Confluence Reduction for Probabilistic Systems

Mark Timmer, Mariëlle Stoelinga, and Jaco van de Pol*

Formal Methods and Tools, Faculty of EEMCS
University of Twente, The Netherlands
{timmer, marielle, vdpol}@cs.utwente.nl

Abstract. This paper presents a novel technique for state space reduction of probabilistic specifications, based on a newly developed notion of confluence for probabilistic automata. We prove that this reduction preserves branching probabilistic bisimulation and can be applied on-the-fly. To support the technique, we introduce a method for detecting confluent transitions in the context of a probabilistic process algebra with data, facilitated by an earlier defined linear format. A case study demonstrates that significant reductions can be obtained.

1 Introduction

Model checking of probabilistic systems is getting more and more attention, but there still is a large gap between the number of techniques supporting traditional model checking and those supporting probabilistic model checking. Especially methods aimed at reducing state spaces are greatly needed to battle the omnipresent state space explosion.

In this paper, we generalise the notion of confluence [8] from labelled transition systems (LTSs) to probabilistic automata (PAs) [14]. Basically, we define under which conditions unobservable transitions (often called τ-transitions) do not influence a PA's behaviour (i.e., they commute with all other transitions). Using this new notion of probabilistic confluence, we introduce a symbolic technique that reduces PAs while preserving branching probabilistic bisimulation.

The non-probabilistic case. Our methodology follows the approach for LTSs from [4]. It consists of the following steps: (i) a system is specified as the parallel composition of several processes with data; (ii) the specification is linearised to a canonical form that facilitates symbolic manipulations; (iii) first-order logic formulas are generated to check symbolically which τ-transitions are confluent; (iv) an LTS is generated in such a way that confluent τ-transitions are given priority, leading to an on-the-fly (potentially exponential) state space reduction. Refinements by [12] make it even possible to perform confluence detection on-the-fly by means of boolean equation systems.

The probabilistic case. After recalling some basic concepts from probability theory and probabilistic automata, we introduce three novel notions of probabilistic

* This research has been partially funded by NWO under grant 612.063.817 (SYRUP) and grant Dn 63-257 (ROCKS), and by the European Union under FP7-ICT-2007-1 grant 214755 (QUASIMODO).

confluence. Inspired by [3], these are *weak probabilistic confluence*, *probabilistic confluence* and *strong probabilistic confluence* (in decreasing order of reduction power, but in increasing order of detection efficiency).

We prove that the stronger notions imply the weaker ones, and that τ-transitions that are confluent according to any of these notions always connect branching probabilistically bisimilar states. Basically, this means that they can be given priority without losing any behaviour. Based on this idea, we propose a reduction technique that can be applied using the two stronger notions of confluence. For each set of states that can reach each other by traversing only confluent transitions, it chooses a representative state that has all relevant behaviour. We prove that this reduction technique yields a branching probabilistically bisimilar PA. Therefore, it preserves virtually all interesting temporal properties.

As we want to analyse systems that would normally be too large, we need to detect confluence symbolically and use it to reduce on-the-fly during state space generation. That way, the unreduced PA never needs to be generated. Since it is not clear how not to detect (weak) probabilistic confluence efficiently, we only provide a detection method for strong probabilistic confluence. Here, we exploit a previously defined probabilistic process-algebraic linear format, which is capable of modelling any system consisting of parallel components with data [10]. In this paper, we show how symbolic τ-transitions can be proven confluent by solving formulas in first-order logic over this format. As a result, confluence can be detected symbolically, and the reduced PA can be generated on-the-fly. We present a case study of leader election protocols, showing significant reductions.

Proofs for all our propositions and theorems can be found in an extended version of this paper [17].

Related work. As mentioned before, we basically generalise the techniques presented in [4] to PAs.

In the probabilistic setting, several reduction techniques similar to ours exist. Most of these are generalisations of the well-known concept of partial-order reduction (POR) [13]. In [2] and [5], the concept of POR was lifted to Markov decision processes, providing reductions that preserve quantitative LTL$\setminus$X. This was refined in [1] to probabilistic CTL, a branching logic. Recently, a revision of POR for distributed schedulers was introduced and implemented in PRISM [7].

Our confluence reduction differs from these techniques on several accounts. First, POR is applicable on state-based systems, whereas our confluence reduction is the first technique that can be used for action-based systems. As the transformation between action- and state-based blows up the state space [11], having confluence reduction really provides new possibilities. Second, the definition of confluence is quite elegant, and (strong) confluence seems to be of a more local nature (which makes the correctness proofs easier). Third, the detection of POR requires language-specific heuristics, whereas confluence reduction acts at a more semantic level and can be implemented by a generic theorem prover. (Alternatively, decision procedures for a fixed set of data types could be devised.)

Our case study shows that the reductions obtained using probabilistic confluence exceed the reductions obtained by probabilistic POR [9].

2 Preliminaries

Given a set S, an element $s \in S$ and an equivalence relation $R \subseteq S \times S$, we write $[s]_R$ for the *equivalence class* of s under R, i.e., $[s]_R = \{s' \in S \mid (s, s') \in R\}$. We write $S/R = \{[s]_R \mid s \in S\}$ for the set of all equivalence classes in S.

2.1 Probability theory and probabilistic automata

Definition 1 (Probability distributions). *A* probability distribution *over a countable set S is a function $\mu\colon S \to [0,1]$ such that $\sum_{s\in S} \mu(s) = 1$. Given $S' \subseteq S$, we write $\mu(S')$ to denote $\sum_{s'\in S'} \mu(s')$. We use* $\mathrm{Distr}(S)$ *to denote the set of all probability distributions over S, and* $\mathrm{Distr}^*(S)$ *for the set of all sub-stochastic probability distributions over S, i.e., where $0 \leq \sum_{s\in S} \mu(s) \leq 1$.*

Given a probability distribution μ with $\mu(s_1) = p_1$, $\mu(s_2) = p_2, \ldots$ ($p_i \neq 0$), we write $\mu = \{s_1 \mapsto p_1, s_2 \mapsto p_2, \ldots\}$ and let $\mathrm{spt}(\mu) = \{s_1, s_2, \ldots\}$ denote its support. For the deterministic distribution *μ determined by $\mu(t) = 1$ we write $\mathbb{1}_t$.*

Given an equivalence relation R over S and two probability distributions μ, μ' over S, we say that $\mu \equiv_R \mu'$ if and only if $\mu(C) = \mu'(C)$ for all $C \in S/R$.

Probabilistic automata (PAs) are similar to labelled transition systems, except that transitions do not have a fixed successor state anymore. Instead, the state reached after taking a certain transition is determined by a probability distribution [14]. The transitions themselves can be chosen nondeterministically.

