Model Checking Concurrent Programs with Nondeterminism and Randomization

Rohit Chadha\textsuperscript{1}, A. Prasad Sistla\textsuperscript{2}, Mahesh Viswanathan\textsuperscript{3}

\textsuperscript{1}INRIA & LSV, ENS Cachan and CNRS
\textsuperscript{2}University of Illinois, Chicago
\textsuperscript{3}University of Illinois, Urbana-Champaign

\textbf{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 \textit{view consistent} and \textit{locally Markovian} schedulers and consider the model checking problem of concurrent, probabilistic programs under these alternate semantics. Specifically, given a Büchi automaton $\text{Spec}$, a threshold $x \in [0, 1]$, and a concurrent program $P$, the model checking problem asks if the measure of computations of $P$ that satisfy $\text{Spec}$ 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.

\section{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 [21] and in cryptographic protocols to achieve semantic security [18]. The formal analysis of such systems has often modelled them as \textit{Markov Decision Processes} [23], 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 \textit{scheduler} or \textit{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, 23] 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 [12, 11, 6, 14, 10] that in a number of applications, taking such a pessimistic view and considering all possible schedulers, can yield incorrect verification results (see Section C for examples). The problem arises when one considers a concurrent system where individual processes exhibit both probabilistic and nondeterministic behavior. For such systems, there are certain \textit{perfect information} schedulers that will resolve local process-level nondeterminism based on information that would not be available to the local process, and thereby exhibit behavior that is unreasonable. Therefore, in analyzing concurrent programs, in many cases, it is better 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 \( \sigma \) and \( \tau \) are computations such that the states as observable to process \( P \) are identical at every step, then the local scheduler for \( P \) must pick the same transition after both \( \sigma \) and \( \tau \). 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 [17, 15, 16]. However, there is subtle difference between distributed schedulers and view consistent schedulers (see Related Work) and the results presented here do not follow from those in [17, 15, 16]. In this paper, we also consider another class of restricted schedulers that we call **locally Markovian** or **locally step dependent.** 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 [22, 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 \( \text{Spec} \), whose input alphabet consists of the states of the program \( P \), and a threshold \( x \in [0, 1] \). We say \( P \models_{\geq x} \text{Spec} \) (\( P \models_{\geq x} \text{Spec} \)), where \( \triangleright \in \{>, \geq\} \), if under all view consistent schedulers (locally Markovian schedulers) the measure of computations of \( P \) accepted by \( \text{Spec} \) is \( \triangleright x \). 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 \( P \models_{\geq 0} \text{Spec} \) and \( P \models_{\leq 0} \text{Spec} \), remain undecidable even when \( \text{Spec} \) is a deterministic Büchi automaton. For the case when \( x \in (0, 1) \), the problems of checking \( P \models_{\geq x} \text{Spec} \), \( P \models_{\geq x} \text{Spec} \), \( P \models_{\geq x} \text{Spec} \), and \( P \models_{\leq x} \text{Spec} \), remain undecidable even when \( \text{Spec} \) is a safety automaton.*

We then investigate the complexity of the verification problems left open by the above undecidability results. Namely, we consider the problems of checking \( \text{Spec} \) 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 \( P \models_{\geq 1} \text{Spec} \) and \( P \models_{=1} \text{Spec} \) are **PSPACE**-complete, where \( \text{Spec} \) is a safety automaton; checking \( P \models_{=1} \text{Spec} \), when \( \text{Spec} \) is deterministic, is **EXPSPACE**-complete; and checking \( P \models_{\geq 0} \text{Spec} \), when \( \text{Spec} \) is a safety automaton, is also **EXPSPACE**-complete. The decidability/complexity of checking \( P \models_{\geq 1} \text{Spec} \) when \( \text{Spec} \) is deterministic, and checking \( P \models_{\geq 0} \text{Spec} \) when \( \text{Spec} \) is a safety property, remain open. However, we show that these model checking problems for view consistent schedulers are decidable in **2-EXPTIME**, for two special classes of programs. The first class is that of programs \( P \) where all processes, except possibly one, are purely probabilistic (i.e., have no local nondeterminism). The second class of programs

---

*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.
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 the specification to be on the global states, 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 [12], 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 proof of Theorem 1 and Lemma 6). Partially Observable MDPs [12] 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. Sections 3 and 4 contain the technical results and the conclusions are presented in Section 5. Motivating examples and proofs of most of the theorems are given in the appendix.

Related Work

Restricting the class of schedulers has been observed to be important in obtaining compositional reasoning principles [13], and in correctly analyzing security protocols and distributed algorithms [12, 11, 6, 14, 10]. The schedulers considered are very similar to the class of view consistent schedulers that we consider. In [13], 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 [11, 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 [10], 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 [17, 15, 16], 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 is manifested by the fact that \( P \models \forall i \geq 0 \text{Spec} \) for safety specifications is undecidable for distributed schedulers \([15, 16]\), whereas it is open for view consistent schedulers. Furthermore, no decidability results are presented in \([17, 15, 16]\).

As indicated in the introduction, the model checking problem and it’s complexity for the related model of Partially Observable MDPs (POMDP) has been investigated in earlier works \([12, 19, 2, 9]\). A POMDP \( \mathcal{P} \) 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 \( P_1 \), and the second process (say \( P_2 \)) plays the role of “scheduling” the next transition of \( P_1 \). They share 3 variables: state that stores the partial state of \( P_1 \) that is visible outside, trans that is used by \( P_2 \) to inform \( P_1 \) what the next transition should be, and turn which is used by the processes to alternate taking turns. In each “round”, \( P_2 \) first picks \( P_1 \)’s next transition, and then \( P_1 \) “executes” that transition; observe, that \( P_1 \) is a purely probabilistic process, and all the nondeterminism has been deferred to \( P_2 \). Under view consistent schedulers, the two processes \( P_1 \) and \( P_2 \) 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 13). 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 \( \mathcal{P} \models \text{Spec} \) when \( \text{Spec} \) is deterministic, and checking \( \mathcal{P} \models \text{Spec} \) when \( \text{Spec} \) is a safety property.

## 2 Definitions

### 2.1 Preliminaries

The powerset of any set \( A \) will be denoted by \( 2^A \). Given any set \( \Sigma, \Sigma^+ \) will denote the set of nonempty finite words over \( \Sigma \) and \( \Sigma^\omega \) the set of infinite words over \( \Sigma \). Given a word \( \alpha \in \Sigma^+ \cup \Sigma^\omega \), we will denote the length of \( \alpha \) by \( \text{length}(\alpha) \) (length of \( \alpha \) is \( \omega \) 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); for reference we have introduced Büchi automata, safety automaton and Rabin automata in Appendix A and POMDPs in Appendix B.

**Probabilistic automata.** We recall the definition of probabilistic Büchi automata (PBAs) \([3]\). 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 \( \Delta \) 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 \( \delta: Q \times \Delta \times Q \rightarrow [0, 1] \) 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 \( \sigma \)-algebra defined using cylinders and the standard probability measure on Markov chains \([24, 20]\). We denote
this measure by \( \mu_{b,B} \). A run of \( B \) is an infinite sequence \( \rho \in Q^\omega \). A run \( \rho \) is accepting if \( \rho \) satisfies the Büchi acceptance condition, i.e., \( \rho[i] \in Q_f \) for infinitely many \( i \). The set of accepting runs is measurable. Given \( \alpha \), the measure of the set of accepting runs will be denoted by \( \mu^\alpha_{B,\alpha} \) and is said to be the probability of accepting \( \alpha \). Given \( x \in [0,1] \) and \( \tau \in \{ >, =, \geq \} \), we let \( L_{0 \tau}(B) = \{ \alpha \in \Delta^\omega \mid \mu^\alpha_{B,\alpha} \triangleright x \} \).

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 BEXP(\( V \)). The value of a Boolean expression BEXP under a truth assignment \( s : V \rightarrow \{0,1\} \) will be denoted by [Bexp]_s. We use \( 2^V \) to denote the set of assignments on \( V \). An update to variables in \( V \) is given by the following grammar, \( A := \text{skip} \mid x := \text{Bexp}, A \). An update \( A \) defines a function inductively \( \text{app}_A : 2^V \rightarrow 2^V \) as follows: if \( A \) is skip then \( \text{app}_A(s) = s \) and if \( A \) is \( x := \text{Bexp}; A' \) then \( \text{app}_A(s) = \text{app}_{A'}(s[x \mapsto [\text{Bexp}]_s]) \), where \( s[x \mapsto v] \) denotes the assignment \( s' \) such that \( s'(x) = v \) and \( s'(y) = s(y) \) for \( y \in V \setminus \{ x \} \). We say that \( s' \) is obtained by applying the update \( A \) to \( s \) if \( \text{app}_A(s) = s' \).

A probabilistic concurrent program \( P \) with \( n \) processes is a tuple \(( V, s_0, (V_1, P_1), ..., (V_n, P_n) ) \). Here \( V_i \) is a finite set of Boolean variables \(^1 \) that process \( i \) reads and writes to, with \( V = \bigcup_{i=1}^n V - i \) being the set of program variables. \( s_0 \in 2^V \) is the initial state of the program, and \( P_i \) is a finite set of transitions of process \( i \) defined as follows. Each transition of a process \( P_i \) is a pair \(( i, \tau ) \) where \( \tau \) 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 to the variables in \( V_i \). For a transition \(( i, \tau ) \) of process \( P_i \), it will often be convenient to drop \( i \) and just say that \( \tau \) is a transition of process \( P_i \). 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 \( P_i \) that are not visible to any other process.

The states of \( P \) 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 \( P_i \). We say that \( \tau \) is enabled in state \( s \) if \( C \) is satisfied in \( s \). The process \( P_i \) is said to be deterministic (or purely probabilistic) if for each state \( s \), there is at most one transition of \( P_i \) enabled in \( s \). Assume that \( \tau \) is enabled in \( s \). If the transition \( \tau \) 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 \( \tau \) is executed in state \( s \). We assume that for each state \( s \), there is some process \( P_j \) such that some transition of \( P_j \) 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 resolve nondeterminism by assigning (depending on the history) which of the enabled actions is fired in a given state.

\(^1\)In proofs we will find it convenient to consider variables to take values in a finite domain, rather than Boolean values.
Classes of Schedulers. Let $P$ be a program with $n$ processes $P_1, \ldots, P_n$. Let $2^V$ be the set of states of $P$ and $\text{Trans}$ be the set of transitions of $P$. A history is an element of $(2^V)^+$. Given a history $h = t_0 \ldots t_m$, we define $\text{last}(h)$ to be the state $t_m$ and $\text{length}(h)$ to be $m + 1$. Given a process index $i$, $0 \leq i \leq m$, we define $h|i$ to be the word $(t_0|i)(t_1|i)\ldots(t_m|i)$. Intuitively, $h|i$ denotes the view of process $i$ in $h$.

A scheduler $\eta : (2^V)^+ \to \text{Trans}$ is a function that associates, with each history of a program $P$, a transition $\tau$ of some process of $P$ that is enabled in the last state of the history. We say that a scheduler $\eta$ 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 only depend on the view of the process; that is, the nondeterminism within a process is resolved uniquely by the view of the process. Note that the above condition does not constrain the scheduler from choosing transitions of different processes for $h$ and $h'$.

We say that $\eta$ 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 $i$, $\text{length}(h) = \text{length}(h')$, and $\text{last}(h)|i = \text{last}(h')|i$, 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 $\eta$, a program $P$ with $2^V$ as set of states can be thought of as an infinite state Markov chain which gives rise to the standard $\sigma$-algebra on $(2^V)^\omega$ and the standard probability measure $[24, 20]$. We shall henceforth denote this Markov chain as $M_{P, \eta}$. The set $(2^V)^\omega$ shall be called the set of paths of $M_{P, \eta}$.

Let $A$ be a Büchi automaton (or a Rabin automaton) with $2^V$ (the set of states of $P$) as it’s input alphabet. We say that $A$ accepts an infinite path $p \in (2^V)^\omega$ of $M_{P, \eta}$, if it accepts the infinite sequence $p$. Let $L(A)$ be the language accepted by $A$. Now $L(A)$ is a measurable set in the space of trajectories defined by $M_{P, \eta}$ and we call the measure of $L(A)$ to be the probability of acceptance of $M_{P, \eta}$. Given a rational number $x$ and $\triangleright \in \{>, =, \geq\}$, we shall write $P, \eta \models_{\triangleright \leq x} \text{Spec}$ if the measure of the set $L(A)$ is $\triangleright x$. The automaton $A$ will be henceforth called a specification automaton.

Predicate automaton. It is often useful to present the specification automaton $A$ succinctly in the following fashion. A predicate automaton $\text{Spec}$ 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 $\rightarrow \subseteq Q \times \text{BEXP}(V) \times Q$ is a finite set of predicate transitions. Given a predicate automaton $\text{Spec}$, we define a specification automaton $[\text{Spec}]$ as follows: $[\text{Spec}] = (2^V, Q, q_s, Q_f, \delta)$ where $(q, s, q') \in \delta$ iff there is a predicate transition $(q, \text{BEXP}, q') \in \rightarrow$ such that $s$ satisfies $\text{BEXP}$. Please note given any specification (Büchi) automaton $A$, there is a predicate automaton $\text{Spec}$ such that $[\text{Spec}] = A$. Furthermore, we will say that $\text{Spec}$ is a deterministic predicate automaton (respectively safety) if $[\text{Spec}]$ is a deterministic automaton (respectively safety automaton). Whenever convenient, we will often confuse $\text{Spec}$ with $[\text{Spec}]$. Given any history $h \in \Sigma^*$ let $\text{Spec}(h)$ be the state that $[\text{Spec}]$ 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 final states, we use a set of Rabin pairs $(B_1, G_1), \ldots (B_n, G_n)$ where $B_i, G_i$ are subsets of the states of the predicate Rabin automaton. As in the case of predicate automaton, a predicate Rabin automaton gives rise to a Rabin automaton.
Table summarizing the 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 $\omega$-EXPTIME-complete if all but one processes of P are deterministic (purely probabilistic). For entries with superscript c, the problem becomes $2\omega$-EXPTIME-decidable if processes are mutually exclusive and specification Spec is on shared state.

