UNIVERSITY OF TWENTE.

Formal Methods & Tools.

Symbolic reductions of probabilistic models using linear process equations

Mark Timmer

February 25, 2010

5^{th} QUASIMODO meeting

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

Contents

Contents

Probabilistic Model Checking

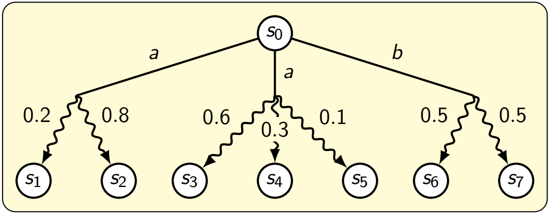
Probabilistic model checking:

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

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

Applications:

- Dependability analysis
- Performance analysis

Limitations and Solutions

Limitations of previous approaches:

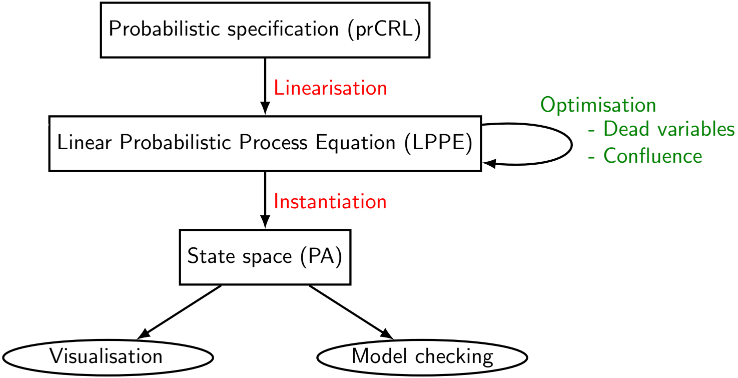
- Susceptible to the state space explosion problem
- Restricted treatment of data

Limitations and Solutions

Limitations of previous approaches:

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

Our approach:



Overview of our approach

Our approach:

1. Specify systems in prCRL: a probabilistic process algebra incorporating both data types and probabilistic choice.
2. Transform specifications to LPPEs: a linear format enabling symbolic optimisations at the language level
3. Reduce state spaces before they are generated by manipulations of the linear format.

Overview of our approach

Our approach:

1. Specify systems in prCRL: a probabilistic process algebra incorporating both data types and probabilistic choice.
2. Transform specifications to LPPEs: a linear format enabling symbolic optimisations at the language level
3. Reduce state spaces before they are generated by manipulations of the linear format: confluence reduction.

Contents

A process algebra with data and probability: prCRL

Specification language prCRL:

- Based on μCRL (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 μCRL (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:

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

Process equations and processes

A process equation is something of the form $X(\vec{g} : \vec{G}) = p$.

An example specification

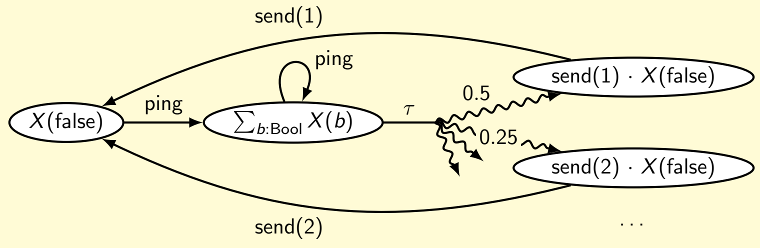
Sending an arbitrary natural number

$$
\begin{aligned}
X(\text{active} : \text{Bool}) = {} & \\
& \text{not}(\text{active}) \Rightarrow \text{ping} \cdot \sum_{b:\text{Bool}} X(b) \\
+ \text{active} \quad & \Rightarrow \tau \sum\!\!\!\bullet_{n:\mathbb{N}^{>0}} \frac{1}{2^n} : \left(\text{send}(n) \cdot X(\text{false})\right)
\end{aligned}
$$

An example specification

Sending an arbitrary natural number

$$
\begin{aligned}
X(\text{active} : \text{Bool}) = \\
&\text{not}(\text{active}) \Rightarrow \text{ping} \cdot \sum_{b:\text{Bool}} X(b) \\
+\ \text{active} \quad &\Rightarrow \tau \sum\!\!\!\!\bullet_{n:\mathbb{N}^{>0}} \tfrac{1}{2^n} : \Big(\text{send}(n) \cdot X(\text{false})\Big)
\end{aligned}
$$



Compositionality using extended prCRL

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

Compositionality using extended prCRL

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

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

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

Compositionality using extended prCRL

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

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

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

$$Z = \qquad (X(1) \,||\, Y(2))$$

Compositionality using extended prCRL

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

$$X(n : \{1,2\}) = \mathrm{write}_X(n) \cdot X(n) + \mathrm{choose} \sum_{n' : \{1,2\}}^{\bullet} \frac{1}{2} : X(n')$$

$$Y(m : \{1,2\}) = \mathrm{write}_Y(m^2) \cdot Y(m) + \mathrm{choose}' \sum_{m' : \{1,2\}}^{\bullet} \frac{1}{2} : Y(m')$$

$$Z = \qquad (X(1) \,||\, Y(2))$$

$$\gamma(\mathrm{choose}, \mathrm{choose}') = \mathrm{chooseTogether}$$

Compositionality using extended prCRL

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

$$X(n : \{1,2\}) = \mathrm{write}_X(n) \cdot X(n) + \mathrm{choose} \sum_{n' : \{1,2\}}\!\!\bullet\ \frac{1}{2} : X(n')$$

$$Y(m : \{1,2\}) = \mathrm{write}_Y(m^2) \cdot Y(m) + \mathrm{choose}' \sum_{m' : \{1,2\}}\!\!\bullet\ \frac{1}{2} : Y(m')$$

$$Z = \partial_{\{\mathrm{choose},\mathrm{choose}'\}}(X(1) \mid\mid Y(2))$$

$$\gamma(\mathrm{choose}, \mathrm{choose}') = \mathrm{chooseTogether}$$

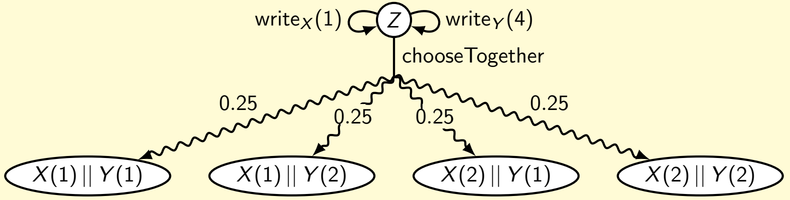
Compositionality using extended prCRL

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

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

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

$$Z = \partial_{\{\text{choose},\text{choose}'\}}(X(1) \mid\mid Y(2))$$
$$\gamma(\text{choose}, \text{choose}') = \text{chooseTogether}$$



Contents

A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

$$
\begin{aligned}
X(\vec{g} : \vec{G}) = & \sum_{\vec{d_1}:\vec{D_1}} c_1 \Rightarrow a_1(b_1) \sum\!\!\!\bullet_{\vec{e_1}:\vec{E_1}} f_1 : X(\vec{n_1}) \\
& \dots \\
+ & \sum_{\vec{d_k}:\vec{D_k}} c_k \Rightarrow a_k(b_k) \sum\!\!\!\bullet_{\vec{e_k}:\vec{E_k}} f_k : X(\vec{n_k})
\end{aligned}
$$

