Automatic Verification of Probabilistic Concurrent Finite-State Programs

Moshe Y. Vardi†
Center for Study of Language and Information
Stanford University,

ABSTRACT

The verification problem for probabilistic concurrent finite-state program is to decide whether such a program satisfies its linear temporal logic specification. We describe an automata-theoretic approach, whereby probabilistic quantification over sets of computations is reduced to standard quantification over individual computations. Using new determination construction for ω-automata, we manage to improve the time complexity of the algorithm by two exponentials. The time complexity of the final algorithm is polynomial in the size of the program and doubly exponential in the size of the specification.

1. Introduction

One of the emerging trends in the area of algorithm design is the introduction of randomization into algorithms. Of particular interest to us is the introduction of randomization into protocols for synchronization, communication, and coordination between concurrent processes (cf. [CLP84, FR80, LR81, Ra80, Ra82, Ra83]). Some of these protocols solve problems that have been proven unsolvable by deterministic protocols [FR80, LR81]. Others improve on deterministic protocols by various measures [Ra80, Ra83]).

Unfortunately, designing a correct concurrent protocol is not an easy task. To quote from [OL82]: "There is rather large body of sad experience to indicate that a concurrent program can withstand very careful scrutiny without revealing its errors.". The introduction of randomization compounds the problem, since "intuition often fails to grasp the full intricacy of the algorithm" [PZ84], and "proofs of correctness for probabilistic distributed systems are extremely slippery" [LR81]. While for non-probabilistic concurrent programs there are reasonable verification methods [MP83, OL82], this is not the case for probabilistic programs.

There have been some investigations into the logic of probabilistic concurrent programs [HS84, LS82]. Nevertheless, it is not clear how these logics can be used to actually verify programs. More promising is the work in [HSP83], which gives a complete decision procedure for checking liveness properties of finite-state programs, that is, programs with a fixed number of processes and variables that range over bounded domains. In this case the program can be viewed as a certain cooperation of several finite Markov chains. The work in [HSP83] constitutes, however, only a partial solution to the general problem of verifying probabilistic concurrent programs, since it gives only an isolated proof principle for liveness properties. Pnueli presented a verification method for probabilistic concurrent programs [Pn83], but his method is incomplete and hard to use (cf. [PZ84]).

We extend here the results in [HSP83] by describing a complete decision procedure for verifying probabilistic concurrent finite-state programs. Our starting point is that the correctness criteria for the programs are specified in linear
temporal logic. We allow not only "vanilla" temporal logic [Pn81], but also various extensions of it [GPSS80, LPZ85, Wo83, WVS83]. Our approach is essentially the so-called model checking approach [CES83, EL85, LP85]. Rather than prove correctness as some theorem of the logic, we check that the program is a model of its specification. We view the program as a generator of computations and we view the specification as an acceptor of computations. In the context of non-probabilistic programs, we say that the program is a model of its specification if every computation generated by the program is accepted by the specification (note that the program can have many computations due to concurrency).

We take here an automata-theoretic approach to model-checking, rather the tableau-based approach of [CES83, EL85, LP85]. The basic idea is that infinite computations of finite-state programs can be viewed as infinite words over some finite alphabet. Temporal logic specifications turn out to be essentially finite-state acceptors of infinite words. More precisely, it is shown in [WVS83,SVW85] that given any linear time formula, one can construct an \( \omega \)-automaton (whose size is exponential in the size of the formula), i.e., a finite-state automaton on infinite words [Buc62,Mu63], that accepts precisely the computations that satisfy the formula. It can be shown that, based on this idea, the model checking problem can be reduced to a purely automata-theoretic problem.

In the context of probabilistic programs, the notion of correctness is also probabilistic: a program is correct if almost all computations satisfy the specification, i.e., the specification is satisfied with probability 1 [HS84,LS82]. (Thus this approach differs from previous attempts to tackle probabilistic programs by quantitative methods (cf. [HSP84, Ko85, Fe83]).) Viewed from the automata-theoretic perspective, and ignoring for the moment the effect of concurrency, the verification problem reduces to what we call the probabilistic universality problem. Recall that the standard universality problem is to determine whether a given automaton accepts all words. Here we have stochastic input, and we ask whether the given automaton accepts with probability 1. That is, we have a finite Markov chain that generates infinite words (i.e., computations), and we want to know whether the \( \omega \)-automaton that corresponds to the temporal logic specification accepts these words with probability 1. Alternatively, we can look at the dual problem, which is whether there is a positive probability that a computation does not satisfy the specification. Viewed from the automata-theoretic perspective, the dual problem reduces to what we call the probabilistic emptiness problems. Recall that the standard emptiness problem is to determine whether a given automaton accepts some word. Here we ask whether there is a positive probability that the automaton accepts.

For deterministic automata, we prove that probabilistic quantification ("there exists a set of words of a positive probability") can be reduced to standard quantification ("there exists a word"). In fact, we reduce both probabilistic emptiness and probabilistic universality to the standard emptiness problem. Using this characterization we show that both the probabilistic emptiness and universality problems are in \( O \Delta^P \) in the logarithmic hierarchy of Ruzzo et al. [RST84] (i.e., \( DL^N \)).

