Partial Order Reduction for Probabilistic Systems: A Revision for Distributed Schedulers[⋆]

Sergio Giro, Pedro R. D'Argenio, and Luis María Ferrer Fioriti

FaMAF, Universidad Nacional de Córdoba - CONICET
Ciudad Universitaria - 5000 Córdoba - Argentina
{sgiro,dargenio,ferrer}@famaf.unc.edu.ar

Abstract. The technique of partial order reduction (POR) for probabilistic model checking prunes the state space of the model so that a maximizing scheduler and a minimizing one persist in the reduced system. This technique extends Peled's original restrictions with a new one specially tailored to deal with probabilities. It has been argued that not all schedulers provide appropriate resolutions of nondeterminism and they yield overly safe answers on systems of distributed nature or that partially hide information. In this setting, maximum and minimum probabilities are obtained considering only the subset of so-called *distributed* or *partial information* schedulers. In this article we revise the technique of partial order reduction (POR) for LTL properties applied to probabilistic model checking. Our reduction ensures that *distributed* schedulers are preserved. We focus on two classes of distributed schedulers and show that Peled's restrictions are valid whenever schedulers use only local information. We show experimental results in which the elimination of the extra restriction leads to significant improvements.

1 Introduction

Markov decision processes (MDPs) are widely used in diverse fields ranging from ecology to computer science. They are useful to model and analyse systems in which both probabilistic and nondeterministic choices interact. Particularly, composition oriented versions of MDPs like probabilistic automata [21] or probabilistic modules [12] are aimed to model concurrent and distributed systems.

In the area of system verification, model checking stands out as a model analysis technique for MDPs [22,3]. Moreover, probabilistic model checkers have been developed, notably PRISM [20] and LiQuor [7]. Probabilistic model checking is a push-button technique to calculate maximum and minimum probability values of the satisfaction of a temporal formula in a given model. To obtain these values, the technique requires to universally quantify on all possible resolutions of the inherent nondeterminism of the MDP. The resolution of such nondeterminism is carried out by the so-called *schedulers* (called also adversaries or policies, see e.g. [22,3,21]). Schedulers transform MDPs into Markov chains by selecting one

[⋆] Supported by ANPCyT project PICT 26135 and CONICET project PIP 6391.

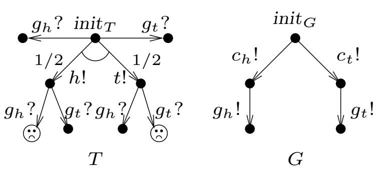

Fig. 1. T tosses a coin, G guesses heads or tails

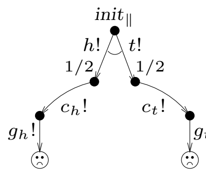

Fig. 2. A dubious scheduling

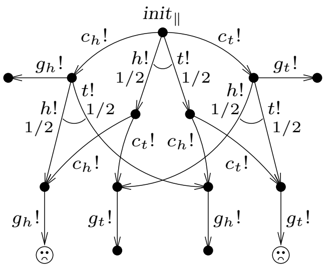

Fig. 3. The product of T and G

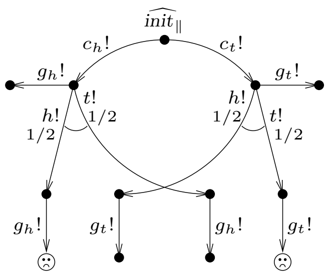

Fig. 4. A POR based reduction

of the enabled transitions at every step in the execution of the system. Therefore, the goal of probabilistic model checking is to find the maximum (or minimum) probability value of a formula over all possible schedulers. Since nondeterminism is an abstraction of deterministic choices in the implementation, concurrent behaviour, or even unknown probabilistic behaviour of the system, such maximum and minimum values are only safe bounds on the actual probability that the formula holds.

In several previous works, it has been argued that quantifying over all possible schedulers yields overly pessimistic bounds in case the components of the system do not share all information [11,12,6,5,13,15,14]. This can be seen in the following example. Consider two persons: one of them tosses a coin and the other one guesses heads or tails. This can be modelled as two processes, respectively depicted as T and G in Fig. 1. (Actions are written in an I/O fashion, with suffixes ? and ! representing input and output respectively.) Process T first tosses a coin that may land heads ($h!$) or tails ($t!$) with probability $\frac{1}{2}$ each. If, after tossing heads, T is notified of a guessed heads ($g_h?$), then T looses the game (represented with ☹); if instead it is notified of a guessed tails ($g_t?$), then T wins the game. The case in which T tossed tails is dual. In addition, if T is notified too early (before tossing the coin), the game is aborted. (Since we deal with input enabled systems, this modelling choice turns to be a convenient simplification.) Process G is simpler: it proceeds to guess by choosing heads ($c_h!$) or tails ($c_t!$) and then it notifies its guess by producing outputs $g_h!$ and $g_t!$, respectively. We take g_h and g_t to be the only two actions shared between T and G. In this way, G does not have any means to know the outcome of the coin toss, nor T to know the guess of G, until they synchronise on g_h or g_t. The composed system is depicted by the product automaton in Fig. 3. Notice, however, that an *almighty* scheduler may let G guess the correct answer with probability 1, as follows. It first lets T toss the coin and then it lets G choose transition $c_h!$ or $c_t!$ depending on whether T tossed heads (i.e. T output $h!$) or it tossed tails (i.e. T output $t!$), respectively. Such scheduler is depicted in Fig. 2. Therefore, by quantifying over all possible schedulers, the maximum probability of G guessing heads or tails (i.e. of reaching ☹) is 1, disregarding the fact that G has no information about the coin. This "total information" assumption underlies all probabilistic model checkers existing today. In conclusion, probabilistic model checkers produce safe upper and lower bounds for the actual probabilities, but they may result too conservative. Needless to say that, in this example, in which T and G do not share all information, we expect the maximum probability of guessing to be $\frac{1}{2}$.

The above observation is fundamental in distributed systems in which components share little information with each other, as well as in security protocols, where the possibility of information hiding is a fundamental assumption (a well-known example of these systems being the *dining cryptographers problem* [4]). The phenomenon we illustrated has been first observed in [21] from the point of view of compositionality, and has been studied on partial information settings in [11,12,5,13,15,14].

In order to limit the variety of behaviours introduced by arbitrary schedulers, classes of schedulers that only consider partial information were proposed in the literature. In particular, we are interested in the class of so-called *distributed schedulers*. Such schedulers were studied in [12] in a synchronous setting and in [5,14] in an asynchronous setting. A distributed scheduler is constructed from *local* schedulers, which are schedulers for single components of the system defined in the usual way, and a mechanism to combine them. Such mechanism could be related to a projection function [12], a passing token [6,5], execution rates of the components [15], or schedulers that define the way in which components interleave [14]. These approaches result in different types of distributed schedulers. We remark that the scheduler of Fig. 2 would not be a valid scheduler in this new setting since the choice for G depends only on information which is external to (and not observed by) G. In fact, a local scheduler for G yielding such a behaviour would not be definable, since the local scheduler depends only on the local history of G, which is certainly the same as long as G does not execute any transition.

