

UNIVERSITY OF TWENTE.

Formal Methods & Tools.

Confluence Reduction for Probabilistic Systems

Mark Timmer

March 30, 2011

TACAS 2011

*Joint work with
Mariëlle Stoelinga and Jaco van de Pol*

The context – probabilistic model checking

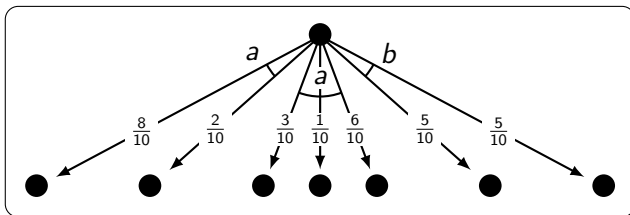
Probabilistic model checking:

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

The context – probabilistic model checking

Probabilistic model checking:

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

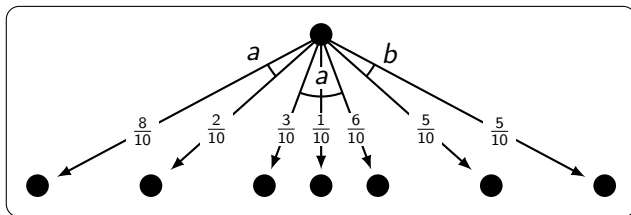


- **Non-deterministically** choose one of the three transitions
- **Probabilistically** choose the next state

The context – probabilistic model checking

Probabilistic model checking:

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

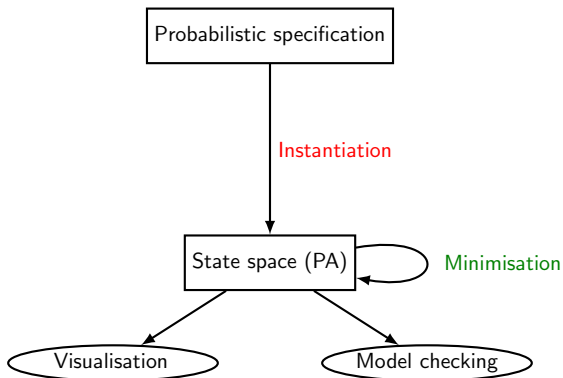


- **Non-deterministically** choose one of the three transitions
- **Probabilistically** choose the next state

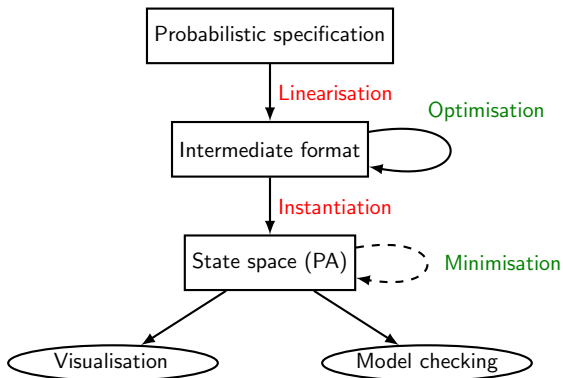
Limitations of previous approaches:

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

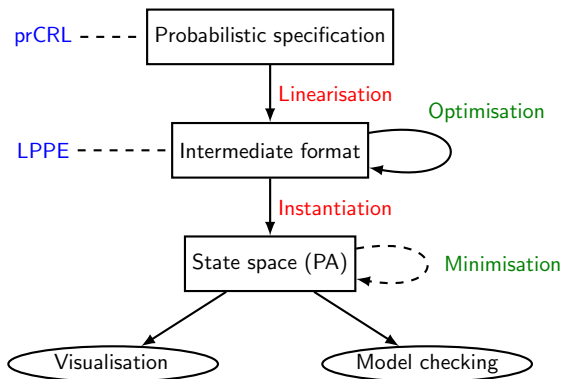
Overview of our approach



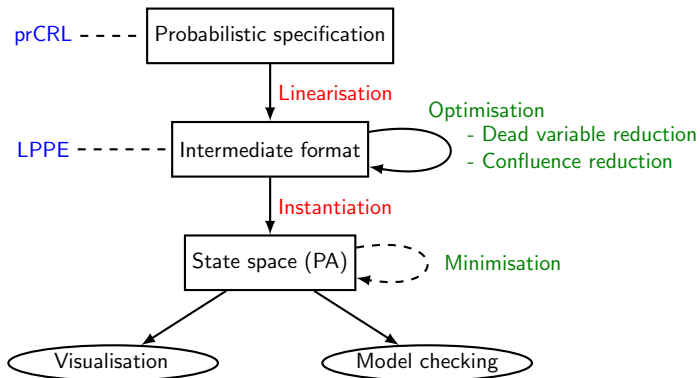
Overview of our approach



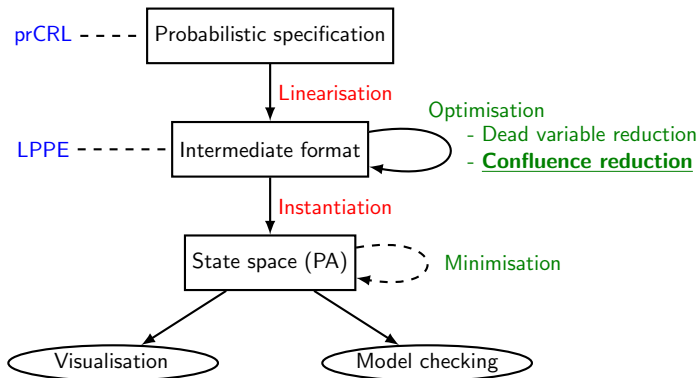
Overview of our approach



Overview of our approach



Overview of our approach



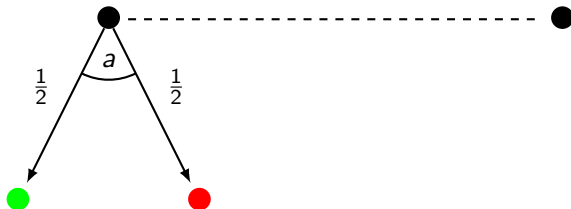
Equivalences: probabilistic bisimulation

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](#)



