An Optimal Automata Approach to LTL Model Checking of Probabilistic Systems

Jean-Michel Couvreur[1], Nasser Saheb[2], and Grégoire Sutre[2]

[1] LSV, ENS Cachan, Cachan, France
[2] LaBRI, Université de Bordeaux I, Talence, France

couvreur@lsv.ens-cachan.fr, {saheb, sutre}@labri.fr

Abstract. Most verification problems on finite systems may be formulated and solved optimally using automata based techniques. Nonetheless LTL verification of (finite) probabilistic systems, i.e. deciding whether a probabilistic system almost surely satisfies an LTL formula, remains one of the few exceptions to this rule. As a matter of fact, existing automata-based solutions to this problem lead to double EXPTIME algorithms, while Courcoubetis and Yannakakis provide an optimal one in single EXPTIME. In this study, we remedy this exception. Our optimal automata based method proceeds in two steps: we present a minimal translation from LTL to ω-automata and point out appropriate properties on these automata ; we then show that checking whether a probabilistic system satisfies an ω-automaton with positive probability can be solved in linear time for this kind of automata. Moreover we extend our study to the evaluation of this probability. Finally, we discuss some experimentations with our implementation of these techniques: the ProbaTaf tool.

1 Introduction

Many techniques have been introduced to check algorithmically that a system satisfies a given set of required properties. These properties are usually specified by formulas in some temporal logic. The automata approach to model checking basically consists in (1) translating temporal formulas into some sort of automata and (2) checking that the behavior of the system (usually given as an infinite branching tree or as a set of linear infinite traces) is "accepted" by this automaton [8, 5]. This approach has been successfully applied by many authors in the area of verification, and it turns out that most verification problems on (non probabilistic) finite-state systems can be solved optimally using automata-based techniques.

Randomized algorithms are designed in numerous field of computer science in particular in the synchronization of concurrent processes and that of distributed computation. In fact, in these areas there are problems that admit no satisfactory deterministic algorithms and, therefore the randomization tool becomes a necessity. It would be desirable to develop efficient techniques for the verification of concurrent probabilistic systems.

Verification of probabilistic systems usually consists in checking that a given property is satisfied with probability 1. The standard automata approach to model checking LTL formulas on probabilistic systems require a determinization of the Büchi ω-automata obtained from LTL formulas [7]. This leads to a doubly exponential time complexity. Courcoubetis and Yannakakis [1] developed another (non automata based) algorithm that runs in single exponential time and polynomial space, which is proven to be optimal.

We present in this paper an automata based method to model-checking LTL formulas on probabilistic systems, that meets the optimal complexity bounds of [1]. As in the non probabilistic case, we synchronize the system with the automaton corresponding to the (negation of) the LTL formula. However, we use a translation from LTL to ω-automata [3] that yields ω-automata with specific properties. We then exploit these properties to avoid determinization of these ω-automata, which is the crucial step leading to a doubly exponential time complexity in [7].

Basic concepts on measures over the set of infinite words and on probabilistic systems are introduced in Section 2. In Section 3, we introduce LTL formulas and the related ω-automata. The probabilistic verification and probabilistic satisfaction problems are discussed in Section 4. Probabilistic evaluation is the subject of Section 5. We finally show some experimentation on ProbaTaf in Section 6. In this short paper, proofs are omitted and left for a complete version of the paper.

2 Probabilistic Systems

2.1 Words and measurability

Let Σ be an *alphabet* (a finite non empty set of *letters*). We write Σ^* (resp. Σ^ω) for the set of all *finite words* $a_0a_1\cdots a_n$ (resp. *infinite words* $a_0a_1\cdots a_n\cdots$) over Σ, and ε denotes the empty (finite) word. A *word* is any finite or infinite word, and we write Σ^∞ for the set of all words. A subset of Σ^∞ is called a *language*.

Given a finite word x and a word y, $x \cdot y$ (shortly written xy) is their *concatenation*. A finite word y is a *left factor* of a (finite or infinite) word x, written $y \leq x$, if $x = y \cdot z$ for some word z and moreover we write $z = y^{-1}x$. These operations are classically extended over languages: for any $X \subseteq \Sigma^*$ and $Y \subseteq \Sigma^\infty$, $X \cdot Y = \{xy \mid x \in X, y \in Y\}$ and $X^{-1}Y = \{x^{-1}y \mid x \in X, y \in Y\}$.

Recall that a *σ-algebra* on a set X is a subset $\mathcal{A}$ of $\mathbb{P}(X)$ such that $\emptyset \in \mathcal{A}$, $\mathcal{A}$ is closed under complementation, and $\mathcal{A}$ is closed under countable union. The pair $(X, \mathcal{A})$ is then called a *measurable space*, and elements of $\mathcal{A}$ are called *measurable sets*. Given two measurable spaces $(X_1, \mathcal{A}_1)$ and $(X_2, \mathcal{A}_2)$, a mapping $f : X_1 \to X_2$ is called a *measurable mapping* if the inverse image $f^{-1}(Y_2)$ of any measurable subset Y_2 of X_2 is a measurable subset of X_1.

A *measure* μ over a measurable space $(X, \mathcal{A})$ is any mapping from $\mathcal{A}$ to $\mathbb{R}^+$ such that (1) $\mu(\emptyset) = 0$, and (2) for any countable sequence $(A_n)_{n\in\mathbb{N}}$ of pairwise disjoint sets in $\mathcal{A}$, $\mu(\bigcup_{n\in\mathbb{N}} A_n) = \sum_{n\in\mathbb{N}} \mu(A_n)$. A *probability measure* over $(X, \mathcal{A})$ is a measure μ over $(X, \mathcal{A})$ such that $\mu(X) = 1$.