- $\vec{G}$ is a type for state vectors
- $\vec{D_i}$ a type for local variable vectors for summand i
- c_i is the enabling condition of summand i
- a_i is an atomic action, with action-parameter vector b_i
- $\vec{n_i}$ is the next-state vector of summand i.
- $\vec{E_i}$ a type for the probabilistic variable for summand i
- f_i is the probability distribution of summand i

Advantages of LPPEs

Advantages of using LPPEs instead of prCRL specifications:

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

Advantages of LPPEs

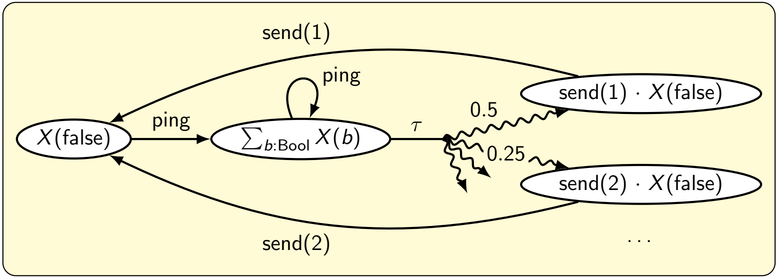
Advantages of using LPPEs instead of prCRL specifications:

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

Theorem

Every specification S (without unguarded recursion) can be linearised to an LPPE S' in such a way that S and S' are strongly probabilistic bisimilar.

Linear probabilistic process equations – An example



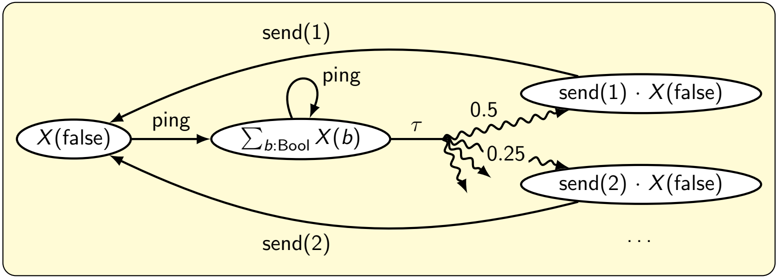
Specification in prCRL

$$X(\text{active} : \text{Bool}) =$$

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

$$+ \text{ active} \Rightarrow \tau \sum\!\!\!\bullet_{n:\mathbb{N}^{>0}} \frac{1}{2^n} : \text{send}(n) \cdot X(\text{false})$$

Linear probabilistic process equations – An example



Specification in prCRL

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

Specification in LPPE

$$X(pc : \{1..3\}, n : \mathbb{N}^{\geq 0}) =$$
$$+ \; pc = 1 \Rightarrow \text{ping} \cdot X(2, 1)$$
$$+ \; pc = 2 \Rightarrow \text{ping} \cdot X(2, 1)$$
$$+ \; pc = 2 \Rightarrow \tau \sum_{n:\mathbb{N}^{>0}} \bullet \; \frac{1}{2^n} : X(3, n)$$
$$+ \; pc = 3 \Rightarrow \text{send}(n) \cdot X(1, 1)$$

Contents

Case study: a leader election protocol

- Implementation in Haskell:
  - Linearisation: from prCRL to LPPE
  - Parallel composition of LPPEs, hiding, renaming, encapsulation
  - Generation of the state space of an LPPE
  - Automatic constant elimination and summand simplification
- Manual dead variable reduction

Case study: a leader election protocol

- Implementation in Haskell:
  - Linearisation: from prCRL to LPPE
  - Parallel composition of LPPEs, hiding, renaming, encapsulation
  - Generation of the state space of an LPPE
  - Automatic constant elimination and summand simplification
- Manual dead variable reduction

Case study

Leader election protocol à la Itai-Rodeh

- Two processes throw a die
  - *One of them throws a 6 $\rightarrow$ this will be the leader*
  - *Both throw 6 or neither throws 6 $\rightarrow$ throw again*

Case study: a leader election protocol

- Implementation in Haskell:
  - Linearisation: from prCRL to LPPE
  - Parallel composition of LPPEs, hiding, renaming, encapsulation
  - Generation of the state space of an LPPE
  - Automatic constant elimination and summand simplification
- Manual dead variable reduction

Case study

Leader election protocol à la Itai-Rodeh

- Two processes throw a die
  - *One of them throws a 6 → this will be the leader*
  - *Both throw 6 or neither throws 6 → throw again*
- More precise:
  - *Passive thread: receive value of opponent*
  - *Active thread: roll, send, compare (or block)*

A prCRL model of the leader election protocol

$$P(id : \{1, 2\}, val : Die, set : Bool) =$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
&P(id : \{1,2\}, val : Die, set : Bool) = \\
&\quad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true))
\end{aligned}
$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
P(id : \{1, 2\}, val : Die, set : Bool) = \\
&\quad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
&+ set = true \Rightarrow getVal(val) \cdot P(id, val, false)
\end{aligned}
$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
&P(id : \{1,2\}, val : Die, set : Bool) = \\
&\qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
&\quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
&A(id : \{1,2\}) =
\end{aligned}
$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
&P(id : \{1,2\}, val : Die, set : Bool) = \\
&\qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
&\quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
&A(id : \{1,2\}) = \\
&\quad roll(id) \sum\nolimits^{\bullet}_{d:Die} \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot
\end{aligned}
$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
&P(id : \{1,2\}, val : Die, set : Bool) = \\
&\qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
&\quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
&A(id : \{1,2\}) = \\
&\quad roll(id) \sum\!\!\!\bullet_{d:Die} \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot
\end{aligned}
$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
&P(id:\{1,2\}, val:Die, set:Bool) = \\
&\qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
&\quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false)) \\
&A(id:\{1,2\}) = \\
&\quad roll(id) \sum_{d:Die}\!\!\bullet\ \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot \\
&\qquad (\ (d = e \vee (d \neq 6 \wedge e \neq 6) \Rightarrow A(id)) \\
&\qquad + (d = 6 \wedge e \neq 6 \Rightarrow leader(id) \cdot A(id)) \\
&\qquad + (e = 6 \wedge d \neq 6 \Rightarrow follower(id) \cdot A(id)))
\end{aligned}
$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
& P(id : \{1,2\}, val : Die, set : Bool) = \\
& \qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
& \quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
& A(id : \{1,2\}) = \\
& \quad roll(id) \sum_{d:Die}\!\!\bullet\ \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot \\
& \qquad (\ (d = e \vee (d \neq 6 \wedge e \neq 6) \Rightarrow A(id)) \\
& \qquad + (d = 6 \wedge e \neq 6 \Rightarrow leader(id) \cdot A(id)) \\
& \qquad + (e = 6 \wedge d \neq 6 \Rightarrow follower(id) \cdot A(id))) \\
& C(id : \{1,2\}) = \qquad P(id, heads, false) \mid\mid A(id)
\end{aligned}
$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
& P(id : \{1,2\}, val : Die, set : Bool) = \\
& \qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
& \quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
& A(id : \{1,2\}) = \\
& \quad roll(id) \sum\!\!\!\bullet_{d:Die} \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot \\
& \qquad \big(\, (d = e \vee (d \neq 6 \wedge e \neq 6) \Rightarrow A(id)) \\
& \qquad + (d = 6 \wedge e \neq 6 \Rightarrow leader(id) \cdot A(id)) \\
& \qquad + (e = 6 \wedge d \neq 6 \Rightarrow follower(id) \cdot A(id))\big) \\
& C(id : \{1,2\}) = \qquad P(id, heads, false) \,||\, A(id)
\end{aligned}
$$

