Symbolic reductions of probabilistic models using linear process equations

Mark Timmer
January 18, 2011

Joint work with
Mariëlle Stoelinga and Jaco van de Pol
Contents

1. Introduction
2. A process algebra with data and probability: prCRL
3. Confluence reduction
4. Detecting confluence symbolically
5. Case study: leader election protocols
6. Conclusions
<table>
<thead>
<tr>
<th></th>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>A process algebra with data and probability: prCRL</td>
</tr>
<tr>
<td>3</td>
<td>Confluence reduction</td>
</tr>
<tr>
<td>4</td>
<td>Detecting confluence symbolically</td>
</tr>
<tr>
<td>5</td>
<td>Case study: leader election protocols</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
</tr>
</tbody>
</table>
**Dependability** of computer systems is becoming more and more important.

---

**Windows blue screen**

-A fatal exception 0E has occurred at 0137:BFFA21C9. The current application will be terminated.

-Press any key to terminate the current application.

-Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

-Press any key to continue

---

**Ariane 5 crash**
Dependability of computer systems is becoming more and more important.

Our aim: use quantitative formal methods to improve system quality.
A popular solution is **model checking**; verifying **properties** of a system by constructing a **model** and ranging over its **state space**.
A popular solution is **model checking**; verifying **properties** of a system by constructing a **model** and ranging over its **state space**.
A popular solution is model checking; verifying properties of a system by constructing a model and ranging over its state space.
Probabilistic model checking:

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

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

![Diagram of a probabilistic model]

- **Non-deterministically** choose one of the three transitions
- **Probabilistically** choose the next state
Introduction – 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
Overview of our approach

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

Probabilistic specification

- Linearisation
- Instantiation

Intermediate format

- Optimisation
- Minimisation

State space (PA)

- Visualisation
- Model checking
Overview of our approach

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

Probabilistic specification

- Linearisation

Intermediate format

- Instantiation

State space (PA)

- Optimisation
  - Dead variable reduction
  - Confluence reduction

Visualisation

Model checking

- Minimisation
Notions of equivalence: **strong/branching probabilistic bisimulation**
Notions of equivalence: **strong/branching probabilistic bisimulation**
Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation

![Diagram showing strong/branching probabilistic bisimulation]

- Probability of green: $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{3} = \frac{5}{12}$
Notions of equivalence: strong/branching probabilistic bisimulation
Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation
Equivalences: probabilistic bisimulation

Notions of equivalence: **strong/branching probabilistic bisimulation**

![Diagram showing two probabilistic models with labeled transitions and probabilities.](diagram.png)
Notions of equivalence: strong/branching probabilistic bisimulation
Notions of equivalence: strong/branching probabilistic bisimulation

Probability of green: \( \frac{1}{2} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{2} \)
Table of Contents

1 Introduction

2 A process algebra with data and probability: prCRL

3 Confluence reduction

4 Detecting confluence symbolically

5 Case study: leader election protocols

6 Conclusions
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**

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

Sending an arbitrary natural number

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

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

\[ + \ \text{active} \Rightarrow \tau \sum_{n : \mathbb{N} > 0} \frac{1}{2^n} : \left( \text{send}(n) \cdot X(\text{false}) \right) \]
Composability using extended prCRL

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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.
Linear Probabilistic Process Equations – an example

Specification in prCRL

\[
X(\text{active : Bool}) =
\begin{align*}
\text{not(\text{active})} & \Rightarrow \text{ping} \cdot \sum_{b: \text{Bool}} X(b) \\
\text{+ active} & \Rightarrow \tau \sum_{n: \mathbb{N} \geq 0} \frac{1}{2^n} : \text{send}(n) \cdot X(\text{false})
\end{align*}
\]
Linear Probabilistic Process Equations – an example

Specification in prCRL

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

Specification in LPPE

\[ X(pc : \{1..3\}, n : \mathbb{N} \geq 0) = \]
\[ + pc = 1 \Rightarrow \text{ping} \cdot X(2, 1) \]
\[ + pc = 2 \Rightarrow \text{ping} \cdot X(2, 1) \]
\[ + pc = 2 \Rightarrow \tau \sum_{n : \mathbb{N} \geq 0} \frac{1}{2^n} : X(3, n) \]
\[ + pc = 3 \Rightarrow \text{send}(n) \cdot X(1, 1) \]
# Table of Contents