Further difficulty arises when the automata are nondeterministic, where our previous techniques are completely inapplicable. In fact, it seems that the only reasonable approach to the problem is to first determinize the given automaton. Unfortunately, unlike the ease for automata on finite words, determining \( \omega \)-automata is no easy task. (We encourage the reader to figure out why the classical subset construction fails for \( \omega \)-automata.) The doubly exponential determinization construction in [McN66] is perhaps the most fundamental result in the theory of \( \omega \)-automata, and it has numerous applications (e.g., [BL69, GH82, Ra69, St82]). The construction has been described as "original and ingenious" [Ch74], and "the most intricate this writer has seen in action" [Buc65]. Unfortunately, the construction is not completely correct. Not only it is "very informal" [Ei74] and "not easily grasped" [Ch74] but also "it contains some inaccuracies which, when properly remedied, make it even more complicated" [Ch74], which explains why "there are those who want to see an error in McNaughton's proof" [Buc83]. In view of the importance of McNaughton's construc-

\footnote{The exponential determinization construction in [ES84] is not general enough for our purposes.}
tion, many authors have taken to supply a rigorous and correct proof of McNaughton’s con-
struction [Buc73, Ei74, Ch74, Ra72, TB73]. Unfor-
nately, in all of these proofs the complexity of the con-
struction is at least triply exponential, rather than doubly exponential as it was in
McN66]. In our context that would mean that
this approach to the probabilistic emptiness prob-
lem yields a decision procedure that runs in at
least doubly exponential space.

We describe here a new doubly exponential
determination construction. While the proof bor-
rows one of McNaughton’s idea, it is rather dif-
fent. Its essence is a generalized subset con-
struction that was introduced in [SVW85]. Our
determination construction is not only more
economical, but it is also more transparent than
the previous construction. Furthermore, using an
intermediate step in the construction, we elimin-
ate the need for a complete determination of the
given automaton. Rather, a “partial” singly
exponential determination is sufficient, yielding
an polynomial space decision procedures for proba-
bilistic emptiness and universality. We also prove
a matching lower bound of PSPACE-hardness.

So far we have dealt with the probabilistic
aspect of the program but not with its concurrent
aspect. Concurrency implies that certain transi-
tions of the program are nondeterministic rather
than probabilistic. We introduce concurrent Mar-
kov chains as models for probabilistic concurrent
programs, and extend our techniques to deal with
concurrency. The bottom line is that we have a
decision procedure for the verification of finite-
state probabilistic concurrent programs, whose
time complexity is linear in the size of the program
and doubly exponential in the size of the specifi-
cation. We discuss the practical significance of
the algorithm at the concluding section of the
paper.

2. Background

2.1. Automata

A (transition) table is a tuple \( \tau = (\Sigma, S, \rho) \),
where \( \Sigma \) is the alphabet, \( S \) is a set of states, and
\( \rho: S \times \Sigma \to 2^S \) is the transition function. \( \tau \) is deter-
mindistic if \( |\rho(s, a)| \leq 1 \) for all states \( s \in S \) and letters
\( a \in \Sigma \). A run of \( \tau \) over a finite word \( w = a_1 \cdots a_n \)
over \( \Sigma \) is a sequence of states \( s = s_0 \cdots s_n \) such
that \( s_{i+1} \in \rho(s_i, a_i) \) for \( 0 \leq i \leq n-1 \). A run of \( \tau \) over an
infinite word \( w = a_1 a_2 \cdots \) is an infinite sequence of states
\( s = s_0, s_1, \cdots \) such that \( s_{i+1} \in \rho(s_i, a_i) \) for
\( i \geq 0 \). For an infinite run \( s \), the set \( \text{inf}(s) \) is the set
of states that repeat infinitely often in \( s \), i.e.,
\( \text{inf}(s) = \{ x \mid |\{ i \in \mathbb{N} \mid x = s_i \}| = \infty \} \).

An automaton consists of a table \( \tau = (\Sigma, S, \rho) \),
a set of starting states \( S_0 \subseteq S \), and a set of accept-
ing states \( F \subseteq F \). The automaton \( \tau = (S_0, F) \) is
deterministic if \( \tau \) is deterministic and \( |S_0| = 1 \). A
accepts a word \( w \) if \( \tau \) has a run \( s \) on \( w \) that starts
with a state in \( S_0 \) and ends with a state in \( F \). The
set of words accepted by \( A \) is \( L(A) \).

An \( \omega \)-automaton, consists of a table
\( \tau = (\Sigma, S, \rho) \), a set of starting states \( S_0 \subseteq S \), and an
acceptance condition. The automaton is deter-
mindistic if \( \tau \) is deterministic and \( |S_0| = 1 \). Various
acceptance conditions give rise to different kinds of
\( \omega \)-automata.

A Büchi acceptance condition is specified by
a set of repeating states [Buc73]. That is, a Büchi
automaton \( A \) is a pair \( (\tau, S_0, F) \), where \( \tau = (\Sigma, S, \rho) \)
is a table, \( S_0 \subseteq S \), and \( F \subseteq S \). A accepts an infinite
word \( w \) if there is a run \( s \) of \( \tau \) on \( w \) such that \( s \)
start with a state in \( S_0 \) and some state in \( F \) repeats
in \( s \) infinitely often, that is, \( \text{inf}(s) \cap F \neq \emptyset \).