Definition 2 (Probabilistic automata). *A* probabilistic automaton (PA) *is a tuple $\mathcal{A} = \langle S, s^0, L, \Delta\rangle$, where S is a countable set of states of which $s^0 \in S$ is initial, L is a countable set of actions, and $\Delta \subseteq S \times L \times \mathrm{Distr}(S)$ is a countable transition relation. We assume that every PA contains an unobservable action $\tau \in L$. If $(s, a, \mu) \in \Delta$, we write $s \xrightarrow{a} \mu$, meaning that state s enables action a, after which the probability to go to $s' \in S$ is $\mu(s')$. If $\mu = \mathbb{1}_t$, we write $s \xrightarrow{a} t$.*

Definition 3 (Paths and traces). *Given a PA $\mathcal{A} = \langle S, s^0, L, \Delta\rangle$, we define a* path *of $\mathcal{A}$ to be either a finite sequence $\pi = s_0 \overset{a_1,\mu_1}{\rightsquigarrow} s_1 \overset{a_2,\mu_2}{\rightsquigarrow} s_2 \overset{a_3,\mu_3}{\rightsquigarrow} \cdots \overset{a_n,\mu_n}{\rightsquigarrow} s_n$, or an infinite sequence $\pi' = s_0 \overset{a_1,\mu_1}{\rightsquigarrow} s_1 \overset{a_2,\mu_2}{\rightsquigarrow} s_2 \overset{a_3,\mu_3}{\rightsquigarrow} \ldots$, where for finite paths we require $s_i \in S$ for all $0 \leq i \leq n$, and $s_i \xrightarrow{a_{i+1}} \mu_{i+1}$ as well as $\mu_{i+1}(s_{i+1}) > 0$ for all $0 \leq i < n$. For infinite paths these properties should hold for all $i \geq 0$. A fragment $s \overset{a,\mu}{\rightsquigarrow} s'$ denotes that the transition $s \xrightarrow{a} \mu$ was chosen from state s, after which the successor s' was selected by chance (so $\mu(s') > 0$).*

- *If $\pi = s_0 \overset{a,\mathbb{1}_{s_1}}{\rightsquigarrow} s_1 \overset{a,\mathbb{1}_{s_2}}{\rightsquigarrow} \ldots \overset{a,\mathbb{1}_{s_n}}{\rightsquigarrow} s_n$ is a path of $\mathcal{A}$ ($n \geq 0$), we write $s_0 \xrightarrow{a}\!\!\!\!\rightarrow s_n$. In case we also allow steps of the form $s_i \overset{a,\mathbb{1}_{s_{i+1}}}{\leftsquigarrow} s_{i+1}$, we write $s_0 \xleftarrow{a}\!\!\!\!\rightarrow s_n$. If there exists a state t such that $s \xrightarrow{a}\!\!\!\!\rightarrow t$ and $s' \xrightarrow{a}\!\!\!\!\rightarrow t$, we write $s \xrightarrow{a}\!\!\!\!\rightarrow\!\!\!\!\leftarrow\!\!\!\!\xleftarrow{a} s'$.*
- *We use prefix(π, i) to denote $s_0 \overset{a_1,\mu_1}{\rightsquigarrow} \ldots \overset{a_i,\mu_i}{\rightsquigarrow} s_i$, and step$(\pi, i)$ to denote the transition (s_{i-1}, a_i, μ_i). When π is finite we define $|\pi| = n$ and last$(\pi) = s_n$.*
- *We use finpaths$_\mathcal{A}$ to denote the set of all finite paths of $\mathcal{A}$, and finpaths$_\mathcal{A}(s)$ for all finite paths where $s_0 = s$.*
- *A path's* trace *is the sequence of actions obtained by omitting all its states, distributions and τ-steps; given $\pi = s_0 \overset{a_1,\mu_1}{\rightsquigarrow} s_1 \overset{\tau,\mu_2}{\rightsquigarrow} s_2 \overset{a_3,\mu_3}{\rightsquigarrow} \ldots \overset{a_n,\mu_n}{\rightsquigarrow} s_n$, we denote the sequence $a_1 a_3 \ldots a_n$ by trace(π).*

2.2 Schedulers

To resolve the nondeterminism in PAs, schedulers are used [16]. Basically, a scheduler is a function defining for each finite path which transition to take next. The decisions of schedulers are allowed to be *randomised*, i.e., instead of choosing a single transition a scheduler might resolve a nondeterministic choice by a probabilistic choice. Schedulers can be *partial*, i.e., they might assign some probability to the decision of not choosing any next transition.

Definition 4 (Schedulers). *A* scheduler *for a PA $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ is a function*

$$\mathcal{S} \colon \mathit{finpaths}_{\mathcal{A}} \to \mathrm{Distr}(\{\bot\} \cup \Delta),$$

such that for every $\pi \in \mathit{finpaths}_{\mathcal{A}}$ the transitions (s, a, μ) that are scheduled by $\mathcal{S}$ after π are indeed possible after π, i.e., $\mathcal{S}(\pi)(s, a, \mu) > 0$ implies $s = \mathit{last}(\pi)$. The decision of not choosing any transition is represented by $\bot$.

We now define the notions of finite and maximal paths of a PA given a scheduler.

Definition 5 (Finite and maximal paths). *Let $\mathcal{A}$ be a PA and $\mathcal{S}$ a scheduler for $\mathcal{A}$. Then, the set of* finite paths of $\mathcal{A}$ under $\mathcal{S}$ *is given by*

$$\mathit{finpaths}^{\mathcal{S}}_{\mathcal{A}} = \{\pi \in \mathit{finpaths}_{\mathcal{A}} \mid \forall 0 \leq i < |\pi| \,.\, \mathcal{S}(\mathit{prefix}(\pi, i))(\mathit{step}(\pi, i+1)) > 0\}.$$

We define $\mathit{finpaths}^{\mathcal{S}}_{\mathcal{A}}(s) \subseteq \mathit{finpaths}^{\mathcal{S}}_{\mathcal{A}}$ as the set of all such paths starting in s. The set of maximal paths of $\mathcal{A}$ under $\mathcal{S}$ *is given by*

$$\mathit{maxpaths}^{\mathcal{S}}_{\mathcal{A}} = \{\pi \in \mathit{finpaths}^{\mathcal{S}}_{\mathcal{A}} \mid \mathcal{S}(\pi)(\bot) > 0\}.$$

Similarly, $\mathit{maxpaths}^{\mathcal{S}}_{\mathcal{A}}(s)$ is the set of maximal paths of $\mathcal{A}$ under $\mathcal{S}$ starting in s.

We now define the behaviour of a PA $\mathcal{A}$ under a scheduler $\mathcal{S}$. As schedulers resolve all nondeterministic choices, this behaviour is fully probabilistic. We can therefore compute the probability that, starting from a given state s, the path generated by $\mathcal{S}$ has some finite prefix π. This probability is denoted by $P^{\mathcal{S}}_{\mathcal{A},s}(\pi)$.

Definition 6 (Path probabilities). *Let $\mathcal{A}$ be a PA, $\mathcal{S}$ a scheduler for $\mathcal{A}$, and s a state of $\mathcal{A}$. Then, we define the function $P^{\mathcal{S}}_{\mathcal{A},s} \colon \mathit{finpaths}_{\mathcal{A}}(s) \to [0,1]$ by*

$$P^{\mathcal{S}}_{\mathcal{A},s}(s) = 1; \qquad P^{\mathcal{S}}_{\mathcal{A},s}(\pi \overset{a,\mu}{\rightsquigarrow} t) = P^{\mathcal{S}}_{\mathcal{A},s}(\pi) \cdot \mathcal{S}(\pi)(\mathit{last}(\pi), a, \mu) \cdot \mu(t).$$

Based on these probabilities we can compute the probability distribution $F^{\mathcal{S}}_{\mathcal{A}}(s)$ over the states where a PA $\mathcal{A}$ under a scheduler $\mathcal{S}$ terminates, when starting in state s. Note that $F^{\mathcal{S}}_{\mathcal{A}}(s)$ is potentially substochastic (i.e., the probabilities do not add up to 1) if $\mathcal{S}$ allows infinite behaviour.