1. Introduction
2. A process algebra with data and probability: prCRL
3. Confluence reduction
4. Detecting confluence symbolically
5. Case study: leader election protocols
6. Conclusions
Unobservable $\tau$-steps might disable behaviour…
Unobservable $\tau$-steps might disable behaviour...
Unobservable $\tau$-steps might disable behaviour...
Unobservable $\tau$-steps might disable behaviour...

...though often, they connect branching bisimilar states
Unobservable $\tau$-steps might disable behaviour...

...though often, they connect branching bisimilar states
Unobservable $\tau$-steps might disable behaviour...

...though often, they connect branching bisimilar states.
Confluence: an introductory example
Confluence: an introductory example

- getPen
- writeName

- throw
  - 1/2
  - 1/2

- loseCoin
  - false
  - true
Confluence: an introductory example

- Introduction
- prCRL
- Confluence reduction
- Detecting confluence symbolically
- Case study
- Conclusions

getPen → writeName

1/2 throw → true, false

false → true

true → false
Confluence: an introductory example

- Introduction
- prCRL
- Confluence reduction
- Detecting confluence symbolically
- Case study
- Conclusions

**Confluence:**

- Throw
- Get Pen
- Lose Coin
- Write Name

**Diagram:**

- Lose Coin
- Throw
- True
- False

- Get Pen
- Write Name

**Probabilities:**

- \( \frac{1}{2} \)
Confluence: an introductory example

- **Confluence reduction**
- Detecting confluence symbolically
- Case study
- Conclusions

**Diagram:**
- Nodes labeled with actions: `getPen`, `writeName`, `loseCoin`, `throw`, `true`, `false`
- Probabilities indicated on branches: $\frac{1}{2}$
- Directed edges connecting nodes
Confluence: an introductory example

Confluence is a property of probabilistic models that ensures that no matter the sequence of events, the outcome is the same. In the example above, the process starts with getting a pen, then writing a name, and finally losing a coin. The probability of each action is represented by the numbers next to the edges. The diagram shows that regardless of whether the coin is lost or not, the process leads to the same outcome, illustrating the property of confluence.
Confluence: an introductory example

![Diagram of a probabilistic model showing processes such as getPen, throw, writeName, and loseCoin with associated probabilities.]
Confluence: an introductory example

- Confluence reduction
- Detecting confluence symbolically
- Case study
- Conclusions

PrCRL
Confluence: an introductory example
Confluence: an introductory example
Confluence: an introductory example

Confluence reduction

Detecting confluence symbolically

Case study

Conclusions
Confluence: an introductory example
Three notions of confluence:
- weak confluence
- confluence
- strong confluence
Three notions of confluence:

- weak confluence
- confluence $\Rightarrow$
- strong confluence

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- weak confluence
- confluence
- strong confluence

\[ \Rightarrow \]

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- weak confluence
- confluence
- strong confluence

⇒

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence

Strong confluence
Three notions of confluence:

- weak confluence
- confluence
- strong confluence

⇒

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence

Strong confluence

Strong probabilistic confluence
Three notions of confluence:

- weak confluence
- confluence
- strong confluence

⇒

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- weak confluence
- confluence
- strong confluence

⇒

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence

Strong confluence

Strong probabilistic confluence

Theorem

States that are connected by confluent $\tau$-steps are branching bisimilar.
State space reduction using confluence
State space reduction using confluence
State space reduction using confluence

![Diagram of state space reduction using confluence](link-to-diagram)
State space reduction using confluence
State space reduction using confluence
State space reduction using confluence
State space reduction using confluence
State space reduction using confluence
State space reduction using confluence
Table of Contents