A Streett acceptance condition is also a col-
clection of pairs of sets of states [St80]. Intuitively,
a pair \( (L, U) \) means that if some state in \( L \) repeats
infinitely often then some state in \( U \) repeats
infinitely often. Formally, a Streett automaton \( A \)
is a pair \( (\tau, S_0, F) \), where \( \tau = (\Sigma, S, \rho) \) is a table and
and \( F \subseteq \mathbb{S} \). A accepts an infinite word \( w \) if
there is a run \( s \) of \( \tau \) on \( w \) such that \( s \)
start with a state in \( S_0 \) and for all \( (L, U) \in F \), if \( \text{inf}(s) \cap L \neq \emptyset \) then
\( \text{inf}(s) \cap U \neq \emptyset \). Note that Büchi acceptance condi-
tion can be viewed as a special case of Streett
acceptance condition where \( F = \{ (S, F) \} \).

For an \( \omega \)-automaton \( A \), \( L_\omega(A) \) is the set of
infinite words accepted by \( A \). If \( L_\omega(A) = \emptyset \), then \( A \)
is said to be empty. If \( L_\omega(A) = \Sigma^\omega \), then \( A \) is said
to be universal.

Theorem 2.1. [Buc73, Ei74, Ch74, McN66, Ra72,
TB73]

1. Let \( A \) be a Streett automaton. There is a
Büchi automaton \( B \) such that \( L_\omega(A) = L_\omega(B) \).
2. Let $B$ be a Büchi automaton. Then there is a deterministic Streett automaton $A$ such that $L(B) = L(A)$. []

2.2. Temporal Logic

Temporal logic is a formalism in which one can make assertions about the ongoing behavior of a program. The simplicity of the formalism stems from the fact that it does not mention time explicitly but rather implicitly. This makes it rather easy to specify different correctness criteria, such as partial correctness, deadlock freedom, mutual exclusion, liveness, and more [La83, Pn81]. We are interested here in linear time temporal logic rather than branching time temporal logic (cf. [EH83]).

The basic idea is that one can describe a state of the computation by a truth assignment to atomic propositions. These propositions may describe the locations of the program control or the values of program variables. Thus we assume a set $\text{Prop}$ of atomic propositions. A computation is an infinite sequence of truth assignments, i.e., a function $\pi : \mathbb{N} \to 2^{\text{Prop}}$ that assigns truth values to the atomic propositions at each state of the computation. (The restriction to infinite computation is without loss of generality, since a finite computation can be viewed as staying forever in its final state.) We use $\pi^i$ to denote the $i$-th tail of $\pi$, i.e., $\pi^i(k) = \pi(i+k)$.

Temporal logic formulas are built from atomic propositions by means of Boolean and temporal connectives. We describe here the temporal logic of [GPSS80]. Some extensions were studied in [JLPZ85, Wo83, WVS83].

The set of formulas in the logic is built from a set $\text{Prop}$ of atomic propositions by Boolean connectives and the temporal connectives $\mathbb{X}$ ("next") and $\mathbb{U}$ ("until"). The formulas are interpreted over computations in the following way:

- $\models Q$ for $Q \in \text{Prop}$ if $Q \in \pi(0)$.
- $\models \phi \land \psi$ if $\models \phi$ and $\models \psi$.
- $\models \neg \phi$ if not $\models \phi$.
- $\models \mathbb{X} \phi$ if $\pi^1 = \phi$, i.e., $\phi$ holds in a state if $\phi$ holds in the next state.
- $\models \mathbb{U} \psi$ if for some $i \geq 0$ we have that $\pi^i = \psi$ and for all $0 \leq j < i$ we have that $\pi^j = \phi$, i.e., $\phi \mathbb{U} \psi$ holds in a state if $\phi$ holds in all states until $\psi$ holds.

The automata-theoretic approach views computations as infinite words on the alphabet $\Sigma = \mathbb{2}^{\text{Prop}}$. The basic results is that temporal logic formulas are essentially $\omega$-automata.

**Theorem 2.2.** [SVW85, VW85, WVS83]. Given a temporal logic formula $\phi$ of length $n$, one can build a a Büchi automaton $A_\phi$ on the alphabet $2^{\text{Prop}}$, such that $L(A_\phi)$ is precisely the set of computations satisfying $\phi$, and $A_\phi$ has at most $2^{O(n)}$ states. []

We note that for most temporal logics the bound in the theorem can be improved to $2^{O(n)}$ [VW85, WVS83].

2.3. Program Verification

Following [HS83, LS82] we model probabilistic programs by Markov chains. A (labelled) Markov chain $\Pi = (W, P, w_0, \nu)$ over an alphabet $\Sigma$ consists of a state space $W$, an initial state $w_0 \in W$, a transition probability $P : W \times \Sigma \to [0,1]$ such that $\Sigma \ni \nu(w, \alpha) = 1$ for all $w \in W$, and a valuation $V : W \to \Gamma$. For an infinite sequence $w = w_0, w_1, \ldots$ of states, we define $V(w)$ as the infinite word $V(w_0) V(w_1) \cdots$.

