UNIVERSITY OF TWENTE.

Formal Methods & Tools.

Symbolic reductions of probabilistic models using linear process equations

Mark Timmer

January 18, 2011

FMT Lunchmeeting

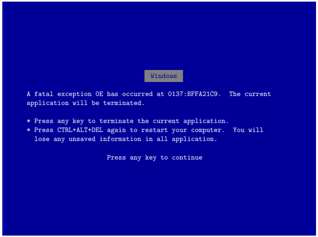
Joint work with
Mariëlle Stoelinga and Jaco van de Pol

Contents

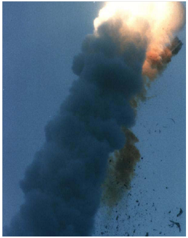
Table of Contents

Introduction – Dependability

Dependability of computer systems is becoming more and more important.



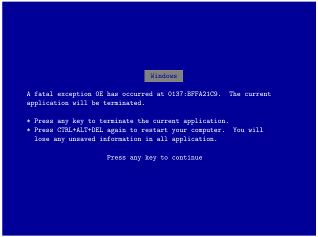
Windows blue screen



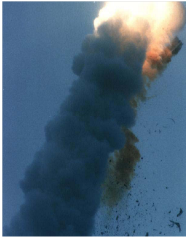
Ariane 5 crash

Introduction – Dependability

Dependability of computer systems is becoming more and more important.



Windows blue screen

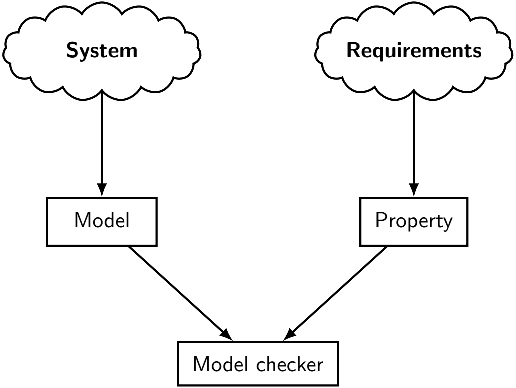


Ariane 5 crash

Our aim: use quantitative formal methods to improve system quality.

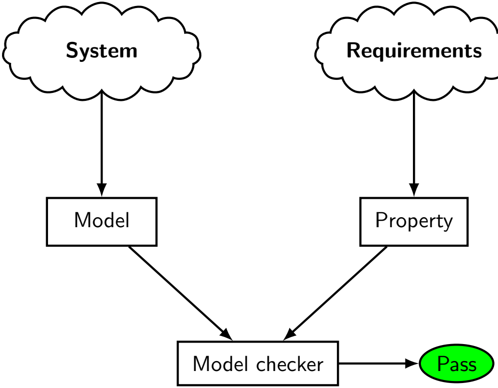
Introduction – Model Checking

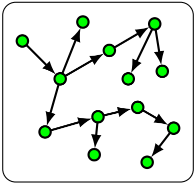
A popular solution is model checking; verifying properties of a system by constructing a model and ranging over its state space.



Introduction – Model Checking

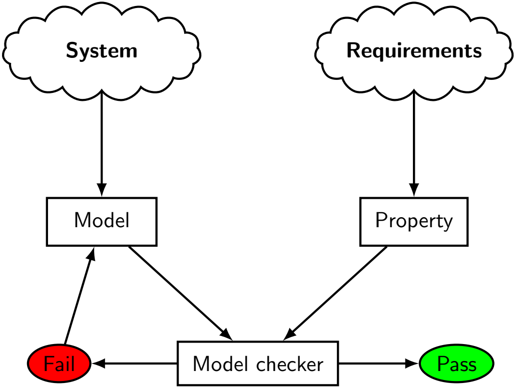
A popular solution is model checking; verifying properties of a system by constructing a model and ranging over its state space.

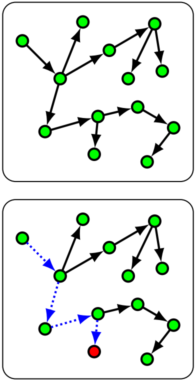




Introduction – Model Checking

A popular solution is model checking; verifying properties of a system by constructing a model and ranging over its state space.





Introduction – probabilistic model checking

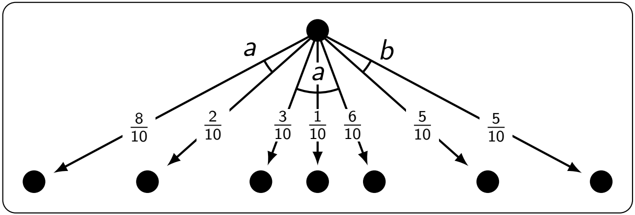
Probabilistic model checking:

- 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)

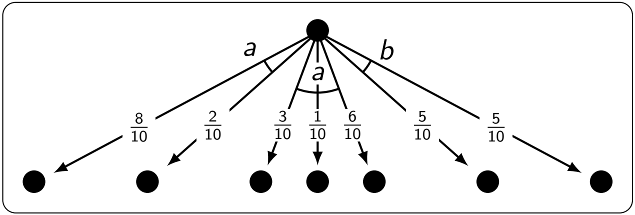


- Non-deterministically choose one of the three transitions
- Probabilistically choose the next state

Introduction – probabilistic model checking

Probabilistic model checking:

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

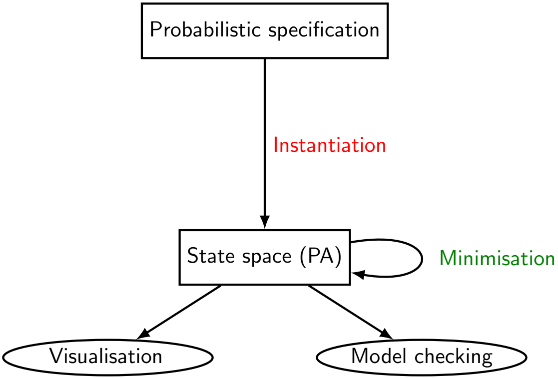


- Non-deterministically choose one of the three transitions
- Probabilistically choose the next state