1 Introduction

2 A process algebra with data and probability: prCRL

3 Confluence reduction

4 Detecting confluence symbolically

5 Case study: leader election protocols

6 Conclusions
Detecting confluence symbolically: LPPEs

Example specification

\[ X(pc : \{1..2\}, active : \text{Bool}) = \]
\[
\sum_{n:\{1,2,3\}} pc = 1 \quad \Rightarrow \quad \text{output}(n) \sum_{b:\text{Bool}} \frac{1}{2} : X(2, b)
\]
\[ + \quad pc = 2 \land active \Rightarrow \text{beep} \cdot X(1, active) \]
Detecting confluence symbolically: LPPEs

Example specification

\[ X(pc : \{1..2\}, active : \text{Bool}) = \]
\[ \sum_{n:\{1,2,3\}} pc = 1 \Rightarrow \text{output}(n) \sum_{b:\text{Bool}} \frac{1}{2} : X(2, b) \]
\[ + \quad pc = 2 \land active \Rightarrow \tau \cdot X(1, active) \]
Detecting confluence symbolically: LPPEs

Example specification

\[ X(pc : \{1..2\}, \text{active} : \text{Bool}) = \]
\[
\sum_{n:\{1,2,3\}} \begin{cases} 
pc = 1 & \Rightarrow \text{output}(n) \sum_{b:\text{Bool}} \frac{1}{2} : X(2, b) \\
+ & \text{pc} = 2 \land \text{active} \Rightarrow \tau \cdot X(1, \text{active})
\end{cases}
\]

How to know whether a summand is confluent?
Detecting confluence symbolically: LPPEs

Example specification

\[
X(pc : \{1..2\}, \text{active} : \text{Bool}) = \\
\sum_{n:\{1,2,3\}} pc = 1 \quad \Rightarrow \quad \text{output}(n) \sum \frac{1}{2} : X(2, b) \\
+ pc = 2 \land \text{active} \Rightarrow \quad \tau \cdot X(1, \text{active})
\]

How to know whether a summand is confluent?

- Its action should be \(\tau\)
Detecting confluence symbolically: LPPEs

Example specification

\[
X(pc : \{1..2\}, \text{active} : \text{Bool}) =
\sum_{n:\{1,2,3\}} pc = 1 \Rightarrow \text{output}(n)\sum_{b:\text{Bool}} \frac{1}{2} : X(2, b)
\]
\[
+ pc = 2 \land \text{active} \Rightarrow \tau \cdot X(1, \text{active})
\]

How to know whether a summand is confluent?

- Its action should be \(\tau\)
- Its next state should be chosen nonprobabilistically
Example specification

\[ X(pc : \{1..2\}, active : \text{Bool}) = \]

\[ \sum_{n:\{1,2,3\}} pc = 1 \Rightarrow \text{output}(n) \sum_{b:\text{Bool}} \frac{1}{2} : X(2, b) \]

\[ + \quad pc = 2 \land active \Rightarrow \tau \cdot X(1, active) \]

How to know whether a summand is confluent?

- Its action should be \( \tau \)
- Its next state should be chosen nonprobabilistically
- It should commute with all the other summands
Symbolic detection of confluence

$$X(g : G) = \sum_{d_i : D_i} c_i \Rightarrow a_i \sum_{e_i : E_i} f_i : X(n_i)$$

... + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)$$

Two summands $i, j$ commute if

Two summands $i, j$ commute if

Two summands $i, j$ commute if

Two summands $i, j$ commute if

Two summands $i, j$ commute if

Two summands $i, j$ commute if

Two summands $i, j$ commute if

Two summands $i, j$ commute if

Two summands $i, j$ commute if
Symbolic detection of confluence

\[ X(g : G) = \sum_{d_i : D_i} c_i \Rightarrow a_i \sum_{e_i : E_i} f_i : X(n_i) \]
\[ + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j) \]

Two summands \( i, j \) commute if \( \forall g, d_i, d_j, e_i, e_j : \)
Symbolic detection of confluence

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

\[ + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j) \]