As in the theory of Markov processes (see [KSK86]), we now define a probability space called the sequence space $\Psi_{\Pi} = (\Omega, \Delta, \mu)$, where $\Omega = W^\omega$ is the set of all infinite sequences of states starting at $w_0$, $\Delta$ is a Borel field generated by the basic cylindric sets

$$\Delta(w_0, w_1, \ldots, w_n) = \{ w \in \Omega : w = w_0, w_1, \ldots, w_n, \ldots \},$$

and $\mu$ is a probability distribution defined by

$$\mu(\Delta(w_0, w_1, \ldots, w_n)) = \nu(w_0) \nu(w_1) \cdots \nu(w_{n-1}, w_n).$$

A program is a Markov chain over the alphabet $2^{\text{Prop}}$. Thus, if $\omega \in \Gamma$, then $V(\omega)$ is a computation. We now want to define satisfaction of a formula by a program. We first have to show that formulas define measurable sets. The proof of the following proposition uses Theorem 2.1.

**Proposition 2.3.** Let $\Pi = (W, P, w_0, \nu)$ be a Markov process over $\Sigma$ with $\Psi_{\Pi} = (\Omega, \Delta, \mu)$ its associated sequence space, and let $B$ be a Büchi automaton on $\Sigma$. Then the set $\Delta(B) = \{ w : V(w) \in L(B) \}$ is measurable. []

Using now Theorem 2.2, we get the following corollary.
Corollary 2.4. Let $\Pi=(W,P,w_0,V)$ be a program with $\psi_\Pi=(\Omega,\Delta,\mu)$ its associated probability space, let $\phi$ be a formula. Then the set $\Delta(\phi)=\{\omega: V(\omega)\models \phi\}$ is measurable. []

(For the temporal logic defined above, Corollary 2.4 can be proven directly. The proof via Proposition 2.3 is, however, more general and holds for extended temporal logics as well.)

We say that the program $\Pi$ satisfies the formula $\phi$ if $\mu(\Delta(\phi))=1$, that is, if almost all computations of the program $\Pi$ satisfy $\phi$.

In general, the state space of the program can be infinite. But for a large class of applications (in particular synchronization and coordination protocols), the state space is finite, which we assume to be the case from now on. The verification problem is to decide, given a program $\Pi$ and a formula $\phi$, whether $\Pi$ satisfies $\phi$. As we shall see later, our algorithms do not depend on the actual transition probabilities. Thus we take the size of the program to be the number of nonzero entries in the transition matrix.

3. Probabilistic Universality and Emptiness

Theorem 2.2 enables us to consider the verification problem from a purely automata-theoretic perspective. Let $\Pi=(W,P,w_0,V)$ be a finite Markov chain over $\Sigma$, with $\psi_\Pi=(\Omega,\Delta,\mu)$ its associated sequence space, and let $B$ be an $\omega$-automaton on $\Sigma$. Recall that $\Delta(B)$ is the set $\{\omega: V(\omega)\in L_B\}$ of sequences accepted by $B$. We say that $B$ is universal with respect to $\Pi$ if $\mu(\Delta(B))=1$. The probabilistic universality problem is to decide, given $\Pi$ and $B$, whether $B$ is universal with respect to $\Pi$. Clearly, the verification problem is reducible to the probabilistic universality problem. For technical reasons it is also useful to investigate the dual notion. We say that $B$ is empty with respect to $\Pi$ if $\mu(\Delta(B))=0$. The probabilistic emptiness problem is to decide, given $\Pi$ and $B$, whether $B$ is empty with respect to $\Pi$.

The standard emptiness and universality problem for Büchi automata (i.e., does a given Büchi automaton accept some word, or does a given Büchi automaton accept all words) have been studied in [SVW85]. It is shown there that the emptiness problem is NL-complete and the universality problem is PSPACE-complete. The standard emptiness problem for Streett automata has been studied in [EL85], where a quadratic time algorithm for the problem is presented.

3.1. Probabilistic Universality and Emptiness for Deterministic Automata

If we restrict ourselves to deterministic automata, then we can replace probabilistic quantification ("there exists a set of words with positive probability") by standard quantification ("there exists a word").

Theorem 3.1. Probabilistic emptiness and universality of deterministic Streett automata are logspace reducible to standard emptiness of Streett automata.

Sketch of Proof. Given a Markov chain $\Pi=(W,P,w_0,V)$ over $\Sigma$ and deterministic table $\tau=(\Sigma,S,\rho)$, we define a new table over one-letter alphabet $\eta=(\{a\},W\times S,\rho_\Pi)$, where $\rho_\Pi(W\times S,\{a\})\rightarrow W\times S$ is defined by:

$$\rho_\Pi((u,s),a)=\{(v,t) : P(u,v)>0 \text{ and } \rho(s,V(u))=\{t\}\}.$$  

(Note that $\rho_\Pi$ essentially ignores its input.)

