24

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 determinization 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*

[†] Research supported by a gift from the System Development Corporation. Address: IBM Almaden Research Center K55/801, 650 Harry Road, San Jose, CA 95120-6099, USA.

0272-5428/85/0000/0327$01.00 © 1985 IEEE

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 *tableaux-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 ω-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 ω-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 Δ_2^L in the logarithmic hierarchy of Ruzzo et al. [RST84] (i.e., DL^{NL}).

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 case for automata on finite words, determinizing ω-automata is no easy task. (We encourage the reader to figure out why the classical ***subset construction*** fails for ω-automata.) The doubly exponential determinization construction in [McN66] is perhaps the most fundamental result in the theory of ω-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-

[†] 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 construction [Buc73, Ei74, Ch74, Ra72, TB73]. Unfortunately, in all of these proofs the complexity of the construction 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 problem yields a decision procedure that runs in at least *doubly exponential space*.

We describe here a new *doubly exponential* determinization construction. While the proof borrows one of McNaughton's idea, it is rather different. Its essence is a *generalized subset construction* that was introduced in [SVW85]. Our determinization 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 eliminate the need for a complete determinization of the given automaton. Rather, a "partial" *singly exponential* determinization is sufficient, yielding an *polynomial space* decision procedures for probabilistic 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 transitions of the program are nondeterministic rather than probabilistic. We introduce ***concurrent Markov 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 specification. 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 Σ is the alphabet, S is a set of states, and $\rho: S\times\Sigma\rightarrow 2^S$ is the transition function. τ is deterministic if $|\rho(s,a)|\leq 1$ for all states $s\in S$ and letters $a\in\Sigma$. A ***run*** of τ over a finite word $w=a_1\cdots a_n$ over Σ is a sequence of states $\mathbf{s}=s_0,\ldots,s_n$ 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 $w=a_1a_2\cdots$ is an infinite sequence of states $\mathbf{s}=s_0,s_1\cdots$ such that $s_{i+1}\in\rho(s_i,a_i)$ for $i\geq 0$. For an infinite run $\mathbf{s}$, the set $inf(\mathbf{s})$ is the set of states that repeat infinitely often in $\mathbf{s}$, i.e., $inf(\mathbf{s})=\{s:|\{i:s_i=s\}|=\infty\}$.

An ***automaton*** consists of a table $\tau=(\Sigma,S,\rho)$, a set of starting states $S_0\subseteq S$, and a set of accepting states $F\subseteq F$. The automaton $A=(\tau,S_0,F)$ is deterministic if τ is ***deterministic*** and $|S_0|=1$. A ***accepts*** a word w if τ has a run $\mathbf{s}$ 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 ***ω-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 ***deterministic*** if τ is deterministic and $|S_0|=1$. Various acceptance conditions give rise to different kinds of ω-automata.

A ***Büchi*** acceptance condition is specified by a set of repeating states [Buc62]. That is, a Büchi automaton A is a pair (τ,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 $\mathbf{s}$ of τ on w such that $\mathbf{s}$ start with a state in S_0 and some state in F repeats in $\mathbf{s}$ infinitely often, that is, $inf(\mathbf{s})\cap F\neq\emptyset$.

A ***Streett*** acceptance condition is also a collection 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 $(\tau,S_0,\mathbf{F})$, where $\tau=(\Sigma,S,\rho)$ is a table and and $F\subseteq(2^S)^2$. A ***accepts*** an infinite word w if there is a run $\mathbf{s}$ of τ on w such that $\mathbf{s}$ starts with a state in S_0 and for all $(L,U)\in F$, if $inf(\mathbf{s})\cap L\neq\emptyset$ then $inf(\mathbf{s})\cap U\neq\emptyset$. Note that Büchi acceptance condition can be viewed as a special case of Streett acceptance condition where $\mathbf{F}=\{(S,F)\}$.

For an ω-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 $L_\omega(B)=L_\omega(A)$. []

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*, *liveliness*, 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 proposition. These propositions may describe the locations of the program control or the values of program variables. Thus we assume a set *Prop* of atomic propositions. A *computation* is an infinite sequence of truth assignments, i.e., a function $\pi:N\rightarrow 2^{Prop}$ 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 π^i to denote the i-th tail of π, 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 [LPZ85, Wo83, WVS83].

