On-the-fly Confluence Detection for Statistical Model Checking[⋆]

Arnd Hartmanns[1] and Mark Timmer[2]

[1] Saarland University – Computer Science, Saarbrücken, Germany
[2] Formal Methods and Tools, University of Twente, The Netherlands

Abstract Statistical model checking is an analysis method that circumvents the state space explosion problem in model-based verification by combining probabilistic simulation with statistical methods that provide clear error bounds. As a simulation-based technique, it can only provide sound results if the underlying model is a stochastic process. In verification, however, models are usually variations of nondeterministic transition systems. The notion of confluence allows the reduction of such transition systems in classical model checking by removing spurious nondeterministic choices. In this paper, we show that confluence can be adapted to detect and discard such choices on-the-fly during simulation, thus extending the applicability of statistical model checking to a subclass of Markov decision processes. In contrast to previous approaches that use partial order reduction, the confluence-based technique can handle additional kinds of nondeterminism. In particular, it is not restricted to interleavings. We evaluate our approach, which is implemented as part of the modes simulator for the Modest modelling language, on a set of examples that highlight its strengths and limitations and show the improvements compared to the partial order-based method.

1 Introduction

Traditional and probabilistic model checking have grown to be useful techniques for finding inconsistencies in designs and computing quantitative aspects of systems and protocols. However, model checking is subject to the state space explosion problem, with probabilistic model checking being particularly affected due to its additional numerical complexity. Several techniques have been introduced to stretch the limits of model checking while preserving its basic nature of performing state space exploration to obtain results that unconditionally, certainly hold for the entire state space. Two of them, partial order reduction (POR) and confluence reduction, work by selecting a subset of the transitions of a model—and thus a subset of the reachable states—in a way that ensures that the reduced system is equivalent to the complete system. POR was first generalised to the probabilistic domain preserving linear time properties [2,10], with a

[⋆] This work has been supported by the DFG/NWO Bilateral Research Program ROCKS, by NWO under grant 612.063.817 (SYRUP), by the EU FP7-ICT project MEALS, contract no. 295261, and by the DFG as part of SFB/TR 14 AVACS.

later extension to preserve branching time properties [1]. Confluence reduction was generalised in [13,23], preserving branching time properties.

A much different approach for probabilistic models is statistical model checking (SMC) [18,21,26]: instead of exploring—and storing in memory—the entire state space, or even a reduced version of it, discrete-event simulation is used to generate traces through the state space. This comes at constant memory usage and thus circumvents state space explosion entirely, but cannot deliver results that hold with absolute certainty. Statistical methods such as sequential hypothesis testing are then used to make sure that the *probability* of returning the wrong result is below a certain threshold. As a simulation-based approach, however, SMC is limited to fully stochastic models such as Markov chains [14].

Previously, an approach based on POR was presented [6] to extend SMC and simulation to the nondeterministic model of Markov decision processes (MDPs). In that approach, simulation proceeds as usual until a nondeterministic choice is encountered; at that point, an on-the-fly check is performed to find a singleton subset of the available transitions that satisfies the *ample set* conditions of probabilistic POR [2,10]. If such an ample set is found, simulation can continue that way with the guarantee that ignoring the other transitions does not affect the verification results, i.e., the nondeterminism was *spurious*. Yet, the ample set conditions are based on the notion of *independence* of actions, which can in practice only feasibly be checked on a symbolic/syntactic level (using conditions such as J1 and J2 in [6]). This limits the approach to resolve spurious nondeterminism only when it results from the *interleaving* of behaviours of concurrently executing (deterministic) components.

In this paper, we present as an alternative to use confluence reduction, which has recently been shown theoretically to be more powerful than branching time POR [13]. It is absolutely vital for the search for a valid singleton subset to succeed in the approach discussed above: one choice that cannot be resolved means that the entire analysis fails and SMC cannot safely be applied to the given model at all. Therefore, any additional reduction power is highly welcome. Furthermore, in practice, confluence reduction is easily implemented on the level of the concrete state space alone, without any need to go back to the symbolic/syntactic level for an independence check. As opposed to the approach in [6], it thus allows even spurious nondeterminism that is internal to components to be ignored during simulation. Of course, models containing non-spurious nondeterminism can still not be dealt with.

Contributions and outline. After the introduction of the necessary preliminaries (Section 2), we present the three main contributions of this paper: (1) Since simulation works with a fully composed, closed system, we can relax the definition of confluence with respect to action labels compared to [13] (Section 3). We thus achieve more reduction/detection power at no computational cost; yet, we can prove that this adapted notion of confluence still preserves PCTL* formulae [3] without the *next* operator. (2) We then introduce an algorithm for detecting our new notion of probabilistic confluence on a concrete state space and state its correctness (Section 4). The algorithm is inspired by, but different

Table 1. SMC approaches for nondeterministic models (with n states)

approach	nondeterminism	probabilities	memory	error bounds
POR-based [6]	spurious interleavings	max = min	$\underline{s \ll n}$	$\underline{\text{unchanged}}$
confluence-based	spurious	max = min	$\underline{s \ll n}$	$\underline{\text{unchanged}}$
learning [17]	$\underline{\text{any}}$	max only	$s \to n$	convergence

from, the one given in [12]; in particular, it does not require initial knowledge of the entire state space and can therefore be used on-the-fly during simulation. (3) Finally, we evaluate the new confluence-based approach to SMC on a set of three representative examples using our implementation within the modes statistical model checker [7] for the Modest modelling language [8] (Section 5). We clearly identify its strengths and limitations. Since the previous POR-based approach is also implemented in modes, we compare the two in terms of reduction power and, on the one case that can actually be handled by the POR-based implementation as well, performance. Proofs for all our results can be found in [16].