$$\gamma(getVal, readVal) = checkVal$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
& P(id : \{1,2\}, val : Die, set : Bool) = \\
& \qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
& \quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
& A(id : \{1,2\}) = \\
& \quad roll(id) \sum\!\!\!\!\bullet_{d:Die} \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot \\
& \qquad (\ (d = e \vee (d \neq 6 \wedge e \neq 6) \Rightarrow A(id)) \\
& \qquad + (d = 6 \wedge e \neq 6 \Rightarrow leader(id) \cdot A(id)) \\
& \qquad + (e = 6 \wedge d \neq 6 \Rightarrow follower(id) \cdot A(id))) \\
& C(id : \{1,2\}) = \partial_{getVal, readVal}(P(id, heads, false) \,||\, A(id))
\end{aligned}
$$

$$\gamma(getVal, readVal) = checkVal$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
& P(id : \{1,2\}, val : Die, set : Bool) = \\
& \qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
& \quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
& A(id : \{1,2\}) = \\
& \quad roll(id) \sum\!\!\!\bullet_{d:Die} \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot \\
& \qquad (\ (d = e \vee (d \neq 6 \wedge e \neq 6) \Rightarrow A(id)) \\
& \qquad + (d = 6 \wedge e \neq 6 \Rightarrow leader(id) \cdot A(id)) \\
& \qquad + (e = 6 \wedge d \neq 6 \Rightarrow follower(id) \cdot A(id))) \\
& C(id : \{1,2\}) = \partial_{getVal, readVal}(P(id, heads, false) \,||\, A(id)) \\
& S = \qquad C(1) \,||\, C(2)
\end{aligned}
$$

$$\gamma(getVal, readVal) = checkVal$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
&P(id : \{1,2\}, val : Die, set : Bool) = \\
&\qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
&\quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
&A(id : \{1,2\}) = \\
&\quad roll(id) \sum_{d:Die}\!\!\bullet\ \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot \\
&\qquad (\ (d = e \vee (d \neq 6 \wedge e \neq 6) \Rightarrow A(id)) \\
&\qquad + (d = 6 \wedge e \neq 6 \Rightarrow leader(id) \cdot A(id)) \\
&\qquad + (e = 6 \wedge d \neq 6 \Rightarrow follower(id) \cdot A(id))) \\
&C(id : \{1,2\}) = \partial_{getVal, readVal}(P(id, heads, false) \,||\, A(id)) \\
&S = \qquad\qquad C(1) \,||\, C(2)
\end{aligned}
$$

$$\gamma(rec, send) = comm \qquad \gamma(getVal, readVal) = checkVal$$

A prCRL model of the leader election protocol

$$
\begin{aligned}
&P(id : \{1,2\}, val : Die, set : Bool) = \\
&\qquad set = false \Rightarrow \sum_{d:Die} rec(id, other(id), d) \cdot P(id, d, true)) \\
&\quad + set = true \Rightarrow getVal(val) \cdot P(id, val, false) \\
&A(id : \{1,2\}) = \\
&\quad roll(id) \sum_{\bullet\, d:Die} \tfrac{1}{6} : send(other(id), id, d) \cdot \sum_{e:Die} readVal(e) \cdot \\
&\qquad (\ (d = e \vee (d \neq 6 \wedge e \neq 6) \Rightarrow A(id)) \\
&\qquad + (d = 6 \wedge e \neq 6 \Rightarrow leader(id) \cdot A(id)) \\
&\qquad + (e = 6 \wedge d \neq 6 \Rightarrow follower(id) \cdot A(id))) \\
&C(id : \{1,2\}) = \partial_{getVal, readVal}(P(id, heads, false) \,||\, A(id)) \\
&S = \partial_{send, rec}(C(1) \,||\, C(2))
\end{aligned}
$$

$$\gamma(rec, send) = comm \qquad \gamma(getVal, readVal) = checkVal$$

Reductions on the leader election protocol model

In order to obtain reductions first linearise

Reductions on the leader election protocol model

In order to obtain reductions first linearise:

$$\sum_{e21:D} pc21 = 3 \wedge pc11 = 1 \wedge set11 \wedge val11 = e21 \Rightarrow$$
$$checkVal(val11) \sum_{(k1,k2):\{*\}\times\{*\}}^{\bullet} multiply(1.0, 1.0):$$
$$Z(1, id11, val11, false, 1, 4, id21, d21, e21,$$
$$pc12, id12, val12, set12, d12, pc22, id22, d22, e22)$$

Reductions on the leader election protocol model

In order to obtain reductions first linearise:

$$\sum_{e21:D} pc21 = 3 \wedge pc11 = 1 \wedge set11 \wedge val11 = e21 \Rightarrow$$
$$checkVal(val11) \sum_{(k1,k2):\{*\}\times\{*\}}^{\bullet} multiply(1.0, 1.0):$$
$$Z(1, id11, val11, false, 1, 4, id21, d21, e21,$$
$$pc12, id12, val12, set12, d12, pc22, id22, d22, e22)$$

Before reductions:

- 18 parameters
- 14 summands
- 3423 states
- 5478 transitions

Reductions on the leader election protocol model

In order to obtain reductions first linearise:

$$
\begin{aligned}
& pc21 = 3 \wedge \quad set11 \quad \Rightarrow \\
& checkVal(val11) \sum_{(k1,k2):\{*\}\times\{*\}} 1.0 : \\
& Z(\quad val11, false, \quad 4, \quad d21, val11, \\
& \qquad val12, set12, \quad pc22, \quad d22, e22)
\end{aligned}
$$

Before reductions:

- 18 parameters
- 14 summands
- 3423 states
- 5478 transitions

After reductions:

- 10 parameters
- 12 summands

Reductions on the leader election protocol model

In order to obtain reductions first linearise:

$$
\begin{aligned}
&\mathit{pc21} = 3 \wedge \quad \mathit{set11} \quad \Rightarrow \\
&\mathit{checkVal}(\mathit{val11}) \sum_{(k1,k2):\{*\}\times\{*\}} \!\!\!\bullet\; 1.0: \\
&Z(\quad 1 \;, \mathit{false}, \; 4, \quad \mathit{d21}, \mathit{val11}, \\
&\qquad \mathit{val12}, \mathit{set12}, \quad \mathit{pc22}, \quad \mathit{d22}, \mathit{e22})
\end{aligned}
$$

Before reductions:

- 18 parameters
- 14 summands
- 3423 states
- 5478 transitions

After reductions:

- 10 parameters
- 12 summands
- 1613 states (-53%)
- 2278 transitions (-58%)

Contents

Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

Confluence reduction: efficiëntly reducing specifications while preserving branching probabilistic bisimulation.

Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

Confluence reduction: efficiëntly reducing specifications while preserving branching probabilistic bisimulation.

