Model Checking Concurrent Programs with Nondeterminism and Randomization

Rohit Chadha[1], A. Prasad Sistla[2], and Mahesh Viswanathan[3]

1 INRIA & LSV, ENS Cachan and CNRS, FRANCE
2 University of Illinois, Chicago, USA
3 University of Illinois, Urbana-Champaign, USA

Abstract

For concurrent probabilistic programs having process-level nondeterminism, it is often necessary to restrict the class of schedulers that resolve nondeterminism to obtain sound and precise model checking algorithms. In this paper, we introduce two classes of schedulers called *view consistent* and *locally Markovian* schedulers and consider the model checking problem of concurrent, probabilistic programs under these alternate semantics. Specifically, given a Büchi automaton Spec, a threshold $x \in [0, 1]$, and a concurrent program $\mathbb{P}$, the model checking problem asks if the measure of computations of $\mathbb{P}$ that satisfy Spec is at least x, under all view consistent (or locally Markovian) schedulers. We give precise complexity results for the model checking problem (for different classes of Büchi automata specifications) and contrast it with the complexity under the standard semantics that considers all schedulers.

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

The use of randomization in concurrent or distributed systems is often key to achieving certain objectives — it is used in distributed algorithms to break symmetry [22] and in cryptographic protocols to achieve semantic security [19]. The formal analysis of such systems has often modelled them as *Markov Decision Processes* [24], that has both nondeterministic and probabilistic transitions.

In Markov Decision Processes (MDPs), the probability of events depends on the way the nondeterministic choices are resolved during a computation. It is customary to resolve the nondeterminism by a *scheduler* or *adversary*, who chooses a probabilistic transition from a state based on the past sequence of states visited during the computation. When verifying MDPs, one considers the worst possible scenario — one checks that no matter which scheduler is chosen, the probabilistic properties of the system hold. Model checking algorithms based on such semantics for MDPs [5, 24] are known, and tools based on these algorithms have been developed that have been used to analyze many examples [1].

Recently, many researchers have observed [13, 12, 6, 15, 11] that in a number of applications, taking such a pessimistic view and considering all possible schedulers, can yield incorrect verification results. The problem arises when one considers a concurrent system where individual processes exhibit both probabilistic and nondeterministic behavior. For such systems, there are certain *perfect information* schedulers that will resolve local process-level nondeterminism based on information that would not be available to the local process, and there by exhibit behavior that is unreasonable. For example, consider the example presented in [18] of two processes "Toss" and "Guess" that do not communicate with each other. The process Toss tosses a fair coin, and Guess guesses (nondeterministically) what the outcome of Toss's coin toss was. Clearly, since Toss and Guess do not communicate, the probability that Guess makes the right guess should be bounded by $\frac{1}{2}$. However under a scheduler that

© Rohit Chadha and A. Prasad Sistla and Mahesh Viswanathan;
licensed under Creative Commons License NC-ND
Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resolves Guess's nondeterminism based on the result of Toss's coin toss, the probability of a correct guess can be as high as 1! (Additional examples can be found in [9].) Therefore, in analyzing concurrent programs, in many cases, it is necessary to restrict attention to certain "reasonable" schedulers that resolve local nondeterminism based only on information that is locally visible to the process.

We call such schedulers to be *view consistent*. More precisely, a view consistent scheduler is the composition of two schedulers — a *global* scheduler that picks the process to schedule, and a *local* scheduler that chooses a probabilistic transition of the process. We assume that the global scheduler can choose the process based on the entire computation thus far. However, the local scheduler's decision must only be based on the local view of the computation. In other words, if σ and τ are computations such that the states as observable to process $\mathcal{P}$ are identical at every step, then the local scheduler for $\mathcal{P}$ must pick the same transition after both σ and τ. Observe that if the individual processes are purely probabilistic, then every scheduler is view consistent; the difference arises only when there is local nondeterminism. A similar class of schedulers called *distributed schedulers* has been considered in [18, 16, 17]. However, there is a subtle difference between distributed schedulers and view consistent schedulers (see Related Work) and the results presented here do not follow from those in [18, 16, 17]. In this paper, we also consider another class of restricted schedulers that we call *locally Markovian*. Locally Markovian schedulers are view consistent schedulers with the additional restriction that the local scheduler's decision only depends on the length of the computation and the current local state, and not on the entire local view of the computation; note, that in a locally Markovian scheduler, the global scheduler can still choose the process to execute based on the entire history. Locally Markovian schedulers are the natural analog in the concurrent case of Markovian schedulers that have been considered in other contexts [23, 4].

In this paper, we investigate the complexity of the verification problem for concurrent programs. We assume that the correctness specification is given by a Büchi automaton Spec, whose input alphabet consists of the states of the program $\mathbb{P}$, and a threshold $x \in [0, 1]$. We say $\mathbb{P} \models^{vc}_{\rhd x} \mathsf{Spec}$ ($\mathbb{P} \models^{lm}_{\rhd x} \mathsf{Spec}$), where $\rhd \in \{>, \geq\}$, if under all view consistent schedulers (locally Markovian schedulers) the measure of computations of $\mathbb{P}$ accepted by Spec is $\rhd x$. Our results are summarized in Figure 1.

We show that the verification problem is in general undecidable, when we restrict to either view consistent or locally Markovian schedulers. When the threshold is 0, both the problems of checking $\mathbb{P} \models^{vc}_{>0} \mathsf{Spec}$ and $\mathbb{P} \models^{lm}_{>0} \mathsf{Spec}$, remain undecidable even when Spec is a deterministic Büchi automaton. For the case when $x \in (0, 1)$, the problems of checking $\mathbb{P} \models^{vc}_{>x} \mathsf{Spec}$, $\mathbb{P} \models^{vc}_{\geq x} \mathsf{Spec}$, $\mathbb{P} \models^{lm}_{>x} \mathsf{Spec}$, and $\mathbb{P} \models^{lm}_{\geq x} \mathsf{Spec}$, remain undecidable even when Spec is a *safety* automaton.[1]