Related work. Aside from [6] and an approach that focuses on planning problems and infinite-state models [20], the only other solution to the problem of nondeterminism in SMC that we are aware of is recent work by Henriques et al. [17]. They use reinforcement learning, a technique from artificial intelligence, to actually learn the resolutions of nondeterminism (by memoryless schedulers) that *maximise* probabilities for a given bounded LTL property. While this allows SMC for models with arbitrary nondeterministic choices (not only spurious ones), scheduling decisions need to be stored for every *explored* state. Memory usage can thus be as in traditional model checking, but is highly dependent on the structure of the model and the learning process. As the number of runs of the algorithm increases, the answer it returns will converge to the actual result, but definite error probabilities are not given. The approaches based on confluence and POR do not introduce any additional overapproximation and thus have no influence on the usual error bounds of SMC. Table 1 gives a condensed overview of the three approaches (where we measure memory usage in terms of the maximal number of states s stored at any time; see Section 5 for concrete values).

2 Preliminaries

Definition 1 (Basics). *A* probability distribution *over a countable set S is a function $\mu\colon S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. We denote by $\mathsf{Distr}(S)$ the set of all such functions. For $S' \subseteq S$, let $\mu(S') = \sum_{s \in S'} \mu(s)$. We let $\mathsf{support}(\mu) = \{s \in S \mid \mu(s) > 0\}$ be the* support *of μ, and write $\mathbb{1}_s$ for the* Dirac distribution *for s, determined by $\mathbb{1}_s(s) = 1$.*

Given an equivalence relation $R \subseteq S \times S$, we write $[s]_R$ for the equivalence class induced by s, i.e. $[s]_R = \{s' \in S \mid (s, s') \in R\}$. We denote the set of all such equivalence classes by S/R. Given two probability distributions μ, μ' over S, we write $\mu \equiv_R \mu'$ to denote that $\mu([s]_R) = \mu'([s]_R)$ for every $s \in S$.

Our analyses are based on the model of Markov decision processes (MDPs, or equivalently probabilistic automata, PAs), which combines nondeterministic and probabilistic choices. In the variant we use states are labelled by a set of atomic propositions.

Definition 2 (MDPs). *A* Markov decision process (MDP) *is a tuple* $\mathcal{A} = (S, \Sigma, P, s^0, \mathsf{AP}, L)$*, where*

- S *is a countable set of* states*, of which* $s^0 \in S$ *is the* initial state*;*
- Σ *is a finite set of* action labels*;*
- $P \subseteq S \times \Sigma \times \mathsf{Distr}(S)$ *is the* probabilistic transition relation*;*
- AP *is the set of* atomic propositions*;*
- $L \colon S \to \mathcal{P}(\mathsf{AP})$ *is the* labelling function*.*

If $(s, a, \mu) \in P$*, we write* $s \xrightarrow{a} \mu$ *and mean that it is possible to take an* a*-action from* s *and have a probability of* $\mu(s')$ *to go to* s'*. Given a state* $s \in S$*, we define its set of* enabled *transitions* $en(s) = \{(s, a, \mu) \in \{s\} \times \Sigma \times \mathsf{Distr}(S) \mid s \xrightarrow{a} \mu\}$*.*

We will use $S_{\mathcal{A}}$*,* $\Sigma_{\mathcal{A}}$*, ..., to refer to the components of an MDP* $\mathcal{A}$*. If the MDP is clear from the context, these subscripts are omitted.*

We work in a state-based verification setting where properties only refer to the atomic propositions of states. The action labels are solely meant for synchronisation during parallel composition. Since we consider closed systems only, we can therefore ignore them. We do care about whether or not transitions change the observable behaviour of the system, i.e., the atomic propositions:

Definition 3 (Visibility and determinism). *A transition* $s \xrightarrow{a} \mu$ *in an MDP* $\mathcal{A}$ *is called* visible *if* $\exists\, t \in \mathsf{support}(\mu) \colon L(s) \neq L(t)$*. Otherwise, it is* invisible*. A transition* $s \xrightarrow{a} \mu$ *is* deterministic *if* $\mu(t) = 1$ *for some* $t \in S$*, i.e.,* $\mu = \mathbb{1}_t$*.*

We write $s \xrightarrow{\tau} \mu$ *to indicate that a transition is invisible. Transitions labelled by a letter different from* τ *can be either visible or invisible.*

For a given MDP, a wide class of reductions can be defined using *reduction functions*. Informally, such a function F decides for each state which outgoing actions are enabled in the reduced MDP. This MDP's transition relation then consists of all transitions enabled according to F, and the set of states consists of all states that are still reachable using the reduced transition function.

Definition 4 (Reduction functions). *For an MDP* $\mathcal{A} = (S_{\mathcal{A}}, \Sigma, P_{\mathcal{A}}, s^0, \mathsf{AP}, L_{\mathcal{A}})$*, a* reduction function *is any function* $F \colon S_{\mathcal{A}} \to \mathcal{P}(P_{\mathcal{A}})$ *such that* $F(s) \subseteq en(s)$ *for every* $s \in S_{\mathcal{A}}$*. Given a reduction function* F*, the* reduced MDP *for* $\mathcal{A}$ *with respect to* F *is the minimal MDP* $\mathcal{A}_F = (S_F, \Sigma, P_F, s^0, \mathsf{AP}, L_F)$ *such that*

- *if* $s \in S_F$ *and* $(s, a, \mu) \in F(s)$*, then* $(s, a, \mu) \in P_F$ *and* $\mathsf{support}(\mu) \subseteq S_F$*;*
- $L_F(s) = L_{\mathcal{A}}(s)$ *for every* $s \in S_F$*,*

where minimal should be interpreted as having the smallest set of states and the smallest set of transitions.

