VERIFYING TEMPORAL PROPERTIES OF FINITE-STATE PROBABILISTIC PROGRAMS

Costas Courcoubetis *Mihalis Yannakakis*

AT&T Bell Laboratories
Murray Hill, NJ 07974

ABSTRACT: We determine the complexity of testing whether a finite state (sequential or concurrent) probabilistic program satisfies its specification expressed in linear temporal logic. For sequential programs we give an exponential time algorithm, and show that the problem is in PSPACE; this improves the previous upper bound by two exponentials and matches the known lower bound. For concurrent programs we show that the problem is complete in double exponential time, improving the previous upper and lower bounds by one exponential each. We also address these questions for specifications described by ω- automata or formulas in extended temporal logic.

1. INTRODUCTION

It has been realized that randomization may lead to algorithms with better complexity than deterministic ones, or even allow for the solution of problems (especially in distributed computation) that cannot be solved deterministically. Verifying properties of these algorithms is in general harder than for the deterministic ones since the effects of randomization are sometimes non-intuitive and difficult to grasp. These observations provide the motivation for the development of formal methods and tools for the verification of probabilistic programs, and explain the flurry of activity in the last several years in the area of probabilistic verification.

In traditional program verification one must first define the *formal specification* of the program. A simple language for specifying temporal requirements on the computations of programs is *temporal logic* introduced by Pnueli [Pn81]. There are two types of temporal logic, linear and branching time. We will be concerned with linear time in this abstract. To verify a program, one specifies the desired properties of the program by a formula in this logic. The program is correct if all its computations satisfy the formula.

In its generality, an algorithmic solution to the verification problem is hopeless. Things become more manageable for finite-state programs, in which the variables range over finite domains [CES83, QS82]. There is a number of communication and synchronization protocols which are in this category. In this case, state properties can be described by atomic propositions, and the specification can be written as a formula in *propositional* temporal logic. Checking that the program is a model of its specification can be done algorithmically.

A number of papers in recent years examine the extension of model checking to probabilistic programs. In the context of probabilistic programs the notion of correctness needs also to become probabilistic: a program is correct if *almost all* computations satisfy the specification, i.e., the specification is satisfied with probability one [HS84, LS82]. Concurrency introduces a degree of nondeterminism, for example, due to the asynchronous execution of processes. In this case, the program is correct if it meets the specifications with probability 1 even under a worst-case scenario [V85, PZ86] (we define formally the model in the next section).

In this paper we show the following results:

1. Testing if a (finite-state) sequential

probabilistic program satisfies its linear temporal logic specification can be done in time exponential in the size of the specification and linear in the size of the program; also in space polynomial in the specification and polylog in the program. The exact probability can be computed in time exponential in the specification and polynomial in the program.

2. Testing if a concurrent probabilistic program satisfies its specification is hard for doubly exponential time, and can be solved in time doubly exponential in the specification and linear in the size of the program.

We note that as a rule, programs tend to be large, and specifications tend to be short. The previously known lower bound for Problem 1 (the sequential case) was PSPACE [V85], and for P. 2 was EXPTIME [VW86]. Thus, P. 1 is PSPACE-complete as in the nonprobabilistic case [CS83]. The previous upper bound was triply exponential time for both problems (double expspace for 1) [V85, V88]. For a restriction of the logic, [PZ86] and [VW86] give different algorithms that run in single exponential time (for both problems). The restricted logic has the same expressive power as the standard temporal logic; the problem is that the known methods for translating from the full to the restricted logic are nonelementary.

There are two basic approaches to model checking: the tableau-based [CES83, LP85] and the automata-theoretic [WVS83]. Neither of them seems convenient for the sequential case; for P.1 we use a different approach. For the concurrent case, we use the automata-theoretic approach. From a formula, one can construct an exponentially larger ω-automaton on infinite words (a so-called Buchi automaton [B62]) which accepts the computations that satisfy the formula. Such automata are strictly more powerful than linear time temporal logic and can also express significant extensions of it [Wo83], at a higher, but still single exponential cost. From a formula f, one can construct an automaton for either f (i.e., accepting the good computations) or for $\neg f$ (accepting the bad computations) at the same cost. It is more convenient to work with the latter automaton, and solve the *probabilistic emptiness* problem: whether the set of computations of the program that are accepted by the automaton has probability 0.