Intuition:

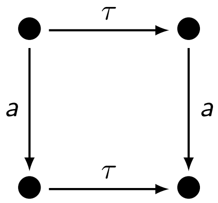
$$\begin{tikzcd} \bullet \arrow[r, "\tau"] \arrow[d, "a"'] & \bullet \arrow[d, "a"] \\ \bullet \arrow[r, "\tau"] & \bullet \end{tikzcd}$$

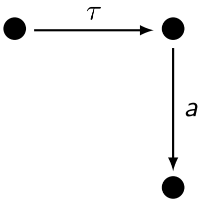
Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

Confluence reduction: efficiëntly reducing specifications while preserving branching probabilistic bisimulation.

Intuition:



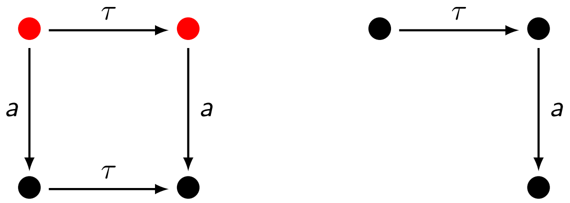


Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

Confluence reduction: efficiëntly reducing specifications while preserving branching probabilistic bisimulation.

Intuition:

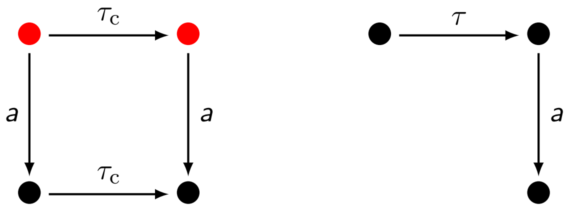


Overview of confluence reduction

Strong probabilistic bisimulation is sometimes too restrictive.

Confluence reduction: efficiëntly reducing specifications while preserving branching probabilistic bisimulation.

Intuition:

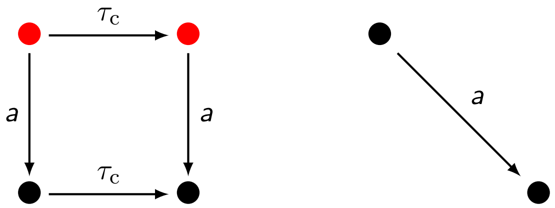


Overview of confluence reduction

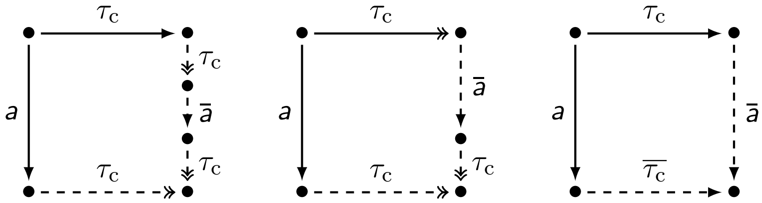
Strong probabilistic bisimulation is sometimes too restrictive.

Confluence reduction: efficiëntly reducing specifications while preserving branching probabilistic bisimulation.

Intuition:

