Verifying ω-Regular Properties of Markov Chains

Doron Bustan[1], Sasha Rubin[2], and Moshe Y. Vardi[1]

[1] Rice University*
[2] The University of Auckland**

Abstract. In this work we focus on model checking of probabilistic models. Probabilistic models are widely used to describe randomized protocols. A Markov chain induces a probability measure on sets of computations. The notion of correctness now becomes probabilistic. We solve here the general problem of linear-time probabilistic model checking with respect to ω-regular specifications. As specification formalism, we use alternating Büchi infinite-word automata, which have emerged recently as a generic specification formalism for developing model checking algorithms. Thus, the problem we solve is: given a Markov chain $\mathcal{M}$ and automaton $\mathcal{A}$, check whether the probability induced by $\mathcal{M}$ of $L(\mathcal{A})$ is one (or compute the probability precisely). We show that these problem can be solved within the same complexity bounds as model checking of Markov chains with respect to LTL formulas. Thus, the additional expressive power comes at no penalty.

1 Introduction

In model checking, we model a system as a transition system $\mathcal{M}$ and a specification as a temporal formula ψ. Then, using formal methods, we check whether $\mathcal{M}$ satisfies ψ [7]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* systems [23, 18, 6, 33]. This derives its significance both from the fact that many synchronization and communication protocols can be modelled as finite-state programs, as well as from the great ease of use of fully algorithmic methods. Looking at model-checking algorithms more closely, we can classify these algorithms according to two criteria. The first criterion is the type of model that we use – nondeterministic or probabilistic. The second criterion is the specification language.

For nondeterministic models and linear temporal logic (LTL), a close and fruitful connection with the theory of automata over infinite words has been developed [32–34]. The basic idea is to associate with each LTL formula a nondeterministic Büchi automaton over infinite words (NBW) that accepts exactly all the computations that satisfy the formula. This enables the reduction of various decision problems, such as satisfiability and model checking, to known automata-theoretic problems, yielding clean and asymptotically optimal algorithms. Furthermore, these reductions are very helpful for

* Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

** For a full version with proofs, see `www.cs.rice.edu/~vardi/papers`.

R. Alur and D.A. Peled (Eds.): CAV 2004, LNCS 3114, pp. 189–201, 2004.
© Springer-Verlag Berlin Heidelberg 2004

implementing temporal-logic based verification algorithms, cf. [14]. This connection to automata theory can also be extended to languages beyond LTL, such as ETL [34] and μTL [30].

In this paper we focus on model checking of probabilistic models. Probabilistic models are widely used to describe randomized protocols, which are often used in distributed protocols [5], communication protocols [8], robotics [27], and more. We use Markov chains as our probabilistic model, cf. [29]. A Markov chain induces a probability measure on sets of computations. The notion of correctness now becomes probabilistic: we say here that a program is correct if the probability that a computation satisfies the specification is one (we also discuss a quantitative notion of correctness, where we compute the probability that a computation satisfies the specification). Early approaches for probabilistic model checking of LTL formulas [19, 29] required determinization of NBW, which involves an additional exponential blow-up over the construction of automata from formulas [24], requiring exponential space complexity, unlike the polynomial space complexity of standard model-checking algorithms for LTL, cf. [28].

The exponential gap for probabilistic model checking was bridged in [9, 10], who provided a polynomial-space algorithm, matching the lower bound of [26]. The algorithm in [9, 10] is specialized to LTL (and ETL) specifications, and proceeds by induction on the structure of the formula. An automata-theoretic account of this algorithm was given in [11]. It is shown there that LTL formulas can be translated to a special type of NBW, which they call *separated automata*. (An NBW is separated if every two states that are located in the same strongly connected component have disjoint languages). As with the standard translation of LTL formulas to NBW, the translation to separated automata is exponential. It is then shown in [11] how to model check Markov chains, in nondeterministic logarithmic space, with respect to separated NBW as complemented specification. This yields a polynomial space upper bound for probabilistic model checking of LTL formulas.

The automata-theoretic framework in [11] is very specifically tailored to LTL. As mentioned earlier contrast, the automata-theoretic framework for model checking nondeterministic models is quite general and can also handle more expressive specification languages such as ETL and μTL. This is not a mere theoretical issue. There has been a major recent emphasis on the development of industrial specification languages. These efforts resulted in a several languages [17, 21, 4, 2], culminating in an industrial standard, PSL 1.01 (`www.accellera.com`). Most of the new languages have the full power of NBW, i.e., they can express all ω-regular languages. Thus, they are strictly more expressive than LTL [35], and, thus, not covered by the framework of [9–11].

In this paper we solve the general problem of probabilistic model checking with respect to ω-regular specifications. As specification formalism, we use alternating Büchi infinite-word automata (ABW); see discussion below. Thus, the problem we solve is: Given a Markov chain $\mathcal{M}$ and an ABW $\mathcal{A}$, check whether the probability induced by $\mathcal{M}$ of $L(\mathcal{A})$ is one (i.e., whether $P_{\mathcal{M}}(L(\mathcal{A})) = 1$). (A more refined problem is to calculate the probability precisely, see the full version.)

The motivation for using ABWs as a specification formalism is derived from recent developments in the area of linear specification languages. First, ABWs have been used as an intermediate formalism between LTL formulas and nondeterministic Büchi word

automata (NBW). As shown in [12, 13], one can exploit the linear translation from LTL formulas to ABWs ([31]) for an early minimization, before the exponential translation to NBW. Second, not only can logics such as ETL and μTL be easily translated to ABW, but also most of the new industrial languages can be translated to ABW. Furthermore, for some of them efficient such translations are known (cf. [1]). Thus, ABW can serve as a generic specification formalism for developing model checking algorithms. Note that applying the techniques of [9, 10] to ABW specifications requires converting them first to NBW at an exponential cost, which we succeed here in avoiding.