We determine the complexity of this problem for automata: In the sequential case, the problem can be solved in time exponential in the size of the automaton and linear in the size of the program, and also in PSPACE; [V85] shows it is PSPACE-hard. In the concurrent case, the problem can be solved with the same time bound (single exponential), and is complete for EXPTIME. Double exponential algorithms for these problems on automata were given in [V85].

The difficulty of the problem for automata stems from their nondeterminism. If the automaton is deterministic, or even "almost" deterministic (in a well-defined sense, see Section 4.3), the probabilistic emptiness problem can be solved in linear time [VW86]. Determinizing ω- automata is highly nontrivial. There have been several constructions and proofs over the last 20 years [McN66, Buc73, Ch74, Ra72, V85]; they tend to be very complex and increase the size by two or three exponentials. Also, deterministic Buchi automata are strictly less powerfull than nondeterministic, so that one has to go to a more general type of ω- automaton. We show that "almost" deterministic Buchi automata (in the above sense) are equivalent to nondeterministic, and provide a simple singly exponential construction for this. Independently, Safra has developed a *full* determinization construction (of a Rabin-type ω- automaton) appearing in this proceedings [S88]. His construction has complexity $2^{O(n\log n)}$, but from his proof one can extract essentially the same "almost" deterministic Buchi automaton with complexity $2^{O(n)}$ [V88].

The rest of this abstract is organized as follows. In Section 2 we give the necessary definitions (temporal logic, automata, and the model of probabilistic programs). Section 3 examines the verification problem for (the standard) temporal logic and Section 4 for automata and extended temporal logic.

2. BACKGROUND

2.1. Temporal Logic

We review the basic definitions of *linear time propositional temporal logic* (PTL) as defined in [GPSS80, Pn81]. Formulas in PTL are built from a set *Prop* of atomic propositions using Boolean connectives, the unary temporal connective X (next), and the binary temporal connective U (until). The formulaes in PTL are interpreted over *computations*. Informally, a computation is an infinite sequence of states, where in each state every atomic proposition has a truth value , i.e., is true or false. (In this context, finite computations are viewed as staying in their final state.) Formally, a computation is a function $\pi : N \to 2^{Prop}$, which assigns truth values to the elements of *Prop* at each time instant (natural number). Satisfaction of a formula by a computation is defined as follows. For a computation π and time instant $i \in N$, we have that:

- $\pi, i \models p$ for $p \in Prop$ iff $p \in \pi(i)$.
- $\pi, i \models \xi \& \psi$ iff $\pi, i \models \xi$ and $\pi, i \models \psi$.
- $\pi, i \models \neg\phi$ iff not $\pi, i \models \phi$.
- $\pi, i \models X\phi$ iff $\pi, i+1 \models \phi$.
- $\pi, i \models \xi U \psi$ iff for some $j \geq i$, $\pi, j \models \psi$ and for all k, $i \leq k < j$ $\pi, k \models \xi$.
  A computation π *satisfies* a formula ϕ, denoted $\pi \models \phi$, iff $\pi, 0 \models \phi$.

A computation π can be viewed as an infinite word over the alphabet $\Sigma = 2^{Prop}$. Each temporal logic formula f viewed as an acceptor of infinite words, defines a language $L_\omega(f)$ over Σ.

2.2. Automata on Infinite Words

A (transition) table is a tuple $\tau = (\Sigma, S, \rho)$, where Σ is the alphabet, S is a set of states, and $\rho : S \times \Sigma \to 2^S$ is the transition function. A state is *deterministic* if $|\rho(s,a)| \leq 1$ for all letters $a \in \Sigma$; the table τ is deterministic if all the states are. A run of τ over a finite word $w = a_1 \cdots a_n$ over Σ is a sequence of states $s = s_0, \ldots, s_n$ such that $s_{i+1} \in \rho(s_i, a_i)$ for $0 \leq i \leq n-1$. A run of τ 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 $inf(s)$ is the set of states that repeat infinitely often in s, i.e., $inf(s) = \{v \mid |\{i : s_i = v\}| = \infty\}$. For the table τ and a state s, we denote by $det(\tau, s)$ the deterministic table that results from applying the usual subset construction to τ starting from state s.

