Partial Order Methods for Statistical Model Checking and Simulation*

Jonathan Bogdoll, Luis María Ferrer Fioriti,
Arnd Hartmanns, and Holger Hermanns

Saarland University – Computer Science, Saarbrücken, Germany

Abstract. Statistical model checking has become a promising technique to circumvent the state space explosion problem in model-based verification. It trades time for memory, via a probabilistic simulation and exploration of the model behaviour—often combined with effective a posteriori hypothesis testing. However, as a simulation-based approach, it can only provide sound verification results if the underlying model is a stochastic process. This drastically limits its applicability in verification, where most models are indeed variations of nondeterministic transition systems. In this paper, we describe a sound extension of statistical model checking to scenarios where nondeterminism is present. We focus on probabilistic automata, and discuss how partial order reduction can be twisted such as to apply statistical model checking to models with spurious nondeterminism. We report on an implementation of this technique and on promising results in the context of verification and dependability analysis of distributed systems.

1 Introduction

Model checking and simulation are complementary techniques in model-based system design. In broad terms, model checkers aim at finding bugs or inconsistencies in a design, and do so by an exhaustive analysis of the system under study. That system is usually modelled as a nondeterministic transition system, which is derived from some program notation. Nondeterminism may be present as a result of concurrent interleaving, or to represent incomplete knowledge about certain implementation decisions at design time, or to reflect the openness of the system to environmental stimuli. When complex designs are modelled by abstract transition systems, nondeterminism is a means to let a small abstract transition system overapproximate the behaviour of the larger concrete design.

Simulation also starts from a model, which is explored in a random manner to get insights into the behaviour of the system. There are many successful simulation environments based on this approach. In artificial intelligence, especially in the planning community, simulation is used on nondeterministic systems with

* This work has been supported by the European Union FP7-ICT project Quasimodo, contract no. 214755, by the DFG as part of SFB/TR 14 AVACS and by the DFG/NWO Bilateral Research Programme ROCKS.

R. Bruni and J. Dingel (Eds.): FMOODS/FORTE 2011, LNCS 6722, pp. 59–74, 2011.

success: Markov chain Monte Carlo techniques just walk through a huge state space in a randomised manner, and find solutions for planning problems, such as games like Go [14], often in the form of a tree or trace through the state space. However, while probabilities are used in the analysis, they are meaningless for the model, and no attempt is made to deduce that a trace or tree has a certain probability to appear.

This is different in discrete-event simulation, where the model is a stochastic process, so the future behaviour at each particular state is determined probabilistically. This makes it possible to perform statistical analysis on many simulation runs to arrive at quantitative insights about the system behaviour [15]. This way of analysing a system model is routinely applied in the systems world, but sometimes the rigidity of the modelling is questionable, and comes with notorious suspicions about hidden assumptions that affect the simulation studies [2,8].

In the probabilistic verification world, discrete-event simulation is more and more applied as a vehicle in statistical model checking [5,24,25]. While conventional probabilistic model checkers solve numerical problems in the size of the entirety of the state space, statistical model checkers perform random sampling to walk through the state space, and perform a statistical analysis of the results to provide an estimate for the property under study. This analysis either uses classical estimates such as confidence intervals, or works with hypothesis testing to validate or refute a probabilistic property. Hypothesis testing can be very fast if the actual probability in the system is very far from the required bound, but it is slow if the two values are close. The basic algorithmic machinery is the same for both simulation approaches however. In this paper, we are discussing this basic machinery, and therefore use the terms statistical model checking and simulation interchangeably.

While some studies [23,13] have made efforts to compare the effectiveness of simulation vs. model checking empirically, such a comparison—which we do not focus on in this paper—is inherently problematic. Superficially, the choice between the two seems to be mainly a tradeoff between memory consumption and time: Model checkers generally need to represent the entirety of a state space in memory (or an encoding thereof, e.g. up to a limited depth), but give highly accurate and precise (or at least safe) answers, while simulation walks through the state space, needing only constant memory, and even applies to infinite-size models. But the accuracy and precision of the simulation results depends on the system parameters and especially (that is, logarithmically) on the number of paths explored. Here, theoretical complexity is practical complexity, and as a result, simulation is most competitive time-wise for low accuracy analysis.

Unfortunately, discrete-event simulation faces the additional problem of requiring the model to be a stochastic process, thus free of nondeterminism. This makes it quite far-fetched to assume that it can be applied inside the ordinary verification trajectory, where nondeterminism is not a bug, but a feature, e.g. resulting from the asynchronous, interleaved execution of components in a distributed system. There is no way to resolve nondeterministic choices without introducing additional assumptions. As it is, simulation can thus far only be

used for deterministic models. While this is an inherent limitation of the approach, this paper aims to show ways of mitigating the problem for a practically relevant class of models. The main contribution is an adaptation of partial order reduction methods [3,10,19,22] that enables statistical model checking of probabilistic automata [20,21] while preserving the low memory demands typical for simulation. We investigate how to apply partial order techniques symbolically and find that ignoring variable valuations renders the method impractical. We therefore adapt the existing techniques to work on-the-fly during simulation, and show the correctness of these modifications.