Given a reduction function F *and a state* $s \in S_F$*, we say that* s *is a* reduced state *if* $F(s) \neq en(s)$*. All outgoing transitions of a reduced state are called* nontrivial transitions*. We say that a reduction function is* acyclic *if there are no cyclic paths when only nontrivial transitions are considered.*

3 Confluence for Statistical Model Checking

Confluence reduction is based on commutativity of invisible transitions. It works by denoting a subset of the invisible transitions of an MDP as *confluent*. Basically, this means that they do not change the observable behaviour; everything that is possible before a confluent transition is still possible afterwards. Therefore, they can be given *priority*, omitting all their neighbouring transitions.

3.1 Confluent Sets of Transitions

Previous work defined conditions for a set of transitions to be confluent. In the non-probabilistic action-based setting, several variants were introduced, ranging from ultra weak confluence to strong confluence [4]. They are all given diagrammatically, and define in which way two outgoing transitions from the same state have to be able to join again. Basically, for a transition $s \xrightarrow{\tau} t$ to be confluent, every transition $s \xrightarrow{a} u$ has to be mimicked by a transition $t \xrightarrow{a} v$ such that u and v are bisimilar. This is ensured by requiring a confluent transition from u to v.

In the probabilistic action-based setting, a similar approach was taken [23]. For a transition $s \xrightarrow{\tau} \mathbb{1}_t$ to be confluent, every transition $s \xrightarrow{a} \mu$ has to be mimicked by a transition $t \xrightarrow{a} \nu$ such that μ and ν are equivalent; as usual in probabilistic model checking, this means that they should assign the same probability to each *equivalence class* of the state space in the bisimulation quotient. Bisimulation is again ensured using confluent transitions.

In this work we are dealing with a state-based context; only the atomic propositions that are assigned to each state are of interest. Therefore, we base our definition of confluence on the state-based probabilistic notions given in [13]. It is still parameterised in the way that distributions have to be connected by confluent transitions, denoted by $\mu \rightsquigarrow_{\mathcal{T}} \nu$. We instantiate this later, in Definition 6.

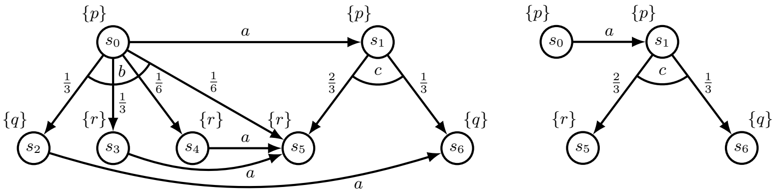
Definition 5 (Probabilistic confluence). *Let $\mathcal{A}$ be an MDP, then a subset $\mathcal{T}$ of transitions from $\mathcal{A}$ is* probabilistically confluent *if it only contains invisible deterministic transitions, and*

$$\forall s \xrightarrow{a} \mathbb{1}_t \in \mathcal{T} \colon \forall s \xrightarrow{b} \mu \colon \; (\mu = \mathbb{1}_t \vee \exists\, t \xrightarrow{c} \nu \colon \mu \rightsquigarrow_{\mathcal{T}} \nu)$$

Additionally, if $s \xrightarrow{b} \mu \in \mathcal{T}$, then so should $t \xrightarrow{c} \nu$ be.

A transition is probabilistically confluent *if there exists a probabilistically confluent set that contains it.*

Compared to [13], the definition is more liberal in two aspects. First, not necessarily $b = c$ anymore. In [13] this was needed to preserve probabilistic visible bisimulation. Equivalent systems according to that notion preserve state-based as well as action-based properties. However, in our setting the actions are only for synchronisation of parallel components, and have no purpose anymore in the final model. Therefore, we can just as well rename them all to a single action. Then, if a transition is mimicked, the action will be the same by construction. Even easier, we chose to omit the required accordance of action names altogether.



(a) Original system

(b) Reduced system

Figure 1. An MDP to demonstrate confluence reduction.

Second, we only require confluent transitions to be invisible and deterministic themselves. In [13], all transitions with the same label had to be so as well (for a more fair comparison with POR). Here, this is not an option, since during simulation we only know part of the state space. However, it is also not needed for correctness, as a local argument about mimicking behaviour until some joining point can clearly never be broken by transitions after this point.

In contrast to POR [2,10], confluence also allows mimicking by differently-labelled transitions, commutativity in triangles instead of diamonds, and local instead of global independence [13]. Additionally, its coinductive definition is well-suited for on-the-fly detection, as we show in this paper. However, as confluence preserves branching time properties, it cannot reduce probabilistic interleavings, a scenario that can be handled by the linear time notion of POR used in [6].

3.2 Equivalence of Probability Distributions

Confluent transitions are used to detect equivalent states. Hence, two distributions are equivalent if they assign the same probabilities to sets of states that are connected by confluent transitions. Given a confluent set $\mathcal{T}$, we denote this by $\mu \rightsquigarrow_{\mathcal{T}} \nu$. For ease of detection, we only consider confluent transitions from the support of μ to the support of ν. In principle, larger equivalence classes could be used when also considering transitions in the other direction and chains of confluent transitions. However, for efficiency reasons we chose not to be so liberal.

Definition 6 (Equivalence up-to $\mathcal{T}$-steps). *Let $\mathcal{A}$ be an MDP, $\mathcal{T}$ a set of deterministic transitions of $\mathcal{A}$ and $\mu, \nu \in \mathsf{Distr}(S)$ two probability distributions. Let R be the smallest equivalence relation containing the set*

$$R' = \{(s,t) \mid s \in \mathsf{support}(\mu), t \in \mathsf{support}(\nu), \exists\, a \colon s \xrightarrow{a} t \in \mathcal{T}\}$$

Then, μ and ν are equivalent up-to $\mathcal{T}$-steps*, denoted by $\mu \rightsquigarrow_{\mathcal{T}} \nu$, if $\mu \equiv_R \nu$.*

Example 1. As an example of Definition 6, consider Figure 1(a). Let $\mathcal{T}$ be the set consisting of all a-labelled transitions. Note that these transitions indeed are all deterministic. We denote by μ the probability distribution associated with the b-transition from s_0, and by ν the one associated with the c-transition from s_1.