As a consequence, a question arises of whether it is possible to do probabilistic model checking under a class of distributed schedulers, i.e., by universally quantifying on such subset of schedulers. Unfortunately, in [13] we showed that the bounds for the probability of properties cannot be calculated nor even approximated if the schedulers are restricted to be distributed. Actually, the framework in [13] is the same synchronous setting of [12]. We later extended these undecidability results to other classes of schedulers [16].

In this paper, we take advantage of the fact that not all schedulers are needed and revise the partial order technique for probabilistic model checking. Partial order reduction (POR) [18,9] is a well-known technique to cope with the state explosion problem. This technique exploits the structure of the product model naturally introduced when interleaving the components. The idea is to eliminate redundant states and transitions but keeping some *representative* ones. Such states and transitions are representative in the sense that, if a given path of the original system is relevant to the property being checked, then an equivalent path should be found in the reduced system. To ensure that representative states and transitions remain in the reduced model, the reduced model must comply with certain conditions [9].

POR for probabilistic model checking was simultaneously introduced in [2] and [10]. These works were oriented to model check LTL properties on MDPs and showed that the reduced model should meet one extra condition apart from those of the non-probabilistic case. This extra condition (call it **A5**) can be stated as follows: for all states in which a probabilistic transition can be reached without executing a transition from the ample set, the set of transitions enabled in the reduced model (called the ample set) is required to either have only one transition or coincide with the original enabled transitions. This technical and non-intuitive condition has been introduced precisely to *not* eliminate the behaviour introduced by schedulers like the one of Fig. 2.

In this paper we study POR techniques for two classes of distributed schedulers and show that condition **A5** can be relaxed for one class and eliminated for the other. Therefore, while the composed system of Fig. 3 is irreducible under condition **A5**, it can be reduced to the one of Fig. 4 in any of our settings.

The way in which we have conceived our technique further exploits the structure of product models and, in particular, the fact that not all information is shared. Moreover, it paves the way for the application of POR to probabilistic *symbolic* model checking. In fact, we are currently busy implementing the technique into PRISM. In this article, we report preliminary promising results of such implementation.

2 Interleaved Probabilistic Input/Output Automata

First, we briefly recall the standard framework of Markov Decision Processes (MDP), just to establish the terminology used in the paper. Later on, we present the framework of Interleaved Probabilistic Input/Output Automata (IPIOA) and show that they are a particular case of MDPs.

A Markov decision process consists of a set of states S, a set of transitions *Trans* and a function $\mathit{enabled} : \mathsf{S} \to \mathcal{P}(\mathit{Trans})$. Each $\alpha \in \mathit{Trans}$ is a function $\alpha : \mathsf{S} \times \mathsf{S} \to [0,1]$ such that $\sum_{s'} \alpha(s, s') = 1$ for all $s \in \mathsf{S}$, $\alpha \in \mathit{enabled}(s)$. A path is a sequence $\sigma = s_1.\alpha_1.\cdots.\alpha_{n-1}.s_n$ such that $\alpha_i \in \mathit{enabled}(s_i)$ and $\alpha_i(s_i, s_{i+1}) > 0$ for all $1 \leq i < n$. Paths can be finite or infinite. Let $\mathit{Paths}^{\mathit{fin}}$ denote the set of all finite paths. Given a finite path σ, the last state of σ is denoted by $\mathit{last}(\sigma)$, and the set of all infinite paths having σ as a prefix is denoted by $\sigma^{\uparrow}$.

The probability of a set of (infinite) paths depends on how the nondeterminism is resolved. So, we define the probability of a set of paths under a given *scheduler*. At every step in the execution of the system, the scheduler chooses one of the enabled transitions. Given a system and a scheduler, the probability of a set of paths is thus completely determined.

In the standard approach, the choice of the next transition is based on the complete history of the system. So, *arbitrary* schedulers are defined as functions mapping finite paths to transitions. A scheduler is then a function $\eta : \mathit{Paths}^{\mathit{fin}} \to \mathit{Trans}$ such that $\eta(\sigma) \in \mathit{enabled}(\mathit{last}(\sigma))$ for all $\sigma \in \mathit{Paths}^{\mathit{fin}}$. The set of all schedulers for an MDP P is denoted by $\mathit{Sched}(P)$.

For all finite path σ, the probability of the set $\sigma^{\uparrow}$ under scheduler η is $\prod_{i=1}^{n-1} \alpha_i(s_i, s_{i+1})$, where $\alpha_i = \eta(s_1.\alpha_1.\cdots.\alpha_{i-1}.s_i)$ for all i, $1 \leq i < n$. The probability of every set of infinite paths considered in this paper is uniquely defined by such probabilities since, in the standard way (namely, by resorting to the Carathéodory extension theorem), the probability of the cylinders $\sigma^{\uparrow}$ can be extended to the least σ-field containing all cylinders.

Next, we present a framework based on the Switched PIOA introduced by Cheung *et al.* [6]. It uses reactive and generative structures (see [17]). Generative transitions model both communication and state change. The component executing a generative transition chooses both a label a to output (e.g. in Fig. 1, h, t) and a new state s according to a given distribution. Reactive transitions specify how a component reacts to a given input. Since the input is not chosen, reactive transitions are simply distributions on states. For a finite set S, we denote by $\mathit{Prob}(S)$ the set of all discrete probability distributions over the set S. Given a set ActLab of action labels and a set S of states, the set of generative transitions $\mathsf{T_G}$ on $(\mathsf{S}, \mathsf{ActLab})$ is $\mathit{Prob}(\mathsf{S} \times \mathsf{ActLab})$, and the set $\mathsf{T_R}$ of reactive transitions is $\mathit{Prob}(\mathsf{S})$. A generative structure on $(\mathsf{S}, \mathsf{ActLab})$ is a function $\mathsf{G} : \mathsf{S} \to \mathcal{P}(\mathsf{T_G})$ and a reactive structure on $(\mathsf{S}, \mathsf{ActLab})$ is a function $\mathsf{R} : \mathsf{S} \times \mathsf{ActLab} \to \mathcal{P}(\mathsf{T_R})$. For $g \in \mathsf{T_G}$, the set of labels a such that $g(s, a) > 0$ for some s is denoted by $\mathsf{ActLab}(g)$. Given a state s, $\mathit{Dirac}(s)$ is the reactive transition r such that $r(s) = 1$.

In our framework, a system is obtained by composing several *probabilistic I/O atoms.*

Definition 1. *A probabilistic I/O atom is a 5-tuple* $(\mathsf{S}, \mathsf{ActLab}, \mathsf{G}, \mathsf{R}, \mathsf{init})$*, where* S *is a finite set of states,* ActLab *is a finite alphabet of actions labels, and* G *(*R*, resp.) is a generative (reactive, resp.) structure in* $(\mathsf{S}, \mathsf{ActLab})$. $\mathsf{init} \in \mathsf{S}$ *is the initial state. We require the atoms to be input-enabled, so* $\mathsf{R}(s, a) \neq \emptyset$ *for every* $s \in \mathsf{S}$*,* $a \in \mathsf{ActLab}$.