We now reduce probabilistic emptiness to standard emptiness. Let $A=(\tau,F,F)$ be a deterministic Streett automaton, we define a new Streett automaton $A_{\Pi}=(\tau_{\Pi},(w_0,\emptyset),G_{\Pi})$, where $\tau_{\Pi}$ is defined as above, and the acceptance condition $G_{\Pi}$ is

$$\{(u,s),(v,t) : (v,t)\in\rho_{\Pi}(\{u,s\},a)\} \cup$$

$$\{(W\times L, W\times U) : (L,U)\in F\}.$$  

Now $A$ is empty with respect to $\Pi$ if $A_{\Pi}$ is empty.

We now reduce probabilistic universality to standard emptiness. Let $A=(\tau,F,F)$ be a deterministic Streett automaton. For each pair of sets $L,U\subseteq 2^S$, we define a new Streett automaton $A_{\Pi,L,U}=(\tau_{\Pi},(w_0,\emptyset),G_{\Pi,L,U})$, where $\tau_{\Pi}$ is defined as above and the acceptance condition $G_{\Pi,L,U}$ is

$$\{(u,s),(v,t) : (v,t)\in\rho_{\Pi}(\{u,s\},a)\} \cup$$

$$\{(W\times S, W\times L),(W\times U,\emptyset)\}.$$  

Now $A$ is universal with respect to $\Pi$ if $A_{\Pi,L,U}$ is empty for all $(L,U)\in F$. []

What we have done is reducing probabilistic quantification to standard quantification over probabilistically fair sequences, that is, sequences where every probabilistic choice is taken infinitely often. (This is closely related to the notion of $\alpha$-fairness. But while they require fairness with
3.2. Determinizing Büchi Automata

Our determination construction is based on the generalized subset construction of [SVW85]. Let \( \tau=(\Sigma, S, \rho) \) be a table, and let \( F \subseteq S \). We construct a determinable table \( \overline{\tau}=(\overline{\Sigma}, \overline{S}, \overline{\rho}) \). \( \overline{\tau} \) captures the behavior of \( \tau \) with respect to \( F \) in the neighborhood. We assume in the following that \( F \) is a regular language.

**Theorem 3.2.** The probabilistic universality and emptiness problems for deterministic Streett automata are in \( \mathsf{DL}^{\mathsf{NL}} \).

**Sketch of Proof.** Recall that \( \mathsf{DL}^{\mathsf{NL}} \) is the class of languages that can be recognized by logspace-bounded oracle Turing machines, with an oracle set that can be recognized by a nondeterministic logspace-bounded Turing machine. (It is the class \( \mathsf{OA}^{\mathsf{L}} \) in the logarithmic hierarchy of Ruzzo et al. [RST84]).

Let \( A=(\tau, s_0, F) \) be a deterministic Streett automaton. Then \( A \) is nonempty if its transition graph has a terminal strongly connected component \( \alpha \) (i.e., \( \alpha \) is a strongly connected component without outgoing edges) such that for all \( (L, U) \in F \) if \( \alpha \) has a node \( (u, s) \) with \( s \in L \) then \( \alpha \) also has a node \( (v, t) \) with \( t \in U \). This condition can be expressed as follows: there exists a node \( x \) such that (1) there exists a node \( y \) such that \( x \) is connected to \( y \), (2) for all nodes \( y \), if \( x \) is connected to \( y \) then \( y \) is connected to \( x \) (so \( x \) is in a terminal strongly connected component) and, (3) for all \( (L, U) \in F \), if \( z \) is connected to a node \( (u, s) \) with \( s \in L \), then \( z \) is also connected to a node \( (v, t) \) with \( t \in U \). Clearly, this can be checked by a logspace-bounded Turing machine with an oracle in \( \mathsf{NL} \). A similar argument holds for probabilistic universality. []

The proof of Theorem 3.1 depends crucially, unfortunately, on the fact that we are dealing with deterministic automata. The automata obtained from temporal logic formulas are, however, non-deterministic. We can use McNaughton's construction (Theorem 2.1) to determine these automata, but, as discussed in §1, the complexity of that construction is apparently triply exponential, yielding a doubly exponential space bound for the probabilistic universality and emptiness problems. In the next section we describe a better determination construction.
Lemma 3.4. We can construct a deterministic Büchi automaton $A_j$ such that $A_j = \{L(A_j) \cap L(A_j)\}^\omega$, and $B_j$ has $3m + 1$ states.

Sketch of Proof. We first need the following fact.

Fact. Let $1 \leq p, q \leq m$. Then there is a unique $1 \leq r \leq m$ such that $L(A_p) \cap L(A_q) \subseteq L(A_r)$.

We abuse notation and denote the unique $r$ for each $p$ and $q$ as $pq$.

