An Automata-Theoretic Approach to Automatic Program Verification

Moshe Y. Vardi

CSLI, Ventura Hall,
Stanford University,
Stanford, CA 94305.

Pierre Wolper

AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974

ABSTRACT

We describe an *automata-theoretic* approach to automatic verification of concurrent finite-state programs by *model checking*. The basic idea underlying this approach is that for any temporal formula we can construct an automaton that accepts precisely the computations that satisfy the formula. The model-checking algorithm that results from this approach is much simpler and cleaner than tableau-based algorithms. We use this approach to extend model-checking to *probabilistic* concurrent finite-state programs.

August 15, 1985

An Automata-Theoretic Approach to Automatic Program Verification

Moshe Y. Vardi

CSLI, Ventura Hall,
Stanford University,
Stanford, CA 94305.

Pierre Wolper

AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974

1. Introduction

While *program verification* was always a desirable, but never an easy task, the advent of *concurrent programming* has made it significantly both more necessary and more difficult. Indeed, the conceptual complexity of concurrency increases the likelihood of the program containing errors. 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 *probabilistic randomization* into algorithms (cf. [FR80, LR81]) 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].

The first step in program verification is to come up with a *formal specification* of the program. One of the more widely used specification languages for concurrent programs is *temporal logic* which was introduced by Pnueli [Pn81] (see the survey in [SM82]). Temporal logic comes in two varieties: linear time and branching time ([EH83, La80]). For simplicity we concentrate here on linear time, though our approach is also applicable to branching time. A linear temporal specification describes the computations of the program, so a program *meets* the specification (is *correct*) if all its computations satisfy the specification.

In the traditional approach to concurrent program verification (cf. [HO83, MP81, OL82, PZ84]) the correctness of the program is expressed as a formula in first-order temporal logic. To prove that the program is correct, one has to prove that the correctness formula is a theorem of a certain deductive system. Constructing this proof is done manually and is usually quite difficult. It often requires an intimate understanding of the program. Furthermore, the only extent of automation that one can hope for, is that the proof be *checked* by a machine.

A different approach was introduced in [CES83, QS82] for *finite-state* programs, i.e., programs in which the variables range over finite domains. The significance of this class follows from the fact that a significant number of the communication and synchronization protocols studied in the literature are in essence finite-

state programs. Since each state is characterized by a finite amount of information, this information can be described by certain *atomic propositions*. This means that a finite-state program can be viewed as a finite *propositional Kripke structure* and that it can be specified using *propositional* temporal logic. Thus, to verify the correctness of the program, one has only to check that the program, viewed as a finite Kripke structure, satisfies (is a model of) the propositional temporal logic specification. This approach is called *verification by model-checking*, and was further studied in [LP85, EL85a, EL85b]. The advantage of the model-checking approach, described in [AK85] as "one of the most exciting development in the theory of program correctness", is that it can be done algorithmically (for a description of an implementation and applications see [CES83, CM83]).

In view of the attractiveness of the model-checking approach, one would like to extend its applicability as much as possible. We are interested here in two extensions. First, we would like to extend the approach to deal with extensions of the standard temporal logic [LPZ85, Wo83, WVS83]. Secondly, we would like to extend the model-checking approach to deal with *probabilistic* programs, since the introduction of probabilistic randomization into algorithms has been shown to be extremely useful [CLP84, FR80, LR81, Ra80, Ra82, Ra83]. Unfortunately, we found that the *tableau-based* model-checking algorithms in the literature (cf. [LP85]) involve the intricacies of the logic at hand and do not make intuitively clear what extensions are possible. On the other hand, we found that an approach based on the connection between propositional temporal logic and formal language theory is much more fruitful.

The connection between propositional temporal logic and formal language theory has been quite extensively studied [GPSS80, Ka68, LPZ85, Pe85, Si83, SVW85, WVS83]. This connection is based on the fact that a computation is essentially an infinite sequence of states. Since every state is completely described by a finite set of atomic propositions, a computation can be viewed as an infinite word over the alphabet of truth assignments to the atomic propositions. One of the most enlightening results in this area is the fact that temporal logic formulas can be viewed as *finite-state acceptors*. More precisely, given any propositional temporal formula, one can construct a finite automaton on infinite words ([Bu62, Mu63]) that accepts precisely the sequences satisfied by the formula [WVS83]. (The inverse construction also exists if one uses one of the extended logics in [WVS83].)

To use the above connection, we view a finite-state program as a *finite-state generator* of infinite words. Thus, if P is the program and ϕ is the specification, then P meets ϕ if every infinite word generated by P, viewed as a finite-state generator, is accepted by ϕ, viewed as a finite-state acceptor. This reduces the model-checking problem to a purely automata-theoretic problem: the problem of determining if the automaton $P \cap \overline{\phi}$ is empty, i.e., if it accepts no word.