We often write S_i to denote the set of states of an atom A_i and similarly for the other elements of the 5-tuple. In addition, we write $\mathsf{T_G}_i$ ($\mathsf{T_R}_i$, resp.) for the set of generative (reactive, resp.) transitions on $(\mathsf{S}_i, \mathsf{ActLab}_i)$.

An interleaved probabilistic I/O system P is a set $\mathit{Atoms}(P)$ of probabilistic I/O atoms $A_1, \cdots, A_N$. The set of states of the system is $\prod_i \mathsf{S}_i$, and the initial state of the system is $\mathsf{init} = (\mathsf{init}_1, \cdots, \mathsf{init}_N)$.

In order to define how the system evolves, we define *compound transitions*, which are the transitions performed by the system as a whole. In such compound transitions, one of the atoms executes a generative transition, and the other atoms execute reactive transitions in case the label generated is in their alphabet. The generative transition g_i involved in this compound transition may output different labels (i.e. the set $\mathsf{ActLab}(g_i)$ may contain more than one action label).

Moreover, the same label may lead to different states, i.e. it is possible that $g_i(s_i, a) > 0$ and $g_i(s'_i, a) > 0$ for two different (local) states s_i and s'_i. Therefore, we have to ensure that each of this equally labelled outputs is matched by the same reactive transition in every atom capable of observing such a label. To this end, we introduce *input choice functions* f_j. An input choice function f_j for g_i maps each label in $\mathsf{ActLab}_j \cap \mathsf{ActLab}(g_i)$ to a reactive transition. The intended meaning is that $f_j(a)$ is executed by atom A_j in case label a is output by g_i. Formally, a compound transition is a tuple $\alpha = (g_i, f_1, \cdots, f_{i-1}, f_{i+1}, \cdots, f_N)$ where g_i is a generative transition in the atom A_i (the *active* atom) and $f_j : \mathsf{ActLab}_j \cap \mathsf{ActLab}(g_i) \to \mathsf{T}_{\mathsf{R}j}$ for all $j \neq i$. The atom A_i that executes the generative transition is denoted by $\mathit{GenAtom}(\alpha)$. The set of all compound transitions is denoted by $\mathit{Trans}(P)$. A compound transition α is enabled in a given state $(s_1, \cdots, s_N)$ if $g_i \in \mathsf{G}_i(s_i)$ and $f_j(a) \in \mathsf{R}_j(s_j, a)$ for all $j \neq i$ and $a \in \mathsf{ActLab}_j \cap \mathsf{ActLab}(g_i)$. We denote by *enabled*$(s)$ the set of all compound transitions enabled in state s. In case $a \notin \mathsf{ActLab}_j$ and a is output by a generative transition, the atom A_j can only remain in its actual state. In order to reflect this fact, given input choice functions f_j for g_i, we define the functions $f^*_{j,s_j} : \bigcup_i \mathsf{ActLab}_i \to \mathsf{T}_{\mathsf{R}i}$ such that $f^*_{j,s_j}(a) = f_j(a)$ if $a \in \mathsf{ActLab}_j \cap \mathsf{ActLab}(g_i)$ and $f^*_{j,s_j}(a) = \mathit{Dirac}(s_j)$, otherwise.

Once a compound transition α is fixed, the probability $\alpha(s, s')$ of reaching a state $s' = (s'_1, \cdots, s'_N)$ from a state $s = (s_1, \cdots, s_N)$ is $\sum_a g_i(s'_i, a) \cdot \prod_{j=1}^{N} f^*_{j,s'_j}(a)(s'_j)$ for all s in which α is enabled. Using the definition of generative and reactive transitions and simple arithmetic, it can be proven that $\sum_{s'} \alpha(s, s') = 1$ for all s and α enabled in s. In consequence, IPIOA are MDPs in which the set of MDP transitions corresponds to IPIOA compound transitions.

In order to ease some definitions, we introduce a fictitious "stutter" compound transition ς. Intuitively, this transition is executed iff the system has reached a state in which no atom is able to generate a transition. The probability $\varsigma(s, s')$ of reaching s' from s using ς is 1, if $s = s'$, or 0, otherwise.

The atoms *involved* in a compound transition α are the active atom and every atom capable of engaging in a communication with labels output by the active atom when α is executed. Let g_i be the generative transition in α, then

$$\mathit{Inv}(\alpha) = \{A_j \mid \exists a \in \mathsf{ActLab}_j, s_i \in \mathsf{S}_i \bullet g(a, s_i) > 0\} .$$

No atom is involved in ς.

In the following, we suppose that input-enabled atoms $A_1, \ldots, A_N$ are given, and we consider the system P comprising all the atoms A_i. We call this system "the compound system". The states (paths, resp.) of the compound system are called global states (global paths, resp.) and the states (paths, resp.) of each atom are called local states (local paths, resp.).

As we have seen, it may be unrealistic to assume that the schedulers are able to see the full history of all the components in the system. In the following, we define restricted classes of schedulers in order to exclude unrealistic behaviours.

For the sake of simplicity, the framework in this paper considers only non-randomized schedulers. However, all of our results also hold for randomized schedulers (see [16]).

Distributed schedulers. Distributed schedulers were introduced in [6] but we use the revised definition of [14]. In a distributed setting as the one we have introduced, different kinds of nondeterministic choices need to be resolved. An atom needs an *output* scheduler to choose the next generative transition. In addition, it may be the case that many reactive transitions are enabled for a single label in the same atom. So, for each atom we need an *input* scheduler in order to choose a reactive transition for each previous history and for each label. Similar types of scheduler have been defined by Cheung *et al.* [6]. Such schedulers are able to make decisions based only on the local history of the atom. Therefore we need to extract the local history out of the global execution for which we define the notion of *projection.*

Given a path σ, the projection $[\sigma]_i$ of the path σ over an atom A_i is defined inductively as follows: **(1)** $[(\mathsf{init}_1, \cdots, \mathsf{init}_N)]_i = \mathsf{init}_i$, **(2)** $[\sigma.\alpha.s]_i = [\sigma]_i$ if $A_i \notin \mathit{Inv}(\alpha)$, **(3)** $[\sigma.(g_j, \cdots).s]_i = [\sigma]_i.g_i.\pi_i(s)$ if $j = i$ (where π_i is the usual projection on tuples) and **(4)** $[\sigma.(g_j, \cdots, f_i, \cdots).s]_i = [\sigma]_i.f_i.\pi_i(s)$ if $i \in \mathit{Inv}(\alpha)$ and $j \neq i$. The set of all the projections over an atom A_i is denoted by $\mathsf{Proj}_i(P)$.

An output scheduler for the atom A_i is a function $\Theta_i : \mathsf{Proj}_i(P) \to \mathsf{T}_{\mathsf{G}i}$ such that, if $\mathsf{G}_i(\mathit{last}([\sigma]_i)) \neq \emptyset$ then $\Theta_i([\sigma]_i) = g \implies g \in \mathsf{G}_i(\mathit{last}([\sigma]_i))$. An input scheduler for an atom A_i is a function $\Upsilon_i : \mathsf{Proj}_i(P) \times \mathsf{ActLab}_i \to \mathsf{T}_{\mathsf{R}i}$ s.t. $\Upsilon_i([\sigma]_i, a) = r \implies r \in \mathsf{R}_i(\mathit{last}([\sigma]_i), a)$. Note that, if the output scheduler Θ_i fixes a generative transition for a given local path $[\sigma]_i$, then the actions in the generative transition can be executed in every global path σ' s.t. $[\sigma']_i = [\sigma]_i$, since we require the atoms to be input-enabled.