Two summands \( i, j \) commute if \( \forall g, d_i, d_j, e_i, e_j : \)

\[ (c_i(g, d_i) \land c_j(g, d_j)) \rightarrow \]
Symbolic detection of confluence

\[
X(g : G) = \sum_{d_i : D_i} c_i \Rightarrow a_i \sum_{e_i : E_i} f_i : X(n_i) \\
\ldots \\
+ \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
\]

Two summands \(i, j\) commute if \(\forall g, d_i, d_j, e_i, e_j:\)

\[
(c_i(g, d_i) \land c_j(g, d_j)) \rightarrow (i = j \land n_i(g, d_i, e_i) = n_j(g, d_j, e_j))
\]
Symbolic detection of confluence

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

\[ + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j) \]

Two summands \( i, j \) commute if \( \forall g, d_i, d_j, e_i, e_j : \)

\[ (c_i(g, d_i) \land c_j(g, d_j)) \rightarrow (i = j \land n_i(g, d_i, e_i) = n_j(g, d_j, e_j)) \]

\[ \lor \]

\[ \begin{pmatrix}
  \end{pmatrix} \]
Symbolic detection of confluence

\[ X(g : G) = \sum_{d_i : D_i} c_i \Rightarrow a_i \sum_{e_i : E_i} f_i : X(n_i) \]
\[ + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j) \]

Two summands \( i, j \) commute if \( \forall g, d_i, d_j, e_i, e_j : \)

\[ (c_i(g, d_i) \land c_j(g, d_j)) \rightarrow (i = j \land n_i(g, d_i, e_i) = n_j(g, d_j, e_j)) \]

\[ \lor \]

\[ c_j(n_i(g, d_i, e_i), d_j) \]
Symbolic detection of confluence

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

\[ \ldots \]

\[ + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j) \]

Two summands \( i, j \) commute if \( \forall g, d_i, d_j, e_i, e_j : \)

\[ (c_i(g, d_i) \land c_j(g, d_j)) \rightarrow (i = j \land n_i(g, d_i, e_i) = n_j(g, d_j, e_j)) \]

\[ \lor \]

\[ c_j(n_i(g, d_i, e_i), d_j) \land c_i(n_j(g, d_j, e_j), d_i) \]
Symbolic detection of confluence

\[ X(g : G) = \sum_{d_i : D_i} c_i \Rightarrow a_i \sum_{e_i : E_i} f_i : X(n_i) \]
\[ + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j) \]

Two summands \( i, j \) commute if \( \forall g, d_i, d_j, e_i, e_j : \)

\[ (c_i(g, d_i) \land c_j(g, d_j)) \rightarrow (i = j \land n_i(g, d_i, e_i) = n_j(g, d_j, e_j)) \]

\( \lor \)

\[ \left( c_j(n_i(g, d_i, e_i), d_j) \land c_i(n_j(g, d_j, e_j), d_i) \right) \]
\[ \land a_j(g, d_j) = a_j(n_i(g, d_i, e_i), d_j) \]
Symbolic detection of confluence

\[ X(g : G) = \sum_{d_i : D_i} c_i \Rightarrow a_i \sum_{e_i : E_i} f_i : X(n_i) \]
\[ + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j) \]

Two summands \( i, j \) commute if \( \forall g, d_i, d_j, e_i, e_j : \)

\[ (c_i(g, d_i) \land c_j(g, d_j)) \rightarrow (i = j \land n_i(g, d_i, e_i) = n_j(g, d_j, e_j)) \]

\[ \land \]

\[ \left( \begin{array}{l}
  c_j(n_i(g, d_i, e_i), d_j) \land c_i(n_j(g, d_j, e_j), d_i) \\
  \land a_j(g, d_j) = a_j(n_i(g, d_i, e_i), d_j) \\
  \land n_j(n_i(g, d_i, e_i), d_j, e_j) = n_i(n_j(g, d_j, e_j), d_i, e_i)
\end{array} \right) \]
Symbolic detection of confluence

\[
X(g : G) = \sum_{d_i : D_i} c_i \Rightarrow a_i \sum_{e_i : E_i} f_i : X(n_i) \\
\ldots \\
+ \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
\]