Definition 7 (Final state probabilities). *Let $\mathcal{A}$ be a PA and $\mathcal{S}$ a scheduler for $\mathcal{A}$. Then, we define the function $F^{\mathcal{S}}_{\mathcal{A}} \colon S \to \mathrm{Distr}^*(S)$ by*

$$F^{\mathcal{S}}_{\mathcal{A}}(s) = \Big\{ s' \mapsto \sum_{\substack{\pi \in \mathit{maxpaths}^{\mathcal{S}}_{\mathcal{A}}(s) \\ \mathit{last}(\pi) = s'}} P^{\mathcal{S}}_{\mathcal{A},s}(\pi) \cdot \mathcal{S}(\pi)(\bot) \mid s' \in S \Big\} \qquad \forall s \in S.$$

3 Branching probabilistic bisimulation

The notion of branching bisimulation for non-probabilistic systems was first introduced in [19]. Basically, it relates states that have an identical branching structure in the presence of τ-actions. Segala defined a generalisation of branching bisimulation for PAs [15], which we present here using the simplified definitions of [16]. First, we intuitively explain weak steps for PAs. Based on these ideas, we then formally introduce branching probabilistic bisimulation.

3.1 Weak steps for probabilistic automata

As τ-steps cannot be observed, we want to abstract from them. Non-probabilistically, this is done via the weak step. A state s can do a weak step to s' under an action a, denoted by $s \stackrel{a}{\Longrightarrow} s'$, if there exists a path $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} s_n \xrightarrow{a} s'$ with $n \geq 0$ (often, also τ-steps after the a-action are allowed, but this will not concern us). Traditionally, $s \stackrel{a}{\Longrightarrow} s'$ is thus satisfied by an *appropriate path*.

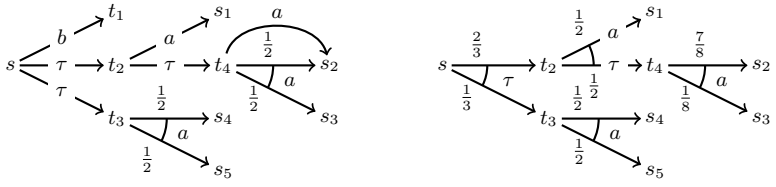
In the probabilistic setting, $s \stackrel{a}{\Longrightarrow} \mu$ is satisfied by an *appropriate scheduler*. A scheduler $\mathcal{S}$ is appropriate if for every maximal path π that is scheduled from s with non-zero probability, $\mathit{trace}(\pi) = a$ and the a-transition is the last transition of the path. Also, the final state distribution $F^{\mathcal{S}}_{\mathcal{A}}(s)$ must be equal to μ.

Example 8. Consider the PA shown in Figure 1(a). We demonstrate that $s \stackrel{a}{\Longrightarrow} \mu$, with $\mu = \{s_1 \mapsto \frac{8}{24}, s_2 \mapsto \frac{7}{24}, s_3 \mapsto \frac{1}{24}, s_4 \mapsto \frac{4}{24}, s_5 \mapsto \frac{4}{24}\}$. Take the scheduler $\mathcal{S}$:

$$
\begin{aligned}
\mathcal{S}(s) &= \{(s, \tau, \mathbb{1}_{t_2}) \mapsto 2/3, (s, \tau, \mathbb{1}_{t_3}) \mapsto 1/3\} \\
\mathcal{S}(t_2) &= \{(t_2, a, \mathbb{1}_{s_1}) \mapsto 1/2, (t_2, \tau, \mathbb{1}_{t_4}) \mapsto 1/2\} \\
\mathcal{S}(t_3) &= \{(t_3, a, \{s_4 \mapsto 1/2, s_5 \mapsto 1/2\}) \mapsto 1\} \\
\mathcal{S}(t_4) &= \{(t_4, a, \mathbb{1}_{s_2}) \mapsto 3/4, (t_4, a, \{s_2 \mapsto 1/2, s_3 \mapsto 1/2\}) \mapsto 1/4\} \\
\mathcal{S}(t_1) &= \mathcal{S}(s_1) = \mathcal{S}(s_2) = \mathcal{S}(s_3) = \mathcal{S}(s_4) = \mathcal{S}(s_5) = \mathbb{1}_{\perp}
\end{aligned}
$$

Here we used $\mathcal{S}(s)$ to denote the choice made for every possible path ending in s.

The scheduler is depicted in Figure 1(b). Where it chooses probabilistically between two transitions with the same label, this is represented as a *combined transition*. For instance, from t_4 the transition $(t_4, a, \{s_2 \mapsto 1\})$ is selected with



(a) A PA $\mathcal{A}$. (b) Tree of $s \stackrel{a}{\Longrightarrow} \mu$.

Fig. 1. Weak steps.

probability 3/4, and $(t_4, a, \{s_2 \mapsto 1/2, s_3 \mapsto 1/2\})$ with probability 1/4. This corresponds to the combined transition $(t_4, a, \{s_2 \mapsto 7/8, s_3 \mapsto 1/8\})$.

Clearly, all maximal paths enabled from s have trace a and end directly after their a-transition. The path probabilities can also be calculated. For instance,

$$P^{\mathcal{S}}_{\mathcal{A},s}(s \overset{\tau,\{t_2\mapsto 1\}}{\rightsquigarrow} t_2 \overset{\tau,\{t_4\mapsto 1\}}{\rightsquigarrow} t_4 \overset{a,\{s_2\mapsto 1\}}{\rightsquigarrow} s_2) = \left(\tfrac{2}{3}\cdot 1\right)\cdot\left(\tfrac{1}{2}\cdot 1\right)\cdot\left(\tfrac{3}{4}\cdot 1\right) = \tfrac{6}{24}$$

$$P^{\mathcal{S}}_{\mathcal{A},s}(s \overset{\tau,\{t_2\mapsto 1\}}{\rightsquigarrow} t_2 \overset{\tau,\{t_4\mapsto 1\}}{\rightsquigarrow} t_4 \overset{a,\{s_2\mapsto 1/2, s_3\mapsto 1/2\}}{\rightsquigarrow} s_2) = \left(\tfrac{2}{3}\cdot 1\right)\cdot\left(\tfrac{1}{2}\cdot 1\right)\cdot\left(\tfrac{1}{4}\cdot \tfrac{1}{2}\right) = \tfrac{1}{24}$$

As no other maximal paths from s go to s_2, $F^{\mathcal{S}}_{\mathcal{A}}(s)(s_2) = \frac{6}{24} + \frac{1}{24} = \frac{7}{24} = \mu(s_2)$. Similarly, it can be shown that $F^{\mathcal{S}}_{\mathcal{A}}(s)(s_i) = \mu(s_i)$ for every $i \in \{1,3,4,5\}$, so indeed $F^{\mathcal{S}}_{\mathcal{A}}(s) = \mu$. □

3.2 Branching probabilistic bisimulation

Before introducing branching probabilistic bisimulation, we need a restriction on weak steps. Given an equivalence relation R, we let $s \overset{a}{\Longrightarrow}_R \mu$ denote that $(s,t) \in R$ for every state t before the a-step in the tree corresponding to $s \overset{a}{\Longrightarrow} \mu$.