An ω-automaton A consists of a table $\tau_A = (\Sigma, S, \rho)$, a set of starting sates $S_0 \subseteq S$, and an acceptance condition. The automaton is deterministic if τ is deterministic and $|S_0| = 1$. In the usual automata on finite strings, acceptance of a run is determined by the final state of the run; in the case of ω- automata and infinite runs s, acceptance is determined by its infinity set $inf(s)$. There are several ways in which one can specify which infinity sets are accepting and which ones are not. The simplest one is the Buchi acceptance condition. A Buchi automaton $A = (\tau_A, S_0, F)$, $F \subseteq S$, is an ω-automaton with the following acceptance condition. A accepts an infinite word w if there is a run s of τ_A over w such that s starts with a state in S_0 and some state in F repeats in s infinitely often, that is, $inf(s) \cap F \neq \emptyset$. We define as $L_\omega(A)$ the set of infinite words accepted by the ω-automaton A. Nondeterministic Buchi automata accept exactly the ω- regular languages; deterministic Buchi automata accept a proper subset.

There is an important result relating Buchi automata and temporal logic formulas viewed as language acceptors. In [WVS83] it is proved that given a PTL formula ϕ of length n, one can build a Buchi automaton A_ϕ on the alphabet $\Sigma = 2^{Prop}$, such that $L_\omega(A_\phi)$ is precisely the set $L_\omega(\phi)$, and A_ϕ has at most $2^{O(n)}$ states.

2.3. Probabilistic Programs

Our model of a sequential probabilistic program is a finite state Markov chain as in [HS84, LS82, V85]. A concurrent program is modelled by a concurrent Markov chain as in [V85]. Informally, this is a Markov chain augmented with nondeterministic states. When the chain is at a randomizing state, a transition is chosen randomly according to a probability transition matrix; when the chain is at a nondeterministic state, a transition is chosen by a *scheduler*, possibly depending on the history of the chain. Formally, a *concurrent Markov chain M* consists of a finite set X of states, which is partitioned into a set N of *nondeterministic* and a set R of *randomising states*; a set $A \subseteq X \times X$ of arcs (transitions); an assignment of nonzero probabilities to arcs coming out of randomising states, so that the probabilities on the arcs coming out of each randomizing state sum to 1; an initial probability distribution, again summing to 1. A *scheduler* is a function $u : X^* \times N \to X$ such that for $x_n \in N$, $u(x_1 \cdots x_n) = x_{n+1}$ for some arc $x_n \to x_{n+1}$. For each scheduler u the process X_n, $n = 0, 1, \cdots$, is not any more

Markov since at time n the distribution of X_n depends on the entire past history of the process. (We can view the process as a Markov chain on an infinite state space, namely X^*.) Clearly if N is empty, then a concurrent Markov chain reduces to an ordinary discrete time finite state Markov chain, which models sequential probabilistic programs.

For a scheduler u, let (X^ω, F, μ_u) be the probability space defined on the sequence space of X_n, where F is the σ-field of the Borel sets on X^ω, and μ_u is the induced probability measure on F corresponding to the scheduler u. We associate with each state of the program the set of the atomic propositions in *Prop* which are true in this state by defining the valuation function $V:X \to \Sigma$ where $\Sigma = 2^{Prop}$. Let $(\Sigma^\omega, F, P_{M,u})$ be the probability space defined on the sequence space of the process $V(X_n)$; for sequential programs (usual Markov chains), we omit the subscript u from P. In [V85] it is shown that for any PTL formula f and Buchi automaton A, $L_\omega(f)$, $L_\omega(A)$ are measurable sets in the above probability space.

We say that a sequential probabilistic program satisfies a formula f if $P_M(L_\omega(f)) = 1$. A concurrent probabilistic program satisfies f if for all schedulers u, $P_{M,u}(L_\omega(f)) = 1$.