We find $R' = \{(s_2, s_6), (s_3, s_5), (s_4, s_5)\}$, and so $R = \mathit{Id} \cup \{(s_2, s_6), (s_6, s_2), (s_3, s_4), (s_4, s_3), (s_3, s_5), (s_5, s_3), (s_4, s_5), (s_5, s_4)\}$ (with Id the identity relation).

Hence, R partitions the state space into $\{s_0\}$, $\{s_1\}$, $\{s_2, s_6\}$, and $\{s_3, s_4, s_5\}$. We find $\mu(\{s_0\}) = \nu(\{s_0\}) = 0$, $\mu(\{s_1\}) = \nu(\{s_1\}) = 0$, $\mu(\{s_2, s_6\}) = \nu(\{s_2, s_6\}) = \frac{1}{3}$ and $\mu(\{s_3, s_4, s_5\}) = \nu(\{s_3, s_4, s_5\}) = \frac{2}{3}$. Therefore, $\mu \equiv_R \nu$ and thus $\mu \rightsquigarrow_{\mathcal{T}} \nu$.

Also note that $\mathcal{T}$ is a valid confluent set according to Definition 5. First, all its transitions are indeed invisible and deterministic. Second, for the a-transitions from s_2, s_3 and s_4, nothing interesting has to be checked. After all, from their source states there are no other outgoing transitions, and every transition satisfies the condition $\mu = \mathbb{1}_t \vee \exists\, t \xrightarrow{c} \nu \colon \mu \rightsquigarrow_{\mathcal{T}} \nu$ for itself due to the clause $\mu = \mathbb{1}_t$. For $s_0 \xrightarrow{a} \mathbb{1}_{s_1}$, we do need to check if the condition holds for $s_0 \xrightarrow{b} \mu$. There is a mimicking transition $s_1 \xrightarrow{c} \nu$, and as we saw above $\mu \rightsquigarrow_{\mathcal{T}} \nu$, as required. □

Our definition of equivalence up-to $\mathcal{T}$-steps is slightly more liberal than the one in [13]. There, the number of states in the support of μ was required to be at least as large as the number of states in the support of ν, since no nondeterministic choice between equally-labelled actions was allowed. Since we do allow this, we take the more liberal approach of just requiring the probability distributions to assign the same probabilities to the same classes of states with respect to confluent connectivity. The correctness arguments are not influenced by this, as the reasoning that confluent transitions connect bisimilar states does not break down if these support sets are potentially more distinct.

3.3 Confluence Reduction

We now define confluence reduction functions. Such a function always chooses to either fully explore a state, or only explore one outgoing confluent transition.

Definition 7 (Confluence reduction). *Given an MDP $\mathcal{A}$, a reduction function F is a* confluence reduction function for $\mathcal{A}$ *if there exists some confluent set $\mathcal{T} \subseteq P$ for which, for every $s \in S$ such that $F(s) \neq en(s)$, it holds that*

- *$F(s) = \{(s, a, \mathbb{1}_t)\}$ for some $a \in \Sigma$ and $t \in S$ such that $(s, a, \mathbb{1}_t) \in \mathcal{T}$.*

In such a case, we also say that F is a confluence reduction function under $\mathcal{T}$.

Confluent transitions might be taken indefinitely, ignoring the presence of other actions. This problem is well known as the *ignoring problem* [11], and is dealt with by the cycle condition of the ample set method of POR. We can just as easily deal with it in the context of confluence reduction by requiring the reduction function to be acyclic. Acyclicity can be checked in the same way as was done for POR in [6]: always check whether in the last l steps at least one state was fully explored (i.e., the state already contained only one outgoing transition).

Example 2. In the system of Figure 1(a), we already saw that the set of all a-labelled transitions is a valid confluent set. Based on this set, we can define the reduction function F given by $F(s_0) = \{(s_0, a, \mathbb{1}_{s_1})\}$ and $F(s) = en(s)$ for every other state s. That way, the reduced system is given by Figure 1(b).

Note that the two models indeed share the same properties, such as that the (minimum and maximum) probability of eventually observing r is $\frac{2}{3}$. □

Confluence reduction preserves $\text{PCTL}^*_{\setminus X}$, and hence basically all interesting quantitative properties (including $\text{LTL}_{\setminus X}$, as was preserved in [6]).

Theorem 1. *Let $\mathcal{A}$ be an MDP, $\mathcal{T}$ a confluent set of its transitions and F an acyclic confluence reduction function under $\mathcal{T}$. Let $\mathcal{A}_F$ be the reduced MDP. Then, $\mathcal{A}$ and $\mathcal{A}_F$ satisfy the same PCTL$^*_{\setminus X}$ formulae.*

4 On-the-fly Detection of Probabilistic Confluence

Non-probabilistic confluence was first detected directly on concrete state spaces to reduce them modulo branching bisimulation [12]. Although the complexity was linear in the size of the state space, the method was not very useful: it required the complete unreduced state space to be available, which could already be too large to generate. Therefore, two directions of improvements were pursued.

The first idea was to detect confluence on higher-level process-algebraic system descriptions [4,5]. Using this information from the symbolic level, the reduced state space could be generated directly without first constructing any part of the original state space. More recently, this technique was generalised to the probabilistic setting [23].

The other direction was to use the ideas from [12] to on-the-fly detect non-probabilistic weak or strong confluence [22,24] during state space generation. These techniques are based on Boolean equation systems and have not yet been generalised to the probabilistic setting.

