Journal of Computer and System Sciences 136 (2023) 113–134

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

journal homepage: www.elsevier.com/locate/jcss

Markov chains and unambiguous automata

Christel Baier [a,1], Stefan Kiefer [b,*,2], Joachim Klein [a,1], David Müller [a,1], James Worrell [b,3]

[a] Technische Universität Dresden, Germany
[b] University of Oxford, United Kingdom

ARTICLE INFO

Article history:
Received 11 May 2019
Received in revised form 24 October 2022
Accepted 23 March 2023
Available online 13 April 2023

Dataset link: https://wwwtcs.inf.tu-dresden.de/ALGI/TR/JCSS19/

Keywords:
Model checking
Probabilistic verification
Unambiguous automata
Markov chains

ABSTRACT

Unambiguous automata are nondeterministic automata in which every word has at most one accepting run. In this paper we give a polynomial-time algorithm for model checking discrete-time Markov chains against ω-regular specifications represented as unambiguous automata. We furthermore show that the complexity of this model checking problem lies in NC: the subclass of P comprising those problems solvable in poly-logarithmic parallel time. These complexity bounds match the known bounds for model checking Markov chains against specifications given as deterministic automata, notwithstanding the fact that unambiguous automata can be exponentially more succinct than deterministic automata. We report on an implementation of our procedure, including an experiment in which the implementation is used to model check LTL formulas on Markov chains.

1. Introduction

Unambiguity is a generalization of determinism that has been widely studied in the theory and applications of automata [14,15]. In this paper we are concerned with unambiguous automata over infinite words, that is, nondeterministic automata in which every word has at most one accepting run. Our main results hold for the most commonly occurring acceptance conditions (Büchi, Rabin, Muller, etc.), however in our examples and experimental results we focus on the case of unambiguous Büchi automata (UBA). An example of a UBA is the automaton on the right-hand side of Fig. 1 in which both states are initial and accepting. This automaton is unambiguous by virtue of the fact that there is exactly one run over every word.

Over infinite words, not only are UBA as expressive as nondeterministic Büchi automata [1], they can also be exponentially more succinct than deterministic automata. For example, for a fixed $k \in \mathbb{N}$ the language "eventually b occurs and a appears k steps before the first b" over the alphabet $\{a, b, c\}$ is recognized by a UBA with $k+1$ states (shown on the left-hand side of Fig. 1). On the other hand, a deterministic automaton for this language requires at least 2^k states, regardless of the acceptance condition, as it needs to store the positions of the a's among the last k input symbols. Languages of this type

* Corresponding author.
E-mail address: stefan.kiefer@cs.ox.ac.uk (S. Kiefer).

[1] The authors are supported by the DFG through the Collaborative Research Center TRR 248 (see https://perspicuous-computing.science, project ID 389792660), the DFG project BA-1679/12-1, the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), and the Research Training Group QuantLA (GRK 1763).
[2] Kiefer is supported by a University Research Fellowship of the Royal Society.
[3] Worrell is supported by EPSRC grant EP/M012298/1.

https://doi.org/10.1016/j.jcss.2023.03.005

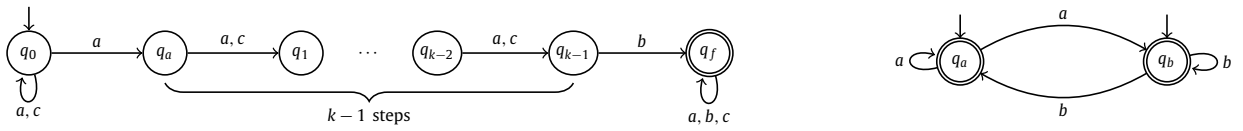

Fig. 1. Left: UBA for "eventually b and a appears k steps before first b", right: A universal and separated UBA.

arise in a number of contexts, e.g., absence of unsolicited response in a communication protocol—if a message is received, then it has been sent in the recent past.

The exponential succinctness of UBA relative to deterministic automata is also manifested in translations of linear temporal logic (LTL) to automata. The nondeterministic Büchi automata that are obtained from LTL formulas by applying the classical closure algorithm of [48,47] are unambiguous. The generated automata moreover enjoy the separation property: different states have disjoint languages. Thus, while the generation of deterministic ω-automata from LTL formulas incurs a double-exponential blow-up in the worst case, the translation of LTL formulas into separated UBA incurs only a single exponential blow-up. This fact has been observed by several authors, see e.g. [18,39], and adapted for LTL with step parameters [49,13]. Besides allowing exactly one accepting run for every accepted word, there exist weaker notions of unambiguity, where one allows a finite number of runs, or a polynomial number of runs [43,26]. These forms of unambiguity led to applications in complementation of Büchi automata with a finite number of accepting runs [42].

In the context of probabilistic model checking, UBA provide an elegant alternative to deterministic automata for computing probabilities of ω-regular properties on finite-state Markov chains. A polynomial-time model checking procedure for UBA that represent safety properties was given [6], while [18] gives a polynomial-time algorithm for separated UBA. However, separation is a strong restriction, as non-separated UBA (and even DBA) can be exponentially more succinct than separated UBA, see [11]. Furthermore, algorithms for the generation of (possibly non-separated) UBA from LTL formulas that are more compact than the separated UBA generated by the classical closure algorithm have been realized in the tool `Tulip` [38,37] and the automata library `SPOT` [19]. This motivates the design of algorithms that operate with general UBA rather than the subclass of separated UBA. For the analysis of finite-state Markov decision processes under ω-regular properties, there exist restricted forms of nondeterministic automata such as limit-deterministic automata [46,17,44] or good-for-games automata [27,32].

The main theoretical contribution of this paper is a polynomial-time algorithm to compute the probability that the trajectory generated by a given finite Markov chain satisfies an ω-regular property specified by a (not necessarily separated) unambiguous automaton. We present this algorithm under a mild assumption on the acceptance condition of the automaton, which is satisfied by commonly occurring acceptance conditions such as Büchi, Rabin, Muller, etc. Furthermore we use our procedure to show that the model checking problem of finite Markov chains against unambiguous automata lies in the complexity class NC: the subclass of P comprising those problems solvable in poly-logarithmic time by a parallel random-access machine using polynomially many processors.

The existence of a polynomial-time algorithm for model checking Markov chains against UBA has previously been claimed in [7,6,38] (see also [37]). However, these previous works share a common fundamental error. Specifically they rely on the claim that if the language of a given UBA $\mathcal{A}$ has positive probability with respect to a Markov chain $\mathcal{M}$, then there exists a state s of $\mathcal{M}$ and a state q of $\mathcal{A}$ such that q accepts almost all trajectories emanating from s (see [7, Lemma 7.1], [6, Theorem 2][4] and [38, Section 3.3.1]). While this claim is true in case $\mathcal{A}$ is deterministic [17], it need not hold when $\mathcal{A}$ is merely unambiguous (see Remark 4.1). We refer the reader to [3] for full details and counterexamples to incorrect claims in these works.

As a corollary of the above-mentioned NC bound we obtain another proof of the fact that model checking LTL formulas on Markov chains can be done in polynomial space (see [12] for a proof of this fact using weak alternating automata and see [16,17] for an automata-free approach). Another corollary of our main result is that one can decide in NC whether a given UBA accepts almost all words with respect to the uniform distribution on ω-words. Recall that while checking universality is known to be in NC for deterministic Büchi automata and PSPACE-complete for NBA, determining the complexity of the universality problem for UBA is a long-standing open problem. Polynomial-time procedures are only known for separated UBA and other subclasses of UBA [11,30].

A second contribution of our paper is an implementation of the new algorithm as an extension of the model checker `PRISM`, using the automata library `SPOT` [19] for the generation of UBA from LTL formulas and the `COLT` library [29] for various linear algebra algorithms. We focus on unambiguous automata with a Büchi acceptance condition. We evaluate our approach using the bounded retransmission protocol case study from the PRISM benchmark suite [36] as well as specific aspects of our algorithm using particularly "challenging" UBA.

We remark that there cannot exist a polynomial-time algorithm for model checking Markov decision processes (MDPs) against UBA. This is because model checking LTL formulas on MDPs is 2EXPTIME-complete [17] and there is a single-exponential procedure for translating LTL formulas into UBA (cf. the proof of Corollary 20).

[4] As the flaw is in the handling of the infinite behavior, the claim and proof of Lemma 1 in [6], dealing with unambiguous automata over finite words, remain unaffected.