Two summands \( i, j \) commute if \( \forall g, d_i, d_j, e_i, e_j : \)

\[
(c_i(g, d_i) \land c_j(g, d_j)) \rightarrow (i = j \land n_i(g, d_i, e_i) = n_j(g, d_j, e_j))
\]

\[
\land \\
c_j(n_i(g, d_i, e_i), d_j) \land c_i(n_j(g, d_j, e_j), d_i) \\
\land a_j(g, d_j) = a_j(n_i(g, d_i, e_i), d_j) \\
\land n_j(n_i(g, d_i, e_i), d_j, e_j) = n_i(n_j(g, d_j, e_j), d_i, e_i) \\
\land f_j(g, d_j, e_j) = f_j(n_i(g, d_i, e_i), d_j, e_j)
\]
Heuristics for detecting confluence

Heuristics for verifying the previous formula for summands $i,j$: 

$\text{Heuristics for verifying the previous formula for summands } i,j:$

$i$: $\text{pc} = 3 \Rightarrow \tau \cdot X(pc := 4)$

$j$: $\text{pc} = 5 \Rightarrow \text{send}(y) \cdot X(pc := 1)$

Both summands use variables that are changed by the other

$i$: $\text{pc}_1 = 2 \land x > 5 \land y > 2 \Rightarrow \tau \cdot X(pc_1 := 3, x := 0)$

$j$: $\text{pc}_2 = 1 \land y > 2 \Rightarrow \text{send}(y) \cdot X(pc_2 := 2)$

$i = j$ and this summand only produces one transition per state

$i$: $\text{pc} = 1 \Rightarrow \tau \cdot X(pc := 2)$

$\text{UNIVERSITY OF TWENTE.}$

Symbolic reductions of probabilistic models

January 18, 2011 23 / 29
Heuristics for detecting confluence

Heuristics for verifying the previous formula for summands $i, j$:

- The conditions of $i$ and $j$ are disjoint
Heuristics for verifying the previous formula for summands $i, j$:

- The conditions of $i$ and $j$ are disjoint
  
  \begin{align*}
  i: \quad pc = 3 &\Rightarrow \tau \cdot X(pc := 4) \\
  j: \quad pc = 5 &\Rightarrow \text{send}(y) \cdot X(pc := 1)
  \end{align*}
Heuristics for verifying the previous formula for summands $i, j$:

- The conditions of $i$ and $j$ are disjoint
  
  $i$: $pc = 3 \Rightarrow \tau \cdot X(pc := 4)$
  
  $j$: $pc = 5 \Rightarrow \text{send}(y) \cdot X(pc := 1)$

- Neither summand uses variables that are changed by the other
Heuristics for verifying the previous formula for summands $i, j$:

- The conditions of $i$ and $j$ are disjoint
  
  $i$: $pc = 3 \Rightarrow \tau \cdot X(pc := 4)$
  
  $j$: $pc = 5 \Rightarrow send(y) \cdot X(pc := 1)$

- Neither summand uses variables that are changed by the other
  
  $i$: $pc1 = 2 \land x > 5 \land y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$
  
  $j$: $pc2 = 1 \land y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$
Heuristics for detecting confluence

Heuristics for verifying the previous formula for summands $i, j$:

- The conditions of $i$ and $j$ are disjoint
  
  $i$: $pc = 3 \Rightarrow \tau \cdot X(pc := 4)$
  
  $j$: $pc = 5 \Rightarrow send(y) \cdot X(pc := 1)$

- Neither summand uses variables that are changed by the other
  
  $i$: $pc1 = 2 \land x > 5 \land y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$
  
  $j$: $pc2 = 1 \land y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$
Heuristics for verifying the previous formula for summands $i, j$:

- **The conditions of $i$ and $j$ are disjoint**
  
  \[ i: \quad pc = 3 \implies \tau \cdot X(pc := 4) \]
  
  \[ j: \quad pc = 5 \implies \text{send}(y) \cdot X(pc := 1) \]