We then investigate the complexity of the verification problems left open by the above undecidability results. Namely, we consider the problems of checking Spec that are deterministic or safety automata, when $x = 1$, and of checking safety properties when $x = 0$. We show that many of these problems are indeed decidable, and we characterize their computational complexity precisely. Specifically, we show that the problems of checking $\mathbb{P} \models^{vc}_{=1} \mathsf{Spec}$ and $\mathbb{P} \models^{lm}_{=1} \mathsf{Spec}$ are **PSPACE**-complete, where Spec is a safety automaton; checking $\mathbb{P} \models^{lm}_{=1} \mathsf{Spec}$, when Spec is deterministic, is **EXPSPACE**-complete; and checking $\mathbb{P} \models^{lm}_{>0} \mathsf{Spec}$, when Spec is a safety automaton, is also **EXPSPACE**-complete. The decidability/complexity of check-

[1] A safety automaton is a deterministic Büchi automaton such that all states are accepting except for a unique rejecting state; all transitions from the rejecting state stay in the rejecting state. Every regular safety property can be recognized by such an automaton, and hence the name.

ing $\mathbb{P} \models^{vc}_{=1}$ Spec when Spec is deterministic, and checking $\mathbb{P} \models^{vc}_{>0}$ Spec when Spec is a safety property, remain open. However, we show that these model checking problems for view consistent schedulers are **2-EXPTIME**-complete, for two special classes of programs. The first class is that of programs $\mathbb{P}$ where all processes, except possibly one, are purely probabilistic (i.e., have no local nondeterminism). The second class of programs are those where each process has a set of global variables, and some local variables that are private to the process. In addition, we restrict the program to be *mutually exclusive*, that requires that in each global state exactly one process is enabled and we require the specification to be on the *shared state*, that requires that the state of the specification depends only on the history of global states visited.

We contrast the above complexity results with the complexity of the same verification question when we consider *all* schedulers (not just view consistent or locally Markovian schedulers). As previously observed [13], the complexity of verification with respect to perfect information schedulers, is easier. We show that for safety specifications Spec and $x = 1$, the verification problem is **PSPACE**-complete, just like in the case of view consistent and locally Markovian schedulers. All the other verification problems, on the other hand, are **EXPTIME**-complete. Note that, in contrast to the complexity results reported in [5] for MDPs, the blowup in complexity when considering concurrent programs can be explained by the state space explosion problem.

We conclude this introduction by comparing our model of concurrent programs under view consistent schedulers to other probabilistic models for which model checking results are known. Probabilistic automata on infinite strings [3], are a special case of programs under locally Markovian schedulers (see Theorem 3.1 and Lemma 3.6). Partially Observable MDPs [13] are a special case of programs where *all, except possibly one*, processes are purely probabilistic (see discussion in Related Work). Finally, MDPs are equivalent to programs where *all* processes are purely probabilistic. Thus, many of the commonly studied models are special kinds of concurrent programs, and we exploit these connections to prove some upper bounds using translations and embeddings in to these models. Our proofs of lower bounds on the complexities are quite nontrivial and do not follow from any relationships to the above models since our programs are given in a different notation.

The paper is organized as follows. Section 2 contains preliminaries, and definitions of programs and schedulers. Section 3 contains the technical results and the conclusions are presented in Section 4. Motivating examples and proofs of most of the theorems are given in [9].

Related Work

Restricting the class of schedulers has been observed to be important in obtaining compositional reasoning principles [14], and in correctly analyzing security protocols and distributed algorithms [13, 12, 6, 15, 11]. The schedulers considered in these papers are very similar to the class of view consistent schedulers that we consider. In [14], the processes are assumed to run synchronously, and thus the scheduler is the composition of local schedulers that resolve nondeterminism based on local views; there is no global scheduler. In [12, 6], the nondeterministic choices are broken into tasks. A task scheduler chooses the task, and this choice is assumed to be *oblivious* of the actual computation, and local scheduler picks the actual transition within the task by looking at the local state. The task scheduler can be seen as our global scheduler; however, the difference is that our global schedulers are not oblivious. Finally, in [11], the authors don't restrict attention to a specific class of scheduler but rather develop a process calculus within which the schedulers can be specified. All these papers, are

primarily interested in defining clean compositional semantics, and do not consider the model checking problem per se. A closely related class of schedulers called *distributed schedulers* is considered in [18, 16, 17], where the problem of model checking safety properties against any threshold is shown to be undecidable. However, distributed schedulers are different than view consistent schedulers that we consider here — in a distributed scheduler, a local scheduler of process i is completely oblivious of steps in which process i did not get scheduled, whereas in view consistent schedulers, it is aware that some other process was scheduled. This difference is manifested in the fact that $\mathbb{P} \models_{>0} \mathsf{Spec}$ for safety specifications is undecidable for distributed schedulers [16, 17], whereas it is open for view consistent schedulers. Furthermore, no decidability results are presented in [18, 16, 17].

As indicated in the introduction, the model checking problem and its complexity for the related model of Partially Observable MDPs (POMDP) has been investigated in earlier works [13, 20, 2, 10]. A POMDP $\mathbb{P}$ can be seen as a special case of a concurrent program with two processes under view consistent semantics as follows. The POMDP itself is process $\mathcal{P}_1$, and the second process (say $\mathcal{P}_2$) plays the role of "scheduling" the next transition of $\mathcal{P}_1$. They share 3 variables: state that stores the partial state of $\mathcal{P}_1$ that is visible outside, trans that is used by $\mathcal{P}_2$ to inform $\mathcal{P}_1$ what the next transition should be, and turn which is used by the processes to alternate taking turns. In each "round", $\mathcal{P}_2$ first picks $\mathcal{P}_1$'s next transition, and then $\mathcal{P}_1$ "executes" that transition; observe, that $\mathcal{P}_1$ is a purely probabilistic process, and all the nondeterminism has been deferred to $\mathcal{P}_2$. Under view consistent schedulers, the two processes $\mathcal{P}_1$ and $\mathcal{P}_2$ are "equivalent" to the POMDP. Also decision problems for any programs whose all, except possibly one processes are purely probabilistic, can in turn be shown to be "equivalent" to a POMDP of size exponential in the length of the program (see Lemma 3.9). This relationship is exploited by us to prove some upper bounds.

Similarly, the "equivalence" between decision problems on probabilistic automata on infinite strings and decision problems on programs under "locally Markovian" schedulers is exploited to obtain the undecidability results using the results of [2, 7]. This equivalence is also exploited to obtain upper bounds for checking $\mathbb{P} \models^{lm}_{=1} \mathsf{Spec}$ when Spec is deterministic, and checking $\mathbb{P} \models^{lm}_{>0} \mathsf{Spec}$ when Spec is a safety property.