There are a number of benefits from this approach. First, we obtain a very simple and clean algorithm for model-checking for linear time temporal logic (compare to the algorithm in [LP85]). This algorithm makes the complexity bounds of [LP85] obvious and even lets us extend them. We can easily show that the space complexity of model checking is polynomial in the size of the specifications and polylogarithmic (in fact $O(\log^2 n)$) in the size of the model. Note that this is quite significant as the programs to which model checking is applied can be very large and using even linear space could make implementation difficult. Another

aspect of model checking that is made much more straightforward is the introduction of a fairness assumption on the execution of the program as is done in [CES83, EL85a, EL85b, LP85].

A second benefit of our approach is that it makes extending model-checking to more expressive temporal logics easy. The standard temporal logic, which consists of the connectives X ("next"), G ("always"), and U ("until"), either cannot express certain properties [Wo83] or cannot express them conveniently [LPZ85]. For that reason, ***extended temporal logics*** was introduced in [Wo83, WVS83], and *past* temporal connectives were introduced in [LPZ85]. To extend our model-checking algorithm to these logics, one only needs to show that given a formula, one can build a finite automaton on infinite strings that accepts the models of the formula. The rest of the algorithm goes unchanged.

For probabilistic programs the situation becomes more complex. In such programs there is a probability measure defined on the set of computations. The notion of correctness now becomes probabilistic: the program is correct if the probability that a computation satisfies the specification is one. However, the automata-theoretic approach is still very useful. A model-checking algorithm for probabilistic programs was obtained using this approach by Vardi [Va85]. The time complexity of his algorithm is, however, doubly exponential in the length of the specification, vs. a singly exponential for non-probabilistic programs [LP85], rendering it quite impractical.

Using a finer analysis we show here that for a somewhat restricted specification language, we can get a model-checking algorithm, which is significantly simpler than the algorithm in [Va85], and has an exponential time complexity in the length of the specification (the time complexity in the size of the program is linear). This complexity is the same as the one obtained in [LP85] in the non-probabilistic case and is considered acceptable, since the specifications tend to be usually quite short. The algorithm is again based on the reduction to the emptiness problem, the only difference with the non-probabilistic case being that the program is augmented to remember some of its history.

2. Temporal Logics and Automata

Linear time propositional temporal logic (PTL) has been defined in a number of publications [GPSS80, Pn81]. We review it briefly and give a full definition in Appendix A. Formulas of PTL are built from a set *Prop* of atomic propositions and are closed under the application of boolean connectives, the unary temporal connective X (next), and the binary temporal connective U (until). PTL is interpreted over ***computations***. A computation is a function $\Pi:\omega\rightarrow 2^{Prop}$, which assigns truth values to the elements of *Prop* at each time instant (natural number). Such computations can also be viewed as infinite words over the alphabet 2^{Prop}. We shall see that the set of computations satisfying a given formula are exactly those accepted by some finite automaton on infinite words.

The type of finite automata on infinite words we consider is the one defined by Büchi [Bu62]. A ***Büchi automaton*** is a tuple $A=(\Sigma,S,\rho,S_0,F)$, where

- Σ is an alphabet,
- S is a set of states,
- $\rho: S \times \Sigma \rightarrow 2^S$ is a nondeterministic transition function,
- $S_0 \subseteq S$ is a set of starting states, and
- $F \subseteq S$ is a set of designated states.

The automaton A is said to be *semi-deterministic* if $|\rho(s,a)| \leq 1$ for each $s \in S$ and $a \in \Sigma$. (If we also have that $|S_0|=1$, then A is *deterministic*.) A *run* of A over a infinite word $w=a_1 a_2 \cdots$, is a sequence $s_0, s_1, \cdots$, where $s_0 \in S_0$ and $s_i \in \rho(s_{i-1}, a_i)$, for all $i \geq 1$. A run $s_0, s_1, \cdots$ is *accepting* if there is some designated state that repeats infinitely often, i.e., for some $s \in F$ there are infinitely many i's such that $s_i = s$. The infinite word w is *accepted* by A if there is an accepting run of A over w. The set of infinite words accepted by A is denoted $L(A)$.

The following theorem establishes the correspondence between PTL and Büchi automata.

Theorem 2.1. [WVS83] Given a PTL formula ϕ, one can build a Büchi automaton $A_\phi = (\Sigma, S, \rho, S_0, F)$, where $\Sigma = 2^{Prop}$ and $|S| \leq 2^{O(|\phi|)}$, such that $L(A_\phi)$ is exactly the set of computations satisfying the formula ϕ. []

The proof of Theorem 2.1 is central to our approach, so we give it in Appendix B. The reader will notice that the construction is actually quite simple.

Theorem 2.1 makes the theory of Büchi automata very relevant to temporal logic. The following theorem states some important results about the *emptiness problem* for Büchi automata, i.e., the problem of determining for a given Büchi automaton A whether A accepts *some* word.

Theorem 2.2.

(1) [EL85a,EL85b] The emptiness problem for Büchi automata is solvable in linear time.

(2) [SVW85] The emptiness problem for Büchi automata is solvable in nondeterministic logarithmic space. []

We shall use Theorems 2.1 and 2.2 to establish upper bounds for model checking.