3. VERIFYING TEMPORAL LOGIC SPECIFICATIONS

3.1. Sequential Probabilistic Programs

We are given a PTL formula f and a Markov chain M modelling a sequential probabilistic program. It is well known that for verification purposes, the exact values of the probabilities are not important. Nevertheless, it is helpful to think in quantitative terms. The algorithm transforms step-by-step the formula and the Markov chain, eliminating one by one the temporal connectives, while preserving the probability of satisfaction of the formula. There are two transformations C_U and C_X corresponding to the two temporal connectives U and X. We will describe below C_U.

Let $\phi U \psi$ be an "innermost" temporal subexpression of f; i.e., a subexpression such that ϕ, ψ are composed of atomic propositions and Boolean connectives. We can evaluate ϕ and ψ on each state of M. The construction C_U produces a new Markov chain M' and a new formula f' as follows.
Construction C_U:

We first partition the states of the Markov chain in the three disjoint subsets $X = X^{YES} \cup X^{NO} \cup X^?$ defined as follows. All states u satisfying ϕ are in X^{YES} and all states satisfying $\neg\phi$ and $\neg\psi$ are in X^{NO}. Viewing the Markov chain as a graph, let H be the subgraph induced on the remaining states satisfying ϕ and $\neg\psi$. A state u of H is placed in X^{NO} iff no successor in H of u has a transition (in M) into a state satisfying property ψ. A state u of H is placed in X^{YES} iff no successor in H of u is in X^{NO} (by the previous rule) or has a transition (in M) into a state satisfying $\neg\phi$ and $\neg\psi$. If a state does not satisfy the above conditions then it is in $X^?$. The interpretation of the above sets is that if a path starts from a state in X^{YES} (X^{NO}) then with probability one (zero) it will satisfy the formula $\phi U \psi$, and if it starts from a state in $X^?$ then both events have nonzero probability.

Let p_{uv} denote the probability of the transition $u \to v$, and let q_v denote the probability that $\phi U \psi$ is satisfied starting from state v (it can be computed from the equations $q_u = \sum_v p_{uv} q_v$ if $u \in X^?$, $q_v = 1$ if $v \in X^{YES}$, $q_v = 0$ if $v \in X^{NO}$).

The new chain M' has a larger state space X' and is defined over the new set of atomic propositions $Prop' = Prop \cup \{\xi\}$ where ξ is a new atomic proposition, as follows.
States of M': For each $u \in X^{YES}$ there is a state (u, ξ) in M'. For each $u \in X^{NO}$ there is a state $(u, \neg\xi)$. For each $u \in X^?$ there are two states (u, ξ), $(u, \neg\xi)$ in M'. A state (u, ξ) satisfies all the atomic propositions that u satisfies including the new atomic proposition ξ and $(u, \neg\xi)$ has similar properties except that it does not satisfy ξ.
Transitions of M': Every transition $u \to v$ in M implies one or two transitions in M'.
1) $u, v \in X^{YES} \cup X^{NO}$. One transition $(u,.) \to (v,.)$ in M' with probability p_{uv}.
2) $u \in X^{YES} \cup X^{NO}$, $v \in X^?$. Two transitions: $(u,.) \to (v, \xi)$ with probability $p_{uv} q_v$ and $(u,.) \to (v, \neg\xi)$ with probability $p_{uv} \bar{q}_v$, where $\bar{q}_v = 1 - q_v$.
3) $u \in X^?$. If $v \in X^?$, we have two transitions: $(u, \xi) \to (v, \xi)$ with probability $p_{uv} q_v / q_u$ and $(u, \neg\xi) \to (v, \neg\xi)$ with probability $p_{uv} \bar{q}_v / \bar{q}_u$. If

$v \in X^{YES}$, we have only the transition $(u,\xi) \rightarrow (v,\xi)$ with probability p_{uv}/q_u, and if $v \in X^{NO}$ we have only the other transition $(u,\neg\xi) \rightarrow (v,\neg\xi)$ with probability $p_{uv}/\bar{q}_u$.
Initial distribution of M': If $u \in X^{YES} \cup X^{NO}$ then $p_0((u,.)) = p_0(u)$. If $u \in X^?$ then $p_0((u,\xi)) = p_0(u)q_u$ and $p_0((u,\neg\xi)) = p_0(u)\bar{q}_u$.