An important modification with respect to the framework in [6,5] is the addition of an *interleaving scheduler* that selects the active component to perform the next output. We first consider arbitrary interleaving schedulers that take decisions based on the complete history of the whole system.

An interleaving scheduler is a map that, for a given (global) history, chooses an active atom that will be the next to execute an output transition (according to its output scheduler). Formally, an *interleaving scheduler* is a function $\mathcal{I} : \mathit{Paths}(P) \to \{A_1, \cdots, A_N\}$ such that, if there exists i with $\mathsf{G}_i(\mathit{last}([\sigma]_i)) > 0$ (that is, if there is some atom being able to generate a transition) then $\mathcal{I}(\sigma) = A_i \implies \mathsf{G}_i(\mathit{last}([\sigma]_i)) \neq \emptyset$.

A distributed scheduler for the compound system results by the appropriate composition of the interleaving scheduler and the output and input schedulers of each atom.

Definition 2. *Given an interleaving scheduler $\mathcal{I}$, input schedulers Υ_i and output schedulers Θ_i for each atom i, the* distributed scheduler *η obtained by composing $\mathcal{I}$, Θ_i and Υ_i is defined as $\eta(\sigma) = (g_i, f_1, \cdots, f_{i-1}, f_{i+1}, \cdots, f_N)$ where $\mathcal{I}(\sigma) = A_i$, $g_i = \Theta_i([\sigma]_i)$, and $f_j(a) = \Upsilon_j([\sigma]_j, a)$ for all $j \neq i$ and $a \in \mathsf{ActLab}_j$. In case there is no generative transition enabled, we require $\eta(\sigma) = \varsigma$. The set of distributed schedulers of P is denoted by $\mathit{Dist}(P)$.*

Note that, even when interleaving schedulers are unrestricted, compound schedulers for the compound system are still restricted, since the local schedulers can only see the portion of the history corresponding to the component.

Strongly distributed schedulers. Strongly distributed schedulers were introduced in [14] as a smaller but yet meaningful class of distributed schedulers.

Distributed schedulers provide an accurate model in case the interleaving scheduler has access to all information. As an example, suppose that the atoms represent processes running on the same computer, and the interleaving scheduler plays the role of the operating system scheduler. In case such a scheduler assigns priorities to the processes by gathering information from all processes states, a total information interleaving scheduler is a natural model.

On the contrary, if we are analysing an agreement protocol and each atom models an independent node in a network, then the order in which two nodes execute cannot change according to information that is not available to them. A simple toy example suffices to show how the worst-case probability is affected by the information available to the interleaving scheduler.

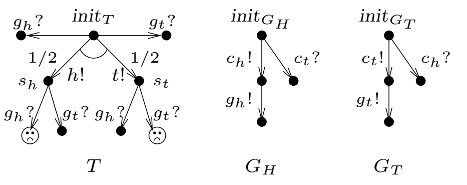

Fig. 5. Motivating strongly distributed schedulers

The atoms in Fig. 5 represent a game in which T plays against G_H and G_T. Atom T loses if it receives a $g_h!$ and the coin has landed heads, and similarly for $g_t!$ and tails. Atoms G_H and G_T take the guess in the following fashion: in case G_H believes that the coin landed heads, it outputs $c_h!$ and then $g_h!$. Conversely, in case G_T believes that the coin landed tails, it outputs $c_t!$ and then $g_t!$. Note that, as soon as one of the players chooses an option, the other one accepts it and stays quiet. Following the intuition given by the example in the introduction, there is no way that G_H and G_T can make an arrangement in such a way that they win all the times. However, the interleaving scheduler that selects T in init, and then G_H for the path $\sigma_{heads} = \mathsf{init}.h!.(s_h, \mathsf{init}_{G_H}, \mathsf{init}_{G_T})$ and G_T for the path $\sigma_{tails} = \mathsf{init}.t!.(s_t, \mathsf{init}_{G_H}, \mathsf{init}_{G_T})$ leads T to ☹ with probability 1.

This scheduler is distributed according to Def. 2, since distributed schedulers restrict the resolution of internal nondeterministic choices, and these atoms have no such choices. In particular, the interleaving scheduler can arrange the execution of G_H and G_T according to the hidden information in T.

The compound model for T, G_H and G_T is very similar to the one in Fig. 3. In fact, since in the graphical representation we omit information concerning the structure of the states, the graphical representation is the same. The unrealistic scheduler in which T losses all the time is also very similar to the unrealistic scheduler in Fig. 2.

The information available to atoms A and B can be defined as $[\sigma]_{A,B} = ([\sigma]_A, [\sigma]_B)$. Note that $[\sigma_{heads}]_{G_H,G_T} = [\sigma_{tails}]_{G_T,G_T} = (\mathsf{init}_{G_H}, \mathsf{init}_{G_T})$. In addition, in the unrealistic scheduler we have $\mathcal{I}(\sigma_{heads}) = G_H$ and $\mathcal{I}(\sigma_{tails}) = G_T$. Generalizing G_H and G_T to be two atoms A, B, and σ_{heads}, σ_{tails} to be two paths σ, σ', we obtain the following condition: for all atoms $A \neq B$ there cannot be two paths σ, σ' such that: **(1)** $[\sigma]_{A,B} = [\sigma']_{A,B}$ and **(2)** atom A is scheduled in σ and **(3)** atom B is scheduled in σ'. Formally:

$$\forall A, B \bullet A \neq B \implies \forall \sigma \bullet \; \nexists \sigma' \bullet [\sigma]_{A,B} = [\sigma']_{A,B} \wedge \mathcal{I}(\sigma) = A \wedge \mathcal{I}(\sigma') = B \; . \quad (1)$$

Definition 3. *A scheduler η is* strongly distributed *iff η is distributed and (1) holds on the interleaving scheduler $\mathcal{I}$ that defines η. The set of strongly distributed schedulers of P is denoted by $SDist(P)$.*

In [14] (where strongly distributed schedulers are introduced for the first time) we prove some properties to further support the fact that (1) is a natural restriction whenever the interleaving nondeterminism is resolved a distributed fashion. In particular, we prove that (1) implies a more general condition in which A and B are replaced with two disjoint sets of atoms $\mathcal{A}$ and $\mathcal{B}$. Strongly distributed schedulers also generalize the *rate schedulers* of [15].

3 Partial Order Reduction under Distributed Schedulers

In this section, we develop two variants of POR for probabilistic systems (each variant corresponding to a class of schedulers) using the *ample sets* construction of [9]. Such variants exploit the distributedness assumptions on schedulers in order to improve the reduction.

