 1 & P5 = 2

Results

Min. unbounded reach.: 1.0
Max. unbounded reach.: 1.0

Min. expected time: 0.0
Max. expected time: 0.2

Min. LRA: 0.0
Max. LRA: 0.4

#GOALS S2

MA

SCOOP (optimised)

UNIVERSITY OF TWENTE.

Efficient Modelling and Generation of Probabilistic Automata as v

June 29, 2012
Conclusions and Future Work

Conclusions:

- We introduced a new process-algebraic framework (MAPA) with data for modelling and generating Markov automata.
- We introduced the MLPPE for easy state space generation, parallel composition and reduction techniques.
Conclusions:

- We introduced a new process-algebraic framework (MAPA) with data for modelling and generating Markov automata.
- We introduced the MLPPE for easy state space generation, parallel composition and reduction techniques.
- We showed an encoding of MAPA into prCRL.
- We showed when prCRL techniques can be used safely by encoding, using a novel notion of bisimulation.

Future Work:

- Generalise confluence reduction to MAs and MAPA.
Conclusions and Future Work

Conclusions:

- We introduced a new process-algebraic framework (MAPA) with data for modelling and generating Markov automata.
- We introduced the MLPPE for easy state space generation, parallel composition and reduction techniques.
- We showed an encoding of MAPA into prCRL.
- We showed when prCRL techniques can be used safely by encoding, using a novel notion of bisimulation.
- All our results apply to LTSs, DTMCs, CTMCs, IMCs and PAs.
- Model checking of MAs and GSPNs is now possible.
Conclusions and Future Work

Conclusions:

- We introduced a new process-algebraic framework (MAPA) with data for modelling and generating Markov automata.
- We introduced the MLPPE for easy state space generation, parallel composition and reduction techniques.
- We showed an encoding of MAPA into prCRL.
- We showed when prCRL techniques can be used safely by encoding, using a novel notion of bisimulation.
- All our results apply to LTSs, DTMCs, CTMCs, IMCs and PAs.
- Model checking of MAs and GSPNs is now possible.

Future Work:

- Generalise confluence reduction to MAs and MAPA.
Questions

Have a look at fmt.cs.utwente.nl/~timmer/scoop