3. Non-Probabilistic Model Checking

3.1. Propositional Temporal Logic

We are given a finite-state program and a PTL formula that specifies the legal computations of the program. The problem is to check whether all computations of the program are legal. Before going further, let us define these notions more precisely.

A *finite-state program* is a structure of the form $P=(W, s_0, R, V)$, where W is a finite set of states, $s_0 \in W$ is the initial state, $R \subseteq W^2$ is a total accessibility relation, and $V: W \rightarrow 2^{Prop}$ assigns truth values to propositions in $Prop$ for each state in W. Let $\mathbf{u}$ be an infinite sequence $u_0, u_1 \cdots$ of states in W such that $u_0 = s_0$, and $u_i R u_{i+1}$ for all $i \geq 0$. Then the sequence $V(u_0), V(u_1) \cdots$ is a *computation* of P. We will say that P *satisfies* an PTL formula ϕ if all computations of P satisfy ϕ.

Note that a finite state program $P=(W,s_0,R,V)$ can also be viewed as a Büchi automaton $A_P=(\Sigma,W,s_0,\rho,W)$ where $\Sigma=2^{Prop}$ and $s' \in \rho(s,a)$ iff $(s,s')\in R$ and $a=V(s)$. As this automaton has a set of accepting states equal to the whole set of states, any infinite run of the automaton is accepting.

Thus, for a finite-state program P and a PTL formula ϕ, the model checking problem is to verify that all sequences accepted by the automaton A_P satisfy the formula ϕ. By Theorem 2.1, we know that we can build an automaton A_ϕ that accepts exactly the sequences satisfying the formula ϕ. The model checking problem thus reduces to the automata-theoretic problem of checking that all sequences accepted by the automaton A_P are also accepted by the automaton A_ϕ. Equivalently, we need to check that the automaton that accepts $L(A_P)\cap\overline{L(A_\phi)}$ is empty.

First, note that one can build an automaton that accepts the language $\overline{L(A_\phi)}$ by building the automaton $A_{\neg\phi}$. By Theorem 2.1, the number of states in this automaton is $2^{O(|\phi|)}$. To take the intersection of the two automata, one can use the following result:

Lemma 3.1. [Ch74]: Given two Büchi sequential automata $A=(\Sigma,S,s_0,\rho,F)$ and $B=(\Sigma,S',s'_0,\rho',F')$, one can construct an automaton with $2|S|\cdot|S'|$ states that accepts $L(A)\cap L(B)$. []

Note that in our case the set of accepting states in one of the automata is exactly the whole set of states. In this case it can be shown that the resulting automaton will have $|S|\cdot|S'|$ states.

Consequently, we can build an automaton for $L(A_P)\cap\overline{L(A_\phi)}$, which has $|W|\cdot 2^{O(|\phi|)}$ states, that we need to check for emptiness. Using Theorem 2.2 gives us the following two results.

Theorem 3.2.

(1) Checking whether a formula ϕ is satisfied by a finite-state program P can be done in time $O(||P||\cdot 2^{O(|\phi|)})$.

(2) Checking whether a formula ϕ is satisfied by a finite-state program P can be done in space $O((\log||P||+|\phi|)^2)$. []

Part (1) is the result appearing in [LP85]. Note however how much simpler and clearer our algorithm is (we urge the reader to compare). Part (2) refines the result in [SC85], which says that model-checking is PSPACE-complete. Given our automata-theoretic approach, it was reasonably straightforward to obtain it using Theorem 2.2(2). Note also that it is quite significant as the programs to which model checking is applied are often very large. For instance, if the finite-state program is given as a product of small components $(P_1,\cdots P_k)$ (cf. [Ha84]), the model checking can be done without building the product machine, using space $O((\log||P_1||+\cdots+\log||P_k||)^2)$ which is usually much less than the space needed to store the product machine.

Finally, note that if we want to add a fairness assumption to the execution of the program we are checking as is done in [CES83, EL85a, EL85b, LP85], this is also easy to do in our framework. In the context of a finite-state program, most fairness conditions studied in the literature can be viewed as a Büchi style acceptance condition imposed on the program viewed as an automaton. Clearly, our approach also works when the program is a Büchi automaton rather than an automaton without acceptance conditions. We will elaborate on this point in the full paper.

3.2. Temporal Logic with Past Connectives

In [LPZ85], temporal logic is extended with past temporal connectives. This extension does not increase the expressive power of the logic, but it facilitates expressing certain properties. The works in [BK84,Pn84] illustrate the utility of the past extension of temporal logic for *modular* program verification. We consider here two past connectives: Y (previous) and S (since). They are formally defined in Appendix A. Given our automata-theoretic approach, extending the model checking algorithm to PTL with past connectives is straightforward. All we need to do is show that Theorem 2.1 holds also for the extension of *PTL* with past connectives. The proof is a simple extension of the proof of Theorem 2.1 and is given in Appendix B. Extending model-checking to PTL with past connectives was first done in [LP85], where the fact that this extension is possible is called "surprising". Our automata-theoretic approach not only makes such an extension easy and quite obvious, but it also yields the space bound of Theorem 3.2(2).