Verification. Given a rational number $x$ and $\triangleright \in \{\geq, =, >\}$, we will write $P \models_{=x} \text{Spec}$ if for every scheduler $\eta$, we have that $P, \eta \models_{=x} [\text{Spec}]$. Similarly, we write $P \models_{\triangleright} \text{Spec}$ ($P \models_{\leq} \text{Spec}$ respectively) if for every view consistent scheduler $\eta$ (locally Markovian scheduler respectively), $P, \eta \models_{\triangleright} [\text{Spec}]$. Thus, the verification problem we consider is one where given $P$, Spec, rational number $x \in [0, 1]$, and $\triangleright \in \{\geq, =, >\}$ as input, we want to determine if $P \models_{\triangleright,x} \text{Spec}$ (or $P \models_{\leq,x} \text{Spec}$ or $P \models_{\geq,x} \text{Spec}$). 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 and formally prove the results. The missing proofs as well as all the proofs/results of all schedulers can be found in the Appendix.

### 3.1 Undecidability

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

**Theorem 1.** Given a program $P$ and a predicate automaton $\text{Spec}$, the following problems are undecidable.

(a) Determining if $P \models_{=1} ^{vc} \text{Spec}$ and $P \models_{=1} ^{lm} \text{Spec}$.
(b) Determining if $P \models_{=0} ^{vc} \text{Spec}$ and $P \models_{>0} ^{lm} \text{Spec}$, even when Spec is a deterministic specification.
(c) Given a rational $x \in (0, 1)$ and $\triangleright \in \{\geq, =, >\}$, determining if $P \models_{\triangleright,x} ^{vc} \text{Spec}$ and $P \models_{\triangleright,x} ^{lm} \text{Spec}$, even when Spec is a safety specification.

The above undecidability results continue to hold even if $P$ 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 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 a given
FPM accepts every input with probability $\triangleright x$. Due to lack of space the details have been moved to
the Appendix.

For a program $P$, the observations in Theorem 1, leave open the decidability of the following
questions.

(a) if $\text{Spec}$ is a safety specification, check if $P \models^c_{\leq 1} \text{Spec}$ (or check if $P \models^m_{\leq 1} \text{Spec}$)?
(b) if $\text{Spec}$ is a safety specification, check if $P \models^c_{> 0} \text{Spec}$ (or check if $P \models^m_{> 0} \text{Spec}$)?
(c) if $\text{Spec}$ is a deterministic specification, check if $P \models^c_{= 1} \text{Spec}$ (or check if $P \models^m_{= 1} \text{Spec}$)?

We address these questions in the forthcoming sections. The decidability of (a) is shown in Theo-
rem 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 pro-
grams.

**Remarks:** Distributed schedulers, introduced in [17] and further studied in [15, 16], are very similar
to view consistent schedulers that we consider here. However, there is one important, subtle differ-
ence 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 [17, 15, 16]. Moreover, in [15, 16], the problem of checking if
$P \models_{> 0} \text{Spec}$ for safety specifications $\text{Spec}$ under all distributed schedulers is shown to be undecid-
able; 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 $C$, is 1. We, in fact, show that for any of the
case of three classes of schedulers that we consider, this problem is $PSPACE$-complete; the proofs are in the
Appendix.

**Proposition 2.** Given a program $P$ and a safety specification $\text{Spec}$, the following problems are
$PSPACE$-complete: determining if $P \models_{= 1} \text{Spec}$, if $P \models^c_{= 1} \text{Spec}$ and if $P \models^m_{= 1} \text{Spec}$.

We now establish that the problems of determining if $P \models^m_{> 0} \text{Spec}$ when $\text{Spec}$ is a safety
specification, and of determining if $P \models^m_{= 1} \text{Spec}$, when $\text{Spec}$ is deterministic, are $EXPSPACE$-
complete. We begin by defining a special class of locally Markovian schedulers that we call $\text{Spec}$-
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 $[\text{Spec}]$ after $h$
(namely, $\text{Spec}(h)$) is the same as $\text{Spec}(h')$.

**Definition 3.** Let $P$ be a program with $n$ processes with $\Sigma$ as the set of states of $P$ and $\text{Trans}$ as
the set of transitions of $P$. Let $\text{Spec}$ be a deterministic specification with $Q$ as the set of states. We say
that a locally Markovian scheduler $\eta : \Sigma^* \rightarrow \text{Trans}$ is $\text{Spec}$-determined if for any pair of histories
$h, h' \in \Sigma^*$ such that $\text{length}(h) = \text{length}(h')$, $\text{Spec}(h) = \text{Spec}(h')$ and $\text{last}(h) = \text{last}(h')$, 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 that is proved in the Appendix.

**Proposition 4.** For any program \(P\) and safety specification Spec, \(P \models ^{lm}_{=1}\) Spec iff for any Spec-determined locally Markovian scheduler \(\eta\); \(P, \eta \models ^{lm}_{>0}\) Spec. For deterministic specification Spec, \(P \models ^{lm}_{=1}\) Spec iff for all Spec-determined locally Markovian schedulers \(\eta\); \(P, \eta \models ^{lm}_{>0}\) Spec.

We need one more definition.

**Definition 5.** Let \(P\) 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 \(g : Q \times 2^V \rightarrow Trans\) 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 \(P_i\) and \(s_1[i] = s_2[i]\). The set of Spec-determined and locally consistent functions of program \(P\) shall be denoted as \(Loc(P, \text{Spec})\).

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 \(P\) and \((Loc(P, \text{Spec}))^\omega\), the set of infinite sequences over \((Loc(P, \text{Spec}))\). We call this function \(\text{Loc}_{P, \text{Spec}}\). The key technical idea exploited in our model checking algorithm is the following. Given a program \(P\) and a specification Spec, one can construct a PBA \(B\) that accepts a word \(\text{Loc}_{P, \text{Spec}}(\eta)\) with the same probability as the computation of \(P\) under scheduler \(\eta\) satisfies Spec; this stated and proved next.

**Lemma 6.** Given a program \(P\) and a deterministic specification Spec let \(\Delta = \text{Loc}(P, \text{Spec})\), the set of Spec-determined locally consistent functions. There is a PBA \(B\) on input alphabet \(\Delta\) such that the following hold–

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

**Proof.** Let \(P\) be a program with \(n\) processes, \(V\) be the set of variables of the program \(P\), \(s_0\) the initial state of \(P\) and \(Trans\) the set of transitions of \(P\). Let \(Q\) be the set of states of Spec, \(q_0\) the initial state of Spec and \(Q_f\) be the set of final states of Spec.

The PBA \(B\) is constructed as follows. The set of states of \(B\) is \(Q \times \Sigma\). The initial state of \(B\) is \((q_0, s_0)\). The set of final states of \(B\) will be \(Q_f \times 2^V\). The transition function is defined as follows. On input \(g \in \text{Loc}(P, \text{Spec})\), the state \((q, s)\) transits to \((q', s')\) with probability \(p\) iff the transition \(g(q, s)\) in the state \(s\) leads to \(s'\) with probability \(p\) and there is a transition \((q, Bexp, q')\) of Spec such that \(s\) satisfies \(Bexp\).

It is easy to see that \(B\) satisfies the first two conditions of the Lemma.

Now, note that if Spec is a safety specification with \(q_r\) as the reject state, we can modify \(B\) to obtain a FPM that satisfies the first two conditions as follows. The set of states of the FPM will be \((Q \setminus \{q_r\}) \times 2^V \cup \{q_{new}\}\) where \(q_r\) is the (unique) reject state of Spec and \(q_{new}\) is a new state which will be the reject state of the monitor. The state \((q_s, s_0)\) continues to be the initial state. The transition function is modified as follows. On input \(g\), the probability of transitioning from \((q, s)\) to \(q_{new}\) is taken to be 1 iff there is a transition \((q, Bexp, q_r)\) of Spec such that \(s\) satisfies \(Bexp\),...
otherwise it is taken to be 0. The probability of transitioning from state \((q, s) \in (Q \setminus \{q_f\}) \times 2^V\) to 
\((q', s') \in (Q \setminus \{q_f\}) \times 2^V\) continues to remain the same as in \(B\). The probability of transitioning from \(q_{\text{new}}\) to \(q_{\text{new}}\) is 1 on any input \(g\).

**Theorem 7.** Given a program \(P\) and a deterministic specification \(\text{Spec}\) the problem of determining if \(P \models^{\text{lm}}_{=1} \text{Spec}\) is \(\text{EXPSPACE}\)-complete. If \(\text{Spec}\) is a safety specification, then the problem of determining if \(P \models^{\text{lm}}_{>0} \text{Spec}\) is also \(\text{EXPSPACE}\)-complete.

We had shown in [7, 8] that given a PBA \(B\), the problem of checking whether all words are accepted with probability 1 is in \(\text{PSPACE}\). We had also shown in [7] that given an FPM \(M\), the problem of checking whether all words are accepted with probability \(> 0\) is in \(\text{PSPACE}\). In light of Lemma 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 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 \(\text{EXPSPACE}\). The inclusion in \(\text{EXPSPACE}\) is achieved by a careful examination of algorithms given in [7, 8] and running the algorithm without explicitly constructing the probabilistic Büchi automata.

### 3.3 Decidability results for view consistent semantics

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

**Theorem 8.** Given a program \(P\) and a deterministic specification \(\text{Spec}\) such that all processes of \(P\) except one are deterministic, the problem of checking if \(P \models^{\text{vc}}_{=1} \text{Spec}\) is \(2\text{-EXPTIME}\)-complete. Given a safety specification \(\text{Spec}\), the problem of checking if \(P \models^{\text{vc}}_{>0} \text{Spec}\) is also \(2\text{-EXPTIME}\)-complete.

The main idea behind the proof of \(2\text{-EXPTIME}\) membership is to reduce it to model checking POMDPs. It is proved in Lemma 13 (in the Appendix) that for any program \(P\) (of the kind in the theorem) and specification \(\text{Spec}\), there is a POMDP \(M\) such that \(P \models^{\text{vc}}_{=1} \text{Spec}\) (or \(P \models^{\text{vc}}_{>0} \text{Spec}\)) iff under every observation based scheduler, the measure of paths of \(M\) that visit a certain set of states infinitely often is 1 (or \(> 0\)). Details are in the Appendix.

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 \(P\) are **mutually exclusive** if in every state the transitions of only one process are enabled. A deterministic specification \(\text{Spec}\) is said to be **on the shared state** if for any two histories \(h_1\) and \(h_2\) of equal length such that the global variables take the same value in both \(h_1\) and \(h_2\), \(\text{Spec}(h_1) = \text{Spec}(h_2)\).
Theorem 9. Let \( P \) be a program where the processes communicate through global variables that is mutually exclusive. The problems of checking \( P \models_{\leq 1} \text{Spec} \) when \( \text{Spec} \) is a deterministic specification on the shared state, and of checking \( P \models_{>0} \text{Spec} \) when \( \text{Spec} \) is a safety specification are in 2-EXPTIME.

The proof of the above theorem relies on showing that if there is a scheduler \( \eta \) such that \( P, \eta \models_{<1} \text{Spec} (P, \eta \models_{=0} \text{Spec}) \) for a deterministic specification on the shared state (safety specification) then there is a “periodic” scheduler \( \eta' \) that witnesses the same fact. Finally, the model checking algorithm searches for such a periodic scheduler by reducing it to \( \mu \)-calculus model checking on a finite (doubly exponentially sized) graph.

4 Complexity results for all-scheduler semantics

We shall now discuss the complexity results for the verification problems under MDP semantics. We had already shown in Proposition 2 that checking whether a program \( P \) satisfies a safety specification under all schedulers with probability 1 is PSPACE-complete. We now establish the remaining results from Figure 1.

Theorem 10. Given a program \( P \), a deterministic specification \( \text{Spec} \) and a rational number \( x \in [0, 1] \), the following hold.

(a) the problem of determining if \( P \models_{>x} \text{Spec} \) is in EXPTIME. The same result also holds for the problem of determining if \( P \models_{\geq x} \text{Spec} \).

(b) If \( y \in [0, 1) \), then the problem of determining if \( P \models_{>y} \text{Spec} \) is EXPTIME-hard. This result holds even for the case when \( \text{Spec} \) is a safety specification.

(c) If \( y \in (0, 1) \), the problem of determining if \( P \models_{\geq y} \text{Spec} \) is EXPTIME-hard. Furthermore, if \( y \in (0, 1) \) and \( \text{Spec} \) is a safety automaton, the problem of determining if \( P \models_{\geq y} \text{Spec} \) is also EXPTIME-hard.

5 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 [12] 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 \( P_{\text{new}} \) 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-\textsc{nexptime}-hard based on observations made in [12]; 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.

References

A Büchi Automata. A Büchi automaton $A$ is a 5-tuple $(\Delta, Q, \delta, q_0, F)$ where $\Delta$ is the input alphabet, $Q$ is its set of states, $\delta \subseteq Q \times \Delta \times Q$ is its transition relation, $q_0$ is its initial state and $F \subseteq Q$ is its set of accepting states. A run on an input $\sigma$ is a $\omega$-sequence of states that $A$ goes through on that input starting from initial state. A run is said to be accepting if some accepting state occurs infinitely often in it. $A$ is said to accept the input $\sigma$ if there is an accepting run on it. We let $L(A)$ denote the set of input sequences accepted by $A$. We say that a language $L \subseteq \Delta^\omega$ is $\omega$-regular iff there is Büchi automata $A$ such that $L = L(A)$.