We present here an algorithm for model checking of Markov chains, using ABWs as specifications. The space complexity of the algorithm is polylogarithmic in $\mathcal{M}$ and polynomial in $\mathcal{A}$. The linear translation of LTL to ABW implies that this complexity matches the lower bound for this problem.

As in [10, 11], our algorithm uses the subset construction to capture the language of every subset of states of $\mathcal{A}$ (an infinite word w is in the language of a set Q of states if $w \in L(s)$ for every state $s \in Q$ and $w \notin L(s)$ for every $s \notin Q$). While for LTL, a straightforward subset construction suffices, this is not the case for ABW. A key technical innovation of this paper is our use of *two* nondeterministic structures that correspond to the alternating automaton $\mathcal{A}$ to capture the language of every set of automaton states. The first nondeterministic structure is an NBW $\mathcal{A}_f$ called the *full automaton*, and the second a "slim" version of the full automaton without accepting conditions, which we call the *local transition system* T_A. Every state q of $\mathcal{A}_f$ and T_A corresponds to a set Q of states in $\mathcal{A}$. While is possible, however, that a several states of $\mathcal{A}_f$ correspond to the same set of states of $\mathcal{A}$, every state of T_A corresponds to a unique set of states of $\mathcal{A}$. The model-checking algorithm make use of the products G and G_f of the Markov chain M with T_A and A_f, respectively.

2 Preliminaries

2.1 Automata

Definition 1. *A* nondeterministic Büchi word automaton *(NBW) is* $\mathcal{A} = \langle \Sigma, S, S_0, \delta, F \rangle$, *where* Σ *is a finite alphabet,* S *is a finite set of states,* $\delta : S \times \Sigma \to 2^S$ *is a transition function,* $S_0 \subseteq S$ *is a set of initial states, and* $F \subseteq S$ *is a set of accepting states.*

Let $w = w_0, w_1, \ldots$ be an infinite word over Σ. For $i \in \mathbb{N}$, let $w^i = w_i, w_{i+1}, \ldots$ denote the suffix of w from its i'th letter. A sequence $\rho = s_0, s_1, \ldots$ in S^ω is a *run* of $\mathcal{A}$ over an infinite word $w \in \Sigma^\omega$, if $s_0 \in S_0$ and for every $i > 0$, we have $s_{i+1} \in \delta(s_i, w_i)$. We use $inf(\rho)$ to denote the set of states that appear infinitely often in ρ. A run ρ of $\mathcal{A}$ is *accepting* if $inf(\rho) \cap F \neq \emptyset$. An NBW $\mathcal{A}$ accepts a word w if $\mathcal{A}$ has an accepting run over w. We use $L(\mathcal{A})$ to denote the set of words that are accepted by $\mathcal{A}$. For $s \in S$, we denote by $\mathcal{A}^{(s)}$ the automaton $\mathcal{A}$ with a single initial state s. We write $L(s)$ (the language of s) for $L(\mathcal{A}^{(s)})$ when $\mathcal{A}$ is clear from the context.

Before we define an alternating Büchi word automaton, we need the following definition. For a given set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false**. Let $Y \subseteq X$. We say that Y *satisfies* a formula $\theta \in \mathcal{B}^+(X)$

if the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X \setminus Y$ satisfies θ. A tree is a set $X \subseteq \mathbb{N}^*$, such that for $x \in \mathbb{N}^*$ and $n \in \mathbb{N}$, if $xn \in X$ then $x \in X$. We denote the length of x by $|x|$.

An *alternating Büchi word automaton* (ABW) is $\mathcal{A} = \langle \Sigma, S, s^0, \delta, F \rangle$, where Σ, S, and F are as in NBW, $s^0 \in S$ is a single initial state, and $\delta : S \times \Sigma \to \mathcal{B}^+(S)$ is a transition function. A run of $\mathcal{A}$ on an infinite word $w = w_0, w_1, \ldots$ is a (possibly infinite) S-labelled tree τ such that $\tau(\varepsilon) = s^0$ and the following holds: if $|x| = i$, $\tau(x) = s$, and $\delta(s, w_i) = \theta$, then x has k children $x_1, \ldots, x_k$, for some $k \leq |S|$, and $\{\tau(x1), \ldots, \tau(xk)\}$ satisfies θ. The run τ is *accepting* if every infinite branch in τ includes infinitely many labels in F. Note that the run can also have finite branches; if $|x| = i$, $\tau(x) = s$, and $\delta(s, a_i) = \mathbf{true}$, then x need not have children.

An *alternating weak word automaton* (*AWW*) is an ABW such that for every strongly connected component C of the automaton, either $C \subseteq F$ or $C \cap F = \emptyset$. Given two AWW $\mathcal{A}_1$ and $\mathcal{A}_2$, we can construct AWW for $\Sigma^\omega \setminus L(\mathcal{A}_1)$, $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, and $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$, which are linear in their size, relative to $\mathcal{A}_1$ and $\mathcal{A}_2$ [22].

Lemma 1. [16] *Let $\mathcal{A}$ be an ABW. Then there exists an AWW $\mathcal{A}_w$ such that $L(\mathcal{A}) = L(\mathcal{A}_w)$ and the size of $\mathcal{A}_w$ is quadratic in the size of $\mathcal{A}$. Furthermore, $\mathcal{A}_w$ can be constructed in time quadratic in the size of $\mathcal{A}$.*

2.2 Markov Chains

We model probabilistic systems by finite *Markov chains*. The basic intuition is that transitions between states are governed by some probability distribution.