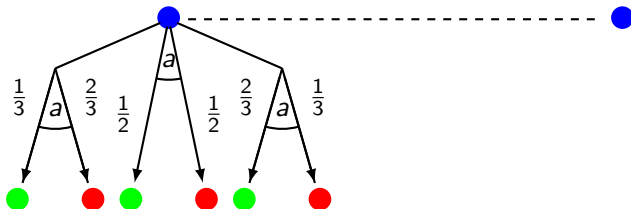
Equivalences: probabilistic bisimulation

Notions of equivalence: [strong/branching probabilistic bisimulation](#)



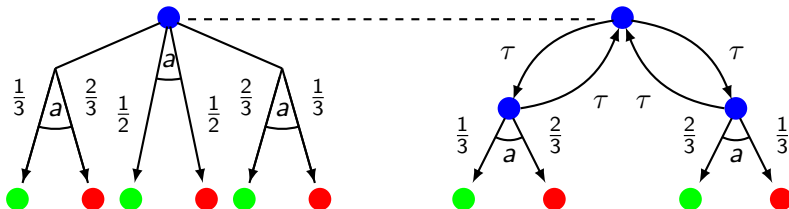
Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation



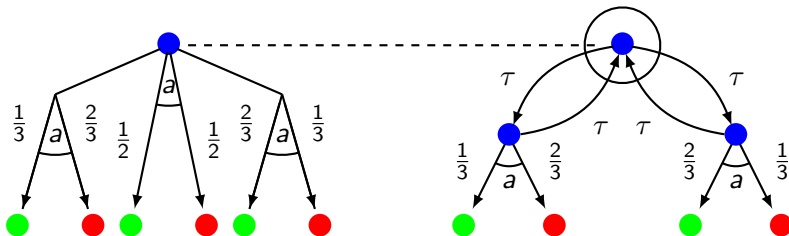
Equivalences: probabilistic bisimulation

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



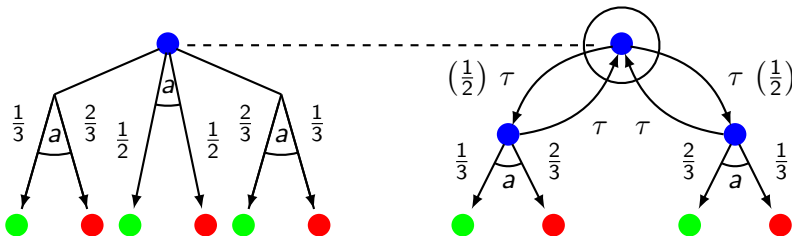
Equivalences: probabilistic bisimulation

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



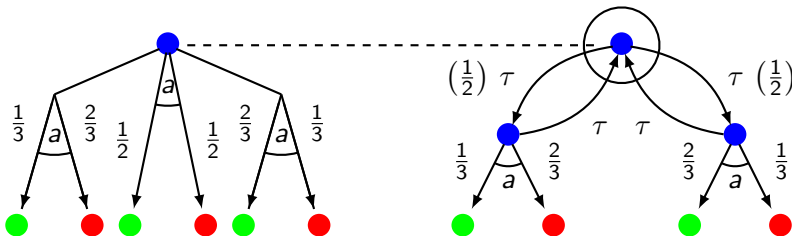
Equivalences: probabilistic bisimulation

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



Equivalences: probabilistic bisimulation

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



$$\text{Probability of green: } \frac{1}{2} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{2}$$

$$\text{Probability of red: } \frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{2}$$

Table of Contents

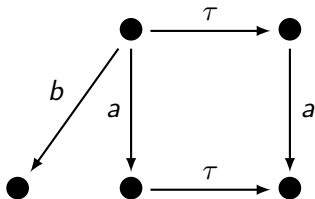
- 1 Introduction
- 2 Confluence reduction
- 3 Detecting confluence symbolically
- 4 Case study: leader election protocols
- 5 Conclusions

Table of Contents

- 1 Introduction
- 2 Confluence reduction
- 3 Detecting confluence symbolically
- 4 Case study: leader election protocols
- 5 Conclusions

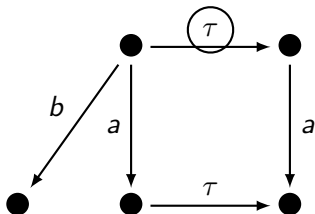
Branching bisimulation preservation by τ -steps

Unobservable τ -steps **might** disable behaviour...



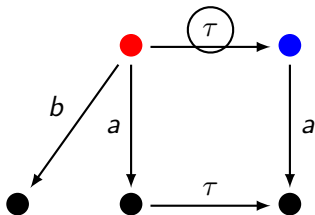
Branching bisimulation preservation by τ -steps

Unobservable τ -steps **might** disable behaviour. . .



Branching bisimulation preservation by τ -steps

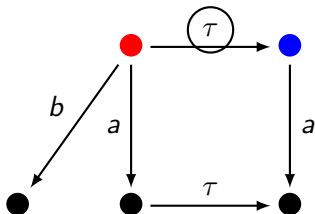
Unobservable τ -steps **might** disable behaviour...



Branching bisimulation preservation by τ -steps

Unobservable τ -steps **might** disable behaviour...

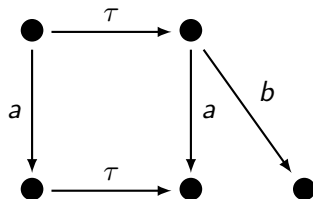
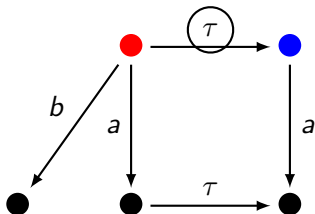
... though often, they **connect branching bisimilar states**



Branching bisimulation preservation by τ -steps

Unobservable τ -steps **might** disable behaviour...

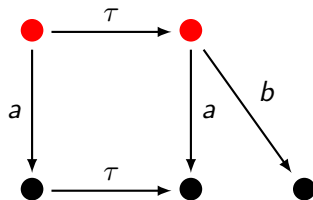
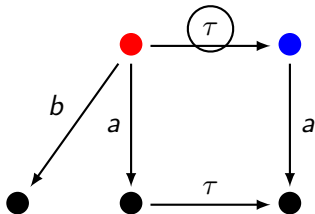
... though often, they **connect branching bisimilar states**



Branching bisimulation preservation by τ -steps

Unobservable τ -steps **might** disable behaviour...