The rest of this article is structured as follows. In Section 2 we provide the necessary definitions and quote standard results on the spectral theory of nonnegative matrices. In Section 3 we give an overview of our methodology. Section 4 contains the main technical development with a polynomial-time model checking procedure. Key technical lemmas are proved in Sections 5 and 6. Section 7 improves the main result to an NC model checking procedure. In Section 8 we describe our implementation and experiments. We conclude in Section 9.

This article is a revised version of the CAV'16 conference paper [3] and its extended version on arxiv [4]. The main differences are that we present here a direct proof technique, while [3,4] first explain how to compute the measure of the language induced by strongly connected UBA and then how to extend these techniques to arbitrary UBA and the probabilistic model checking problem as discussed here. In contrast to [3,4] we consider here acceptance conditions beyond Büchi acceptance. Furthermore, the material on experiments has been extended.

2. Preliminaries

We assume the reader to be familiar with basic notions of Markov chains and finite automata over infinite words, see, e.g., [25,34] and complexity theory, see, e.g., [41]. In what follows, we provide a brief summary of our notation for words, finite automata, vectors and matrices, and Markov chains. We also briefly summarize basic facts on the complexity class NC and collect facts in the spectral theory of nonnegative matrices.

Words Throughout the article, we suppose that Σ is a finite non-empty alphabet. For $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^\omega$ we write $L_1 \cdot L_2$ for the concatenation of L_1 and L_2, i.e., $L_1 \cdot L_2 = \{vw \in \Sigma^\omega : v \in L_1,\ w \in L_2\}$. If $L_1 = \{v\}$ for some $v \in \Sigma^*$ then we may write vL_2 for $L_1 \cdot L_2$.

Finite automata A (nondeterministic finite) automaton (over infinite words) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, Acc)$ where Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, Σ is the alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $Acc \subseteq 2^Q$ is the (Muller) acceptance condition. We extend the transition function to $\delta : Q \times \Sigma^* \to 2^Q$ and to $\delta : 2^Q \times \Sigma^* \to 2^Q$ in the standard way. Given states $q, r \in Q$ and a finite word $w = a_0 a_1 \cdots a_{n-1} \in \Sigma^*$, a *run* for w from q to r is a sequence $q_0 q_1 \cdots q_n \in Q^{n+1}$ with $q_0 = q$, $q_n = r$ and $q_{i+1} \in \delta(q_i, a_i)$ for $i \in \{0, \ldots, n-1\}$. A *run* in $\mathcal{A}$ for an infinite word $w = a_0 a_2 a_3 \cdots \in \Sigma^\omega$ is an infinite sequence $\rho = q_0 q_1 \cdots \in Q^\omega$ such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, a_i)$ for all $i \in \mathbb{N}$. Run ρ is called *accepting* if $\inf(\rho) \in Acc$ where $\inf(\rho) \subseteq Q$ is the set of states that occur infinitely often in ρ. The *language* $\mathcal{L}(\mathcal{A})$ of accepted words consists of all infinite words $w \in \Sigma^\omega$ that have at least one accepting run. If $R \subseteq Q$ then $\mathcal{A}[R]$ denotes the automaton $\mathcal{A}$ with R as set of initial states. If $\mathcal{A}$ is understood from the context and $q \in Q$ then we may write $\mathcal{L}_q$ for $\mathcal{L}(\mathcal{A}[\{q\}])$. $\mathcal{A}$ is called *deterministic* if Q_0 is a singleton and $|\delta(q,a)| \leqslant 1$ for all $q \in Q$ and $a \in \Sigma$, and *unambiguous* if each word $w \in \Sigma^\omega$ has at most one accepting run in $\mathcal{A}$. Clearly, each deterministic automaton is unambiguous.

We assume that, given any set $R \subseteq Q$, one can compute in polynomial time whether $R \in Acc$. This is the case, e.g., if Acc is given as a *Büchi* condition, i.e., as a set $F \subseteq Q$ of *accepting* states such that $R \in Acc$ if and only if $R \cap F \neq \emptyset$. We use the acronym UBA for unambiguous Büchi automata.

We say that an automaton $\mathcal{A}$ has a *diamond* from q to r (where $q, r \in Q$) if there is a finite word $w \in \Sigma^*$ such that there are at least two different runs for w from q to r. If $\mathcal{A}$ has no diamonds, we say that $\mathcal{A}$ is *diamond-free*. If $\mathcal{A}$ is an unambiguous automaton then one can make it diamond-free in polynomial time and even in NC: First remove all states that are unreachable from Q_0, along with their incoming and outgoing transitions. Then compute, in polynomial time, all states q, r such that there is a diamond from q to r. By unambiguousness, we have $\mathcal{L}_r = \emptyset$, so we can remove r from $\mathcal{A}$, along with all incoming and outgoing transitions, without changing $\mathcal{L}(\mathcal{A})$. Therefore we can and will generally assume that unambiguous automata are diamond-free.

Complexity theory Let $\{C_n\}_{n\in\mathbb{N}}$ be a family of Boolean circuits such that C_n has n input gates and any number of output gates. Such a family is said to be *uniform* if there is a $\log n$-space bounded Turing machine which on input 1^n outputs C_n. Such a family is moreover said to compute a function $f : \{0,1\}^* \to \{0,1\}^*$ if for each $n \in \mathbb{N}$ and every word $x \in \Sigma^n$ the output of C_n on input x equals $f(x)$. Function f is said to be computable in NC if there exists a positive integer d such that f is computed by a uniform family of circuits $\{C_n\}_{n\in\mathbb{N}}$ where C_n has depth $O(\log^d n)$. NC is widely considered as the subclass of polynomial-time computable functions comprising those functions that can be computed efficiently (i.e., in poly-logarithmic time) in parallel (see, e.g., [41, Chapter 15]).

A standard result of complexity theory is that a function computable in NC is computable by a Turing machine using poly-logarithmic space [9, Theorem 4]. Two facts that will be used below are that reachability in directed graphs and matrix determinants (and hence solving systems of linear equations) are both computable in NC [41].

Vectors and matrices We consider vectors and square matrices indexed by a finite set S. We use boldface for (column) vectors such as $\vec{v} \in \mathbb{R}^S$, and write $\vec{v}^\top$ for the transpose (a row vector) of $\vec{v}$. The zero vector and the all-ones vector are denoted by $\vec{0}$ and $\vec{1}$, respectively. A matrix $M \in [0,1]^{S\times S}$ is called *stochastic* if $M\vec{1} = \vec{1}$, i.e., if every row of M sums to one. For a set $U \subseteq S$ we write $\vec{v}_U \in \mathbb{R}^U$ for the restriction of $\vec{v}$ to U. Similarly, for $T, U \subseteq S$ we write $M_{T,U}$ for the submatrix of M obtained by deleting the rows not indexed by T and the columns not indexed by U. The (directed) *graph* of a nonnegative matrix $M \in \mathbb{R}^{S\times S}$ has vertices $s \in S$ and edges (s,t) if $M_{s,t} > 0$. We may implicitly associate M with its graph and speak

about graph-theoretic concepts such as reachability and strongly connected components (SCCs) in M. For $s, t \in S$ we write $Paths_s(M) := \{s_0 \cdots s_n \in sS^* : \bigwedge_{i=0}^{n-1} M_{s_i,s_{i+1}} > 0\}$, $Paths_{s,t}(M) := \{s_0 \cdots s_n \in Paths_s(M) : s_n = t\}$, and $Paths_s^\omega(M) := \{s_0 s_1 \cdots \in sS^\omega : \bigwedge_{i=0}^{\infty} M_{s_i,s_{i+1}} > 0\}$.

Markov chains A (finite-state discrete-time) *Markov chain* is a pair $\mathcal{M} = (S, M)$ where S is the finite set of states, and $M \in [0,1]^{S\times S}$ is a stochastic matrix that specifies transition probabilities. An *initial distribution* is a function $\iota : S \to [0,1]$ satisfying $\sum_{s\in S} \iota(s) = 1$. Such a distribution induces a probability measure $\Pr_\iota^{\mathcal{M}}$ on the measurable subsets of S^ω in the standard way. We may write $\Pr_\iota$ for $\Pr_\iota^{\mathcal{M}}$ if $\mathcal{M}$ is understood. If ι is concentrated on a single state s then we may write $\Pr_s$ for $\Pr_\iota$. Note that $\Pr_s(Paths_s^\omega(M)) = 1$.

Spectral theory The *spectral radius* of a matrix $M \in \mathbb{R}^{S\times S}$, denoted $\rho(M)$, is the largest absolute value of the eigenvalues of M. The following result summarizes some facts in the spectral theory of nonnegative matrices that will be used in the sequel. In the formulation below, we restrict attention to right eigenvectors.