We present a novel on-the-fly algorithm that works on concrete probabilistic states spaces and does not require the unreduced state space, making it perfectly applicable during simulation for statistical model checking of MDPs.

4.1 Detailed description of the algorithm

Our algorithm is presented on the next page. Given a deterministic transition $s \xrightarrow{a} \mathbb{1}_t$, the function call *checkConfluence*($s \xrightarrow{a} \mathbb{1}_t$) tells us whether or not this transition is confluent. We first discuss this function *checkConfluence*, and then the function *checkEquivalence* on which it relies (which determines whether or not two distributions are equivalent up-to confluent steps).

These functions do not yet fully take into account the fact that confluent transitions have to be mimicked by confluent transitions. Therefore, we have an additional function *checkConfluentMimicking* that is called after termination of *checkConfluence* to see if indeed no violations of this condition occur.

The function *checkConfluence* first checks if a transition is invisible and was not already detected to be confluent before. Then, it is added to the global set of confluent transitions $\mathcal{T}$. To check whether this is valid, a loop checks if indeed all outgoing transitions from s commute with $s \xrightarrow{a} \mathbb{1}_t$. If so, we return *true* and keep the transition in $\mathcal{T}$. Otherwise, all transitions that were added to $\mathcal{T}$ during these checks are removed again and we return *false*. Note that it would not be sufficient to only remove $s \xrightarrow{a} \mathbb{1}_t$ from $\mathcal{T}$, since during the loop some transitions might have been detected to be confluent (and hence added to $\mathcal{T}$) based on the fact that $s \xrightarrow{a} \mathbb{1}_t$ was in $\mathcal{T}$. As $s \xrightarrow{a} \mathbb{1}_t$ turned out not to be confluent, we can also not be sure anymore if these other transitions are indeed actually confluent.

Algorithm 1: Detecting confluence on a concrete state space.

```
global Set⟨Transition⟩ $\mathcal{T} := \varnothing$
global Set⟨Transition, Transition⟩ $M := \varnothing$

bool checkConfluence($s \xrightarrow{a} \mathbb{1}_t$) {
  if $L(s) \neq L(t)$ then
      return false
  else if $s \xrightarrow{a} \mathbb{1}_t \in \mathcal{T}$ then
      return true

  Set⟨Transition⟩ $\mathcal{T}_{\text{old}} := \mathcal{T}$
  Set⟨Transition, Transition⟩ $M_{\text{old}} := M$
  $\mathcal{T} := \mathcal{T} \cup \{s \xrightarrow{a} \mathbb{1}_t\}$
  foreach $s \xrightarrow{b} \mu$ do
      if $\mu = \mathbb{1}_t$ then continue
      foreach $t \xrightarrow{c} \nu$ do
          if checkEquivalence($\mu, \nu$) and
          ($s \xrightarrow{b} \mu \notin \mathcal{T}$ or ($\exists\, u \colon \nu = \mathbb{1}_u$ and checkConfluence($t \xrightarrow{c} \mathbb{1}_u$))) then
              $M := M \cup \{(s \xrightarrow{b} \mu, t \xrightarrow{c} \nu)\}$
              continue outermost loop
          end
      $\mathcal{T} := \mathcal{T}_{\text{old}}$
      $M := M_{\text{old}}$
      return false
  return true
}

bool checkEquivalence($\mu, \nu$) {
  $Q := \{\{p\} \mid p \in \text{support}(\mu) \cup \text{support}(\nu)\}$
  foreach $u \xrightarrow{d} \mathbb{1}_v$ such that $u \in support(\mu),\ v \in support(\nu)$ do
      if checkConfluence($u \xrightarrow{d} \mathbb{1}_v$) then
          $Q := \{q \in Q \mid u \notin q \wedge v \notin q\} \cup \{ \bigcup_{\substack{q \in Q \\ u \in q \vee v \in q}} q\}$
  if $\mu(q) = \nu(q)$ for every $q \in Q$ then
      return true
  else
      return false
  end
}

bool checkConfluentMimicking {
  foreach $(s \xrightarrow{b} \mu, t \xrightarrow{c} \nu) \in M$ do
      if $s \xrightarrow{b} \mu \in \mathcal{T}$ and $t \xrightarrow{c} \nu \notin \mathcal{T}$ then
          if checkConfluence($t \xrightarrow{c} \nu$) then
              return checkConfluentMimicking
          else
              return false
          end
  return true
```

The loop to check whether all outgoing transitions commute with s follows directly from the definition of confluent sets, which requires for every $s \xrightarrow{b} \mu$ that either $\mu = \mathbb{1}_t$, or that there exists a transition $t \xrightarrow{c} \nu$ such that $\mu \rightsquigarrow_{\mathcal{T}} \nu$, where $t \xrightarrow{c} \nu$ has to be in $\mathcal{T}$ if $s \xrightarrow{b} \mu$ is. Indeed, if $\mu = \mathbb{1}_t$ we immediately continue to the next transition (this includes the case that $s \xrightarrow{b} \mu = s \xrightarrow{a} \mathbb{1}_t$). Otherwise, we range over all transitions $t \xrightarrow{c} \nu$ to see if there is one such that $\mu \rightsquigarrow_{\mathcal{T}} \nu$. For this, we use the function *checkEquivalence*(μ, ν), described below. Also, if $s \xrightarrow{b} \mu \in \mathcal{T}$, we have to check if also $t \xrightarrow{c} \nu \in \mathcal{T}$. We do this by checking it for confluence, which immediately returns if it is already in $\mathcal{T}$, and otherwise tries to add it.