We report on a selection of case studies that provide empirical evidence concerning the potential of the method and our implementation. The case studies also look at continuous timed probabilistic models from the area of architectural dependability evaluation. In fact, the original motivation for our work stems from the observation that the compositional semantics of the architectural design notation Arcade [7,16] gives rise to nondeterminism which is spurious (i.e., irrelevant) *by construction*. Still, simulative approaches were not applicable to it, because of nondeterminism. The present paper overcomes this deficiency, and as such enables—for the first time—a memory-efficient analysis.

2 Preliminaries

We will use networks of probabilistic automata (PA) with variables as the model on which to explain our techniques in this paper. We will later sketch how to extend our techniques to stochastic timed automata (STA) [6], but the additional features of STA—continuous probability distributions and real time—are largely orthogonal to the issues and techniques we focus on.

2.1 Probabilistic Automata

The model of PA [20,21]—which is akin to Markov decision processes (MDPs)—combines nondeterministic and probabilistic choices. Informally, a PA consists of states from which action-labelled transitions lead to distributions over target states. In every state, there is thus a nondeterministic choice over transitions followed by a probabilistic choice over target states.

Formally, a PA is a 4-tuple $P = (S, s_0, \Sigma, \rightarrow)$ where S is a countable set of states, s_0 is the initial state, Σ is a finite set of actions, $\rightarrow \subseteq S \times \Sigma \times \mathsf{Distr}(S)$ is the transition relation and for a set X, $\mathsf{Distr}(X)$ is the set of probability distributions, i.e. functions $X \rightarrow [0,1]$ such that the sum of the probabilities of all elements of X is 1. $T(s)$ denotes the set of outgoing transitions of a state. We say that P is finite if S and $\rightarrow$ are finite, and P is deterministic if $|T(s)| \leq 1$ for all states s (i.e., the only choices are probabilistic ones).

2.2 Networks of PA

It is often useful to describe complex and, in particular, distributed systems as the parallel composition of several independently specified interacting components.

PA are closed under the interleaving semantics of parallel composition. Using a CSP-style mandatory synchronisation on the actions of the shared alphabet, we define the parallel composition of PA as follows [21]: Given two PA $P_i = (S_i, s_{0_i}, \Sigma_i, \rightarrow_i)$ for $i \in \{1, 2\}$, the parallel composition $P_1 \parallel P_2$ is

$$P_1 \parallel P_2 = (S_1 \times S_2, (s_{0_1}, s_{0_2}), \Sigma_1 \cup \Sigma_2, \rightarrow)$$

where $((s_1, s_2), a, \mu) \in \rightarrow$ if and only if

$$\begin{aligned} & a \in \Sigma_1 \backslash \Sigma_2 \wedge \exists\, (s_1, a, \mu_1) \in \rightarrow_1 \colon \mu = \mu_1 \cdot \{(s_2, 1)\} \\ \text{or } & a \in \Sigma_2 \backslash \Sigma_1 \wedge \exists\, (s_2, a, \mu_2) \in \rightarrow_2 \colon \mu = \mu_2 \cdot \{(s_1, 1)\} \\ \text{or } & a \in \Sigma_1 \cap \Sigma_2 \wedge \exists\, (s_i, a, \mu_i) \in \rightarrow_i \text{ for } i = 1, 2 \colon \mu = \mu_1 \cdot \mu_2. \end{aligned} \tag{1}$$

and $\cdot$ is defined as $(\mu_1 \cdot \mu_2)((s_1, s_2)) = \mu_1(s_1) \cdot \mu_2(s_2)$. A special *silent* action τ is often assumed to be part of all alphabets, but τ-labelled transitions never synchronise. This binary parallel composition operator is associative and commutative; its extension to sets of PA is thus straightforward. We use the term *network of PA* to refer to both a set of PA and its parallel composition.

In Section 4, we will need to identify transitions that appear, in some way, to be the same. For this purpose, we define an equivalence relation $\equiv$ on transitions and denote the equivalence class of t under $\equiv$ by $[t]_{\equiv}$; this notation can naturally be lifted to sets of transitions. In our setting of a network of PA, it is natural to identify those transitions in the product automaton that result from the same (set of) transitions in the component automata.

2.3 PA with Variables

It is common to extend transition systems with variables that take values in some discrete domain D (see e.g. [4, Chapter 2] for the general recipe), and this can be lifted to PA. We call this class of variable decorated automata VPA. In a VPA P_V with a set of variables V, the transition relation is extended with a guard g—a Boolean expression over the variables that determines whether the transition is enabled—and a transition's target becomes a distribution in $\mathsf{Distr}(S \times \mathsf{Assgn}(V))$ where $\mathsf{Assgn}(V) = \mathit{Val}(V) \rightarrow \mathit{Val}(V)$ is the set of assignment functions and $\mathit{Val}(V) = V \rightarrow D$ the set of valuations for the variables. Let $V(A)$ for $A \in \mathsf{Assgn}(V)$ denote the set of variables that are modified by A in some valuation. For a function f and $S \subseteq \mathit{Dom}(f)$, we write $f|_S$ to denote f with domain restricted to S.

A VPA P_V can be transformed into a *concrete* PA P by unrolling the variables' values into the states: Every state $s \in S$ of P_V is replaced by a state $(s, v) \in S \times \mathit{Val}(V)$, the guards are precomputed according to the states' valuations, and assignments to discrete variables redirect edges to states with matching valuations. Note that P is finite iff P_V is finite and the ranges of all discrete variables used are finite. In this case, we say that the VPA is finitary.