Theorem 1. *Let $M \in \mathbb{R}^{S\times S}$ be a nonnegative matrix. Then the following all hold:*

1. *The spectral radius $\rho(M)$ is an eigenvalue of M and there is a nonnegative eigenvector $\vec{x}$ with $M\vec{x} = \rho(M)\vec{x}$. Such a vector $\vec{x}$ is called* dominant.
2. *If $T \subseteq S$ then $\rho(M_{T,T}) \leq \rho(M)$.*
3. *There is $C \subseteq S$ such that $M_{C,C}$ is strongly connected and $\rho(M_{C,C}) = \rho(M)$.*

Theorem 2. *Let $M \in \mathbb{R}^{S\times S}$ be a strongly connected nonnegative matrix. We have the following facts:*

1. *There is an eigenvector $\vec{x}$ with $M\vec{x} = \rho(M)\vec{x}$ such that $\vec{x}$ is strictly positive in all components.*
2. *The eigenspace associated with $\rho(M)$ is one-dimensional.*
3. *If $T \subsetneq S$ then $\rho(M_{T,T}) < \rho(M)$.*
4. *If $\vec{x} \geqslant 0$ and $M\vec{x} \leqslant \rho(M)\vec{x}$ then $M\vec{x} = \rho(M)\vec{x}$.*
5. *If M is strictly positive, i.e., $M_{i,j} > 0$ for all $i, j \in S$, then $\lim_{i\to\infty} (M/\rho(M))^i$ exists and is strictly positive.*

These results can mostly be found in [8, Chapter 2]. Specifically, Theorems 1(1) and 2(1–2) are part of the Perron-Frobenius theorem, see [8, Theorems 2.1.1, 2.1.4]; Theorem 1(2) is [8, Corollary 2.1.6(a)]; Theorem 1(3) follows from [8, Corollary 2.1.6(b)]; Theorem 2(3) follows from [8, Corollary 2.1.6]; Theorem 2(4) follows from [8, Corollary 2.1.11]; Theorem 2(5) follows from [28, Theorem 8.2.7].

3. Overview of the methodology

Given a finite Markov chain $\mathcal{M}$ and an unambiguous automaton $\mathcal{A}$, our goal is to compute the probability that a trajectory generated by $\mathcal{M}$ is accepted by $\mathcal{A}$ for a given initial distribution ι. Suppose that $\mathcal{M}$ has set of states S and $\mathcal{A}$ has set of states Q. Let vector $\vec{z} \in \mathbb{R}^{Q\times S}$ be such that for each $q \in Q$ and $s \in S$, $\vec{z}_{\langle qs\rangle}$ is the probability that a trajectory of $\mathcal{M}$ starting in state s is accepted by $\mathcal{A}$ starting in state q. It suffices to compute $\vec{z}$.

Our strategy to compute $\vec{z}$ is to find a system of linear equations that has $\vec{z}$ as unique solution. To this end, the first step is to form the product of the transition functions of $\mathcal{A}$ and $\mathcal{M}$, thereby obtaining a nonnegative matrix $B \in \mathbb{R}^{(Q\times S)\times(Q\times S)}$, and then to show that $B\vec{z} = \vec{z}$. However this system of linear equations, being homogeneous, certainly does not determine $\vec{z}$ uniquely.

In case $\mathcal{A}$ is deterministic, it is relatively straightforward to write down extra linear equations that pin down $\vec{z}$ uniquely: one looks at the directed graph underlying matrix B and classifies the bottom SCCs of this graph as being either accepting or rejecting, according to the automaton states that appear in them. One then adds an equation $\vec{z}_{\langle qs\rangle} = 1$ for every pair (q, s) in an accepting bottom SCC and an equation $\vec{z}_{\langle qs\rangle} = 0$ for every pair (q, s) in a rejecting bottom SCC.

In order to generalize the above analysis to unambiguous automata we rely extensively on the spectral theory of nonnegative matrices. In particular, rather than looking at bottom SCCs of matrix B we focus on SCCs that induce submatrices of B with spectral radius one. We call the latter *recurrent SCCs*. Recurrent SCCs need not be bottom SCCs when $\mathcal{A}$ is unambiguous. We classify recurrent SCCs as being accepting or rejecting in similar manner to the deterministic case. For each pair (q, s) in a rejecting recurrent SCC we have an equation $\vec{z}_{\langle qs\rangle} = 0$. For an accepting recurrent SCC $D \subseteq Q \times S$ we do not in general have $\vec{z}_{\langle qs\rangle} = 1$ for each $(q, s) \in D$. Rather, writing $\vec{z}_D$ for the restriction of $\vec{z}$ to D, we have $\vec{\mu}^\top \cdot \vec{z}_D = 1$ for some weight vector $\vec{\mu} \in [0,1]^D$. While such a weight vector can be computed from the determinization of $\mathcal{A}$, we show how to compute a weight vector in polynomial time (and even NC) by exploiting structural properties of unambiguous automata. Given such a weight vector $\vec{\mu}$ we add the single linear equation $\vec{\mu}^\top \cdot \vec{z}_D = 1$ to our system and thereby ensure a unique solution.

3.1. Structure of Section 4

The following section, Section 4, follows this approach to prove our main result, Theorem 3. Section 4.1 sets up the basic linear system, see Lemma 4. The linear system can be written in matrix form as $\vec{\zeta} = B\vec{\zeta}$, see (4), where $\vec{\zeta}$ is a vector of variables. This system is satisfied by the vector $\vec{z}$ which contains the probabilities in question; i.e., we have $\vec{z} = B\vec{z}$. In Section 4.1, specifically in Proposition 6, we start deriving further properties of the matrix B. Matrix B can be viewed as a weighted graph representing a product of the automaton and the Markov chain. This dual view of B (on the one hand containing coefficients of the linear system, on the other representing the product as a weighted graph) drives the technical development in the rest of Section 4.

In Section 4.2 we first show, in Proposition 7, that the spectral radius of B is at most 1. Recurrent SCCs of B are defined to be those where the corresponding submatrix of B has spectral radius exactly 1. Recurrent SCCs are the analogues of bottom SCCs in the product of a deterministic automaton and a Markov chain. In Lemma 8 (proved in Section 5) we provide crucial properties of recurrent SCCs. Specifically we show that a coordinate of $\vec{z}$ is strictly positive if and only if it belongs to an accepting recurrent SCC (where accepting means that the set of automaton states associated with the SCC is accepting).

As mentioned above, the system $\vec{\zeta} = B\vec{\zeta}$ does not uniquely determine $\vec{z}$. Therefore, for each accepting recurrent SCC D, we add an equation $\vec{\mu}_D^{\top}\vec{\zeta}_D = 1$, where $\vec{\mu}_D \in [0,1]^D$ is a weight vector which we call *D-normalizer*. We show in Section 4.3 that such normalizers can be taken as the characteristic vectors of certain subsets of D called *cuts*. Again we benefit from a dual view: on the one hand cuts provide the necessary normalizing equations, on the other hand they describe a combinatorial property: intuitively, a cut is a minimal subset of D that cannot be "driven extinct" by the Markov chain. Lemma 10 shows important properties of cuts, including their polynomial-time computability. The algorithm (given in Section 6) is purely combinatorial and heavily exploits the diamond-freeness of the automaton.

With the necessary ingredients at hand, in Section 4.4 we give the full linear system, pinning down $\vec{z}$ uniquely; see Lemma 12. Then we prove the main theorem, Theorem 3, by giving the overall algorithm.

4. A polynomial-time model checking procedure

Given a Markov chain $\mathcal{M}$, an initial distribution ι, and an automaton $\mathcal{A}$ whose alphabet is the state space of $\mathcal{M}$, the *probabilistic model-checking problem* is to compute $\Pr_\iota(\mathcal{L}(\mathcal{A}))$. This problem is solvable in polynomial time in case $\mathcal{A}$ is a deterministic automaton and in polynomial space for nondeterministic automata [17,12]. The main result of this article extends the polynomial-time bound from deterministic automata to unambiguous automata.