Given an alphabet Σ, we denote by $\mathcal{C}_\Sigma$ the set of all *basic cylindric sets* $w \cdot \Sigma^\omega$ with $w \in \Sigma^*$, and by $\mathcal{B}_\Sigma$ the *σ-algebra* (on Σ^ω) generated by $\mathcal{C}_\Sigma$. For the rest of the paper, $(\Sigma^\omega, \mathcal{B}_\Sigma)$ will be the considered *measurable space* (where Σ will depend on the context).

Proposition 2.1. *Let Σ be an alphabet. For any language $L \subseteq \Sigma^\omega$, the following assertions are equivalent:*

i) *L is measurable*
ii) *$K \cdot L$ is measurable for every language $K \subseteq \Sigma^*$*
iii) *$w \cdot L$ is measurable for some finite word $w \in \Sigma^*$*
iv) *$K^{-1}L$ is measurable for every language $K \subseteq \Sigma^*$*
v) *$a^{-1}L$ is measurable for every letter $a \in \Sigma$.*

For the rest of this section, we consider two alphabets Σ_1 and Σ_2.

Definition 2.2. *For any mapping $f : \Sigma_1 \to \Sigma_2$, the ω-*extension *$\bar{f}$ of f is the mapping $\bar{f} : \Sigma_1^\omega \to \Sigma_2^\omega$ defined by $\bar{f}(a_0 a_1 \cdots a_n \cdots) = f(a_0) f(a_1) \cdots f(a_n) \cdots$.*

Proposition 2.3. *The ω-extension of any mapping $f : \Sigma_1 \to \Sigma_2$ is measurable.*

Notation. In the remaining of the paper, we will shortly write f instead of $\bar{f}$.

Theorem 2.4 (Measure Extension). *For any mapping $f : \mathcal{C}_\Sigma \to \mathbb{R}^+$, the following assertions are equivalent:*

i) *we have $f(w \cdot \Sigma^\omega) = \sum_{a\in\Sigma} f(wa \cdot \Sigma^\omega)$ for every $w \in \Sigma^*$*
ii) *there exists a unique measure μ over $(\Sigma^\omega, \mathcal{B}_\Sigma)$ such that μ and f are equal on $\mathcal{C}_\Sigma$.*

2.2 Finite graphs

Probabilistic systems and ω-automata will later be defined as "enriched" finite labeled graphs. Transitions in these graphs will play a major role in the rest of the paper (for instance, the measurable space associated with a probabilistic system and acceptance conditions in ω-automata will be defined in terms of transitions). Hence we choose to represent transitions in graphs explicitly (instead of the usual representation of transitions as tuples). Moreover, thanks to this representation, probabilistic systems can contain two times the same transition with different probabilities attached to them.

A *finite labeled graph* (over Σ) is a 5-tuple $G = \langle V, T, \alpha, \beta, \lambda \rangle$ where V is a finite set of *vertices*, T is a finite set of *transitions*, $\alpha : T \to V$ and $\beta : T \to V$ are the *source* and the *target* mappings, and $\lambda : T \to \Sigma$ is a *transition labeling*.

Notation. We will use throughout the paper the following notations:

- for any transition $t \in T$, the source $\alpha(t)$ of t is shortly written ${}^\bullet t$ and the target $\beta(t)$ of t is shortly written $t^\bullet$.
- for any vertex $v \in V$, the set $\beta^{-1}(v)$ of transitions entering v is denoted by ${}^\bullet v$ and the set $\alpha^{-1}(v)$ of transitions leaving v is denoted by $v^\bullet$.
- we will also write ${}^\bullet X$ and $X^\bullet$ when $X \subseteq V$ or $X \subseteq T$: ${}^\bullet X$ and $X^\bullet$ are then defined as expected (e.g. ${}^\bullet X = \{{}^\bullet x \mid x \in X\}$ if $X \subseteq T$).

The *transition relation* $\rightarrow$ of G is the relation on V defined by: $v \rightarrow v'$ if $v^\bullet \cap {}^\bullet v' \neq \emptyset$. We denote by $\xrightarrow{+}$ (resp. $\xrightarrow{*}$) the transitive closure of $\rightarrow$ (resp. the reflexive and transitive closure of $\rightarrow$). Observe that $\xrightarrow{*}$ is a preorder on V.

A *finite path* (resp. *infinite path*) in G is any non empty finite word $\rho = t_0 t_1 \cdots t_n$ (resp. any infinite word $\rho = t_0 t_1 \cdots t_n \cdots$) over T such that $t_i{}^\bullet = {}^\bullet t_{i+1}$ for all $i \leq n-1$ (resp. for all $i \in \mathbb{N}$). Moreover, we say that ρ is a finite (resp. infinite) path *originating from* v_0, where $v_0 = {}^\bullet t_0$. We write $Path^*(G)$ (resp. $Path^*(G,v)$, $Path^\omega(G)$, $Path^\omega(G,v)$) for the set of all finite paths (resp. finite paths from v, infinite paths, infinite paths from v) in G. The sets $Trace^*(G)$ of all *finite traces*, $Trace^*(G,v)$ of all *finite traces from* v, $Trace^\omega(G)$ of all *infinite traces* and $Trace^\omega(G,v)$ of all *infinite traces from* v are the image under λ of the corresponding sets of infinite paths (e.g. $Trace^*(G) = \lambda(Path^*(G))$).