If indeed we find a mimicking transition, we continue. If $s \xrightarrow{b} \mu$ cannot be mimicked, confluence of $s \xrightarrow{a} \mathbb{1}_t$ cannot be established. Hence, we reset $\mathcal{T}$ as discussed above, and return *false*. If this did not happen for any of the outgoing transitions of s, then $s \xrightarrow{a} \mathbb{1}_t$ is indeed confluent and we return *true*.

The function *checkEquivalence* checks whether $\mu \rightsquigarrow_{\mathcal{T}} \nu$. Since $\mathcal{T}$ is constructed on-the-fly, during this check some of the transitions from the support of μ might have not been detected to be confluent yet, even though they are. Therefore, instead of checking for connecting transitions that are already in $\mathcal{T}$, we try to add possible connecting transitions to $\mathcal{T}$ using a recursive call.

In accordance to Definition 6, we first determine the smallest equivalence relation that relates states from the support of μ to states from the support of ν in case there is a confluent transition connecting them. We do so by constructing a set of equivalence classes Q, i.e., a partitioning of the state space according to this equivalence relation. We start with the smallest possible equivalence relation, in which each equivalence class is a singleton. Then, for each confluent transition $u \xrightarrow{d} \mathbb{1}_v$, with $u \in \mathsf{support}(\mu)$ and $v \in \mathsf{support}(\nu)$, we merge the equivalence classes containing u and v. Finally, we can easily compute the probability of reaching each equivalence class of Q by either μ or ν. If all of these probabilities coincide, indeed $\mu \equiv_R \nu$ and we return *true*; otherwise, we return *false*.

The function *checkConfluentMimicking* is called after *checkConfluence* designated a transition to be confluent, to verify if $\mathcal{T}$ satisfies the requirement that confluent transitions are mimicked by confluent transitions. After all, when a mimicking transition for some transition $s \xrightarrow{b} \mu$ was found, it might have been the case that $s \xrightarrow{b} \mu$ was not yet in $\mathcal{T}$ while in the end it is. Hence, *checkConfluence* keeps track of the mimicking transitions in a global set M. If a transition $s \xrightarrow{a} \mathbb{1}_t$ is shown to be confluent, all pairs $(s \xrightarrow{b} \mu, t \xrightarrow{c} \nu)$ of other outgoing transitions from s and the transitions that were found to mimic them from t are added to M. If $s \xrightarrow{a} \mathbb{1}_t$ turns out not to be confluent after all, the mimicking transitions that were found in the process are removed again.

Based on M, *checkConfluentMimicking* ranges over all pairs $(s \xrightarrow{b} \mu, t \xrightarrow{c} \nu)$, checking if one violates the requirement. If no such pair is found, we return *true*. Otherwise, the current set $\mathcal{T}$ is not valid yet. However, it could be the case that $t \xrightarrow{c} \nu$ is not in $\mathcal{T}$, while it is confluent (but since $s \xrightarrow{b} \mu$ was not in $\mathcal{T}$ at the moment the pair was added to M, this was not checked earlier). Therefore, we still try to denote $t \xrightarrow{c} \nu$ as confluent. If we fail, we return *false*. Otherwise, we check again for confluent mimicking using the new set $\mathcal{T}$.

4.2 Correctness

The following theorem states that the algorithm is sound. We assume that M and $\mathcal{T}$ are not reset to their initial value $\varnothing$ after termination of *checkConfluence*.

Theorem 2. *Given a transition $p \xrightarrow{l} \mathbb{1}_q$, checkConfluence($p \xrightarrow{l} \mathbb{1}_q$) and checkConfluentMimicking together imply that $p \xrightarrow{l} \mathbb{1}_q$ is confluent.*

Note that the converse of this theorem does not always hold. To see why, consider the situation that *checkConfluentMimicking* fails because a transition $s \xrightarrow{b} \mu$ was mimicked by a transition $t \xrightarrow{c} \nu$ that is not confluent, and $s \xrightarrow{b} \mu$ was added to $\mathcal{T}$ later on. Although we then abort, there might have been another transition $t \xrightarrow{d} \rho$ that could also have been used to mimic $s \xrightarrow{b} \mu$ and that *is* confluent. We chose not to consider this due to the additional overhead of the implementation. Additionally, in none of our case studies this situation occurred.

5 Evaluation

The modes tool[3] provides SMC for models specified in the Modest language [7]. It allowed SMC for MDPs using the POR-based approach of [6]. We have now implemented the confluence-based approach presented in this paper in modes as well. In this section, we apply it to three examples to evaluate its applicability and performance impact. They were selected so as to allow us to clearly identify its strengths and limitations. For each, we (1) give an overview of the model, (2) discuss, if POR fails, why it does and which, if any, modifications were needed to apply the confluence-based approach, and (3) evaluate memory use and runtime.

The performance results are summarised in Table 2. For the runtime assessment, we compare to simulation with uniformly-distributed probabilistic resolution of nondeterminism. Although such a hidden assumption cannot lead to trustworthy results in general (but is implemented in many tools), it is a good *baseline* to judge the *overhead* of confluence checking. We generated 10 000 runs per model instance to compute probabilities p_{smc} for case-specific properties. Using reasoning based on the Chernoff-Hoeffding bound [25], this guarantees the following probabilistic error bound: $\mathrm{Prob}(|p - p_{\mathrm{smc}}| > 0.01) < 0.017$, where p is the actual probability of the property under consideration.

We measure memory usage in terms of the maximum number of extra states kept in memory at any time during confluence (or POR) checking, denoted by s. We also report the maximum number of "lookahead" steps necessary in the confluence/POR checks as k, which is equivalent to $k_{\min} - 1$ in [6], as well as the average length t of a simulation trace and the average number c of nontrivial confluence checks, i.e., of nondeterministic choices encountered, per trace.