Definition 2. *A Markov chain is a tuple $\mathcal{M} = \langle X, P_T, P_I \rangle$ such that X is a set of states, $P_T : (X \times X) \to [0, 1]$ is a transition probability distribution that assigns to every transition (x_1, x_2) its probability. P_T satisfies that for every $x_1 \in X$ we have $\sum_{x_2 \in X} P_T(x_1, x_2) = 1$. $P_I : X \to [0, 1]$ is an initial probability distribution that satisfies $\sum_{x \in X} P_I(x) = 1$.*

We denote by $\mathcal{M}^{(x)}$ the Markov chain $\mathcal{M}$ with P_I that maps x to 1. Sometimes we consider a Markov chain as a graph $\langle X, E \rangle$ where $(x_1, x_2) \in E$ iff $P_T(x_1, x_2) > 0$. For an alphabet $\Sigma = 2^{AP}$, let $V : X \to \Sigma$ be a labelling function, then each path ρ in X^* or in X^ω in $\mathcal{M}$ is mapped by V to a word w in Σ^* or in Σ^ω respectively. For simplicity we assume that $\Sigma = X$ and that $V(x) = x$ for every $x \in X$, this simplification does not change the complexity of verifying the Markov chain [10]. Note, that every infinite path of $\mathcal{M}$ is a word in X^ω but the converse does not necessarily hold. The probability space on the set of pathes of $\mathcal{M}$ is defined as in [29].

The following property of Markov chains is called *ergodicity* and is proved in [15]. Let $\mathcal{M}$ be a Markov chain, then a path of $\mathcal{M}$, with probability one, enters a bottom strongly connected component (BSCC) K of $\mathcal{M}$, and contains every finite path in K infinitely often. In other words, let L_e be the set of infinite words of $\mathcal{M}$ such that every word w in L_e has a suffix that is contained in a BSCC K of $\mathcal{M}$, and contains every finite path in K infinitely often. Then, $P_M(L_e) = 1$.

3 The Full Automaton and the Local Transition System

In this section we capture the behavior of an AWW $\mathcal{A}$ using two nondeterminstic systems. First we define the full automaton, which captures the languages of subsets of the states of $\mathcal{A}$. Then we define the local transition system, which captures the local relations between subsets of states of $\mathcal{A}$.

3.1 The Full Automaton

Given a AWW $\mathcal{A} = \langle \Sigma, S, \delta, F \rangle$ (we ignore its initial state), we define its dual AWW $\hat{\mathcal{A}} = \langle \Sigma, S, \hat{\delta}, \hat{F} \rangle$, where the Boolean formula $\hat{\delta}(s, \sigma)$ is obtained from $\delta(s, \sigma)$ by replacing every **true** with **false** and vice versa, and every $\vee$ with $\wedge$ and vice versa, in addition we define $\hat{F} = S \setminus F$. It is easy to see that $\hat{\mathcal{A}}$ is an AWW.

Lemma 2. [22] *Let $\mathcal{A}$ be an AWW and $\hat{\mathcal{A}}$ be its dual AWW. For every state s we have that $L(\mathcal{A}^{(s)}) = \Sigma^\omega \setminus L(\hat{\mathcal{A}}^{(s)})$.*

Given an AWW $\mathcal{A}$ and its dual AWW $\hat{\mathcal{A}}$ we define the state space of the full automaton as a subset of $2^S \times 2^S \times 2^S \times 2^S$. We start with the following definition.

Definition 3. *A tuple (Q_1, Q_2, Q_3, Q_4) is* consistent *if $Q_2 = S \setminus Q_1$, $Q_3 \subseteq Q_1 \setminus F$, and $Q_4 \subseteq Q_2 \setminus \hat{F}$.*

Definition 4. *Given an AWW $\mathcal{A} = \langle \Sigma, S, \delta, F \rangle$ we define its* full automaton *as the NBW $\mathcal{A}_f = \langle \Sigma, S_f, \delta_f, F_f \rangle$ where*

- *S_f is the set of consistent tuples over $2^S \times 2^S \times 2^S \times 2^S$.*
- *A state (Q'_1, Q'_2, Q'_3, Q'_4) is in $\delta_f((Q_1, Q_2, Q_3, Q_4), \sigma)$ if $Q'_1 \models \wedge_{s \in Q_1} \delta(s, \sigma)$, $Q'_2 \models \wedge_{s \in Q_2} \hat{\delta}(s, \sigma)$, and either:*
  1. *$Q_3 = Q_4 = \emptyset$, $Q'_3 = Q'_1 \setminus F$, and $Q'_4 = Q'_2 \setminus \hat{F}$*
  2. *$Q_3 \neq \emptyset$ or $Q_4 \neq \emptyset$, there exists $Y_3 \subseteq Q'_1$ such that $Y_3 \models \wedge_{s \in Q_3} \delta(s, \sigma)$ and $Q'_3 = Y_3 \setminus F$, and there exists $Y_4 \subseteq Q'_2$ such that $Y_4 \models \wedge_{s \in Q_4} \hat{\delta}(s, \sigma)$ and $Q'_4 = Y_4 \setminus \hat{F}$*
- *$F_f = \{(Q_1, Q_2, Q_3, Q_4) \in S_f | Q_3 = Q_4 = \emptyset)\}$*

Theorem 1. *Let $\mathcal{A}$ be an AWW and let $\mathcal{A}_f$ be its full automaton, let $Q \subseteq S$ be a set of states, then for every state (Q_1, Q_2, Q_3, Q_4) such that $Q_1 = Q$ we have that*

$$\bigcap_{s \in Q} L(\mathcal{A}^{(s)}) \bigcap_{s \notin Q} \overline{L(\mathcal{A}^{(s)})} = L(\mathcal{A}_f^{(Q_1, Q_2, Q_3, Q_4)})$$

We now present more properties of the full automaton. We use these properties later.