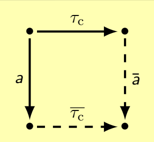


Variants of confluence

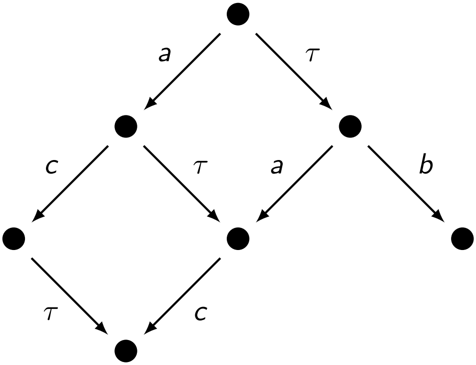


Weak confluence Confluence Strong confluence

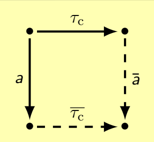
Examples



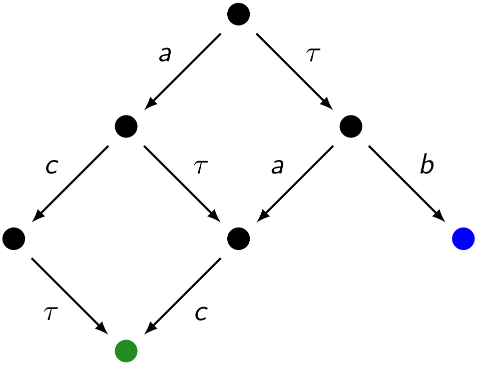
Reduction based on strong confluence



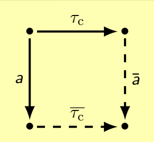
Examples



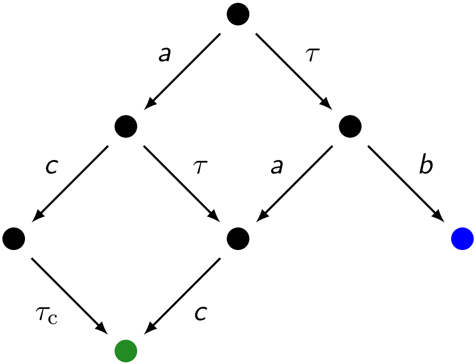
Reduction based on strong confluence



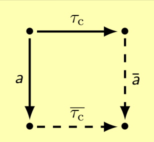
Examples



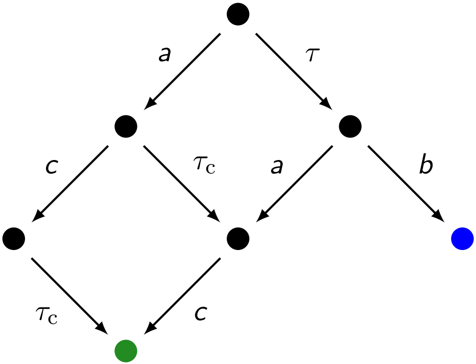
Reduction based on strong confluence



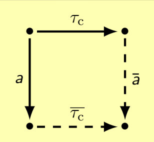
Examples



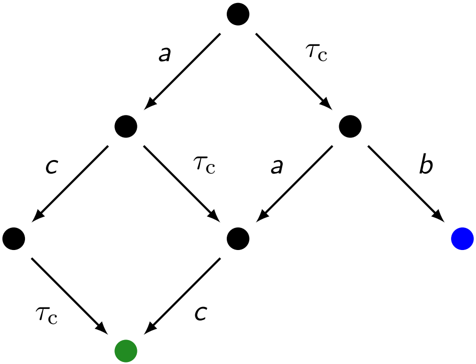
Reduction based on strong confluence



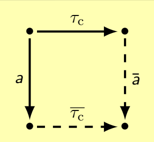
Examples



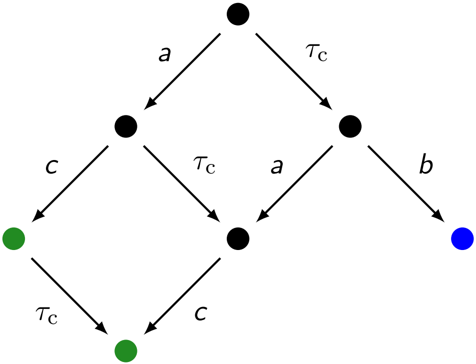
Reduction based on strong confluence



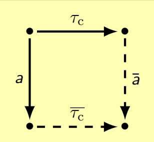
Examples



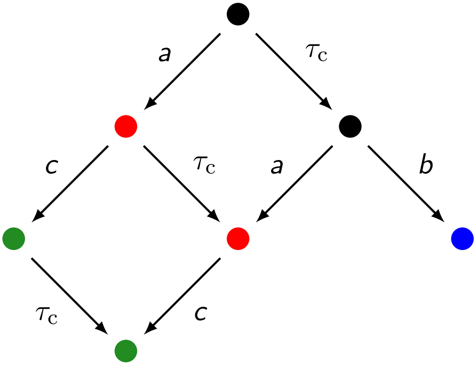
Reduction based on strong confluence



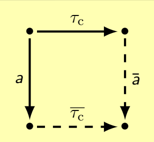
Examples



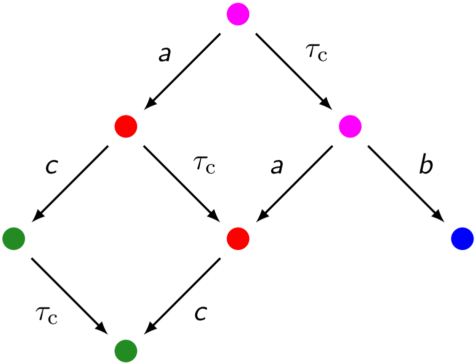
Reduction based on strong confluence



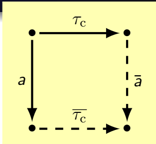
Examples



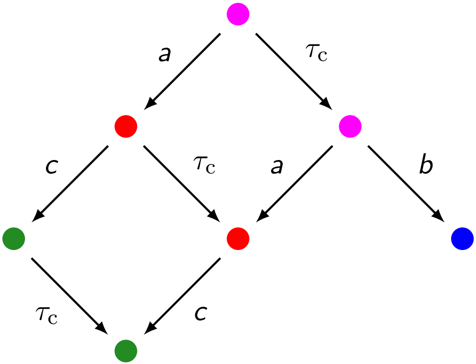
Reduction based on strong confluence

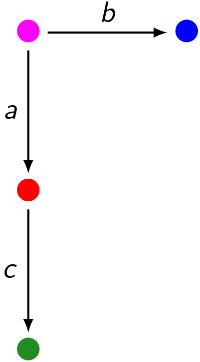


Examples

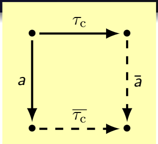


Reduction based on strong confluence

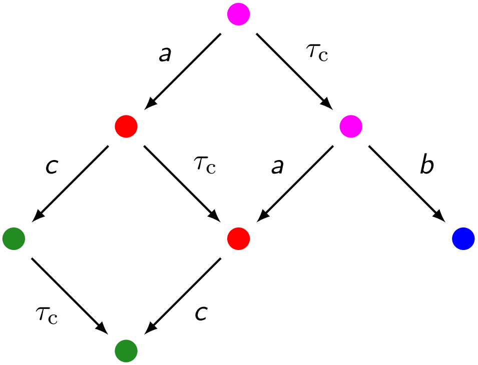


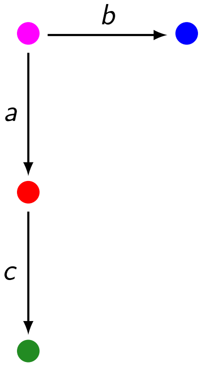


Examples



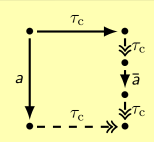
Reduction based on strong confluence



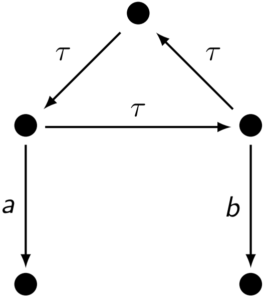


Giving τ_c steps priority works because of the absence of τ_c loops.