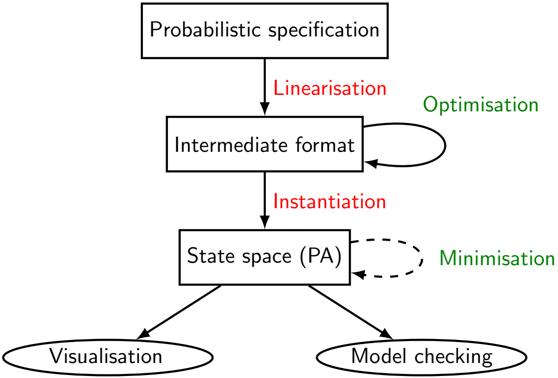
Limitations of previous approaches:

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

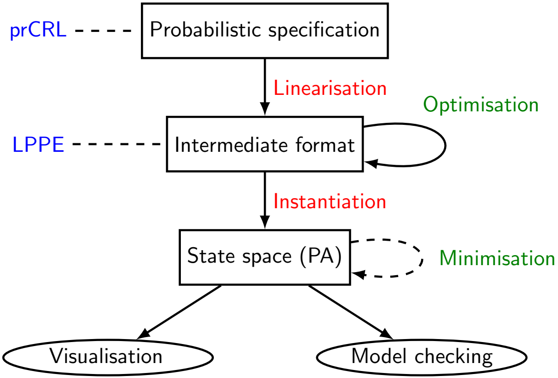
Overview of our approach



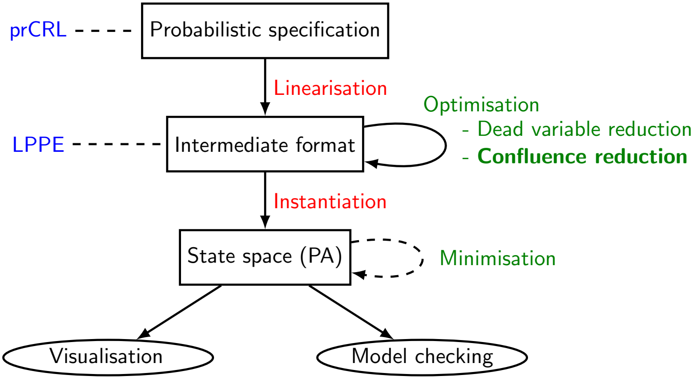
Overview of our approach



Overview of our approach

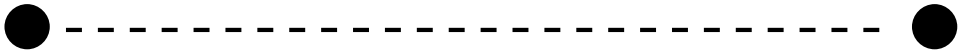


Overview of our approach



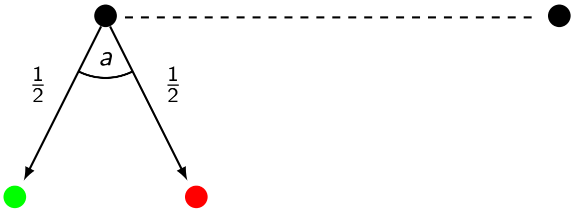
Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation



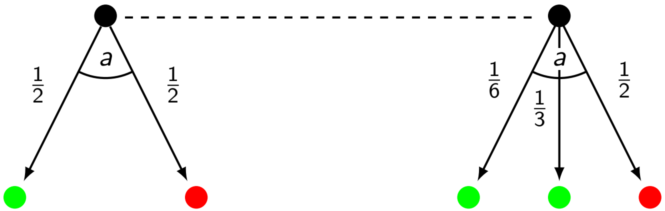
Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation



Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation

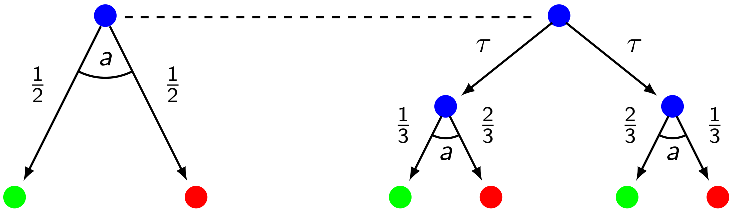


Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation

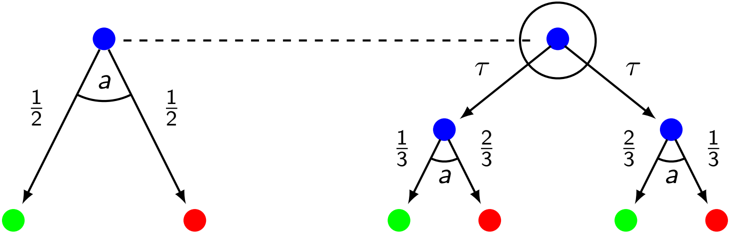
Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation



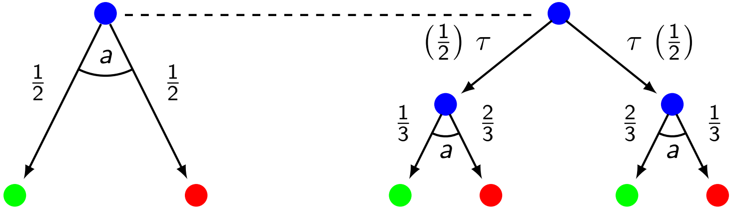
Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation



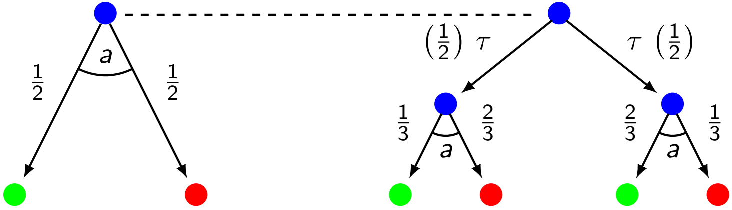
Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation



Equivalences: probabilistic bisimulation

Notions of equivalence: strong/branching probabilistic bisimulation



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

Table of 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:

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

Process equations and processes

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

An example specification

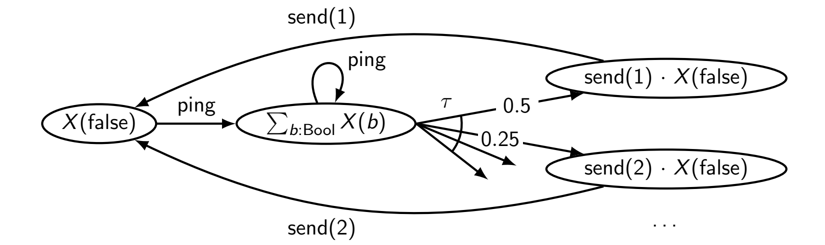
Sending an arbitrary natural number

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

An example specification

Sending an arbitrary natural number

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



Composability using extended prCRL

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

Composability using extended prCRL

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

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

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

Composability using extended prCRL

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

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

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

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

Composability using extended prCRL

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

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

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

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

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

Composability using extended prCRL

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

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

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

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

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

Composability using extended prCRL

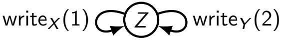
For composability we introduced extended prCRL. It extends prCRL by parallel composition, encapsulation, hiding and renaming.