We say that $A$ is safety automaton if it is deterministic and there is a unique state $q_{\text{reject}}$ such that, for every $a \in \Delta$, $(q_{\text{reject}}, a, q_{\text{reject}}) \in \delta$ and $F = Q \setminus \{q_{\text{reject}}\}$. If $A$ is a safety automaton then $L(A)$ is a safety property.

Rabin Automata. Rabin Automata are like Büchi automata except that instead of the set of final states, we have a set of pairs of subsets of states. Formally, a Rabin automata $A$ is a 5-tuple $(\Delta, Q, \delta, q_0, RP)$ where $\Delta, Q, \delta, q_0$ are defined as in the case of Büchi automata. The set $RP \subseteq 2^Q \times 2^Q$ is a finite set of Rabin pairs. A run on an input is defined as before. A run is said to be accepting if there is some Rabin pair $(B, G) \in RP$ such that some state in $G$ occurs infinitely often in the run and no state in $B$ occurs infinitely often in the run. As in the case of Büchi automata, we can thus define the language of a Rabin automata as the set of infinite words which have an accepting run.

B Partially Observable Markov Decision Processes POMDP

A POMDP $M$ is a 5-tuple $(Q, Act, q_0, \delta, \equiv)$ where $Q$ is a finite set of states, Act is a finite set of actions, $q_0$ is the initial state, $\delta \subseteq Q \times Act \times Q \rightarrow [0,1]$ is the transition function such that for each $q \in Q$ and $a \in Act$, $\sum_{q' \in Q} \delta(q, a, q') = 1$ and $\equiv$ is an equivalence relation on $Q$. We assume that $\delta(q, a, q')$ is a rational number for all $q, q' \in Q$ and $a \in Act$. A scheduler $S$ on $M$ is a function $S : Q^+ \rightarrow Act$. For POMDPs, we are usually interested in observation-based schedulers. A scheduler $S$ is said to observation-based if for any two words $h = q_1 \ldots q_n \in Q^+$ and $h' = q'_1 \ldots q'_m \in Q^+$ such
that $m = n$ and $q_i \equiv q'_i$ for each $1 \leq i \leq n$, we have that $S(h) = S(h')$. In presence of a scheduler $S$, $\mathcal{M}$ can be thought of as an infinite state Markov chain which gives rise to the standard $\sigma$-algebra on $Q^\omega$ and the standard probability measure \cite{24, 20}. An element of $Q^\omega$ is said to be a path. Given a set $Q_f \subseteq Q$ the set of all paths that visit $Q_f$ infinitely often is measurable and the measure of this set is said to be the probability of visiting $Q_f$ infinitely often.

### C Motivating Examples

In this section, we present examples of concurrent processes for which considering all possible schedulers yields incorrect verification results. The examples motivate the need for considering view consistent schedulers instead.

**Example 1.** Many researchers \cite{11, 6, 14, 10} have observed that in analyzing security protocols, certain schedulers can leak “information” and thereby compromise security, even though the protocols themselves can be argued to be correct. Examples of anonymity and fair exchange protocols that exhibit such an anomaly can be found in \cite{14, 10}. Here we present another simple example very similar to the one presented in \cite{17}. Consider a situation where Alice picks either 0 or 1 by tossing a fair coin, and then sends this bit to Bob over a secure channel. After sharing this bit Alice and Bob update a public variable to 1 to indicate that they have exchanged a bit. Now Charlie, on learning about the exchange, nondeterministically guesses the value of the bit that Alice and Bob shared. Given that the exchange between Alice and Bob took place over a secure channel, it must be the case that the probability that Charlie’s guess right is at most $\frac{1}{2}$. However, if one models this as an MDP and considers all possible schedulers, then there are certain “all knowing” schedulers under which the probability that Charlie guesses right can be as high as 1. In contrast, under any view consistent scheduler, Charlie has only a 50% chance of getting the right answer.

**Example 2.** We recall an example from \cite{12}. Consider a telephone network to which two users $A$ and $B$ are connected. $A$ nondeterministically decides whether she should call $B$ and the call is successful provided the network is not overloaded. The number of connections the network can support, is itself modelled purely probabilistically. Alfaro shows formally that no matter how many connections are available on the average between $A$ and $B$, the long run fraction of the number of successful calls is in the worst case 0! Informally, this happens under an adversary that schedules calls from $A$ at exactly the times when the network is overloaded.

**Example 3.** Consider the following program that has two processes $P_0$ and $P_1$. The programs are described in a guarded command language notation, where every time a process is scheduled, a command whose guard is true is executed.

$P_0$:

\begin{align*}
flag &= 0 \rightarrow flag := 1, \ 0.5 : p := 0, \ 0.5 : p := 1 \\
flag &= 2 \land p = x \rightarrow flag := 3 \\
flag &= 2 \land p \neq x \rightarrow flag := 0 \\
flag &= 3 \rightarrow flag := 3
\end{align*}

$P_1$: 


flag = 1 \rightarrow flag := 2, 0.5 : y := 0, 0.5 : y := 1, x := y

Variable \( p \) is private to \( P_0 \), and \( y \) is private to \( P_1 \). On the other hand, \( flag \) and \( x \) are shared between the processes. Initially all variables have value 0. The computation proceeds in “rounds”, with each process taking alternating steps within a round, using the variable \( flag \). First, \( P_0 \) randomly chooses a value for \( p \), and then \( P_1 \) picks a value for \( y \). \( P_1 \) communicates the value of \( y \) through the variable \( x \) to \( P_0 \). If \( P_0 \) finds that \( P_1 \)’s guess is the same as hers, then \( flag \) is set to 3 and the computation is “over”. On the other hand, if the guesses are different, a new round begins where the processes guess again. In every round, there is a nonzero probability that \( P_0 \) and \( P_1 \)’s guesses will be the same, and so the probability that \( flag \) is eventually set to 3 holds with probability 1.

Suppose we abstract process \( P_1 \) by ignoring the private variable \( y \). The abstracted process \( P_1 \) will now be as follows.

\[
P_1^R :=
\begin{align*}
  flag = 1 & \rightarrow flag := 2, x := 0 \\
  flag = 1 & \rightarrow flag := 2, x := 1
\end{align*}
\]

Thus, the abstracted process \( P_1^R \) sets the value of \( x \) nondeterministically. Now, for the concurrent process \( P_0 || P_1^R \) there is an “all knowing” scheduler that can always choose to set the value of \( x \) to be different from the one picked by \( P_0 \), and thus, under such a scheduler \( P_0 || P_1^R \) never sets \( flag \) to 3. However, this is unreasonable. Eventhough \( P_1^R \) sets the value of \( x \) nondeterministically, it cannot set the value based on \( p \) because \( p \) is a private variable. Under view consistent schedulers, \( P_0 || P_1^R \) behaves like \( P_0 || P_1 \).

### D Proof of Theorem 1

**Part (a).** Baier et. al. [2] show that the problem of deciding if a given PRA \( R \) accepts all input strings with nonzero probability is undecidable. More specifically, given a PRA \( R \) over an alphabet \( \Sigma \), determining if \( L_{=1}(R) = \Sigma^\omega \), is undecidable. We will reduce this problem to checking whether \( P \models_{\text{rc}} \text{Spec} \) and/or \( P \models_{\text{lm}} \text{Spec} \).

Let \( R = (Q, q_0, \Sigma, \delta) \) be a PRA over an alphabet \( \Sigma \) and let the Rabin pairs \( R \) be \((B_1, G_1), \ldots, (B_k, G_k)\). We first construct a program \( P \) from the \( Q, q_0, \delta \) as follows. \( P \) has two processes \( P_1, P_2 \). Intuitively \( P_1 \) acts like the automaton \( R \), while \( P_2 \) generates the input symbols to \( P_1 \). The processes take turns. We number the execution steps of \( P \) starting from 1. Process \( P_2 \) is enabled in all odd steps and \( P_1 \) is enabled in all even steps. This alternate enabling of the processes is achieved using a shared variable \( turn \) which takes value 1, 2 and which is initially set to 2. In addition to \( turn \), \( P_1 \) and \( P_2 \) share another variable \( in\_symbol \), which is used to communicate the input symbol generated by \( P_2 \). Thus, when \( turn = 2 \), \( P_2 \) nondeterministically sets \( in\_symbol \) to some value in \( \Sigma \) and also sets \( turn \) to 1.

Process \( P_1 \) has a local variable \( aut\_state \) in which it stores the current state of \( R \). This variable is initially set to \( q_0 \). Corresponding to each pair \((q, a) \in Q \times \Sigma \), process \( P_1 \) has a transition that is enabled when \( turn = 1, aut\_state = q \) and \( in\_symbol = a \); this transition sets \( turn \) to 2 and sets \( aut\_state \) to \( q' \) with probability \( \delta(q, a, q') \), for each \( q' \in \Sigma \). These are the only transitions of the processes in \( P \). Note that \( aut\_state \) is local to \( P_1 \) and is not visible to \( P_2 \), while \( turn \) and \( in\_symbol \) are shared variables. It should be easy to see that in every odd step, starting from 1, \( P_2 \) generates an input symbol, and \( P_1 \) simulates \( R \) on this input symbol in the succeeding step. Moreover, any scheduler is locally Markovian iff it is view consistent.
Now consider a locally Markovian scheduler (or view consistent) \( \eta \) for the program \( P \). Let \( M_{P, \eta} \) be the Markov chain denoting the computation of \( P \) with respect to \( \eta \). Let \( s_0 \) be the initial state of \( P \). Now consider any two histories \( h, h' \) of same length reachable from \( s_0 \) in \( M_{P, \eta} \). It should be easy to see that the sequence of values taken by the variable \( \text{in\_symbol} \) in both these histories is same. Similarly, if \( h_0, ..., h_i, ... \) is an infinite path, i.e., sequence of histories in \( M_{P, \eta} \), then the limit of the sequences of values taken by \( \text{in\_symbol} \) in these histories denotes an input string from \( \Sigma^\omega \), and this infinite sequence of input symbols is same for all infinite paths in \( M_{P, \eta} \). Let this input string be given by \( a_\eta \). Now, let \( \text{Spec} \) be a specification automaton with the set of variables being the set of variables of \( P \) such that \( \text{Spec} \) accepts \( \bigcup_{1 \leq k \leq \ell} L_k \) where \( L_k \) is the sequence of states such that the variable \( \text{aut\_state} \) takes value in the set \( B_k \) finitely many times and in \( G_k \) infinitely many times. It should be easy to see that the probability of acceptance of \( M_{P, \eta} \) by \( \text{Spec} \) is equal to the probability of acceptance of \( a_\eta \) by \( \mathcal{R} \).

From the above observation, we see that every computation of \( P \) generated by a locally Markovian (or view consistent) scheduler is accepted by \( \text{Spec} \) with probability 1 iff \( \mathcal{R} \) accepts every input with probability 1.

**Part (b).** We prove this result for the class of locally Markovian schedulers and the result for view consistent schedulers follows automatically from the same proof. In earlier works [2], it has been shown that the problem of deciding if a given PBA \( B \) accepts all input strings with nonzero probability is shown to be undecidable. More specifically, given a PBA \( B \) over an alphabet \( \Sigma \), the problem of determining if \( \mathcal{L}_{>0}(B) = \Sigma^\omega \), was shown to be undecidable. We show that this problem is reducible to the problem of determining if all computations of a given program \( P \), generated by a locally Markovian scheduler, is accepted by a given deterministic specification \( \text{Spec} \) with nonzero probability.

Let \( B = (Q, q_s, Q_f, \delta) \) be a PBA over an alphabet \( \Sigma \). We first construct a program \( P \) from \( Q, q_s \) and \( \delta \) in the same fashion as we constructed \( P \) in the proof of part (a). The specification \( \text{Spec} \) is taken to be a deterministic specification which accepts those sequences of states of \( P \) in which the variable \( \text{aut\_state} \) takes a value in \( Q_f \) infinitely often. The result then follows from the observation that every computation of \( P \) generated by a locally Markovian (or view consistent) scheduler is accepted by \( \text{Spec} \) with nonzero probability iff \( B \) accepts every input with nonzero probability.

**Part (c).** In our previous work [7], we had shown that, given \( \triangleright \in \{ >, \geq, \leq \} \) a FPM \( \mathcal{M} \) and rational number \( x \in (0, 1) \), the problem of determining if \( \mathcal{M} \) accepts all inputs with probability \( \triangleright x \) was shown to be undecidable. We show that this problem is reducible to the problem of determining if all computations of a given program \( P \), generated by a locally Markovian scheduler, is accepted by a given specification \( \text{Spec} \) with probability \( \triangleright x \).

Let \( \mathcal{M} = (Q, q_s, Q_f, \delta) \) be a FPM over an alphabet \( \Sigma \) with \( q_f \) as the reject state. We first construct a program \( P \) from \( Q, q_s \) and \( \delta \) in the same fashion as we constructed \( P \) in the proof of part (a). The specification \( \text{Spec} \) is taken to be a safety specification which accepts those sequences of states of \( P \) in which the variable \( \text{aut\_state} \) never takes the value \( q_f \). The result then follows from the observation that every computation of \( P \) generated by a locally Markovian (or view consistent) scheduler is accepted by \( \text{Spec} \) with probability \( \triangleright x \) iff \( \mathcal{M} \) accepts every input probability \( \triangleright x \).
### E  Proof of Proposition 2

(Lower Bound). The lower bound follows from the fact that for a program $P$ consisting of a single process that has no probabilistic and no nondeterministic transitions, the problem of determining if its single computation is accepted by a safety automaton, is $PSPACE$-hard.