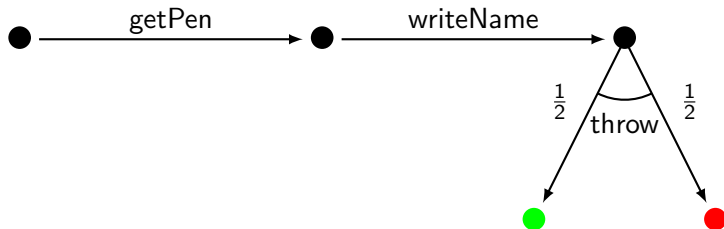
... though often, they **connect branching bisimilar states**



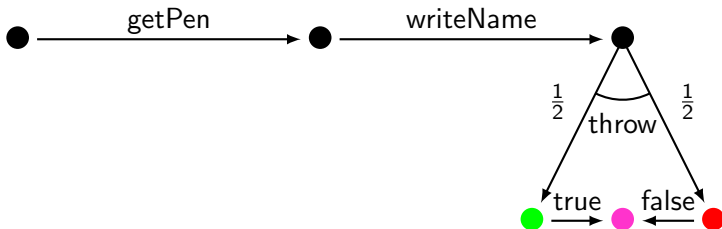
Confluence: an introductory example



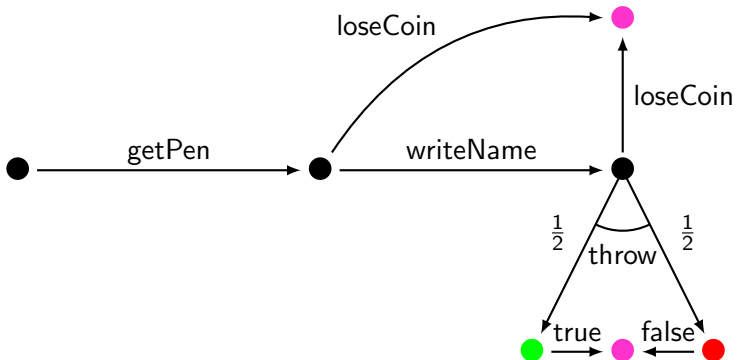
Confluence: an introductory example



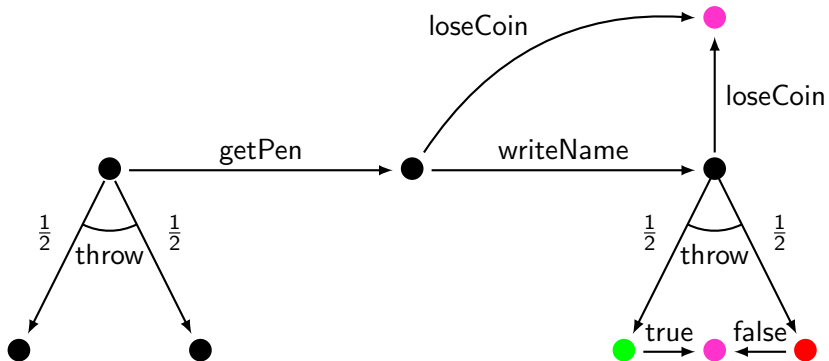
Confluence: an introductory example



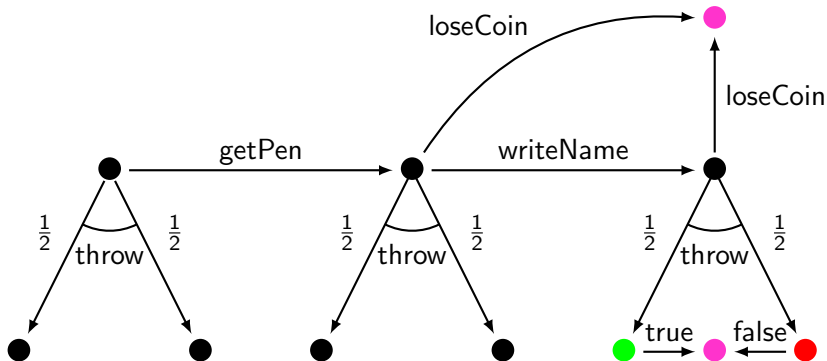
Confluence: an introductory example



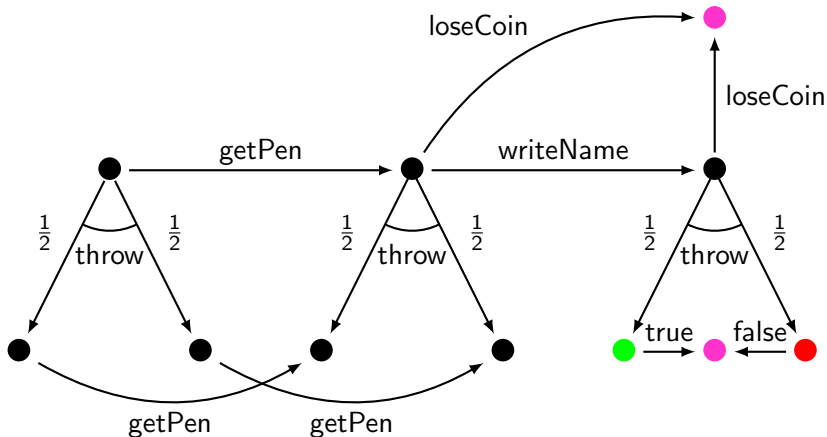
Confluence: an introductory example



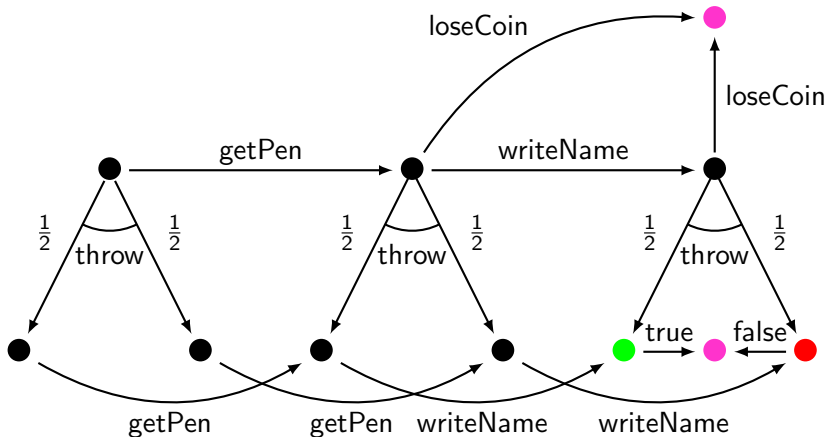
Confluence: an introductory example



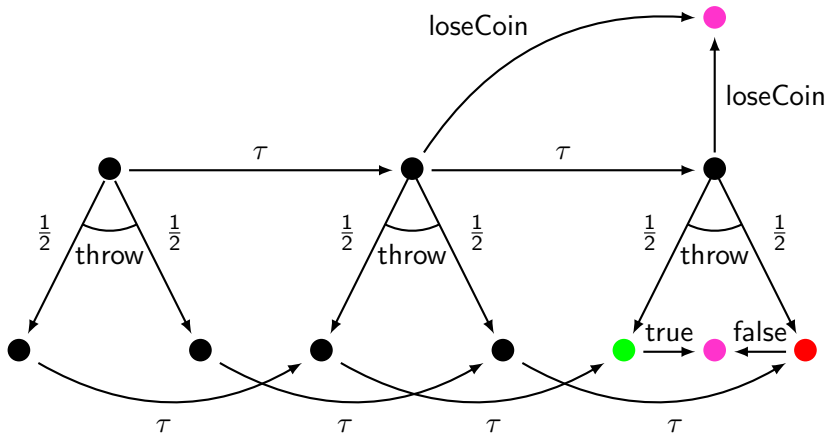
Confluence: an introductory example



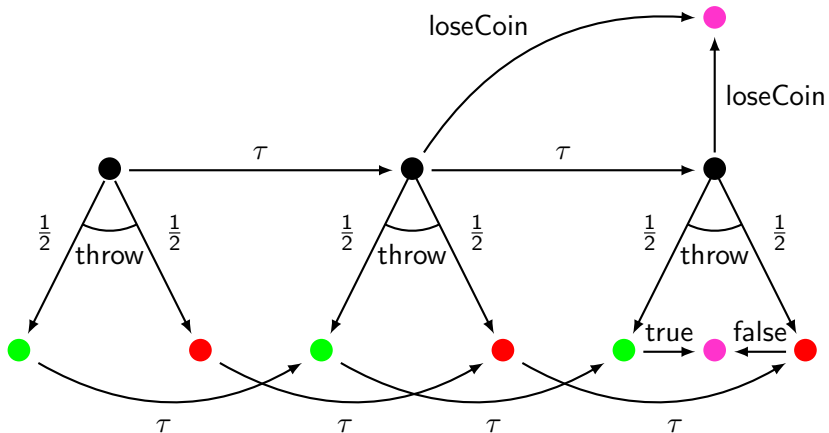
Confluence: an introductory example



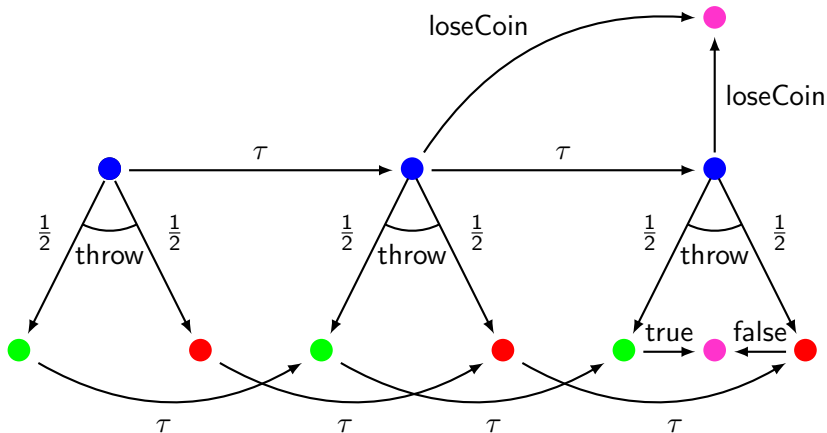
Confluence: an introductory example



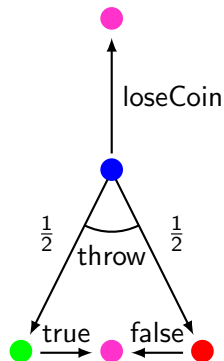
Confluence: an introductory example



Confluence: an introductory example



Confluence: an introductory example



Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- weak confluence
- confluence
- strong confluence

Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- | | | |
|---------------------|---------------|-----------------------------------|
| • weak confluence | | • weak probabilistic confluence |
| • confluence | \Rightarrow | • probabilistic confluence |
| • strong confluence | | • strong probabilistic confluence |

Confluence: non-probabilistic versus probabilistic

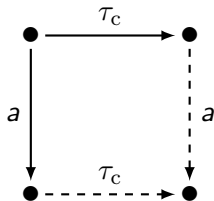
Three notions of confluence:

- | | | |
|---------------------|---------------|-----------------------------------|
| • weak confluence | | • weak probabilistic confluence |
| • confluence | \Rightarrow | • probabilistic confluence |
| • strong confluence | | • strong probabilistic confluence |

Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- | | | |
|---|---------------|---|
| <ul style="list-style-type: none"> • weak confluence • confluence • strong confluence | \Rightarrow | <ul style="list-style-type: none"> • weak probabilistic confluence • probabilistic confluence • strong probabilistic confluence |
|---|---------------|---|