Definition 9 (Branching steps). *Let $\mathcal{A} = \langle S, s^0, L, \Delta\rangle$ be a PA, $s \in S$, and R an equivalence relation over S. Then, $s \overset{a}{\Longrightarrow}_R \mu$ if either (1) $a = \tau$ and $\mu = \mathbb{1}_s$, or (2) there exists a scheduler $\mathcal{S}$ such that $F^{\mathcal{S}}_{\mathcal{A}}(s) = \mu$ and for every maximal path $s \overset{a_1,\mu_1}{\rightsquigarrow} s_1 \overset{a_2,\mu_2}{\rightsquigarrow} s_2 \overset{a_3,\mu_3}{\rightsquigarrow} \dots \overset{a_n,\mu_n}{\rightsquigarrow} s_n \in maxpaths^{\mathcal{S}}_{\mathcal{A}}(s)$ it holds that $a_n = a$, as well as $a_i = \tau$ and $(s, s_i) \in R$ for all $1 \le i < n$.*

Definition 10 (Branching probabilistic bisimulation). *Let $\mathcal{A} = \langle S, s^0, L, \Delta\rangle$ be a PA, then an equivalence relation $R \subseteq S \times S$ is a* branching probabilistic bisimulation *for $\mathcal{A}$ if for all $(s,t) \in R$*

$$s \overset{a}{\to} \mu \text{ implies } \exists \mu' \in \mathrm{Distr}(S) \,.\, t \overset{a}{\Longrightarrow}_R \mu' \wedge \mu \equiv_R \mu'.$$

We say that $p, q \in S$ are branching probabilistically bisimilar, *denoted $p \leftrightarroweq_{\mathrm{bp}} q$, if there exists a branching probabilistic bisimulation R for $\mathcal{A}$ such that $(p,q) \in R$.*

Two PAs are branching probabilistically bisimilar if their initial states are (in the disjoint union of the two systems; see Remark 5.3.4 of [16] for the details).

This notion has some appealing properties. First, the definition is robust in the sense that it can be adapted to using $s \overset{a}{\Longrightarrow}_R \mu$ instead of $s \overset{a}{\to} \mu$ in its condition. Although this might seem to strengthen the concept, it does not. Second, the relation $\leftrightarroweq_{\mathrm{bp}}$ induced by the definition is an equivalence relation.

Proposition 11. *Let $\mathcal{A} = \langle S, s^0, L, \Delta\rangle$ be a PA. Then, an equivalence relation $R \subseteq S \times S$ is a branching probabilistic bisimulation for $\mathcal{A}$ if and only if for all $(s,t) \in R$*

$$s \overset{a}{\Longrightarrow}_R \mu \text{ implies } \exists \mu' \in \mathrm{Distr}(S) \,.\, t \overset{a}{\Longrightarrow}_R \mu' \wedge \mu \equiv_R \mu'.$$

Proposition 12. *The relation $\leftrightarroweq_{\mathrm{bp}}$ is an equivalence relation.*

Moreover, Segala showed that branching bisimulation preserves all properties that can be expressed in the probabilistic temporal logic WPCTL (provided that no infinite path of τ-actions can be scheduled with non-zero probability) [15].

4 Confluence for probabilistic automata

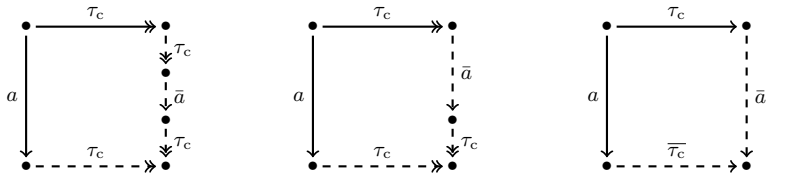
As branching probabilistic bisimulation minimisation cannot easily be performed on-the-fly, we introduce a reduction technique based on sets of *confluent* τ-transitions. Basically, such transitions do not influence a system's behaviour, i.e., a confluent step $s \xrightarrow{\tau} s'$ implies that $s \leftrightarrows_{\mathrm{bp}} s'$. Confluence therefore paves the way for state space reductions modulo branching probabilistic bisimulation (e.g., by giving confluent τ-transitions priority). Not all τ-transitions connect bisimilar states; even though their actions are unobservable, τ-steps might disable behaviour. The aim of our analysis is to efficiently underapproximate which τ-transitions are confluent.

For non-probabilistic systems, several notions of confluence already exist [3]. Basically, they all require that if an action a is enabled from a state that also enables a confluent τ-transition, then (1) a will still be enabled after taking that τ-transition (possibly requiring some additional confluent τ-transitions first), and (2) we can always end up in the same state traversing only confluent τ-steps and the a-step, no matter whether we started by the τ- or the a-transition.

Figure 2 depicts the three notions of confluence we will generalise [3]. Here, the notation τ_{c} is used for confluent τ-transitions. The diagrams should be interpreted as follows: for any state from which the solid transitions are enabled (universally quantified), there should be a matching for the dashed transitions (existentially quantified). A double-headed arrow denotes a path of zero of more transitions with the corresponding label, and an arrow with label $\overline{a}$ denotes a step that is optional in case $a = \tau$ (i.e., its source and target state may then coincide). The weaker the notion, the more reduction potentially can be achieved (although detection is harder). Note that we first need to find a subset of τ-transitions that we believe are confluent; then, the diagrams are checked.

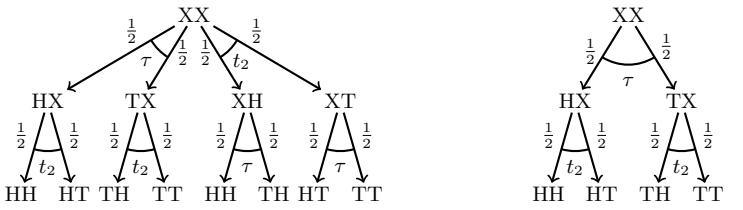
For probabilistic systems, no similar notions of confluence have been defined before. The situation is indeed more difficult, as transitions do not have a single target state anymore. To still enable reductions based on confluence, only τ-transitions with a unique target state might be considered confluent. The next example shows what goes wrong without this precaution. For brevity, from now on we use *bisimilar* as an abbreviation for *branching probabilistically bisimilar*.

Example 13. Consider two people each throwing a die. The PA in Figure 3(a) models this behaviour given that it is unknown who throws first. The first charac-



(a) Weak confluence. (b) Confluence. (c) Strong confluence.

Fig. 2. Three variants of confluence.



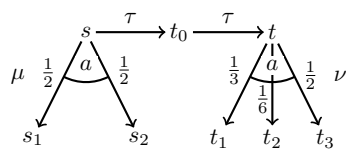
(a) The original specification. (b) A wrong reduction.