Definition 5. *Let $\mathcal{A}$ be an ABW and let w be an infinite word. We define the* type *of w w.r.t. $\mathcal{A}$ as the set $type_{\mathcal{A}}(w) = \{s | \mathcal{A}^{(s)} \text{ accepts } w\}$.*

The following lemma is a direct consequence of Theorem 1.

Lemma 3. *Let $\mathcal{A}_f$ be full automaton, and let (Q_1, Q_2, Q_3, Q_4) and (Q'_1, Q'_2, Q'_3, Q'_4) be states of $\mathcal{A}_f$. Then,*

- *If* $Q_1 = Q'_1$ *then* $L(\mathcal{A}_f^{(Q_1,Q_2,Q_3,Q_4)}) = L(\mathcal{A}_f^{(Q'_1,Q'_2,Q'_3,Q'_4)})$.
- *If* $Q_1 \neq Q'_1$ *then* $L(\mathcal{A}_f^{(Q_1,Q_2,Q_3,Q_4)}) \cap L(\mathcal{A}_f^{(Q'_1,Q'_2,Q'_3,Q'_4)}) = \emptyset$.

Lemma 3 and Theorem 1 imply that the first element Q_1 of the states of $\mathcal{A}_f$ characterizes a distinct language.

Definition 6. *The language of* Q_1 *is defined as* $L(Q_1) = L(\mathcal{A}_f^{(Q_1,S\setminus Q_1,\emptyset,\emptyset)})$.

3.2 The Local Transition System

As observed above, it is sufficient to look at Q_1 in order to determine the language of a state of $\mathcal{A}_f$. Recall that $L(Q_1) = L(\mathcal{A}^{(Q_1,S\setminus Q_1,\emptyset,\emptyset)})$. We observe that if there exists a transition $((Q_1,Q_2,Q_3,Q_4),\sigma,(Q'_1,Q'_2,Q'_3,Q'_4))$ in δ_f, then for every state of the form (Q_1,Q_2,Y_3,Y_4) there exists a state (Q'_1,Q'_2,Y'_3,Y'_4) such that the transition $((Q_1,Q_2,Y_3,Y_4),\sigma,(Q'_1,Q'_2,Y'_3,Y'_4))$ is in δ_f.

These observations imply that there are some local relationships between the languages of the states of $\mathcal{A}_f$. Indeed, if a word w is in $L((Q'_1,Q'_2,Q'_3,Q'_4))$ then for the word $\sigma \cdot w$ that is in $L((Q_1,Q_2,Q_3,Q_4))$, there exists a state of the form (Q'_1,Q'_2,Y'_3,Y'_4) that is in $\delta_f((Q_1,Q_2,Q_3,Q_4),\sigma)$. Thus, we can say that there exists a transition on σ from $L(Q_1)$ to $L(Q'_1)$. The local transition system captures these relationships.

Definition 7. *Given an AWW* $\mathcal{A} = \langle \Sigma, S, \delta, F\rangle$ *we define its* local transition system *as* $T_{\mathcal{A}} = \langle \Sigma, S_T, \delta_T\rangle$ *where*

- S_T *is the set of subsets of* S *and* δ_T *is a function from* S_T *to* 2^{S_T}.
- *A state* Q' *is in* $\delta_T(Q,\sigma)$ *if* $Q' \models \wedge_{s\in Q}\delta(s,\sigma)$ *and* $(S\setminus Q') \models \wedge_{s\notin Q}\hat{\delta}(s,\sigma)$.

Example 1. We now present an example of a full automaton and a local transition system. The example is presented at Figure 1. For simplicity we use a deterministic automaton $\mathcal{A}$. The figure shows $\mathcal{A}$'s dual automaton $\hat{\mathcal{A}}$, the full automaton $\mathcal{A}_f$, and the local transition system T_A. Note that $\hat{\mathcal{A}}$ has $\hat{F} = S$, thus for every state (Q_1,Q_2,Q_3,Q_4) of $\mathcal{A}_f$, we have $Q_4 = \emptyset$. For this reason, and since Q_2 is always equal to $S \setminus Q_1$, we only write the sets Q_1 and Q_3 inside the states. □

The definitions of the full automaton and the local transition system implies the following lemma:

Lemma 4. *Let* $\mathcal{A}$ *be an AWW and let* $\mathcal{A}_f$ *and* $T_{\mathcal{A}}$ *be its full automaton and local transition system respectively. Let* (Q_1,Q_2,Q_3,Q_4) *be a state of* $\mathcal{A}_f$*, and let* σ *be a letter in* Σ*. Then, for every state* Q'_1 *we have that* Q'_1 *is in* $\delta_T(Q_1,\sigma)$ *iff there exists a state of the form* (Q'_1,Q'_2,Q'_3,Q'_4) *in* $\delta_f((Q_1,Q_2,Q_3,Q_4),\sigma)$.

The proof of Lemma 4 is straightforward from the definitions of $\mathcal{A}_f$ and T_A. In particular for every state (Q_1,Q_2,Q_3,Q_4) and infinite word w we have that $\mathcal{A}^{(Q_1,Q_2,Q_3,Q_4)}$ has a run on w iff $T_A^{(Q_1)}$ has a run on w. However, we do not define accepting conditions for the local transition system. Thus, it is possible that $T_A^{(Q_1)}$ has a run on w, but $\mathcal{A}_f^{(Q_1,Q_2,Q_3,Q_4)}$ does not have an accepting run on w.