Partial order reduction for $\mathsf{LTL}_{\backslash\{\text{next}\}}$. Given a system and a property, the technique of partial order reduction yields another system with less transitions. The reduced system is constructed by traversing the state space. When expanding a given state, not all the transitions enabled are considered. An *ample set* $ample(s)$ must be calculated for each state s, and only transitions in the ample set are considered during the search. POR techniques impose restrictions on the ample sets to ensure that, for each property, the reduced system complies with the property iff the original system does.

We focus on the case where LTL properties do not contain the next operator. Given a set AP of atomic propositions and a labelling function $\mathsf{L} : \mathsf{S} \to \mathcal{P}(\mathsf{AP})$, the set of $\mathsf{LTL}_{\backslash\{\text{next}\}}$ formulae are generated by the following grammar.

$$\phi ::= True \mid l \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathsf{U} \phi_2 \; ,$$

where *True* is a constant and $l \in \mathsf{AP}$. Intuitively, an infinite path ρ satisfies $\phi_1 \mathsf{U} \phi_2$ (denoted by $\rho \models \phi_1 \mathsf{U} \phi_2$) iff there is position in ρ in which ϕ_2 holds, and ϕ_1 holds in all intermediate positions of ρ from the beginning until the position in which ϕ_2 holds. As usual, we write $\mathsf{F}\phi$ for $True\,\mathsf{U}\,\phi$, and $\Pr^{\eta}(\phi)$ for $\Pr^{\eta}(\{\rho \mid \rho \models \phi\})$.

Restrictions to the ample sets are based on the notion of *independence*. We say that two transitions α, β are *independent* iff $(\exists s : \{\alpha, \beta\} \subseteq enabled(s)) \implies Inv(\alpha) \cap Inv(\beta) = \emptyset$. So, two transitions are independent only if the execution of one of them does not interfere with the execution of the other one. Note that the order of execution is irrelevant and that neither of them can disable the other. Notice also that this definition is of a more structural nature than the one in [2]. This is no surprise, since our improvements profit from the structure of the model.

We need some additional definitions before presenting the restrictions for POR. A compound transition α is *stutter* iff $\alpha(s, s') = 0$ for all s such that $\alpha \in \mathit{enabled}(s)$ and s' such that $\mathsf{L}(s) \neq \mathsf{L}(s')$. An *end component* (EC) is a pair (T, A) where $A : T \to \mathcal{P}(\mathit{Trans})$ and T is a set of states such that: **(1)** $\emptyset \neq A(s) \subseteq \mathit{enabled}(s)$ for all $s \in T$, **(2)** $\alpha(s, t) > 0$ implies $t \in T$, for all $s \in T$, $\alpha \in A(s)$ **(3)** for every $s, t \in T$ there exists a path from s to t.

The restrictions for the ample sets of [2] to preserve $\mathsf{LTL}_{\setminus\{\text{next}\}}$ properties under unrestricted full-history dependent schedulers are listed below. $\widehat{\mathsf{S}}$ denotes the set of reachable states in the reduced system $\widehat{P}$, which is constructed by taking $\mathit{ample}(s)$ to be the set of enabled transitions in $s \in \widehat{\mathsf{S}}$.

(**A1**) *For all states* $s \in \mathsf{S}$, $\emptyset \neq \mathit{ample}(s) \subseteq \mathit{enabled}(s)$,
(**A2**) *If* $s \in \widehat{\mathsf{S}}$ *and* $\mathit{ample}(s) \neq \mathit{enabled}(s)$, *then each transition* $\alpha \in \mathit{ample}(s)$ *is stutter,*
(**A3**) *For each path* $\sigma = s.\alpha_1.s_1.\alpha_2.\cdots.\alpha_n.s_n.\gamma\cdots$ *in* P *where* $s \in \widehat{\mathsf{S}}$ *and* γ *is dependent on* $\mathit{ample}(s)$ *there exists an index* $1 \leq i \leq n$ *such that* $\alpha_i \in \mathit{ample}(s)$,
(**A4**) *If* (T, A) *is an EC in* $\widehat{P}$ *and* $\alpha \in \bigcap_{t \in T} \mathit{enabled}(t)$, *then* $\alpha \in \bigcup_{t \in T} \mathit{ample}(t)$
(**A5**) *If* $s.\alpha_1.s_1.\alpha_2.s_2.\cdots.\alpha_n.s_n.\gamma.s_{n+1}$ *is a path in* P *where* $s \in \widehat{\mathsf{S}}$, $\alpha_1, \cdots, \alpha_n$, $\gamma \notin \mathit{ample}(s)$ *and* γ *is probabilistic (i.e.* $0 < \gamma(s', t') < 1$ *for some* s', t'*) then* $|\mathit{ample}(s)| = 1$.

Conditions **A1**–**A3** are original for POR on non-probabilistic systems [9]. **A1** ensures that the reduced model is a submodel of the original one, and that it does not have terminal states (since the original model does not have either). **A3** enforces that any finite sequence of transitions leaving a state s that does not contain a transition in $\mathit{ample}(s)$ can be extended with such transition. Together with **A2**, they ensure that any execution in the original system can be mimicked by an observational equivalent trace in the reduced system. Besides, notice that **A3** is the only condition that is concretely related to the notion of (in)dependence. Condition **A4** is a probabilistic variant of Peled's cycle condition on non-probabilistic models. In such models this condition enforces that, if a transition is enabled indefinitely along a path in the original system, then the transition is enabled in the reduced system in at least one state in such a path. Therefore, condition **A4** ensures that all fair paths are also represented in the reduced system. This variant is somewhat weaker than the original one: since the restriction does not concern cycles, but end components, some cycles are not required to comply with the restriction. Condition **A5** is particular for probabilistic models. Contrarily to the other conditions, **A5** is technical and non-intuitive and has been introduced precisely to *not* eliminate the behaviour introduced by (non-distributed!) schedulers like the one of the example in Fig. 2. We remark that if the model P is non-probabilistic, condition **A5** has no effect and condition **A4** reduces to Peled's original cycle condition. As a consequence, conditions **A1**–**A5** behave exactly in the same way as Peled's original conditions for POR on non-probabilistic models.

In case we assume that the schedulers are distributed, we can replace **A5** by (**A5′**) *If* $s.\alpha_1.s_1.\alpha_2.s_2\cdots\alpha_n.s_n.\gamma.s_{n+1}$ *is a path in* P *where* $s \in \widehat{S}$, $\alpha_1,\cdots,\alpha_n,\gamma \notin$ *ample*(s) *and* γ *is a probabilistic transition, then either ample*$(s) =$ *enabled*(s) *or GenAtom*$(\beta) =$ *GenAtom*(β')*, for all* $\beta, \beta' \in$ *ample*(s).

Condition **A5′** relaxes condition **A5** in the sense that if a probabilistic transition can be reached from state s without executing a transition from the ample set, all transitions enabled in the reduced model (i.e. in *ample*(s)) are generated by the same atom. Contrarily to **A5**, **A5′** does not requires *ample*(s) to be a singleton; *ample*(s) may contain several transitions as long as they are generated by a single atom.

The result is formalized in the following theorem.