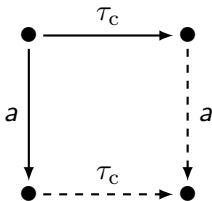


Strong confluence

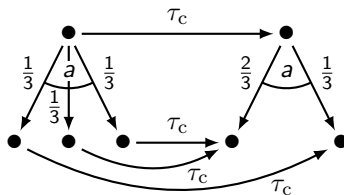
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- | | | |
|---|---|---|
| <ul style="list-style-type: none"> • weak confluence • confluence • strong confluence | ⇒ | <ul style="list-style-type: none"> • weak probabilistic confluence • probabilistic confluence • strong probabilistic confluence |
|---|---|---|



Strong confluence

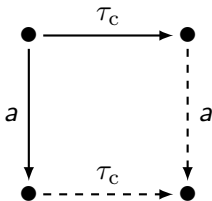


Strong probabilistic confluence

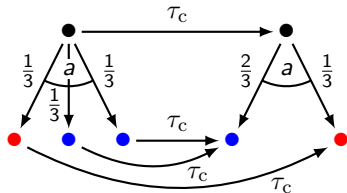
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- | | | |
|---|---------------|---|
| <ul style="list-style-type: none"> • weak confluence • confluence • strong confluence | \Rightarrow | <ul style="list-style-type: none"> • weak probabilistic confluence • probabilistic confluence • strong probabilistic confluence |
|---|---------------|---|



Strong confluence

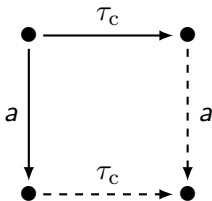


Strong probabilistic confluence

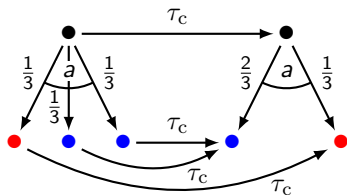
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- | | | |
|---------------------|---------------|-----------------------------------|
| • weak confluence | \Rightarrow | • weak probabilistic confluence |
| • confluence | | • probabilistic confluence |
| • strong confluence | | • strong probabilistic confluence |



Strong confluence

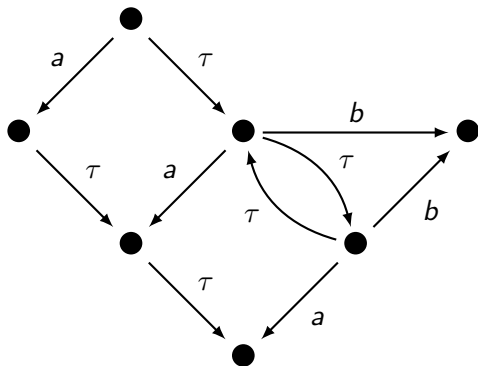


Strong probabilistic confluence

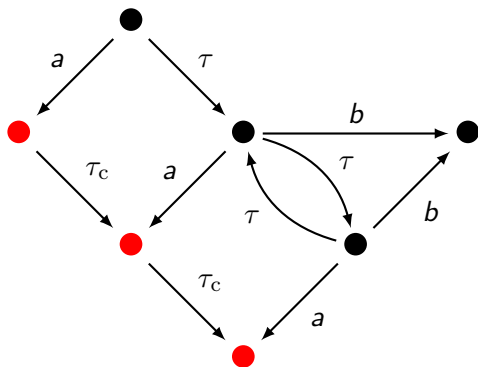
Theorem

States that are *connected by confluent τ -steps* are *branching bisimilar*.