Let f' be the PTL formula with atomic propositions in *Prop'* obtained from f by substituting the new atomic proposition ξ in the place of $\phi U \psi$.

Proposition 3.1: $|M'| \leq 2|M|$ and $P_M(L_\omega(f)) = P_{M'}(L_\omega(f'))$.

Sketch of the Proof: Let g be the mapping of trajectories of M' into trajectories of M by projecting on the first component of the states of M'. Then for any measurable set S of paths of M we get that $P_M(S) = P_{M'}(g^{-1}(S))$. Also $(\phi U \psi \equiv \xi)$ holds at all times at all trajectories of M' with probability one. □

A similar transformation C_X can be defined for the X connective. If f has k temporal operators we can compute $P_M(L_\omega(f))$ as follows. We apply k times the appropriate transformations C_U, C_X in order to get the sequence f^1, M^1, $\cdots$, f^k, M^k, where f^k is a simple propositional formula. Then $P_M(L_\omega(f))$ is the sum of the initial probabilities in M^k over all states satisfying f^k; clearly, it can be computed in time exponential in the size of the formula and polynomial in the size of the program.

For verification purposes, we only need to check that f^k is satisfied by all initial states of M^k (states with nonzero probability in the initial distribution). It is clear from the construction that the underlying graph of M' (the states and the transitions of M') depends only on the graph of M; furthermore, it can be constructed in time linear in the size of M. It follows that the graph of the final chain M^k can be constructed in time $O(2^k |M|)$. It can be seen also that there is a space-efficient algorithm that determines whether a given state exists in the final chain. Summarizing, we have:

Theorem 3.1: We can test if a finite state probabilistic program M satisfies a formula f in time $O(|M| 2^{|f|})$, or in space polynomial in f and polylogarithmic in M. □

The algorithm can be easily extended to handle the extension of temporal logic with *past connectives* [LPZ85] with the same complexity bounds. Also, it can be extended to CTL^* branching time temporal logic [EH83, EL85].

3.2. Concurrent Probabilistic Programs

Given a formula f and a concurrent Markov chain M, we first conctruct a Buchi automaton A for $\neg f$, and then test if there is a scheduler u such that $P_{M,u}(L_\omega(A)) > 0$ as in Section 4.2. The time complexity is doubly exponential in the size of the formula and linear in the size of the program. The following lower bound shows that both steps have an inherent exponential cost.

Theorem 3.2: Determining whether a concurrent probabilistic program satisfies a formula is complete for double exponential time. □

The reduction for the lower bound is from space bounded alternating Turing machines [CKS]. Given an alternating machine T using exponential space and an input x, we construct in polynomial time a formula f and a concurrent probabilistic program M such that T accepts x iff M does not satisfy f. We give a brief, high-level description of the construction. Consider the verification problem as a game between an indifferent probabilistic player P and a purposeful player S (the scheduler); the formula f serves as the referee who checks that S follows the rules and does not cheat, and decides who wins the game. The program consists basically of three nested loops. A pass of the inner loop specifies a tape cell and its contents; a pass of the middle loop specifies a configuration and a choice for the next move; a pass of the outer loop corresponds to a complete computation: a path from the root to an accepting leaf in the tree of computations of the machine T. (All this assumming that S does not cheat.) If a rejecting state is ever reached, then the program exits the loops, and the scheduler loses. Thus, the goal of S is to keep playing the game forever. The probabilistic player only chooses (randomly) the next transition from a universal state, and the scheduler does all the rest of the computation. In an execution of the middle loop, the scheduler goes around the inner loop an exponential number of times specifying the current configuration cell by cell. Then P

specifies randomly the index of a cell, and the scheduler is supposed to reproduce the contents of the relevant cells from the previous configuration to justify its current contents. Of course, the scheduler can try to alter the contents. The main point is that one can construct a polynomial size formula that forces the scheduler to play according to the rules.