3.3. Extended Temporal Logic

In [Wo83] it was shown that *PTL* is not as expressive as one would like it to be. For example, one cannot express in *PTL* the following statement: the property p holds in every even state of the computation. To remedy this deficiency, an extended temporal logic that incorporates nondeterministic finite automata as temporal connectives was introduced [Wo83]. In [WVS83], three different versions of this extension were defined and studied further. The difference between the three versions is the type of acceptance conditions used for the finite automata defining the connectives. The three types of acceptance are *finite acceptance* (some prefix is accepted by the standard notion of acceptance for finite words), *looping acceptance* (the automaton has some infinite run over the word) and *repeating acceptance* (the automaton has a Büchi acceptance condition). These acceptance conditions give rise to three logics: ETL_f, ETL_l, and ETL_r, correspondingly. These logics are defined in Appendix A.

Again, all we need to do to obtain a model checking algorithm for these logics is to show that given a formula of the logic, we can build a Büchi automaton accepting the models of the formula. For ETL_f and ETL_l, it was shown in [WVS83] that the exact analogue of Theorem 2.1 holds. Our model checking algorithm is thus applicable to these logics. For ETL_r, the following theorem was shown in [SVW85]:

Theorem 3.3. Given an ETL_r formula ϕ, one can build a Büchi sequential automaton $A_\phi=(\Sigma,S,\rho,S_0,F)$, where $\Sigma=2^{Prop}$ and $|S|\leq O(\exp|\phi|^2)$ such that $L(A_\phi)$ is exactly the set of computation satisfying the formula ϕ. []

Note that the only difference between Theorem 3.3 and Theorem 2.1 is that the automaton built in Theorem 3.3 is exponential in the square of the length of the formula as opposed to exponential in the length of the formula in Theorem 2.1. This makes model checking for ETL_r somewhat less practical. However, we believe that this case shows very well the power of our automata-theoretic approach. It separates the hard part of the model checking algorithm (building the automaton) from the rest and enables us to use immediately whatever results are available for that problem. We challenge anybody to develop a model checking algorithm for ETL_r without using an automata-theoretic approach.

4. Probabilistic Model Checking

4.1. Probabilistic Programs

Following [HS84, 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 function* $P: W^2\to[0,1]$, such that $\sum_{v\in W} P(u,v)=1$ for all $u\in W$, and a *valuation* $V: W\to\Sigma$. 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 *probabilistic program* is a Markov chain over the alphabet 2^{Prop}. Thus, if $\mathbf{w}\in\Omega$, then $V(\mathbf{w})$ is a *computation* of the program. Let ϕ be a formula, and let $\Delta(\phi)$ be the set $\{\mathbf{w} : V(\mathbf{w})\models\phi\}$. It can be shown that $\Delta(\phi)$ is a measurable set. 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. The *probabilistic model-checking* problem is to determine if a probabilistic finite-state program satisfies a given formula.

As we shall see later, our model-checking 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.

4.2. Probabilistic Universality and Emptiness

Theorems 2.1 and 3.3 enable us reduce the probabilistic model-checking algorithm to a purely automata-theoretic problem, as we did for non-probabilistic model checking. 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. Let $\Delta(B)$ be the set $\{\mathbf{w}: V(\mathbf{w})\in L(B)\}$. It can be shown that $\Delta(B)$ is a measurable set. 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 probabilistic model checking is reducible to the probabilistic universality problem. For technical reason 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.

Probabilistic emptiness and universality for ω-automata were studied in [Va85]. It was shown there that both problems are PSPACE-complete, when the complexity is measured relative to the size of the automata. (On the other hand, standard emptiness is NLOGSPACE-complete, while standard universality is PSPACE-complete [SVW85]). Since our construction of automata from formulas (Theorems 2.1 and 3.3) is exponential, reducing probabilistic model-checking to either probabilistic universality or probabilistic

emptiness yields an algorithm that requires exponential space [Va85]. In the following sections we show that for some restricted versions of temporal logic we can improve this upper bound by one exponential.

To achieve this improvement, we have to use a more general definition of ω-automata. A *Streett automaton* is a tuple $A=(\Sigma,S,\rho,S_0,\mathbf{F})$, where Σ, S, ρ, and S_0 are the alphabet, state set, transition function, and starting states, respectively, and $\mathbf{F}\subseteq(2^S)^2$ is a collection of pairs of sets of states [St82]. A run of A is accepting if for each pair $(L,U)\in\mathbf{F}$, if some state in L occurs in the run infinitely often, then some state in U occurs in the run infinitely often. We call this a *Streett acceptance condition.* Note that a Büchi automaton $A=(\Sigma,S,\rho,S_0,F)$ can be describe as a Streett automaton $A=(\Sigma,S,\rho,S_0,(S,F))$. The opposite translation (given a Büchi automaton, construct an equivalent Street automaton) is possible, but at the cost of an exponential increase in the size of the automaton.