Theorem 3. *Given a Markov chain $\mathcal{M}$, an initial distribution ι, and an unambiguous automaton $\mathcal{A}$, the value $\Pr_\iota(\mathcal{L}(\mathcal{A}))$ is computable in polynomial time.*

Remark 4.1. The statement of Theorem 3 has already been presented in [5] (see also [38,6]). However, the presented algorithm is flawed. The error stems from the incorrect claim that if $\Pr_\iota(\mathcal{L}(\mathcal{A}))$ is strictly positive then there is necessarily a state q of $\mathcal{A}$ and state s on $\mathcal{M}$ such that q accepts almost all trajectories emanating from s. A counterexample is obtained by taking $\mathcal{A}$ to be the automaton on the right in Fig. 1 and $\mathcal{M}$ the Markov chain that generates the uniform distribution on $\{a,b\}^\omega$. Clearly state q_a of $\mathcal{A}$ accepts all words that begin with a, while state q_b accepts all words that begin with b. Thus $\mathcal{A}$ is universal and its language has probability 1 under the uniform distribution. However each of the two languages $\mathcal{L}_{q_a}$ and $\mathcal{L}_{q_b}$ has probability $\frac{1}{2}$ under the uniform distribution.

The remainder of this section gives a proof of Theorem 3. The development heavily relies on two technical lemmas (Lemmas 8 and 10), whose proofs are given in Sections 5 and 6, respectively.

4.1. The basic linear system

Let $\mathcal{M} = (S, M)$ be a Markov chain, ι an initial distribution, and $\mathcal{A} = (Q, S, \delta, Q_0, Acc)$ an unambiguous automaton.

Lemma 4. *The following equations hold:*

$$\Pr_\iota(\mathcal{L}(\mathcal{A})) \;=\; \sum_{s\in S} \iota(s)\cdot \sum_{q\in Q_0} \Pr_s(\mathcal{L}_q) \tag{1}$$

$$\text{for all } q\in Q \text{ and } s\in S:\;\; \Pr_s(\mathcal{L}_q) \;=\; \sum_{t\in S}\sum_{r\in\delta(q,s)} M_{s,t}\cdot \Pr_t(\mathcal{L}_r) \tag{2}$$

Proof. For all $q\in Q$ and $s\in S$ we have $\mathcal{L}_q \cap sS^\omega = s\bigcup_{r\in\delta(q,s)}\mathcal{L}_r$. Hence:

$$\Pr_s(\mathcal{L}_q) \;=\; \Pr_s(\mathcal{L}_q\cap sS^\omega) \;=\; \Pr_s\Big(s\bigcup_{r\in\delta(q,s)}\mathcal{L}_r\Big)$$

$$= \sum_{t \in S} M_{s,t} \cdot \Pr_t\Big(\bigcup_{r \in \delta(q,s)} \mathcal{L}_r \Big)$$

Since $\mathcal{A}$ is unambiguous, the sets $\mathcal{L}_r$ are pairwise disjoint. Hence (2) follows. Equation (1) is shown similarly. □

Define the vector $\vec{z} \in [0,1]^{Q \times S}$ by $\vec{z}_{\langle qs \rangle} = \Pr_s(\mathcal{L}_q)$. By Lemma 4, to prove Theorem 3 it suffices to compute $\vec{z}$ in polynomial time. To this end, define a square matrix $B \in [0,1]^{(Q \times S) \times (Q \times S)}$ by

$$B_{\langle qs \rangle, \langle rt \rangle} = \begin{cases} M_{s,t} & \text{if } r \in \delta(q,s) \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

As $\mathcal{A}$ is unambiguous, B need not to be a stochastic matrix, whereas if $\mathcal{A}$ is deterministic, B is a stochastic matrix. By (2) we have $\vec{z} = B\vec{z}$, i.e., $\vec{z}$ solves the system of linear equations

$$\vec{\zeta} = B\vec{\zeta}\,, \tag{4}$$

where $\vec{\zeta}$ is a vector of variables indexed by $Q \times S$.

Example 5. Consider the UBA $\mathcal{A}$, shown above left in Fig. 2. This automaton is unambiguous since any two distinct states have disjoint languages:

$$\begin{aligned} \mathcal{L}_{q_0} &= \{a^{2m} w : w \in (b^+a^*)^\omega,\ m \geqslant 1\} \\ \mathcal{L}_{q_1} &= \{a^{2m-1} w : w \in (b^+a^*)^\omega,\ m \geqslant 1\} \\ \mathcal{L}_{q_2} &= \{w : w \in (b^+a^*)^\omega\} \end{aligned}$$

Consider moreover the two-state Markov chain $\mathcal{M}$, shown above right in Fig. 2. The weighted graph on the bottom of Fig. 2 represents the matrix B, obtained from $\mathcal{A}$ and $\mathcal{M}$ according to Equation (3). It is natural to think of B as a product of $\mathcal{A}$ and $\mathcal{M}$. Notice that B is not stochastic: the sum of the entries in each row (equivalently, the total outgoing transition weight of a graph node) may be strictly less than one and may be strictly greater than one.

Although (4) contains one equation for each $\langle qs \rangle \in Q \times S$, the vector $\vec{z}$ is not necessarily the unique solution of (4), e.g., any scalar multiple of $\vec{z}$ is also a solution. Below we identify suitable equations that, together with (4), have $\vec{z}$ as a unique solution. These extra equations are based on analysis of the strongly connected components (SCCs) of matrix B.

The following proposition shows that the powers of (submatrices of) B admit a probabilistic interpretation. Roughly, Proposition 6 states that the entries of the nth matrix power are probabilities of length-n paths in the Markov chain that trigger certain runs in the automaton.

Proposition 6. *Let $C \subseteq Q \times S$ and $\langle qs \rangle, \langle rt \rangle \in C$. Let $n \in \mathbb{N}$. Define $A := B_{C,C}$ and*

$$E^{C,n}_{\langle qs \rangle, \langle rt \rangle} := \{s_0 s_1 \cdots \in sS^\omega : \exists\, q_1 \cdots q_n \,.\, \langle q_0 s_0 \rangle \cdots \langle q_n s_n \rangle \in \mathit{Paths}_{\langle qs \rangle, \langle rt \rangle}(A)\}\,.$$

Then $(A^n)_{\langle qs \rangle, \langle rt \rangle} = \Pr_s\big(E^{C,n}_{\langle qs \rangle, \langle rt \rangle}\big)$. In particular:

$$(B^n)_{\langle qs \rangle, \langle rt \rangle} = \Pr_s(\{s_0 s_1 \cdots \in sS^\omega : r \in \delta(q, s_0 \cdots s_{n-1}),\ s_n = t\})$$

Proof. Let $n \in \mathbb{N}$ and $s_0 \cdots s_n \in \mathit{Paths}_{s,t}(M)$. Since the unambiguous automaton $\mathcal{A}$ is diamond-free, it has at most one run for $s_0 \cdots s_{n-1}$ from q to r. A sequence $q_0 \cdots q_n$ is such a run if and only if $\langle q_0 s_0 \rangle \cdots \langle q_n s_n \rangle \in \mathit{Paths}_{\langle qs \rangle, \langle rt \rangle}(B)$. Such a path is in $\mathit{Paths}_{\langle qs \rangle, \langle rt \rangle}(A)$ if and only if $\langle q_0 s_0 \rangle, \ldots, \langle q_n s_n \rangle \in C$. In that case we have:

$$\Pr_s(s_0 \cdots s_n S^\omega) = M_{s_0,s_1} \cdot \ldots \cdot M_{s_{n-1},s_n} = A_{\langle q_0 s_0 \rangle, \langle q_1 s_1 \rangle} \cdot \ldots \cdot A_{\langle q_{n-1} s_{n-1} \rangle, \langle q_n s_n \rangle}$$

We conclude that, for fixed $\langle qs \rangle, \langle rt \rangle$, the probability of those $s_0 \cdots s_n \in \mathit{Paths}_{s,t}(M)$ for which there are $q_0 \cdots q_n$ with $\langle q_0 s_0 \rangle \cdots \langle q_n s_n \rangle \in \mathit{Paths}_{\langle qs \rangle, \langle rt \rangle}(A)$ equals $(A^n)_{\langle qs \rangle, \langle rt \rangle}$. The proposition follows. □

4.2. Recurrent SCCs

In this section we define a notion of recurrent SCC for the matrix B, generalizing the familiar notion of the same name for finite Markov chains. We classify each recurrent SCC D as either accepting or non-accepting, showing that $\vec{z}_D$ is nonzero just in case D is accepting.