(Upper Bound). We show the result for view consistent semantics. The case for locally Markovian semantics and for MDP semantics is similar. Assume that we are given a program $P$ over variables $V$ and a safety predicate automaton $Spec$ with $Q$ as the set of states of $Spec$ and $q_r$ as the reject state. It is easy to see that there is a view consistent scheduler such that the measure of computations generated by $Spec$ is accepted by $Spec$ with probability $< 1$ iff there is a $n \geq 1$, a sequence of program states $s_1, \ldots, s_n$ and a sequence, $q_1, \ldots, q_n$, of states of $Spec$, such that the following conditions hold:

- $s_1$ is the initial state of $P$;
- for each $1 \leq i < n$, there is a transition $tr_i$ of $P$ enabled in $s_i$ such that application of $tr_i$ to $s_i$ leads to state $s_{i+1}$ with nonzero probability;
- $q_1$ is the initial state of $Spec$ and $q_n$ is the reject state of $Spec$; and
- for each $1 \leq i < n$, there is a transition $(q_i, Bexp_i, q_{i+1})$ of $Spec$ such that $s_i$ satisfies $Bexp_i$.

The $PSPACE$ algorithm now guesses the states $s_i, q_i, tr_i, (q_i, Bexp_i, q_{i+1})$ one by one and checks that the above four conditions hold.

### F  Proof of Proposition 4

We begin by showing that $P \models_{\sim}^{lm} Spec$ iff for any Spec-determined locally Markovian scheduler $\eta$; $P, \eta \models_{\sim} Spec$. The ($\Rightarrow$) direction is straightforward since every Spec-determined locally Markovian scheduler is also locally Markovian. We show the ($\Leftarrow$) direction. For this it suffices to show that if there is a locally Markovian scheduler $\eta$ such that $P, \eta \models_{\sim} Spec$ then there is a Spec-determined locally Markovian scheduler $\eta'$ such that $P, \eta' \models_{\sim} Spec$. The construction of the scheduler $\eta'$ is along the lines of the proof of Lemma 6.11 in [7] and explained below.

Fix a locally Markovian scheduler $\eta$ such that $P, \eta \models_{\sim} Spec$. Note that the computation $\mathcal{M}_{P,\eta}$ is an infinite state Markov chain with $(2^V)^+$, the set of histories of $P$, as the set of states of $\mathcal{M}_{P,\eta}$. The transition probability of a history $h$ to another history $h'$ is $p$ if $h'$ is $hs$ and the probability of transitioning from $last(h)$ to $s$ upon applying the transition $\eta(h)$ is $p$. In all other cases the probability of transitioning from $h$ to $h'$ is 0. If $Q$ is the set of states of $Spec$ then consider the “product” Markov Chain- $Spec \times \mathcal{M}_{P,\eta}$ defined as follows. The set of states of $Spec \times \mathcal{M}_{P,\eta}$ is $Q \times (2^V)^+$. The probability of transitioning from $(q, h)$ to $(q', h')$ is $p$ if there is a transition from $q$ to $q'$ in $[Spec]$ on input $last(h)$ and the probability of transitioning from $h$ to $h'$ is $p$ in $\mathcal{M}_{P,\eta}$. The initial state of $Spec \times \mathcal{M}_{P,\eta}$ is $(q_0, s_0)$ where $q_0$ is the initial state of Spec and $s_0$ is the initial state of $\mathcal{M}_{P,\eta}$. If $q_r$ is the reject state of Spec then the probability of $P$ not satisfying $Spec$ is exactly the probability of visiting the set of states $\{q_r\} \times (2^V)^+$ in $Spec \times \mathcal{M}_{P,\eta}$. Since $P, \eta \models_{\sim} Spec$, the latter probability is 1.

Now for any natural number $n > 0$, let $H_n$ be the set of pairs $(q, h) \in Q \times (2^V)^n$ such that the probability of being in $(q, h)$ after exactly $n - 1$ steps in the Markov chain $Spec \times \mathcal{M}_{P,\eta}$ is $> 0$. Let $post_n \subseteq Q \times 2^V$ be the set $\{(q, s) \mid \exists (q, h) \in H_n \text{ s.t. } last(h) = s\}$. Since $Q \times 2^V$ is a finite set it follows that there is a set $Repeat \subseteq Q \times 2^V$ such that $Repeat = post_n$ are infinitely many $n$.

Let $n_0$ be the smallest $n$ such that $Repeat = post_n$. For each $(q, s) \in Repeat$ fix a history $h_{q, s} \in (2^V)^{n_0}$ such that $last(h_{q, s}) = s$ and the probability of being in $(q, h_{q, s})$ after exactly $n_0 - 1$
For any history \( n \) and let \( n \) steps in the Markov chain \( \text{Spec} \times \mathcal{M}_{P, \eta} \) is \( > 0 \) (note there could be more than one such history—we just fix one). Observe that since the probability of visiting the set of states \( \{q_t\} \times (2^V)^+ \) is \( 1 \) in \( \text{Spec} \times \mathcal{M}_{P, \eta} \) and \( \text{Repeat} = \text{post}_n \) for infinitely many \( n \), we have that for each \( h_{q,s} \) there is a natural number \( n_{q,s} > 0 \) such that the following hold:

- the probability of visiting \( \{q_t\} \times (2^V)^+ \) in \( \text{Spec} \times \mathcal{M}_{P, \eta} \) within \( n_{q,s} \) steps having started in \( h_{q,s} \) is \( > 0 \) and,
- if \((q', h')\) is reached in \( \text{Spec} \times \mathcal{M}_{P, \eta} \) after exactly \( n_{q,s} \) steps having started in \( h_{q,s} \) is \( > 0 \) then 
\((q', \text{last}(h')) \in \text{Repeat} \).

For each \((q, s) \in \text{Repeat} \), fix one natural number \( n_{q,s} > 0 \) that satisfies the above properties. And let \( n_1 \) be the maximum amongst the set \( \{n_{q,s} \mid (q, s) \in \text{Repeat}\} \). Now, we construct \( \eta' \) in two steps as follows.

First we construct a sequence of schedulers \( \eta'' \) as follows—\( \eta''(h) = \eta(h) \) for each history \( h \) of length \( < n_0 \). For histories of length \( n_0 + n_1 + n_2 \) where \( 0 \leq n_2 < n_1 \), \( \eta''((h_1)(h_2)h_3) = \eta(h_1h_3) \) where \( h_1 \in (2^V)^{n_0}, h_2 \in (2^V)^{n_1} \) and \( h_3 \in (2^V)^{n_2} \). Clearly, \( \eta'' \) is periodic and from the construction of \( n_0 \) and \( n_1 \), it follows that \( P, \eta'' \models 0 \text{ Spec} \). Furthermore, for each history \( h \) of length \( n_0 + n_1 \), if \( \text{Spec}(h) = q \) and measure of \( h \) in \( \mathcal{M}_{P, \eta''} \) is \( > 0 \) then \((q, h) \in \text{Repeat} \).

Now \( \eta'' \) is periodic and locally Markovian but may not be Spec-determined. This is because there might be histories \( h_1 \) and \( h_2 \) such that \( \text{length}(h_1) = \text{length}(h_2) \), \( \text{Spec}(h_1) = \text{Spec}(h_2) \) and \( \text{last}(h_1) = \text{last}(h_2) \) but the scheduler \( \eta'' \) picks transitions belonging to different processes. However, it is easy to see that one can construct a periodic and Spec-determined scheduler \( \eta' \) from \( \eta'' \) such that \( P, \eta' \models 0 \text{ Spec} \) as follows.

We first construct a sequence of schedulers \( \eta''_0, \ldots, \eta''_{n_0+n_1-1} \). Suppose we have constructed \( \eta''_0, \ldots, \eta''_{i-1} \). Then, we construct \( \eta''_i \) as follows. First, consider an equivalence relation \( \equiv \) on histories of length \( n_0 + n_1 - i \) defined as—\( h = h' \) if \( \text{Spec}(h) = \text{Spec}(h') \) and \( \text{last}(h) = \text{last}(h') \).

For any history \( h \) of length \( n_0 + n_1 - i \), we denote the equivalence class of \( h \) under \( \equiv \) by \( [h] \). Given an equivalence class \( [h] \), we let \( R([h]) = \{h' \in [h] \mid \text{probability of being in } h' \text{ after exactly } n_0 + n_1 - i - 1 \text{ steps in } \mathcal{M}_{P, \eta''_{i-1}} \} \) and \( G([h]) = \{h' \in [h] \mid \text{probability of visiting } q_t \times (2^V)^+ \in \mathcal{M}_{P, \eta''_{i-1}} \} \) within \( i \) steps having started in \( \text{Spec}(h'), h' \text{ is } > 0 \} \). Now we fix an element \( \text{rep}([h]) \in [h] \) such that \( \text{rep}([h]) \in R([h]) \cap G([h]) \) if \( R([h]) \cap G([h]) \neq \emptyset \) and \( \text{rep}([h]) \in R([h]) \cap G([h]) \neq \emptyset \).

Now for histories \( h \) of length \( n_0 + n_1 - i \), we let \( \eta(h) = \eta(\text{rep}([h])) \).

Now the desired \( \eta' \) is obtained as follows. \( \eta'(h) = \eta''_{n_0+n_1-1}(h) \) for each history \( h \) of length \( < n_0 \). For histories of length \( n_0 + kn_1 + n_2 \) where \( 0 \leq n_2 < n_1 \), \( \eta''((h_1)(h_2)h_3) = \eta''_{n_0+n_1-1}(h_1h_3) \) where \( h_1 \in (2^V)^{n_0}, h_2 \in (2^V)^{n_1} \) and \( h_3 \in (2^V)^{n_2} \).

## G Proof of Theorem 7

### Upper Bound: We first start by establishing that the problems are in EXPSPACE. We begin by introducing some notation.

#### Notation: Given a PBA \( B = (Q, q_s, Q_f, \delta) \) over a finite alphabet \( \Delta \) The transition function \( \delta \) of PBA \( B \) on input \( a \) can be seen as a square matrix \( \delta_a \) of order \( |Q| \) with the rows labeled by “current” state, columns labeled by “next state” and the entry \( \delta_a(q, q') \) equal to \( \delta(q, a, q') \). Given a word \( u = a_0a_1 \ldots a_n \in \Delta^* \), \( \delta_u \) is the matrix product \( \delta_{a_0} \delta_{a_1} \ldots \delta_{a_n} \). For an empty word \( \epsilon \in \Delta^* \) we take \( \delta_{\epsilon} \).
to be the identity matrix. Finally for any \( Q_0 \subseteq Q \), we say that \( \delta_u(q, Q_0) = \sum_{q' \in Q_0} \delta_u(q, q') \). Given a state \( q \in Q \) and a word \( u \in \Delta^+ \), \( \text{post}(q, u) = \{ q' \mid \delta_u(q, q') > 0 \} \). Given a set \( Q_1 \subseteq Q \) and word \( u \in \Delta^+ \), \( \text{post}(Q_1, u) = \{ q' \mid \exists q'' \in Q_1, \delta_u(q, q'') > 0 \} \).

We also need the following proposition which follows from the results in [8, 7].

**Proposition 11.** Given a PBA, \( B = (Q, q_s, Q_f, \delta) \) on alphabet \( \Delta \), \( L_{\leq 1}(B) \neq \Delta^\omega \) iff there is a set \( Q_1 \subseteq Q \setminus Q_f \) and finite words \( v, u \in \Delta^\omega \) such that– a) for any prefix \( v[0 : i] \) of \( v \), \( \text{post}(Q_1, v[0 : i]) \cap Q_f = \emptyset \), b) \( \text{post}(Q_1, v) \subseteq Q_1 \) and c) \( \text{post}(q_s, u) \cap Q_1 \neq \emptyset \).

Consider first the problem of checking whether \( \mathcal{P} \) satisfies a deterministic specification \( \text{Spec} \) with probability 1. Let \( \mathcal{P} \) be program consisting of \( n \) processes with \( V \) as the set of variables, \( s_0 \) as the initial state and \( \text{trans} \) as the set of transitions. Let \( Q \) be the set of states of \( \mathcal{P} \) with \( q_s \) and \( Q_f \) as the set of final states of \( \text{Spec} \). Consider the PBA \( B = (Q \times 2^V, (q_s, s_0), Q_f \times 2^V, \delta) \) on input \( \Delta = \text{Loc}(\mathcal{P}, \text{Spec}) \) (the set of locally view consistent functions) constructed as in the proof of Lemma 6. Thanks to Proposition 4 and Lemma 6 there is a locally Markovian scheduler \( S \) of \( \mathcal{P} \) such that the computation generated by \( S \) is accepted by probability \(< 1 \) iff \( L_{\leq 1}(B) \) is not universal. The \text{EXPSPACE} \text{ algorithm will indeed check for nonuniversality of } L_{\leq 1}(B) \text{ without actually constructing } B. \)

By Proposition 11, we know that there is some word in \( \Delta^\omega \) is accepted by \( B \) with probability \(< 1 \) iff there is a set of states \( \text{GoodStates} \subseteq (Q \times 2^V) \setminus (Q_f \times 2^V) \), two finite words \( v = g_0g_1 \ldots g_u, u \in \Delta^+ \) such that following conditions hold–

1. for any prefix \( v_i = g_0g_1 \ldots g_i \) of \( v \), \( \text{post}(\text{GoodStates}, v_i) \cap (Q_f \times 2^V) = \emptyset \),
2. \( \text{post}(\text{GoodStates}, v) \subseteq \text{GoodStates} \),
3. There is some state \( (q, s) \in \text{GoodStates} \) such that \( (q, s) \in \text{post}((q_s, s_0), u) \).