Extending parallel composition to VPA is straightforward. Guards of synchronising transitions are the conjunctions of the original transitions' guards,

while the assignments are the union of the original assignments. We allow global variables, which can be used in all component automata. With global variables and synchronising transitions, it is possible to obtain inconsistent assignments (that assign different values to the same variable at the same instant). Such an assignment is no longer a function, and we consider this to be a modelling error.

By using networks of PA with variables, we choose a model that is compositional, but whose semantics is still a PA—so all results for PA (and MDPs) from the literature apply—and which features a distinction between control (via states) and data (via variables). It is the model underlying, among others, PRISM [18], MODEST [6], and prCRL [12].

2.4 Paths, Schedulers and Probabilities

The behaviour of a PA is characterised by its paths or traces. Paths are words from $\rightarrow^{\omega} \cup \rightarrow^{*}$ where the start state of every transition must occur with positive probability in the preceding transition's distribution (or be s_0 for the first one), and finite paths end in deadlock states. In a state-based verification setting, where states are labelled with atomic propositions, a *visible transition* is a transition whose execution changes the set of valid atomic propositions.

In order to be able to reason about probabilities for sets of paths, nondeterminism has to be resolved first by a *scheduler* (or *adversary*). This yields a Markov chain for which a probability measure on paths is induced. Since different schedulers lead to different chains, the answer to questions such as "What is the probability of reaching an error state?" is actually a set of probabilities, which always is a singleton for deterministic models. For nondeterministic models, one is typically interested in maximum and minimum probabilities over all possible resolutions of nondeterminism.

For Markov chains, there exists the notion of terminal strongly connected components. The corresponding notion for PA is that of end components: pairs (S_e, T_e) where $S_e \subseteq S$ and $T_e \colon S \to (\rightarrow)$ with $T_e(s) \subseteq T(s)$, such that for all $s \in S_e$ and transitions $(s, a, \mu) \in T_e(s)$, we have $\mu(s') > 0 \Rightarrow s' \in S_e$ and the underlying directed graph of (S_e, T_e) is strongly connected.

3 Nondeterminism in Models for Simulation

Probabilistic model checkers such as PRISM [18] derive probabilistic quantities by first exploring the complete (reachable) state space of the model and afterwards using numerical techniques such as solvers for systems of linear equations or value iteration to compute minimum and maximum probabilities.

If no nondeterminism is present in the model, it is also possible to perform simulation. In this approach, a large number of concrete, finite paths of the model are explored, using random-number generators to resolve probabilistic choices. Only one state of the model is in memory at any time, and for every path explored, we only need to postprocess which of the properties of interest were satisfied and which were not, and this is evaluated with statistical means.

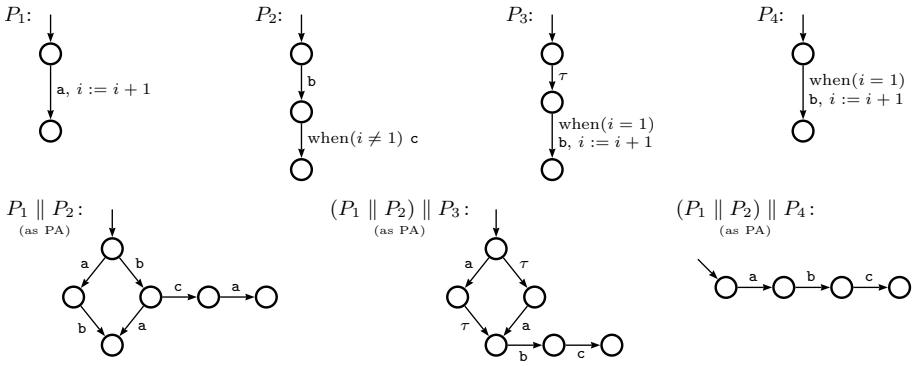

Fig. 1. Nondeterminism and parallel composition

The simulation approach fails in the presence of nondeterminism, because of the absence of a well-defined probability measure. However, the interleaving semantics employed for networks of VPA may result in nondeterminism that is spurious in the sense that the Markov chains induced by arbitrary schedulers all result in the same probability for the property under study to hold. The effects of parallel composition are manifold, as can be seen in the following example.

Example 1. The examples in Figure 1 illustrate possible effects of parallel composition on (non)determinism. i is a global variable that is initially 0. P_1 through P_4 are deterministic VPA, but the parallel composition $P_1 \parallel P_2$ contains a nondeterministic choice. If we ask for the probability of eventually seeing action c, the minimum and maximum values w.r.t. nondeterminism will be different. If we add P_3 to the composition, the result is still nondeterministic, but the minimum and the maximum probabilities are now 1. If we use P_4 instead of P_3 (where P_4 just omits the internal preprocessing step of P_3), the probabilities remain the same, but the final model is fully deterministic. In this case, the parallel composition with P_4 has removed the nondeterminism present in $P_1 \parallel P_2$.

We discuss ways to safely deal with the problem of simulating models with nondeterminism that is introduced by parallel composition in the next section.