If automaton $\mathcal{A}$ is deterministic then the matrix B in (4) is stochastic. While B need not be stochastic if $\mathcal{A}$ is merely unambiguous, we have

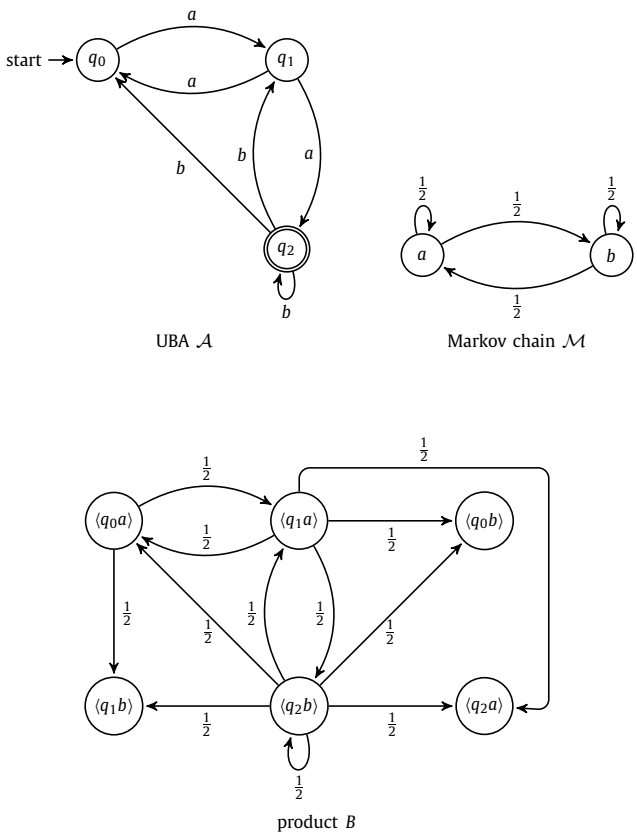

Fig. 2. A UBA $\mathcal{A}$, Markov chain $\mathcal{M}$, and their product B, as described in Example 5.

Proposition 7. $\rho(B) \leqslant 1$.

Proof. By Proposition 6, for all n, all entries of B^n are at most 1. Let $\vec{x} \neq \vec{0}$ be a dominant eigenvector, i.e., $B\vec{x} = \rho(B)\vec{x}$. Then $\rho(B)^n\vec{x} = B^n\vec{x}$ is bounded over all $n \geqslant 0$. It follows that $\rho(B) \leqslant 1$. □

Using Theorem 1(2), it follows from Proposition 7 that for any $D \subseteq Q \times S$ we have $\rho(B_{D,D}) \leqslant 1$. An SCC $D \subseteq Q \times S$ of B is called *recurrent* if $\rho(B_{D,D}) = 1$. Such an SCC D is said to be *accepting* if $\{q : \exists s \,.\, \langle qs\rangle \in D\} \in Acc$, i.e., if the collection of all automaton states in D is an accepting set.

The following lemma summarizes the key properties of recurrent SCCs that we will need.

Lemma 8. *Let D be a recurrent SCC.*

1. *We have $\vec{z}_D = B_{D,D}\vec{z}_D$.*
2. *For all $d \in D$, we have $\vec{z}_d > 0$ iff D is accepting.*

Lemma 8 is proved in Section 5.

Example 9. Consider the matrix B from Example 5, shown in Fig. 2. This matrix has a single recurrent SCC, namely $D = \{\langle q_0 a\rangle, \langle q_1 a\rangle, \langle q_2 b\rangle\}$. It is accepting, as the automaton uses a Büchi acceptance condition. We have:

$$B_{D,D} = \begin{pmatrix} 0 & 1/2 & 0 \\ 1/2 & 0 & 1/2 \\ 1/2 & 1/2 & 1/2 \end{pmatrix}$$

The vector $(1 \;\; 2 \;\; 3)^\top$ is a dominant eigenvector of $B_{D,D}$.

4.3. Cuts

In this subsection we introduce the notion of a cut of an SCC. Among other things, this yields a purely graph-theoretic characterization of recurrent SCCs: an SCC is recurrent just in case it has a cut.

Let $D \subseteq Q \times S$ be an SCC of B. A set $\alpha \subseteq D$ is called a *fiber* of D if it can be written $\alpha = \alpha' \times \{s\}$ for some $\alpha' \subseteq Q$ and $s \in S$. Given such a fiber α and $t \in S$, if $M_{s,t} > 0$ then we define a fiber

$$\alpha \triangleright t := \{\langle qt \rangle \in D : q \in \delta(\alpha', s)\}.$$

If $M_{s,t} = 0$ then $\alpha \triangleright t$ is undefined. We extend this definition inductively by $\alpha \triangleright \varepsilon := \alpha$ and $\alpha \triangleright wt := (\alpha \triangleright w) \triangleright t$ for $t \in S$ and $w \in S^*$. We have that $\alpha \triangleright w$ is defined if and only if sw describes a path in M. If α is a singleton $\{d\}$, we may write $d \triangleright w$ for $\alpha \triangleright w$.

We call a fiber $\alpha \subseteq D$ a *cut* of D if (i) $\alpha = d \triangleright v$ for some $d \in D$ and $v \in S^*$, and (ii) $\alpha \triangleright w \neq \emptyset$ holds for all $w \in S^*$ such that $\alpha \triangleright w$ is defined. Clearly if α is a cut then so is $\alpha \triangleright w$ when the latter is defined. Given a cut $\alpha \subseteq D$, we call its characteristic vector $\vec{\mu} \in \{0,1\}^D$ a *cut vector*.

The following lemma summarizes the key properties of cuts that we will need.

Lemma 10. *Let D be an SCC. Then*

1. *D is recurrent if and only if it has a cut.*
2. *If D is accepting recurrent and $\vec{\mu}$ is a cut vector then $\vec{\mu}^\top \vec{z}_D = 1$.*
3. *If D is recurrent one can compute a cut in polynomial time.*

Lemma 10 is proved in Section 6.

Given an accepting recurrent SCC D, say that a vector $\vec{\mu} \in [0,1]^D$ is a *D-normalizer* if $\vec{\mu}^\top \vec{z}_D = 1$. Then Lemma 10(2) says that a cut vector for D is a D-normalizer.

Example 11. Consider the matrix B from Example 5, shown in Fig. 2, and its single recurrent SCC $D = \{\langle q_0 a \rangle, \langle q_1 a \rangle, \langle q_2 b \rangle\}$. Then $\langle q_0 a \rangle \triangleright ab = \{\langle q_2 b \rangle\}$ is a cut and $\langle q_2 b \rangle \triangleright a = \{\langle q_0 a \rangle, \langle q_1 a \rangle\}$ is another cut. Both the associated cut vectors are D-normalizers. For example, let $\vec{\mu}$ be the cut vector associated with $\{\langle q_0 a \rangle, \langle q_1 a \rangle\}$, i.e., $\vec{\mu}^\top = (1 \quad 1 \quad 0)$. Since $\vec{z}_D = (1/3 \quad 2/3 \quad 1)^\top$, we have $\vec{\mu}^\top \vec{z}_D = 1$.

4.4. The augmented linear system

We now extend (4) to a linear system that has $\vec{z}$ as unique solution.

Lemma 12. *Let $\mathcal{D}_+$ be the set of accepting recurrent SCCs, and $\mathcal{D}_0$ the set of non-accepting recurrent SCCs. For each $D \in \mathcal{D}_+$ let $\vec{\mu}_D \in [0,1]^D$ be a D-normalizer (which exists by Lemma 10(2)). Then $\vec{z}$ is the unique solution of the following linear system:*

$$\begin{aligned} \vec{\zeta} &= B\vec{\zeta} \\ \text{for all } D \in \mathcal{D}_+ : \quad \vec{\mu}_D^\top \vec{\zeta}_D &= 1 \\ \text{for all } D \in \mathcal{D}_0 : \quad \vec{\zeta}_D &= \vec{0} \end{aligned} \tag{5}$$

Proof. The vector $\vec{z}$ solves (5): indeed, this follows from the equality $\vec{z} = B\vec{z}$, the definition of a D-normalizer, and Lemma 8(2).