The \text{EXPSPACE} \text{ algorithm now first guesses the set } \text{GoodStates} \text{; then it guesses the word } v \text{ by guessing the } \text{Spec}-\text{determined locally consistent functions } g_0, g_1, \ldots \text{ one-by-one, checking at each step that the guess is indeed a locally consistent function. After guessing } g_i+1 \text{ the guess } g_i \text{ is deleted and the set } \text{post}(\text{GoodStates}, v_i+1) \text{ is computed. Note that the set } \text{post}(\text{GoodStates}, v_i+1) \text{ only depends on } \text{post}(\text{GoodStates}, v_i) \text{ and } g_i+1 \text{ and can be computed again in space exponential in the size of the input. After computing } \text{post}(\text{GoodStates}, v_i+1), \text{ we check if } \text{post}(\text{GoodStates}, v_i+1) \cap (Q_f \times 2^V) = \emptyset \). \text{ If it is the case, } \text{post}(\text{GoodStates}, v_i) \text{ is deleted and then we guess if the end of } v \text{ is reached. If we guess that the end of } v \text{ is not reached then we continue with guessing } g_{i+2}. \)

Otherwise, we check if \( \text{post}(\text{GoodStates}, v) \subseteq \text{GoodStates} \). If it indeed is the case, we then start guessing \( u \) one-by-one, computing \( \text{post}((q_s, s_0), u[0, i]) \) (just as we computed \( \text{post}((\text{GoodStates}, v_i)) \) above) along the way. When we guess that the end of \( u \) is reached, then we check if \( \text{post}((q_s, s_0), u) \) contains some set in \( \text{GoodStates} \). If it is the case we can declare that there is a locally Markovian scheduler \( S \) such that the computation generated by \( S \) is accepted by \( \text{Spec} \) by probability \(< 1 \).

The case of checking whether a program satisfies a safety specification with nonzero probability can similarly be obtained by appealing to the following proposition proved in [7].

**Proposition 12.** Given a FPM, \( M = (Q, q_s, Q_f, \delta) \) on alphabet \( \Delta \) with reject state \( q_r \), \( L_{< 0}(B) \neq \Delta^\omega \) iff there is a set \( Q_1 \subseteq Q \) and finite words \( v, u \in \Delta^\omega \) such that, a) \( \text{post}(q_s, u) = Q_1 \), b) \( \text{post}(Q_1, v) = Q_1 \) and c) for each \( q \in Q_1, q_r \in \text{post}(q, v) \).

**Lower Bound:** We are now ready to establish the \text{EXPSPACE}-hardness of the problems. Once again we begin by fixing some notation.
Let us fix some language \( L \in \text{EXPSPACE} \) and a single tape deterministic Turing machine \( M \) that witnesses \( L \)'s membership in \( \text{EXPSPACE} \). Let \( p(n) \) be a polynomial such that \( M \) uses at most \( 2^{p(n)} \) space on any input of length \( n \). We will assume that \( M \) halts on all inputs; it accepts by halting in state \( q_a \) and rejects by halting in state \( q_r \). Let \( M \) be given by the tuple \((Q, \Sigma, \Gamma, B, \delta, q_0, q_a, q_r)\). Here \( Q \) is the set of states of the finite control of \( M \); \( \Sigma, \Gamma \) are the input and tape alphabets and \( \Sigma \subseteq \Gamma \) and the blank symbol \( B \) is in \( \Gamma \setminus \Sigma \); \( \delta : Q \times \Gamma \rightarrow \Gamma \times Q \times \{+1, -1\} \); \( q_0 \) is the initial state and \( q_a, q_r \) are the accepting and rejecting states, respectively. If \( \delta(q, a) = (a', q', d) \) it means that when \( M \) is in state \( q \), scanning a cell containing the symbol \( a \), then \( M \) writes value \( a' \) in the current cell, changes to state \( q' \). The value of \( d \) denotes the direction in which the head moves. Here \( -1, +1 \) denote left, right motions, respectively.

Let us fix an input \( \sigma \) for \( M \) of length \( n \) and take \( m = 2^{p(n)} \). Let \( \Phi' = \Gamma \times Q \) and \( \Phi = \Phi' \cup \Gamma \); fix \( c = |\Phi| \). Recall that a configuration of \( M \) appearing during a computation on \( \sigma \) can be described by a string of length \( m \) over \( \Phi \), that has exactly one symbol from \( \Phi' \); the symbol from \( \Phi' \) indicates the position of the head and records the control state of \( M \). Observe that \( k = c^m \) is a bound on the number of configurations of \( M \), and we can, without loss of generality, assume that \( M \) halts within \( k \) steps on \( \sigma \). The computation of \( M \) on \( \sigma \) can be described by a sequence of configurations of length at most \( k \), such that the control state in the last configuration is either \( q_a \) or \( q_r \). An element of the set \( \Sigma_n = \{1, \ldots, m\} \times \Gamma \times Q \) can be seen as describing the current head position, the current symbol being read and the current state of \( M \). Thus, every valid computation corresponds to a unique sequence of symbols from \( \Sigma_n \). Conversely, given a sequence of symbols from \( \Sigma_n \), it is easy to see when it corresponds to a valid computation of \( M \) on \( \sigma \). Thus, we will find it convenient to think about sequences over \( \Sigma_n \) as computations of \( M \).

Having set up a lot of the notation that we will use in the proofs, we are ready to present the lower bounds results.

We first show that given a program \( \mathcal{P} \) and a safety specification \( \text{Spec} \) the problem of determining if \( \mathcal{P} \models_{\text{true}}^m \text{Spec} \) is \( \text{EXPSPACE} \)-hard. Let us a fix a language \( L \in \text{EXPSPACE} \), a Turing machine \( M \) computing \( L \), and input \( \sigma \) of length \( n \). To prove \( \text{EXPSPACE} \)-hardness, we will construct a concurrent program \( \mathcal{P}_{\sigma} \) and a safety specification \( \text{Spec} \) such that \( \mathcal{P}_{\sigma} \models_{\text{true}}^m \text{Spec} \) iff \( \sigma \) is accepted by \( M \). The intuitions behind the construction are as follows. \( \mathcal{P}_{\sigma} \) will consist of two processes \( \mathcal{P}_1 \) and \( \mathcal{P}_2 \); \( \mathcal{P}_2 \) will guess a sequence of symbols from \( \Sigma_n \), and \( \mathcal{P}_1 \) will “check” whether the sequence corresponds to the head position, symbol scanned, and state in a valid computation of \( M \) on \( \sigma \). Initially, \( \mathcal{P}_1 \) will randomly pick a cell position (between 1 and \( m \)) to keep track of, and record the contents of that cell in the initial configuration and whether the head is currently scanning the chosen cell. Let us assume that \( \mathcal{P}_1 \) chooses to keep track of cell \( j \). The information \( \mathcal{P}_1 \) keeps track of (like the cell position and contents) are private to \( \mathcal{P}_1 \) and do not influence the behavior of \( \mathcal{P}_2 \). After this, \( \mathcal{P}_2 \) and \( \mathcal{P}_1 \) alternate taking turns, with \( \mathcal{P}_2 \) first guessing the next head position, symbol being scanned, and state, and \( \mathcal{P}_1 \) then “checking” the correctness of the guess; \( \mathcal{P}_1 \) checks that if \( \mathcal{P}_2 \) guesses that the head moves to position \( j \) then the symbol guessed by \( \mathcal{P}_2 \) is indeed the same as the one it has recorded (i.e. correct), and if the head was at \( j \) in the previous time instant, then \( \mathcal{P}_1 \) updates the cell contents correctly, and checks that \( \mathcal{P}_2 \)’s guess of head position and state are consistent with the transition function of \( M \). If \( \mathcal{P}_1 \) ever detects an error then it goes to an accepting state from which it never leaves. \( \mathcal{P}_1 \) also goes to the accepting state if \( \mathcal{P}_2 \) ever guesses that the state is \( q_a \) (the unique accepting state of \( M \)), and it goes to a reject state if \( \mathcal{P}_2 \)’s guess for the state is \( q_r \) (the unique rejecting state of \( M \)). Notice that \( \mathcal{P}_1 \)’s computation after initialization is completely deterministic. Next, observe that, any sequence over
\(\Sigma_a\) guessed by \(\mathcal{P}_2\) which does not correspond to a valid computation, is detected by \(\mathcal{P}_1\) with nonzero probability, and in such a case \(\mathcal{P}_1\) reaches it’s accepting state with nonzero probability. If \(\mathcal{P}_2\) guesses a correct computation that ends in \(q_a\), then \(\mathcal{P}_1\) reaches it’s accepting state with probability 1, and if \(\mathcal{P}_2\) guesses a correct computation ending in \(q_r\), then \(\mathcal{P}_1\) rejects with probability 1. Thus, \(\mathcal{P}_1\) reaches it’s accepting state with nonzero probability under all locally Markovian schedulers iff \(\sigma \in L\). The safety specification \(\text{Spec}\) goes to a reject state whenever process \(\mathcal{P}_1\) of \(\mathcal{P}_\sigma\) goes to it’s reject state. This ensures that \(\sigma \in L\) iff \(\mathcal{P}_\sigma \models_{\geq 0} \text{Spec}\).

We now show that given a program \(\mathcal{P}\) and a deterministic \(\text{Spec}\), the problem of deciding whether \(\mathcal{P} \models_{= 1} \text{Spec}\) is \(\text{EXPSPACE}\)-hard. The proof is similar to that of the first part of the Theorem but with important differences. Let us fix \(L, M\) and input \(\sigma\) as before. The concurrent program \(\mathcal{P}_\sigma\) we will construct will once again consist of two processes \(\mathcal{P}_1\) and \(\mathcal{P}_2\), where \(\mathcal{P}_2\) will be responsible for guessing a sequence from \(\Sigma_a\), and \(\mathcal{P}_1\) will check if it corresponds to a valid computation of \(M\) on \(\sigma\). Before, we outline the construction of \(\mathcal{P}_\sigma\), let us point out why the reduction in first part does not work in this case — for \(\sigma \in L\) if we consider a guess of \(\mathcal{P}_2\) that is not valid, the probability with which \(\mathcal{P}_1\) will detect the error may not be 1. Informally, we will fix this problem by forcing \(\mathcal{P}_2\) to repeatedly guess the computation, so that the probability that \(\mathcal{P}_1\) detects the error is bumped up.

Process \(\mathcal{P}_1\) can be in one of 3 modes: \(\text{start, normal, and accept}\). The mode of process \(\mathcal{P}_1\) is private and is not known to \(\mathcal{P}_2\). Initially \(\mathcal{P}_1\) is in \(\text{start}\) mode. When in \(\text{start}\) mode, as in the proof of the first part, \(\mathcal{P}_1\) will probabilistic pick a cell location (say \(j\)) to monitor, and will remember the contents of cell \(j\) in the initial configuration. The cell being monitored by \(\mathcal{P}_1\) and it’s contents remembered will once again be private. After initialization \(\mathcal{P}_1\) will change to \(\text{normal}\) mode. As in the first part of lower bound proof, the processes \(\mathcal{P}_1\) and \(\mathcal{P}_2\) will now alternate, with \(\mathcal{P}_2\) first nondeterministically guessing the next head position, symbol scanned and state, and then \(\mathcal{P}_1\) “checking” to see if the guess is correct. As before, if \(\mathcal{P}_1\) detects an error, or if \(\mathcal{P}_2\) guesses the next state of \(M\) to be \(q_a\), \(\mathcal{P}_1\) will move to \(\text{accept}\) mode and it will stay in that mode for the rest of the computation. In the other cases, \(\mathcal{P}_1\) behaves differently from the first part of the lower bound proof. If \(\mathcal{P}_1\) does not detect an error, and \(\mathcal{P}_2\) guesses the next state to be \(q_r\), then \(\mathcal{P}_2\) changes it’s mode to \(\text{start}\), reinitializes, changes mode to \(\text{normal}\) before giving \(\mathcal{P}_2\) a turn; it thus forces \(\mathcal{P}_2\) to “restart” the computation. On the other hand, if \(\mathcal{P}_1\) does not detect an error and \(\mathcal{P}_2\)’s guess for the next state is neither \(q_a\) nor \(q_r\), then with probability \(\frac{1}{2}\), \(\mathcal{P}_1\) will choose to continue monitoring the current cell, or with probability \(\frac{1}{2}\), \(\mathcal{P}_1\) will choose to start monitoring the cell where \(\mathcal{P}_2\) guesses the head to be next; in the second case \(\mathcal{P}_1\) takes the cell contents to be the symbol guessed by \(\mathcal{P}_2\). We will take the deterministic specification \(\text{Spec}\) to be the one accepting all computations of \(\mathcal{P}_\sigma\) in which \(\mathcal{P}_1\) is in the \(\text{accept}\) mode infinitely often.

Observe that suppose \(\sigma \notin L\), then the scheduler that forces \(\mathcal{P}_2\) to guess the correct rejecting computation of \(M\) repeatedly, will ensure that \(\mathcal{P}_1\) never goes to the mode \(\text{accept}\). Thus if \(\sigma \notin L\) then \(\mathcal{P}_\sigma \not\models_{= 1} \text{Spec}\). Consider the case of \(\sigma \in L\). If we consider a scheduler under which \(\mathcal{P}_2\) guesses a computation (correct or incorrect) on which the state of \(M\) is \(q_a\) at some point, then \(\mathcal{P}_1\) reaches mode \(\text{accept}\) with probability 1. The crux of the correctness proof is in arguing that even on incorrect guesses by \(\mathcal{P}_2\) where \(q_a\) is never guessed to be the state of \(M\), the process \(\mathcal{P}_1\) will move to mode \(\text{accept}\) with probability 1. This reduction is similar to one that is used in the proof of Theorem 6.6 in [7]. Using ideas outlined in that proof, one can show that there is a constant \(p > 0\) such that, from every point of every branch of the computation, within \(2k\) turns of process \(\mathcal{P}_1\), it’s mode will be set to \(\text{accept}\) with probability greater than or equal to \(p\); recall that \(k\) is the bound on the running time of \(M\) on \(\sigma\). From all this, we see that on all computations, generated by locally Markovian schedulers,
\( P_1 \)'s mode will eventually be set to \textit{accept} with probability 1. Thus, \( \sigma \in L \) iff \( \mathcal{P}_\sigma \models^{lm}_{=1} \text{Spec} \).

H Proof of Theorem 8

Decidability of the model checking questions in this case is established by exploiting a connection between POMDPs and concurrent programs with at most one nondeterministic process. This connection is stated in Lemma 13 and proved first.