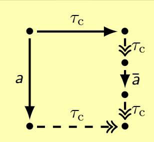
Examples



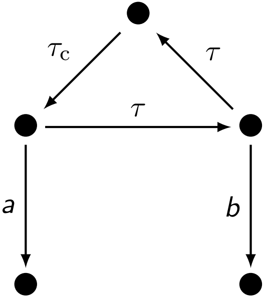
Reduction based on weak confluence



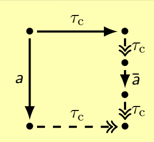
Examples



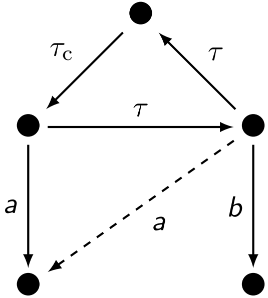
Reduction based on weak confluence



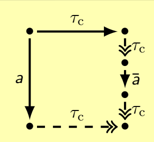
Examples



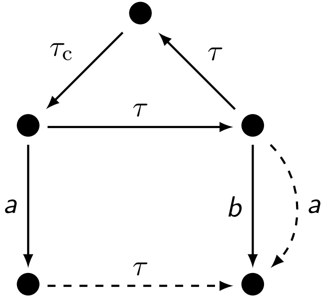
Reduction based on weak confluence



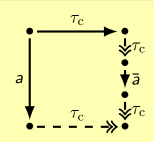
Examples



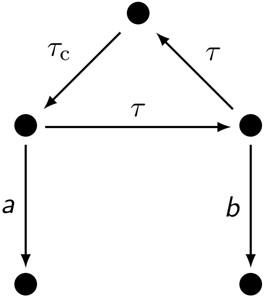
Reduction based on weak confluence



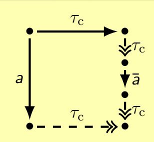
Examples



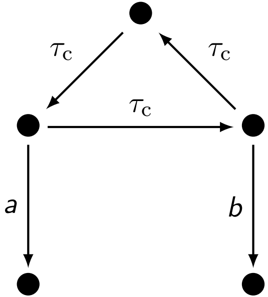
Reduction based on weak confluence



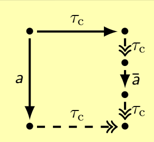
Examples



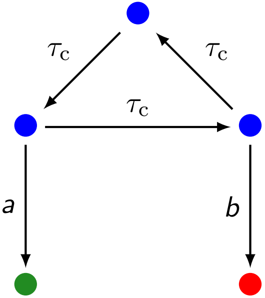
Reduction based on weak confluence



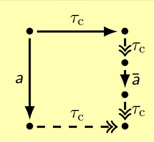
Examples



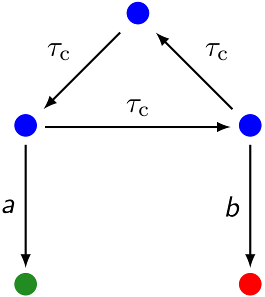
Reduction based on weak confluence

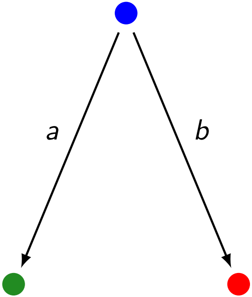


Examples

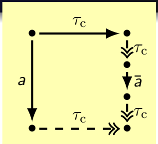


Reduction based on weak confluence

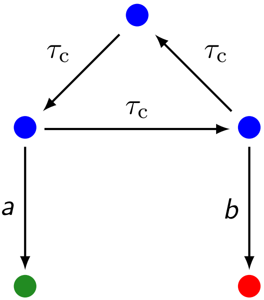


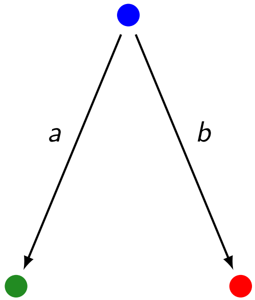


Examples



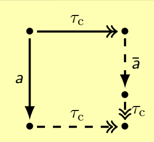
Reduction based on weak confluence



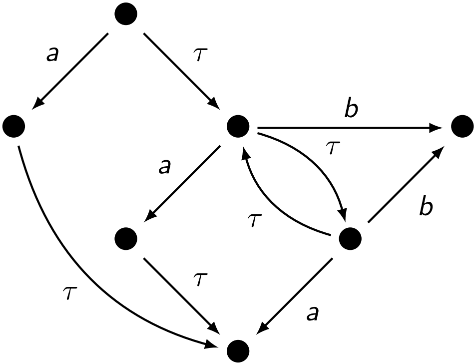


Here we used the equivalence classes of $\mathcal{A}/\overset{\tau_c}{\twoheadleftarrow\!\!\twoheadrightarrow}$ as nodes.
(None of the blue nodes could be chosen as representative, as none of them can done both an a an a b transition.)

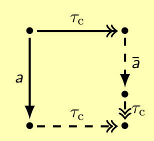
Examples



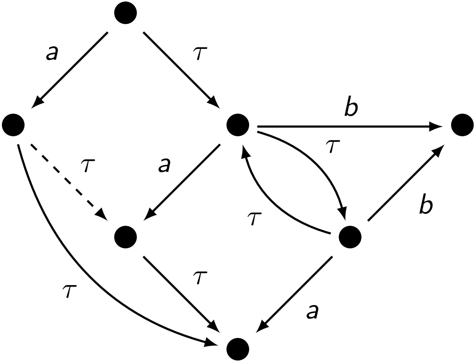
Reduction based on confluence
using representatives



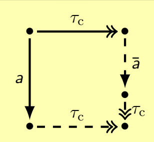
Examples



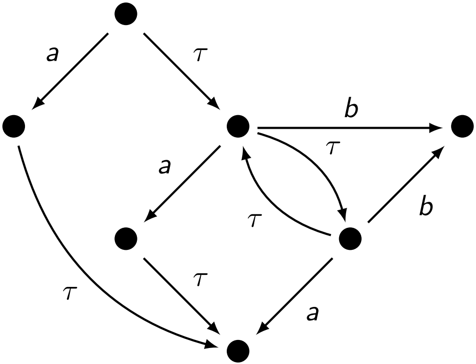
Reduction based on confluence
using representatives



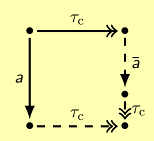
Examples



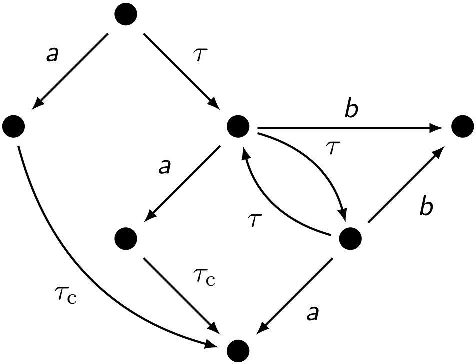
Reduction based on confluence
using representatives



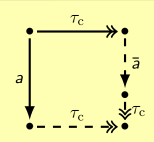
Examples



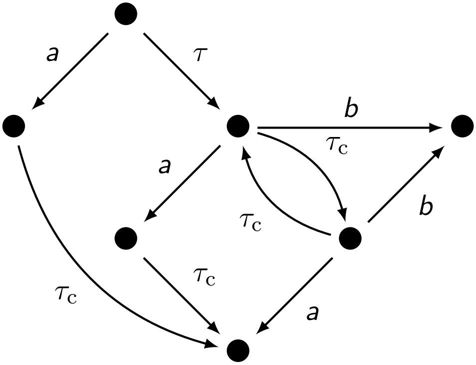
Reduction based on confluence
using representatives



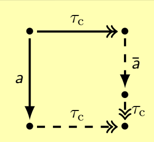
Examples



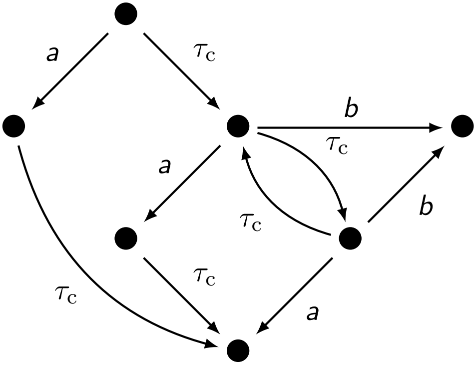
Reduction based on confluence
using representatives



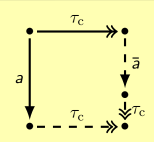
Examples



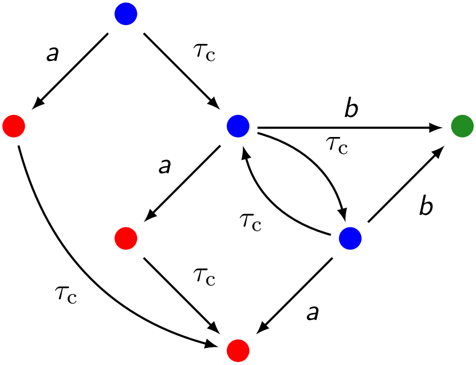
Reduction based on confluence
using representatives



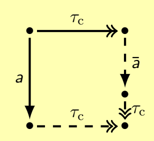
Examples



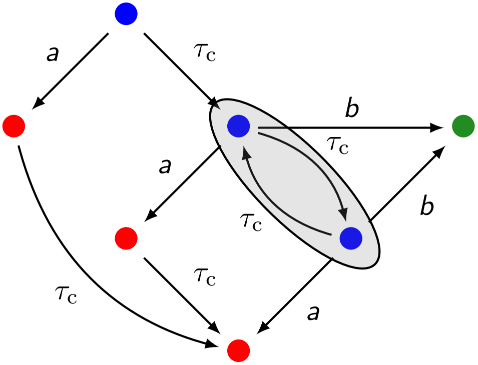
Reduction based on confluence
using representatives



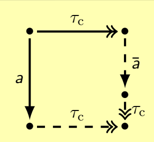
Examples



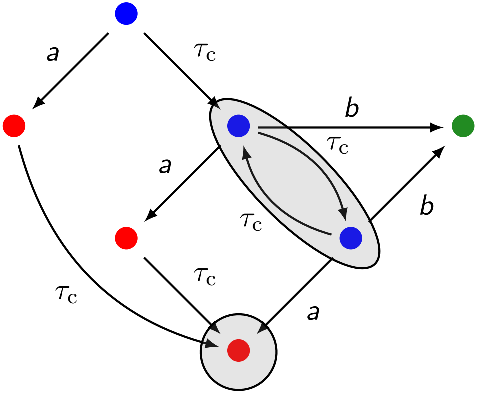
Reduction based on confluence
using representatives