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

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

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

Theorem 2.2. [SVW85, VW85, WVS83]. Given a temporal logic formula ϕ of length n, one can build a a Büchi automaton A_ϕ on the alphabet 2^{Prop}, such that $L_\omega(A_\phi)$ is precisely the set of computations satisfying ϕ, and A_ϕ has at most $2^{O(n^2)}$ 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* $\Pi=(W,P,w_0,V)$ over an alphabet Σ consists of a *state space* W, an *initial state* $w_0\in W$, a *transition probability* $P:W^2\rightarrow[0,1]$ such that $\sum_{v\in W}P(u,v)=1$ for all $u\in W$, and a *valuation* $V:W\rightarrow\Sigma$. For an infinite sequence $\mathbf{w}=w_0,w_1,\cdots$ of states, we define $V(\mathbf{w})$ as the infinite word $V(w_0)V(w_1)\cdots$.

As in the theory of Markov processes (see [KSK66]), 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, Δ is a Borel field generated by the *basic cylindric sets*

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

and μ is a probability distribution defined by

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

A *program* is a Markov chain over the alphabet 2^{Prop}. Thus, if $\mathbf{w}\in\Omega$, then $V(\mathbf{w})$ 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 $\Pi=(W,P,w_0,V)$ be a Markov process over Σ with $\Psi_\Pi=(\Omega,\Delta,\mu)$ its associated sequence space, and let B be a Büchi automaton on Σ. Then the set $\Delta(B)=\{\mathbf{w}:V(\mathbf{w})\in L_\omega(B)\}$ 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 ϕ be a formula. Then the set $\Delta(\phi)=\{\mathbf{w} : V(\mathbf{w})\models\phi\}$ 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 Π *satisfies* the formula ϕ if $\mu(\Delta(\phi))=1$, that is, if almost all computations of the program Π satisfy ϕ.

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 Π and a formula ϕ, whether Π satisfies ϕ. 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 ification problem from a purely automata-theoretic perspective. Let $\Pi=(W,P,w_0,V)$ be a finite Markov chain over Σ, with $\Psi_\Pi=(\Omega,\Delta,\mu)$ its associated sequence space, and let B be an ω-automaton on Σ. Recall that $\Delta(B)$ is the set $\{\mathbf{w}: V(\mathbf{w})\in L_\omega(B)\}$ of sequences accepted by B. We say that B is ***universal*** with respect to Π if $\mu(\Delta(B))=1$. The ***probabilistic universality problem*** is to decide, given Π and B, whether B is universal with respect to Π. 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 Π if $\mu(\Delta(B))=0$. The ***probabilistic emptiness problem*** is to decide, given Π and B, whether B is empty with respect to Π.

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 ndard emptiness problem for Streett automata nas 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 Σ and deterministic table $\tau=(\Sigma,S,\rho)$, we define a new table over one-letter alphabet $\tau_\Pi=(\{a\},W\times S,\rho_\Pi)$, where $\rho_\Pi:(W\times S)\times\{a\}\rightarrow 2^{W\times S}$ is defined by:

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

(Note that ρ_Π essentially ignores its input.)

We now reduce probabilistic emptiness to standard emptiness. Let $A=(\tau,s_0,\mathbf{F})$ be a deterministic Streett automaton, we define a new Streett automaton $A_\Pi=(\tau_\Pi,(w_0,s_0),\mathbf{G}_\Pi)$, where τ_Π is defined as above, and the acceptance condition $\mathbf{G}_\Pi$ is

$$(\{(u,s)\},\{v,t\}) : (v,t)\in\rho_\Pi((u,s),a)\}\cup$$
$$\{(W\times L, W\times U) : (L,U)\in\mathbf{F}\}.$$

Now A is empty with respect to Π iff A_Π is empty.