$A_j$ is the deterministic Büchi automaton $(\Sigma, T, \theta, p_0, G)$, where $T = \{p_0\} \cup (\Sigma \times \{0, 1, 2\})$, $G = \Sigma \times \{2\}$, and $\theta: T \times \Sigma \rightarrow T$ is defined as follows:

- $\theta(p_0, a) = <\mu(p_0, a), 0>$.
- $\theta(<p_i, 0>, a) = <p_k, 0>$, if $p_k = \mu(p_i, a)$ and $k \neq jk$.
- $\theta(<p_i, 1>, a) = <p_k, 1>$, if $p_k = \mu(p_i, a)$ and $k = jk$.
- $\theta(<p_i, 1>, a) = <p_k, 1>$, if $p_k = \mu(p_i, a)$, $i \neq j$, and $k \neq jk$.
- $\theta(<p_i, 2>, a) = <p_k, 2>$, if $p_k = \mu(p_i, a)$ and $k = jk$.
- $\theta(<p_i, 2>, a) = <p_k, 2>$, if $p_k = \mu(p_i, a)$ and $i = j$.
- $\theta(<p_i, 2>, a) = <p_k, 1>$, if $p_k = \mu(p_i, a)$ and $i \neq j$.
- $\theta(<p_i, 2>, a) = <p_k, 1>$, if $p_k = \mu(p_i, a)$ and $i \neq j$.
- $\theta(<p_i, 2>, a) = <p_k, 1>$, if $p_k = \mu(p_i, a)$ and $i \neq j$.

We now prove two important lemmas about the languages $Y_i$.

Lemma 3.5. $\bigcap_{\omega} Y_i = \Sigma^\omega$. [1]

Lemma 3.5 is proven by a combinatorial analysis (which is a refinement of the analysis in [SVW85]) of infinite runs of $\tau$, and uses Ramsey's Theorem.

Let now $A = \{r, S_0, F\}$ be a Büchi automaton.

Lemma 3.6. For $1 \leq i, j \leq m$, either $Y_i \cap L(A) = \emptyset$ or $Y_i \subseteq L(A)$.

Proof. Consider the languages $Z_i = L(A_i) \cap L(A_j)$. Clearly, $Y_i \subseteq Z_i$. It is shown in [SVW85] that either $Z_i \cap L(A) = \emptyset$ or $Z_i \subseteq L(A)$. The claim follows. [1]

Corollary 3.7.

1. $L(A) = \bigcup \{Y_i | Y_i \subseteq L(A)\}$.

2. $\Sigma^\omega(L(A)) = \bigcup \{Y_i \cap L(A) = \emptyset\}$. [1]

Finally we prove:

Theorem 3.8. Let $B$ be a Büchi automaton. Then we can construct a deterministic Street automata $A_1$ and $A_2$ with $O(\exp^{2^{(n^2)}})$ such that $L(A_1) = L(B)$ and $L(A_2) = \Sigma^\omega(L(A_2))$.

Sketch of Proof. The first step is to construct deterministic automata $A_j$ for the languages $Y_i$. By Lemma 3.4, $Y_i \subseteq L(A_i) \cap L(A_i)$. We can construct deterministic automata for $Y_i$ using Choueka's flag construction [Ch74]. (The flag construction is a construction for running many automata in parallel under a central control.) This construction, which is implicit in [McN86], is exponential in the size of $A_i$. (The flag construction is the only part in out determination construction that is borrowed from McNaughton's construction.) Thus the $A_j$ are doubly exponential in the size of $A$. Now we use Corollary 3.7, and we get $B_1$ and $B_2$ by taking the cross product of the appropriate $A_j$. [1]

3.3. Probabilistic Universality and Emptiness for Nondeterministic Automata

We can now combine Theorems 3.2 and 3.8 to solve the probabilistic universality and emptiness problems. This would yield an exponential space upper bound. Studying carefully the results of §3.2, we observe that it is not really necessary to determinize the given automata. Rather we can use directly Lemma 3.4 and Corollary 3.7 without going through the exponential flag construction of Theorem 3.8.

Theorem 3.9. Let $\Pi = (W, P, w_0, V)$ be a Markov chain over $\Sigma$, and let $A = (r, S_0, F)$ be a Büchi automaton over $\Sigma$. Let $A_i$ and $A_j$, $1 \leq i \leq 4^{n^2}$, the automata described in the previous section.

1. $A$ is nonempty with respect to $\Pi$ if there are some $A_i$, $A_j$, and a finite sequence $w = w_1, \ldots, w_k$ of states in $W$ such that

- $L(A_i) \cap L(A_j) \subseteq L(A)$,
- $P(w_i, w_j) > 0$ for $1 \leq i \leq k - 1$.
- $A_j$ is nonempty with respect to $(W, P, w_k, V)$.
- $V(w) \in L(A_i)$.
2. A is nonuniversal with respect to Π if there are some \(A_i\), \(A_j\), and a finite sequence \(w = w_1, \ldots, w_k\) of states in \(W\) such that
   
   - \((L(A_i) \cap L(A_j)) \cap L_w(A) = \emptyset\),
   - \(P(w_i, w_j) > 0\) for \(1 \leq i \leq k-1\).
   - \(A_j\) is nonempty with respect to \((W, P, w_k, V)\).
   - \(V(w) \in L(A_i)\).

We can now give tight bounds for the probabilistic universality and emptiness problem.

**Theorem 3.10.** The probabilistic universality and emptiness problems for non-deterministic Büchi automata are in \(DL^{NL}\) with respect to the size of the chain and \(PSPACE\)-complete with respect to the size of the automaton.