2 Definitions

2.1 Preliminaries

The powerset of any set A will be denoted by 2^A. Given any set Σ, Σ^+ will denote the set of nonempty finite words over Σ and Σ^ω the set of infinite words over Σ. Given a word $\alpha \in \Sigma^+ \cup \Sigma^\omega$, we will denote the length of α by $length(\alpha)$ (length of α is ω for $\alpha \in \Sigma^\omega$). We assume that the reader is familiar with basic measure theory. We will also assume familiarity with finite automata on infinite strings and Partially Observable Markov Decision Processes (POMDP).

2.1.1 Probabilistic Automata

We recall the definition of probabilistic Büchi automata (PBA)s [3]. Informally, a PBA is like a deterministic Büchi automata except that the transition function from a state on a given input is described as a probability distribution that determines the probability of the next state. Formally, a PBA over a finite alphabet Δ is a tuple $\mathcal{B} = (Q, q_s, Q_f, \delta)$ where Q is a finite set of *states*, $q_s \in Q$ is the *initial state*, $Q_f \subseteq Q$ is the set of *accepting/final states*, and $\delta : Q \times \Delta \times Q \to [0, 1]$ is the *transition relation* such that for all $q \in Q$ and $a \in \Delta$,

$\sum_{q' \in Q} \delta(q, a, q') = 1$. For this paper, we assume that $\delta(q, a, q')$ is a rational number for all $q, q' \in Q$ and $a \in \Delta$.

Intuitively, the PBA $\mathcal{B}$ starts in the initial state q_s and if after reading $a_0, a_1 \ldots, a_i$ results in state q, it moves to state q' with probability $\delta(q, a_{i+1}, q')$ on symbol a_{i+1}. Given a word $\alpha \in \Delta^\omega$, $\mathcal{B}$ can be thought of as an infinite-state Markov chain which gives rise to the standard σ-algebra defined using cylinders and the standard probability measure on Markov chains [25, 21]. We denote this measure by $\mu_{\alpha,\mathcal{B}}$. A *run* of $\mathcal{B}$ is an infinite sequence $\rho \in Q^\omega$. A run ρ is *accepting* if ρ satisfies the Büchi acceptance condition, *i.e.*, $\rho[i] \in Q_f$ for infinitely many i.

The set of accepting runs is measurable. Given α, the measure of the set of accepting runs will be denoted by $\mu^{acc}_{\mathcal{B},\alpha}$ and is said to be the *probability of accepting* α. Given $x \in [0, 1]$ and $\triangleright \in \{>, =, \geq\}$, we let $\mathcal{L}_{\triangleright x}(\mathcal{B}) = \{\alpha \in \Delta^\omega \mid \mu^{acc}_{\mathcal{B},\alpha} \triangleright x\}$.

We identify one useful syntactic restriction of PBAs, called *finite probabilistic monitors* (FPM)s [7]. In a FPM, all the states are accepting except a special absorbing *reject* state (a state q_r is said to be *absorbing* if $\delta(q_r, a, q_r) = 1$ for each input $a \in \Delta$). By using a set of Rabin pairs instead of a set of final states, we can define *Probabilistic Rabin automata* (PRAs).

2.2 Programs

We will denote the set of Boolean expressions over Boolean variables V by $\mathsf{BEXP}(V)$. The value of a Boolean expression BEXP under a truth assignment $s : V \to \{0, 1\}$ will be denoted by $[\![\mathsf{Bexp}]\!]_s$. We use 2^V to denote the set of assignments on V. An *update* to variables in V is a set of assignments of the form $x := \mathsf{Bexp}$, such that each variable appears the left hand side of at most one assignment in the set. An update A defines a function $\mathsf{app}_A : 2^V \to 2^V$ as follows: if $x := \mathsf{Bexp} \in A$ then $\mathsf{app}_A(s)(x) = [\![\mathsf{Bexp}]\!]_s$, and if x is not on the left hand side of any assignment in A then $\mathsf{app}_A(s)(x) = s(x)$. We say that s' is obtained by applying the update A to s if $\mathsf{app}_A(s) = s'$.

A probabilistic concurrent program $\mathbb{P}$ with n processes is a tuple $(V, s_0, (V_1, \mathcal{P}_1), ..., (V_n, \mathcal{P}_n))$. Here V_i is a finite set of Boolean variables that process i reads and writes to, with $V = \cup_{i=1}^n V_i$ being the *set of program variables*. $s_0 \in 2^V$ is the *initial state* of the program, and $\mathcal{P}_i$ is a finite set of transitions of process i defined as follows. Each transition τ of process $\mathcal{P}_i$ is of the form $(C,\ p_1 : A_1, p_2 : A_2, ..., p_k : A_k)$ where C is a Boolean expression on V_i, and $(p_1, ..., p_k)$ is a sequence of nonzero rational probabilities that add up to 1 and $A_1, ..., A_k$ are *updates* such that all the variables appearing (on the left hand side or right hand side of an assignment) in A_j are in V_i. For any i, j, we say that processes i, j communicate if $V_i \cap V_j \neq \emptyset$. For any i, $1 \leq i \leq n$, let $L_i = V_i - \cup_{j \neq i} V_j$, namely, the set of variables of $\mathcal{P}_i$ that are not visible to any other process. The variables in L_i are said to be *local variables* of process i. We will also assume, without loss of generality, that each process has at least one variable — a process i without any variables can be modeled in our framework as a process with one local variable whose value remains constant.

The *states* of $\mathbb{P}$ will be 2^V. Let $\tau = (C,\ p_1 : A_1, p_2 : A_2, ..., p_k : A_k)$ be a transition of a process $\mathcal{P}_i$. We say that τ is *enabled* in state s if C is satisfied in s. The process $\mathcal{P}_i$ is said to be *deterministic (or purely probabilistic)* if for each state s, there is at most one transition of $\mathcal{P}_i$ enabled in s. Assume that τ is enabled in s. If the transition τ is *executed* in state s, then one of the updates $A_1, ..., A_k$ is chosen, with the probability distribution given by $p_1, ..., p_k$ and applied to the state s. Let t_i be the state obtained by performing the update A_i to the state s. We say that the probability that the next state is t_i is p_i when transition τ is executed in state s. We assume that for each state s, there is some process $\mathcal{P}_i$ such that

some transition of $\mathcal{P}_i$ is enabled in s. For any state s and process index i, $1 \leq i \leq n$, we let $s|i$ denote the restriction of s to the variables in V_i. Intuitively, $s|i$ denotes the part of the state that is visible to process i, i.e., the local state.