We now reduce probabilistic universality to standard emptiness. Let $A=(\tau,s_0,\mathbf{F})$ be a deterministic Streett automaton, For each pair of sets $L,U\in 2^S$, we define a new Streett automaton $A_{\Pi,L,U}=(\tau_\Pi,(w_0,s_0),\mathbf{G}_{\Pi,L,U})$, where τ_Π is defined as above and the acceptance condition $\mathbf{G}_{\Pi,L,U}$ 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 Π iff $A_{\Pi,L,U}$ is empty for all $(L,U)\in\mathbf{F}$. []

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 α-fairnessin But while they require fairness with respect to *all* regular expressions, we require only fairness with respect to the given automaton.

In general, emptiness of Streett automata can be tested in quadratic time [EL85], so probabilistic universality and emptiness of Street automata can be tested in quadratic time. It turns out that the automata that we get in the reduction of Theorem 3.1 have a special structure, so we can get a better upper bound.

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

Sketch of Proof. Recall that DL^{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 machines. (It is the class $O\Delta_2^L$ in the logarithmic hierarchy of Ruzzo et al. [RST84]).

Let $A=(\tau,s_0,\mathbf{F})$ be a deterministic Streett automaton. Then A_Π is nonempty iff its transition graph has a terminal strongly connected component α (i.e., α is a strongly connected component without outgoing edges) such that for all $(L,U)\in\mathbf{F}$ if α has a node (u,s) with $s\in L$ then α 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 x is connected to a node (u,s) with $s\in L$, then x 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 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, nondeterministic. We can use McNaughton's construction (Theorem 2.1) to determinize 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 determinization construction.

3.2. Determinizing Büchi Automata

Our determinization 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 deterministic table $\overline{\tau}=(\Sigma,\overline{S},\overline{\rho})$. $\overline{\tau}$ captures the behavior of τ with respect to F. The information that we are trying to capture is as follows. Given a finite nonempty finite word x and two states $u,v\in S$:

1. is there a run of t on x starting with u and ending with v?
2. is there a run of A on x starting with u, ending with v, and containing some state in F?

Let $S=\{s_1,\ldots,s_n\}$. Define $S'=S\times\{0,1\}$ and $\underline{S}=(2^{S'})^n$. $\underline{S}$ has m states, denoted $p_1,\ldots,p_m$, where $m=4^{n^2}$. Intuitively, a state in $\underline{S}$ is an n-tuple of sets of states of S labeled by 0 or 1. We need an n-tuple of sets rather then a single set, because we are trying to capture information about runs that can start in any state of S. The label on the state (0 or 1) indicates whether the run contains a state in F. The state set of $\overline{\tau}$ is $\overline{S}=\underline{S}\cup\{p_0\}$, i.e., we add to $\underline{S}$ a special state p_0..

The deterministic transition function $\overline{\rho}:\overline{S}\times\Sigma\rightarrow\overline{S}$ is defined as follows:

- $\overline{\rho}(p_0,a)=<S_1,\ldots,S_n>$, where

  $S_i=\{<u,0>:u\in\rho(s_i,a)\}\cup\{<u,1>:u\in\rho(s_i,a)\cap F\}$.

- $\overline{\rho}(<T_1,\ldots,T_n>,a)=<S_1,\ldots,S_n>$, where

  $S_i=\{<u,0>:u\in\rho(v,a)$ for some $<v,j>\in T_i\}\cup$

  $\{<u,1>:u\in\rho(v,a)$ for some $<v,1>\in T_i\}\cup$

  $\{<u,1>:u\in\rho(v,a)\cap F$ for some $<v,j>\in T_i\}$.

We now define automata on finite words $\overline{A}_i$, and $\overline{A}_{ii}$,$1\leq i\leq m$: $\overline{A}_i=(\overline{\tau},p_0,\{p_i\})$, and $\overline{A}_{ii}=(\overline{\tau},p_i,\{p_i\})$. The following lemma follows immediately from the fact that the $\overline{A}_i$'s are deterministic.

Lemma 3.3. $L(\overline{A}_1),\ldots,L(\overline{A}_m)$ is a partition of Σ^+. []

Consider now the languages $Y_{ij}=L(\overline{A}_i)\cdot(L(\overline{A}_j)\cap L(\overline{A}_{jj}))^\omega$, where $1\leq i,j\leq m$. The crucial fact about Y_{ij} is that it is the product of two languages (one of finite words, and the other of infinite words) that are accepted by deterministic automata of exponential size.