Fig. 3. Two people throwing dice.

ter of each state name indicates whether the first player has not thrown yet (X), or threw heads (H) or tails (T), and the second character indicates the same for the second player. For lay-out purposes, some states were drawn twice.

We hid the first player's throw action, and kept the other one visible. Now, it might appear that the order in which the t_2- and the τ-transition occur does not influence the behaviour. However, the τ-step does not connect bisimilar states (assuming HH, HT, TH, and TT to be distinct). After all, from state XX it is possible to reach a state (XH) from where HH is reached with probability 0.5 and TH with probability 0.5. From HX and TX no such state is reachable anymore. Giving the τ-transition priority, as depicted in Figure 3(b), therefore yields a reduced system that is *not* bisimilar to the original system anymore. □

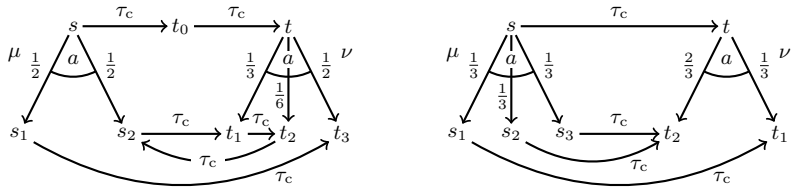
Another difficulty arises in the probabilistic setting. Although for LTSs it is clear that a path $a\tau$ should reach the same state as τa, for PAs this is more involved as the a-step leads us to a distribution over states. So, how should the model shown here on the right be completed for the τ-steps to be confluent?



Since we want confluent τ-transitions to connect bisimilar states, we must assure that s, t_0, and t are bisimilar. Therefore, μ and ν must assign equal probabilities to each *class* of bisimilar states. Basically, given the assumption that the other confluent τ-transitions already connect bisimilar states, this is the case if $\mu \equiv_R \nu$ for $R = \{(s, s') \mid s \xrightarrow{\tau} \!\!\!\!\twoheadrightarrow\!\!\!\!\twoheadleftarrow^{\tau} s'$ using only confluent τ-steps$\}$. The following definition formalises these observations. Here we use the notation $s \xrightarrow{\tau_c} s'$, given a set of τ-transitions c, to denote that $s \xrightarrow{\tau} s'$ and $(s, \tau, s') \in c$.

We define three notions of probabilistic confluence, all requiring the target state of a confluent step to be able to mimic the behaviour of its source state. In the weak version, mimicking may be postponed and is based on joinability (Definition 14a). In the default version, mimicking must happen immediately, but is still based on joinability (Definition 14b). Finally, the strong version requires immediate mimicking by directed steps (Definition 16).

Definition 14 ((Weak) probabilistic confluence). *Let $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ be a PA and $c \subseteq \{(s, a, \mu) \in \Delta \mid a = \tau, \mu$ is deterministic$\}$ a set of τ-transitions. (a) Then, c is* weakly probabilistically confluent *if $R = \{(s, s') \mid s \xrightarrow{\tau_c} \!\!\!\!\twoheadrightarrow\!\!\!\!\twoheadleftarrow^{\tau_c} s'\}$ is*



(a) Weak probabilistic confluence. (b) Strong probabilistic confluence.

Fig. 4. Weak versus strong probabilistic confluence.

an equivalence relation, and for every path $s \xrightarrow{\tau_c}\!\!\!\!\rightarrow t$ *and all* $a \in L, \mu \in \mathrm{Distr}(S)$

$$s \xrightarrow{a} \mu \implies \exists t' \in S \,.\, t \xrightarrow{\tau_c}\!\!\!\!\rightarrow t' \wedge \big(\big(\exists \nu \in \mathrm{Distr}(S) \,.\, t' \xrightarrow{a} \nu \wedge \mu \equiv_R \nu\big) \vee (a = \tau \wedge \mu \equiv_R \mathbb{1}_{t'})\big).$$

(b) If for every path $s \xrightarrow{\tau_c}\!\!\!\!\rightarrow t$ *and every transition* $s \xrightarrow{a} \mu$ *the above implication can be satisfied by taking* $t' = t$*, then we say that* c *is* probabilistically confluent.

For the strongest variant of confluence, moreover, we require the target states of μ to be connected by direct τ_c-transitions to the target states of ν:

Definition 15 (Equivalence up to τ_c-steps). *Let* μ, ν *be two probability distributions, and let* $\nu = \{t_1 \mapsto p_1, t_2 \mapsto p_2, \dots\}$*. Then,* μ *is* equivalent to ν up to τ_c-steps*, denoted by* $\mu \overset{\tau_c}{\rightsquigarrow} \nu$*, if there exists a partition* $\mathrm{spt}(\mu) = \biguplus_{i=1}^{n} S_i$ *such that* $n = |\mathrm{spt}(\nu)|$ *and* $\forall 1 \le i \le n \,.\, \mu(S_i) = \nu(t_i) \wedge \forall s \in S_i \,.\, s \xrightarrow{\tau_c}\!\!\!\!\rightarrow t_i$*.*

Definition 16 (Strong probabilistic confluence). *Let* $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ *be a PA and* $c \subseteq \{(s, a, \mu) \in \Delta \mid a = \tau, \mu \text{ is deterministic}\}$ *a set of* τ*-transitions, then* c *is* strongly probabilistically confluent *if for all* $s \xrightarrow{\tau_c} t, a \in L, \mu \in \mathrm{Distr}(S)$

$$s \xrightarrow{a} \mu \implies \big(\big(\exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a} \nu \wedge \mu \overset{\tau_c}{\rightsquigarrow} \nu\big) \vee (a = \tau \wedge \mu = \mathbb{1}_t)\big)\,.$$

Proposition 17. *Strong probabilistic confluence implies probabilistic confluence, and probabilistic confluence implies weak probabilistic confluence.*

A transition $s \xrightarrow{\tau} t$ is called (weakly, strongly) probabilistically confluent if there exists a (weakly, strongly) probabilistically confluent set c such that $(s, \tau, t) \in c$.

Example 18. Observe the PAs in Figure 4. Assume that all transitions of s, t_0 and t are shown, and that all s_i, t_i, are potentially distinct. We marked all τ-transitions as being confluent, and will verify this for some of them.

In Figure 4(a), both the upper τ_c-steps are weakly probabilistically confluent, most interestingly $s \xrightarrow{\tau_c}\!\!\!\!\rightarrow t_0$. To verify this, first note that $t_0 \xrightarrow{\tau_c}\!\!\!\!\rightarrow t$ is (as t_0 has no other outgoing transitions), from where the a-transition of s can be mimicked. To see that indeed $\mu \equiv_R \nu$ (using R from Definition 14), observe that R yields three equivalence classes: $C_1 = \{s_2, t_1, t_2\}$, $C_2 = \{s_1, t_3\}$ and $C_3 = \{s, t_0, t\}$. As

required, $\mu(C_1) = \frac{1}{2} = \nu(C_1)$ and $\mu(C_2) = \frac{1}{2} = \nu(C_2)$. Clearly $s \xrightarrow{\tau_c} t_0$ is not probabilistically confluent, as t_0 cannot immediately mimic the a-transition of s.