It remains to show uniqueness. To this end, let $\vec{x}$ solve (5). We show that $\vec{x} = \vec{z}$. We proceed by induction over the DAG of SCCs of B. Let $D \subseteq Q \times S$ be any SCC (possibly trivial, i.e., $D = \{d\}$ for some $d \in Q \times S$, and $B_{d,d} = 0$). Let us write $D{\downarrow}$ for the set of SCCs directly below D. By the induction hypothesis, we have $\vec{x}_C = \vec{z}_C$ for all SCCs $C \in D{\downarrow}$. We have to show that $\vec{x}_D = \vec{z}_D$. Since $\vec{x} = B\vec{x}$ and $\vec{z} = B\vec{z}$, we have

$$\begin{aligned} \vec{x}_D - \vec{z}_D \;&=\; B_{D,Q\times S}(\vec{x} - \vec{z}) \;=\; B_{D,D}(\vec{x}_D - \vec{z}_D) + \sum_{C \in D{\downarrow}} B_{D,C}(\vec{x}_C - \vec{z}_C) \\ &=\; B_{D,D}(\vec{x}_D - \vec{z}_D) \quad \text{by the induction hypothesis.} \end{aligned} \tag{6}$$

- Let D be non-recurrent. Then we must have $\vec{x}_D = \vec{z}_D$, as otherwise, by (6), the vector $\vec{x}_D - \vec{z}_D$ would be an eigenvector of $B_{D,D}$ associated with eigenvalue 1, implying $\rho(B_{D,D}) \geq 1$, and thus contradicting the assumption that D is not recurrent.
- Let D be recurrent. If $D \in \mathcal{D}_0$, then $\vec{x}_D = \vec{0} = \vec{z}_D$. Therefore, we can assume that $D \in \mathcal{D}_+$. By Lemma 8(1), $\vec{z}_D = B_{D,D}\vec{z}_D$. Thus, with (6), $\vec{x}_D = B_{D,D}\vec{x}_D$. By Theorem 2(2), the eigenspace of $B_{D,D}$ associated with the spectral radius is one-dimensional, implying that $\vec{x}_D$ is a scalar multiple of $\vec{z}_D$. We have $\vec{\mu}_D^\top \vec{x}_D = 1 = \vec{\mu}_D^\top \vec{z}_D$, hence $\vec{x}_D = \vec{z}_D$. □

Now we can prove our main result, Theorem 3.

Proof (of Theorem 3). Given a Markov chain $\mathcal{M}$, an initial distribution ι, and a diamond-free unambiguous automaton $\mathcal{A}$, proceed as follows.

1. Set up the matrix B from Section 4.1.
2. Compute the SCCs of B.
3. For any SCC C, check whether C is recurrent, by seeing if the linear system $B_{C,C}\vec{x} = \vec{x}$ has a nonzero solution.
4. For any accepting recurrent SCC D, compute a cut vector $\vec{\mu}$ using Lemma 10(3).
5. Solve the linear system (5) in Lemma 12.
6. Compute $\Pr_\iota(\mathcal{L}(\mathcal{A}))$ using (1) in Lemma 4. □

5. Proof of Lemma 8

In this section we prove Lemma 8, which is restated here.

Lemma 8. *Let D be a recurrent SCC.*

1. *We have $\vec{z}_D = B_{D,D}\vec{z}_D$.*
2. *For all $d \in D$, we have $\vec{z}_d > 0$ iff D is accepting.*

5.1. Proof of Lemma 8(1)

Proof of Lemma 8(1) Recall that $\vec{z} = B\vec{z}$. Thus, $\vec{z}_D = B_{D,Q\times S}\vec{z} \geqslant B_{D,D}\vec{z}_D$. Since $\rho(B_{D,D}) = 1$ and D is strongly connected, it follows, by Theorem 2(4), that $\vec{z}_D = B_{D,D}\vec{z}_D$. □

5.2. Proof of Lemma 8(2)

The proof of Lemma 8(2) requires some auxiliary definitions and results.

Let $C, D \subseteq Q \times S$ be two SCCs of matrix B. We write $C \preceq D$ if C is reachable from D. Note that $\preceq$ is a partial order on the SCCs. In case matrix B is stochastic the recurrent SCCs are just the bottom SCCs. While this does not hold in general, we have the following result.

Proposition 13. *Let $D \subseteq Q \times S$ be a recurrent SCC. Then all SCCs $C \prec D$ are such that $\vec{z}_C = \vec{0}$.*

Proof. Recall that $B\vec{z} = \vec{z}$. Thus, $B_{D,Q\times S}\vec{z} = \vec{z}_D = B_{D,D}\vec{z}_D$ by Lemma 8(1). Hence, we have for any $c \in (Q \times S) \setminus D$ that $B_{D,c}\vec{z}_c = \vec{0}$. So if $B_{D,c} \neq \vec{0}$ then $\vec{z}_c = 0$. It follows that $\vec{z}_C = \vec{0}$ for all SCCs C directly below D. Using the definitions of $\vec{z}$ and B it follows that $\vec{z}_C = \vec{0}$ must hold for all SCCs $C \prec D$. □

In the following two propositions we consider paths in M along with "corresponding" paths in B. The gist of Propositions 14 and 15 is that paths in M that have "corresponding" paths in B that linger in a strict subset of an SCC are negligible.

Let $\pi_Q : Q \times S \to Q$ and $\pi_S : Q \times S \to S$ denote the obvious projection maps. We extend both maps pointwise to finite and infinite sequences; e.g., we have $\pi_Q : (Q \times S)^* \to Q^*$.

Proposition 14. *Let D be a recurrent SCC, let $C \subseteq D$ be strongly connected, and let $c = \langle qs\rangle \in C$. Define $A := B_{C,C}$ and write*

$$E_c^C := \{w \in Paths_s^\omega(M) : \exists\, v \in Paths_c^\omega(A) \text{ s.t. } \pi_S(v) = w\}.$$

Then $\Pr_s(E_c^C) > 0$ iff $C = D$.

Proof. Let $\vec{x} \geqslant \vec{0}$ be a dominant eigenvector of A and write $x_{\min}$ and $x_{\max}$ for the respective minimum and maximum entries of $\vec{x}$. Since C is strongly connected, by Theorem 2(1) and (2), we have $x_{\min} > 0$.

Intuitively, E_c^C contains those paths in M that correspond to a path in A. For $n \in \mathbb{N}$ and $d \in C$, recall the notation $E_{c,d}^{C,n}$ from Proposition 6. Write also $E_c^{C,n} := \bigcup_{d\in C} E_{c,d}^{C,n}$. Intuitively, $E_c^{C,n}$ contains those paths in M that correspond to a path in A for at least n steps. Then $E_c^C = \bigcap_{n\in\mathbb{N}} E_c^{C,n}$ by Kőnig's Lemma. We have for any $d \in C$:

$$(A^n)_{c,d} \overset{\text{Prop. 6}}{=} \Pr_s(E_{c,d}^{C,n}) \leqslant \Pr_s(E_c^{C,n}) \leqslant \sum_{d'\in C} \Pr_s(E_{c,d'}^{C,n}) = \sum_{d'\in C} (A^n)_{c,d'} \tag{7}$$

Suppose now that $C = D$. Then $\rho(A) = 1$ and thus

$$\begin{aligned} |C| \cdot \Pr_s(E_c^{C,n}) &\geqslant \sum_{d \in C} (A^n)_{c,d} && \text{by Equation (7)} \\ &= (A^n \vec{1})_c \\ &\geqslant \tfrac{1}{x_{\max}} (A^n \vec{x})_c && \text{since } \vec{x} \leqslant x_{\max} \cdot \vec{1} \\ &= \tfrac{\vec{x}_c}{x_{\max}} && \text{since } A\vec{x} = \vec{x} \end{aligned}$$

By continuity of measures it follows that $\Pr_s(E_c^C) > 0$.

Conversely, suppose that $C \neq D$. Then

$$\begin{aligned} \Pr_s(E_c^{C,n}) &\leqslant \sum_{d \in C} (A^n)_{c,d} && \text{by Equation (7)} \\ &= (A^n \vec{1})_c \\ &\leqslant \tfrac{1}{x_{\min}} (A^n \vec{x})_c && \text{since } x_{\min} \cdot \vec{1} \leqslant \vec{x} \\ &= \rho(A)^n \tfrac{\vec{x}_c}{x_{\min}} && \text{since } A\vec{x} = \rho(A)\vec{x} \end{aligned}$$

But $\rho(A) < 1$ by Theorem 2(3). By continuity of measures it follows that $\Pr_s(E_c^C) = 0$. □

In the following proposition, $E^{<D}$ includes those infinite paths in M that correspond to paths in B that eventually linger in $B_{C,C}$ for a strict subset C of an SCC D in B. By Proposition 14 such paths have measure zero.

Proposition 15. *Let D be a recurrent SCC. Write*

$$E^{<D} := \bigcup_{u \in S^*} \bigcup_{C \subsetneq D} \bigcup_{d \in C} u E_d^C .$$

Then $\Pr_s(E^{<D}) = 0$ for all $s \in S$.