4.3. A Restricted Temporal Logic

Let $F\phi$ be a shorthand for the formula $\mathbf{true}U\phi$. Intuitively, $F\phi$ says that *eventually* ϕ has to be true. Let $TL(X,F)$ be the fragment of PTL where X and F are the only temporal connectives allowed. We will develop an efficient probabilistic model checking for $TL(X,F)$. To get the essential idea across, we start by considering the sublanguage $TL(F)$, where the only temporal connective is F.

We want to check, that given a program P, the set of computations of P that satisfy a formula ϕ is of measure 1, or equivalently, that the measure of the set of computations that satisfy $\neg\phi$ is 0. The basic idea underlying probabilistic model checking is to replace probabilistic quantification, i.e., "there exists a set of computations of positive measure", by standard quantification, i.e., "there exists a computation". This can be done if we manage to describe computations that "represent" sets of computations. We now develop this notion of "representation".

Let $w=a_1,a_2\cdots$ be an infinite word over an alphabet Σ. Define $let(w)$ to be the set of letters in w, i.e., $let(w)=\{a\in\Sigma:\exists\, i\geq 1, a=a_i\}$. Define $lim(w)$ to be the set of letters that occur in w infinitely often, i.e., $lim(w)=\{a\in\Sigma:|\{i:a_i=a\}|=\infty\}$. Two infinite words w and w' are *limit equivalent* iff $w=uav$ and $w'=uav'$, where $u\in\Sigma^*$, $a\in\Sigma$ and $let(av)=lim(w)=lim(w')=let(av')$.

Lemma 4.1. [SC85] Let $\Pi,\Pi':\omega\rightarrow 2^{Prop}$ be two time structures such that Π and Π' are limit equivalent, and let ϕ be a formula of $TL(F)$. Then $\Pi,0\models\phi$ if and only if $\Pi',0\models\phi$. []

Lemma 4.1 can be interpreted as saying that a computation "represents" all computations that are limit equivalent to it. To reduce probabilistic quantification to standard quantification we also have to ensure that the computation "represents" a set of positive measure. It turns out that this can be ensured by requiring the computation to be *probabilistically fair*, i.e, whenever a state appears in the computation infinitely often all probabilistic choices are taken infinitely often. The nice thing about this condition is that it can be expressed by a Streett acceptance condition.

Lemma 4.2. Probabilistic model-checking for $TL(F)$ is reducible to the emptiness problem for Streett automata.

Sketch of Proof. As in §3.1, we view the program as a finite automaton. Here we impose on this automaton a Street acceptance condition to ensure that it accepts only probabilistically fair sequences. For a program P, call the resulting automaton A_P. We now have to check that the language $L(A_P) \cap L(A_{\neg\phi})$ is empty. We now combine A_P and $A_{\neg\phi}$ in a construction similar to that of Lemma 3.1 to get a Streett automaton $A_{P,\neg\phi}$, such that P satisfies ϕ iff $A_{P,\neg\phi}$ is empty. []

The emptiness problem for Streett automata was studied in [EL85a,EL85b], where a quadratic time algorithm is given. Thus, one would expect the reduction of Lemma 4.2 to yield an algorithm whose time complexity is quadratic in the size of the program and exponential in the size of the formula. It turns out that the Streett automata generated in the reduction have a special structure that enables us to get better bounds.

Theorem 4.3: Checking whether a $TL(F)$ formula ϕ is satisfied by a probabilistic finite-state program P can be done in time $O(||P||\cdot 2^{O(|\phi|)})$. []

We do not know whether the space bound of Theorem 3.2(2) holds for probabilistic model checking.

We now want to extend these ideas to the logic $TL(X,F)$. Unfortunately, Lemma 4.2 does not hold for $TL(X,F)$. The solution is to generalize our notion of limit to deal with sequences of letters. Let $w = a_1 a_2 \cdots$ be an infinite word over an alphabet Σ. For $0 \leq i < j$, let $w_{i,j}$ denote the finite word $a_i a_{i+1} \cdots a_{j-1}$. Define *k–let*$(w)$ to be the set of k-words in w, i.e., *k–let*$(w) = \{u \in \Sigma^k : \exists\, i \geq 1, w_{i,i+k} = u\}$. Define *k–lim*$(w)$ to be the set of k-words that occur in w infinitely often, i.e., *k–lim*$(w) = \{u \in \Sigma^k : |\{i : w_{i,i+k} = u\}| = \infty\}$. Two infinite words w and w' are *k–limit equivalent* iff $w = uxv$ and $w' = uxv'$, where $u \in \Sigma^*$, $x \in \Sigma^k$ and *k–let*(xv)=*k–lim*(w)=*k–lim*(w')=*k–let*(xv') Note that our previous notion of limit equivalence is really 1*–limit* equivalence.

Lemma 4.4. Let ϕ be a formula of $TL(X,F)$ with k–1 occurrences of the connective X. Let $\Pi, \Pi' : \omega \rightarrow 2^{Prop}$ be two time structures such that Π and Π' are k-limit equivalent, Then $\Pi, 0 \models \phi$ if and only if $\Pi', 0 \models \phi$. []

We see now that our notion of "representation" of a set of computation by a single computation is not an absolute one, but is rather relative to the formula at hand. Again, we need to ensure that the computation represent a set of positive measure, and again the necessary condition is one of "probabilistic fairness". Here, however, we have to require probabilistic fairness with respect to subcomputations of length k. This can be done by augmenting the program so that every states "remembers" the previous computation of length k. Details will be given in the full paper.