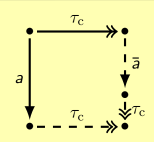
Examples



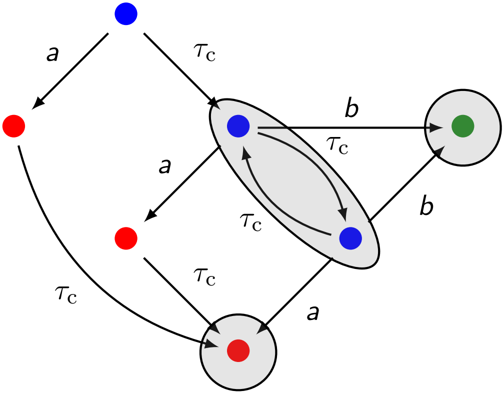
Reduction based on confluence
using representatives



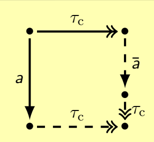
Examples



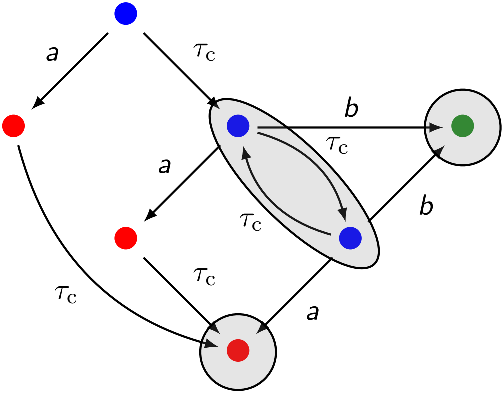
Reduction based on confluence
using representatives

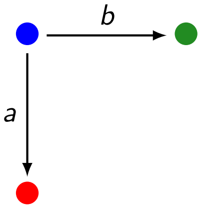


Examples



Reduction based on confluence
using representatives

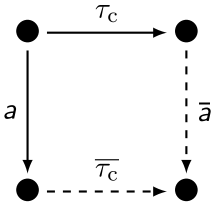




Confluence for probabilistic automata

For simplicity we only consider strong confluence from now on.

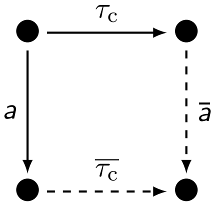
Non-probabilistic:



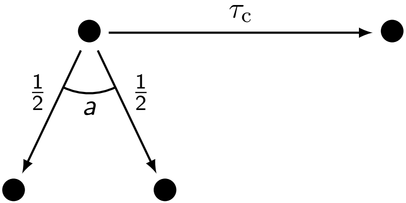
Confluence for probabilistic automata

For simplicity we only consider strong confluence from now on.

Non-probabilistic:



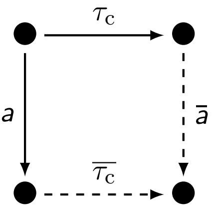
Probabilistic:



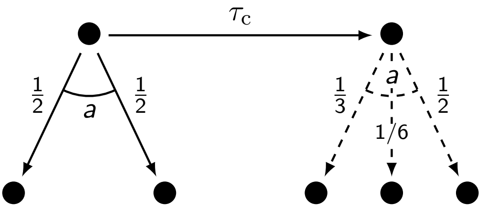
Confluence for probabilistic automata

For simplicity we only consider strong confluence from now on.

Non-probabilistic:



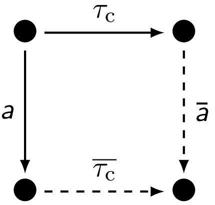
Probabilistic:



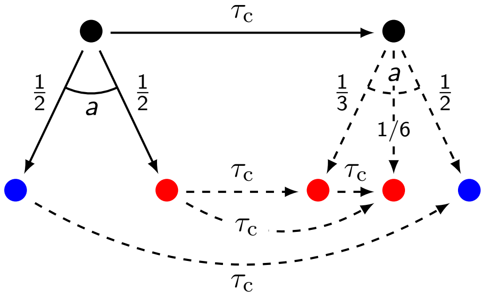
Confluence for probabilistic automata

For simplicity we only consider strong confluence from now on.

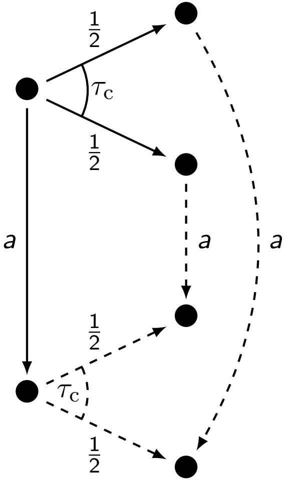
Non-probabilistic:



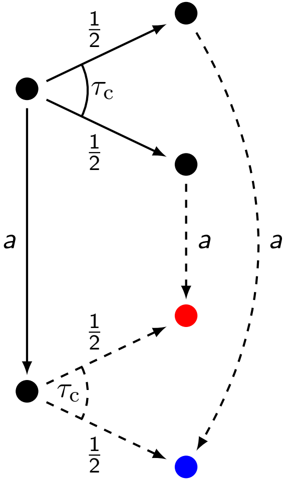
Probabilistic:



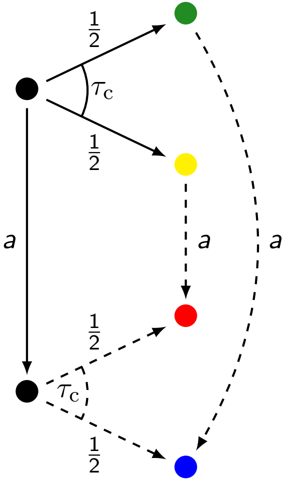
Why τ_c steps should have a Dirac distribution



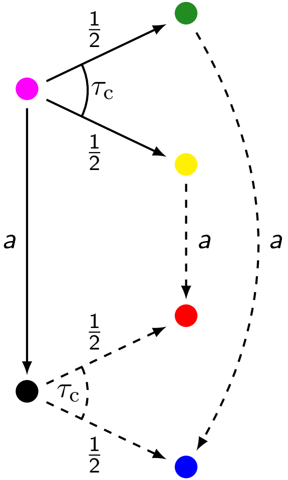
Why τ_c steps should have a Dirac distribution



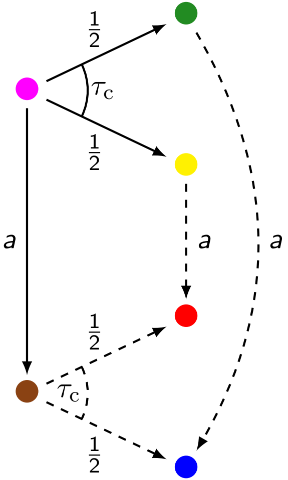
Why τ_c steps should have a Dirac distribution



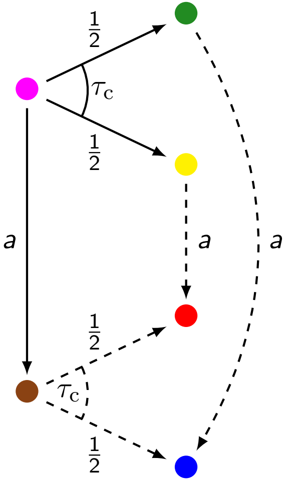
Why τ_c steps should have a Dirac distribution



Why τ_c steps should have a Dirac distribution



Why τ_c steps should have a Dirac distribution



As all states are (potentially) different, no reduction can be obtained.