4 Partial Order Techniques for Simulation

When simulating nondeterministic models, we need to be sure that the results obtained from a batch of simulation runs, i.e. a set of paths through the PA under study, were not affected by any nondeterministic choices. A sufficient condition for this would be that the model is equivalent—according to some equivalence notion that preserves the properties of interest—to a deterministic one. However, since only a part of the model is explored during the simulation runs, we use the following conditions:

Given a PA $P = (S, s_0, \Sigma, \rightarrow)$ and the set of finite paths $\Pi = \{\pi_1, \ldots, \pi_n\}$, all starting in s_0, encountered during a number of simulation runs, let

$$P|_\Pi = (S_\Pi, s_0, \Sigma, \cup_{i=1}^{n} \pi_i)$$

where $S_\Pi = \{s_0\} \cup \{s \mid \exists\, \pi \in \Pi \colon (s', a, \mu) \in \pi \wedge \mu(s) > 0\}$ denote the sub-PA of P explored by the simulation runs. For brevity, we identify a path $\pi \in \rightarrow^*$ with the set of transitions it contains, i.e. implicitly project to $\pi \subseteq \rightarrow$. Let

$$P_\Pi = (S, s_0, \Sigma, \rightarrow_\Pi)$$

where $t = (s, a, \mu) \in \rightarrow_\Pi$ iff $(s \notin S'_\Pi \wedge t \in \rightarrow) \vee (s \in S'_\Pi \wedge \exists\, \pi \in \Pi \colon t \in \pi)$ and $S'_\Pi = \{s \mid \exists\, \pi \in \Pi \colon (s, a, \mu) \in \pi\}$ denote P restricted to the decisions taken during the simulation runs. Note that P_Π's behaviour is not restricted for states that were not encountered during simulation.

In order to be sure that the result computed from a number of simulation runs was not influenced by nondeterminism, we require that

C1: $P|_\Pi$ is deterministic, and
C2: $P \sim P_\Pi$ for a relation $\sim$ that preserves the properties we consider.

Useful candidates for $\sim$ would be trace equivalence, simulation or bisimulation relations. In the proofs later in this section, $\sim$ will be stutter equivalence, which preserves quantitative (i.e., the probabilities of) $\mathrm{LTL}_{\setminus \mathrm{X}}$ properties [3].

The central practical question we face is how these conditions can be ensured before or during simulation without negating the memory advantages of the simulation approach. In the area of *model checking*, an efficient method already exists to reduce a model containing spurious nondeterminism resulting from the interleaving of parallel processes to smaller models that contain only those paths of interleavings necessary to not affect the end result, namely partial order reduction [3,10,19,22]. In the remainder of this section, we will first recall how partial order reduction for PA works, and then present approaches to harvest partial order reduction to ensure conditions C1 and C2 for *simulation*.

4.1 Partial Order Reduction for PA

The aim of partial order techniques for model checking is to avoid building the full state space corresponding to a model. Instead, a smaller state space is constructed and analysed where the spurious nondeterministic choices resulting from the interleaving of independent actions are resolved deterministically. The reduced system is not necessarily deterministic, but smaller, which increases the performance and reduces the memory demands of model checking (if the reduction procedure is less expensive than analysing the full model right away).

Partial order reduction techniques for PA [3] are extensions of the *ample set method* [19] for nonprobabilistic systems. The essence is to identify an ample set of transitions $\mathsf{ample}(s)$ for every state $s \in S$ of the PA $P = (S, s_0, \Sigma, \rightarrow)$, yielding a reduced PA $\hat{P} = (\hat{S}, s_0, \Sigma, \hat{\rightarrow})$—where $\hat{S}$ is the smallest set that

Table 1. Conditions for the ample sets

A0	For all states $s \in S$, $\mathsf{ample}(s) \subseteq T(s)$.
A1	If $s \in \hat{S}$ and $\mathsf{ample}(s) \neq T(s)$, then no transition in $\mathsf{ample}(s)$ is visible.
A2	For every path $(t_1 = (s, a, \mu), \ldots, t_n, t, t_{n+1}, \ldots)$ in P where $s \in \hat{S}$ and t is dependent on some transition in $\mathsf{ample}(s)$, there exists an index $i \in \{1, \ldots, n\}$ such that $t_i \in [\mathsf{ample}(s)]_{\equiv}$.
A3	In each end component (S_e, T_e) of $\hat{P}$, there exists a state $s \in S_e$ that is fully expanded, i.e. $\mathsf{ample}(s) = T(s)$.
A4	If $\mathsf{ample}(s) \neq T(s)$, then $\lvert\mathsf{ample}(s)\rvert = 1$.

satisfies $s_0 \in \hat{S}$ and $(\exists s \in \hat{S}\colon (s, a, \mu) \in \mathsf{ample}(s) \wedge \mu(s') > 0) \Rightarrow s' \in \hat{S}$, and $\hat{\rightarrow} = \cup_{s \in \hat{S}}(\mathsf{ample}(s))$—such that conditions A0-A4 (Table 1) are satisfied.