Proof. Let $s \in S$, and $u \in S^*$, and $C \subsetneq D$ be strongly connected, and $\langle rt \rangle = d \in C$. Then $\Pr_s(u E_d^C) \leqslant \Pr_t(E_d^C) = 0$ by Proposition 14. Thus $E^{<D}$ is a countable union of sets of measure zero with respect to $\Pr_s$. It follows $\Pr_s(E^{<D}) = 0$. □

Proof of Lemma 8(2) Let $d = \langle qs \rangle \in D$. We show that $\vec{z}_d > 0$ iff D is accepting.

Suppose that D is accepting. Recall the definitions of E_d^C and $E^{<D}$ in Propositions 14 and 15. Since D is accepting, we have $E_d^D \setminus E^{<D} \subseteq \mathcal{L}_q$. Hence:

$$\begin{aligned} \vec{z}_d &= \Pr_s(\mathcal{L}_q) \\ &\geqslant \Pr_s(E_d^D \setminus E^{<D}) \\ &\geqslant \Pr_s(E_d^D) - \Pr_s(E^{<D}) \\ &> 0 && \text{by Propositions 14 and 15} \end{aligned}$$

Suppose now that D is not accepting. Let $v \in Paths_d^\omega(B)$ be such that $\inf(\pi_Q(v)) \in Acc$. Since D is not accepting, v must eventually either remain in a strongly connected set $C \subsetneq D$ or reach an SCC $C \prec D$. Thus we have

$$Paths_s^\omega(M) \cap \mathcal{L}_q \subseteq E^{<D} \cup \bigcup_{u \in S^*} \bigcup_{C \prec D} \bigcup_{\langle rt \rangle \in C} u(Paths_t^\omega(M) \cap \mathcal{L}_r) . \tag{8}$$

We have $\Pr_s(E^{<D}) = 0$ by Proposition 15. For any SCC $C \prec D$, $\langle rt \rangle \in C$, and $u \in S^*$, we have

$$\Pr_s(u(Paths_t^\omega(M) \cap \mathcal{L}_r)) \leqslant \Pr_t(Paths_t^\omega(M) \cap \mathcal{L}_r) = \vec{z}_{\langle rt \rangle} = 0 ,$$

with the last equality following from Proposition 13. Thus the right-hand side of (8) is a countable union of sets of measure zero with respect to $\Pr_s$. It follows that $\vec{z}_{\langle qs \rangle} = \Pr_s(Paths_s^\omega(M) \cap \mathcal{L}_q) = 0$. □

6. Proof of Lemma 10

In this section we prove Lemma 10, which is restated here.

Lemma 10. *Let D be an SCC. Then*

1. *D is recurrent if and only if it has a cut.*
2. *If D is accepting recurrent and $\vec{\mu}$ is a cut vector then $\vec{\mu}^\top \vec{z}_D = 1$.*
3. *If D is recurrent one can compute a cut in polynomial time.*

6.1. Proof of Lemma 10(1)

Proof of Lemma 10(1) Let $D \subseteq Q \times S$ be an SCC. We have for all $\langle qs\rangle, \langle rt\rangle \in D$:

$$\Pr_s\big(E^{D,n}_{\langle qs\rangle,\langle rt\rangle}\big) \;=\; \Pr_s(\{s_0 s_1 s_2 \cdots \in \mathit{Paths}^\omega_s(M) : \langle rt\rangle \in \langle qs\rangle \triangleright s_1 \cdots s_n\}) \tag{9}$$

Define $A := B_{D,D}$. By Theorem 2(1), A has a dominant eigenvector $\vec{x}$, positive in all entries, with $A\vec{x} = \rho(A)\vec{x}$. Write $x_{\min} > 0$ for the smallest entry of $\vec{x}$. For any $d_0 = \langle q_0 s_0\rangle \in D$ and $n \in \mathbb{N}$ define a random variable $X^{d_0}_n : \mathit{Paths}^\omega_{s_0}(M) \to \mathbb{R}_{\geqslant 0}$ by

$$X^{d_0}_n(s_0 s_1 \cdots) \;=\; \sum_{d \in d_0 \triangleright s_1 \cdots s_n} \vec{x}_d\,.$$

We have:

$$\mathrm{Ex}_{s_0}(X^{d_0}_n) \overset{(9)}{=} \sum_{d\in D} \Pr_{s_0}\big(E^{D,n}_{d_0,d}\big)\cdot \vec{x}_d \overset{\text{Prop. 6}}{=} (A^n\vec{x})_{d_0} \;=\; \rho(A)^n \vec{x}_{d_0}\,, \tag{10}$$

where Ex_{s_0} denotes expectation with respect to $\Pr_{s_0}$.

Towards the direction "$\Longleftarrow$", let $d_0 \triangleright s_1 \cdots s_k$ be a cut, where $d_0 = \langle q_0 s_0\rangle$. Then for all $w \in \mathit{Paths}_{s_k}(M)$ we have $d_0 \triangleright s_1 \cdots s_k w \neq \emptyset$. So we have for all $n \geqslant k$, using Markov's inequality:

$$\begin{aligned} 0 \;<\; \Pr_{s_0}(s_0 \cdots s_k S^\omega)\cdot x_{\min} \;&\leqslant\; \Pr_{s_0}(X^{d_0}_n \geqslant x_{\min})\cdot x_{\min} \\ &\leqslant\; \mathrm{Ex}_{s_0}(X^{d_0}_n) \overset{(10)}{=} \rho(A)^n \vec{x}_{d_0} \end{aligned}$$

This gives a uniform lower bound on $\rho(A)^n$ for all $n \geqslant k$. Hence $\rho(A) \geqslant 1$, and so D is recurrent.

Towards the converse "$\Longrightarrow$", suppose that D has no cuts. Let $d_0 = \langle q_0 s_0\rangle \in D$. Since D has no cuts, for any set $d_0 \triangleright v \subseteq D$ there exists $w \in S^*$ with $d_0 \triangleright v w = \emptyset$. It follows that there are $\ell \in \mathbb{N}$ and $y > 0$ such that for all $s_0 \cdots s_n \in \mathit{Paths}_{s_0}(M)$:

$$\Pr_{s_0}(0 = X^{d_0}_{n+\ell} = X^{d_0}_{n+\ell+1} = \ldots \mid s_0 \cdots s_n S^\omega) \;\geqslant\; y$$

Thus, for all $m \geqslant 0$:

$$\Pr_{s_0}(0 = X^{d_0}_{\ell m} = X^{d_0}_{\ell m+1} = \ldots) \;\geqslant\; 1 - (1-y)^m$$

Hence, $X^{d_0}_0, X^{d_0}_1, \ldots$ converges to 0 almost surely with respect to $\Pr_{s_0}$. Since $X^{d_0}_0, X^{d_0}_1, \ldots$ is bounded (by $\sum_{d\in D} \vec{x}_d$), it follows $\lim_{n\to\infty} \mathrm{Ex}_{s_0}(X^{d_0}_n) = 0$. By (10), we conclude $\rho(A) < 1$, i.e., D is not recurrent. □

6.2. Proof of Lemma 10(2)

For the proof of Lemma 10(2) we use the following standard fact about model checking Markov chains:

Lemma 16. *Let $\mathcal{M} = (S, M)$ be a Markov chain, and $\mathcal{L} \subseteq S^\omega$ an ω-regular language. Suppose $s_0 \in S$ such that $\Pr_{s_0}(\mathcal{L}) < 1$. Then there exists $s_0 \cdots s_n \in \mathit{Paths}_{s_0}(M)$ such that $\Pr_{s_n}(\{w \in s_n S^\omega : s_0 \cdots s_{n-1} w \in \mathcal{L}\}) = 0$.*

Proof (Sketch). Let $\mathcal{A}$ be a deterministic Muller automaton that accepts $\mathcal{L}$. Let q_0 be the initial state of $\mathcal{A}$. Since $\Pr_{s_0}(\mathcal{L}) < 1$, there is a path $\langle q_0 s_0\rangle \cdots \langle q_n s_n\rangle$ in the product of $\mathcal{A}$ and $\mathcal{M}$ that leads to a bottom SCC that is non-accepting, i.e., whose corresponding set of automaton states does not satisfy the Muller acceptance condition. Then $s_0 \cdots s_n$ has the required properties. □