**Lemma 13.** Given a program \( P \) with \( \Sigma \) as the set of states and a deterministic specification \( \text{Spec} \) with \( Q \) as the set of states and \( Q_f \subseteq Q \) as the set of final states, let \( \Delta \) be \( \text{Loc}(P, \text{Spec}) \), the set of \( \text{Spec} \)-determined and locally consistent functions. If all but one processes of \( P \) are deterministic, then there is a POMDP \( \mathcal{M} \) with \( \Delta \) as the set of actions such that the following hold.

- The states of \( \mathcal{M} \) are \( Q \times \Sigma \).
- \( P \models^{vc}_{=1} \text{Spec} \) iff for any observation-based scheduler \( S \), the measure of paths of \( \mathcal{M} \) generated by \( S \) that visit \( Q_f \times \Sigma \) infinitely often is \( 1 \).
- If \( \text{Spec} \) is a safety specification then \( P \models^{vc}_{=0} \text{Spec} \) iff for any observation-based scheduler \( S \), the measure of paths of \( \mathcal{M} \) generated by \( S \) that visit \( Q_f \times \Sigma \) infinitely often is \( >0 \).

**Proof.** Let \( P \) be a program of \( n \) processes, all of which except possibly one, are deterministic. Let the index of process that is not deterministic be \( i \). Similar to the Proposition 4, we can show that in order to decide whether \( P \models^{vc}_{=0} \text{Spec} \) for deterministic \( \text{Spec} \) (and deciding whether \( P \models^{vc}_{=1} \text{Spec} \) for safety specification), it suffices to consider view consistent schedulers \( \eta \) such that \( \eta \) is view consistent and satisfies the following.

- For any two histories \( h \) and \( h' \) such that \( h|i = h'|i \), \( \text{Spec}(h) = \text{Spec}(h') \) and last\( (h) = \text{last}(h') \), we have that \( \eta(h) = \eta(h') \).

Let us called these schedulers \( \text{Spec} \)-constrained.

\( \mathcal{M} \) is constructed as follows. The set of states of \( \mathcal{M} \) are \( Q \times \Sigma \) and the set of actions of \( \mathcal{M} \) is \( \Delta = \text{Loc}(P, \text{Spec}) \). The initial state of \( \mathcal{M} \) is \((q_0, s_0)\) where \( q_0 \) is the initial state of \( \text{Spec} \) and \( s_0 \) is the initial state of \( P \). We have to construct the transition function and the equivalence relation on \( Q \times \Sigma \). The construction of the transition function of \( \mathcal{M} \) is exactly the construction of the transition relation of the PBA \( B \) constructed in Lemma 6. Finally, we declare two states \((q_1, s_1)\) and \((q_2, s_2)\) of \( \mathcal{M} \) equivalent iff \( s_1|i = s_2|i \).

The result now follows from the observation that there is a bijection \( g \) between the set of \( \text{Spec} \)-constrained view consistent schedulers of \( P \) and the set of observation-based schedulers on \( \mathcal{M} \) such that for any \( \text{Spec} \)-constrained view consistent scheduler \( \eta \), the probability that the computation generated by \( \eta \) satisfies \( \text{Spec} \) is exactly the measure of paths of \( \mathcal{M} \) generated by \( g(\eta) \) that visit \( Q_f \) infinitely often.

**Proof.** (of Theorem) Membership in \textit{2-EXPTIME} is proved first. We first consider the case of checking if \( P \models^{vc}_{=0} \text{Spec} \) for a given safety specification \( \text{Spec} \). Now, note it was shown in [2] that given a POMDP, \( \mathcal{M}_1 = (Q, \text{Act}, q_0, \delta, \equiv) \) and a set of states \( Q_f \subseteq Q \), the problem of checking if there is an observation-based scheduler \( S \) on \( \mathcal{M}_1 \) such that the measure of paths generated by \( S \) that visit \( Q_f \) infinitely often is \( 1 \), is decidable in \textit{EXPTIME}.

Now, given program \( P \) and safety specification \( \text{Spec} \), construct a POMDP \( \mathcal{M} \) as in the proof of Lemma 13. If \( q_r \) is the reject state of \( \text{Spec} \) and \( \Sigma \) the set of states of \( P \), it follows from the construction in Lemma 13 that \( P \models^{vc}_{=0} \text{Spec} \) iff there is no observation-based scheduler \( S \) on \( \mathcal{M} \) such that measure
of paths generated by $S$ that visit $\{q_r\} \times \Sigma$ infinitely often is $1$. The latter can be decided thanks to the above mentioned result in [2]. Apriori, it may seem that using this decision problem to check if $P \models_{\oplus}^c \varphi$ would result in a $3$-$\text{EXPTIME}$ decision procedure (because the set of actions of $M_1$ is doubly exponential in the size of the input). However, a careful examination of the algorithm in [2] shows that the algorithm when applied to $M$ will run in time doubly exponential in the size of $P$ and $\varphi$ (this is because the algorithm in [2] is exponential in size of the states of the input POMDP, but polynomial in the size of actions, the size of numbers used in probabilities and the number of equivalence classes).

Similarly, we can show that given program $P$ and deterministic specification $\varphi$, the problem of checking if $P \models^c_{\oplus} \varphi$ is decidable in $2$-$\text{EXPTIME}$ using the following result in [9] (which also follows from [12]) -- given a POMDP $M_1 = (Q, \text{Act}, q_0, \delta, \equiv)$ and a set of states $Q_f \subseteq Q$, the problem of deciding if there is an observation-based scheduler $S$ on $M_1$ such that the measure of paths generated by $S$ that visit $Q_f$ finitely often is $> 0$, is decidable in $\text{EXPTIME}$.

(Lower bounds.) We first consider the case that $\varphi$ is a safety specification. We show that in this case that the problem of checking if $P \models^c_{\oplus} \varphi$ is $2$-$\text{EXPTIME}$-hard. The proof is similar to the lower bound proof of Theorem 7. In this case, $L$ is a language that can be decided in double exponential time. We assume $M$ is an alternating Turing machine that decides $L$ in exponential space.

In the proof of the first part of the lower bound proof of Theorem 7, we make the following changes to get the lower bound proof. Given an input $x$, we construct $P_x$ from consisting of two processes $P_1$ and $P_2$. The construction of $P_1$ and $P_2$ is the same as in the first part of the lower bound proof of Theorem 7 except for the following changes. Process $P_2$ remembers the previous state that it guessed. Call it $q'$. It also keeps in a shared variable $i$ the previously guessed head position. If $q'$ is a universal state then in the current turn it nondeterministically guesses the move of $M$, that is the new state $q$, the new contents of the cell pointed to by the head position $i$, and also the direction of head movement $d$. If $q'$ is an existential state these values are guessed probabilistically. In both the cases, the guessed direction is added to $i$. After this it gives the turn to $P_1$.

Everything else is identical to the first part of the lower bound proof of Theorem 7. It is not difficult to see that $x$ is in $L$ iff under $P_x$ satisfies the specification $\varphi$ with nonzero probability under all all view consistent schedulers. $\varphi$ is the same safety spec given in the first part of the lower bound proof of Theorem 7.

The lower bound for checking if $P \models^c_{\oplus} \varphi$ for deterministic specification $\varphi$ is similarly obtained by modifying the the second part of the lower bound proof of Theorem 7 along the same lines.

I Proof of Theorem 10

(a) Consider first that $\varphi$ is a deterministic specification. We recall here some standard definitions/results of model-checking Markov Decision Processes from [5]. Formally, given a set of propositions $AP$ a Markov Decision Process (MDP) $M$ is a tuple $(Q, q_0, \delta, L)$ where $Q$ is a finite set of states, $q_0$ is the initial state, $\delta : Q \to 2^{\text{Dist}(Q)}$ is the transition function which given a state $q \in Q$ gives a finite nonempty set of discrete probability distributions over $Q$ (for our purposes we assume that probabilities are always rational numbers), and $L : Q \to 2^{AP}$ is a labeling function which labels states with propositions true in it. As in the case of our programs, a history is a finite sequence of states, i.e., a history is an element of $Q^+$. Similar to the case of
programs, a scheduler $S$ is a function $S : Q^+ \to \text{Dist}(Q)$ such that $S(h) \in \delta(\text{last}(h))$. As in the case of programs, a scheduler gives rise to a measure space $(Q^\omega, \Gamma, \mu)$.

A logic $\text{PCTL}^*$ is defined in [5] which is interpreted over MDPs. It is shown in [5] that the model checking problem for this logic is decidable in polynomial time. For our purposes, we will be interested in two kinds of $\text{PCTL}^*$ formulas: $\text{Pr}_{\geq r}(\Box P)$ and $\text{Pr}_{> r}(\Box P)$ where $r$ is a rational number in $[0, 1]$ and $P \in \text{AP}$. $\mathcal{M}$ is said to satisfy $\text{Pr}_{\geq r}(\Box P)$ ($\text{Pr}_{> r}(\Box P)$ respectively) iff for all schedulers, the measure of the set $\{\alpha \in Q^\omega \mid P \in L(\alpha[i])\}$ for infinitely many $i$ is $\geq r$ ($> r$ respectively).

We now proceed with the proof of the theorem. Given a program $\mathcal{P}$, a deterministic specification $\text{Spec}$ and a rational number $x$, we will convert our verification problem to a $\text{PCTL}^*$ model checking problem on an appropriate MDP $\mathcal{M}$. The size of MDP $\mathcal{M}$ will be exponential in the input size. The set of propositions will be chosen to be $\{P\}$. The construction of $\mathcal{M}$ will be carried out in two steps.

First, we construct $\mathcal{M}_1$ from $\mathcal{P}$ as follows. The set of states of $\mathcal{M}_1$ will be the set of states of $\mathcal{P}$. The initial state of $\mathcal{M}_1$ will be the initial state of $\mathcal{P}$. The labeling function will assign each state to the empty set. Finally the transition function $\delta_1$ is constructed as follows. For each state program $s$, $\delta_1(s)$ is the set $\{\tilde{tr} \mid tr$ is a transition of $\mathcal{P}$ enabled in $s\}$, where $\tilde{tr}$ is the probability distribution such that $\tilde{tr}(s')$ is the probability of obtaining $s'$ after $tr$ is executed in state $s$.

Now, $\mathcal{M}$ is constructed by taking the “cross-product” of $\text{Spec}$ and $\mathcal{M}_1$ as follows. The set of states of $\mathcal{M}$ are pairs $(q, s)$ where $q$ is a state of $\text{Spec}$ and $s$ is a state of $\mathcal{M}$ (and hence of $\mathcal{P}$). The initial state of $\mathcal{M}$ is $(q_0, s_0)$ where $q_0$ is the initial state of $\text{Spec}$ and $s_0$ the initial state of $\mathcal{M}$. The labeling function $L$ of $\mathcal{M}$ is defined as follows. $L((q, s)) = \{P\}$ iff $q$ is a final state of $\text{Spec}$; otherwise $L((q, s)) = \emptyset$. The transition function $\delta$ of $\mathcal{M}$ is defined as follows. For each state $q$ of $\text{Spec}$ and $s$ of $\mathcal{M}$,

$$\delta(q, s) = \{< q', \mu > \mid q'$ is the unique state of $\text{Spec}$ s.t. $(q, s, q')$ is a transition of $\text{Spec}$ and $\mu \in \delta_1(s)\}$$

where $< q', \mu >$ is the probability distribution over states of $\mathcal{M}$ which assigns probability $\mu(s_1)$ to $(q_1, s_1)$ iff $q_1 = q'$, otherwise it assigns probability 0.

It is easy to see that $\mathcal{M}$ can be constructed in time exponential in the size of $\mathcal{P}$ and $\text{Spec}$. Furthermore, it is easy to see that $\mathcal{P}$ satisfies the specification $\text{Spec}$ with probability $\geq x$ ($> x$ respectively) iff $\mathcal{M}$ satisfies the $\text{PCTL}^*$ formula $\text{Pr}_{\geq x}(\Box P)$ ($\text{Pr}_{> x}(\Box P)$ respectively). The result now follows.

We can similarly prove the upper bound for the case $\text{Spec}$ is a deterministic predicate Rabin automaton.

(b) It suffices to show the result for the case $y = 0$. We will prove the problem to be $\text{EXPTIME}$-hard for the case when $\text{Spec}$ is a safety specification. We show that every language in $\text{EXPTIME}$ is reducible to the given problem. Let $L$ be a language in $\text{EXPTIME}$. Clearly, the complement $\overline{L} \in \text{EXPTIME}$. Hence there exists a single tape Alternating Turing Machine (ATM) $M$ that recognizes $\overline{L}$ using space $p(n)$, for some polynomial $p(n)$, where $n$ is the length of the input. We assume that $M$ always halts in either an accepting state or a rejecting state. Corresponding to any input string $x$ to $M$, we construct a program $P_x$ that has a single process $P$. Let $n$ be the length of $x$ and $m = p(n)$. Essentially, $P$ maintains $m$ number of variables, say $x_0, \ldots, x_{m-1}$ where $x_i$ captures the contents of the $i^{th}$ cell of the tape. Note each $x_i$ takes only
a bounded number of values and can be represented by a bounded number of boolean variable. In addition, it also maintains two other variables position and state that give the head position and the current state of the finite control.

All the variables are initialized to represent the initial tape contents which is the string $x$ and initial state and head position. Corresponding to each value of the head position and each control state, it has the following transitions that are enabled. If the value of state is an existential state of $M$, then $\mathcal{P}$ nondeterministically changes the value state, position and the value of the variable $x_{\text{position}}$ according to the transitions of $M$. If the value of state is a universal state then $\mathcal{P}$ probabilistically changes the value state, position and the value of the variable $x_{\text{position}}$ according to the transitions of $M$. Let Spec be the safety specification which takes an execution of $\mathcal{P}_x$ as input and goes to a reject state if the variable state ever has the accepting state of $M$ as it's value. It is straightforward to see that there is a computation of $\mathcal{P}_x$ that is rejected by Spec with probability 1 if $x$ is accepted by $M$. Equivalently every computation of $\mathcal{P}_x$ is accepted by Spec with nonzero probability iff $x \in L$.