Detecting confluence using LPPEs

Given an LPPE, confluence can be detected using theorem proving.

$$X(\vec{g} : \vec{G}) = \sum_{\vec{d_1}:\vec{D_1}} c_1 \Rightarrow \tau \sum\!\!\!\!\bullet_{\vec{e_1}:\vec{E_1}} f_1 : X(\vec{n_1})$$
$$\dots$$
$$+ \sum_{\vec{d_k}:\vec{D_k}} c_k \Rightarrow a_k(b_k) \sum\!\!\!\!\bullet_{\vec{e_k}:\vec{E_k}} f_k : X(\vec{n_k})$$

To check the first τ-summand is confluent, we check whether indeed

- $|E_1| = 1$, or $f_1 = 1.0$ for one $e_i \in E_1$.
- the summand is confluent with all other summands.

Detecting confluence using LPPEs

$$
\begin{aligned}
X(\vec{g} : \vec{G}) = & \sum_{\vec{d_1}:\vec{D_1}} c_1 \Rightarrow \tau \cdot X(\vec{n_1}) \\
& \cdots \\
& + \sum_{\vec{d_k}:\vec{D_k}} c_k \Rightarrow a_k(b_k) \sum\!\!\!\!\bullet_{\vec{e_k}:\vec{E_k}} f_k : X(\vec{n_k})
\end{aligned}
$$

Detecting confluence using LPPEs

$$
\begin{aligned}
X(\vec{g} : \vec{G}) = & \sum_{\vec{d_1}:\vec{D_1}} c_1 \Rightarrow \tau \cdot X(\vec{n_1}) \\
& \dots \\
& + \sum_{\vec{d_k}:\vec{D_k}} c_k \Rightarrow a_k(b_k) \sum\!\!\!\bullet_{\vec{e_k}:\vec{E_k}} f_k : X(\vec{n_k})
\end{aligned}
$$

To prove:

$$c_1(g, d_1) \wedge c_k(g, d_k) \quad \rightarrow$$

Detecting confluence using LPPEs

$$
\begin{aligned}
X(\vec{g} : \vec{G}) = & \sum_{\vec{d_1}:\vec{D_1}} c_1 \Rightarrow \tau \cdot X(\vec{n_1}) \\
& \dots \\
& + \sum_{\vec{d_k}:\vec{D_k}} c_k \Rightarrow a_k(b_k) \sum\!\!\!\!\bullet_{\vec{e_k}:\vec{E_k}} f_k : X(\vec{n_k})
\end{aligned}
$$

To prove:

$$
c_1(g, d_1) \wedge c_k(g, d_k) \ \ \rightarrow
$$

$$
\left(\quad c_k(n_1(g, d_1), d_k) \right.
$$

Detecting confluence using LPPEs

$$
\begin{aligned}
X(\vec{g} : \vec{G}) = & \sum_{\vec{d_1}:\vec{D_1}} c_1 \Rightarrow \tau \cdot X(\vec{n_1}) \\
& \dots \\
& + \sum_{\vec{d_k}:\vec{D_k}} c_k \Rightarrow a_k(b_k) \sum\nolimits^{\bullet}_{\vec{e_k}:\vec{E_k}} f_k : X(\vec{n_k})
\end{aligned}
$$

To prove:

$$
c_1(g, d_1) \wedge c_k(g, d_k) \quad \rightarrow
$$

$$
\left(\begin{array}{ll} & c_k(n_1(g, d_1), d_k) \\ \wedge & c_1(n_k(g, d_k, e_k), d_1) \\ & \end{array} \right.
$$

Detecting confluence using LPPEs

$$
\begin{aligned}
X(\vec{g} : \vec{G}) = & \sum_{\vec{d_1}:\vec{D_1}} c_1 \Rightarrow \tau \cdot X(\vec{n_1}) \\
& \dots \\
& + \sum_{\vec{d_k}:\vec{D_k}} c_k \Rightarrow a_k(b_k) \sum_{\vec{e_k}:\vec{E_k}}\!\!\bullet\; f_k : X(\vec{n_k})
\end{aligned}
$$

To prove:

$$
c_1(g, d_1) \wedge c_k(g, d_k) \quad \to
$$

$$
\left(\begin{array}{ll}
 & c_k(n_1(g, d_1), d_k) \\
\wedge & c_1(n_k(g, d_k, e_k), d_1) \\
\wedge & a_k(g, d_k) = a_k(n_1(g, d_1), d_k)
\end{array} \right.
$$

Detecting confluence using LPPEs

$$
\begin{aligned}
X(\vec{g} : \vec{G}) = & \sum_{\vec{d_1}:\vec{D_1}} c_1 \Rightarrow \tau \cdot X(\vec{n_1}) \\
& \dots \\
& + \sum_{\vec{d_k}:\vec{D_k}} c_k \Rightarrow a_k(b_k) \sum\!\!\!\!\bullet_{\vec{e_k}:\vec{E_k}} f_k : X(\vec{n_k})
\end{aligned}
$$

To prove:

$$
c_1(g, d_1) \wedge c_k(g, d_k) \quad \rightarrow
$$

$$
\left(\begin{array}{ll}
 & c_k(n_1(g, d_1), d_k) \\
\wedge & c_1(n_k(g, d_k, e_k), d_1) \\
\wedge & a_k(g, d_k) = a_k(n_1(g, d_1), d_k) \\
\wedge & n_k(n_1(g, d_1), d_k, e_k) = n_1(n_k(g, d_k, e_k), d_1)
\end{array} \right)
$$

Reducing LPPEs based on confluent τ steps

After τ_c steps have been identified, two types of reductions are possible:

1. Symbolic prioritisation: change the LPPE
   - Let c be a confluent summand
   - Find a non-confluent summand n such that c is always enabled after executing n
   - Change the next state of n, basically merging n and c

As we only do this for non-confluent summands, loops are avoided.

Reducing LPPEs based on confluent τ steps

After τ_c steps have been identified, two types of reductions are possible:

1. Symbolic prioritisation: change the LPPE
   - Let *c* be a confluent summand
   - Find a non-confluent summand *n* such that *c* is always enabled after executing *n*
   - Change the next state of *n*, basically merging *n* and *c*

   As we only do this for non-confluent summands, loops are avoided.
2. On-the-fly state space generation using representatives
   - Generate the state space from the LPPE
   - For each transition that is visited, go to the representative of the target state
   - When no representative it known yet, compute it (using a variation on Tarjan's SCC algoritm)

Contents

Conclusions and Future Work

Conclusions / Results

- 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 generalised reduction techniques from LPEs to the probabilistic case; constant elimination, confluence reduction

Conclusions and Future Work

Conclusions / Results

- 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 generalised reduction techniques from LPEs to the probabilistic case; constant elimination, confluence reduction

Future work

- Finish work on confluence reduction: proofs, case study, implementation
- Develop additional reduction techniques
- Generalise proof techniques such as cones and foci to the probabilistic case

Questions

Questions?