A program is interpreted using schedulers which, depending on the history, resolve nondeterminism by assigning which of the enabled actions is fired in a given state.

Classes of Schedulers. Let $\mathbb{P}$ be a program with n processes $\mathcal{P}_1, \ldots \mathcal{P}_n$. Let 2^V be the set of states of $\mathbb{P}$ and $Trans$ be the set of transitions of $\mathbb{P}$. A *history* is an element of $(2^V)^+$. Given a history $h = t_0 ... t_m$, we define $last(h)$ to be the state t_m and $length(h)$ to be $m+1$. Given a process index i, $0 \leq i \leq n$, we define $h|i$ to be the word $(t_0|i)(t_1|i)...(t_m|i)$. Intuitively, $h|i$ denotes the view of process i in h.

A scheduler $\eta : (2^V)^+ \rightarrow Trans$ is a function that associates, with each history of a program $\mathbb{P}$, a transition τ of some process of $\mathbb{P}$ that is enabled in the last state of the history. We say that a scheduler η is *view consistent* if the following property holds for every pair of histories h, h' and every process index $1 \leq i \leq n$: if $\eta(h)$, $\eta(h')$ are both transitions of process i and $h|i = h'|i$ then $\eta(h) = \eta(h')$. Intuitively, view consistency requires that the transition of a process, chosen by the scheduler, should depend only on the view of the process; that is, the nondeterminism within a process is resolved based purely on process' view of the computation history. Note that the above condition does not prevent the scheduler from choosing transitions of different processes for h and h'.

We say that η is *locally Markovian* (or *locally step dependent*) if the following property holds for every pair of histories h, h' and every process i: if $\eta(h)$, $\eta(h')$ are both transitions of process i, $length(h) = length(h')$, and $last(h)|i = last(h')|i$, then $\eta(h) = \eta'(h)$. Note that in this case, the transition scheduled should only depend on the length of the history and the current local state of the process. Observe that every locally Markovian scheduler is also view consistent.

Computations. In presence of a scheduler η, a program $\mathbb{P}$ with 2^V as set of states can be thought of as an infinite-state Markov chain which gives rise to the standard σ-algebra on $(2^V)^\omega$ and the standard probability measure [25, 21]. We will denote this Markov chain as $\mathcal{M}_{\mathbb{P},\eta}$ and the standard probability measure generated as $\mu_{\mathcal{M}_{\mathbb{P},\eta}}$. The set $(2^V)^\omega$ shall be called the set of paths of $\mathcal{M}_{\mathbb{P},\eta}$.

Let $\mathcal{A}$ be a Büchi automaton with 2^V (the set of states of $\mathbb{P}$) as its input alphabet. We say that $\mathcal{A}$ accepts an infinite path $\rho \in (2^V)^\omega$ of $\mathcal{M}_{\mathbb{P},\eta}$, if it accepts the infinite sequence ρ. Let $\mathcal{L}(\mathcal{A})$ be the language accepted by $\mathcal{A}$. Now $\mathcal{L}(\mathcal{A})$ is a measurable set in the space of paths defined by $\mathcal{M}_{\mathbb{P},\eta}$ and we call the measure of $\mathcal{L}(\mathcal{A})$, $\mu_{\mathcal{M}_{\mathbb{P},\eta}}(\mathcal{L}(\mathcal{A}))$, to be the *probability of acceptance of* $\mathcal{M}_{\mathbb{P},\eta}$. Given a rational number x and $\triangleright \in \{>, =, \geq\}$, we shall write $\mathbb{P}, \eta \models_{\triangleright x} \mathcal{A}$ if $\mu_{\mathcal{M}_{\mathbb{P},\eta}}(\mathcal{L}(\mathcal{A})) \triangleright x$. The automaton $\mathcal{A}$ will be henceforth called a *specification automaton.*

Predicate automaton. It is often useful to present the specification automaton $\mathcal{A}$ succinctly in the following fashion. A *predicate automaton* Spec is a tuple $(V, Q, q_s, Q_f, \rightarrow)$ where V is a finite set of boolean variables, Q is a finite set of *states*, $q_s \in Q$ is the *initial state*, $Q_f \subseteq Q$ is the set of *final states* and $\rightarrow \subseteq Q \times \mathsf{BEXP}(V) \times Q$ is a *finite* set of *predicate transitions.* Given a predicate automaton Spec, we define a specification automaton $[\![\mathsf{Spec}]\!]$ as follows: $[\![\mathsf{Spec}]\!] = (2^V, Q, q_s, Q_f, \delta)$ where $(q, s, q') \in \delta$ iff there is a predicate transition $(q, \mathsf{Bexp}, q') \in \rightarrow$ such that s satisfies Bexp. Please note given any specification (Büchi) automaton $\mathcal{A}$, there is a predicate automaton Spec such that $[\![\mathsf{Spec}]\!] = \mathcal{A}$. Furthermore, we will say that Spec is a deterministic predicate automaton (respectively safety) if $[\![\mathsf{Spec}]\!]$ is a deterministic automaton (respectively safety automaton). Whenever convenient, we will often confuse Spec with $[\![\mathsf{Spec}]\!]$. Given any history $h \in \Sigma^*$ let $\mathsf{Spec}(h)$ be the state that

	ω-REGULAR SPECIFICATION	DETERMINISTIC SPECIFICATION	SAFETY SPECIFICATION
$\mathbb{P} \models^{vc}_{=1}$ Spec	Undecidable	?[b]	**PSPACE**-complete
$\mathbb{P} \models^{lm}_{=1}$ Spec	Undecidable	**EXPSPACE**-complete	**PSPACE**-complete
$\mathbb{P} \models_{=1}$ Spec	**EXPTIME**-complete[a]	**EXPTIME**-complete	**PSPACE**-complete
$\mathbb{P} \models^{vc}_{>0}$ Spec	Undecidable	Undecidable	?[b]
$\mathbb{P} \models^{lm}_{>0}$ Spec	Undecidable	Undecidable	**EXPSPACE**-complete
$\mathbb{P} \models_{>0}$ Spec	**EXPTIME**-complete[a]	**EXPTIME**-complete	**EXPTIME**-complete
$\mathbb{P} \models^{vc}_{\triangleright x}$ Spec	Undecidable	Undecidable	Undecidable
$\mathbb{P} \models^{lm}_{\triangleright x}$ Spec	Undecidable	Undecidable	Undecidable
$\mathbb{P} \models_{\triangleright x}$ Spec	**EXPTIME**-complete[a]	**EXPTIME**-complete	**EXPTIME**-complete