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

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

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

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



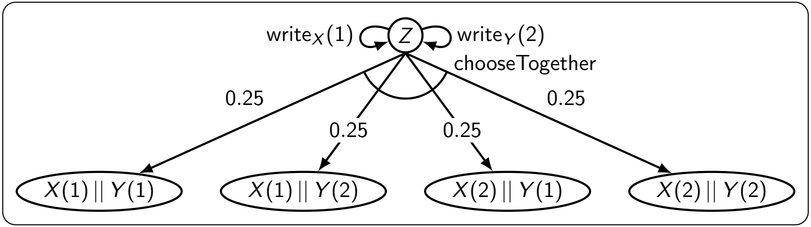
Composability using extended prCRL

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

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

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

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



A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

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

A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

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

Advantages of using LPPEs instead of prCRL specifications:

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

A linear format for prCRL: the LPPE

LPPEs are a subset of prCRL specifications:

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

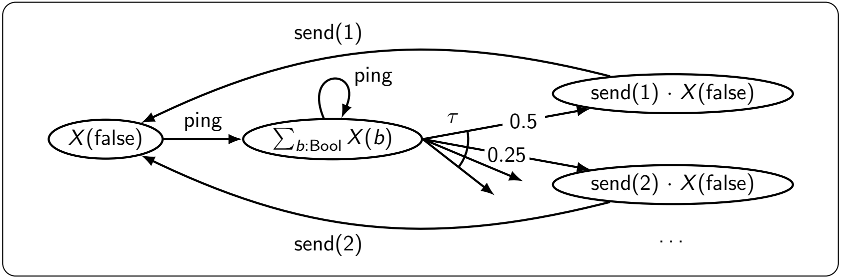
Advantages of using LPPEs instead of prCRL specifications:

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

Theorem

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

Linear Probabilistic Process Equations – an example



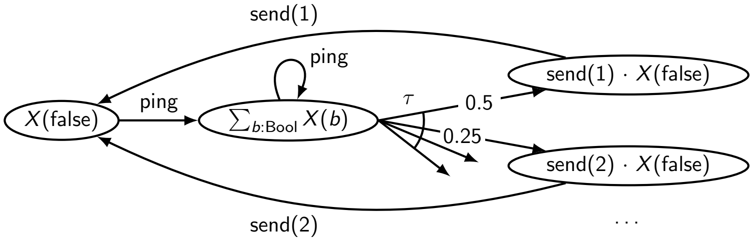
Specification in prCRL

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

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

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

Linear Probabilistic Process Equations – an example



Specification in prCRL

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

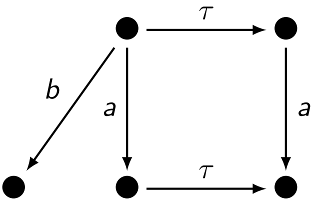
Specification in LPPE

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

Table of Contents

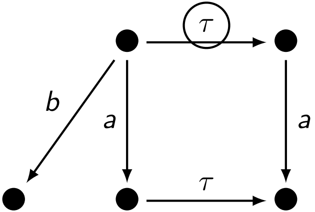
Branching bisimulation preservation by τ-steps

Unobservable τ-steps might disable behaviour...



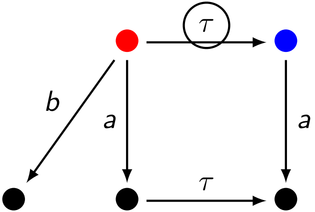
Branching bisimulation preservation by τ-steps

Unobservable τ-steps might disable behaviour. . .



Branching bisimulation preservation by τ-steps

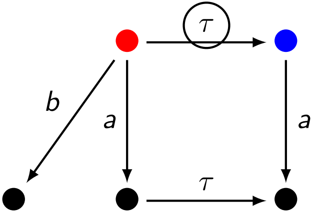
Unobservable τ-steps might disable behaviour...



Branching bisimulation preservation by τ-steps

Unobservable τ-steps might disable behaviour...

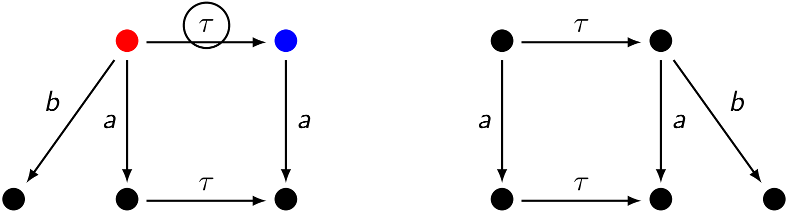
...though often, they connect branching bisimilar states



Branching bisimulation preservation by τ-steps

Unobservable τ-steps might disable behaviour...

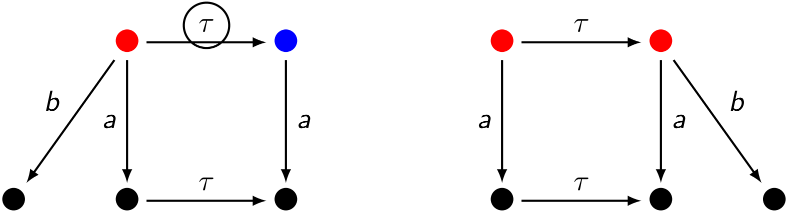
...though often, they connect branching bisimilar states



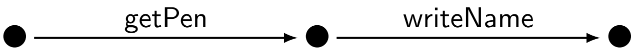
Branching bisimulation preservation by τ-steps

Unobservable τ-steps might disable behaviour...