For partial order reduction, the notion of *(in)dependent* transitions[1] (rule A2) is crucial. Intuitively, the order in which two independent transitions are executed is irrelevant in the sense that they do not disable each other (forward stability) and that executing them in a different order from a given state still leads to the same states with the same probability distribution (commutativity). Formally, two equivalence classes $[t'_1]_{\equiv} \neq [t'_2]_{\equiv}$ of transitions of P are independent iff for all states $s \in S$ with $t_1, t_2 \in T(s)$, $t_1 = (s, a_1, \mu_1) \in [t'_1]_{\equiv}$, $t_2 = (s, a_2, \mu_2) \in [t'_2]_{\equiv}$,

I1: $\mu_1(s') > 0 \Rightarrow t_2 \in [T(s')]_{\equiv}$ and vice-versa *(forward stability)*, and then also
I2: $\forall s' \in S\colon \sum_{s'' \in S} \mu_1(s'') \cdot \mu_2^{s''}(s') = \sum_{s'' \in S} \mu_2(s'') \cdot \mu_1^{s''}(s')$ *(commutativity)*,

where $\mu_i^{s''}$ is the single element of $\{\mu \mid (s'', a_i, \mu) \in T(s'') \cap [t_i]_{\equiv}\}$. Checking dependence by testing these conditions on all pairs of transitions for all states of the product automaton is impractical. Instead, sufficient and easy-to-check conditions on the VPA level that rely on the fact that the automaton under study results from a network of VPA are typically used, such as the following (where the g_i are the guards and $\mu_i \in \mathsf{Distr}(S \times \mathsf{Assgn}(V))$):

J1: The sets of VPA that t_1 and t_2 originate from (according to the rules of (1), Section 2.2) must be disjoint, and
J2: $\forall v\colon \mu_1(s', A_1) > 0 \wedge \mu_2(s'', A_2) > 0 \Rightarrow (g_2(v) \Rightarrow g_2(A_1(v)) \wedge A_2(v)|_{V(A_2)} = A_2(A_1(v))|_{V(A_2)})$ and vice-versa.

J1 ensures that the only way for the two transitions to influence each other is via global variables, and J2 makes sure that this does actually not happen, i.e., each transition modifies variables only in ways that do not change the other's assignments or disable its guard. This check can be implemented on a syntactical level for the guards and the expressions occurring in assignments.

Using the ample set method with conditions A0-A4 and I1-I2 (or J1-J2) gives the following result:

[1] By abuse of language, we use the word "transition" when we actually mean "equivalence class of transitions under $\equiv$".

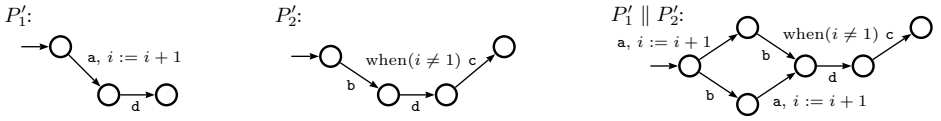

Fig. 2. Spuriousness that can be detected on the VPA level

Theorem 1 ([3]). *If a PA P is reduced to a PA $\hat{P}$ using the ample set method, then $P \sim \hat{P}$ where $\sim$ is stutter equivalence [3].*

For simulation, we are not particularly interested in smaller state spaces, but we can use partial order reduction to distinguish between spurious and actual nondeterminism. A first, naïve approach would be to expand the model given as a network of VPA into a product PA, use partial order reduction to reduce it, and check the result for nondeterminism. However, this neither preserves the low (constant) memory demand typical of simulation nor its general ability to deal with infinite-state systems.

4.2 Deciding Spuriousness on VPA Symbolically

We first investigate the practicality of applying partial order reduction to identify nondeterminism 'symbolically', that is, on the VPA level. By treating data in an abstract manner (as in conditions J1-J2, or using more advanced checks for the independence of operations à la Godefroid [10]), we could hope to avoid the blowup introduced by unrolling the variable valuations. Because synchronisation over shared actions becomes explicit and may remove actual nondeterminism, all nondeterminism that remains may already be spurious.

Example 2. Figure 2 shows an example similar to the ones presented in Figure 1 where this approach works. $P_1' \parallel P_2'$ is shown as a VPA. According to J1-J2, the only (nontrivially) dependent transitions are those labelled `a` and `c`. Choosing any singleton ample set for the initial state of $P_1' \parallel P_2'$ satisfies conditions A0 to A4, so the reduced system is deterministic, thus the nondeterminism is spurious.

The advantage of this approach is that it is a preprocessing step that will not induce a performance penalty in the simulation itself. It can also deal with infinite-data systems, but not infinite control. Yet, whenever an intricate but clever use of variables and guards is responsible for the removal of actual nondeterminism, this method will usually fail.

Unfortunately, such models are not uncommon in practice. They appear e.g. as the PA underlying specifications given in a guarded command language—such as that of PRISM—where the state information is entirely encoded in variables.

Example 3. In the two parallel compositions from Figure 1 that did not exhibit actual nondeterminism, the spuriousness cannot be detected by the symbolic approach because of the dependency between the assignment $i := i + 1$ and the guard $i = 1$. In both cases, the problematic transitions would go away once we look at the actually possible variable valuations.