Figure 1 Summary of complexity results. For entries with superscript a, we assume that Spec is given as a deterministic predicate Rabin automaton. For entries with superscript b, the problem becomes 2-**EXPTIME**-complete if either (i) all but one processes of $\mathbb{P}$ are deterministic (purely probabilistic), or (ii) if the processes communicate through global variables and are mutually exclusive, and Spec is on the shared state.

$[\![\mathsf{Spec}]\!]$ is in after reading h from its initial state.

Similar to the predicate automaton, we can define a *predicate Rabin automaton*, in which instead of using a set of final states, we use a set of Rabin pairs. As in the case of predicate automaton, a predicate Rabin automaton gives rise to a Rabin automaton.

Verification. Given a rational number x and $\triangleright \in \{\geq, =, >\}$, we will write $\mathbb{P} \models_{\triangleright x}$ Spec if for every scheduler η, we have that $\mathbb{P}, \eta \models_{\triangleright x} [\![\mathsf{Spec}]\!]$. Similarly, we write $\mathbb{P} \models^{vc}_{\triangleright x}$ Spec ($\mathbb{P} \models^{lm}_{\triangleright x}$ Spec, respectively) if for every view consistent scheduler η (locally Markovian scheduler, respectively), $\mathbb{P}, \eta \models_{\triangleright x} [\![\mathsf{Spec}]\!]$. Thus, the verification problem we consider is one where given $\mathbb{P}$, Spec, rational number $x \in [0, 1]$, and $\triangleright \in \{\geq, =, >\}$ as input, we want to determine if $\mathbb{P} \models_{\triangleright x}$ Spec (or $\mathbb{P} \models^{vc}_{\triangleright x}$ Spec or $\mathbb{P} \models^{lm}_{\triangleright x}$ Spec). The predicate automaton Spec is often called the *specification*.

3 Complexity and decidability for general programs

In this section, we present results on the decidability and complexity of the verification problems defined in Section 2. Our results are summarized in Figure 1. We will now state the results. The missing proofs as well as all the proofs/results of all schedulers can be found in [9].

3.1 Undecidability

We start by establishing the undecidability of the model checking problem for concurrent programs in a variety of settings.

▶ **Theorem 3.1.** *Given a program $\mathbb{P}$ and a predicate automaton* Spec*, the following problems are undecidable.*

(a) Determining if $\mathbb{P} \models^{vc}_{=1}$ Spec and $\mathbb{P} \models^{lm}_{=1}$ Spec.

(b) Determining if $\mathbb{P} \models^{vc}_{>0}$ Spec and $\mathbb{P} \models^{lm}_{>0}$ Spec, even when Spec *is a deterministic specification.*

(c) Given a rational $x \in (0, 1)$ and $\triangleright \in \{>, \geq\}$, determining if $\mathbb{P} \models^{vc}_{\triangleright x}$ Spec and $\mathbb{P} \models^{lm}_{\triangleright x}$ Spec, even when Spec *is a safety specification.*

The above undecidability results continue to hold even if $\mathbb{P}$ is restricted to be a program consisting of two processes, one of which is purely nondeterministic (no probabilistic transitions) and the other is purely probabilistic (no nondeterministic transitions).

Theorem 3.1 is proved as follows: for part (a) we reduce the problem of checking if a given PRA accepts every input with probability 1; for part (b) we reduce the problem of checking if a given PBA accepts every input with probability > 0; and for part (c) we reduce the problem of checking if a given FPM accepts every input with probability $\triangleright x$.

For a program $\mathbb{P}$, the observations in Theorem 3.1, leave open the decidability of the following questions.

(a) if Spec is a safety specification, check if $\mathbb{P} \models^{vc}_{=1}$ Spec (or check if $\mathbb{P} \models^{lm}_{=1}$ Spec)?
(b) if Spec is a safety specification, check if $\mathbb{P} \models^{vc}_{>0}$ Spec (or check if $\mathbb{P} \models^{lm}_{>0}$ Spec)?
(c) if Spec is a deterministic specification, check if $\mathbb{P} \models^{vc}_{=1}$ Spec (or check if $\mathbb{P} \models^{lm}_{=1}$ Spec)?

We address these questions in the forthcoming sections. The decidability of (a) is shown in Theorem 3.2 in Section 3.2. The problems in (b) and (c) are also shown to be decidable for locally Markovian schedulers, in Section 3.2; these problems remain open for the case of view consistent schedulers. However, in Section 3.3, we show that these problems are decidable for two special classes of programs.

▶ Remark. Distributed schedulers, introduced in [18] and further studied in [16, 17], are very similar to view consistent schedulers that we consider here. However, there is one important, subtle difference between them. In a view consistent scheduler, the local scheduler of process i is aware of both the steps when process i was scheduled and those when it was not scheduled; in distributed schedulers the local scheduler is only aware of the steps when it was scheduled. Thus, the undecidability results presented here do not follow from [18, 16, 17]. Moreover, in [16, 17], the problem of checking if $\mathbb{P} \models_{>0}$ Spec for safety specifications Spec under all distributed schedulers is shown to be undecidable; however, that proof does not extend to view consistent schedulers and this problem for view consistent schedulers (as stated in the discussion above) is open.

3.2 Decidability results for locally Markovian semantics

We begin by establishing the decidability of checking if the measure of computations accepted by a safety specification, under every scheduler in a class $\mathcal{C}$, is 1. We, in fact, show that for any of the three classes of schedulers that we consider, this problem is **PSPACE**-complete.