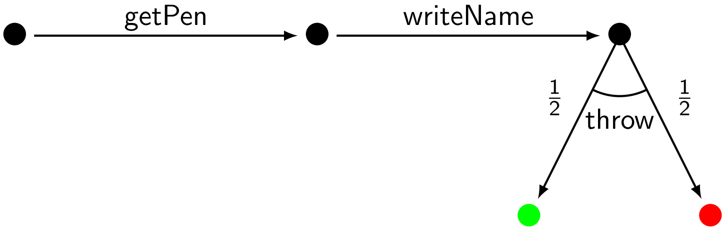
...though often, they connect branching bisimilar states



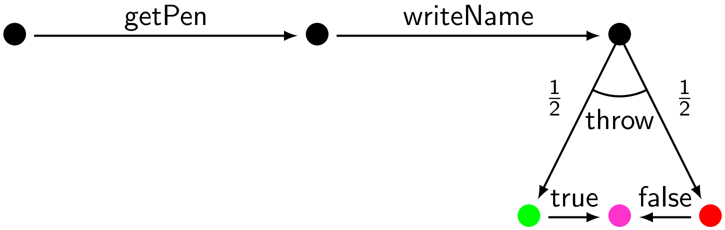
Confluence: an introductory example



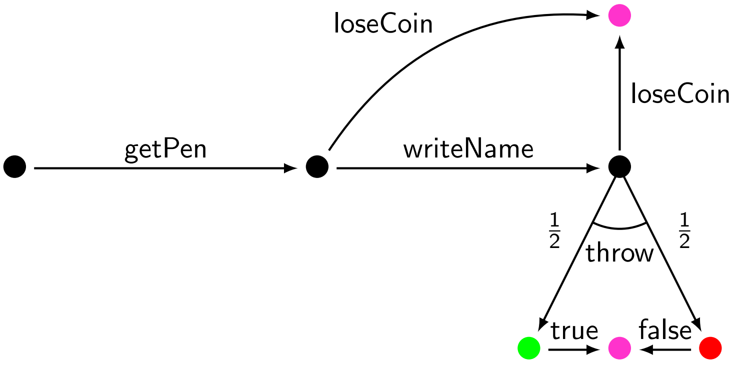
Confluence: an introductory example



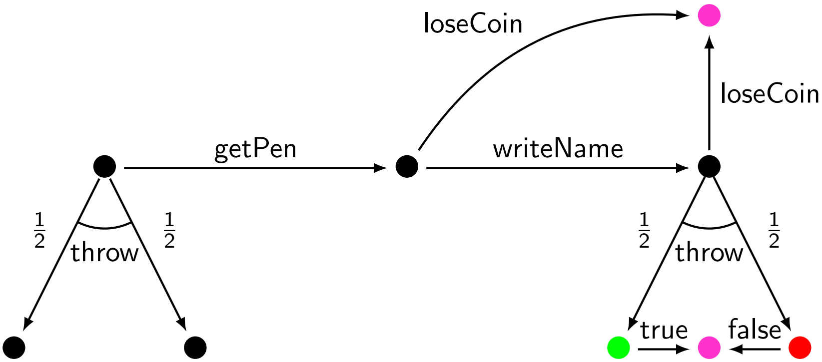
Confluence: an introductory example



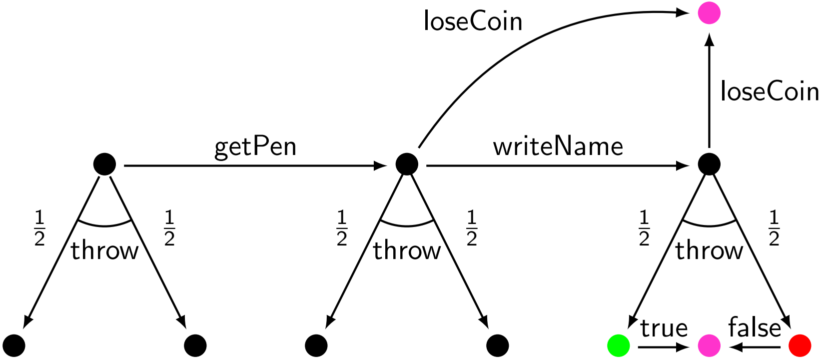
Confluence: an introductory example



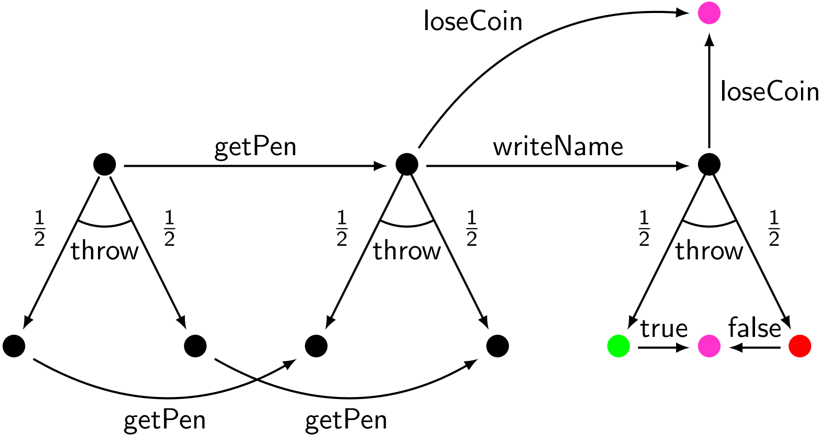
Confluence: an introductory example



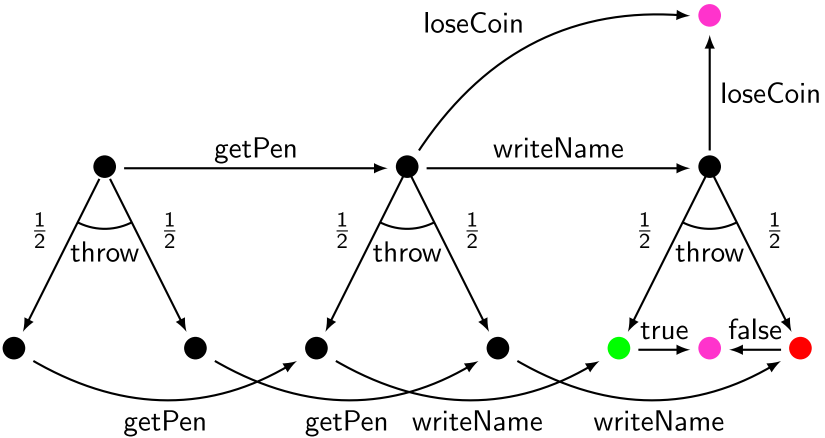
Confluence: an introductory example



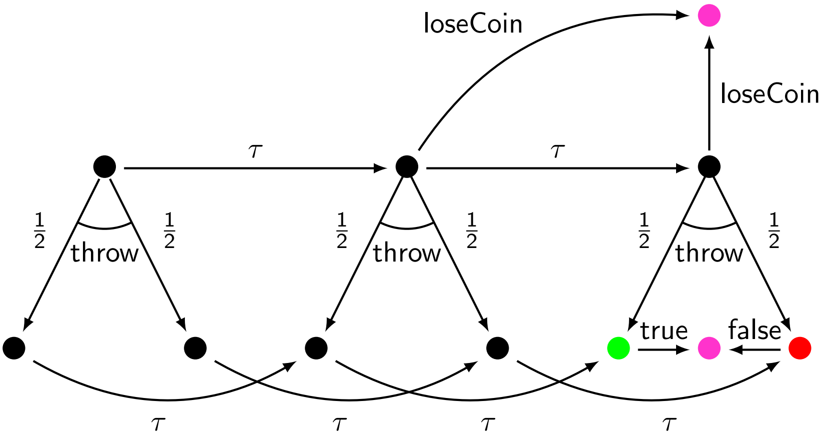
Confluence: an introductory example



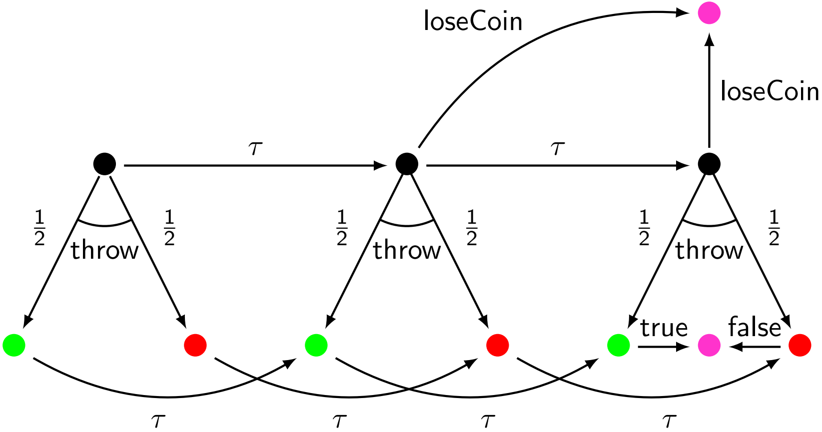
Confluence: an introductory example



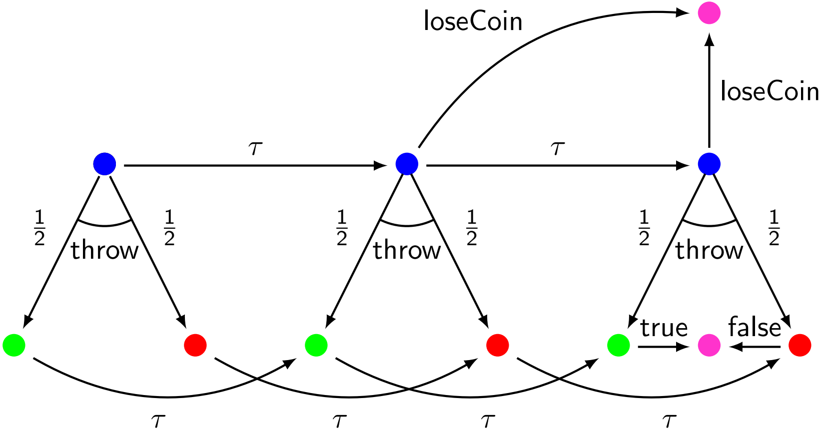
Confluence: an introductory example



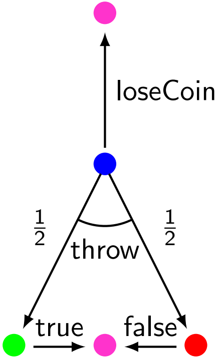
Confluence: an introductory example



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

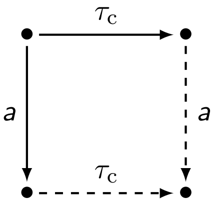
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

- weak confluence
- confluence