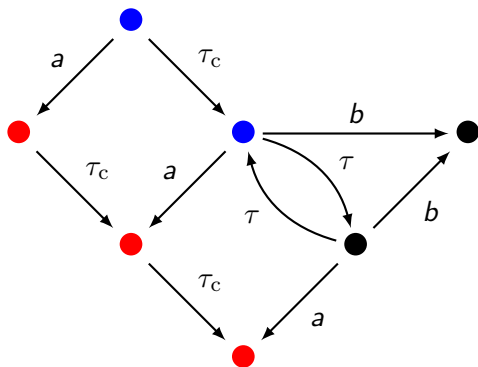
State space reduction using confluence



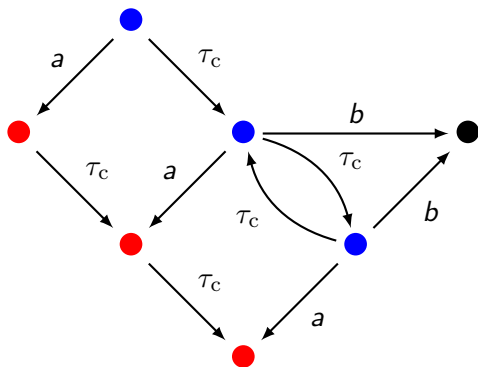
State space reduction using confluence



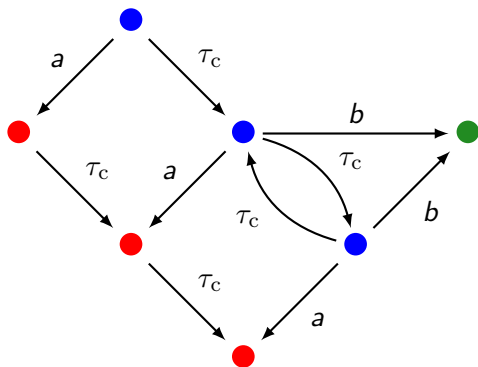
State space reduction using confluence



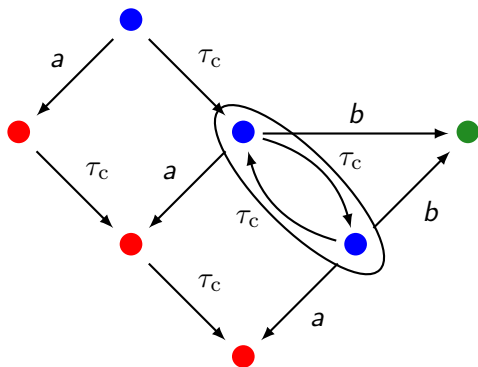
State space reduction using confluence



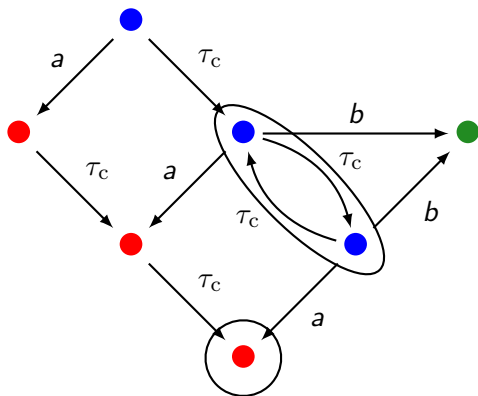
State space reduction using confluence



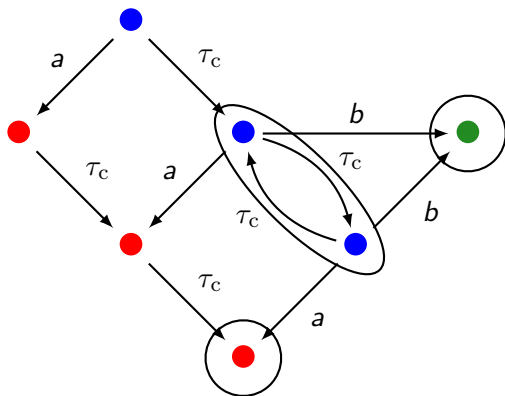
State space reduction using confluence



State space reduction using confluence



State space reduction using confluence



State space reduction using confluence

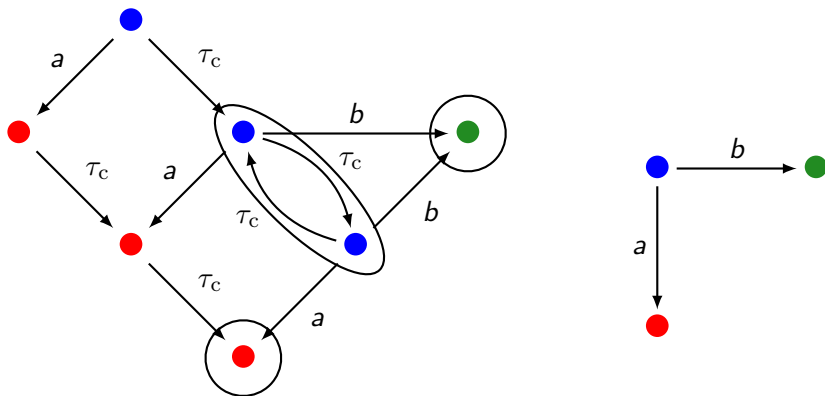


Table of Contents

- 1 Introduction
- 2 Confluence reduction
- 3 Detecting confluence symbolically**
- 4 Case study: leader election protocols
- 5 Conclusions

A process algebra with data and probability: prCRL

Specification language prCRL:

- Based on μ 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 μ 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

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

An example prCRL specification

Sending an arbitrary natural number

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

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

$+ \text{active} \quad \Rightarrow \tau \sum_{n:\mathbb{N}^{>0}} \frac{1}{2^n} : \left(\text{send}(n) \cdot X(\text{false}) \right)$

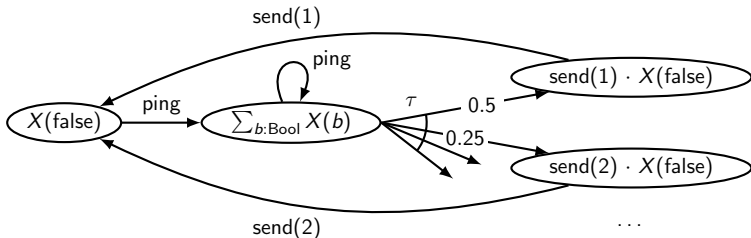
An example prCRL specification

Sending an arbitrary natural number

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

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

$+ \text{active} \quad \Rightarrow \tau \sum_{n:\mathbb{N}^{>0}} \frac{1}{2^n} : \left(\text{send}(n) \cdot X(\text{false}) \right)$



A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

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

A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

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

Advantages of using LPPEs instead of prCRL specifications:

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

A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

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

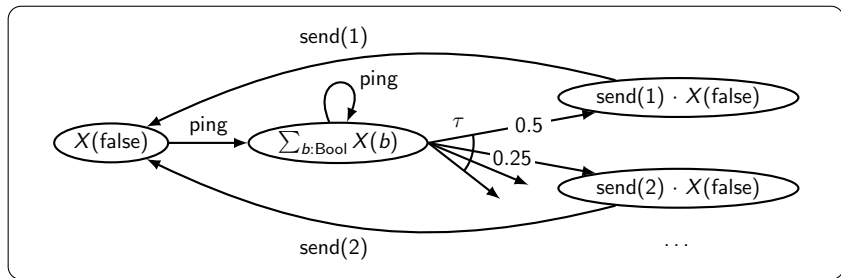
Advantages of using LPPEs instead of prCRL specifications:

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

Theorem

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

Linear Probabilistic Process Equations – an example



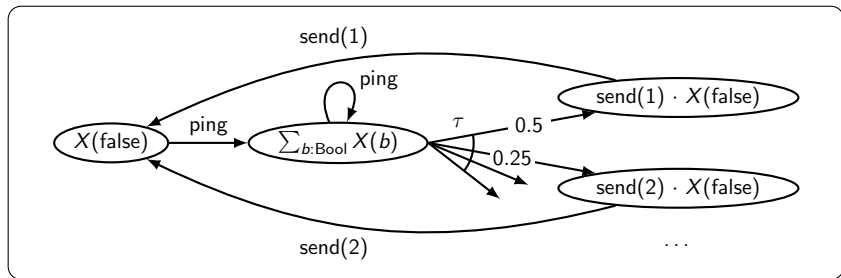
Specification in prCRL

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

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

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

Linear Probabilistic Process Equations – an example



Specification in prCRL

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

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

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

Specification in LPPE

$X(\text{pc} : \{1..3\}, n : \mathbb{N}^{\geq 0}) =$

$$+ \text{pc} = 1 \Rightarrow \text{ping} \cdot X(2, 1)$$

$$+ \text{pc} = 2 \Rightarrow \text{ping} \cdot X(2, 1)$$

$$+ \text{pc} = 2 \Rightarrow \tau \sum_{n:\mathbb{N}^{>0}} \frac{1}{2^n} : X(3, n)$$

$$+ \text{pc} = 3 \Rightarrow \text{send}(n) \cdot X(1, 1)$$

Detecting confluence symbolically on LPPEs

Symbolic detection of confluence: denote **entire summands** to be confluent (i.e., **all their concrete transitions** are confluent)

Detecting confluence symbolically on LPPEs

Symbolic detection of confluence: denote **entire summands** to be confluent (i.e., **all their concrete transitions** are confluent)

- Underapproximation of the actual confluent transitions

Detecting confluence symbolically on LPPEs

Symbolic detection of confluence: denote **entire summands** to be confluent (i.e., **all their concrete transitions** are confluent)

- Underapproximation of the actual confluent transitions

How to know whether a summand is confluent?