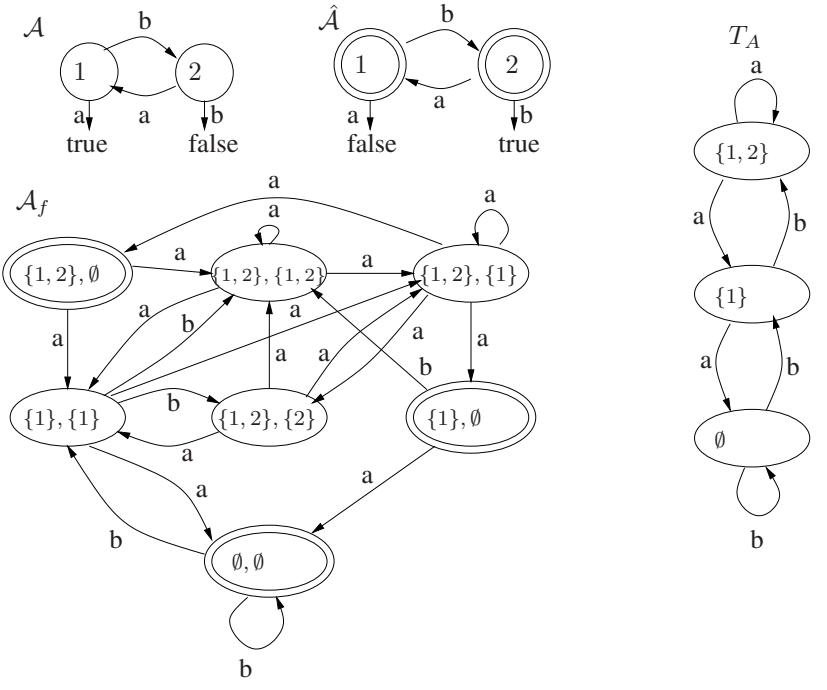

Fig. 1. An example of a full automaton $\mathcal{A}_f$ and a local transition system T_A.

Lemma 5. *Let T_A be a local transition system, and let Q, Q' and Q'' be states of $T_\mathcal{A}$. Let σ be a letter in Σ. If $Q'' \in \delta_T(Q', \sigma)$ and $Q'' \in \delta_T(Q, \sigma)$, then $Q = Q'$.*

When a transition system satisfies the property shown in Lemma 5, we say that the transition system is *reverse deterministic.*

4 Verifying Markov Chains

In this section we construct a product $G_{\mathcal{M},\mathcal{A}}$ of the Markov chain $\mathcal{M}$ and the local transition system T_A. We show that the problem of checking whether $P_M(L(\mathcal{A})) = 1$, can be reduced to checking for a state (x, Q) of G whether the probability of $L(Q) \cap x \cdot \Sigma^\omega$ is positive. Then, we show how to use the full automaton to solve this problem.

Definition 8. *Let $\mathcal{A}$ be an AWW, $\mathcal{T}_A$ be $\mathcal{A}$'s local transition system, and $\mathcal{M}$ be a Markov chain. We define the graph $G_{\mathcal{M},\mathcal{A}}$ as having vertex set (x, Q) such that x is a state of $\mathcal{M}$ and Q is a state of T_A. An edge $(x, Q) \to (x', Q')$ is included in $G_{\mathcal{M},\mathcal{A}}$ if $\mathcal{M}$ has a transition $x \to x'$ and (Q, x, Q') is a transition in T_A.*

When $\mathcal{A}$ and $\mathcal{M}$ are clear from the context, we write G instead of $G_{\mathcal{M},\mathcal{A}}$. Lemma 5 implies that for every three states (x, Q), (x', Q'), and (x'', Q''), if there is a transition from (x, Q) to (x'', Q'') and there is a transition from (x', Q') to (x'', Q''), then $x \neq x'$. We say that G is *reverse deterministic*.

Example 2. We present in Figure 2 two Markov chains $\mathcal{M}_1$ and $\mathcal{M}_2$. We assume that the initial probability for each state in the Markov chains is $\frac{1}{2}$. The figure also presents the products G_1 and G_2 of $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively, with the local transition system T_A from Example 1. □

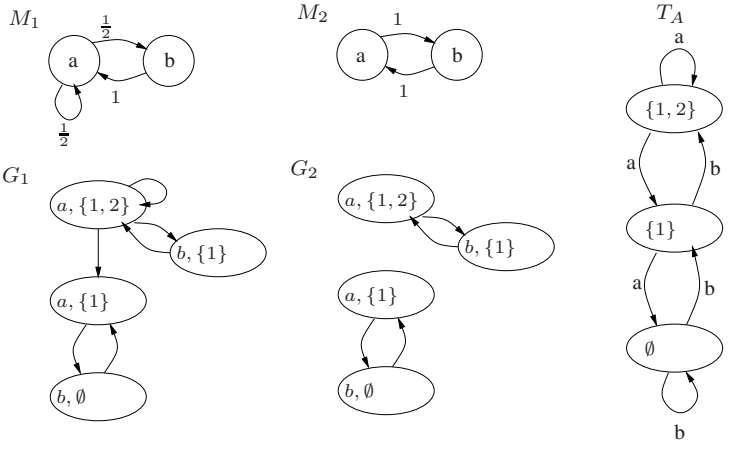

Fig. 2. Two Markov chains $\mathcal{M}_1$ and $\mathcal{M}_2$, and the graphs they impose G_1 and G_2.

Every infinite path in G projected on the first component gives a path in $\mathcal{M}$. Conversely, every path of $\mathcal{M}$ is the projection of at least one path in G. In fact, let $w = x_0 \cdot x_1 \cdot \ldots$ be a path of $\mathcal{M}$. For each j, let Q_j be the type of the suffix $x_j \cdot x_{j+1} x_{j+2} \cdots$. Then for each j there is a transition (Q_j, x_j, Q_{j+1}) in δ_T and thus an edge $(x_j, Q_j) \to (x_{j+1}, Q_{j+1})$ in G. We call this path the *augmented path* corresponding to w.