- strong confluence

$\Rightarrow$

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence



Strong confluence

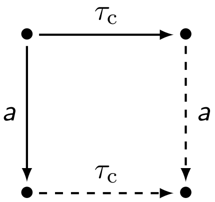
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

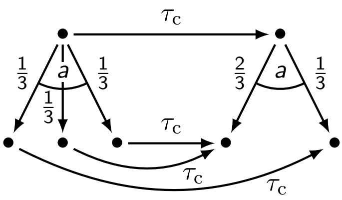
- weak confluence
- confluence
- strong confluence

$\Rightarrow$

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence



Strong confluence



Strong probabilistic confluence

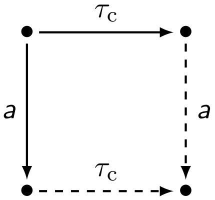
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

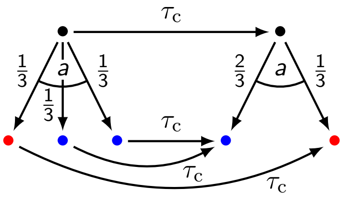
- weak confluence
- confluence
- strong confluence

$\Rightarrow$

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence



Strong confluence



Strong probabilistic confluence

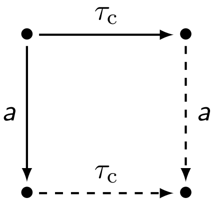
Confluence: non-probabilistic versus probabilistic

Three notions of confluence:

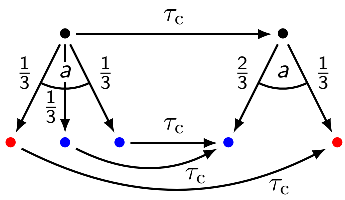
- weak confluence
- confluence
- strong confluence

$\Rightarrow$

- weak probabilistic confluence
- probabilistic confluence
- strong probabilistic confluence