**Idea of Proof.** To get the upper bounds we combine Theorems 3.2 and Theorem 3.9. To prove the lower bound we use a reduction from the universality problem for automata on finite words, which was shown to be \(PSPACE\)-complete in [MS72]. Given an automaton \(A\), we construct a Büchi automaton \(B\) and a Markov chain \(Π\) such that if \(A\) is universal, then \(B\) is also universal, and in particular it is universal with respect to \(Π\), and if \(A\) is not universal, then \(B\) is empty with respect to \(Π\). Thus the universality problem is reduced to both the probabilistic universality problem and the probabilistic emptiness problem.

4. **Verifying Probabilistic Concurrent Programs**

Theorems 2.2 together with Theorem 3.10 yield a complete solution to the verification problem for probabilistic sequential programs. The space complexity of the algorithm is \(O(\log^2 n)\) in the size of the program and \(O(\exp n^2)\) in the size of the specification.

Unfortunately, we do not believe in the approach of modelling probabilistic concurrent programs by Markov chains (as suggested in [HS83, LS82]). The problem with this model is that it assumes that all transitions of the programs are probabilistic. This is adequate for sequential programs, since a nonprobabilistic transition can be viewed as a transition with probability 1. But for concurrent programs, where many processes are running concurrently, some transitions are, inherently nondeterministic. The nondeterminism arises from two sources. The first source is the processes themselves. First, processor can die and restart at arbitrary times. Furthermore, processes start running certain protocols only when they need to, e.g., when they are trying to use some shared resource, and we do not want to make any probabilistic assumptions about that. The second source of nondeterminism is the asynchronicity of the system; some processes may run much faster than other process. It is convenient to imagine a scheduler, that decide which process is going to perform the next step. Though we do not want to make any probabilistic assumption about the scheduler, we will assume that it is not a pathological one, i.e., it satisfies some fairness condition.

We now describe a model for probabilistic concurrent programs that allows for nondeterminism.

A **concurrent Markov chain** \(Π = (W, N, F, P, w_0, V)\) over an alphabet \(Σ\) consists of a state space \(W\), a set of **nondeterministic** states \(N \subseteq W\), a set of **fair** states \(F \subseteq N\), transition probability \(P: W^2 \to [0,1]\) such that \(Σ F(Σ) = 1\) for all \(w \in W\), a starting state \(w_0 \in W\), and a valuation \(V: W \to Σ\). The idea is that \(W - N\) is the set of states where a probabilistic transition has to be made, \(N\) is the set of states where a nondeterministic transition has to be made, and \(F\) is the set of states where the nondeterminism comes from the fair scheduler. If \(w \in N\), then we interpret \(P(w, v)\) to mean that there is a possible transition from \(w\) to \(v\) if and only if \(P(w, v) > 0\). A concurrent probabilistic program is a concurrent Markov chain over the alphabet \(Σ^{PROP}\). This model is more general than that in [HS83] where a concurrent probabilistic program is modelled by an interleaving of many Markov chains.

To define the sequence of a concurrent Markov chain we need the notion of a scheduler. A scheduler for a concurrent Markov chain \(Π = (W, N, F, P, w_0, V)\) is a function \(σ : W^* \to W\), i.e., a function that assigns a state to each sequence of states that end with a nondeterministic state, such that \(σ(w_1, \ldots, w_n) = w\) only if \(P(w_n, w) > 0\). A sequence \(w = w_1, w_2, \ldots\) is fair if for all states \(w \in F\), if \(\{i : w_i = v\} = \omega\) and \(P(u, v) > 0\), then \(\{i : w_i = v\} = \omega\). That is, if a fair state occurs in the sequence infinitely often, then all possible transitions are taken infinitely often. This notion is called state fairness in [Pn83] and fair choice from states in [QS82].
Let $\Pi=(W,N,F,P,w_0,V)$ be a concurrent Markov chain. A scheduler $\sigma$ for $\Pi$ gives rise to a Markov chain $\Pi_\sigma=(W^xP,w_0V)$, where $w_0(V[w_1,\ldots,w_k]=V[w_k])$, and $PW^x \times W^x \rightarrow [0,1]$ is defined as follows (where $x$ and $y$ are arbitrary members of $W^x$):
- $\overline{P}(zu,zuv)=P(u,v)$ if $u \in W-N$,
- $\overline{P}(zu,zuv)=1$, if $u \in N$ and $\sigma(zu)=v$, and
- $\overline{P}(z,y)=0$, otherwise.

It is easy to verify that $\forall x \in W^x, \exists y \in W^x, \Sigma \ni P(x,y)=1$, so $\Pi_\sigma$ is indeed a Markov chain. Intuitively, $\Pi_\sigma$ describes the behavior of the system under the scheduler $\sigma$. Note that $\Pi_\sigma$ has infinitely many states even when $\Pi$ has finitely many states.