Proposition 2.5. *Given a finite labeled graph G and a vertex v of G, the sets $Path^\omega(G)$, $Path^\omega(G,v)$, $Trace^\omega(G)$ and $Trace^\omega(G,v)$ are measurable.*

A *strongly connected component* (shortly *SCC*) in G is any equivalence class for the equivalence relation $\xleftrightarrow{*}$ on V defined by: $v \xleftrightarrow{*} v'$ if $v \xrightarrow{*} v'$ and $v' \xrightarrow{*} v$. Given a vertex v, we write $SCC(v)$ for the SCC containing v (in other words $SCC(v)$ is the equivalence class of v). We denote by $SCC(G)$ the set of all SCCs in G (in other words $SCC(G)$ is the quotient set $V/\xleftrightarrow{*}$). Observe that $\xrightarrow{*}$ is a partial order on $SCC(G)$. An SCC C is called *maximal* when it is maximal with respect to $\xrightarrow{*}$.

Given a function $f : X \rightarrow Y$ and a subset $X' \subseteq X$, we denote by $f_{|X'}$ the *restriction of f to X'*. Given a subset $V' \subseteq V$, the *restriction of G to V'* is the graph $G_{|V'} = \langle V', T', \alpha', \beta', \lambda' \rangle$ where $T' = {}^\bullet V' \cap V'^\bullet$, $\alpha' = \alpha_{|V'}$, $\beta' = \beta_{|V'}$ and $\lambda' = \lambda_{|V'}$.

We will use finite labeled graphs to define ω-automata and probabilistic systems. Vertices of an ω-automaton (resp. probabilistic system) will be called *locations* (resp. *states*) and they will be denoted by $q, q_0, q_1, \ldots$ (resp. $s, s_0, s_1, \ldots$).

2.3 Probabilistic Systems

A *probabilistic system* (over Σ) is a 7-tuple $M = \langle S, T, \alpha, \beta, \lambda, P_0, P \rangle$ where:

1. $\langle S, T, \alpha, \beta, \lambda \rangle$ is a finite labeled graph over Σ, and
2. $P_0 : S \rightarrow [0,1]$ is an *initial probability distribution* satisfying $\sum_{s \in S} P_0(s) = 1$,

3. $P : T \to]0,1]$ is a *transition probability function* satisfying $\sum_{t \in s^\bullet} P(t) = 1$, for all $s \in S$.

Observe that the third condition above enforces every state to be the source of at least one transition (there is no "deadlock state").

Following Theorem 2.4, we define μ_M as the unique *probability measure* over $(T^\omega, \mathcal{B}_T)$ defined on basic cylindric sets by:

1. $\mu_M(T^\omega) = 1$, and
2. for any non empty word $t_0 t_1 \cdots t_n \in T^*$,

$$\mu_M(t_0 t_1 \cdots t_n \cdot T^\omega) = \begin{cases} P_0({}^\bullet t_0) P(t_0) P(t_1) \cdots P(t_n) & \text{if } t_0 t_1 \cdots t_n \in Path^*(M) \\ 0 & \text{otherwise.} \end{cases}$$

For the rest of the paper, $(T^\omega, \mathcal{B}_T, \mu_M)$ will be the considered *probability space*.

Proposition 2.6. *We have $\mu_M(Path^\omega(M)) = 1$.*

The states $s \in S$ such that $P_0(s) > 0$ are called *initial states*. Given a state s, we denote by $M[s]$ the probabilistic system M with s as unique initial state (i.e. the initial probability distribution P'_0 of $M[s]$ is such that $P'_0(s) = 1$). The following observation easily follows from the uniqueness of μ_M:

Proposition 2.7. *For every measurable language $L \subseteq T^\omega$, we have:*

$$\mu_M(L) = \sum_{s \in S_0} P_0(s) \cdot \mu_{M[s]}(L).$$

Proposition 2.8. *For any state s, we have $\mu_{M[s]}(Path^\omega(M, s)) = 1$.*

Proposition 2.9. *For every state s and for every measurable language $L \subseteq T^\omega$, we have:*

$$\mu_{M[s]}(L) = \sum_{t \in s^\bullet} P(t) \cdot \mu_{M[t^\bullet]}(t^{-1} L).$$

Lemma 2.10. *Let $K \subseteq T^*$. If there is $k \in \mathbb{N}$ such that for all $\sigma \in Path^*(M)$ we have $\sigma \cdot \sigma' \in K \cdot T^*$ for some $\sigma' \in T^k$, then we have $\mu_M(K \cdot T^\omega) = 1$.*

The two following propositions follow from Lemma 2.10, and state strong equity properties of (finite) probabilistic systems: (1) a path ends almost surely in a maximal SCC, and (2) an infinite path in a maximal SCC visits almost surely infinitely often all finite paths of the SCC.

Proposition 2.11. *Let $Path^*_{max}$ denote the set of all finite paths ending in a maximal SCC. We have $\mu_M(Path^*_{max} \cdot T^\omega) = 1$.*

Proposition 2.12. *Let ρ be a finite path contained in some maximal SCC C, and let $s \in C$. We have $\mu_{M[s]}((T^* \cdot \rho)^\omega) = 1$.*

3 Temporal properties

3.1 LTL

We review the basic definitions of *linear temporal logic* (LTL). Formulas in LTL are build from letters of a given alphabet Σ, the boolean operators, the unary temporal operator X (Next) and the binary temporal operator U (Until). The formulas in LTL are interpreted over infinite words in Σ^ω. For an infinite word $w = w_0 \cdot w_1 \cdots$ and a position i, we define the satisfaction of a formula f by w at position i (denoted $w, i \models f$) inductively on the structure of f as follows:

- $w, i \models a$ iff $w_i = a$,
- $w, i \models f \vee g$ iff $w, i \models f$ or $w, i \models g$,
- $w, i \models \neg f$ iff not $w, i \models f$,
- $w, i \models Xf$ iff $w, i+1 \models f$,
- $w, i \models fUg$ iff for some $j \geq i$, $w, i \models g$ and for all k, $j > k \geq i$, $w, k \models f$.