- **Neither summand uses variables that are changed by the other**
  
  \[ i: \quad pc1 = 2 \land x > 5 \land y > 2 \implies \tau \cdot X(pc1 := 3, x := 0) \]
  
  \[ j: \quad pc2 = 1 \land y > 2 \implies \text{send}(y) \cdot X(pc2 := 2) \]
Heuristics for detecting confluence

Heuristics for verifying the previous formula for summands $i, j$:

- The conditions of $i$ and $j$ are disjoint
  
  $i$: $pc = 3 \Rightarrow \tau \cdot X(pc := 4)$
  
  $j$: $pc = 5 \Rightarrow send(y) \cdot X(pc := 1)$

- Neither summand uses variables that are changed by the other
  
  $i$: $pc1 = 2 \land x > 5 \land y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$
  
  $j$: $pc2 = 1 \land y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$

- $i = j$ and this summand only produces one transition per state
Heuristics for detecting confluence

Heuristics for verifying the previous formula for summands $i, j$:

- The conditions of $i$ and $j$ are disjoint
  
  $i$: $pc = 3 \Rightarrow \tau \cdot X(pc := 4)$
  
  $j$: $pc = 5 \Rightarrow send(y) \cdot X(pc := 1)$

- Neither summand uses variables that are changed by the other
  
  $i$: $pc1 = 2 \land x > 5 \land y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$
  
  $j$: $pc2 = 1 \land y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$

- $i = j$ and this summand only produces one transition per state
  
  $i$: $pc = 1 \Rightarrow \tau \cdot X(pc := 2)$
Table of Contents

1. Introduction
2. A process algebra with data and probability: prCRL
3. Confluence reduction
4. Detecting confluence symbolically
5. Case study: leader election protocols
6. Conclusions
Case study: leader election protocols

Basic leader election protocol

- **Two processes** each throw a die
- They *synchronously* communicate the results
- The one that threw highest wins
- In case of a tie: start over again
## Case study: leader election protocols

### Basic leader election protocol
- **Two processes** each throw a die
- They *synchronously* communicate the results
- The one that threw highest wins
- In case of a tie: start over again

### More advanced leader election protocol
- **Several processes** each throw a die
- They *asynchronously* communicate the results
- The one that threw highest wins
- In case of a tie: continue with those processes
## Applying confluence to the protocols

<table>
<thead>
<tr>
<th>Specification</th>
<th>Original</th>
<th>Reduced</th>
<th>Runtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>States</td>
<td>Trans.</td>
<td>Before</td>
</tr>
<tr>
<td>basicOriginal</td>
<td>3,763</td>
<td>6,158</td>
<td>0.45</td>
</tr>
<tr>
<td>basicReduced</td>
<td>1,693</td>
<td>2,438</td>
<td>0.22</td>
</tr>
<tr>
<td>leader-3-12</td>
<td>161,803</td>
<td>268,515</td>
<td>67.37</td>
</tr>
<tr>
<td>leader-3-15</td>
<td>311,536</td>
<td>515,328</td>
<td>145.17</td>
</tr>
<tr>
<td>leader-3-18</td>
<td>533,170</td>
<td>880,023</td>
<td>277.08</td>
</tr>
<tr>
<td>leader-3-21</td>
<td>840,799</td>
<td>1,385,604</td>
<td>817.67</td>
</tr>
<tr>
<td>leader-3-24</td>
<td>1,248,517</td>
<td>2,055,075</td>
<td>1069.71</td>
</tr>
<tr>
<td>leader-3-27</td>
<td>out of memory</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>leader-4-5</td>
<td>443,840</td>
<td>939,264</td>
<td>206.56</td>
</tr>
<tr>
<td>leader-4-6</td>
<td>894,299</td>
<td>1,880,800</td>
<td>429.87</td>
</tr>
<tr>
<td>leader-4-7</td>
<td>1,622,682</td>
<td>3,397,104</td>
<td>1658.38</td>
</tr>
<tr>
<td>leader-4-8</td>
<td>out of memory</td>
<td></td>
<td>–</td>
</tr>
<tr>
<td>leader-5-2</td>
<td>208,632</td>
<td>561,630</td>
<td>125.78</td>
</tr>
<tr>
<td>leader-5-3</td>
<td>1,390,970</td>
<td>3,645,135</td>
<td>1504.33</td>
</tr>
<tr>
<td>leader-5-4</td>
<td>out of memory</td>
<td></td>
<td>–</td>
</tr>
</tbody>
</table>
## Appyling confluenec to the protocols