In Figure 4(b) the upper τ_c-transition is strongly probabilistically confluent (and therefore also (weakly) probabilistically confluent), as t is able to directly mimic the a-transition from s via $t \xrightarrow{a} \nu$. As required, $\mu \overset{\tau_c}{\rightsquigarrow} \nu$ also holds, which is easily seen by taking the partition $S_1 = \{s_1\}, S_2 = \{s_2, s_3\}$. □

The following theorem shows that weakly probabilistically confluent τ-transitions indeed connect bisimilar states. With Proposition 17 in mind, this also holds for (strong) probabilistic confluence. Additionally, we show that confluent sets can be joined (so there is a unique maximal confluent set of τ-transitions).

Theorem 19. *Let $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ be a PA, $s, s' \in S$ two of its states, and c a weakly probabilistically confluent subset of its τ-transitions. Then,*

$$s \mathrel{\overset{\tau_c}{\twoheadleftarrow\!\!\twoheadrightarrow}} s' \text{ implies } s \leftrightarrows_{\mathrm{bp}} s'.$$

Proposition 20. *Let c, c' be (weakly, strongly) probabilistically confluent sets of τ-transitions. Then, $c \cup c'$ is also (weakly, strongly) probabilistically confluent.*

5 State space reduction using probabilistic confluence

As confluent τ-transitions connect branching probabilistic bisimilar states, all states that can reach each other via such transitions can be merged. That is, we can take the original PA modulo the equivalence relation $\overset{\tau_c}{\twoheadleftarrow\!\!\twoheadrightarrow}$ and obtain a reduced and bisimilar system. The downside of this method is that, in general, it is hard to compute the equivalence classes according to $\overset{\tau_c}{\twoheadleftarrow\!\!\twoheadrightarrow}$. Therefore, a slightly adapted reduction technique was proposed in [3], and later used in [4]. It chooses a representative state s for each equivalence class, such that all transitions leaving the equivalence class are directly enabled from s. This method relies on (strong) probabilistic confluence, and does not work for the weak variant.

To find a valid representative, we first look at the directed (unlabeled) graph $G = (S, \xrightarrow{\tau_c})$. It contains all states of the original system, and denotes precisely which states can reach each other by taking only τ_c-transitions. Because of the restrictions on τ_c-transitions, the subgraph of G corresponding to each equivalence class $[s]_{\overset{\tau_c}{\twoheadleftarrow\!\!\twoheadrightarrow}}$ has exactly one terminal strongly connected component (TSCC), from which the representative state for that equivalence class should be chosen. Intuitively, this follows from the fact that τ_c-transitions always lead to a state with at least the same observable transitions as the previous state, and maybe more. (This is not the case for weak probabilistic confluence, therefore the reduction using representatives does not work for that variant of confluence.) The next definition formalises these observations.

Definition 21 (Representation maps). *Let $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ be a PA and c a subset of its τ-transitions. Then, a function $\phi_c \colon S \to S$ is a* representation map *for $\mathcal{A}$ under c if*

- $\forall s, s' \in S \,.\, s \xrightarrow{\tau_c} s' \implies \phi_c(s) = \phi_c(s')$;
- $\forall s \in S \,.\, s \overset{\tau_c}{\twoheadrightarrow} \phi_c(s)$.

The first condition ensures that equivalent states are mapped to the same representative, and the second makes sure that every representative is in a TSCC. If c is a probabilistically confluent set of τ-transitions, the second condition and Theorem 19 immediately imply that $s \leftrightarrows_{\mathrm{bp}} \phi_c(s)$ for every state s.

The next proposition states that for finite-state PAs and probabilistically confluent sets c, there always exists a representation map. As τ_{c}-transitions are required to always have a deterministic distribution, probabilities are not involved and the proof is identical to the proof for the non-probabilistic case [3].

Proposition 22. *Let $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ be a PA and c a probabilistically confluent subset of its τ-transitions. Moreover, let S be finite. Then, there exists a function $\phi_c \colon S \to S$ such that ϕ_c is a representation map for $\mathcal{A}$ under c.*

We can now define a PA modulo a representation map ϕ_c. The set of states of such a PA consists of all representatives. When originally $s \xrightarrow{a} \mu$ for some state s, in the reduced system $\phi_c(s) \xrightarrow{a} \mu'$ where μ' assigns a probability to each representative equal to the probability of reaching any state that maps to this representative in the original system. The system will not have any τ_{c}-transitions.

Definition 23 ($\mathcal{A}/\phi_c$). *Let $\mathcal{A} = \langle S, s^0, L, \Delta \rangle$ be a PA and c a set of τ-transitions. Moreover, let ϕ_c be a representation map for $\mathcal{A}$ under c. Then, we write $\mathcal{A}/\phi_c$ to denote the PA $\mathcal{A}$ modulo ϕ_c. That is,*

$$\mathcal{A}/\phi_c = \langle \phi_c(S), \phi_c(s^0), L, \Delta_{\phi_c} \rangle,$$

where $\phi_c(S) = \{\phi_c(s) \mid s \in S\}$, and $\Delta_{\phi_c} \subseteq \phi_c(S) \times L \times \mathrm{Distr}(\phi_c(S))$ such that $s \xrightarrow{a}_{\phi_c} \mu$ if and only if $a \neq \tau_{\mathrm{c}}$ and there exists a transition $t \xrightarrow{a} \mu'$ in $\mathcal{A}$ such that $\phi_c(t) = s$ and $\forall s' \in \phi_c(S) \,.\, \mu(s') = \mu'(\{s'' \in S \mid \phi_c(s'') = s'\})$.

From the construction of the representation map it follows that $\mathcal{A}/\phi_{\mathrm{c}} \leftrightarrows_{\mathrm{bp}} \mathcal{A}$ if c is (strongly) probabilistically confluent.

Theorem 24. *Let $\mathcal{A}$ be a PA and c a probabilistically confluent set of τ-transitions. Also, let ϕ_c be a representation map for $\mathcal{A}$ under c. Then, $(\mathcal{A}/\phi_c) \leftrightarrows_{\mathrm{bp}} \mathcal{A}$.*

Using this result, state space generation of PAs can be optimised in exactly the same way as has been done for the non-probabilistic setting [4]. Basically, every state visited during the generation is replaced on-the-fly by its representative. In the absence of τ-loops this is easy; just repeatedly follow confluent τ-transitions until none are enabled anymore. When τ-loops are present, a variant of Tarjan's algorithm for finding SCCs can be applied (see [3] for the details).

6 Symbolic detection of probabilistic confluence

Before any reductions can be obtained in practice, probabilistically confluent τ-transitions need to be detected. As our goal is to prevent the generation of large state spaces, this has to be done symbolically.