We say that a word w *satisfies* a formula f, denoted $w \models f$, iff $w, 0 \models f$. The set of all infinite words satisfying f, called the *language of* f, is denoted $L(f)$.

Usual boolean constants and connectives are defined classically (e.g. *true* is defined as $a \vee \neg a$ for some given letter $a \in \Sigma$).

3.2 Automata on infinite words

An *ω-automaton* (over Σ) is a 7-tuple $A = \langle Q, T, \alpha, \beta, \lambda, Q_0, Acc \rangle$ where:

1. $\langle Q, T, \alpha, \beta, \lambda \rangle$ is a finite labeled graph over Σ, and
2. $Q_0 \subseteq Q$ is a set of *initial locations*, and
3. $Acc \subseteq 2^T$ is an *acceptance condition*.

A *run* of A over an infinite word $w \in \Sigma^\omega$ is an infinite path ρ from some initial location $q_0 \in Q_0$ such that $\lambda(\rho) = w$. Acceptance of runs is defined in terms of limits of transition sets. Given a run $\rho = t_0 t_1 \cdots t_n \cdots$ (over some infinite word), the set $lim(\rho)$ of all transitions that appear infinitely often in ρ is defined by $lim(\rho) = \bigcap_{n \in \mathbb{N}} \{t_i \mid i \geq n\}$. A run ρ is *accepting* if $lim(r) \in Acc$.

An infinite word $w \in \Sigma^\omega$ is *accepted* by A if there exists an accepting run ρ over w. The set of all infinite words accepted by A, called the *language of* A, is denoted $L(A)$. Given an ω-automaton A and a state q, we denote $A[q]$ the ω-automaton A with q as unique initial location. Observe that $L(A) = \bigcup_{q \in Q_0} L(A[q])$. If $Acc = 2^T$ then we have $L(A[q]) = Trace^\omega(A, q)$ for all $q \in Q_0$.

Recall that languages of ω-automata are measurable:

Theorem 3.1 ([7]). *For any ω-automaton A, the language $L(A)$ is a measurable subset of Σ^ω.*

We then deduce that the set of infinite paths (of a probabilistic system) satisfying an LTL formula is measurable, and hence its probability is well defined.

Corollary 3.2. *Given an ω-automaton A and a probabilistic system M, the set of infinite paths of M "accepted" by A (i.e. $Path^{\omega}(M) \cap \lambda^{-1}(L(A))$) is a measurable subset of T^{ω}.*

Probabilistic verification can be applied efficiently only when the considered ω-automata fulfill some structural properties. We will in particular consider upward closed acceptance conditions: an acceptance condition $Acc \subseteq 2^T$ is said to be *upward closed* if for every $U \in Acc$ we have $U' \in Acc$ for every $U' \supseteq U$. Observe that the usual Büchi and multi-Büchi acceptance conditions are upward closed. Local properties on the automaton's graph will also be helpful.

- A is *deterministic* if for every $t_1, t_2 \in T$ such that ${}^{\bullet}t_1 = {}^{\bullet}t_2$ and $\lambda(t_1) = \lambda(t_2)$ we have $t_1 = t_2$.
- A is *unambiguous* if for every $t_1, t_2 \in T$ such that ${}^{\bullet}t_1 = {}^{\bullet}t_2$ and $\lambda(t_1) = \lambda(t_2)$ we have $t_1 = t_2$ or $L(A[t_1{}^{\bullet}]) \cap L(A[t_2{}^{\bullet}]) = \emptyset$.
- A is *separated* if for every $q_1, q_2 \in Q$ such that $q_1 \neq q_2$ we have $L(A[q_1]) \cap L(A[q_2]) = \emptyset$.
- A is *transition duplicate free* if for every $t_1, t_2 \in T$ such that ${}^{\bullet}t_1 = {}^{\bullet}t_2$, $t_1{}^{\bullet} = t_2{}^{\bullet}$ and $\lambda(t_1) = \lambda(t_2)$ we have $t_1 = t_2$.

Given a subset $Q' \subseteq Q$, the *restriction of A to Q'* is the ω-automaton $A_{|Q'} = \langle Q', T', \alpha', \beta', \lambda', Q'_0, Acc' \rangle$ where $T' = {}^{\bullet}V' \cap V'^{\bullet}$, $\alpha' = \alpha_{|V'}$, $\beta' = \beta_{|V'}$, $\lambda' = \lambda_{|V'}$, $Q'_0 = Q_0 \cap Q'$ and $Acc' = \{X \subseteq T' \mid X \in Acc\}$. We say that A is *deterministic on Q'* (resp. *unambiguous on Q'*, *separated on Q'*) if $A_{|Q'}$ is deterministic (resp. unambiguous, separated).

Proposition 3.3. *Let A be an ω-automaton. The three following assertions hold:*

i) if A is deterministic then A is unambiguous,
ii) if A is separated and transition duplicate free then A is unambiguous.

3.3 From LTL to automata

We briefly present an adaptation of a translation from LTL to ω-automata given in [3]. This method produces ω-automata which are unambiguous, separated, and with upward closed acceptance conditions. The construction described here is slightly different than the classical ones (as [8]) by the notion of closure of a formula (interesting sub formulas): it contains only formulas associated to temporal operators. As a result, our technique produces automata with only $O(2^{|cl(f)|})$ states and $O(|\ \Sigma\ | \cdot 2^{|cl(f)|})$ transitions; this point is crucial to state the time complexity of the probabilistic verification.