Theorem 4.5: Checking whether a $TL(X,F)$ formula f is satisfied by a probabilistic finite-state program P can be done in time $O(||P||\cdot 2^{O(|f|^2)})$. []

Note the increase in the complexity with respect to the length of the formula. We do not know if this bound can be improved.

4.4. Adding Past Connectives

We now consider $TL(X,F)$ extended with the past connectives Y and S. We do not allow, however, future connectives inside past formulas. That is, we do not allow an X or F to appear in the scope of Y or S. We call this fragment *past–*$TL(X,F)$. Though it seems somewhat restricted, this fragment has in fact the same expressive power as PTL [LPZ85]. (Note, however, that the translation in [LPZ85] from PTL to *past–*$TL(X,F)$ has a nonelementary complexity.)

Beforing dealing with *past–*$TL(X,F)$, we examine the pure past fragment $TL(Y,S)$, where only the temporal connectives Y and S are allowed.

Lemma 4.6. Let ϕ be a $TL(Y,S)$ formula. Then the automaton A_ϕ is semi-deterministic. []

While probabilistic emptiness and universality for non-deterministic Büchi automata are PSPACE-complete [Va85], they are much easier for semi-deterministic automata.

Theorem 4.7.

(1) Checking whether a $TL(Y,S)$ formula ϕ is satisfied by a probabilistic finite-state program P can be done in time $O(||P||\cdot 2^{O(|\phi|)})$.

(2) Checking whether a $TL(Y,S)$ formula ϕ is satisfied by a probabilistic finite-state program P can be done in space $O((\log||P||+|\phi|)^2)$. []

Theorem 4.7 suggests that if we can separate the future and past components of *past–*$TL(X,F)$ formulas, then we might be able to extend Theorem 4.5 to *past–*$TL(X,F)$. We show in the full paper that this is indeed the case.

4.5. Probabilistic Concurrent Programs

So far we have viewed programs as Markov chains. This model 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, processors 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 processes. 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.

In [Va85] *concurrent Markov chains* are suggested as a model for probabilistic concurrent programs. 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 $F\subseteq N$, a transition probability function $P: W^2\rightarrow[0,1]$, 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 *probabilistic concurrent program* is a concurrent Markov chain over the alphabet 2^{Prop}. In the full paper we will show how our approach to model checking is easily extensible to probabilistic concurrent programs.

5. Concluding Remarks

We have described an automata-theoretic approach to model checking and its application to probabilistic model checking. We strongly believe, and hope we have convinced the reader, that an automata-theoretic approach to the applications of propositional temporal logic is extremely helpful. This approach summarizes the relevant facts about temporal logic into one proposition: the fact that for any temporal formula ϕ we can construct an automaton A_ϕ that accepts precisely the computations that satisfy ϕ. This has the tremendous advantage of enabling us to separate the problems: the logical problem, which is to build the automaton from the formula, and the automata-theoretic problem, which is to relate the program to the automaton. We cannot stress too much how helpful we have found this separation in understanding and extending model checking. We again urge the reader to compare the clarity of the results obtained using our method to that of the model-checking results obtained using tableau-based approaches.

Our results on probabilistic model checking make a first step towards obtaining usable algorithms for this difficult problem. The two most obvious open questions in this area are the optimality of our upper bounds and whether the results of §4 can be extended to full temporal logic.

References

[AK85] K. R. Apt, D. C. Kozen, *"Limits for Automatic Program Verification"*, IBM Research Report RC11095, 1985.

[BK84] H. Barringer, R. Kuiper, "A Temporal Logic Specification Method Supporting Hierarchical Development", *Proc. NSF/SERC Seminar on Concurrency,* Pittsburgh, 1984.

[Bu62] J.R. Büchi, "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.

[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, 1981, 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.

[CM83] E. M. Clarke, B. Mishra, "Automatic Verification of Asynchronous Circuits", *Proc. Workshop on Logics of Porgrams,* Pittsburgh, 1983, Lecture Notes in Computer Science, vol. 164, Springer-Verlag, pp. 101-115.

[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.

[EL85a] E.A. Emerson, C.L. Lei, "Temporal Model Checking under Generalized Fairness Constraints". *Proc. 18th Hawaii Int'l Conference on System Sciences,* 1985.

[EL85b] E.A. Emerson, C.L. Lei, "Modalities for Model Checking: Branching Time Strikes Back", *Proc. 12th ACM Symp. on Principles of Programming Languages,* New Orleans, 1985.

[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.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah and J. Stavi, "The Temporal Analysis of Fairness", *Proc. 7th ACM Symp. on Principles of Programming Languages,* Las Vegas, 1980, pp. 163-173.

[Ha84] D. Harel, *"Statecharts: A Visual Approach to Complex Systems",* Technical Report CS84-05, Weizmann Inst. of Science, 1984.