We propose to do so in the framework of prCRL and LPPEs [10], where systems are modelled by a process algebra and every specification is *linearised*

to an intermediate format: the LPPE (linear probabilistic process equation). Basically, an LPPE is a process X with a vector of global variables $\boldsymbol{g}$ of type $\boldsymbol{G}$ and a set of *summands*. A summand is a symbolic transition that is chosen nondeterministically, provided that its guard is enabled (similar to a guarded command). Each summand i is of the form

$$\sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i(\boldsymbol{g},\boldsymbol{d_i}) \Rightarrow a_i(\boldsymbol{g},\boldsymbol{d_i}) \sum_{\boldsymbol{e_i}:\boldsymbol{E_i}} f_i(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e_i})\colon X(\boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e_i})).$$

Here, $\boldsymbol{d_i}$ is a (possibly empty) vector of local variables of type $\boldsymbol{D_i}$, which is chosen nondeterministically such that the condition c_i holds. Then, the action $a_i(\boldsymbol{g},\boldsymbol{d_i})$ is taken and a vector $\boldsymbol{e_i}$ of type $\boldsymbol{E_i}$ is chosen probabilistically (each $\boldsymbol{e_i}$ with probability $f_i(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e_i})$). Then, the next state is set to $\boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e_i})$.

The semantics of an LPPE is given as a PA, whose states are precisely all vectors $\boldsymbol{g} \in \boldsymbol{G}$. For all $\boldsymbol{g} \in \boldsymbol{G}$, there is a transition $\boldsymbol{g} \xrightarrow{a} \mu$ if and only if for at least one summand i there is a choice of local variables $\boldsymbol{d_i} \in \boldsymbol{D_i}$ such that

$$c_i(\boldsymbol{g},\boldsymbol{d_i}) \wedge a_i(\boldsymbol{g},\boldsymbol{d_i}) = a \wedge \forall \boldsymbol{e_i} \in \boldsymbol{E_i}\ .\ \mu(\boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e_i})) = \sum_{\substack{\boldsymbol{e'_i} \in \boldsymbol{E_i} \\ \boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e_i}) = \boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e'_i})}} f_i(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e'_i}).$$

Example 25. As an example of an LPPE, observe the following specification:

$$X(pc : \{1,2\}) = \sum_{n:\{1,2,3\}} pc = 1 \Rightarrow \mathrm{output}(n) \sum_{i:\{1,2\}} \tfrac{i}{3}\colon X(i) \qquad (1)$$

$$+ \qquad pc = 2 \Rightarrow \mathrm{beep} \sum_{j:\{1\}} 1\colon X(j) \qquad (2)$$

The system has one global variable pc (which can be either 1 or 2), and consists of two summands. When $pc = 1$, the first summand is enabled and the system nondeterministically chooses n to be 1, 2 or 3, and outputs the chosen number. Then, the next state is chosen probabilistically; with probability $\frac{1}{3}$ it will be $X(1)$, and with probability $\frac{2}{3}$ it will be $X(2)$. When $pc = 2$, the second summand is enabled, making the system beep and deterministically returning to $X(1)$.

In general, the conditions and actions may depend on both the global variables (in this case pc) and the local variables (in this case n for the first summand), and the probabilities and expressions for determining the next state may additionally depend on the probabilistic variables (in this case i and j). □

Instead of designating *individual* τ-transitions to be probabilistically confluent, we designate *summands* to be so in case we are sure that *all* transitions they might generate are probabilistically confluent. For a summand i to be confluent, clearly $a_i(\boldsymbol{g},\boldsymbol{d_i}) = \tau$ should hold for all possible values of $\boldsymbol{g}$ and $\boldsymbol{d_i}$. Also, the next state of each of the transitions it generates should be unique: for every possible valuation of $\boldsymbol{g}$ and $\boldsymbol{d_i}$, there should be a single $\boldsymbol{e_i}$ such that $f_i(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e_i}) = 1$.

Moreover, a confluence property should hold. For efficiency, we detect a strong variant of strong probabilistic confluence. Basically, a confluent τ-summand i has

to commute properly with every summand j (including itself). More precisely, when both are enabled, executing one should not disable the other and the order of their execution should not influence the observable behaviour or the final state. Additionally, i commutes with itself if it generates only one transition. Formally:

$$\big(c_i(\boldsymbol{g},\boldsymbol{d_i}) \wedge c_j(\boldsymbol{g},\boldsymbol{d_j})\big) \rightarrow \big(i = j \wedge \boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i}) = \boldsymbol{n_j}(\boldsymbol{g},\boldsymbol{d_j})\big) \vee \begin{pmatrix} c_j(\boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i}),\boldsymbol{d_j}) \wedge c_i(\boldsymbol{n_j}(\boldsymbol{g},\boldsymbol{d_j},\boldsymbol{e_j}),\boldsymbol{d_i}) \\ \wedge\ a_j(\boldsymbol{g},\boldsymbol{d_j}) = a_j(\boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i}),\boldsymbol{d_j}) \\ \wedge\ f_j(\boldsymbol{g},\boldsymbol{d_j},\boldsymbol{e_j}) = f_j(\boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i}),\boldsymbol{d_j},\boldsymbol{e_j}) \\ \wedge\ \boldsymbol{n_j}(\boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i}),\boldsymbol{d_j},\boldsymbol{e_j}) = \boldsymbol{n_i}(\boldsymbol{n_j}(\boldsymbol{g},\boldsymbol{d_j},\boldsymbol{e_j}),\boldsymbol{d_i}) \end{pmatrix} \quad (1)$$

where $\boldsymbol{g}, \boldsymbol{d_i}, \boldsymbol{d_j}$ and $\boldsymbol{e_j}$ universally quantify over $\boldsymbol{G}$, $\boldsymbol{D_i}$, $\boldsymbol{D_j}$, and $\boldsymbol{E_j}$, respectively. We used $\boldsymbol{n_i}(\boldsymbol{g},\boldsymbol{d_i})$ to denote the unique target state of summand i given global state $\boldsymbol{g}$ and local state $\boldsymbol{d_i}$ (so $\boldsymbol{e_i}$ does not need to appear).

As these formulas are quantifier-free and in practice often trivially false or true, they can easily be solved using an SMT solver for the data types involved. For n summands, n^2 formulas need to be solved; the complexity of this depends on the data types. In our experiments, all formulas could be checked with fairly simple heuristics (e.g., validating them vacuously by finding contradictory conditions or by detecting that two summands never use or change the same variable).

Theorem 26. *Let X be an LPPE and $\mathcal{A}$ its PA. Then, if for a summand i we have $\forall \boldsymbol{g} \in \boldsymbol{G}, \boldsymbol{d_i} \in \boldsymbol{D_i}\ .\ a_i(\boldsymbol{g},\boldsymbol{d_i}) = \tau \wedge \exists \boldsymbol{e_i} \in \boldsymbol{E_i}\ .\ f_i(\boldsymbol{g},\boldsymbol{d_i},\boldsymbol{e_i}) = 1$ and formula (1) holds, the set of transitions generated by i is strongly probabilistically confluent.*

7 Case study

To illustrate the power of probabilistic confluence reduction, we applied it on two leader election protocols. We implemented a prototype tool in Haskell for confluence detection using heuristics and state space generation based on confluence information, relying on Theorem 26 and Theorem 24. The results were obtained on a 2.4 GHz, 2 GB Intel Core 2 Duo MacBook[1].