Theorem 1. *Let* ϕ *be an* $\mathsf{LTL}_{\backslash\{\mathrm{next}\}}$ *formula and* P *be an IPIOA. Let* $\widehat{P}$ *be a reduction of* P *complying with conditions* **A1**–**A4**, **A5′**. *Then,*

$$\sup\nolimits_{\eta \in \mathit{Dist}(P)} \Pr^{\eta}(\phi) \le \sup\nolimits_{\eta \in \mathit{Sched}(\widehat{P})} \Pr^{\eta}(\phi) .$$

In case we assume *strongly distributed schedulers*, **A5** can be disregarded.

Theorem 2. *Let* ϕ, P *be as in Theorem 1. Let* $\widehat{P}$ *be a reduction of* P *complying with conditions* **A1**–**A4**. *Then,* $\sup_{\eta \in \mathit{SDist}(P)} \Pr^{\eta}(\phi) \le \sup_{\eta \in \mathit{Sched}(\widehat{P})} \Pr^{\eta}(\phi)$.

As an example, recall atoms T and G in Fig. 1 and the non-distributed scheduler η^w in Fig. 2. According to Theorem 1 the reduction in Fig. 4 is correct in case distributed schedulers are assumed. However, in the original system P we have $\Pr^{\eta^w}(\mathsf{F}☹) = 1$, while in $\widehat{P}$ we have $\Pr^{\eta}(\mathsf{F}☹) \le \frac{1}{2}$ for all η. This is due to the fact that η^w is not distributed. In fact, the supremum over all distributed schedulers in P is $\frac{1}{2}$, which coincides with $\sup_{\eta \in \mathit{Sched}(\widehat{P})} \Pr^{\eta}(\mathsf{F}☹)$. Recall now the example in Fig. 5 with atoms T, G_H and G_T. We mentioned that the scheduler of Fig. 2 *is* distributed in this setting. Call this scheduler η^d. If we assume strongly distributed schedulers, the reduction in Fig. 4 is allowed, and there is no scheduler yielding probability 1 in the reduced system. This is correct, since the scheduler η^d is *not* strongly distributed. However, if we want to preserve all distributed schedulers (even those that are *not* strongly distributed) then condition **A5′** prevents the reduction in Fig. 4, since $c_h!$ and $c_t!$ are generated by atoms G_H and G_T, resp. This is exactly what we want, since the scheduler η^d is a valid distributed scheduler for T, G_H and G_T, and so a corresponding scheduler yielding probability 1 must exist in the reduced system.

Correctness of our techniques. The proofs of Theorems 1 and 2 are quite technical and several details are involved. However, these proofs rely on the same principle as in the non-probabilistic case. Our aim is to give an explanation so that the reader can have proper insight on the validity of our techniques. For fully detailed proofs, see [16].

In the non-probabilistic case, the standard argument is as follows. For every property ϕ, we need to prove that ϕ is satisfied in all paths in P if and only if ϕ is satisfied in all paths in $\widehat{P}$. Since $\widehat{P}$ is a subgraph of P, one implication is trivial. For the other implication, the conditions on the reduction are used to prove that, if some path ρ in P does not satisfy ϕ, then $\widehat{\rho} \not\models_{\widehat{P}} \phi$ for some $\widehat{\rho}$.

Similarly, in our case it is sufficient to prove that, for each scheduler η in the original system, there exists a corresponding $\widehat{\eta}$ in the reduced system. The probability values for η and $\widehat{\eta}$ must coincide for all paths relevant to ϕ. We prove that, for each *distributed* (*strongly distributed*, resp.) scheduler, there is a corresponding scheduler in the reduced system that yields the same probability value. As a consequence, it may be the case that, for some non-distributed schedulers, there are no corresponding schedulers in the reduced system. However, this causes no harm since schedulers are assumed to be distributed.

Given a non-probabilistic system P, let $\rho = s_1.\alpha_1.s_2.\alpha_2.\cdots$ and ϕ such that $\rho \not\models_{\widehat{P}} \phi$. We sketch how the corresponding path $\widehat{\rho}$ is constructed in the standard approach. If $\alpha_1 \in \mathit{ample}(s_1)$, then $\widehat{\rho}$ starts with $s_1.\alpha_1$ and the construction continues from $s_2.\alpha_2.\cdots$. On the contrary side, if $\alpha_1 \notin \mathit{ample}(s_1)$, then $\widehat{\rho}$ cannot start with $s_1.\alpha_1$, since α_1 is not enabled for s_1 in $\widehat{P}$. However, condition **A1** ensures that $\mathit{ample}(s_1) \neq \emptyset$. For simplicity, let's consider the case in which some $\beta \in \mathit{ample}(s_1)$ is eventually executed in ρ. W.l.o.g., we can take such a β to be the first transition α_n in ρ such that $\alpha_n \in \mathit{ample}(s_1)$. Then, by condition **A3** and definition of independence, we have that $\rho' = s_1.\alpha_n.s'_2.\alpha_1.\cdots.s'_{n-1}.\alpha_{n-1}.s_n.\alpha_{n+1}.\cdots$ is a path in P. (Here, s'_i denotes the state such that $\alpha_{i-1}(s'_i, s'_{i+1}) = 1$, since the system is non-probabilistic.) Let $\ell_i = \mathsf{L}(s_i)$ for all i. Then, since **A2** requires the transitions in $\mathit{ample}(s)$ to be stuttering, the sequence $\mathsf{L}(\rho)$ of labels in ρ has the form $\ell_1 \cdots \ell_n \ell_n \ell_{n+2} \cdots$. Condition **A2** can be used to prove that $\mathsf{L}(\rho') = \ell_1 \ell_1 \ell_2 \cdots$. So, since $\mathsf{L}(\rho)$ and $\mathsf{L}(\rho')$ differ only in the amount of times that each ℓ_i appears, and $\mathsf{LTL}_{\setminus\{\text{next}\}}$ formulae are stuttering-invariant, $\phi \not\models_P \rho'$. Having found ρ', we let $\widehat{\rho}$ start with $s_1.\alpha_n$ and continue the construction using $s'_2.\alpha_1.\cdots.s'_{n-1}.\alpha_{n-1}.s_n.\alpha_{n+1}.\cdots$. The case in which no transition β is executed in ρ is similar (see [9]).

In summary, the key step of the construction is to "move" β across the α_i's so that it executes immediately after s_1. In the probabilistic case, we must deal with schedulers (which have a tree-like structure) instead of mere paths, and so it is not clear how a transition can be moved. Consider the scheduler η in Fig. 6 (a) and the reduction in Fig. 4. The corresponding scheduler in $\widehat{T \parallel G}$ cannot start with the probabilistic transition $\frac{1}{2}h! + \frac{1}{2}t!$, since it is not enabled in $\widehat{\mathsf{init}_{\parallel}}$. However, the same probabilistic effect is obtained by the scheduler $\widehat{\eta}$ that executes $c_h!$ in the first place, as illustrated in Fig. 6 (b). In this figure, $c_h!$ is moved across both $h!$ and $t!$. In the general case, the transition in the ample set is moved across the transitions in all branches. Note that, in order to move $c_h!$ after init, we rely on the fact that $c_h!$ is executed after both $h!$ and $t!$. In fact, there is no way to transform the non-distributed scheduler in Fig. 2 into a scheduler for the reduced system in Fig. 4: although $c_h!, c_t! \in \mathit{ample}(\mathsf{init}_{\parallel})$, we have that $c_h!$ is chosen in one of the branches, while $c_t!$ is chosen in the other.