[HO83] B. T. Hailpern and S. S. Owicki, "Modular Verification of Computer Communication Protocols", *IEEE Trans. on Comm.,* Vol. COM-31, No. 1, January, 1983, pp. 56-68.

[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.

[Ka68] J. A. W. Kamp, *Tense Logic and the Theory of Linear Order,* Ph.D. Thesis, University of California, Los Angeles, 1968

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

[La80] L. Lamport, ""Sometimes" is Sometines "Not Never"", *Proc. 7th ACM Symp. on Principles of Programming Languages,* 1980, pp. 174-185.

[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.

[LPZ85] O. Lichtenstein, A. Pnueli, L. Zuck, "The Glory of the Past", *Proc. Workshop on Logic 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.

[MP81] Z. Manna, A. Pnueli, "Verification of Concurrent Programs: the temporal framework", in *The Correctness Problem in Computer Science* (R.S. Boyer and J.S. Moore, eds.), Academic Press, 1981, pp. 215-273.

[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.

[Pe85] R. Peikert, "ω-Regular Languages and Propositional Temporal Logic", Report No. 85-01, ETH, Zurich, 1985.

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

[Pn84] A. Pnueli, "In Transition from Global to Modular Temporal Reasoning about Programs", *Proc. Advanced NATO Inst. on Logic and Models for Verification and Specification of Concurrent Systems,* La Colle-Sur-Loupe, 1984.

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

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

[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.

[SC85] A. P. Sistla, E. M. Clarke, "The Complexity of Propositional Linear Temporal Logics", *J. ACM,* vol. 32, no. 3, July 1985, pp. 733-749.

[Si83] A. P. Sistla, *Theoretical Issues in The Design and Verification of Distributed Systems,* Ph.D. Thesis, Harvard University, 1983.

[SM82] R. L. Schwartz, P. M. Melliar-Smith, "From State Machines to Temporal Logic: Specification Methods for Protocol Standards", *IEEE Transactions on Communications,* December 1982.

[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,* Lecture Notes in Computer Science, vol. 194, Springer-Verlag, 1985, pp. 465-474.

[Va85] M. Vardi, "Automatic Verification of Probabilistic Concurrent Finite-State Programs", *Proc. 26th Symp. on Foundations of Computer Science,* Portland, to appear.

[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.

Appendix A: Propositional Temporal Logic

Future Fragment

Formulas of PTL are built from a set of atomic propositions *Prop* and are closed under boolean operations, and the application of the unary temporal connective X (next) and of the binary temporal connective U (until). PTL is interpreted over computations $\Pi:\omega\rightarrow 2^{Prop}$. The function Π can be thought of giving the set of propositions true at each time point $i\in\omega$. For a computation Π and a point $i\in\omega$, we have that:

- $\Pi,i\models p$ for $p\in Prop$ iff $p\in\Pi(i)$
- $\Pi,i\models\xi/\backslash\psi$ 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, $i\leq k<j$ $\Pi,k\models\xi$

We will say that Π *satisfies* a formula ϕ, denoted $\Pi\models\phi$, iff $\Pi,0\models\phi$.

Past Fragment

We consider PTL extended with two past operators: Y (previous) and S (since). The operator Y is unary and is the past analogous of next (Y) and the operator S is binary and is the past analogous of until (U). Their semantics are defined as follows:

- $\Pi,i\models Y\phi$ iff $i>0$ and $\Pi,i-1\models\phi$
- $\Pi,i\models\xi S\psi$ iff for some $0\leq j\leq i$, $\Pi,j\models\psi$ and for all k, $j<k\leq j$ $\Pi,k\models\xi$.

Extended Temporal Logic

We consider the extended temporal logic (ETL) where we have nondeterministic finite-state automata as connectives:

Every nondeterministic finite automaton $A=(\Sigma,S,R,s_0,F)$, where Σ is the input alphabet $\{a_1,\ldots,a_n\}$, S is the set of states, $\rho:S\times\Sigma\rightarrow 2^S$ is the transition relation, $s_0\in S$ is the initial state, and $F\subseteq S$ is a set of accepting states (or a set of designated states, see below), is considered as an n-ary temporal connective. That is, if $\phi_1,\ldots,\phi_n$ are formulas, then so is $A(\phi_1,\ldots,\phi_n)$.

Semantically, we have:

- $\Pi,i\models A(\phi_1,\ldots,\phi_n)$, where $A=(\Sigma,S,\rho,s_0,F)$, iff there there exists an infinite word $w=w_0w_1\cdots$ over Σ, accepted by A, such that for all $j\geq 0$, if w_j is a_k, then $\Pi,i+j\models\phi_k$.

Depending on the notion of acceptance we use for the automaton A, we define the three different versions of the extended temporal logic ETL.

- ETL_f (finite acceptance): an infinite word w is accepted by A if there is a finite run of A on a prefix of w that ends in a state in F.

- ETL_l (looping acceptance): an infinite word w is accepted by A if there is some infinite computation of A on w.
- ETL_r (repeating or Büchi acceptance): an infinite word w is accepted by A if there is some infinite computation of A on w that goes infinitely often through a state in F.