Lemma 3.4. We can construct a deterministic Büchi automaton A_j such that $A_j) = (L(\overline{A_j}) \cap L(\overline{A_{jj}}))^\omega$, and B_j has $3m+1$ states.

Sketch of Proof. We first need the following fact.

Fact. Let $1 \le p, q \le \underline{m}$. Then there is a unique $1 \le r \le m$ such that $L(\overline{A_p}) \cdot L(\overline{A_q}) \subseteq L(\overline{A_r})$.

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 $T = \{p_0\} \cup (\underline{S} \times \{0,1,2\})$, $G = \underline{S} \times \{2\}$, and $\theta: T \times \Sigma \rightarrow T$ is defined as follows:

- $\theta(p_0, a) = <\overline{\rho}(p_0, a), 0>$.
- $\theta(<p_i, 0>, a) = <p_k, 0>$, if $p_k = \overline{\rho}(p_i, a)$ and $k \ne jk$.
- $\theta(<p_i, 0>, a) = <p_k, 1>$, if $p_k = \overline{\rho}(p_i, a)$ and $k = jk$.
- $\theta(<p_i, 1>, a) = <p_k, 1>$, if $p_k = \overline{\rho}(p_i, a)$, $i \ne j$, and $k \ne jk$.
- $\theta(<p_i, 1>, a) = <p_k, 2>$, if $p_k = \overline{\rho}(p_i, a)$, $i \ne j$, and $k = jk$.
- $\theta(<p_i, 1>, a) = <p_k, 1>$, if $p_k = \overline{\rho}(p_0, a)$ and $i = j$.

  $\theta(<p_i, 2>, a) = <p_k, 1>$, if $p_k = \overline{\rho}(p_1, a)$ and $i \ne j$.
- $\theta(<p_i, 2>, a) = <p_k, 1>$, if $p_k = \overline{\rho}(p_0, a)$ and $i = j$. []

We now prove two important lemmas about the languages Y_{ij}.

Lemma 3.5. $\bigcup_{ij} Y_{ij} = \Sigma^\omega$. []

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

Let now $A = (\tau, S_0, F)$ be a Büchi automaton.

Lemma 3.6. For $1 \le i, j \le m$, either $Y_{ij} \cap L_\omega(A) = \emptyset$ or $Y_{ij} \subseteq L_\omega(A)$.

Proof. Consider the languages $Z_{ij} = L(\overline{A_i}) \cdot L(\overline{A_j})^\omega$. Clearly, $Y_{ij} \subseteq Z_{ij}$. It is shown in [SVW85] that either $Z_{ij} \cap L_\omega(A) = \emptyset$ or $Z_{ij} \subseteq L_\omega(A)$. The claim follows. []

Corollary 3.7.

1. $L_\omega(A) = \bigcup \{ Y_{ij} \mid Y_{ij} \subseteq L(A) \}$.
2. $\Sigma^\omega - (L_\omega(A)) = \bigcup \{ Y_{ij} \mid Y_{ij} \cap L(A) = \emptyset \}$. []

Finally we prove:

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

Sketch of Proof. The first step is to construct deterministic automata $\underline{A_{ij}}$ for the languages Y_{ij}. By Lemma 3.4, $Y_{ij} = L(A_i) \cdot L_\omega(A_j)$. We can construct deterministic automata for Y_{ij} 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 [McN66], is exponential in the size of A_j. (The flag construction is the only part in out determinization construction that is borrowed from McNaughton's construction.) Thus the A_{ij} 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_{ij}. []

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 Σ, and let $\underline{A} = (\tau, S_0, F)$ be a Büchi automaton over Σ. Let $\overline{A_i}$ and A_i, $1 \le i \le 4^{n^2}$, the automata described in the previous section.

1. A is nonempty with respect to Π iff there are some $\overline{A_i}$, A_j, and a finite sequence $\mathbf{w} = w_1, \ldots, w_k$ of states in W such that
   - $L(\overline{A_i}) \cdot L_\omega(A_j) \subseteq L_\omega(A)$,
   - $P(w_i, w_j) > 0$ for $1 \le i \le k-1$.
   - A_j is nonempty with respect to (W, P, w_k, V).
   - $V(\mathbf{w}) \in L(\overline{A_i})$.