First, we analysed the leader election protocol introduced in [10]. This protocol, between two nodes, decides on a leader by having both parties throw a die and compare the results. In case of a tie the nodes throw again, otherwise the one that threw highest will be the leader. We hid all actions needed for rolling the dice and communication, keeping only the declarations of leader and follower. The complete model in LPPE format can be found in [17].

In [10] we showed the effect of dead-variable reduction [18] on this system. Now, we apply probabilistic confluence reduction both to the LPPE that was already reduced in this way (`basicReduced`) and the original one (`basicOriginal`).

The results are shown in Table 1; we list the size of the original and reduced state space, as well as the number of states and transitions that were visited

[1] The implementation, case studies and a test script can be downloaded from `http://fmt.cs.utwente.nl/~timmer/prcrl/papers/TACAS2011`.

Table 1. Applying confluence reduction to two leader election protocols.

	Original		Reduced		Visited		Runtime (sec)	
Specification	States	Trans.	States	Trans.	States	Trans.	Before	After
`basicOriginal`	3,763	6,158	631	758	3,181	3,290	0.45	0.22
`basicReduced`	1,693	2,438	541	638	1,249	1,370	0.22	0.13
`leader-3-12`	161,803	268,515	35,485	41,829	130,905	137,679	67.37	31.53
`leader-3-15`	311,536	515,328	68,926	80,838	251,226	264,123	145.17	65.82
`leader-3-18`	533,170	880,023	118,675	138,720	428,940	450,867	277.08	122.59
`leader-3-21`	840,799	1,385,604	187,972	219,201	675,225	709,656	817.67	211.87
`leader-3-24`	1,248,517	2,055,075	280,057	326,007	1,001,259	1,052,235	1069.71	333.32
`leader-3-27`	out of memory		398,170	462,864	1,418,220	1,490,349	–	503.85
`leader-4-5`	443,840	939,264	61,920	92,304	300,569	324,547	206.56	75.66
`leader-4-6`	894,299	1,880,800	127,579	188,044	608,799	655,986	429.87	155.96
`leader-4-7`	1,622,682	3,397,104	235,310	344,040	1,108,391	1,192,695	1658.38	294.09
`leader-4-8`	out of memory		400,125	581,468	1,865,627	2,005,676	–	653.60
`leader-5-2`	208,632	561,630	14,978	29,420	97,006	110,118	125.78	30.14
`leader-5-3`	1,390,970	3,645,135	112,559	208,170	694,182	774,459	1504.33	213.85
`leader-5-4`	out of memory		472,535	847,620	2,826,406	3,129,604	–	7171.73

during its generation using confluence. Probabilistic confluence reduction clearly has quite an effect on the size of the state space, as well as the running time. Notice also that it nicely works hand-in-hand with dead-variable reduction.

Second, we analysed several versions of a leader election protocol that uses asynchronous channels and allows for more parties (Algorithm $\mathcal{B}$ from [6]). We denote by `leader-i-j` the variant with `i` parties each throwing a `j`-sided die, that was already optimised using dead-variable reduction. Confluence additionally reduces the number of states and transitions by 77% – 92% and 84% – 94%, respectively. Consequently, the running times more than halve. With probabilistic POR, relatively smaller reductions were obtained for similar protocols [9].

For each experiment, linearisation and confluence detection only took a fraction of time. For the larger state spaces swapping occured, explaining the growth in running time. Confluence clearly allows us to do more before reaching this limit.

8 Conclusions

This paper introduced three new notions of confluence for probabilistic automata. We first established several facts about these notions, most importantly that they identify branching probabilistically bisimilar states. Then, we showed how probabilistic confluence can be used for state space reduction. As we used representatives in terminal strongly connected components, these reductions can even be applied to systems containing τ-loops. We discussed how confluence can be detected in the context of a probabilistic process algebra with data by proving formulas in first-order logic. This way, we enabled on-the-fly reductions when generating the state space corresponding to a process-algebraic specification. A case study illustrated the power of our methods.

References

[1] C. Baier, P.R. D'Argenio, and M. Größer. Partial order reduction for probabilistic branching time. In *Proc. of the 3rd Workshop on Quantitative Aspects of*

Programming Languages (QAPL), volume 153(2) of *ENTCS*, pages 97–116, 2006.

[2] C. Baier, M. Größer, and F. Ciesinski. Partial order reduction for probabilistic systems. In *Proc. of the 1st International Conference on Quantitative Evaluation of Systems (QEST)*, pages 230–239. IEEE Computer Society, 2004.

[3] S.C.C. Blom. Partial τ-confluence for efficient state space generation. Technical Report SEN-R0123, CWI, Amsterdam, 2001.

[4] S.C.C. Blom and J.C. van de Pol. State space reduction by proving confluence. In *Proc. of the 14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *LNCS*, pages 596–609. Springer, 2002.

[5] P.R. D'Argenio and P. Niebert. Partial order reduction on concurrent probabilistic programs. In *Proc. of the 1st International Conference on Quantitative Evaluation of Systems (QEST)*, pages 240–249. IEEE Computer Society, 2004.

[6] W. Fokkink and J. Pang. Simplifying Itai-Rodeh leader election for anonymous rings. In *Proc. of the 4th International Workshop on Automated Verification of Critical Systems (AVoCS)*, volume 128(6) of *ENTCS*, pages 53–68, 2005.

[7] S. Giro, P.R. D'Argenio, and L. María Ferrer Fioriti. Partial order reduction for probabilistic systems: A revision for distributed schedulers. In *Proc. of the 20th International Conference on Concurrency Theory (CONCUR)*, volume 5710 of *LNCS*, pages 338–353. Springer, 2009.

[8] J.F. Groote and M.P.A. Sellink. Confluence for process verification. *Theoretical Computer Science*, 170(1-2):47–81, 1996.

[9] M. Größer. *Reduction Methods for Probabilistic Model Checking*. PhD thesis, Technische Universität Dresden, 2008.

[10] J.-P. Katoen, J.C. van de Pol, M.I.A. Stoelinga, and M. Timmer. A linear process-algebraic format for probabilistic systems with data. In *Proc. of the 10th International Conference on Application of Concurrency to System Design (ACSD)*, pages 213–222. IEEE Computer Society, 2010.

[11] R. De Nicola and F.W. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Systems of Concurrent Processes*, volume 469 of *LNCS*, pages 407–419. Springer, 1990.

[12] G.J. Pace, F. Lang, and R. Mateescu. Calculating τ-confluence compositionally. In *Proc. of the 15th International Conference on Computer Aided Verification (CAV)*, volume 2725 of *LNCS*, pages 446–459. Springer, 2003.

[13] D. Peled. All from one, one for all: on model checking using representatives. In *Proc. of the 5th International Conference on Computer Aided Verification (CAV)*, volume 697 of *LNCS*, pages 409–423. Springer, 1993.

[14] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

[15] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computation*, 2(2):250–273, 1995.

[16] M.I.A. Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, 2002.

[17] M. Timmer, M.I.A. Stoelinga, and J.C. van de Pol. Confluence reduction for probabilistic systems (extended version). Technical Report 1011.2314, ArXiv e-prints, 2010.

[18] J.C. van de Pol and M. Timmer. State space reduction of linear processes using control flow reconstruction. In *Proc. of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 5799 of *LNCS*, pages 54–68. Springer, 2009.

[19] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.