Strong confluence

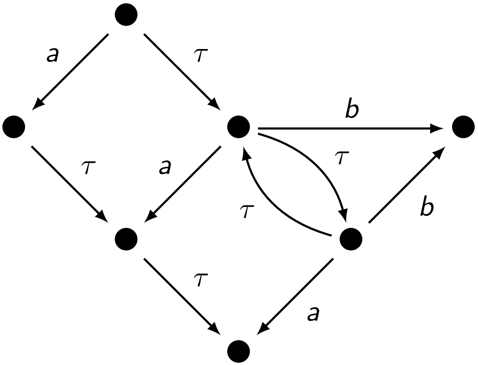


Strong probabilistic confluence

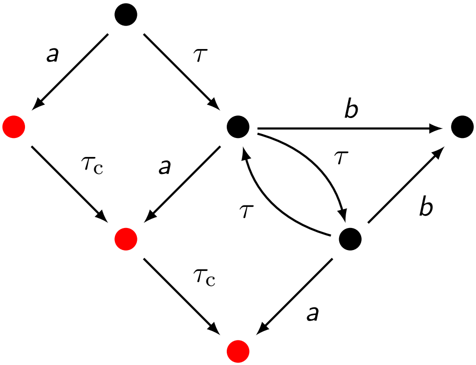
Theorem

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

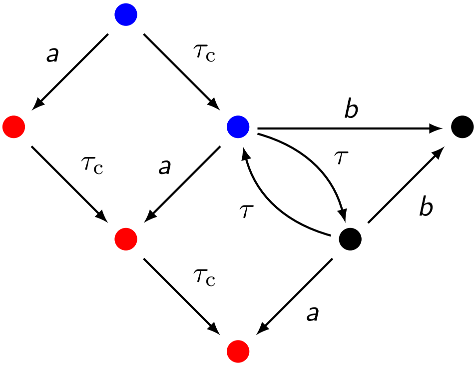
State space reduction using confluence



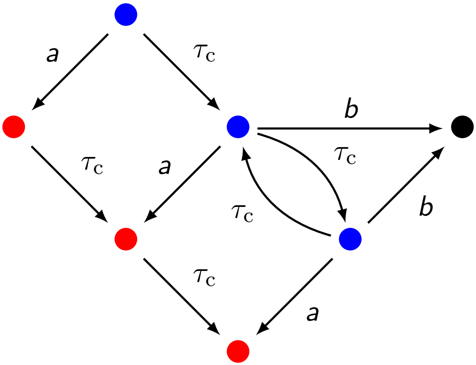
State space reduction using confluence



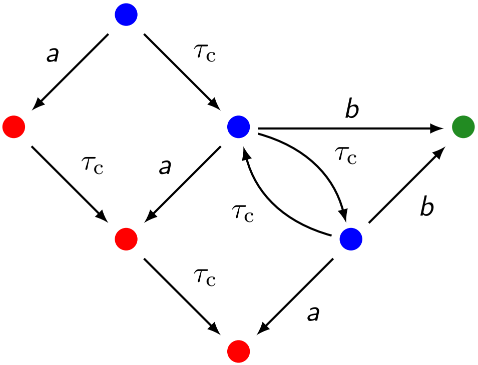
State space reduction using confluence



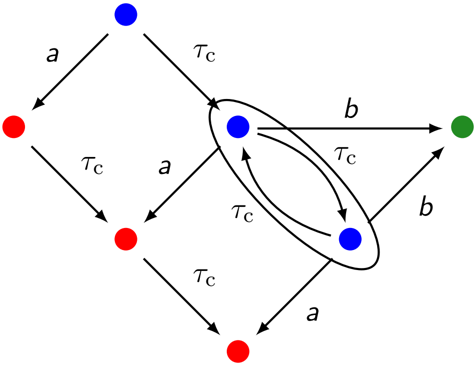
State space reduction using confluence



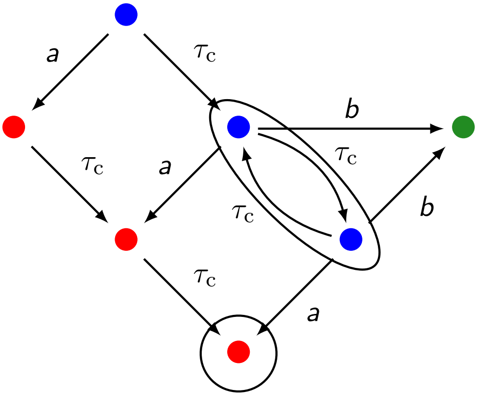
State space reduction using confluence



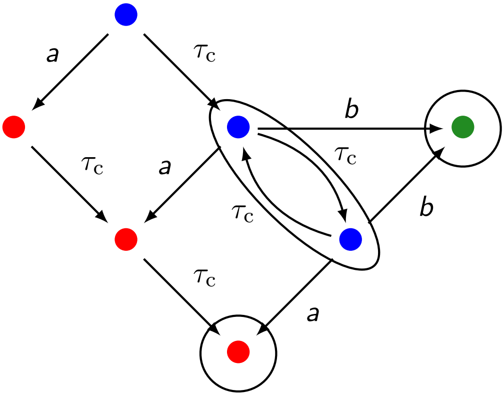
State space reduction using confluence



State space reduction using confluence



State space reduction using confluence



State space reduction using confluence

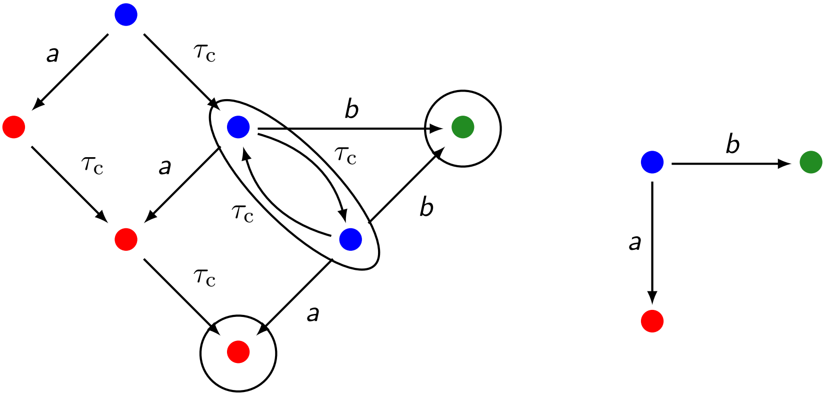


Table of Contents

Detecting confluence symbolically: LPPEs

Example specification

$$
\begin{aligned}
X(\mathrm{pc} : \{1..2\}, \mathrm{active} : \mathrm{Bool}) = \\
\sum_{n:\{1,2,3\}} \quad \mathrm{pc} = 1 \quad & \Rightarrow \mathrm{output}(n) \sum\!\!\!\bullet_{\mathrm{b:Bool}} \tfrac{1}{2} : X(2, \mathrm{b}) \\
+ \qquad \mathrm{pc} = 2 \wedge \mathrm{active} & \Rightarrow \mathrm{beep} \cdot X(1, \mathrm{active})
\end{aligned}
$$