2. A is nonuniversal with respect to Π iff there are some $\overline{A_i}$, A_j, and a finite sequence $\mathbf{w}=w_1, \ldots, w_k$ of states in W such that
   - $(L(\overline{A_i})\cdot L_\omega(A_j))\cap L_\omega(A)=\emptyset$,
   - $P(w_i, w_j)>0$ for $1\leq i\leq k-1$.
   - A_j is nonempty with respect to (W,P,w_k,V).
   - $V(\mathbf{w})\in L(\overline{A_i})$. []

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* $\Pi=(W,N,F,P,w_0,V)$ 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 $P: W^2\rightarrow[0,1]$ such that $\sum_{v\in W} P(u,v)=1$ for all $u\in W-N$, a starting state $w_0\in W$, and a valuation $V: W\rightarrow\Sigma$. 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 $u\in N$, then we interpret $P(u,v)$ to mean that there is a possible transition from u to v if and only if $P(u,v)>0$. A ***concurrent probabilistic program*** is a concurrent Markov chain over the alphabet 2^{Prop}. This model is more general than that in [HSP83] 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 $\Pi=(W,N,F,P,w_0,V)$ is a function $\sigma: W^*\cdot N\rightarrow W$, i.e., a function that assigns a state to each sequence of states that end with a nondeterministic state, such that $\sigma(w_1, \ldots, w_n)=w$ only if $P(w_n,w)>0$. A sequence $\mathbf{w}=w_1,w_2,\cdots$ is ***fair*** if for all states $u\in F$, if $|\{i: w_i=u\}|=\omega$ 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 σ for Π gives rise to a Markov chain $\Pi_\sigma=(W^*,\bar{P},w_0,\bar{V})$, where $\bar{V}(w_1,\ldots,w_k)=V(w_k)$, and $\bar{P}W^*\times W^*\to[0,1]$ is defined as follows (where x and y are arbitrary members of W^*):

- $\bar{P}(xu,xuv)=P(u,v)$ if $u\in W-N$,
- $\bar{P}(xu,xuv)=1$, if $u\in N$ and $\sigma(xu)=v$, and
- $\bar{P}(x,y)=0$, otherwise.

It is easy to verify that $\sum_{y\in W^*} \bar{P}(x,y)=1$ for all $x\in W^*$, so Π_σ is indeed a Markov chain. Intuitively, Π_σ describes the behavior of the system under the scheduler σ. Note that Π_σ has infinitely many states even when Π has finitely many states.

We can now define the sequence space $\Psi_{\Pi,\sigma}=(\Omega,\Delta,\mu_\sigma)$ relative to the scheduler σ, where $\Omega=W^\omega$, Δ is the Borel field generated by the cylindric sets

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

and μ_σ is the probability distribution defined by

$$\mu_\sigma(\Delta(w_0,w_1,\ldots,w_n)=$$
$$\bar{P}(w_0,w_0w_1)\cdot\bar{P}(w_0w_1,w_0w_1w_2)\cdots$$
$$\bar{P}(w_0\cdots w_{n-1},w_0\cdots w_{n-1}w_n).$$

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

A scheduler is *fair* is 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 Π *satisfies* a formula ϕ, if the set of sequences that satisfy ϕ in the sequence space $\Psi_{\Pi,\sigma}$ has probability 1 for all fair schedulers σ of Π.

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 ω-automaton. Recall that $\Delta(B)$ is the set $\{\mathbf{w}: V(\mathbf{w})\in L_\omega(B)\}$ of sequences accepted by B. B is *universal* with respect to Π if $\mu_\sigma(\Delta(B))=1$ for each fair scheduler σ for Π. B is *empty* with respect to Π if $\mu_\sigma(\Delta(B))=0$ for each fair scheduler σ for Π.