To get a sense for the size of the models considered, we also attempt model checking (using mcpta [15], which relies on PRISM [19]). Note that we do not intend to perform a rigorous comparison of SMC and traditional model checking in this paper and instead refer the interested reader to dedicated comparison

[3] modes is part of the Modest Toolset, available at www.modestchecker.net.

Table 2. Confluence simulation runtime overhead and comparison

		uniform:	partial order:			**confluence**:					model checking:	
model	params	time	time	k	s	time	k	s	c	t	states	time
dining crypto-graphers (N)	(3)	1 s	–	–	–	3 s	4	9	4.0	8.0	609	1 s
	(4)	1 s	–	–	–	11 s	6	25	6.0	10.0	3 841	2 s
	(5)	1 s	–	–	–	44 s	8	67	8.0	12.0	23 809	7 s
	(6)	1 s	–	–	–	229 s	10	177	10.0	14.0	144 705	26 s
	(7)	1 s	–	–	–	– timeout –					864 257	80 s
CSMA/CD (RF, BC_{max})	(2, 1)	2 s	–	–	–	4 s	3	46	5.4	16.4	15 283	11 s
	(1, 1)	2 s	–	–	–	4 s	3	46	5.4	16.4	30 256	49 s
	(2, 2)	2 s	–	–	–	10 s	3	150	5.1	16.0	98 533	52 s
	(1, 2)	2 s	–	–	–	10 s	3	150	5.1	16.0	194 818	208 s
BEB (K, N, H)	(4, 3, 3)	1 s	3 s	3	4	1 s	3	7	3.3	11.6	$> 10^3$	> 0 s
	(8, 7, 4)	2 s	7 s	4	8	4 s	4	15	5.6	16.7	$> 10^7$	> 7 s
	(16,15,5)	3 s	18 s	5	16	11 s	5	31	8.3	21.5	– memout –	
	(16,15,6)	3 s	40 s	6	32	34 s	6	63	11.2	26.2	– memout –	

studies such as [27]. Model checking for the BEB example was performed on a machine with 120 GB of RAM [6]; all other measurements used a dual-core Intel Core i5 M450 system with 4 GB of RAM running 64-bit Windows 7.

5.1 Dining Cryptographers

As a first example, we consider the classical dining cryptographers problem [9]: N cryptographers use a protocol that has them toss coins and communicate the outcome with some of their neighbours at a restaurant table in order to find out whether their master or one of them just paid the bill, without revealing the payer's identity in the latter case. We model this problem as the parallel composition of N instances of a `Cryptographer` process that communicate via synchronisation on shared actions, and consider as properties the probabilities of (a) protocol termination and (b) correctness of the result.

The model is a nondeterministic MDP. In particular, the order of the synchronisations between the cryptographer processes is not specified, and could conceivably be relevant. It turns out that all nondeterminism can be discarded as spurious by the confluence-based approach though, allowing the application of SMC to this model. The computed probability p_{smc} is 1.0 for both properties, which coincides with the actual probabilities.

The POR-based approach does not work: Although the nondeterministic ordering of synchronisations between non-neighbouring cryptographers is due to interleaving, the choice of which neighbour to communicate with first for a given cryptographer process is a nondeterministic choice *within* that process.

Concerning performance, we see that runtime increases drastically with the number of cryptographers, N. An increase is expected, since the number of steps until independent paths from nondeterministic choices join again (k) depends directly on N. It is so drastic due to the sheer amount of branching that is present in this model. At the same time, the model is extremely symmetric and can thus be handled easily with a symbolic model checker like PRISM.

5.2 IEEE 802.3 CSMA/CD

As a second example, we take the Modest model of the Ethernet (IEEE 802.3) CSMA/CD approach that was introduced in [15]. It consists of two identical stations attempting to send data at the same time, with collision detection and a randomised backoff procedure that tries to avoid collisions for subsequent retransmissions. We consider the probability that both stations eventually manage to send their data without collision. The model is a probabilistic timed automaton (PTA), but delays are fixed and deterministic, making it equivalent to an MDP (with real variables for clocks, updated on transitions that explicitly represent the delays; modes does this transformation automatically and on-the-fly). The model has two parameters: a time reduction factor RF (i.e., delays of t time units with $RF = 1$ correspond to delays of $\frac{t}{2}$ time units with $RF = 2$), and the maximum value used in the exponential backoff part of the protocol, BC_{max}.

Unfortunately, modes immediately reports nondeterminism that cannot be discarded as spurious. Inspection of the reported lines in the model quickly shows a nondeterministic choice between two probabilistic transitions—which confluence cannot handle. Fortunately, this problem can easily be eliminated through an additional synchronisation, leading to $p_{\text{smc}} = 1.0$ (which is the correct result). POR also fails, for reasons similar to the previous example: initially, both stations send at the same time, the order being determined nondeterministically. In the process representing the shared medium, this must be an *internal* nondeterministic choice. In contrast to the problem for confluence this cannot be fixed.

In terms of runtime, the confluence checks incur a moderate overhead for this example. Compared to the dining cryptographers, the slowdown is much less even where more states need to be explored in each check (s); performance appears to more directly depend on k, which stays low in this case.

5.3 Binary Exponential Backoff

The previous two examples clearly indicate that the added power of confluence reduction pays off, allowing SMC for models where it is not possible with POR. Still, we also need a comparison of the two approaches. For this purpose, we revisit the MDP model of the binary exponential backoff (BEB) procedure that was used to evaluate the POR-based approach in [6]. The probability we compute is that of some host eventually getting access to the shared medium, for different values of the model parameters K (maximum backoff counter value), N (number of tries per station before giving up) and H (number of stations/hosts involved).

Again, for the confluence check to succeed, we first need to minimally modify the model by making a probabilistic transition synchronise. This appears to be a recurring issue, yet the relevant model code could quite clearly be identified as a modelling artifact without semantic impact in both examples where it appears. We then obtain $p_{\text{smc}} = 0.91$ for model instance $(4, 3, 3)$, otherwise $p_{\text{smc}} = 1.0$.