4. VERIFYING AUTOMATA SPECIFICATIONS

Let A be a Buchi automaton and M a (concurrent) Markov chain over the alphabet Σ. A first observation is that we can simplify the problem by allowing with no loss of generality that from now on $\Sigma = X$. This holds since we can always replace A with the automaton A' with transition function $\rho'(s,x) = \rho(s,V(x))$ where ρ is the transition function of A. Then for any scheduler u, $P_{M,u}(L_\omega(A)) = P_{M,u}(L_\omega(A'))$. We may also assume with no loss of generality that A has a transition from every state on each letter. Finally, we will assume in the following for simplicity that the chain and the automaton have unique initial states x_0 and s_0 respectively; the extension to the general case is straightforward.

We define for a Buchi automaton $A = (\tau_A, s_0, F)$ and a concurrent Markov chain M the product transition table $\tau_{M\times A} = (X,\ X\times S,\ \rho_{M\times A})$ as follows. $\tau_{M\times A}:(X\times S)\times X \rightarrow 2^{X\times S}$ such that for x_i, x_j, $x_k \in X$, s_i, $s_j \in S$, $(x_k,s_j) \in \rho_{M\times A}((x_i,s_i),\ x_j)$ if $x_k = x_j$, the chain M has an arc $x_i \rightarrow x_j$, and $s_j \in \rho_A(s_i,x_j)$. Observe that when applying the subset construction on $\tau_{M\times A}$ (starting from any state), every state of the resulting deterministic table is a set of pairs (x,s) that have the same first component.

4.1. Sequential Probabilistic Programs

Definition: A state (x,s), $x\in X$, $s\in S$ is *controllably recurrent* if the graph of $det(\tau_{M\times A},(x,s))$ has a bottom strongly connected component containing a state y for which $(x,s)\in y$.

It can be shown that in a strong component of $\tau_{M\times A}$ either all states are controllably recurrent or none is; also, to test if (x,s) is controllably recurrent we only have to determinize the strong component of (x,s).

Let M_A be the Markov chain defined by the transition table $\det(\tau_{M\times A},(x_0,s_0))$ by associating the corresponding transition probabilities of M and having initial state (x_0,s_0).

Proposition 4.1: $P_M(L_\omega(A))$ is equal to the probability that M_A will hit any bottom strongly connected component containing a state y such that y contains some controllably recurrent state (x,f) with $f\in F$. □

Theorem 4.1: We can compute the probability $P_M(L_\omega(A))$ in time exponential in $|A|$ and polynomial in $|M|$. Furthermore, we can determine if this probability is 1 (or 0, or in-between) in time $O(|M|2^{|A|})$, or in space polynomial in $|A|$ and polylogarithmic in $|M|$. □

4.2. Concurrent Probabilistic Programs

For this subsection we assume that A accepts the bad computations. Thus, we want to solve the probabilistic emptiness problem: decide if $P_{M,u}(L_\omega(A)) = 0$ for all schedulers u.

Definition: A state (x,s), $x\in X$, $s\in S$ is *controllably recurrent in a broad sense* if in $det(\tau_{M\times A},(x,s))$ there exist a strongly connected subset C with a state $y\in C$, $(x,s)\in y$, such that
(a) There is no transition out of C corresponding to a probabilistic transition in M, and
(b) There is a finite string $w\in X^*$ with the following properties: the run of $det(\tau_{M\times A},(x,s))$ over w starting at y visits only states in C, and the run of $det(\tau_{M\times A},(x,s))$ on w starting at $\{(x,s)\}$ ends inside C.

Proposition 4.2: There exists a scheduler u for which $P_{M,u}(L_\omega(A)) > 0$ if and only if there exists a (x,f), $x\in X$, $f\in F$, such that (x,f) is controllably recurrent in the broad sense and reachable in $\tau_{M\times A}$ from (x_0,s_0).

Theorem 4.2: The probabilistic emptiness problem for a Buchi automaton A and a concurrent Markov chain M can be solved in time $O(|M|2^{O(|A|)})$, and is complete for exponential time. □

The result can be extended to concurrent Markov chains with fairness conditions as defined in [V85].