▶ **Proposition 3.2.** *Given a program $\mathbb{P}$ and a safety specification* Spec*, the following problems are* **PSPACE***-complete: determining if $\mathbb{P} \models^{vc}_{=1}$* Spec*, if $\mathbb{P} \models^{vc}_{=1}$* Spec *and if $\mathbb{P} \models^{lm}_{=1}$* Spec*.*

We now establish that the problems of determining if $\mathbb{P} \models^{lm}_{>0}$ Spec when Spec is a safety specification, and of determining if $\mathbb{P} \models^{lm}_{=1}$ Spec when Spec is deterministic are **EXPSPACE**-complete. We begin by defining a special class of locally Markovian schedulers that we call Spec-*determined*. These are schedulers that are required to choose the same transition after equal length histories h and h' that end in the same state, if the state reached by the specification $[\![\mathsf{Spec}]\!]$ after h (namely, $\mathsf{Spec}(h)$) is the same as $\mathsf{Spec}(h')$.

▶ **Definition 3.3.** Let $\mathbb{P}$ be a program with n processes, and let Σ be the set of states of $\mathbb{P}$ and $Trans$ be the set of transitions of $\mathbb{P}$. Let Spec be a deterministic specification with Q as the set of states. We say that a locally Markovian scheduler $\eta : \Sigma^+ \to Trans$ is Spec-*determined* if for any pair of histories $h, h' \in \Sigma^+$ such that $length(h) = length(h')$, $\mathsf{Spec}(h) = \mathsf{Spec}(h')$ and $last(h) = last(h')$, we have that $\eta(h) = \eta(h')$.

The reason for considering Spec-determined schedulers is because we can show that for the problems of verifying safety with non-zero probability and verifying deterministic specifications with probability 1, we can restrict our attention to Spec-determined schedulers. This is the content of the next proposition.

▶ **Proposition 3.4.** *For any program* $\mathbb{P}$ *and safety specification* Spec*,* $\mathbb{P} \models^{lm}_{>0}$ Spec *iff for any* Spec*-determined locally Markovian scheduler* η*;* $\mathbb{P}, \eta \models_{>0}$ Spec*. For deterministic specification* Spec*,* $\mathbb{P} \models^{lm}_{=1}$ Spec *iff for all* Spec*-determined locally Markovian schedulers* η*;* $\mathbb{P}, \eta \models_{=1}$ Spec*.*

We need one more definition.

▶ **Definition 3.5.** Let $\mathbb{P}$ be a program with n processes with V as the set of variables and $Trans$ as the set of transitions. Let Spec be a deterministic specification. We say a function $g : Q \times 2^V \to Trans$ is Spec-*determined* and *locally consistent* if for all $q \in Q$ and $s \in 2^V$, $g((q, s))$ is enabled in s; and $g((q_1, s_1)) = g((q_2, s_2))$ whenever $(q_1, s_1), (q_2, s_2) \in Q \times 2^V$ are such that $g(q_1, s_1), g(q_2, s_2)$ belong to the same process $\mathcal{P}_i$ and $s_1|i = s_2|i$. The set of Spec-determined and locally consistent functions of program $\mathbb{P}$ shall be denoted as $\mathsf{Loc}(\mathbb{P}, \mathsf{Spec})$.

Given a deterministic specification Spec, it is easy to see that there is a bijection between the set of Spec-determined locally Markovian schedulers of a program $\mathbb{P}$ and $(\mathsf{Loc}(\mathbb{P}, \mathsf{Spec}))^\omega$, the set of infinite sequences over $(\mathsf{Loc}(\mathbb{P}, \mathsf{Spec}))$. We call this function $\mathsf{Loc}_{\mathbb{P},\mathsf{Spec}}$. The key technical idea exploited in our model checking algorithm is the following. Given a program $\mathbb{P}$ and a specification Spec, one can construct a PBA $\mathcal{B}$ that accepts a word $\mathsf{Loc}_{\mathbb{P},\mathsf{Spec}}(\eta)$ with the same probability as the computation of $\mathcal{P}$ under scheduler η satisfies Spec.

▶ **Lemma 3.6.** *Given a program* $\mathbb{P}$ *and a deterministic specification* Spec*, let* Δ *be* $\mathsf{Loc}(\mathbb{P}, \mathsf{Spec})$*, the set of* Spec*-determined locally consistent functions. There is a* PBA $\mathcal{B}$ *on input alphabet* Δ *such that the following hold–*

- *The number of states of* $\mathcal{B}$ *is exponential in the size of* $\mathbb{P}$ *and* Spec*.*
- *For any* Spec*-determined locally Markovian scheduler* η*, the probability that the computation* $\mathcal{M}_{\mathbb{P},\eta}$ *satisfies* Spec *is the probability of* $\mathsf{Loc}_{\mathbb{P},\mathsf{Spec}}(\eta)$ *being accepted by* $\mathcal{B}$*.*
- $\mathcal{B}$ *can be taken to be a* FPM *if* Spec *is a safety specification.*

We have the following theorem.

▶ **Theorem 3.7.** *Given a program* $\mathbb{P}$ *and a deterministic specification* Spec *the problem of determining if* $\mathbb{P} \models^{lm}_{=1}$ Spec *is* **EXPSPACE***-complete. If* Spec *is a safety specification, then the problem of determining if* $\mathbb{P} \models^{lm}_{>0}$ Spec *is also* **EXPSPACE***-complete.*

We had shown in [7, 8] that given a PBA $\mathcal{B}$, the problem of checking whether all words are accepted with probability 1 is in **PSPACE**. We had also shown in [7] that given a FPM $\mathcal{M}$, the problem of checking whether all words are accepted with probability > 0 is in **PSPACE**. In light of Lemma 3.6, this immediately implies that the problems of determining whether a program satisfies a deterministic specification with probability 1 and whether a program satisfies a safety specification with nonzero probability are decidable. However, note that as the input alphabet constructed in Lemma 3.6 is doubly-exponential in the size of the input, the straightforward application of the results in [7, 8] do not lead to inclusion in **EXPSPACE**. The inclusion in **EXPSPACE** is achieved by a careful examination of algorithms given in [7, 8] and running the algorithm without explicitly constructing the PBA.

3.3 Decidability results for view consistent semantics

Proposition 3.2 already establishes that checking whether every computation of a program $\mathbb{P}$ generated by a view consistent scheduler satisfies a safety specification with probability 1 is **PSPACE**-complete. We now consider the remaining questions, namely, checking whether $\mathbb{P} \models^{vc}_{>0}$ Spec when Spec is a safety property, and checking whether $\mathbb{P} \models^{vc}_{=1}$ Spec when Spec is a deterministic specification. While the decidability of these problems is open, we prove decidability for special classes of programs $\mathbb{P}$. First we show that these problems are **2-EXPTIME**-complete when all processes of $\mathbb{P}$, except possibly one, are deterministic.