Table 2. On-the-fly conditions for every state s encountered during simulation

A0	For all states $s \in S$, $\mathsf{ample}(s) \subseteq T(s)$.
A1	If $s \in \hat{S}$ and $\mathsf{ample}(s) \neq T(s)$, then no transition in $\mathsf{ample}(s)$ is visible.
A2'	Every path in P starting in s has a finite prefix $(t_1, \ldots, t_n)$ of length at most k (i.e. $n \leq k$) such that $t_n \in [\mathsf{ample}(s)]_{\equiv}$ and for all $i \in \{1, \ldots, n-1\}$, t_i is independent of all transitions in $\mathsf{ample}(s)$.
A3'	If more than l states have been explored, one of the last l states was fully expanded.
A4'	For all states $s \in \hat{S}$, either $\lvert\mathsf{ample}(s)\rvert = 1$ or $T(s) = \varnothing$.

4.3 Bounded On-the-Fly Partial Order Checking

The examples above show that in order to detect more nondeterminism as spurious, we have to move from the (symbolic) VPA to the (concrete) PA level. However, we do not want to build the concrete model in its entirety before simulation. The goal of our second approach is thus to keep the general advantage of simulation in terms of memory demand, but trade in some simulation performance to obtain a spuriousness check that will work better in practice.

The idea is as follows: We simulate the model as usual on the PA level and without any preprocessing. For every state we encounter, we try to identify a valid singleton ample set. If successful, we can use that transition as the next simulation step. If there is more than one singleton ample set, we deterministically select the first set according to a total order on transitions. If all valid ample sets contain more than one transition, we cannot conclude that the nondeterminism between these is spurious, and simulation is aborted with an error message. To keep memory usage constant, the ample sets are not stored.

The ample set construction relies on conditions A0 through A4, but looking at their formulation in Table 1, conditions A2 and A3 cannot be checked on-the-fly without possibly exploring and storing lots of states—potentially the entire PA. To bound this number of states and ensure termination for infinite-state systems, we instead use the conditions shown in Table 2, which are parametric in k and l. Condition A2 is replaced by A2', which bounds the lookahead inherent in A2 to paths of length at most k. Notably, choosing $k = 1$ is equivalent to not checking for spuriousness at all but aborting on the first nondeterministic choice. Instead of checking for end components as in Condition A3, we use A3' that replaces the notion of an end component with the notion of a set of at least l states.

Conditions A0, A1, A2', A3' and A4' enable us to construct ample sets 'on-the-fly' during simulation. This can be proven correct, i.e. conditions C1 and C2 hold whenever the procedure terminates without raising an error:

Lemma 1. *If simulation with on-the-fly partial order checking applied to a PA P encounters the set of paths Π without aborting, then $P|_\Pi$ is deterministic* (C1).

Lemma 2. *If simulation with on-the-fly partial order checking applied to a PA P encounters the set of paths Π without aborting and all $\pi \in \Pi$ end in a fully expanded state, then $P \sim P_\Pi$ where $\sim$ is stutter equivalence* (C2).

Example 4. For the parallel composition $(P_1 \parallel P_2) \parallel P_3$ in Figure 1, all nondeterminism will correctly be identified as spurious for $k \geq 2$ and $l \geq 2$.

Requiring all runs to end in a fully expanded state is a technical requirement that can always be satisfied if all nondeterminism is spurious, and is satisfied by construction for typical simulation run termination criteria.

For finite systems and large enough k and l, this on-the-fly approach will always be superior to the symbolic approach discussed in Section 4.2 in terms of detecting spuriousness—but if k needs to be increased to detect all spurious nondeterminism as such, the performance in terms of runtime and memory demand will degrade. Note, though, that it is not the actual user-chosen value of k that is relevant for the performance penalty, but what we denote k_{min}, the smallest value of k necessary for condition A2' to succeed in the model at hand—if a larger value is chosen for k, A2' will still only cause paths of length k_{min} to be explored[2]. The size of l actually has no performance impact since A3' can be implemented by just counting the distance to the last fully expanded state.

More precisely, the memory usage of this approach is bounded by $b \cdot k_{min}$ where b is the maximum fan-out of the PA; for a given model (i.e., a fixed b) and small k_{min}, we can consider this as constant, thus the approach still has the same flavour as standard simulation with respect to memory demand. Regarding runtime, exploring parts of the state space that are not part of the path currently being explored (up to $b^{k_{min}}$ states per invocation of A2') induces a performance penalty. The magnitude of this penalty is highly dependent on the structure of the model. In practice, however, we expect small values for k_{min}, which limits the penalty, and this is evidenced in our case studies.

The on-the-fly approach also naturally extends to infinite-state systems, both in terms of control and data. In particular, the kind of behaviour that condition A3 is designed to detect—the case of a certain choice being continuously available, but also continuously discarded—can, in an infinite system, also come in via infinite-state "end components", but since A3' strengthens the notion of end components by the notion of sufficiently large sets of states, this is no problem.

To summarise: For large enough k and l, this approach will allow us to use simulation for any network of VPA where the nondeterminism introduced by the parallel composition is spurious. Nevertheless, if conditions J1 and J2 are used to check for independence instead of I1 and I2, nondeterministic choices *internal to the component VPA*, if present, must be removed by synchronization (via shared variables or actions). While avoiding internal nondeterminism is manageable during the modelling phase, parallel composition and the nondeterminism it creates naturally occur in models of distributed or component-based systems.

[2] Our implementation therefore uses large defaults for k and l so the user usually need not worry about these parameters. If simulation aborts, the cause and its location is reported, including how it was detected, which may be that k or l was exceeded.