Let f be an LTL formula. We define the *closure* of f as follows:

$$\begin{aligned} cl(a) &= \emptyset \text{ when } a \in \Sigma \\ cl(\neg f) &= cl(f) \\ cl(f \vee g) &= cl(f) \cup cl(g) \\ cl(Xf) &= \{f\} \cup cl(f) \\ cl(fUg) &= \{fUg\} \cup cl(f) \cup cl(g). \end{aligned}$$

Any LTL formula g can be expended as a boolean combination of letters and next formulas (Xh with $h \in cl(g)$). Letters represent what has to be true immediatly, and next formulas represent what has to be true in the next state. The fundamental assertion, which is used for this expansion, is : $gUh \equiv h \vee (g \wedge X(gUh))$. Based on this principle, we define, for any subformula g of f, the next predicate $\Phi_g(b, v)$, with $b \in \Sigma$ and $v \in 2^{cl(f)}$ as follows:

$$\begin{aligned}
\Phi_a(b, v) &= \mathrm{P}_a(b, v) \text{ when } a \in \Sigma \\
\Phi_{\neg g}(b, v) &= \neg \Phi_g(b, v) \\
\Phi_{g \vee h}(b, v) &= \Phi_g(b, v) \vee \Phi_h(b, v) \\
\Phi_{Xg}(b, v) &= \mathrm{Next}_g(b, v) \\
\Phi_{gUh}(b, v) &= \Phi_h(b, v) \vee (\Phi_g(b, v) \wedge \mathrm{Next}_{gUh}(b, v))
\end{aligned}$$

where $\mathrm{P}_a(b, v)$ and $\mathrm{Next}_g(b, v)$ are two predicates defined by: $\mathrm{P}_a(b, v) = (a = b)$ and $\mathrm{Next}_g(b, v) = g \in v$.

The interpretation of next predicate is made clear with the following proposition.

Lemma 3.4. *Let $w = a_0 a_1 \cdots a_i \cdots$ be an infinite word over Σ. Then, for any subformula g of f, and for any position i :*

$$w, i \models g \quad \textit{iff} \quad \Phi_g(a_i, \{h \in cl(f) \mid w, i+1 \models h\}) \textit{ is satisfied.}$$

For the automaton construction, each location is labelled by a set of formulas, and each transition is designed using the next predicate in accordance with lemma 3.4. Moreover, the acceptance takes into account the fair constraint of the "until" formulas. Given an LTL formula f, we define the ω-automaton $A_f = \langle Q, T, \alpha, \beta, \lambda, Q_0, Acc \rangle$ as follows:

- $Q = \{f\} \cup 2^{cl(f)}$,
- $T = \{(f, a, v) \in \{f\} \times \Sigma \times 2^{cl(f)} \mid \Phi_f(a, v)\} \cup$
  $\{(u, a, v) \in 2^{cl(f)} \times \Sigma \times 2^{cl(f)} \mid \forall g \in cl(f), g \in u \text{ iff } \Phi_g(a, v)\}$,
- for all $(u, a, v) \in T$, $\alpha(u, a, v) = u$, $\beta(u, a, v) = v$, and $\lambda(u, a, v) = a$,
- $Q_0 = \{f\}$,
- for all $R \subseteq T$, $R \in Acc$ if $R \cap Acc_{gUh} \neq \emptyset$ for every $gUh \in cl(f)$, where
  $Acc_{gUh} = \{(u, a, v) \in T \mid u \in 2^{cl(f)} \wedge (gUh) \in u \Rightarrow \Phi_h(a, v)\}$.

Proposition 3.5. *For every LTL formula f, we have $L(f) = L(A_f)$. Moreover, A_f is unambiguous, separated on each SCC, and has an upward closed acceptance condition.*

The acceptance condition Acc of A_f is entirely defined by the sets Acc_{gUh} with $gUh \in cl(f)$. Hence, in the following proposition, the size of Acc is defined as the size of these sets.

Proposition 3.6. *For every LTL formula f, the size of $A(f)$ and its computation time are in $O(\mid \Sigma \mid \cdot \mid cl(f) \mid \cdot 2^{|cl(f)|})$.*

4 Probabilistic Verification

This section presents our optimal automata based approach to verification of probabilistic systems.

Linear time probabilistic verification consists in checking that the set of all paths in a given probabilistic system almost surely satisfies a given (linear time) specification. Formally, the *LTL probabilistic verification problem* (resp. *ω-regular probabilistic verification problem*) is the set of all pairs (M, f) (resp. (M, A)) where M is a probabilistic system and f is an LTL formula (resp. A is an ω-automaton) such that $\mu_M(Path^\omega(M) \cap \lambda_M^{-1}(L(f))) = 1$ (resp. such that $\mu_M(Path^\omega(M) \cap \lambda_M^{-1}(L(A))) = 1$).

Automata based approaches to verification usually consider the dual problem, where the negation of the specification is considered. We follow the same principles, and we reduce the probabilistic verification problem to the so-called probabilistic satisfaction problem, which consists in checking that the set of all paths in a given probabilistic system satisfies a given (linear time) specification with positive probability. Formally, the *LTL probabilistic satisfaction problem* (resp. *ω-regular probabilistic satisfaction problem*) is the set of all pairs (M, f) (resp. (M, A)) where M is a probabilistic system and f is an LTL formula (resp. A is an ω-automaton) such that $\mu_M(Path^\omega(M) \cap \lambda_M^{-1}(L(f))) > 0$ (resp. such that $\mu_M(Path^\omega(M) \cap \lambda_M^{-1}(L(A))) > 0$).