4.3. Semideterminizing Buchi Automata

A Buchi automaton is *deterministic in the limit* if the accepting states can only reach deterministic states. Thus, once a run goes through an accepting state, it continues deterministically. As shown in [VW86], the probabilistic emptiness problem for such automata with respect to concurrent Markov chains can be solved in linear time. (This can be seen also from the condition of Proposition 4.2, which can be simplified in this case.)

From a Buchi automaton $A = (\tau_A, s_0, F)$ we construct a new Buchi automaton B that is deterministic in the limit and accepts the same language. If A has n states, then B has at most 4^n states. The automaton B consists of two parts C and D. The first part C is just A itself with all the states nonaccepting and the same initial state s_0, which is also the initial state of B. (We could take also C to be the result of applying the subset construction on A.) The second part D has states corresponding to pairs (P,Q) of sets of states of A. For each $f \in F$ there is an ε transition from state f (in C) to state $(\{f\},\{f\})$ of D. (If desired, ε transitions to a state can be replaced in the usual way by transitions to the appropriate successors.) The accepting states of automaton B are the states of the form (P,P), i.e., with equal first and second components. (We could also take as accepting only those states (P,P) where $P \cap F \neq \emptyset$.) The transitions from a state (P,Q) are as follows. The second component of a pair follows the usual subset construction; that is, on letter a the state (P,Q) goes to a state (P',Q') where $Q' = \rho_A(Q,a) = \cup\{\rho_A(s,a) \mid s \in Q\}$. There are two cases for the first component: if the state is not accepting (i.e., $P \neq Q$) then $P' = \rho_A(P,a) \cup \rho_A(Q \cap F, a)$; if the state is accepting (i.e., $P = Q$) then $P' = \rho_A(Q \cap F, a)$.

Theorem 4.3: The Buchi automata A and B accept the same language. □

Combining the algorithm from [VW86] with this construction, one can derive an alternative exponential algorithm for the probabilistic emptiness problem for automata.

4.4. Extended Temporal Logics

In [Wo83] it was shown that standard temporal logic cannot express certain properties which can be expressed by automata. For this reason, it was extended using automata as temporal connectives. Three such logics were defined in [WVS83]. The most general and powerful of them uses Buchi automata. Let $A = (\tau_A, s_0, F)$ be a Buchi automaton over the alphabet $\Sigma = \{a_1, \ldots, a_n\}$. If $f_1, \ldots, f_n$ are formulas of the extended temporal logic (ETL), then so is $A(f_1, \ldots, f_n)$. Satisfaction of this formula by a computation π at time instant i is defined as follows.
$\pi, i \models A(f_1, \cdots, f_n)$ iff there is an infinite word $w = a_{j_0} a_{j_1} \cdots$ accepted by A, such that $\pi, i+k \models f_{j_k}$ for all $k \geq 0$.

It was shown in [SVW87] that from a formula f of the extended temporal logic, one can build a Buchi automaton of size $2^{O(|f|^2)}$ that accepts the same language, where $|f|$ includes the sizes of the automata appearing in the formula. (In the case of the two other simpler extensions, the size of f appears linearly in the exponent [WVS83].) Thus, by Theorem 4.2, we can test if a concurrent probabilistic program satisfies an ETL formula in time linear in the size of the program and doubly exponential in the square of the formula.

In the case of sequential probabilistic programs, we can combine the techniques of Sections 3.1 and 4.1, and with some more work show:

Theorem 4.4: We can test if a finite state probabilistic program M satisfies an ETL formula f in time $O(|M| 2^{|f|})$, or in space polynomial in f and polylogarithmic in M. □

Acknowledgment. We thank Moshe Vardi for helpful discussions, and for pointing out a simplification in the construction of Section 4.3.

REFERENCES

[Bu62] J. R. Buchi, "On a Decision Method in Restricted Second Order Arithmetic", *Proc. Int'l Congr. Logic. Method and Philos. Sci.*, 1962, pp. 1-12.

[Bu73] J. R. Buchi, "The Monadic Second-Order Theory of ω_1", in *Decidable Theories II,* Lecture Notes in Mathematics Vol. 328, Springer Verlag, 1973, pp. 1-128.