Detecting confluence symbolically: LPPEs

Example specification

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

$$\sum_{n:\{1,2,3\}} \text{pc} = 1 \Rightarrow \text{output}(n) \sum_{\text{b:Bool}} \bullet \tfrac{1}{2} : X(2, \text{b})$$

$$+ \quad \text{pc} = 2 \land \text{active} \Rightarrow \tau \cdot X(1, \text{active})$$

Detecting confluence symbolically: LPPEs

Example specification

$$X(\mathrm{pc} : \{1..2\}, \mathrm{active} : \mathrm{Bool}) =$$
$$\sum_{n:\{1,2,3\}} \mathrm{pc} = 1 \Rightarrow \mathrm{output}(n) \sum_{\mathrm{b:Bool}}\bullet \tfrac{1}{2} : X(2, \mathrm{b})$$
$$+ \quad \mathrm{pc} = 2 \land \mathrm{active} \Rightarrow \tau \cdot X(1, \mathrm{active})$$

How to know whether a summand is confluent?

Detecting confluence symbolically: LPPEs

Example specification

$$X(\text{pc} : \{1..2\}, \text{active} : \text{Bool}) =$$
$$\sum_{n:\{1,2,3\}} \text{pc} = 1 \Rightarrow \text{output}(n) \sum_{\text{b:Bool}} \tfrac{1}{2} : X(2, \text{b})$$
$$+ \quad \text{pc} = 2 \wedge \text{active} \Rightarrow \tau \cdot X(1, \text{active})$$

How to know whether a summand is confluent?

- Its action should be τ

Detecting confluence symbolically: LPPEs

Example specification

$$
\begin{aligned}
&X(\text{pc} : \{1..2\}, \text{active} : \text{Bool}) = \\
&\quad \sum_{n:\{1,2,3\}} \text{pc} = 1 \Rightarrow \text{output}(n) \sum_{\text{b:Bool}} \tfrac{1}{2} : X(2, \text{b}) \\
&+ \quad \text{pc} = 2 \land \text{active} \Rightarrow \tau \cdot X(1, \text{active})
\end{aligned}
$$

How to know whether a summand is confluent?

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

Detecting confluence symbolically: LPPEs

Example specification

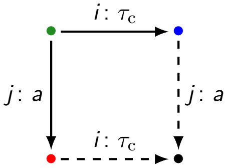
$$X(\mathrm{pc} : \{1..2\}, \mathrm{active} : \mathrm{Bool}) =$$
$$\sum_{n:\{1,2,3\}} \mathrm{pc} = 1 \Rightarrow \mathrm{output}(n) \sum_{\mathrm{b:Bool}} \tfrac{1}{2} : X(2, \mathrm{b})$$
$$+ \quad \mathrm{pc} = 2 \wedge \mathrm{active} \Rightarrow \tau \cdot X(1, \mathrm{active})$$

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

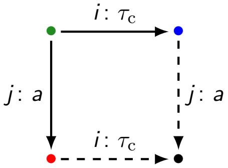
$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i \sum_{\boldsymbol{e_i}:\boldsymbol{E_i}}\!\!\!\bullet\; f_i : X(\boldsymbol{n_i})$$
$$\dots$$
$$+ \sum_{\boldsymbol{d_j}:\boldsymbol{D_j}} c_j \Rightarrow a_j \sum_{\boldsymbol{e_j}:\boldsymbol{E_j}}\!\!\!\bullet\; f_j : X(\boldsymbol{n_j})$$



Two summands i, j commute if

Symbolic detection of confluence

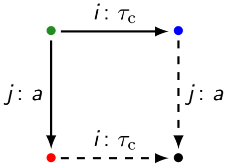
$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i \sum_{\boldsymbol{e_i}:\boldsymbol{E_i}}\!\!\bullet\; f_i : X(\boldsymbol{n_i})$$
$$\dots$$
$$+ \sum_{\boldsymbol{d_j}:\boldsymbol{D_j}} c_j \Rightarrow a_j \sum_{\boldsymbol{e_j}:\boldsymbol{E_j}}\!\!\bullet\; f_j : X(\boldsymbol{n_j})$$



Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

Symbolic detection of confluence

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}}\!\!\bullet\; f_i : X(\boldsymbol{n_i})$$
$$\dots$$
$$+ \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow a_j \sum_{\boldsymbol{e_j} : \boldsymbol{E_j}}\!\!\bullet\; f_j : X(\boldsymbol{n_j})$$

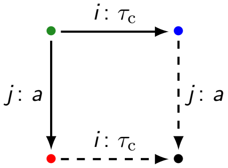


Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

$$\left(c_i(\boldsymbol{g}, \boldsymbol{d_i}) \wedge c_j(\boldsymbol{g}, \boldsymbol{d_j})\right) \rightarrow$$

Symbolic detection of confluence

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i \sum\!\!\!\!\bullet_{\boldsymbol{e_i}:\boldsymbol{E_i}} f_i : X(\boldsymbol{n_i})$$
$$\cdots$$
$$+ \sum_{\boldsymbol{d_j}:\boldsymbol{D_j}} c_j \Rightarrow a_j \sum\!\!\!\!\bullet_{\boldsymbol{e_j}:\boldsymbol{E_j}} f_j : X(\boldsymbol{n_j})$$

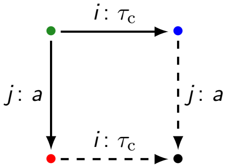


Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

$$\big(c_i(\boldsymbol{g}, \boldsymbol{d_i}) \wedge c_j(\boldsymbol{g}, \boldsymbol{d_j})\big) \rightarrow \big(i = j \wedge \boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}) = \boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j})\big)$$