Detecting confluence symbolically on LPPEs

Symbolic detection of confluence: denote **entire summands** to be confluent (i.e., **all their concrete transitions** are confluent)

- Underapproximation of the actual confluent transitions

How to know whether a summand is confluent?

- Its action should be τ

Detecting confluence symbolically on LPPEs

Symbolic detection of confluence: denote **entire summands** to be confluent (i.e., **all their concrete transitions** are confluent)

- Underapproximation of the actual confluent transitions

How to know whether a summand is confluent?

- Its action should be τ
- Its next state should be chosen **nonprobabilistically**

Detecting confluence symbolically on LPPEs

Symbolic detection of confluence: denote **entire summands** to be confluent (i.e., **all their concrete transitions** are confluent)

- Underapproximation of the actual confluent transitions

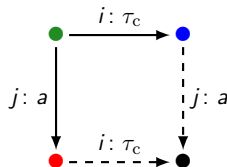
How to know whether a summand is confluent?

- Its action should be τ
- Its next state should be chosen **nonprobabilistically**
- It should **commute** with all the other summands

Symbolic detection of confluence

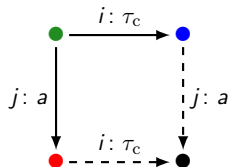
$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$

Two summands i, j commute if



Symbolic detection of confluence

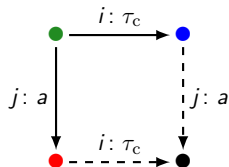
$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$



Two summands i, j **commute** if $\forall g, d_i, d_j, e_j :$

Symbolic detection of confluence

$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$

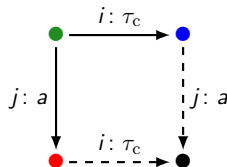


Two summands i, j **commute** if $\forall g, d_i, d_j, e_j :$

$$(c_i(g, d_i) \wedge c_j(g, d_j)) \rightarrow$$

Symbolic detection of confluence

$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$

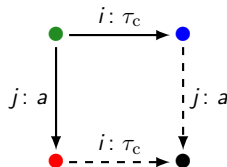


Two summands i, j **commute** if $\forall g, d_i, d_j, e_j :$

$$(c_i(g, d_i) \wedge c_j(g, d_j)) \rightarrow (i = j \wedge n_i(g, d_i) = n_i(g, d_j))$$

Symbolic detection of confluence

$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$



Two summands i, j **commute** if $\forall g, d_i, d_j, e_j :$

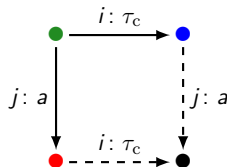
$$(c_i(g, d_i) \wedge c_j(g, d_j)) \rightarrow (i = j \wedge n_i(g, d_i) = n_i(g, d_j))$$

\vee

$$\left(\begin{array}{c} \\ \\ \\ \end{array} \right)$$

Symbolic detection of confluence

$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$



Two summands i, j **commute** if $\forall g, d_i, d_j, e_j :$

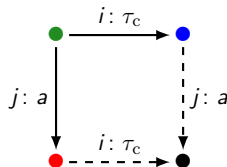
$$(c_i(g, d_i) \wedge c_j(g, d_j)) \rightarrow (i = j \wedge n_i(g, d_i) = n_i(g, d_j))$$

\vee

$$\left(c_j(n_i(g, d_i), d_j) \right)$$

Symbolic detection of confluence

$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$



Two summands i, j **commute** if $\forall g, d_i, d_j, e_j :$

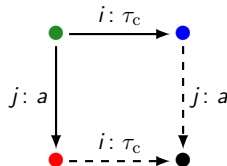
$$(c_i(g, d_i) \wedge c_j(g, d_j)) \rightarrow (i = j \wedge n_i(g, d_i) = n_i(g, d_j))$$

\vee

$$\left(c_j(n_i(g, d_i), d_j) \wedge c_i(n_j(g, d_j, e_j), d_i) \right)$$

Symbolic detection of confluence

$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$



Two summands i, j **commute** if $\forall g, d_i, d_j, e_j :$

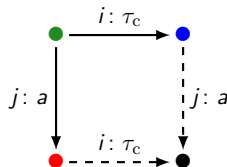
$$(c_i(g, d_i) \wedge c_j(g, d_j)) \rightarrow (i = j \wedge n_i(g, d_i) = n_i(g, d_j))$$

\vee

$$\left(\begin{array}{l} c_j(n_i(g, d_i), d_j) \wedge c_i(n_j(g, d_j, e_j), d_i) \\ \wedge \\ a_j(g, d_j) = a_j(n_i(g, d_i), d_j) \end{array} \right)$$

Symbolic detection of confluence

$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$



Two summands i, j commute if $\forall g, d_i, d_j, e_j$:

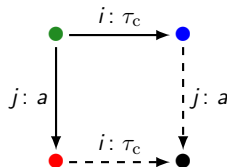
$$(c_i(g, d_i) \wedge c_j(g, d_j)) \rightarrow (i = j \wedge n_i(g, d_i) = n_i(g, d_j))$$

∨

$$\left(\begin{array}{l}
 c_j(n_i(g, d_i), d_j) \wedge c_i(n_j(g, d_j, e_j), d_i) \\
 \wedge a_j(g, d_j) = a_j(n_i(g, d_i), d_j) \\
 \wedge n_j(n_i(g, d_i), d_j, e_j) = n_i(n_j(g, d_j, e_j), d_i)
 \end{array} \right)$$