▶ **Theorem 3.8.** *Given a program $\mathbb{P}$ and a deterministic specification* Spec *such that all processes of $\mathbb{P}$ except one are deterministic, the problem of checking if $\mathbb{P} \models^{vc}_{=1}$* Spec *is* **2-EXPTIME***-complete. Given a safety specification* Spec*, the problem of checking if $\mathbb{P} \models^{vc}_{>0}$* Spec *is also* **2-EXPTIME***-complete.*

The main idea behind the proof of **2-EXPTIME** membership is to reduce it to model checking POMDPs. The following is proved in [9].

▶ **Lemma 3.9.** *Given a program $\mathbb{P}$ and a deterministic specification (safety specification, respectively)* Spec *such that all processes of $\mathbb{P}$ except one are deterministic, there is a* POMDP *$\mathcal{M}$ and a subset Q of states of $\mathcal{M}$ such that $\mathbb{P} \models^{vc}_{=1}$* Spec *($\mathbb{P} \models^{vc}_{>0}$* Spec*, respectively) iff under every observation based scheduler, the measure of paths of $\mathcal{M}$ that visit Q infinitely often is 1 (> 0, respectively).*

The second special class of programs that we consider are the following. Processes in a program are said to communicate through *global variables* if every pair of processes share the same set of variables. We say that processes in $\mathbb{P}$ are *mutually exclusive* if in every state the transitions of only one process are enabled. A deterministic specification Spec is said to be *on the shared state* if whenever (q, Bexp, q') is a transition of Spec, Bexp evaluates to the same value for any two program states in which the global variables take the same value.

▶ **Theorem 3.10.** *Given a program $\mathbb{P}$ where the processes communicate through global variables and are mutually exclusive, and a deterministic specification (safety specification, respectively)* Spec *on the shared state, the problem of checking $\mathbb{P} \models^{vc}_{=1}$* Spec *($\mathbb{P} \models^{vc}_{>0}$* Spec*, respectively) is* **2-EXPTIME***-complete.*

The proof of the **2-EXPTIME**-decidability relies on showing that if there is a scheduler η such that $\mathbb{P}, \eta \models_{<1}$ Spec ($\mathbb{P}, \eta \models_{=0}$ Spec) for a deterministic specification on the shared state (safety specification) then there is a "periodic" scheduler η' that witnesses the same fact. The model checking algorithm searches for such a periodic scheduler by reducing it to μ-calculus model checking on a finite (doubly exponentially sized) bi-partite graph $\mathcal{G}$. We illustrate the construction of the graph $\mathcal{G}$ for the case when Spec is a deterministic specification.

Let $\mathbb{P}$ be a program where the processes communicate through global variables and are mutually exclusive, and Spec be a deterministic specification on the shared state. We start by some definitions. Assume that $\mathbb{P}$ has n processes, V is the set of shared variables and V_i is the set of local variables of process i. It is easy to see that the set of states of $\mathbb{P}$ can be taken to be $2^V \times 2^{V_1} \times \ldots \times 2^{V_n}$. We henceforth refer to this set as $States(\mathbb{P})$. If Q is the set of states of Spec then the set $Q \times States(\mathbb{P})$ is said to be the set of *extended states* and will be referred to by $EStates(\mathsf{Spec}, \mathbb{P})$. An extended state $es = (q, s)$ is said to be *feasible* if there is a history h of $\mathbb{P}$ such that the measure of h is > 0, $h(0)$ is the initial state of $\mathbb{P}$, $last(h) = s$ and $\mathsf{Spec}(h) = q$. Let $\pi_{\mathbb{P}} : EStates(\mathsf{Spec}, \mathbb{P}) \to States(\mathbb{P})$ be the map $\pi_{\mathbb{P}}((q, s)) = s$. Given $es = (q, s) \in EStates(\mathsf{Spec}, \mathbb{P})$ and a Spec-determined and locally consistent function g (see Definition 3.5), let $\mathsf{succ}_g(es) = \{(q', s') \mid (q, s, q')$ is a transition of $[\![\mathsf{Spec}]\!]$ & s' is obtained with nonzero probability when $g((q, s))$ is executed in $s\}$. Given a set $U \subseteq EStates(\mathsf{Spec}, \mathbb{P})$, let $\mathsf{succ}_g(U) = \cup_{es \in U} \mathsf{succ}_g(es)$.

A set of states $S \subseteq States(\mathbb{P})$ is said to be *closed* if $S = \{v\} \times S_1 \times \ldots \times S_n$ for some $v \in 2^V$ and $S_i \in 2^{V_i}$. A set $U \subseteq EStates(\mathsf{Spec}, \mathbb{P})$ is said to be an *extended closed set* if $\pi_{\mathbb{P}}(U)$ is closed. We show in [9] that for any extended closed set U, $\mathsf{succ}_g(U)$ can be partitioned into a union of *disconnected* extended closed sets. Two extended closed sets U_1 and U_2 are said to be disconnected if for each $es_1 \in U_1$ and $es_2 \in U_2$ and each process i, $\pi_{\mathbb{P}}(es_1)|i \neq \pi_{\mathbb{P}}(es_2)|i$. We shall call these disconnected sets *components* of $\mathsf{succ}_g(U)$.

The bi-partite graph $\mathcal{G}$ consists of 2 partitions, W_1 and W_2. The set W_1 is the set of extended closed sets. W_2 is the set of pairs (U, g) where U is an extended closed set and g a Spec-determined and locally consistent function. There is an edge from $U \in W_1$ to $(U', g) \in W_2$ iff $U = U'$. There is an edge from (U, g) to U' iff U' is a component of $\mathsf{succ}_g(U)$. We convert $\mathcal{G}$ into a Kripke structure by labeling each node of $\mathcal{G}$ by a special proposition F or its negation $\neg F$ as follows. A node U in W_1 is labeled with F iff there is an extended state $es = (q, s) \in U$ such that q is a final state of Spec. Every other node of W_1 and each node of W_2 is labeled by $\neg F$. We denote the resulting Kripke structure by $\mathcal{G}(\mathsf{Spec}, \mathbb{P})$. The 2-**EXPTIME**-decidability of checking if $\mathbb{P} \models^{vc}_{=1} \mathsf{Spec}$ follows from the following lemma shown in [9].