5 Implementation

A prototype of the on-the-fly approach presented above has been implemented in version 1.3 of modes, a discrete-event simulator for the MODEST language with a sound treatment of nondeterminism in simulation. MODEST is a high-level compositional modelling formalism for stochastic timed systems [6]. modes can be downloaded at `www.modestchecker.net`.

The full semantic model underlying MODEST is that of stochastic timed automata (STA), which generalise VPA by adding time via clock variables as in timed automata (TA, [1]) as well as infinite and continuous probability distributions, and some of the cases we consider later use this expressiveness. There are, to our knowledge, no results on partial order reduction techniques that deal with real time or continuous probabilistic choices[3]. However, we can treat the additional features of STA in an orthogonal way: The passage of *time* introduces an implicit synchronisation over all component automata by incrementing the values of all clocks. modes thus provides a separate choice of scheduler for time and treats resulting non-zero deterministic delay steps like visible transitions. Nondeterminism can thus be detected as spurious only if the interleaving happens in zero time. In order to correctly detect the spuriousness of nondeterminism in presence of assignments with *continuous probability distributions* (like $x := \mathit{Exponential}(2 \cdot y)$), modes overapproximates by treating them like a nondeterministic assignment to some value from the distribution's support.

6 Case Studies

This section reports on experimental studies with modes on some practically relevant cases. As mentioned in the introduction, we do not aim for a comparison of simulative and numerical model checking approaches in this paper, but instead focus on a comparative study of standard simulation vs. the on-the-fly approach; the existing results (e.g., [23]) then still apply to this new approach modulo its overhead, which we investigate in this section.

Strictly speaking, though, such a comparison is impossible: Standard simulation cannot be applied if the model is nondeterministic. If instead it is deterministic, the on-the-fly approach will have virtually no overhead, because $k_{min} = 1$. To get a meaningful comparison, we use models that are provably equivalent to Markov chains, but contain manifold spurious nondeterminism. We then adjust standard simulation to use uniform distributions to resolve any nondeterministic choices as the zero-overhead baseline for comparison. Due to spuriousness, any resolution would do, and would lead to the same statistical results. For all experiments, modes was run on a 64-bit Core 2 Duo T9300 system; the values for time are for 1000 actual simulation runs, while for memory, the bytes after garbage collections during simulation were measured.

[3] All approaches for TA (see e.g. Minea [17] for one approach and an overview of related work) rely on modified semantics or more restricted models or properties. We are not aware of any approaches for models with continuous distributions.

Table 3. Results for PRISM and modes on the IEEE 802.3 BEB protocol

state space gen.		m.-check	model			uniform sim.		partial order sim.			
states	time	memory	K	N	H	time	memory	time	memory	l	k_{min}
5 371	0 s	1 MB	4	3	3	1 s	1.5 MB	1 s	1.8 MB	16	4
$3.3 \cdot 10^7$	7 s	856 MB	8	7	4	1 s	1.5 MB	2 s	1.4 MB	16	5
$3.8 \cdot 10^{12}$	1052 s	> 100 GB	16	15	5	1 s	1.5 MB	7 s	1.8 MB	16	6
$8.8 \cdot 10^{14}$	4592 s	> 100 GB	16	15	6	1 s	1.6 MB	94 s	3.2 MB	16	7

6.1 Binary Exponential Backoff

We first deal with a model of the IEEE 802.3 Binary Exponential Backoff (BEB) protocol for a set of hosts on a network, adapted from the PRISM model used in [9]. It gives rise to a network of PA, and this makes it possible to use model checking for MODEST [11] in conjunction with PRISM (on a 100 GB RAM system). We report the total number of states, the time it took to generate the state space, and the memory necessary to model check a simple property with PRISM[4] using the default "hybrid" engine in Table 3 (left three columns). The model under study is determined by K, the maximum backoff counter value, i.e. the maximum number of slots to wait, by N, the number of times each host tries to seize the channel, and by H, the number of hosts. The remaining columns give results for uniformly resolved nondeterminism and our on-the-fly approach with parameters l and $k = k_{min}$. The number of component VPA is $H + 1$: the hosts plus a global timer process.

We observe that for larger models, model checking with PRISM is hopeless, while simulation scales smoothly. Simulation runs ended when a host seized the channel. In all cases the nondeterminism is identified as spurious for small values of k and l. The values for uniform resolution show that these models are extremely simple to simulate, while the on-the-fly partial order approach induces a time overhead. We see a clear dependence between the number of components and k_{min}, with $k_{min} = H + 1$. In line with our expectations concerning the performance impact from Section 4.3, we see a moderate increase of memory usage, while runtime is affected more drastically by increasing H.

6.2 Arcade Dependability Models

As a second case study, we focus on ARCADE models and their translation into MODEST [16]. ARCADE is a compositional dependability framework based on a semantics in terms of I/O-IMCs [7]. The ARCADE models under study are known to be weakly bisimilar to Markov chains; yet they exhibit nondeterminism. We studied two simple examples and two very large case studies from the literature:

1bc-1ruded: One basic component with a dedicated repair unit.
2bc-1rufcfs: Two basic components with one first-come-first-serve repair unit.

[4] PRISM needs more memory for model checking than just for state space generation.