<table>
<thead>
<tr>
<th>Specification</th>
<th>Original States</th>
<th>Trans.</th>
<th>Reduced States</th>
<th>Trans.</th>
<th>Runtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>basicOriginal</td>
<td>3,763</td>
<td>6,158</td>
<td>631</td>
<td>758</td>
<td>0.45</td>
</tr>
<tr>
<td>basicReduced</td>
<td>1,693</td>
<td>2,438</td>
<td>541</td>
<td>638</td>
<td>0.22</td>
</tr>
<tr>
<td>leader-3-12</td>
<td>161,803</td>
<td>268,515</td>
<td>35,485</td>
<td>41,829</td>
<td>67.37</td>
</tr>
<tr>
<td>leader-3-15</td>
<td>311,536</td>
<td>515,328</td>
<td>68,926</td>
<td>80,838</td>
<td>145.17</td>
</tr>
<tr>
<td>leader-3-18</td>
<td>533,170</td>
<td>880,023</td>
<td>118,675</td>
<td>138,720</td>
<td>277.08</td>
</tr>
<tr>
<td>leader-3-21</td>
<td>840,799</td>
<td>1,385,604</td>
<td>187,972</td>
<td>219,201</td>
<td>817.67</td>
</tr>
<tr>
<td>leader-3-24</td>
<td>1,248,517</td>
<td>2,055,075</td>
<td>280,057</td>
<td>326,007</td>
<td>1069.71</td>
</tr>
<tr>
<td>leader-3-27</td>
<td>out of memory</td>
<td></td>
<td>398,170</td>
<td>462,864</td>
<td>503.85</td>
</tr>
<tr>
<td>leader-4-5</td>
<td>443,840</td>
<td>939,264</td>
<td>61,920</td>
<td>92,304</td>
<td>206.56</td>
</tr>
<tr>
<td>leader-4-6</td>
<td>894,299</td>
<td>1,880,800</td>
<td>127,579</td>
<td>188,044</td>
<td>429.87</td>
</tr>
<tr>
<td>leader-4-7</td>
<td>1,622,682</td>
<td>3,397,104</td>
<td>235,310</td>
<td>344,040</td>
<td>1658.38</td>
</tr>
<tr>
<td>leader-4-8</td>
<td>out of memory</td>
<td></td>
<td>400,125</td>
<td>581,468</td>
<td>653.60</td>
</tr>
<tr>
<td>leader-5-2</td>
<td>208,632</td>
<td>561,630</td>
<td>14,978</td>
<td>29,420</td>
<td>125.78</td>
</tr>
<tr>
<td>leader-5-3</td>
<td>1,390,970</td>
<td>3,645,135</td>
<td>112,559</td>
<td>208,170</td>
<td>1504.33</td>
</tr>
<tr>
<td>leader-5-4</td>
<td>out of memory</td>
<td></td>
<td>472,535</td>
<td>847,620</td>
<td>7171.73</td>
</tr>
</tbody>
</table>

Number of states: **−85%**

Number of transitions: **−90%**
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
We developed the process algebra prCRL, incorporating both data and probability, including a normal form (the LPPE) as starting point for symbolic optimisations.

We developed three new notions of confluence for PAs that preserve branching probabilistic bisimulation.

We showed how these notions can be used for state space reduction (even in the presence of $\tau$-loops).

We discussed how to detect the strongest notion symbolically.

We illustrated the power of our methods using a case study.
We developed the process algebra prCRL, incorporating both data and probability, including a normal form (the LPPE) as starting point for symbolic optimisations.

We developed three new notions of confluence for PAs that preserve branching probabilistic bisimulation.

We showed how these notions can be used for state space reduction (even in the presence of $\tau$-loops).

We discussed how to detect the strongest notion symbolically.

We illustrated the power of our methods using a case study.

Questions