▶ **Lemma 3.11.** Given a program $\mathbb{P}$ where the processes communicate through global variables and are mutually exclusive, and a deterministic specification Spec on the shared state, let $\mathcal{G}(\mathsf{Spec}, \mathbb{P})$ be the Kripke structure obtained as described above. There is a view consistent scheduler η such that $\mathcal{M}_{\mathbb{P},\eta}$ satisfies the specification Spec with probability < 1 iff there is a feasible extended state es such that the node $\{es\}$ in $\mathcal{G}(\mathsf{Spec}, \mathbb{P})$ satisfies the modal μ-calculus formula $f = \nu X(\neg F \wedge \Diamond\Box X)$ where $\Diamond$ and $\Box$ are the existential and universal "nexttime" operators and νX is the greatest fixpoint operator.

4 Conclusions

Randomization and nondeterminism play an important role in concurrent processes, and in this paper we showed that to get accurate verification results, one needs to consider restricted classes of schedulers. Tight complexity bounds for verifying linear time properties under restricted classes of schedulers were established. Our complexity results confirm observations made in [13] that restricting the class of schedulers makes the verification problem more difficult.

The global schedulers we consider can observe all the variables in the program. There may be situations when we want to restrict the power of the global scheduler as well. However, this is easily captured in our setting by adding a new process P_{new} that can only see a part of the state that is visible to the restricted global scheduler. This new process will execute odd step (ensured by adding a new turn variable), and will pick the process to schedule based on the partial state it sees.

View consistent and locally Markovian schedulers are just some of the classes that might be useful in the concurrent setting. One natural class of schedulers we have not explicitly mentioned in this paper are memoryless schedulers, where the choice made by the scheduler depends only on the current state and not on the history. It is easy to see that the verification problems are in co-**NEXPTIME**— guess the scheduler that violates the property, and check that under the scheduler the system (which is now a finite state MDP) violates the property. The verification problems are also likely to be co-**NEXPTIME**-hard based on observations made in [13]; once again the blowup in complexity being explained by the state space explosion problem. One restriction of the schedulers we consider here is that the local scheduler for a process is "aware" of the fact that other processes were scheduled. This may or may not be reasonable in some settings. In the future we would like to expand the current investigations to other useful classes of schedulers.

Acknowledgements. The authors would like to thank anonymous referees for sending pointers to [18, 16, 17]. A. Prasad Sistla was supported by NSF-0720525, NSF CCF-0916438, NSF CNS-1035914 and Mahesh Viswanathan was supported by NSF CCF 0448178, NSF CCF 1016989, and NSF CNS 1016791.

References

1. PRISM — Probabilistic Symbolic Model Checker. `http://www.prismmodelchecker.org`.
2. C. Baier, N. Bertrand, and M. Größer. On decision problems for probabilistic Büchi automata. In *Proceedings of FoSSaCS*, pages 287–301, 2008.
3. C. Baier and M. Grö„er. Recognizing ω-regular languages with probabilistic automata. In *Proceedings of LICS*, pages 137–146, 2005.
4. C. Baier, B. Haverkrot, H. Hermanns, and J.-P. Katoen. Efficient computation of time-bounded reachability probabilisties in uniform continuous-time Markov decision processes. In *Proceedings of TACAS*, pages 61–76, 2004.
5. A. Bianco and L. de Alfaro. Model checking of probabalistic and nondeterministic systems. In *Proceedings of FSTTCS*, pages 499–513, 1995.
6. R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, P. Pereira, and R. Segala. Task-Structured Probabilistic I/O Automata. In *Workshop on Discrete Event Systems*, 2006.
7. R. Chadha, A. P. Sistla, and M. Viswanathan. On the expressiveness and complexity of randomization in finite state monitors. *Journal of the ACM*, 56(5), 2009.
8. R. Chadha, A. P. Sistla, and M. Viswanathan. Power of randomization in automata on infinite strings. In *Proceedings of CONCUR*, pages 229–243, 2009.
9. R. Chadha, A. P. Sistla, and M. Viswanathan. Model checking concurrent programs with nondeterminism and randomization. Technical Report LSV-10-15, LSV, ENS Cachan, 2010.
10. K. Chatterjee, L. Doyen, and T. Henzinger. Qualitative Analysis of Partially-observed Markov Decision Processes. *CoRR*, abs/0909.1645, 2009.
11. K. Chatzikokolakis and C. Palamidessi. Making Random Choices Invisible to the Scheduler. *Information and Computation*, 2010, to appear.
12. L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. PhD thesis, Radboud University of Nijmegen, 2006.
13. L. de Alfaro. The Verification of Probabilistic Systems under Memoryless Partial Information Policies is Hard. In *Proceedings of PROBMIV*, 1999.
14. L. de Alfaro, T. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In *Proceedings of CONCUR*, pages 351–365, 2001.
15. F.D. Garcia, P. van Rossum, and A. Sokolova. Probabilistic Anonymity and Admissible Schedulers. *CoRR*, abs/0706.1019, 2007.
16. S. Giro. Undecidability results for distributed probabilistic systems. In *Proceedings of SBMF*, pages 220–235, 2009.
17. S. Giro. *On the automatic verification of Distributed Probabilistic Automata with Partial Information*. PhD thesis, Universidad Nacional de Córdoba, 2010.
18. S. Giro and P.R. D'Argenio. Quantitative model checking revisited: Neither decidable nor approximable. In *Proceedings of FORMATS*, pages 179–194, 2007.
19. S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of STOC*, pages 365–377, 1982.
20. M. Größer. *Reduction Meth. for Prob. Model Checking*. PhD thesis, TU Dresden, 2008.
21. J. Kemeny and J. Snell. *Denumerable Markov Chains*. Springer-Verlag, 1976.
22. N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
23. M.L. Puterman. *Markov Decision Processes: Discrete Stocastic Dynamical Programming*. John Wiley & Sons, 1994.
24. J. M. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*. AMS, 2004.
25. M. Vardi. Automatic verification of probabilistic concurrent systems. In *Proceedings of FOCS*, pages 327–338, 1985.