Definition 9. *For a state (x, Q) of G we denote by $P(x, Q)$ the probability that a path that starts in state x has type Q, namely $P(x, Q) = P_M(\mathcal{M}^{(x)}$ has type $Q)$. We call* (x, Q) probable *if $P(x, Q) > 0$.*

The importance of the probable states is demonstrated in the following two lemmas.

Lemma 6. $P_M(L(Q)) = \sum_{x \in X} P_I(x) \cdot P(x, Q)$.

Let x be a state of a Markov chain with $P_I(x) > 0$. Then for every state (x, Q) we have that if (x, Q) is probable, then $P_M(L(Q)) > 0$. Thus we conclude Lemma 7.

Lemma 7. *Let s be a state of an AWW $\mathcal{A}$ and let $\mathcal{M}$ be a Markov chain. Then, $P_M(L(s)) < 1$ iff there exists a state x of $\mathcal{M}$ and a set $Q \subseteq S$ such that $P_I(x) > 0$, $s \notin Q$ and the state (x, Q) is probable.*

Thus, in order to determine whether $P_M(L(s_0)) = 1$, it is enough to determine the probable states of G. In the rest of this section we show how to identify the probable states. We define H as the restriction of G to the probable states.

Example 3. Look again at Figure 2. It is easy to see that given that a path of $\mathcal{M}_1$ starts at state a, the path is of the form $a((ba)+a)^\omega$ with probability 1. $a((ba)+a)^\omega$ is contained in $L(\{1,2\}) \cup \{(ab)^\omega\}$, since $P_{\mathcal{M}_1}((ab)^\omega) = 0$, we have $P(a,\{1,2\}) = 1$. Similarly, a path of $\mathcal{M}_1$ that starts at b is with probability one in $L(\{1\})$, thus, $P(b,\{1\}) = 1$. This implies that in G_1, H is the subgraph induced by the states $(a,\{1,2\})$ and $(b,\{1\})$. On the other hand, looking at $\mathcal{M}_2$, we see that a path that starts at a is of the form $(ab)^\omega$ and a path that starts at b is of the form $(ba)^\omega$. Thus, in G_2, H is the subgraph induced by the states $(a,\{1\})$ and $(b,\emptyset)$. □

We start with the following observation. Partition the language $L(Q)$ according to the first letter of a word and the type of the suffix that starts from the second letter. Then, for every state (x,Q) of G we have $P(x,Q) = \sum_{(x,Q)\to(x',Q')} P_T(x,x') \cdot P(x',Q')$. Note that if $(x,Q) \to (x',Q')$ is an edge of G, then $P_T(x,x') > 0$ and thus $P(x,Q) \geq P_T(x,x') \cdot P(x',Q')$. Hence, if (x',Q') is probable then all its ancestors are probable. This implies that it is sufficient to identify the BSCCs of H and then to construct the set of probable states using backward reachability.

Let C be an SCC of G. If it contains some probable state (x,Q), then since all the states in C are ancestors of (x,Q), all states in C are probable. That is, either C is an SCC of H or $C \cap H = \emptyset$. Recall that every path in C projects to a path in $\mathcal{M}$. So the set of first components of all members of C are in the same SCC, say K, of $\mathcal{M}$, which is the SCC of $\mathcal{M}$ containing x. We say that C *corresponds* to K. Note that distinct SCC's of G may correspond to the same K.

Theorem 2 characterizes the SCCs of G which are the BSCCs of H. Before we present the theorem we need the following notation. For a tuple $E = \langle E_1, E_2, \ldots E_n \rangle$, we define $\pi_i(E) = E_i$ to be the i'th element in E. This notation is extended naturally to sequences of tuples.

Definition 10. *A finite path ρ_G in G is* fulfilling *if there exists a path ρ_f in $\mathcal{A}_f$ such that $\pi_2(\rho_G) = \pi_1(\rho_f)$, the first state of ρ_f is of the form $(Q_1, Q_2, (Q_1 \setminus F), (Q_2 \setminus \hat{F}))$, and the last state of ρ_f is of the form $(Q'_1, Q'_2, \emptyset, \emptyset)$.*

Theorem 2. *Let C be an SCC of G. Then C is a BSCC of H iff it satisfies the following conditions:*

1. *C corresponds to a BSCC K of $\mathcal{M}$.*
2. *Every finite path of K is a projection of a path in C.*
3. *C contains a fulfilling path.*

5 Algorithms

In Figure 3 we present the algorithm that determines for an AWW $\mathcal{A}$ and a Markov chain $\mathcal{M}$ whether $P_M(L(\mathcal{A})) = 1$. An extension for exact probability is presented in full version. Theorem 2 implies that the algorithm mark the BSCCs of H. Thus, B is the set of probable states. Lemma 7 implies that the algorithm returns **true** iff $P_M(L(\mathcal{A})) = 1$. Finding SCCs in G that correspond to BSCCs in $\mathcal{M}$, and doing backward reachability can be done in time linear in the size of G. The most complex part of the algorithm is to identify, SCCs C of G that satisfy:

```
Inputs: Markov chain M = ⟨X, P_I, P_T⟩, AWW A = ⟨Σ, S, s_0, δ, F⟩.
Construct the full automaton A_f of A.
Construct the local transition system T_A and the graph G.
Mark all SCCs C of G that satisfy:
   1. C corresponds to a BSCC K of M.
   2. Every finite path of K is a projection of a path in C.
   3. C contains a fulfilling path.
Construct the set B of all states of G from which the marked
SCCs are reachable.
return true iff for every state (x, Q) ∈ B, if P_I(x) > 0, then s_0 ∈ Q.
```

Fig. 3. The model-checking algorithm.