4.1 Synchronized Product

Classical automata based verification methods first construct the synchronized product of the system and of the (negated) specification ω-automaton. This synchronized product is basically a cartesian product of the two graphs, keeping only the transition pairs having the same label ; viewed as an ω-automaton over transitions of the system, it recognizes precisely the set of all infinite paths in the system which fulfill the specification ω-automaton. We give the definition of the synchronized product, recall some basic properties and fix some notations.

From now on, we will consider a probabilistic system $M = \langle S, T_M, \alpha_M, \beta_M, \lambda_M, P_0, P\rangle$ and an ω-automaton $A = \langle Q, T_A, \alpha_A, \beta_A, \lambda_A, Q_0, Acc\rangle$.

The *projection onto* M (resp. *projection onto* A) is the mapping π_M (resp. π_A) defined on $(S \times Q) \cup (T_M \times T_A)$ by $\pi_M(z_M, z_A) = z_M$ (resp. $\pi_A(z_M, z_A) = z_A$).

Definition 4.1. *The* synchronized product *of M and A is the ω-automaton $M \otimes A = \langle S \times Q, T_\otimes, \alpha_\otimes, \beta_\otimes, \lambda_\otimes, S_0 \times Q_0, Acc_\otimes\rangle$ where:*

- *$T_\otimes$ is defined by $T_\otimes = \{(t_M, t_A) \in T_M \times T_A \mid \lambda_M(t_M) = \lambda_A(t_A)\}$, and*
- *$\alpha_\otimes$ and $\beta_\otimes$ are defined by ${}^\bullet(t_M, t_A) = ({}^\bullet t_M, {}^\bullet t_A)$ and $(t_M, t_A)^\bullet = (t_M{}^\bullet, t_A{}^\bullet)$, and*
- *$\lambda_\otimes$ is the restriction of π_M to $T_\otimes$, and*
- *$Acc_\otimes$ is equal to $\{U \subseteq T_\otimes \mid \pi_A(U) \in Acc\}$.*

Remark 4.2. As expected, we have the following equality:

$$L(M \otimes A) = Path^{\omega}(M) \cap \lambda_M^{-1}(L(A)).$$

We want to determine whether M satisfies A with positive probability, i.e. whether $\mu_M(L(M \otimes A)) > 0$. The language $L(M \otimes A)$ can be written as a union of $L(M \otimes A[s, q])$ where (s, q) are initial locations. Hence, the probabilistic satisfaction problem reduces to checking whether $\mu_M(L(M \otimes A[s, q])) > 0$ for some $(s, q) \in S_0 \times Q_0$.

For convenience, we will shortly write $L(s, q)$ instead of $L(M \otimes A[s, q])$, and we define the mapping $V : S \times Q \to [0, +\infty[$ as $V(s, q) = \mu_{M[s]}(L(s, q))$. Note that we also have : $L(s, q) = Path^{\omega}(M[s]) \cap \lambda^{-1}(L(A[q]))$. Moreover, as $M \otimes A$ is an ω-automaton, it is readily seen that for all $(s, q) \in S \times Q$, we have:

$$L(s, q) = \bigcup_{(t_M, t_A) \in (s,q)^{\bullet}} t_M \cdot L(t_M{}^{\bullet}, t_A{}^{\bullet}).$$

We deduce from the previous equality that $V(s, q) > 0$ iff $V(s', q') > 0$ for some (s', q') reachable from (s, q) (i.e. $(s, q) \xrightarrow{*} (s', q')$). Hence, we obtain that $V^{-1}(]0, +\infty[)$ is downward closed w.r.t. $\xrightarrow{*}$, and each SCC is either contained in $V^{-1}(]0, +\infty[)$ or contained in $V^{-1}(\{0\})$. We come to the following fundamental definitions:

Definition 4.3. *An SCC C of $M \otimes A$ is called:*

- null *if $C \subseteq V^{-1}(\{0\})$,*
- persistent *if C is an SCC which is maximal among the non null SCCs,*
- transient *otherwise.*

It turns out that probabilistic verification reduces to checking the existence of a reachable non null SCC, or, equivalently, a reachable persistent SCC. Unfortunately, these notions are not *local*: for instance, persistence of a given SCC depends on the other SCCs of $M \otimes A$. In order to perform probabilistic verification efficiently, we investigate local notions on SCCs.

Given a SCC C of $M \otimes A$ and a location $(s, q) \in C$, we define $L_C(s, q)$ and $V_C(s, q)$ by: $L_C(s, q) = L((M \otimes A)_{|C}[s, q])$ and $V_C(s, q) = \mu_{M[s]}(L_C(s, q))$.

Definition 4.4. *An SCC C of $M \otimes A$ is called* locally positive *if $V_C(s, q) > 0$ for all $(s, q) \in C$.*

Remark 4.5. Any SCC C is locally positive iff $V_C(s, q) > 0$ for some $(s, q) \in C$.

It is readily seen that every persistent SCC is locally positive. The converse does not hold, however every locally positive SCC is non null. Hence, we come to the following proposition:

Proposition 4.6. *We have $\mu_M(L(M \otimes A)) > 0$ iff there exists a locally positive SCC C that is reachable from an initial location of $M \otimes A$.*

4.2 Probabilistic Satisfaction

As locally positive SCCs play a major role in probabilistic verification, we look for an easily checkable characterization of locally positive SCCs. This characterization is based on two properties on SCCs: *completeness* and *acceptance*. We show that when A is unambiguous on each SCC or has an upward closed acceptance condition, then for any SCC C of $M \otimes A$, C is locally positive iff C is complete and accepted. We then prove that completeness and acceptance can be checked efficiently when A is unambiguous and separated (which is the case for ω-automata obtained through our translation of LTL formulas).