(c) We first consider the case when Spec is a deterministic specification and $y \in \{0, 1\}$. It suffices to show the result for the case $y = 1$. For this case, the proof is a simple modification of the proof of part (b) of the theorem. Let $L, M, x$ be as given in that proof. Let $q_a, q_f$ be the accept and reject states of $M$, respectively. The program $\mathcal{P}_x$ is as given in the proof of part (b), with the following change: if the variable state has the value which is the accepting state of $M$, i.e. $\text{state} = q_a$, then $\mathcal{P}$ resets all the variables $x_i$, $0 < i < m$, the variable position and state so that they represent the initial configuration of $M$ with input $x$. Thus, if in any execution the state of $M$ is the accepting state, then $\mathcal{P}_x$ repeats the simulation of $M$ starting from the initial configuration. Now the following is easily seen. If $x \in L$, i.e., $x$ is not accepted by $M$, then on all computations of $\mathcal{P}_x$ the reject state of $M$ will eventually be reached, i.e., eventually $\text{state} = q_f$, with probability 1. On the other hand, if $x \notin L$, i.e., $x$ is accepted by $M$, then there exists a computation, generated by some scheduler, on which variable state is never equal to $q_f$. Now, let Spec be the deterministic specification that accepts an execution of $\mathcal{P}_x$ if eventually $\text{state} = q_f$ is satisfied. It is not difficult to see that $x \in L$ iff every computation of $\mathcal{P}_x$ is accepted by Spec with probability 1.

We now show the case when $y \in \{0, 1\}$ and Spec is a safety automaton (this automatically implies the result for the case $y \in \{0, 1\}$ and Spec is a deterministic automaton). For this case, it suffices to show the result for the case $y < \frac{1}{2}$.

As in the proof of part (b) we reduce a language in EXPTIME to this problem. Let $L$ be a language in EXPTIME. Let $M$ be a ATM that recognizes the complement of $L$, i.e., $\overline{L}$, using polynomial space and space $p(n)$ on input of length $n$. Let $c$ be the maximum value of the number of successor states of a universal state of $M$. For any given input $x$, let $P_x$ be as given in proof of part (b). We modify $P_x$ and construct a program $P'_x$ such that $P'_x$ satisfies the safety specification Spec of part (b) with probability $\geq y$ iff $x \in L$. Let $n$ be the length of $x$, $m = 2^{p(n)}$ and $q = (\frac{1}{2})^m$. It should be easy to see that if $x \in L$ then on every computation of $P_x$, eventually the variable state is the accepting state with probability $\geq q$. The program $P'_x$ acts as follows. With probability $y(1 - q)$ it goes to the accept state immediately and with the remaining probability it behaves like $P_x$. To do the above, in the beginning with probability $y$, it goes and runs $P_x$ and with probability $(1 - y)$ it goes to an intermediate state $r$. In location $r$, it initializes a counter of length $p(n)$ and counts it up from
0 to the maximum value. Before each increment, with probability \( \frac{1}{e} \), it goes to the accepting state, and with probability \( (1 - \frac{1}{e}) \) it remains in state \( r \). When the counter has maximum value then it simulates program \( P_x \). It is straightforward to see that the probability that \( P'_x \) goes to the accepting state in the beginning is \( y(1 - (\frac{1}{e}))^m \), i.e., \( y(1 - q) \), and with the remaining probability it simulates \( P_x \).

Clearly, if \( x \notin L \) then \( P'_x \) goes to the accepting state with probability \( y(1 - q) \), i.e., with probability \( < y \). Now, assume \( x \in L \). The probability that \( P'_x \) accepts is \( \geq y(1 - q) + (1 - y(1 - q))q \) which is seen to be \( \geq y \) for values of \( y \leq \frac{1}{2} \).

\[ \square \]

### J Proof of Theorem 9

We consider programs with two processes. The results are easily extended to an arbitrary number of processes. First we prove some properties of programs in which processes are mutually exclusive.

Recall that, for \( u = 1, 2 \), \( V_u \) is the set of variables visible to process \( P_u \). For \( u = 1, 2 \), a \( P_u \)-state is a function that associates a value with each local variable of \( P_u \), i.e., with each variable in \( V_u - (V_1 \cap V_2) \). A shared state is a function that associates an appropriate value with each variable in \( V_1 \cap V_2 \), i.e., to each shared variable. Suppose \( s_1, s_2 \) are \( P_1 \)- and \( P_2 \)-states respectively and \( s_{12} \) is a shared state then \( s = (s_1, s_2, s_{12}) \) represents a unique state of \( P \). With out any confusion, we let \( S \), the set of states of \( P \), be represented by the set of all triples of the above form.

We say that, for \( u = 1, 2 \), process \( P_u \) is enabled in the state \( (s_1, s_2, s_{12}) \) of \( P \) if some transition of \( P_u \) is enabled in it. Since we assumed that the processes \( P_1, P_2 \) are mutually exclusive, exactly one of them is enabled in this state.

**Lemma 14.** If processes in \( P \) are mutually exclusive, then in any global state, the process that is enabled is uniquely determined by the shared state. That is, for each shared state \( s_{12} \), there is a unique \( u = 1, 2 \) such that \( P_u \) is enabled in all global states of the form \( (s_1, s_2, s_{12}) \).

**Proof:** The lemma of the lemma is seen as follows. Let \( S_1, S_2 \) be the sets of all \( P_1 \)-states and all \( P_2 \)-states, respectively. Also, let \( S_{12} \) be the set of all shared states. Let \( s_{12} \in S_{12} \). Suppose \( P_1 \) is enabled in some state \( s = (s_1, s_2, s_{12}) \) of \( P \). Due to the mutual exclusion assumption, observe that \( P_2 \) is disabled in \( s \), and hence is disabled in all states in \( S_1 \times \{s_2\} \times \{s_{12}\} \); this is because the enabling of \( P_2 \) depends only on \( s_2 \) and \( s_{12} \); hence \( P_1 \) is enabled in all these states also, as one of them should be enabled in every state; since enabling of \( P_1 \) does not depend on \( s_2 \), we see that \( P_1 \) is enabled in all states \( S_1 \times S_2 \times \{s_{12}\} \). Thus, we see that the shared state uniquely determines which of the processes is enabled, and together with the local state determines which actual transitions of that process are enabled.

Let \( \eta \) be a view consistent scheduler for \( P \) and \( M_{P,\eta} \) be the computation of \( P \), i.e., the infinite Markov chain, generated by \( \eta \). Now we prove some properties of \( M_{P,\eta} \). Recall that nodes of \( M_{P,\eta} \) are histories, i.e., finite sequence of states of \( P \) with the history \( s_0 \) being that initial node, where \( s_0 \) is the initial state of \( P \). For any \( k > 0 \), let \( H_k \) be the set of histories of length \( k \), in \( M_{P,\eta} \), that are reachable from \( s_0 \). We define two equivalence relations \( R_{k,1} \) and \( R_{k,2} \) on \( H_k \). Two histories \( h, h' \) are equivalent with respect to \( R_{k,i} \), for \( i = 1, 2 \), iff the view of process \( P_i \) is same in \( h \) and \( h' \), i.e., \( h \upharpoonright i = h' \upharpoonright i \).
Let $E_{k,1}$ and $E_{k,2}$ be the two sets of equivalence classes of the equivalence relations $R_{k,1}$ and $R_{k,2}$, respectively. It is straightforward to see that for any $C \in E_{k,1}$ and any $D \in E_{k,2}$ there is at most one element in $C \cap D$.

Now, we define an undirected bipartite graph $G_k$ whose nodes are the equivalence classes in $E_{k,1} \cup E_{k,2}$. The edges of $G_k$ are defined as follows. There is an edge from an equivalence class $C \in E_{k,1}$ to an equivalence class $D \in E_{k,2}$ iff $C \cap D \neq \emptyset$. Clearly $G_k$ is a bipartite graph. The following theorem shows that the connected components of of $G_k$ are complete bipartite graphs.

**Lemma 15.** The connected components of $G_k$ are complete bipartite graphs when the processes in $P$ are mutually exclusive or each of them locally stutter free.

**Proof:** The lemma is proved by induction on $k$. The lemma trivially holds for $k = 1$, since $H_1$ consists of the single history $z_0$. Assume that the processes in $P$ are mutually exclusive.

Assume that the lemma holds for some $k$. We prove this for $k + 1$. Consider any connected component $C$ in $G_k$. Let $C, D$ be any two nodes in $G_k$. It should be easy to see that for any two histories $h \in C$ and $h' \in D$, the projections of $h$ and $h'$ on the shared states are identical. As a consequence, the shared state at the end of $h, h'$ are identical. From lemma 14, we see that, for exactly one $u \in \{1, 2\}$, process $P_u$ is enabled on all the last states of histories in the nodes (i.e., equivalence classes) of $C$. Without loss of generality, assume that $P_1$ is enabled on all of them. Scheduler $\eta$ may schedule different transitions of $P_1$ on histories belonging to different equivalences in $C$, but it schedules the same transition on all histories belonging to the same equivalence class. Let $r$ be the maximum number of successors of any transition of $P_1$ in any state. It should be easy to see that each history $h$ in any equivalence class of $C$ splits into at most $r$ histories in $H_{k+1}$. Also, the values of the private variables of process $P_2$ in the last state of $h$ will remain unchanged.

Consider any equivalence class $C$ of $R_{k,1}$ in $C$. Let $\tau_j$ be the transition of $P_1$ that is scheduled on these histories. Let $r'$ be the number of successor states of $\tau_j$ on histories in $C$. The equivalence class $C$ is going to be split into $r'$ number of equivalence classes. Note that another equivalence class in of $R_{k,1}$ in $C$ may be split in to a different number of equivalence classes which is bounded by $r$. Now, each equivalence class of $R_{k,2}$ in $C$ is going to be split into at most $r$ equivalence classes as the number of values for the shared variables in $X$. From all these discussion, it can be shown that the lemma holds for $k + 1$ also. \qed

Now, we give a method for checking if there is a view consistent scheduler $\eta$ such that $M_{P, \eta}$ satisfies a deterministic specification Spec with probability less than 1.

Let $q_0$ be the start state of Spec. For any finite or infinite history $h$ and state $q$ of Spec, let $\text{run}(q, h)$ denote the run of Spec on $h$ starting from the state $q$. Recall that if $h$ is a finite history then let $\text{Spec}(q, h)$ denote the last state of the run $\text{run}(q, h)$. The following lemma gives a necessary condition for satisfaction of Spec by $M_{P, \eta}$ with probability less than 1.

**Lemma 16.** Let $\eta$ be any scheduler such that $M_{P, \eta}$ satisfies Spec with probability less than 1. Then, for every integer $m > 0$, there exists a history $h$ reachable in $M_{P, \eta}$ such that for every non-empty history $h'$ of length $\leq m$ such that $hh'$ is reachable in $M_{P, \eta}$, $\text{Spec}(hh')$ is not an accepting state of Spec.

**Proof:** The lemma is proved on similar lines as lemma 6.2 in [7]. Let $\eta$ be any scheduler such that $M_{P, \eta}$ satisfies Spec with probability less than 1 and $m$ be any integer such that $m > 0$. We prove the lemma by contradiction. Contrary to the lemma assume that for every reachable history $h$ in $M_{P, \eta}$,
there exists a non-empty history $h'$ of length $\leq m$, such that $hh'$ is reachable in $\mathcal{M}_{P,\eta}$ and $\text{Spec}(hh')$ is an accepting state of $\text{Spec}$. For any integers $l > i \geq 0$, let $F_{i,j}$ denote the probability that, for any reachable history $h$ in $\mathcal{M}_{P,\eta}$, of length $l$, $\text{run}(q_0, h)$ has at least $i$ number of occurrences of accepting states in it. We let $F_{0,i}$ denote the limit of $F_{i,j}$ as $l \to \infty$.

Let $p > 0$ be the smallest probability of any path of length at most $m$ in $\mathcal{M}_{P,\eta}$ (note that the probability of any such path is the product of probabilities of transitions taken in it).

For any $l > i > 0$, it is easy to see that $(F_{i,j+1} - F_{i,j})$ is the probability that for a history $h$ of length $l$, reachable in $\mathcal{M}_{P,\eta}$, $\text{run}(q_0, h)$ has $i - 1$ occurrences of accepting states in it.

Let $l, u, i$ be positive integers such that $l, u > i$. It should be easy to see that the following holds:

$$F_{l+u,m,i} \geq F_{l+(u-1)m,i} + (F_{l+(u-1)m,i-1} - F_{l+(u-1)m,i}) \cdot p$$

After simplification we get,

$$F_{l+u,m,i} \geq p \cdot F_{l+(u-1)m,i-1} + (1 - p) \cdot F_{l+(u-1)m,i}$$

By a simple induction on $u$, it is easy to prove that

$$F_{l+u,m,i} \geq p \cdot F_{l,m,i-1} \cdot \sum_{0 \leq j < u} (1 - p)^j$$

By letting $u$ go to $\infty$, we see that $F_{0,i} \geq F_{0,i-1}$ and this holds for any $l > 0$ and any $i > 0$. From this, we see that $F_{0,i} \geq F_{0, i-1}$. Since $F_{0,0} = 1$, we see that $F_{0,i} = 1$ for every $i > 0$. An infinite history in $\mathcal{M}_{P,\eta}$ is the limit of an an increasing sequence of histories reachable in it. From the above we see that with probability $1$, an infinite history in $\mathcal{M}_{P,\eta}$ satisfies $\text{Spec}$. This shows that $\mathcal{M}_{P,\eta}$ satisfies $\text{Spec}$ with probability $1$, which contradicts the assumption of the lemma. □

Let $X \subseteq \Sigma$ be a set of global states. We say that $X$ is closed if for every pair of states $(s_1, s_2, s_{12})$ and $(t_1, t_2, t_{12})$ in $X$ the following hold: $s_{12} = t_{12}$ and $(s_1, t_2, s_{12})$, $(t_1, s_2, t_{12})$ are also in $X$. From lemma 15, it is easily seen that the set of last states of a connected component in $\mathcal{G}_k$ is closed.