1. C corresponds to a BSCC K of $\mathcal{M}$.
2. Every finite path of K is a projection of a path in C.
3. C contains a fulfilling path.

The following lemma is proved in [10]. The only property of G that they use is that G is reverse deterministic.

Lemma 8. *Let C be an SCC of G that corresponds to an SCC K of $\mathcal{M}$. Then the following are equivalent:*

1. *Every finite path in K is a projection of a path in C.*
2. *No other SCC of G corresponding to K is an ancestor of C.*

Lemma 8 implies that the second task is equivalent to checking whether there is no ancestor SCC of C that corresponds to K. This check can be easily done while scanning the SCCs of G.

Example 4. In G_1 at Figure 2 there are two SCC's that correspond to the single BSCC of $\mathcal{M}_1$. The SCC of $(a, \{1,2\})$ and $(b, \{1\})$ does not have ancestors and contains the fulfilling path $(a, \{1,2\}), (a, \{1,2\}), (a, \{1,2\})$ that corresponds to the path $(\{1,2\}, \{1,2\}), (\{1,2\}, \{1\}), (\{1,2\}, \emptyset)$ in $\mathcal{A}_f$, thus, this SCC is the BSCC of H. In G_2 there are two SCCs that correspond to the single BSCC of $\mathcal{M}_2$ and neither of them have an ancestor. However, only the SCC of $(a, \{1\})$ and $(b, \emptyset)$ has a fulfilling path, thus it is the BSCC of H.□

We now explain how to check whether an SCC C of G contains a fulfilling path. We construct the product $G_f = \mathcal{M} \times \mathcal{A}_f$, similarly to the construction of G.

Definition 11. *Let $\mathcal{A}$ be an AWW, $\mathcal{A}_f$ be $\mathcal{A}$'s full automaton, and $\mathcal{M}$ be a Markov chain. We define the* full graph *G_f as having vertex set $(x, (Q_1, Q_2, Q_3, Q_4))$ such that x is a state of $\mathcal{M}$ and (Q_1, Q_2, Q_3, Q_4) is a state of $\mathcal{A}_f$. An edge $(x, (Q_1, Q_2, Q_3, Q_4)) \to (x', (Q'_1, Q'_2, Q'_3, Q'_4))$ is included in G_f if $\mathcal{M}$ has a transition $x \to x'$ and (Q'_1, Q'_2, Q'_3, Q'_4) is in $\delta_f((Q_1, Q_2, Q_3, Q_4), x)$.*

Lemma 9. *An SCC C of G contains a fulfilling path iff there exists a path $(x_0, (Q_1^0, Q_2^0, Q_3^0, Q_4^0)), (x_1, (Q_1^1, Q_2^1, Q_3^1, Q_4^1)), \ldots, (x_n, (Q_1^n, Q_2^n, \emptyset, \emptyset))$ in G_f such that the path $(x_0, Q_1^0), (x_1, Q_1^1), \ldots (x_n, Q_1^n)$ is contained in C, $Q_3^0 = Q_1^0 \setminus F$, and $Q_4^0 = Q_2^0 \setminus \hat{F}$.*

Complexity. Finding SCCs in G that correspond to BSCCs in $\mathcal{M}$, and doing backward reachability can be done in linear time and polylogarithmic space in $|G|$. Constructing BSCCs of $\mathcal{M}$ can be done in time linear in $|\mathcal{M}|$, identifying SCCs of G that correspond to these BSCC can be done in time linear in $|G|$. Marking SCCs that do not have ancestors that correspond to the same BSCC in $\mathcal{M}$ can also be done in time linear in $|G|$. Checking that an SCC of G contains a fulfilling path can be done in time linear in $|G_f|$, simply by scanning G_f and G in parallel, thus, the algorithm can be implemented in time linear in $|\mathcal{M} \times \mathcal{A}_f|$. Since the size of $\mathcal{A}_f$ is $2^{O(|\mathcal{A}|)}$, we have that the time complexity of the algorithm is $|\mathcal{M}| \cdot 2^{O(|\mathcal{A}|)}$.

As for space complexity we show that algorithm works in space polynomial in $|\mathcal{A}|$ and polylogarithmic in $|\mathcal{M}|$. We rewrite the conditions of Theorem 2, Lemma 7, and Lemma 8 as follows: $P_M(L(\mathcal{A})) < 1$ iff there exists a probable state (x_0, Q_0) such that $s_0 \notin Q$ and $P_I(x_0) > 0$. This is true iff (x_0, Q_0) reaches a state (x, Q) that is in a BSCC of H, $s_0 \notin Q_0$, and $P_I(x_0) > 0$. That is

1. (x, Q) is reachable from a state (x_0, Q_0) such that $P_I(x_0) > 0$ and $s_0 \notin Q_0$.
2. x is in a BSCC of $\mathcal{M}$ (Theorem 2, (1)). This condition is equivalent to the following: for every state x' of $\mathcal{M}$ we have that if there exists a path in $\mathcal{M}$ from x to x' then there exists a path from x' to x.
3. No other SCC of G that corresponds to the SCC of x in $\mathcal{M}$ is the ancestor of the SCC of (x, Q) (Lemma 8). This condition is equivalent to the following: for every state (x', Q'), if there exists a path from (x', Q') to (x, Q), then either there exists a path from (x, Q) to (x', Q'), or there is no path from x to x'.
4. The SCC of (x, Q) contains a fulfilling path (Theorem 2, (3)). By Lemma 9 this condition is equivalent to the following: there exists a path in G_f from a state $(x', (Q'_1, Q'_2, Q'_1 \setminus F, Q'_2 \setminus \hat{F}))$ to a state $(x'', (Q''_1, Q''_2, \emptyset, \emptyset))$ such that the projection of the path on G is contained in the SCC of (x, Q). This condition is equivalent to: there is a path from a state $(x', (Q'_1, Q'_2, Q'_1 \setminus F, Q'_2 \setminus \hat{F}))$ to a state $(x'', (Q''_1, Q''_2, \emptyset, \emptyset))$ in G_f and there are paths from (x, Q) to (x'', Q''_1), from (x'', Q''_1) to (x', Q'), and from $((x', Q')$ to (x, Q) in G.