Definition 4.7. *Let C be an SCC in $M \otimes A$. C is said to be complete if for all $s \in \pi_M(C)$, we have:*

$$Path^*(M, s) = \bigcup_{q \in Q \ | \ (s,q) \in C} Trace^*((M \otimes A)_{|C}, (s, q)). \tag{1}$$

Remark 4.8. Any SCC C is complete iff (1) is satisfied for some $s \in \pi_M(C)$.

We say that an SCC C is accepted when the set of transitions of A that are "contained" in C fulfills the acceptance condition of A, formally:

Definition 4.9. *An SCC C in $M \otimes A$ is* accepted *if* $({}^{\bullet}C \cap C^{\bullet}) \in Acc_{\otimes}$.

The following result shows that every complete and accepted SCC is locally positive. The proposition actually states a stronger result that will prove useful for evaluation.

Proposition 4.10. *Let C be an SCC in $M \otimes A$. If C is both complete and accepted, then for all $s \in \pi_M(C)$ we have:*

$$\mu_{M[s]} \left(\bigcup_{q \in Q \ | \ (s,q) \in C} L_C(s, q) \right) = 1.$$

The two following propositions show that the converse holds when A is unambiguous on each SCC or has an upward closed acceptance condition.

Proposition 4.11. *If C is a locally positive SCC in $M \otimes A$, then C is complete.*

Proposition 4.12. *If A is unambiguous on C or A has an upward closed acceptance condition and if C is locally positive then C is accepted.*

We have proved so far that when A is unambiguous on each SCC or has an upward closed acceptance condition, then the two following assertions are equivalent:

i) M satisfies A with positive probability,
ii) there exists a complete and accepted SCC reachable from some initial location.

Observe that checking whether an SCC C is accepted amounts to checking whether $({}^\bullet C \cap C^\bullet) \in Acc_\otimes$. We now show that completeness of an SCC can also be checked efficiently when for every SCC C of $M \otimes A$, $M \otimes A$ is both unambiguous and separated on C, or deterministic on C. Basically, under these assumptions, completeness checking reduces to counting transitions, as expressed by the following lemmas.

Lemma 4.13. *Let C be an SCC in $M \otimes A$. If $M \otimes A$ is deterministic on C then the two following properties are equivalent:*

- *C is complete,*
- *for all $(s,q) \in C$ and $t_M \in s^\bullet$, we have $\mid \pi_M^{-1}(t_M) \cap (s,q)^\bullet \cap {}^\bullet C \mid = 1$.*

Lemma 4.14. *Let C be an SCC in $M \otimes A$. If $M \otimes A$ is unambiguous on C and separated on C then the two following properties are equivalent:*

- *C is complete*
- *for all $(s,q) \in C$, $SCC(s)$ is maximal and*

$$\sum_{s \in \pi_M(C)} \mid {}^\bullet s \cap SCC(s)^\bullet \mid \cdot \mid \pi_M^{-1}(s) \cap C \mid \quad = \quad \mid {}^\bullet C \cap C^\bullet \mid .$$

The following theorems follow from the previous propositions, and from the translation from LTL to ω-automata presented in section 3.3.

Theorem 4.15. *Assume that for every SCC C of $M \otimes A$, $M \otimes A$ is both unambiguous and separated on C, or deterministic on C. Then checking whether M satisfies A with positive probability can be done in $K_{Acc} \cdot O(\mid M \otimes A \mid)$.*

Theorem 4.16. *Let f be an LTL formula. Checking whether M satisfies f with positive probability can be done in $O(\mid cl(f) \mid) \cdot O(\mid M \mid) \cdot O(2^{\mid cl(f) \mid})$.*

Remark 4.17. Checking whether M satisfies f with positive probability can be done in polynomial space.

5 Probabilistic Evaluation

The evaluation problem may be studied reasonably when automata are non ambiguous: this property induces that $V(s,q)$ probabilities fulfill a linear equation system. The system may be solved on each non null SCC with respect to a topology order. We state the nature of this linear equation system: for transient SCC, the system has a unique solution, while for persistent SCC, only one equation is necessary to complete the resolution. We give some missing linear properties when the persistent SCC is separated or deterministic. One can notice that these results lead to a technique for the evaluation problem for non ambiguous and SCC separated automaton, in particularly a method to evaluate LTL formulas.

Proposition 5.1. *Let A be unambiguous. Let C be strongly connected component in $M \otimes A$. Let E_C be the equation system*

$$V(s,q) = \sum_{(t_M,t_A)\in(s,q)^\bullet} P(t_M)\cdot V({t_M}^\bullet, {t_A}^\bullet)$$

with $(s,q) \in C$. If C is persistent then $rank(E_C) = |C| - 1$. If C is transient then $rank(E_C) = |C|$.

Proposition 5.2. *Let C be a persistent component in $M \otimes A$. Then*

- *if C is deterministic:* $\forall (s,q) \in C : V(s,q) = 1$,
- *if C is separated:* $\sum_{q:(s,q)\in C} V(s,q) = 1$.

6 Experimentation, the tool ProbaTaf

We have implemented our techniques in the ProbaTaf tool. The probabilistic systems are described by (bounded) Petri nets. The LTL formulas are defined from transitions, propositions on marking (e. g. "P" for Place P marked) and proposition *dead* for each dead transition attach to any dead markings. The tool allows checking the satisfaction of a formula and its evaluation. Several advanced techniques have been applied:

- Binary decision diagram (bdd) technology to compute, to store and to simplify the ω-automaton for a formula. The enumeration of the successor locations for a given location and values of the propositions may be obtained directly from the bdd structure after some bdd operations.
- An on-the-fly algorithm based on [2].
- A simple Gauss elimination algorithm to solve linear systems associated to each persistent and transient SCC: we may select all pivots on the diagonal.