Since we also need to capture the runs of the specification $\text{Spec}$ on the histories, we consider extended states which are quadruples of the form $(s_1, s_2, s_{12}, q)$ where $(s_1, s_2, s_{12})$ is a state of $P$ and $q$ is a state of $\text{Spec}$. Let $Y$ be a set of extended states. We say that $Y$ is closed if the set \{ $(s_1, s_2, s_{12}) : \exists q (s_1, s_2, s_{12}, q) \in Y$ \} is closed. Let $Y$ be a closed set of extended states. Let $\phi$ be a function with domain $Y$ that associates a transition $\phi(s)$ of $P$ with each extended state $s$ in $Y$ that is enabled in $s$. We say that $\phi$ is a one step view consistent scheduler if for any two elements $s, s'$ in $Y$ and for each $u = 1, 2$ the following holds: if $s, s'$ have the same $P$-state and $\phi(s), \phi(s')$ are transitions of process $P_u$ then $\phi(s) = \phi(s')$.

Now, consider an extended state $s = (s_1, s_2, s_{12}, q)$ and $\tau$ be a transition of process $P_u$ (for $u = 1, 2$) enabled in $s$. Let $r$ be the unique next state of $\text{Spec}$ from state $q$ on input given by $(s_1, s_2, s_{12})$ (note that states of $P$ are inputs to $\text{Spec}$). We define $\text{Successors}(s, \tau)$ to be the set of all extended states of the form $(t_1, t_2, t_{12}, r)$ such that $(t_1, t_2, t_{12})$ is a successor of $s$ when $\tau$ is executed (note that each such successor is reached with some non-zero probability). Now, for any set $Y$ of extended states and one step view consistent scheduler $\phi$, let $\text{Successors}(Y, \phi) = \cup_{s \in Y} (\text{Successors}(s, \phi))$.

For any set $Y$ of extended states, we define an undirected graph $\mathcal{H}_Y$ whose nodes are extended states of $Y$ and whose edges are defined as follows: if $s = (s_1, s_2, s_{12}, q)$ and $t = (t_1, t_2, t_{12}, r)$
are two extended states in \( Y \) then there is an edge from \( s \) to \( t \) iff \( s_{12} = t_{12} \), and either \( s_1 = t_1 \) or \( s_2 = t_2 \), i.e., iff their shared states are identical and local states of one of the processes is same. Let \( Y_1, Y_2, ..., Y_k \) be the connected components of \( \mathcal{H}_Y \). We call these sets as the connected subsets of \( Y \). Now, we have the following lemma whose proof is left out.

**Lemma 17.** If \( Y \) is a closed set of extended states and \( \phi \) is a one step view consistent scheduler for \( Y \), then each of the connected subsets of \( \text{Successors}(Y, \phi) \) is a closed set.

Now, we define a bipartite graph \( G = (W, E) \) as follows. \( W = W_1 \cup W_2 \) where \( W_1 \) is the set of all closed sets of extended states of \( \mathcal{P} \) and \( \mathcal{Spec} \). The sets \( W_2 \) and \( E \) are defined as follows. Consider \( U \in W \). For each such \( U \) and for each one step view consistent scheduler \( \phi \) for \( U \), we have a node \((U, \phi)\) in \( W_2 \) and an edge from \( U \) to \((U, \phi)\) in \( E \). These are all the edges from \( U \). The edges from each node of the form \((U, \phi)\) are defined as follows. Now consider the set \( V = \text{Successors}(U, \phi) \) and let \( V_1, V_2, ..., V_k \) be all the connected subsets of \( V \). From lemma 17, we see that each \( V_i \), for \( 1 \leq i \leq k \), is a closed set of extended states and hence is in \( W_1 \). We have edges from \((U, \phi)\) to each of \( V_1, ..., V_k \). Note that \( G \) is a bi-partite graph. Let \( n \) be the size of \( \mathcal{P} \) which is the sum of the lengths of transitions in \( \mathcal{P} \). Let \( m \) be the number of states \( \mathcal{Spec} \). Note that the total number of extended states is \( O(m \cdot 2^n) \). Observe that each \( U \) represents a subset of the set of all extended states of \( \mathcal{P} \). Hence the number of elements in \( W_1 \) is \( O(2^{m-2^{\Omega(n)}}) \). For any given \( U \), since it’s cardinality is \( O(m \cdot 2^n) \), we see that the number of one step view consistent schedulers for \( U \) is \( O(2^{m-2^{\Omega(n)}}) \). From all these we see that the size of \( G \) is \( O(2^{m-2^{\Omega(n)}}) \).

Recall that, we want to check if there is a view consistent scheduler \( \eta \) such that the computation \( \mathcal{M}_{\mathcal{P}, \eta} \) satisfies \( \mathcal{Spec} \) with probability less than 1; equivalently, we want to check if the set of infinite histories of \( \mathcal{M}_{\mathcal{P}, \eta} \) that are not accepted by \( \mathcal{Spec} \) has non-zero probability. We convert \( G \) into a Kripke structure by labeling its nodes with a special proposition \( Q \) or it’s negation. Every node in \( W_2 \) is labeled with \( \neg Q \). A node \( U \) in \( W_1 \) is labeled with \( Q \) if there is an extended state \((s_1, s_2, s_{12}, q) \in U \) such that \( q \) is an accepting state of \( \mathcal{Spec} \), otherwise it is labeled with \( \neg Q \).

Let \((s, q)\) be an extended state of \( \mathcal{P} \) and \( \mathcal{Spec} \). We say that \((s, q)\) is feasible in \( \mathcal{P} \) if for some scheduler \( \eta \) there is a history of the form \( h s \) reachable from the initial history \( s_0 \) with non-zero probability in the Markov chain \( \mathcal{M}_{\mathcal{P}, \eta} \) and \( q = \mathcal{Spec}(h) \). Note that in this definition, we do not require \( \eta \) to be view consistent. Checking feasibility of an extended state \((s, q)\) can be decided in time \( O(m \cdot 2^{\Omega(n)}) \).

Observe that a set consisting of a single extended state is closed and hence is a node in \( G \). Now we have the following lemma. In this lemma, it is assumed that \( \mathcal{Spec} \) is defined on the shared state of the program, i.e., it’s input symbols are the shared states. Note that for any two histories \( h, h' \) that are in the same connected component of \( G_k \) (for any \( k > 0 \)), their projections on the shared state is same and hence \( \mathcal{Spec}(h) = \mathcal{Spec}(h') \).

**Lemma 18.** There is a view consistent scheduler \( \eta \) such that \( \mathcal{M}_{\mathcal{P}, \eta} \) satisfies the specification \( \mathcal{Spec} \) with probability less than 1 iff there is a feasible extended state \((s, q)\) such that the node \( \{(s, q)\} \) in \( G \) satisfies the modal \( \mu \)-calculus formula \( f = vX (\neg Q \land \diamond X) \) where \( \diamond \) and \( \square \) are the existential and universal “nexttime” operators and \( vX \) is the greatest fixpoint operator.

**Proof:** If there is a feasible state \((s, q)\) such that \( \{(s, q)\} \) satisfies \( f \) then it is fairly easy to show that there is a view consistent scheduler \( \eta \) such that \( \mathcal{M}_{\mathcal{P}, \eta} \) satisfies \( \mathcal{Spec} \) with probability < 1.

Now assume that there is a view consistent scheduler \( \eta \) such that \( \mathcal{M}_{\mathcal{P}, \eta} \) satisfies \( \mathcal{Spec} \) with
probability $< 1$. Let $l$ be the total number of $W_1$-nodes in the graph $G$. From lemma 16, we see that there exists a history $h$ reachable in $\mathcal{M}_{P, \eta}$ such that for every non-empty history $h'$ of length $\leq l + 2$ such that $hh'$ is reachable in $\mathcal{M}_{P, \eta}$, $\text{Spec}(hh')$ is not an accepting state of $\text{Spec}$. Let $i_1$ be a state of $\mathcal{P}$ such that the history $h_1 = hi_1$ is reachable in $\mathcal{M}_{P, \eta}$ and let $r_1 = \text{Spec}(h)$. Now we construct a sub-graph $G'$ of $G$ containing the node $\{(t_1, q_1)\}$ such that each node in $G'$ satisfies $\neg Q$, each $\phi_1$-node in $G'$ has a single successor and each $W_2$-node in $G'$ has all it's successors in $G'$. Such a sub-graph guarantees that $\{(t_1, q_1)\}$ satisfies $f$. In any $W_1$-node of $G'$, the automaton state components of all extended states in it are going to be identical. Together with $G'$, we also define a function $\psi$ that maps each extended state in a $W_1$-node, i.e., each pair $(X, s)$ where $X$ is a $W_1$-node in $G'$ and $s \in X$, to a history $\psi(X, s)$ in $\mathcal{M}_{P, \eta}$ satisfying the following two properties.

(A) If $s = (s_1, s_2, s_{12}, q) \in X$ then the last state of $\psi(X, s)$ is $(s_1, s_2, s_{12})$ and $q = \text{Spec}(h')$ where $h'$ is the prefix of $\psi(X, s)$ excluding it’s last state.

(B) If $s = (s_1, s_2, s_{12}, q)$ and $t = (t_1, t_2, s_{12}, r)$ any two extended states in $X$, then for each $i = 1, 2$, $s_i = t_i$ iff $\psi(X, s)[i] = \psi(X, s_1)[i]$.

$G'$ and $\psi$ are defined in successive steps. In each step zero or more new $W_1$-nodes are added and the process terminates with in $l + 1$ steps. We let $G'_i$ and $\psi_i$ denote the partially constructed $G'$ and $\psi$ after the $i$th step and $Z_i$ denote the new $W_1$-nodes added during this step. By induction $i$, we prove that properties (A) and (B) are satisfied by $G'_i$ and $\psi_i$.

$G'_i$ consists of the single $W_1$-node $\{(t_1, q_1)\}$, $\psi_i(\{(t_1, q_1), (t_1, q_1)\}) = h_1$ and $Z_1 = \{(t_1, q_1)\}$. $G'_i$ and $\psi_i$ trivially satisfy (A) and (B).

Assume $G'_i$ and $\psi_i$ have been defined. Now we define $G_{i+1}$ and $\psi_{i+1}$ as follows. For each $X$ in $Z_i$ we do as follows. Let $\phi$ be a function, that associates, with each $s \in X$, the transition $\eta(\psi_i(s))$, i.e., $\phi(s) = \eta(\psi_i(s))$. Since $\psi_i$ satisfies property (B) and $\eta$ is a view consistent scheduler, it follows that $\phi$ is a one step view consistent scheduler. We add the $W_2$-node $(X, \phi)$ and the edge from $X$ to $(X, \phi)$ to $G_i$; further more for each successor node $Y$ of $(X, \phi)$ in $G_i$, we add $Y$ to $G_i$ if it is not already present in $G_i$; we also add the edge from $(X, \phi)$ to $Y$ to $G_i$. Let $G_{i+1}$ be the graph that results after all the above additions to $G_i$ and $Z_{i+1}$ be the set of $W_1$-nodes added in the above step.

Consider any $W_1$-node $Y$ in $Z_{i+1}$. Let $u = (u_1, u_2, u_{12}, r) \in Y$. We define $\psi_{i+1}(Y, u)$ as follows. There exists a $W_1$-node $X$ in $Z_i$, a $W_2$-node $(X, \phi)$ and an extended state $s \in X$ such that $u$ is the extended state obtained by executing the transition $\phi(s)$. Now consider the history $h' = \psi_i(s)u$. Let $p$ be the length $h$. From lemma 15, we see that $h'$ is a member of some connected component $C$ of $\mathcal{G}_{p,i+1}$. Suppose $v$ is another node in $Y$ and is obtained by executing a transition $\phi(t)$ from another node $t \in X$, then it is easy to see that the history $h'' = \psi_i(t)v$ also belongs to the same connected component $C$. Let $S_1 = \{w_1 : (w_1, w_2, w_{12}, r) \in Y\}$, i.e., $S_1$ is the set of all possible $P_1$-states appearing in $Y$. Similarly, let $S_2$ be the set of all $P_2$-states appearing in $Y$. Let $\delta_1$ be any function that associates with each element in $S_1$ an equivalence class $\delta_1(w_1)$ in $E_{p+1,1}$ that belongs to the connected component $C$ so that $w_1$ is the $P_1$-state in all last states of histories in $\delta_1(w_1)$. Similarly, let $\delta_2$ be any function associating with each $w_2 \in S_2$ an equivalence class $\delta_2(w_2)$ in $E_{p+1,2}$ that belongs to the connected component $C$. Now, $\psi_{i+1}(u)$ is defined to be the unique history in $\delta_1(u_1) \cap \delta_2(u_2)$. Property (B) is trivially satisfied for $Y$ from the way $\psi_{i+1}$ is defined. From the fact that the inputs to the automaton $\text{Spec}$ are the shared states, we see that for every pair of histories $h'_1$ and $h'_2$ each belonging to any possibly distinct equivalence classes of $C$, $\text{Spec}(h'_1) = \text{Spec}(h'_2)$; this is because the projections of all these histories on shared states are equal. Using this and the fact that property (A) holds for $X$, it is not difficult to see that property (A) holds for $Y$ also.
Since the \( \mu \)-calculus formula \( f \) can be evaluated in time linear in the size of \( G \) and since the size of \( G \) is \( O(2^{O(n)}) \), we see that the condition of the lemma can be checked in time \( O(2^{O(n)}) \).