Table 4. Results for modes on ARCADE models

model	‖	t	uniform	partial order	l	k_{min}
1bc-1ruded	3	100	1 s / 1.3 MB	1 s / 1.5 MB	16	2
2bc-1rufcfs	4	100	2 s / 1.1 MB	4 s / 1.1 MB	16	3
dda-scaled	41	15	41 s / 1.5 MB	379 s / 2.6 MB	16	6
rcs-scaled	36	15	24 s / 1.6 MB	132 s / 2.0 MB	16	4

dda-scaled: A distributed disk architecture: 24 disks in 6 clusters, 4 controllers, and 2 processors; repairs are performed in a first-come-first-serve manner.

rcs-scaled: The model of a reactor cooling system with a heat exchanger, two pump lines and a bypass system.

The rates in the last two models are scaled to obtain more frequent events, in a variation of importance sampling. Table 4 shows the results; column ‖ indicates the number of concurrently running component automata of the respective model and t is the simulation time bound (model time). We see that the value of k_{min} varies, but again stays relatively small, even for the models consisting of many components. The impact of enabling the on-the-fly approach is again in line with Section 4.3: Memory usage increases moderately, but the time necessary to complete all runs does increase significantly, though not as drastically as in the previous case study. Overall, these are encouraging results. Improving the performance of the lookahead procedure is the first point on our future work agenda for this currently prototypical implementation.

7 Conclusion

This paper has presented an on-the-fly, partial order reduction-based approach that enables statistical model checking and simulation of probabilistic automata with spurious nondeterminism arising from the parallel composition of largely independent, distributed, and intermittently synchronising components. The tool modes has been shown to be able to apply this technique successfully to two very different case studies. In fact, the work on connecting ARCADE to a simulation engine was the nucleus for the work presented here: a method was looked for to certify that simulation results obtained on these models were not affected by any actual nondeterminism.

Acknowledgments. We are grateful to Christel Baier (TU Dresden) for the initial inspiration to combine partial order methods with simulation and to Pedro R. D'Argenio (UNC Cordoba) for fruitful discussions and insights.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS 126(2), 183–235 (1994)
2. Andel, T.R., Yasinsac, A.: On the credibility of MANET simulations. IEEE Computer 39(7), 48–54 (2006)

3. Baier, C., D'Argenio, P.R., Größer, M.: Partial order reduction for probabilistic branching time. Electr. Notes Theor. Comput. Sci. 153(2), 97–116 (2006)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
5. Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. In: Hatcliff, J., Zucca, E. (eds.) FMOODS 2010. LNCS, vol. 6117, pp. 32–46. Springer, Heidelberg (2010)
6. 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)
7. Boudali, H., Crouzen, P., Haverkort, B.R., Kuntz, M., Stoelinga, M.: Architectural dependability evaluation with Arcade. In: DSN, pp. 512–521. IEEE Computer Society Press, Los Alamitos (2008)
8. Cavin, D., Sasson, Y., Schiper, A.: On the accuracy of MANET simulators. In: POMC, pp. 38–43. ACM, New York (2002)
9. Giro, S., D'Argenio, P.R., Ferrer Fioriti, L.M.: Partial order reduction for probabilistic systems: A revision for distributed schedulers. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 338–353. Springer, Heidelberg (2009)
10. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem. LNCS, vol. 1032. Springer, Heidelberg (1996)
11. Hartmanns, A., Hermanns, H.: A Modest approach to checking probabilistic timed automata. In: QEST, pp. 187–196. IEEE Computer Society, Los Alamitos (2009)
12. Katoen, J.P., van de Pol, J., Stoelinga, M., Timmer, M.: A linear process algebraic format for probabilistic systems with data. In: ACSD, pp. 213–222. IEEE Computer Society, Los Alamitos (2010)
13. Katoen, J.P., Zapreev, I.S.: Simulation-based CTMC model checking: An empirical evaluation. In: QEST, pp. 31–40. IEEE Computer Society, Los Alamitos (2009)
14. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
15. Law, A.M., Kelton, D.W.: Simulation Modelling and Analysis. McGraw-Hill Education, Europe (2000)
16. Maaß, S.: Translating Arcade models into MoDeST code. B.Sc. Thesis (May 2010)
17. Minea, M.: Partial order reduction for model checking of timed automata. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 431–446. Springer, Heidelberg (1999)
18. Parker, D.: Implementation of Symbolic Model Checking for Probabilistic Systems. Ph.D. thesis, University of Birmingham (2002)
19. Peled, D.: Combining partial order reductions with on-the-fly model-checking. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 377–390. Springer, Heidelberg (1994)
20. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. Ph.D. thesis. MIT, Cambridge (1995)
21. Stoelinga, M.: Alea jacta est: Verification of Probabilistic, Real-Time and Parametric Systems. Ph.D. thesis. Katholieke U. Nijmegen, The Netherlands (2002)

22. Valmari, A.: A stubborn attack on state explosion. In: Clarke, E., Kurshan, R.P. (eds.) CAV 1990. LNCS, vol. 531, pp. 156–165. Springer, Heidelberg (1991)
23. Younes, H.L.S., Kwiatkowska, M.Z., Norman, G., Parker, D.: Numerical vs. Statistical probabilistic model checking: An empirical study. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 46–60. Springer, Heidelberg (2004)
24. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002)
25. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to simulink/stateflow verification. In: HSCC, pp. 243–252. ACM, New York (2010)