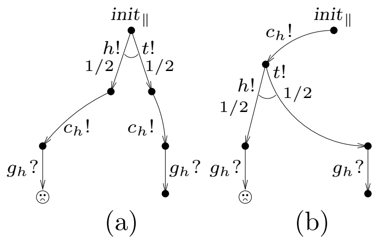

Fig. 6. Transforming a scheduler in the coin example

Groesser *et al.* [2] showed how schedulers for the original system can be mapped to schedulers in the reduced system. They require condition **A5** because the transformation is not possible for some schedulers and some reductions, even if such reductions comply with **A1**–**A4**. However, we proved in [16] that a similar transformation can be carried out for all schedulers η complying with the following condition:

$$\eta(\sigma) \in \mathit{ample}(s_1) \ \wedge\ \eta(\sigma') \in \mathit{ample}(s_1) \implies \eta(\sigma) = \eta(\sigma') \tag{2}$$

for all $\sigma = s_1.\alpha_1.\cdots.\alpha_{n-1}.s_n$, $\sigma' = s_1.\alpha'_1.\cdots.\alpha'_{n'-1}.s'_{n'}$ such that the α_k's and the α'_k's are independent from $\mathit{ample}(s)$. Roughly speaking, the ample transition must be the same in all branches in which an ample transition appears.

We show that (2) holds if: **(1)** η is distributed and **A5′** holds or **(2)** η is strongly distributed. If η is distributed, let I be $\cup_{\beta \in \mathit{ample}(s_1)} \mathit{Inv}(\beta)$ and let σ, σ' be as in (2). Since the α_k's and the α'_k's are independent from all the transitions in $\mathit{ample}(s_1)$, we have $I \cap \mathit{Inv}(\alpha_k) = I \cap \mathit{Inv}(\alpha_{k'}) = \emptyset$ for all k. Then, $[\sigma]_i = [\sigma']_i = s_1$ for all $A_i \in I$. By **A5′**, we have $\mathit{GenAtom}(\eta(\sigma)) = \mathit{GenAtom}(\eta(\sigma'))$. Let $A_i = \mathit{GenAtom}(\eta(\sigma))$ and let Θ_i be the output scheduler that defines η. Then, $\Theta_i([\sigma]_i) = \Theta_i([\sigma']_i)$, and so the generative transition is the same in both $\eta(\sigma)$ and $\eta(\sigma')$. The same argument can be used to show that the input choice functions are the same in both $\eta(\sigma)$ and $\eta(\sigma')$, and so $\eta(\sigma) = \eta(\sigma')$.

In case η is strongly distributed, we define $A_i = \mathit{GenAtom}(\eta(\sigma))$ and $A_{i'} = \mathit{GenAtom}(\eta(\sigma'))$. Then, $[\sigma]_i = [\sigma']_i\ (= s_1)$ and $[\sigma]_{i'} = [\sigma']_{i'}\ (= s_1)$. Let $\mathcal{I}$ be the interleaving scheduler that defines η. By Eqn. (1), we have $A_i = \mathcal{I}(\sigma) = \mathcal{I}(\sigma') = A_{i'}$, and so $\mathit{GenAtom}(\eta(\sigma)) = \mathit{GenAtom}(\eta(\sigma'))$. Following the same reasoning as in the case of distributed schedulers, we conclude that $\eta(\sigma) = \eta(\sigma')$.

The bottom line is that the restrictions imposed to schedulers (together with **A5′**, in case distributed schedulers are assumed) allow to transform every scheduler in P into a scheduler in $\widehat{P}$ without requiring **A5**.

Using our technique with existing model checking algorithms. We emphasize that, although the correctness of the reduction relies on the assumption that the schedulers are distributed (strongly distributed, resp.), the reduced system is analysed assuming *total information* (because of the undecidability result in [13], the verification under partial information cannot be carried out in a fully automated fashion). The result of the verification thus corresponds to a *pessimistic* analysis of the reduced system. As a consequence, the bounds obtained are still safe, but they are not so tight as for distributed (strongly distributed, resp.) schedulers.

As an example, suppose that we are interested in finding the supremum probability that a system P fails under distributed schedulers. Suppose that 0.1 is the highest probability of failure allowed by the specification. Moreover, suppose that, by using the standard model checking algorithm for MDPs (e.g. [3]), we calculate that the supremum probability of a failure quantifying over all schedulers is 0.15. According to this analysis, the system would not meet the specification. However, the schedulers yielding probabilities greater than 0.1

might be "unrealistic" schedulers as the one in Fig. 2. Suppose that we construct $\widehat{P}$ as described above. Then, we can use the algorithm in [3] to calculate $S = \sup_{\eta \in \mathit{Sched}(\widehat{P})} \Pr^{\eta}(\mathsf{F}\ \mathit{Fail})$. If $S = 0.05$, then Theorem 1 above ensures that $\sup_{\eta \in \mathit{Dist}(P)} \Pr^{\eta}(\mathsf{F}\ \mathit{Fail}) \leq 0.05$, and so the system meets the specification. In this sense, the bounds are *safe* with respect to $\mathit{Dist}(P)$. Note that, in this case, the reduction has prevented some schedulers that are not distributed and so the verification on $\widehat{P}$ is more accurate than the verification on P.

4 Concluding Remarks

We have presented a theoretical framework to perform partial order reduction for probabilistic model checking. Our technique is a revision of previous works [2,10]. We showed that, in the context of distributed systems, the bounds for the probability values calculated by the technique on those works may result overly safe. We then showed that the new condition of [2,10] to construct the ample set may be relaxed or even dropped. This simplifies the algorithm and results in smaller reductions.

The POR technique for symbolic representation introduced in [1] constructs ample sets with transitions from several atoms. So, Theorem 2 allows us to apply a similar technique for probabilistic systems. We are currently busy implementing the technique into PRISM using these ideas. Preliminary results are shown in Table 1.

We have selected two notable case studies. Table 1(a) reports results checking anonymity on the dining cryptographers problem [4]. Column "%" indicates in percentage how small is the reduced model with respect of the full system. Thus, for instance, the size of the state space of the reduced model is 23.58% of the size of the state space of the full model for 11 cryptographers (i.e., more than 4 times smaller). Note that, in general, the construction time of the system is significantly more expensive for POR when compared to the construction time of the full system. Nonetheless, the calculation time of the probability values is significantly larger in the full model. Thus, the total processing time on large systems is better under POR (see the 11 cryptographers). We remark that the old POR reduction (including **A5**) achieves the same results in this case study. However, our results using POR for the symbolic representation and explicit vectors for calculation (the so-called "hybrid" approach [20]) significantly improve the explicit approach of LiQuor [8].