We can now define the sequence space $\Psi_{\Pi,\sigma}=(\Omega, \Delta, \mu_\sigma)$ relative to the scheduler $\sigma$, where $\Omega=W^\omega$, $\Delta$ is the Borel field generated by the cylindric sets
$$\Delta(w_0,w_1,\ldots,w_n)=\{w \in \Omega: w=w_0,w_1,\ldots,w_n,\ldots\},$$
and $\mu_\sigma$ is the probability distribution defined by
$$\mu_\sigma(\Delta(w_0,w_1,\ldots,w_n))=\overline{P}(w_0,w_1)\overline{P}(w_0,w_1,w_2)\cdots\overline{P}(w_0,\cdots,w_{n-1},w_0,\cdots,w_{n-1}w_n).$$

**Lemma 4.1.** The set of fair executions is measurable. [1]

A scheduler is *fair* if the set of fair sequences in the sequence space $\Psi_{\Pi,\sigma}$ has probability 1, i.e., almost all sequences are fair. A probabilistic concurrent program $\Pi$ satisfies a formula $\phi$, if the set of sequences that satisfy $\phi$ in the sequence space $\Psi_{\Pi,\sigma}$ has probability 1 for all fair schedulers $\sigma$ of $\Pi$.

We can now define emptiness and universality with respect to concurrent Markov chains. Let $\Pi=(W,N,F,P,w_0,V)$ be a concurrent Markov chain, and let $B$ be an $\omega$-automaton. Recall that $\Delta(B)$ is the set \{w: V(w) \in L_\omega(B)\} of sequences accepted by $B$. $B$ is universal with respect to $\Pi$ if $\mu_\sigma(\Delta(B))=1$ for each fair scheduler $\sigma$ for $\Pi$. $B$ is empty with respect to $\Pi$ if $\mu_\sigma(\Delta(B))=0$ for each fair scheduler $\sigma$ for $\Pi$.

**Theorem 4.2.** Probabilistic emptiness and universality of deterministic Streett automata with respect to concurrent Markov chain are logspace reducible to standard emptiness of Streett automata.

**Sketch of Proof.** The proof is similar to the proof of Theorem 3.1, so we show only the reduction from probabilistic emptiness to standard emptiness. Given a concurrent Markov chain $\Pi=(W,N,F,P,w_0,V)$ over $\Sigma$ and deterministic table $\tau=(\Sigma, S, \rho)$, we define a new table over one-letter alphabet $\tau_T=((a), W \times S, \rho_T)$, where $\rho_T(W \times S)\times \{a\} \rightarrow W \times S$ is defined by:
$$\rho_T((u,v),a)=\{(v,t): P(u,v)>0 \text{ and } \rho(s,V(u))=\{t\}\}.$$ Let A=$\tau,T,F$ be a deterministic Streett automaton, we define a new Streett automaton $A_T=(\tau_T, (\tau_T, (u,v), a))$, where $\tau_T$ is defined as above, and the acceptance condition $G_T$ is

\[
\{(u,v), \{v,t\}\} \in ((u,v), a) \in \rho_T((u,v), a) \}
\{(u,v) \times S, (v,t) \times S: u \in E \text{ and } P(u,v)>0 \}
\{(W \times L, W \times U) : (L, U) \in F\}.
\]

The above reduction and the quadratic time algorithm for testing emptiness of Streett automata [EL85] yields quadratic time algorithms for probabilistic universality and emptiness. Since the Streett automata produced by the reduction have a special structure, we can do better, though not as good as we did for non-concurrent Markov chain.

**Theorem 4.3.** Probabilistic emptiness and universality of deterministic Streett automata with respect to concurrent Markov chains are solvable in linear time. [1]

To test probabilistic and emptiness for nondeterministic automata we use the construction of Theorem 3.9.

**Theorem 4.4.** Probabilistic universality and emptiness of nondeterministic Buchi automata with respect to concurrent Markov chains are solvable in time that is linear in the size of the chain and exponential in the size of the automaton. [1]

Finally, combining Theorem 2.2 and Theorem 4.4, we get an upper bound for the verification problem.

**Theorem 4.5.** The verification problem for probabilistic concurrent programs can be solved in time polynomial in the size of the program and doubly exponential in the size of the specification. [1]
The only lower bound that we know how to prove for the verification problem is essentially the $PSPACE$-hardness of Theorem 3.10.

5. Concluding Remarks

We have developed an algorithm for the verification of probabilistic concurrent finite-state programs. The time complexity of the algorithm is linear in the size of the program and doubly exponential in the size of the specification. The reader may feel that this complexity renders our algorithm rather impractical. But the truth is that in practice most correctness specifications are rather short (cf. [LR81, OL82, Pa81]). We believe that this fact, together with several heuristics that can be used to improve the running time of the algorithm, may render the algorithm tractable in some applications. Also, it turns out that for certain fragments of temporal logic the time complexity of the algorithm can be improved by one exponential [PZ85, VW85].

Another weakness of the algorithm is that it deals only with finite-state programs. While protocols for distributed systems are often finite state, we usually want to prove their correctness for an arbitrary number of processes. Our algorithm, on the other hand, will work only for a fixed number of processes. It is known that the verification problem for concurrent programs with an arbitrary number of processes is undecidable [AK85]. Nevertheless, we believe that our algorithm will constitute a basic step in any verification method for such protocols.

Finally, we note that we have dealt only with qualitative correctness. One has often quantitative correctness conditions, such as bounded waiting time [Ra80], or real-time response [RS84]. Verification of these conditions requires totally different techniques.

Acknowledgements. I am grateful to Eli Upfal and Pierre Wolper for many stimulating discussions and helpful suggestions. I'd also like to thank Ron Fagin for commenting on a previous draft of this paper.

References