Symbolic detection of confluence

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



Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

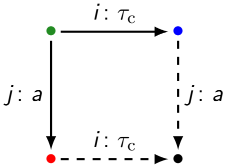
$$\left(c_i(\boldsymbol{g}, \boldsymbol{d_i}) \wedge c_j(\boldsymbol{g}, \boldsymbol{d_j})\right) \rightarrow \left(i = j \wedge \boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}) = \boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j})\right)$$

$$\vee$$

$$\left(\qquad \right)$$

Symbolic detection of confluence

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i \sum_{\boldsymbol{e_i}:\boldsymbol{E_i}}\!\!\bullet\; f_i : X(\boldsymbol{n_i}) \quad \cdots \quad + \sum_{\boldsymbol{d_j}:\boldsymbol{D_j}} c_j \Rightarrow a_j \sum_{\boldsymbol{e_j}:\boldsymbol{E_j}}\!\!\bullet\; f_j : X(\boldsymbol{n_j})$$



Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

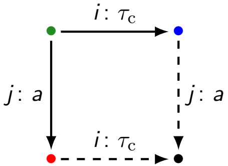
$$\left(c_i(\boldsymbol{g}, \boldsymbol{d_i}) \wedge c_j(\boldsymbol{g}, \boldsymbol{d_j})\right) \rightarrow \left(i = j \wedge \boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}) = \boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j})\right)$$

$$\vee$$

$$\left(c_j(\boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}), \boldsymbol{d_j}) \right)$$

Symbolic detection of confluence

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i \sum_{\boldsymbol{e_i}:\boldsymbol{E_i}}\!\!\!\bullet\; f_i : X(\boldsymbol{n_i})$$
$$\dots$$
$$+ \sum_{\boldsymbol{d_j}:\boldsymbol{D_j}} c_j \Rightarrow a_j \sum_{\boldsymbol{e_j}:\boldsymbol{E_j}}\!\!\!\bullet\; f_j : X(\boldsymbol{n_j})$$



Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

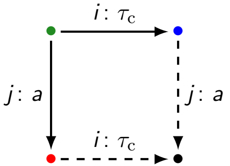
$$\left(c_i(\boldsymbol{g}, \boldsymbol{d_i}) \wedge c_j(\boldsymbol{g}, \boldsymbol{d_j})\right) \rightarrow \left(i = j \wedge \boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}) = \boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j})\right)$$

$$\vee$$

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

Symbolic detection of confluence

$$
\begin{aligned}
X(\boldsymbol{g} : \boldsymbol{G}) = & \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i \sum_{\bullet\, \boldsymbol{e_i} : \boldsymbol{E_i}} f_i : X(\boldsymbol{n_i}) \\
& \cdots \\
& + \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow a_j \sum_{\bullet\, \boldsymbol{e_j} : \boldsymbol{E_j}} f_j : X(\boldsymbol{n_j})
\end{aligned}
$$



Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

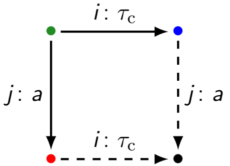
$$
(c_i(\boldsymbol{g}, \boldsymbol{d_i}) \wedge c_j(\boldsymbol{g}, \boldsymbol{d_j})) \rightarrow (i = j \wedge \boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}) = \boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j}))
$$

$$
\vee
$$

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

Symbolic detection of confluence

$$
\begin{aligned}
X(\boldsymbol{g} : \boldsymbol{G}) = & \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i \sum_{\bullet\, \boldsymbol{e_i} : \boldsymbol{E_i}} f_i : X(\boldsymbol{n_i}) \\
& \dots \\
& + \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow a_j \sum_{\bullet\, \boldsymbol{e_j} : \boldsymbol{E_j}} f_j : X(\boldsymbol{n_j})
\end{aligned}
$$



Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

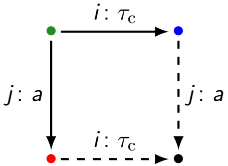
$$
(c_i(\boldsymbol{g}, \boldsymbol{d_i}) \wedge c_j(\boldsymbol{g}, \boldsymbol{d_j})) \rightarrow (i = j \wedge \boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}) = \boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j}))
$$

$$
\vee
$$

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

Symbolic detection of confluence

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}}\!\!\bullet\; f_i : X(\boldsymbol{n_i}) \quad \dots \quad + \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow a_j \sum_{\boldsymbol{e_j} : \boldsymbol{E_j}}\!\!\bullet\; f_j : X(\boldsymbol{n_j})$$



Two summands i, j commute if $\forall \boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}, \boldsymbol{e_i}, \boldsymbol{e_j}$:

$$\left(c_i(\boldsymbol{g}, \boldsymbol{d_i}) \wedge c_j(\boldsymbol{g}, \boldsymbol{d_j})\right) \rightarrow \left(i = j \wedge \boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}) = \boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j})\right)$$

$$\vee$$

$$\begin{pmatrix} & c_j(\boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}), \boldsymbol{d_j}) \wedge c_i(\boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j}), \boldsymbol{d_i}) \\ \wedge & a_j(\boldsymbol{g}, \boldsymbol{d_j}) = a_j(\boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}), \boldsymbol{d_j}) \\ \wedge & \boldsymbol{n_j}(\boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}), \boldsymbol{d_j}, \boldsymbol{e_j}) = \boldsymbol{n_i}(\boldsymbol{n_j}(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j}), \boldsymbol{d_i}, \boldsymbol{e_i}) \\ \wedge & f_j(\boldsymbol{g}, \boldsymbol{d_j}, \boldsymbol{e_j}) = f_j(\boldsymbol{n_i}(\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{e_i}), \boldsymbol{d_j}, \boldsymbol{e_j}) \end{pmatrix}$$

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

  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: $pc = 3 \Rightarrow \quad \tau \quad \cdot X(pc := 4)$
  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: $pc = 3 \Rightarrow \quad \tau \quad \cdot X(pc := 4)$
  j: $pc = 5 \Rightarrow send(y) \cdot X(pc := 1)$

- Neither summand uses variables that are changed by the other

  i: $pc1 = 2 \wedge x > 5 \wedge y > 2 \Rightarrow \quad \tau \quad \cdot X(pc1 := 3, x := 0)$
  j: $pc2 = 1 \wedge y > 2 \quad \Rightarrow send(y) \cdot X(pc2 := 2)$

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

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

  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

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

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 States	Original Trans.	Reduced States	Reduced Trans.	Runtime (sec) Before	Runtime (sec) 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%

Table of Contents

Conclusions

Conclusions

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

M. Timmer, M.I.A. Stoelinga, and J.C. van de Pol.
Confluence reduction for probabilistic systems.
In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2011)*.

Questions

Questions?