The runtime overhead necessary to get trustworthy results by enabling either confluence or POR is again moderate. This is despite longer paths being explored in the confluence checks compared to the CSMA/CD example (k). The confluence-based approach is somewhat faster than POR in this implementation. As noted in [6], large instances of this model cannot be solved with classical model checking due to the state space explosion problem.

6 Conclusion

We defined a more liberal variant of probabilistic confluence, tailored for the core simulation step of statistical model checking. It has more reduction potential than a previous variant at no extra computational cost, but still preserves $\mathrm{PCTL}^*_{\setminus X}$. We provided an algorithm for on-the-fly detection of confluence during simulation and implemented this algorithm in the modes SMC tool. Compared to the previous approach based on partial order reduction [6], the use of confluence allows new kinds of nondeterministic choices to be handled, in particular lifting the limitation to spurious interleavings. In fact, for two of the three examples we presented, SMC is only possible using the new confluence-based technique, showing the additional power to be relevant. In terms of performance, it is somewhat faster than the POR-based approach, but the impact relative to (unsound) simulation using an arbitrary scheduler largely depends on the amount of lookahead that needs to be performed, for both approaches. Again, on two of our examples, the impact was moderate and should in general be acceptable to obtain trustworthy results. Most importantly, the memory overhead is negligible, and one of the central advantages of SMC over traditional model checking is thus retained.

As confluence preserves branching time properties, it cannot handle the interleaving of probabilistic choices. Although—as we showed—these can often be avoided, for some models POR might work while confluence does not. Hence, neither of the techniques subsumes the other, and it is best to combine them: if one cannot be used to resolve a nondeterministic choice, the SMC algorithm can still try to apply the other. Implementing this combination is trivial and yields a technique that handles the union of what confluence and POR can deal with.

Acknowledgments. We thank Luis María Ferrer Fioriti (Saarland University) for his help in analysing the behaviour of the partial order check on the case studies.

References

1. Baier, C., D'Argenio, P.R., Größer, M.: Partial order reduction for probabilistic branching time. ENTCS 153(2) (2006)
2. Baier, C., Größer, M., Ciesinski, F.: Partial order reduction for probabilistic systems. In: QEST. pp. 230–239. IEEE Computer Society (2004)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
4. Blom, S.C.C.: Partial τ-confluence for efficient state space generation. Tech. Rep. SEN-R0123, CWI (2001)
5. Blom, S.C.C., van de Pol, J.C.: State space reduction by proving confluence. In: CAV. LNCS, vol. 2404, pp. 596–609. Springer (2002)

6. Bogdoll, J., Fioriti, L.M.F., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: FMOODS/FORTE. LNCS, vol. 6722, pp. 59–74. Springer (2011)
7. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for Modestly nondeterministic models. In: MMB/DFT. LNCS, vol. 7201, pp. 249–252. Springer (2012)
8. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: MoDeST: A compositional modeling formalism for hard and softly timed systems. IEEE Transactions on Software Engineering 32(10), 812–830 (2006)
9. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1(1), 65–75 (1988)
10. D'Argenio, P.R., Niebert, P.: Partial order reduction on concurrent probabilistic programs. In: QEST. pp. 240–249. IEEE Computer Society (2004)
11. Evangelista, S., Pajault, C.: Solving the ignoring problem for partial order reduction. Int. Journal on Software Tools for Technology Transfer 12(2), 155–170 (2010)
12. Groote, J.F., van de Pol, J.C.: State space reduction using partial tau-confluence. In: MFCS. LNCS, vol. 1893, pp. 383–393. Springer (2000)
13. Hansen, H., Timmer, M.: A comparison of confluence and ample sets in probabilistic and non-probabilistic branching time. Submitted to TCS. (2013)
14. Hartmanns, A.: Model-checking and simulation for stochastic timed systems. In: FMCO. LNCS, vol. 6957, pp. 372–391. Springer (2010)
15. Hartmanns, A., Hermanns, H.: A Modest approach to checking probabilistic timed automata. In: QEST. pp. 187–196. IEEE Computer Society (2009)
16. Hartmanns, A., Timmer, M.: On-the-fly confluence detection for statistical model checking (extended version). Tech. Rep. TR-CTIT-13-04, CTIT, University of Twente (2013)
17. Henriques, D., Martins, J., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for Markov decision processes. In: QEST. pp. 84–93. IEEE Computer Society (2012)
18. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. LNCS, vol. 2937, pp. 73–84. Springer (2004)
19. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011)
20. Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large Markov decision processes. In: SAC. pp. 1314–1319. ACM (2012)
21. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: RV. LNCS, vol. 6418, pp. 122–135. Springer (2010)
22. Mateescu, R., Wijs, A.: Sequential and distributed on-the-fly computation of weak tau-confluence. Science of Computer Programming 77(10-11), 1075–1094 (2012)
23. M.Timmer, Stoelinga, M.I.A., van de Pol, J.C.: Confluence reduction for probabilistic systems. In: TACAS. LNCS, vol. 6605, pp. 311–325. Springer (2011)
24. Pace, G.J., Lang, F., Mateescu, R.: Calculating-confluence compositionally. In: CAV. LNCS, vol. 2725, pp. 446–459. Springer (2003)
25. PRISM manual: The APMC method, `http://www.prismmodelchecker.org/manual/RunningPRISM/ApproximateModelChecking`
26. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV. LNCS, vol. 2404, pp. 223–235. Springer (2002)
27. Younes, H.L.S., Kwiatkowska, M.Z., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking: An empirical study. In: TACAS. LNCS, vol. 2988, pp. 46–60. Springer (2004)