[CES83] E.M. Clarke, E. A. Emerson, A. P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach", *Proc. 10th ACM Symp. on Principles of Programming Languages,* 1983, pp. 117-126.

[Ch74] Y. Choueka, "Theories of Automata on ω-Tapes: A Simplified Approach", *J. Computer and System Sciences,* 8(1974), pp. 117-141.

[CKS81] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer, "Alternation", *J.ACM* 28(1981), pp. 114-133.

[EH83] E. A. Emerson, J. Y. Halpern, " "Sometimes" and "Not Never" Revisited: On Branching vs. Linear Time", *Proc. of 10th ACP Symp. on Principles of Programming Languages,* 1983.

[EL85] E. A. Emerson, C. L. Lei, "Modalities for Model Checking: Branching Time Strikes Back", *Proc. 12th ACM Symp. on Principles of Programming Languages,* 1985, pp. 84-96.

[GPSS80] D. Gabby, A. Pnuelli, S. Shelah, J. Stavi, "On the Temporal Analysis of Fairness", *Proc. of 7th ACP Symp. on Principles of Programming Languages,* 1980, pp. 163-173.

[HS84] S. Hart and M. Sharir, "Probabilistic Temporal Logic for Finite and Bounded Models", *Proc. of 16th ACM STOC,* 1984.

[HSP83] S. Hart, M. Sharir and A. Pnueli, "Termination of Probabilistic Concurrent Programs", *ACM TOPLAS,* 5(1983), pp. 356-380.

[LP85] O. Lichtenstein, A. Pnueli, "Checking that Finite-State Concurrent Programs Satisfy Their Linear Specifications", *Proc. 12th ACM POPL,* 1985, pp. 97-107.

[LS82] D. Lehman and S. Shelah, "Reasoning with Time and Chance", *Information and Control* 53 (1982), pp. 165-198.

[McN66] R. McNaughton, "Testing and Generating Infinite Sequences by a Finite Automaton", *Information and Control,* 9(1966), pp. 521-530.

[PZ86] A. Pnueli and L. Zuck, "Probabilistic Verification by Tableaux", *Proc. 1st Symp. on Logic in Computer Science,* 1986.

[Pn81] A. Pnueli, "The Temporal Logic of Concurrent Programs", *Theoretical Computer Science* 13(1981), pp. 45-60.

[Pn83] A. Pnueli, "On the Extremely Fair Treatment of Probabilistic Algorithms", *Proc. of 15th ACM STOC,* 1983.

[QS82] J. P. Queille, J. Sifakis, "Fairness and Related Properties in Transition Systems", Research Report #292, IMAG, Grenoble, 1982.

[Ra72] M. O. Rabin, "Automata on Infinite Objects and Church's Problem", *Proc. Regional AMS Conf. Series in Math.* 13(1972), pp. 1-22.

[S88] S. Safra, "On the Complexity of ω- automata", this Proceedings, 1988.

[SC85] A.P. Sistla, E. M. Clarke, ""The Complexity of Propositional Linear Temporal Logics", *J.ACM,* 32(1985), pp. 733-749.

[SVW] A. P. Sistla, M. Y. Vardi, P. Wolper, "The Complementation Problem for Buchi Automata with Application to Temporal Logic", *Theoretical Computer Science,* 49(1987), pp. 217-237.

[V85] M. Vardi, "Automatic Verification of Probabilistic Concurrent Finite-State Programs", *Proc. of 26th STOC,* 1985.

[V88] M. Vardi, private communication.

[VW86] M. Vardi and P. Wolper, "An automata-Theoretic Approach to Automatic Program Verification", *Proc. 1st Symp. on Logic in Computer Science,* 1986.

[WVS83] P. Wolper, M. Y. Vardi A. P. Sistla, "Reasoning about Infinite Computation Paths", *Proc. of 24th IEEE Symp. on Foundations of Computer Science,* 1983, pp. 185-194.

[Wo83] P. Wolper, "Temporal Logic Can Be More Expressive", *Information and Control,* 56(1983), pp. 72-99.