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 Σ and deterministic table $\tau=(\Sigma,S,\rho)$, we define a new table over one-letter alphabet $\tau_\Pi=(\{a\},W\times S,\rho_\Pi)$, where $\rho_\Pi:(W\times S)\times\{a\}\to 2^{W\times S}$ is defined by:

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

Let $A=(\tau,s_0,\mathbf{F})$ be a deterministic Streett automaton, we define a new Streett automaton $A_\Pi=(\tau_\Pi,(w_0,s_0),\mathbf{G}_\Pi)$, where τ_Π is defined as above, and the acceptance condition $\mathbf{G}_\Pi$ is

$$(\{(u,s)\},\{v,t\}) : (v,t)\in\rho_\Pi((u,s),a)\}\cup$$
$$\{(\{u\}\times S,\{v\}\times S) : u\in F \text{ and } P(u,v)>0\}\cup$$
$$\{(W\times L, W\times U) : (L,U)\in\mathbf{F}\}.$$

[]

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. []

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. []

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. []

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, Pn81]). 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 constitutes 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

[AK85] K.R. Apt, D.C. Kozen, *"Limits for Automatic Program Verification"*, IBM Research Report RC 11095, 1985.

[BL69] J.R. Buchi, L.H. Landweber, "Solving Sequential Conditions by Finite-State Strategies", *Trans. AMS* 138(1969), pp. 295-311.

[Buc62] J.R. Buchi, "On a Decision Method in Restricted Second Order Arithmetic", *Proc. Int'l Congr. Logic, Method and Philos. Sci. 1960,* Stanford University Press, 1962, pp. 1-12.

[Buc65] J.R. Buchi, "Decision Methods in the Theory of Ordinals", *Bull. AMS* 71(1965), pp. 767-770.

[Buc73] J.R. Buchi, "The Monadic Second-Order Theory of ω_1", in *Decidable Theories II,* Lecture Notes in Math. - Vol. 328, Springer-Verlag, 1973, pp. 1-128.

[Buc83] J.R. Buchi, "State-Strategies for Games in $F_{\sigma\delta} \cap G_{\delta\sigma}$", *J. Symbolic Logic* 48(1983), pp. 1171-1198.

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

[CLP84] S. Cohen, D. Lehman, A. Pnueli, "Symmetric and Economical Solutions to the Mutual Exclusion Problem in a Distributed System" *Theoretical Computer Science* 34(1984), pp. 215-225.

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

[Ei74] S. Eilenberg, *Automata, Languages, and Machines,* Academic Press, 1974.

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

[ES84] E.A. Emerson, A.P. Sistla, "Deciding Branching Time Logic", *Proc. 16th ACM Symp. on Theory of Computing,* Washington, 1984, pp. 14-24.

[Fe83] Y.A. Feldman, "A Decidable Propositional Probabilistic Dynamic Logic", *Proc. 15th ACM Symp. on Theory of Computing,* Boston, 1983, pp. 298-309.

[FR80] N. Francez, M. Rodeh., "A Distributed Data Type Implemented by Probabilistic Communication Scheme", *Proc. 21st IEEE Symp. on Foundations of Computer Science,* 1980, pp. 373-379.

[GH82] Y. Gurevich, L. Harrington, "Trees, Automata, and Games", *Proc. 14th ACM Symp. on Theory of Computing,* San Francisco, 1982, pp. 60-65.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, J. Stavi: "On the Temporal Analysis of Fairness", *Proc. 7th ACM Symp. on Principles of Programming Languages,* 1980, pp. 163-17173.

[HS84] S. Hart, M. Sharir, "Probabilistic Temporal Logics for Finite and Bounded Models", *Proc. 16th ACM Symp. on Theory of Computing,* Washington, 1984, pp. 1-13.

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

[HSP84] S. Hart, M. Sharir, A. Pnueli, "Verification of Probabilistic Programs", *SIAM J. Computing,* 13(1984), pp. 292-314.

[Ko85] D. Kozen, "Probabilistic PDL", *J. Computer and System Sciences,* 30(1985), pp. 162-178.

[KSK66] J.G. Kemeny, J.L. Snell, A.W. Knapp, *"Denumerable Markov Chains",* D. van Nostrad Company, 1966.

[La83] L. Lamport, "Specifying Concurrent Program Modules", *ACM Trans. on Programming Languages and Systems,* 5(1983), pp. 190-222.