Symbolic detection of confluence

$$\begin{aligned}
 X(g : G) = & \sum_{d_i : D_i} c_i \Rightarrow \tau \cdot X(n_i) \\
 & \dots \\
 & + \sum_{d_j : D_j} c_j \Rightarrow a_j \sum_{e_j : E_j} f_j : X(n_j)
 \end{aligned}$$



Two summands i, j commute if $\forall g, d_i, d_j, e_j$:

$$(c_i(g, d_i) \wedge c_j(g, d_j)) \rightarrow (i = j \wedge n_i(g, d_i) = n_i(g, d_j))$$

∨

$$\left(\begin{array}{l}
 c_j(n_i(g, d_i), d_j) \wedge c_i(n_j(g, d_j, e_j), d_i) \\
 \wedge a_j(g, d_j) = a_j(n_i(g, d_i), d_j) \\
 \wedge n_j(n_i(g, d_i), d_j, e_j) = n_i(n_j(g, d_j, e_j), d_i) \\
 \wedge f_j(g, d_j, e_j) = f_j(n_i(g, d_i), d_j, e_j)
 \end{array} \right)$$

Heuristics for detecting confluence

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

Heuristics for detecting confluence

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

- The conditions of i and j are **disjoint**

Heuristics for detecting confluence

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

- The conditions of i and j are **disjoint**

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

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

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

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

$$i: pc1 = 2 \wedge x > 5 \wedge y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$$

$$j: pc2 = 1 \wedge y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$$

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

$$i: pc1 = 2 \wedge x > 5 \wedge y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$$

$$j: pc2 = 1 \wedge y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$$

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

$$i: pc1 = 2 \wedge x > 5 \wedge y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$$

$$j: pc2 = 1 \wedge y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$$

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

$$i: pc1 = 2 \wedge x > 5 \wedge y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$$

$$j: pc2 = 1 \wedge y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$$

- $i = j$ 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
 - $i: pc1 = 2 \wedge x > 5 \wedge y > 2 \Rightarrow \tau \cdot X(pc1 := 3, x := 0)$
 - $j: pc2 = 1 \wedge y > 2 \Rightarrow send(y) \cdot X(pc2 := 2)$
- $i = j$ and this summand only produces **one transition** per state
 - $i: pc = 1 \Rightarrow \tau \cdot X(pc := 2)$

Table of Contents

- 1 Introduction
- 2 Confluence reduction
- 3 Detecting confluence symbolically
- 4 Case study: leader election protocols**
- 5 Conclusions

Case study: leader election protocols

Basic leader election protocol

- Two processes each throw a die
- They synchronously communicate the results
- The one that throws 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 throws highest wins
- In case of a tie: start over again

More advanced leader election protocol

- **Several processes** each throw a die
- They communicate the results over **one-place buffers**
- The one that threw highest wins
- In case of a tie: continue with those processes

Applying confluence to the protocols

Specification	Original		Reduced		Runtime (sec)	
	States	Trans.	States	Trans.	Before	After
basicOriginal	3,763	6,158	631	758	0.45	0.22
basicReduced	1,693	2,438	541	638	0.22	0.13
leader-3-12	161,803	268,515	35,485	41,829	67.37	31.53
leader-3-15	311,536	515,328	68,926	80,838	145.17	65.82
leader-3-18	533,170	880,023	118,675	138,720	277.08	122.59
leader-3-21	840,799	1,385,604	187,972	219,201	817.67	211.87
leader-3-24	1,248,517	2,055,075	280,057	326,007	1069.71	333.32
leader-3-27	out of memory		398,170	462,864	–	503.85
leader-4-5	443,840	939,264	61,920	92,304	206.56	75.66
leader-4-6	894,299	1,880,800	127,579	188,044	429.87	155.96
leader-4-7	1,622,682	3,397,104	235,310	344,040	1658.38	294.09
leader-4-8	out of memory		400,125	581,468	–	653.60
leader-5-2	208,632	561,630	14,978	29,420	125.78	30.14
leader-5-3	1,390,970	3,645,135	112,559	208,170	1504.33	213.85
leader-5-4	out of memory		472,535	847,620	–	7171.73

Applying confluence to the protocols

Specification	Original		Reduced		Runtime (sec)	
	States	Trans.	States	Trans.	Before	After
basicOriginal	3,763	6,158	631	758	0.45	0.22
basicReduced	1,693	2,438	541	638	0.22	0.13
leader-3-12	161,803	268,515	35,485	41,829	67.37	31.53
leader-3-15	311,536	515,328	68,926	80,838	145.17	65.82
leader-3-18	533,170	880,023	118,675	138,720	277.08	122.59
leader-3-21	840,799	1,385,604	187,972	219,201	817.67	211.87
leader-3-24	1,248,517	2,055,075	280,057	326,007	1069.71	333.32
leader-3-27	out of memory		398,170	462,864	–	503.85
leader-4-5	443,840	939,264	61,920	92,304	206.56	75.66
leader-4-6	894,299	1,880,800	127,579	188,044	429.87	155.96
leader-4-7	1,622,682	3,397,104	235,310	344,040	1658.38	294.09
leader-4-8	out of memory		400,125	581,468	–	653.60
leader-5-2	208,632	561,630	14,978	29,420	125.78	30.14
leader-5-3	1,390,970	3,645,135	112,559	208,170	1504.33	213.85
leader-5-4	out of memory		472,535	847,620	–	7171.73

Number of **states**: –85%

Number of **transitions**: –90%

(shrinking further when parameters increase)

Table of Contents

- 1 Introduction
- 2 Confluence reduction
- 3 Detecting confluence symbolically
- 4 Case study: leader election protocols
- 5 Conclusions**

Conclusions

Conclusions

- 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 τ -loops)
- We discussed how to **detect** the strongest notion symbolically
- We illustrated the power of our methods using a **case study**

Conclusions

Conclusions

- 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 τ -loops)
- We discussed how to **detect** the strongest notion symbolically
- We illustrated the power of our methods using a **case study**

Future work

- **Comparing** confluence reduction to **partial-order reduction**
- **Symbolic** confluence reduction

Questions

Questions?