Still more interesting is our second case study. It reports on the verification of the Binary Exponential Backoff protocol of the IEEE 802.3. (The model is the same used in [19] adapted to PRISM notation.) We calculated the maximum and minimum probability that a colliding host aborts transmission after multiple collisions. The numbers n, N, and K are respectively the number of colliding hosts, the maximum number of attempts to seize the channel, and the maximum time slots. Table 1(b) shows reductions yielding sizes up to 5% of the full state space. More interesting is that reduction using only **A1**–**A4** is significantly more efficient than the old reduction with **A1**–**A5** (up to 58.88% in case 6/3/4). We

Table 1. Summary of Experimental Results

(a) Dining Cryptographers

	Full			**A1**–**A4** reduct.			
n	size	constr.	total	size	%	constr.	total
7*	287666	0m00.19	0m03.53	115578	40.18	0m13.01	0m16.59
8*	1499657	0m00.30	0m16.18	526329	35.10	0m36.69	0m52.96
9*	7695856	0m00.44	1m24.84	2363896	30.72	1m46.16	2m29.15
10	39005612	0m00.70	4m41.10	10495991	26.91	4m48.19	6m40.37
11	195718768	0m01.11	29m43.34	46159864	23.58	13m12.84	21m02.46

(b) Binary exponential backoff (size comparison)

Model	Full	**A1**–**A5** reduct.		**A1**–**A4** reduct.		
n / N / 2^K	size	size	% full	size	% full	% **A5**
4 / 3 / 4	532326	191987	36.07	126629	23.79	65.96
5 / 3 / 4	13866186	2752750	19.85	1690227	12.19	61.40
6 / 3 / 4	357387872	36974560	10.35	21771724	6.09	58.88
4 / 3 / 8	3020342	913379	30.24	604457	20.01	66.18
5 / 3 / 8	115442928	18569442	16.09	11585347	10.04	62.39
6 / 3 / 8	4318481408	353075296	8.18	212917856	4.93	60.30

(c) Binary exponential backoff (time comparison)

Model	Full		**A1**–**A5** reduct.		**A1**–**A4** reduct.	
n / N / 2^K	constr.	total	constr.	total	constr.	total
4 / 3 / 4	0m01.39	1m04.27	0m18.96	1m22.98	0m18.02	1m13.36
5 / 3 / 4	0m03.49	11m32.99	1m16.82	8m15.60	1m14.53	6m50.12
6 / 3 / 4	0m07.55	5h03m39.81	4m00.95	1h13m11.39	5m15.06	53m35.43
4 / 3 / 8	0m02.05	3m33.62	0m23.85	3m01.88	0m22.78	2m28.50
5 / 3 / 8	0m05.41	1h21m13.54	1m36.41	30m42.82	1m41.18	22m09.23
6 / 3 / 8	0m13.30	—	5m14.95	12h31m57.39	6m44.82	7h45m46.75

Entries marked with * run on a Pentium 4 630, 3.0Ghz with 2Gb memory, while all the others run on an Opteron 8212 (dual core) with 32Gb memory.

obtained similar satisfactory results on time comparison, notably (again) in case 6/3/4. We note that the 6/3/8 full model could not be analysed because the state space was too large to fit in the hybrid engine of PRISM.

Of course, not all examples we ran yielded such impressive results. We have experienced very little reduction in cases in which components depend very much from each other. This is nonetheless reasonable as our technique is precisely devised for distributed system with little sharing. In particular, both case studies have few communication points and significant local processing.

It is in our plans to report soon on the details of the implementation of the tool under development.

References

1. Alur, R., Brayton, R.A., Henzinger, T.A., Qadeer, S., Rajamani, S.K.: Partial-order reduction in symbolic state-space exploration. Formal Methods in System Design 18(2), 97–116 (2001)
2. Baier, C., Größer, M., Ciesinski, F.: Partial order reduction for probabilistic systems. In: QEST 2004, Washington, DC, USA, pp. 230–239. IEEE CS, Los Alamitos (2004)

3. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 288–299. Springer, Heidelberg (1995)
4. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. J. Cryptology 1(1), 65–75 (1988)
5. Cheung, L.: Reconciling Nondeterministic and Probabilistic Choices. PhD thesis, Radboud Universiteit Nijmegen (2006)
6. Cheung, L., Lynch, N., Segala, R., Vaandrager, F.: Switched PIOA: Parallel composition via distributed scheduling. Theor. Comput. Sci. 365(1-2), 83–108 (2006)
7. Ciesinski, F., Baier, C.: LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In: QEST 2006, pp. 131–132. IEEE CS, Los Alamitos (2006)
8. Ciesinski, F., Baier, C., Größer, M., Klein, J.: Reduction techniques for model checking markov decision processes. In: QEST 2008, pp. 45–54 (2008)
9. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (2000)
10. D'Argenio, P., Niebert, P.: Partial order reduction on concurrent probabilistic programs. In: QEST 2004, Washington, DC, USA, pp. 240–249. IEEE CS, Los Alamitos (2004)
11. de Alfaro, L.: The verification of probabilistic systems under memoryless partial-information policies is hard. In: PROBMIV 1999. TR CSR-99-8, University of Birmingham, pp. 19–32 (1999)
12. de Alfaro, L., Henzinger, T.A., Jhala, R.: Compositional methods for probabilistic systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 351–365. Springer, Heidelberg (2001)
13. Giro, S., D'Argenio, P.: Quantitative model checking revisited: neither decidable nor approximable. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 179–194. Springer, Heidelberg (2007)
14. Giro, S., D'Argenio, P.: On the expressive power of schedulers in distributed probabilistic systems. In: Proc. of QAPL 2009, York, UK, March 28-29 (2009), Extended version, `cs.famaf.unc.edu.ar/~sgiro/QAPL09-ext.pdf`
15. Giro, S., D'Argenio, P.: On the verification of probabilistic I/O automata with unspecified rates. In: Proc. of 24th SAC, pp. 582–586. ACM Press, New York (2009)
16. Giro, S., D'Argenio, P.: Partial order reduction for probabilistic systems assuming distributed schedulers. Technical Report Serie A, Inf. 2009/02, FaMAF, UNC (2009), `http://cs.famaf.unc.edu.ar/~sgiro/TR-A-INF-09-2.pdf`
17. Glabbeek, R.v., Smolka, S., Steffen, B.: Reactive, generative, and stratified models of probabilistic processes. Information and Computation 121, 59–80 (1995)
18. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems. LNCS, vol. 1032. Springer, Heidelberg (1996)
19. Jeannet, B., D'Argenio, P., Larsen, K.: Rapture: A tool for verifying Markov Decision Processes. In: Cerna, I. (ed.) Tools Day 2002, Brno, Czech Republic, Technical Report, Faculty of Informatics, Masaryk University Brno (2002)
20. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. International Journal on Software Tools for Technology Transfer (STTT) 6(2), 128–142 (2004)
21. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, Laboratory for Computer Science, MIT (1995)
22. Vardi, M.: Automatic verification of probabilistic concurrent finite state programs. In: Procs. of 26th FOCS, pp. 327–338. IEEE Press, Los Alamitos (1985)