In [25] it is shown that checking whether there is a path from one state to another in a graph with n states requires $log^2(n)$ space. This implies that the conditions above can be checked in space $O(log^2(|G_f|)) = log^2(|\mathcal{M}| \cdot 2^{O(|\mathcal{A}|)}) = O(log^2(|\mathcal{M}|) + log(|\mathcal{M}|) \cdot |\mathcal{A}| + |\mathcal{A}|^2) = O(log^2(|\mathcal{M}|) + |\mathcal{A}|^2)$.

6 Concluding Remarks

We presented here an optimal solution to the general problem of linear-time probabilistic model checking with respect to ω-regular specifications, expressed by alternating automata. Beyond the interest in the problem itself, our solution is interesting from a theoretical perspective, since the concept of full automaton may have other applications. More work is needed in reducing our result to practice. One direction is to extend the *ProbaTaf* system, which currently handles LTL specifications of Markov chain [11], to ABW specifications. Another, is to combine the symbolic approach to alternating automata [20] with the symbolic approach to probabilistic model checking [3].

References

1. R. Armoni, D. Bustan, O. Kupferman, and M. Y. Vardi. Resets vs. aborts in linear temporal logic. In *Int'l Conf. on Tools and Algorithms for Construction and Analysis of Systems*, pages 65 – 80, 2003.
2. R. Armoni, L. Fix, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, A. Tiemeyer, E. Singerman, M.Y. Vardi, and Y. Zbar. The ForSpec temporal language: A new temporal property-specification language. In *Proc. 8th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, LNCS 2280, pages 296–311, 2002.
3. C. Baier, E.M. Clarke, V. Hartonas-Garmhausen, M.Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming, 24th Inte.l Colloq.*, LNCS 1256, pages 430–440, 1997.
4. I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic sugar. In *Proc. Conf. on Computer-Aided Verification*, LNCS 2102, pages 363–367, 2001.
5. P. Berman and J.A Garay. Randomized distributed agreement revisited. In *Proceedings of the 23rd Inte'l Symp. on Fault-Tolerant Computing (FTCS '93)*, pages 412–421, 1993.
6. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986.
7. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
8. R. Cole, B.M. Maggs, F. M. auf der Heide, A. Richa M. Mitzenmacher, K. Schroder, R. Sitaraman, and B. Vocking. Randomized protocols for low-congestion circuit routing in multistage interconnection networks. In *30th ACM Symp. on Theo. of Comp. (STOC)*, pages 378–388, 1998.
9. C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In *Proc. 17th Int'l Coll. on Automata Languages and Programming*, volume 443, pages 336–349. LNCS, 1990.
10. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
11. J. M. Couvreur, N. Saheb, and G. Sutre. An optimal automata approach to LTL model checking of probabilistic systems. In *Proc. 10th Int. Conf. Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'2003), Almaty, Kazakhstan, Sep. 2003*, volume 2850 of *Lecture Notes in Artificial Intelligence*. Springer, 2003.
12. C. Fritz and T. Wilke. State space reductions for alternating Büchi automata: Quotienting by simulation equivalences. In *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science: 22nd Conf.*, volume 2556 of *LNCS*, pages 157–168, 2002.
13. P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Computer Aided Verification, Proc. 13th Inte'l Conf.*, volume 2102 of *LNCS*, pages 53–65, 2001.
14. G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.
15. J.G. Kemeny, J.L. Snell, and A.W. Knapp. *Denumerable Markov Chains*. Springer-Verlag, 1976.
16. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symp. on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.
17. R.P. Kurshan. *FormalCheck User's Manual*. Cadence Design, Inc., 1998.
18. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.

19. O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Brooklyn, June 1985. Springer-Verlag.
20. S. Merz. Weak alternating automata in Isabelle/HOL. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference*, volume 1869 of *Lecture Notes in Computer Science*, pages 423–440. Springer-Verlag, 2000.
21. M.J. Morley. Semantics of temporal ***e***. In T. F. Melham and F.G. Moller, editors, Banff'99 *Higher Order Workshop (Formal Methods in Computation)*. University of Glasgow, Department of Computing Science Technic al Report, 1999.
22. D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Intel Colloq. on Automata, Languages and Programming*, volume 226 of *LNCS*, 1986.
23. J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Inte'l Symp. on Programming*, volume 137 of *LNCS*, pages 337–351, 1981.
24. S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.
25. W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal on Computer and System Sciences*, 4:177–192, 1970.
26. A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.
27. S. Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109, 2000.
28. M. Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Formal Methods for Real-Time and Probabilistic Systems: 5th Inte'l AMAST Workshop*, volume 1601 of *LNCS*, pages 265–276, 1999.
29. M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th IEEE Symp. on Foundations of Computer Science*, pages 327–338, Portland, October 1985.
30. M.Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th ACM Symp. on Principles of Programming Languages*, pages 250–259, San Diego, January 1988.
31. M.Y. Vardi. Nontraditional applications of automata theory. In *Proc. Inte'l Symp. on Theoretical Aspects of Computer Software*, volume 789, pages 575–597. LNCS, 1994.
32. M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 238–266, 1996.
33. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.
34. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.
35. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.