Appendix B: Proofs of Theorem 2.1

The construction uses the notion of the closure of a PTL formula ϕ, denoted $cl(\phi)$ defined as follows:

- $\phi \in cl(\phi)$
- $\xi /\backslash \psi \in cl(\phi) \rightarrow \xi, \psi \in cl(\phi)$
- $\neg\psi \in cl(\phi) \rightarrow \psi \in cl(\phi)$
- $\psi \in cl(\phi)$ not of the form $\neg\xi \rightarrow \neg\psi \in cl(\phi)$
- $X\psi \in cl(\phi) \rightarrow \psi \in cl(\phi)$
- $\xi U\psi \in cl(\phi) \rightarrow \xi, \psi \in cl(\phi)$.

Intuitively, the closure of a formula ϕ is the set of its subformulas and their negation. Note that we have $|cl(\phi)| \leq 2|\phi|$. The Büchi automaton we build for a formula ϕ is taken as the combination of two automata: the *local automaton* and the *eventuality automaton*. The local automaton checks for "local inconsistencies" in the model, i.e., it checks for inconsistencies between consecutive states. The only thing the local automaton does not check is that for *eventuality formulas* (i.e. formulas of the form $\xi U\psi$) a point where ψ is satisfied is indeed eventually reached. This is done by the eventuality automaton.

Constructing the Local Automaton

The local automaton is $L=(2^{cl(\phi)}, N_L, \rho_L, N_\phi, N_L)$. The state set N_L will be the set of all sets u of formulas in $cl(\phi)$ that do not have any propositional inconsistency. Namely they must satisfy the following conditions (we identify a formula ψ with $\neg\neg\psi$):

- $\psi \in u$ iff $\neg\psi \notin u$.
- $\xi /\backslash \psi \in u$ iff $\xi \in u$ and $\psi \in u$.

For the transition relation ρ_L, we have that $v \in \rho_L(u, a)$ iff $a = u$ and:

- $X\psi \in u$ iff $\psi \in v$
- $\xi U\psi \in u$ iff $\psi \in u$ or, $\xi \in u$ and $\xi U\psi \in v$

Finally, the set of starting states N_ϕ consists of all sets u such that $\phi \in u$. The local automaton does not impose any acceptance conditions.

The Eventuality Automaton

Given an PTL formula ϕ, we define the set $e(\phi)$ of its eventualities as the subset of $cl(\phi)$ that contains all formulas of the form $\xi U\psi$ The eventuality automaton is $E=(2^{cl(\phi)}, 2^{e(\phi)}, \rho_E, \{\emptyset\}, \{\emptyset\})$, where for the transition relation ρ_E, we have that $v \in \rho_E(u, a)$ iff:

- $u=\emptyset$ and for all $\xi U\psi \in a$, either $\psi \in a$ or $\xi U\psi \in v$.
- $u\neq\emptyset$ and for all $\xi U\psi \in u$, either $\psi \in a$ or $\xi U\psi \in v$.

Intuitively, the eventuality automaton tries to satisfy the eventualities in the model. When the current state is $\emptyset$, it looks at the model to see which eventualities have to be satisfied. Thereafter, the current state says which eventualities have yet to be satisfied.

Combining the Automata

We now combine the local and eventuality automata to get the ***model automaton***. The model automaton $M=(2^{cl(\phi)}, N_M, \rho_M, N_{M0}, F_M)$ is obtained by taking the cross product of L and E. Its sets of states is $N_M=N_L\times 2^{e(\phi)}$. The transition relation ρ_M is defined as follows: $(w,x)\in\rho_M((u,v),a)$ iff $w\in\rho_L(u,a)$ and $x\in\rho_E(v,a)$. The set of starting states is $N_{M0}=N_\phi\times\{\emptyset\}$, and the set of designated states is $F_M=N_L\times\{\emptyset\}$. Note that $|N_M|\leq 2^{|cl(\phi)|}\times 2^{|e(\phi)|}\leq 2^{3|\phi|}$

The automaton we have constructed, accepts strings over $2^{cl(\phi)}$. However, the models of ϕ are defined by strings over 2^{Prop}. So, the last step of our construction is to take the projection of our automaton on 2^{Prop}. This is done by mapping each element $b\in 2^{cl(\phi)}$ into the element $a\in 2^{Prop}$ such that $b\cap Prop=a$. []

Adding Past Connectives

We need to slightly modify the definition of the local automaton to take into account the past connectives. The change is in the definition of the transition relation and of the initial states. For the transition relation, we need to add the clause that $v\in\rho_L(u,a)$ iff

- $Y\psi\in v$ iff $\psi\in v$, and
- $\xi S\psi\in v$ iff $\psi\in v$ or, $\xi\in v$ and $\xi U\psi\in u$.

And, the set of starting states now consists of all states u such that:

- $\phi\in u$,
- $Y\psi\notin u$, for all $\psi\in cl(\phi)$, and
- $\xi S\psi\in u$ iff $\psi\in u$. []