We conclude the study with two illustrative examples of applications. They are not exhaustive and do not present elaborated use of the techniques. Our goal is rather to display shortly the scope of the presented models.

Example 6.1. The first example is borrowed from a paper by Knuth and Yao [4]. They introduce a number of essential questions related to the generation of random variables with nonuniform distributions. The first tackled technique is on the generation of a uniform random variable taking values in a finite set. We consider a variant of their introductory example. Suppose that one wants to simulate a (uniform) dice game by a number of flips of a possibly biased coin. We toss four times the coin and discard the result if the numbers of heads and that of tails differ. Otherwise (two heads and two tails) we let : 0011 → o, 0101 → oo, 0110 → ooo, 1001 → oooo, 1010 → ooooo, 1100 → oooooo (head is encoded as 1 and tail as 0). When the three first tosses are identical, we preventively discard the result.

Using the ProbaTaf tool, we easily verify the uniform property of the game for possibly biased coin. However, we have observed that the expected number of flips grows with the biasness of the coin. To confirm this assertion, we have considered the property "get a value with at most n flips", expressed in LTL by the formula $X^n dead$, and computed its probability for different biased coins. However, the satisfaction and the evaluation of this formula lead to an exponential time computation (see Table 1 where times are expressed in milliseconds). Indeed, the automata, produced by our techniques, have an exponential number of states. Adding the tautology $G(dead \Rightarrow X dead)$ to our formula lead to linear time computation (see Table 1) : the automata have linear sizes. This experimentation brings to the fore the well-known state explosion problem of model-checking. This problem may be partially solved by adapting the classical LTL automata construction (as [2]) to produce non-ambiguous automata.

n		3	4	5	6	7	8	9	10	11	12	13
$X^n dead$	sat	2	2	11	46	99	268	836	1815	4205	9713	19686
	eval	10	24	36	73	161	378	977	2114	5053	9665	21057
$X^n dead \wedge G(dead \Rightarrow X dead)$	sat	1	1	2	4	3	6	10	6	10	13	9
	eval	2	6	6	8	11	23	26	32	40	50	56

Table 1. Time performance of sat. and eval. problem applied to dice game

Example 6.2. In [6], the authors design a probabilistic one-passage distributed algorithm in trees which eliminates leaves one-by-one until the tree is reduced to a single vertex, considered as the elected vertex. The main particularity of the election algorithm is that it assigns to *all* vertices of the tree graph the same probability of being elected, by giving to each leaf or vertex which has become a leaf a life duration exponentially distributed with a parameter computed as follows. Initially all vertices have the same weight 1. Any leaf of the tree generates its lifetime, which is a positive-real-valued random variable with an exponential distribution of parameter equal to its weight. Once the lifetime has expired the leaf is eliminated and its father recuperates the weight of the removed son. The process continues until the tree is reduced to one vertex.

Using the ProbaTaf tool, we observe the uniform property of the election algorithm for some trees. This experimental verification consists in evaluating the probability of formula $F(dead \wedge E_i)$ where E_i states that the vertex i is still a candidate. By accident, during our experimentation, we obtained a surprising result: for a given tree, the probability of being a finalist vertex (i.e. one of the two last candidate) depends only on the degree of the vertex. A formal proof of this result is out the scope of our study and is left as a conjecture. We observe this property on Table 2 for the balanced binary tree of depth 5, where the index i represents the depth of a vertex.

Some performance experimentation has been over made the election algorithm applied to balanced binary trees. The size of the probabilistic system grows in double exponential with the depth of the tree and leads to 459829 states and

i	0	1	2	3	4
$F(E_i \wedge X(dead \wedge \neg E_i))$	0.034	0.068	0.068	0.068	0.0010

Table 2. Probability to be second on balanced binary tree of depth 5

3599198 transitions for the depth 5. Table 3 displays the time computation for the satisfaction and evaluation algorithm. The time performance difference between these two problems is mainly explained by the on-the-fly technology used to solve the satisfaction problem, amplified by the cost of linear systems to be solved for the evaluation problem.

deaph		2	3	4	5
$F(dead \wedge E_0)$	sat	3	7	8	4841
	eval	21	19	94	99797
$F(E_0 \wedge X(dead \wedge \neg E_0))$	sat	3	3	19	4894
	eval	17	28	242	336821

Table 3. Time performance for sat. and eval. problems applied to election algorithm

References

[1] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, July 1995.

[2] Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. In *FM'99—Formal Methods, Volume I*, volume 1708 of *Lecture Notes in Computer Science*, pages 253–271. Springer, 1999.

[3] Jean-Michel Couvreur. Un point de vue symbolique sur la logique temporelle linéaire. In *Actes du Colloque LaCIM 2000*, volume 27 of *Publications du LaCIM*, pages 131–140. Université du Québec à Montréal, August 2000.

[4] Knuth and Yao. The complexity of nonuniform random number generation. In *Algorithms and Complexity: New Directions and Recent Results, Ed. J. F. Traub.* Academic Press, 1976.

[5] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.

[6] Yves Métivier, Nasser Saheb, and Akka Zemmari. A uniform randomized election in trees. In *SIROCCO 10*, volume 17 of *Proceedings in Informatics*, pages 259–274. Carleton Scientific, 2003.

[7] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *focs85*, pages 327–338, 1985.

[8] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.