[LP85] O. Lichtenstein, A. Pnueli, "Checking that Finite-State Concurrent Programs Satisfy Their Linear Specifications", *Proc. 12th ACM Symp. on Principles of Programming Languages,* New Orleans, 1985, pp. 97-107.

[LPZ85] O. Lichtenstein, A. Pnueli, L. Zuck, "The Glory of the Past", *Proc. Worskhop on Logics of Programs,* Brooklyn, 1985, Lecture-Notes in Computer Science - Vol. 193, Springer-Verlag, pp. 196-218.

[LR81] D. Lehman, M.O. Rabin, "On the Advantage of Free Choice: A Fully Symmetric and Fully Distributed Solution to the Dining Philosophers Problem", *Proc. 10th ACM Symp. on Principles of Programming Languages,* Williamsburg, 1981, pp. 133-138.

[LS82] D. Lehman, 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

[MP83] Z. Manna, A. Pnueli, "How to Cook a Temporal System for Your Pet Language", *Proc. 10th ACM Symp. on Principles of Programming Languages,* Austin, 1983, pp. 101-114.

[MS72] A.R. Meyer, L.J. Stockmeyer, "The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Time", *Proc. 13th IEEE Symp. on Switching and Automata Theory,* Long Beach, 1972, pp. 125-129.

[Mu63] D.E. Muller, "Infinite Sequences and Finite Machines", *Proc. 4th IEEE Symp. on Switching Circuit Theory and Logical Design", New York, 1963, pp.3-16.*

[OL82] S. Owicki, L. Lamport, "Proving Liveness Properties of Concurrent Programs", *ACM Trans. on Programming Languages and Systems,* 4(1982), pp. 455-495.

[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. 15th ACM Symp. on Theory of Computing,* Boston, 1983, pp. 278-290.

[PZ84] A. Pnueli, L. Zuck, "Verification of Multiprocess Probabilistic Protocols", *Proc. 3rd ACM Symp. on Principles of Distributed Computing,* Vancouver, 1984, pp. 12-27.

[PZ85] A. Pnueli, L. Zuck, *"The Gallant Model-Checker",* Unpublished manuscript, 1985.

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

[Ra69] M.O. Rabin, "Decidability of Second Order Theories and Automata on Infinite Trees", *Trans. AMS,* 141(1969), pp. 1-35.

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

[Ra80] M.O. Rabin, "N-Process Synchronization by 4 logN-valued Shared Variable" *Proc. 21st IEEE Symp. on Foundations of Computer Science,* 1980, pp. 407-410.

[Ra82] M.O. Rabin, "The Choice Coordination Problem", *Acta Informatica* 17(1982), pp. 121-134.

[Ra83] M.O. Rabin, "Randomized Byzantine Generals", *Proc. 24st IEEE Symp. on Foundations of Computer Science,* Tuscon, 1983, pp. 403-409.

[RS84] J.H. Reif, P.G. Spirakis, "Real-Time Synchronization of Interprocess Communication", *ACM Trans. on Programming Languages and Systems,* 6(1984), pp. 215-238.

[RST84] W.L. Ruzzo, J. Simon, M. Tompa, "Space-Bounded Hierarchies and Probabilistic Computations", *J. Computer and System Sciences,* 28(1984), pp. 216-230.

[St82] R.S. Streett, "Propositional Dynamic Logic of Looping and Converse", *Information and Control* 54(1982), pp. 121-141.

[SVW85] A.P. Sistla, M.Y. Vardi, P. Wolper, "The Complementation Problem for Buchi Automata with Applications to Temporal Logic", *Proc. 12th Int. Colloq. on Automata, Languages and Programming,* Nafplion, 1985, Lecture Notes in Computer Science - Vol. 194, Springer-Verlag, pp. 465-474.

[TB73] B.A. Trakhtenbrot, Y.M. Barzdin, *"Finite Automata Behavior and Synthesis",* North-Holland, 1973.

[VW85] M.Y. Vardi, P. Wolper, *"Efficient Verification of Finite-State Programs: An Automata-Theoretic Approach",* Unpublished manuscript, 1985.

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

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