Proof of Lemma 10(2) Let $D \ni \langle q_0 s_0 \rangle$ be a recurrent SCC, and let $\alpha = \beta \times \{s_n\} = \langle q_0 s_0 \rangle \triangleright s_1 \cdots s_n \subseteq D$ be a cut. Recall that $\beta \subseteq \delta(q_0, s_0 \cdots s_{n-1})$ and define $\beta' := \delta(q_0, s_0 \cdots s_{n-1}) \setminus \beta$. Since all elements of $\delta(q_0, s_0 \cdots s_{n-1}) \times \{s_n\}$ are reachable from $\langle q_0 s_0 \rangle$ in B, by Proposition 13 we have $\vec{z}_{\beta' \times \{s_n\}} = \vec{0}$, thus $\sum_{q \in \beta'} \Pr_{s_n}(\mathcal{L}_q) = 0$. Let $\vec{\mu} \in \{0,1\}^D$ be the cut vector associated with α. Then we have:

$$\begin{aligned} \vec{\mu}^\top \vec{z}_D &= \sum_{d \in \alpha} \vec{z}_d = \sum_{q \in \beta} \Pr_{s_n}(\mathcal{L}_q) = \sum_{q \in \beta} \Pr_{s_n}(\mathcal{L}_q) + \sum_{q \in \beta'} \Pr_{s_n}(\mathcal{L}_q) \\ &= \sum_{q \in \delta(q_0, s_0 \cdots s_{n-1})} \Pr_{s_n}(\mathcal{L}_q) = \Pr_{s_n}(\mathcal{L}(\mathcal{A}[\delta(q_0, s_0 \cdots s_{n-1})]))\,, \end{aligned}$$

where the last equality holds as the sets $\mathcal{L}_q$ are disjoint by unambiguousness. Hence $\vec{\mu}^\top \vec{z}_D \leqslant 1$. Suppose $\vec{\mu}^\top \vec{z}_D < 1$. Then $\Pr_{s_n}(\mathcal{L}(\mathcal{A}[\delta(q_0, s_0 \cdots s_{n-1})])) < 1$. Then by Lemma 16 there exists $s_n \cdots s_m \in \mathit{Paths}_{s_n}(M)$ such that

$$\Pr_{s_m}(\{w \in s_m S^\omega : s_n \cdots s_{m-1} w \in \mathcal{L}(\mathcal{A}[\delta(q_0, s_0 \cdots s_{n-1})])\}) = 0\,.$$

Equivalently,

$$\Pr_{s_m}(\{w \in s_m S^\omega : w \in \mathcal{L}(\mathcal{A}[\delta(q_0, s_0 \cdots s_{m-1})])\}) = 0\,. \tag{11}$$

But since α is a cut, we have $\langle q_0 s_0 \rangle \triangleright s_1 \cdots s_m \neq \emptyset$, i.e., there exists $q \in \delta(q_0, s_0 \cdots s_{m-1})$ with $\langle q s_m \rangle \in D$. By (11) we have $\Pr_{s_m}(\mathcal{L}_q) = 0$. With Lemma 8(2) it follows that D is not accepting. □

6.3. Proof of Lemma 10(3)

Let $D \subseteq Q \times S$ be a recurrent SCC. In this section we show how to compute a cut in polynomial time. By Lemma 10(2), this also yields a D-normalizer if D is accepting.

Since automaton $\mathcal{A}$ is diamond-free, we have that if $d \triangleright v \supseteq \{d_1, d_2\}$ with $d_1 \neq d_2$ then any sets $d_1 \triangleright w$ and $d_2 \triangleright w$ are disjoint.

Lemma 17. *Let $D \subseteq Q \times S$ be a recurrent SCC. Let $d \in D$. Suppose $w \in S^*$ is such that $d \triangleright w \ni d$ is not a cut. Then there are $v \in S^*$ and $d' \neq d$ with $d \triangleright v \supseteq \{d, d'\}$ and $d' \triangleright w \neq \emptyset$. Hence $d \triangleright vw \supseteq \{d, d'\} \triangleright w = d \triangleright w \cup d' \triangleright w \supsetneq d \triangleright w$.*

Proof. Since D is recurrent, by Lemma 10(1), D has a cut α, say $\alpha = d_1 \triangleright v_1$. Since D is an SCC, there is v_0 with $d \triangleright v_0 \ni d_1$, hence $d \triangleright v_0 v_1 \supseteq \alpha$ is also a cut. Again, since D is an SCC, there is v_2 with $d \triangleright v_0 v_1 v_2 \ni d$. Define $v := v_0 v_1 v_2$. Then $d \triangleright v \ni d$ is a cut. Moreover, we have $d \triangleright vw \supseteq d \triangleright w$. Since $d \triangleright vw$ is a cut but $d \triangleright w$ is not, we have:

$$d \triangleright w \cup (d \triangleright v \setminus \{d\}) \triangleright w = d \triangleright vw \supsetneq d \triangleright w$$

So there is $d' \in d \triangleright v \setminus \{d\}$ with $d' \triangleright w \neq \emptyset$. □

Lemma 18. *Let $D \subseteq Q \times S$ be a recurrent SCC. Let $d = \langle q_0 s_0 \rangle \in D$. The following algorithm is a polynomial-time algorithm that computes $w \in S^*$ with $|w| \leqslant |Q|^3 |S|$ such that $d \triangleright w$ is a cut of D:*

1. *$w := \varepsilon$ (the empty word)*
2. *while $\exists\, v \in S^*$ and $\exists\, d' \neq d$ such that $d \triangleright v \supseteq \{d, d'\}$ and $d' \triangleright w \neq \emptyset$:*
   $w := vw$
3. *return $d \triangleright w$*

Proof. By Lemma 17 the algorithm returns a cut. In every iteration, the set $d \triangleright w$ increases (cf. Lemma 17), so the algorithm terminates after at most $|Q|$ iterations.

Consider the directed graph $G = (V, E)$ with

$$\begin{aligned} V &= \{(q, q', s) \in Q \times Q \times S : \langle qs \rangle, \langle q's \rangle \in D\} \\ E &= \{(q, q', s) \to (r, r', t) : \delta(q, s) \ni r,\ \delta(q', s) \ni r',\ M_{s,t} > 0\}\,. \end{aligned}$$

Then for any $v = s_1 \cdots s_n \in S^*$ and $d_n = \langle q_n s_n \rangle, d'_n = \langle q'_n s_n \rangle \in D$ we have $d \triangleright v \supseteq \{d_n, d'_n\}$ if and only if there are $q_1, q'_1, \ldots, q_{n-1}, q'_{n-1}$ such that

$$(q_0, q_0, s_0) \to (q_1, q'_1, s_1) \to \cdots \to (q_n, q'_n, s_n)$$

is a path in G. It follows that with a (polynomial-time) reachability analysis of G one can compute all $d' \in D$ for which there exists $v_{d'} \in S^*$ with $d \triangleright v_{d'} \supseteq \{d, d'\}$. The shortest such $v_{d'}$ correspond to shortest paths in G, hence satisfy $|v_{d'}| \leqslant |V| \leqslant |Q|^2 |S|$. Moreover, one can check in polynomial time whether $d' \triangleright w \neq \emptyset$. □

7. An NC model checking procedure

In this section we show that one can model check Markov chains against unambiguous automata in NC. To achieve this we strengthen our assumptions on the acceptance condition: we assume that the given automaton $\mathcal{A} = (Q, \Sigma, \delta, Q_0, Acc)$ has an NC-decidable acceptance condition; i.e., given $\mathcal{A}$ and a set $R \subseteq Q$, one can compute in NC whether $R \in Acc$. This is the case, e.g., if Acc is given as a Büchi condition. We show:

Theorem 19. *Given a Markov chain $\mathcal{M}$, an initial distribution ι, and an unambiguous automaton $\mathcal{A}$ with NC-decidable acceptance condition, the value $\Pr_\iota(\mathcal{L}(\mathcal{A}))$ is computable in NC.*

As a consequence, our approach yields optimal complexity for model checking Markov chains against LTL specifications:

Corollary 20. *Given a Markov chain $\mathcal{M}$, an initial distribution ι, and an LTL formula φ, the value $\Pr_\iota(\mathcal{L}(\varphi))$ is computable in PSPACE.*

Proof. There is a classical polynomial-space procedure that translates φ into an (exponential-sized) Büchi automaton $\mathcal{A}_\varphi$ [47]. As noted by several authors (e.g., [13,18]), this procedure can easily be adapted to ensure that $\mathcal{A}_\varphi$ be a UBA.

Now recall from Section 2 that a function that is computable in NC is also computable in poly-logarithmic space. By Theorem 19 it follows that we can compute $\Pr_\iota(\mathcal{L}(\varphi))$ in poly-logarithmic space in $\mathcal{A}_\varphi$ and $\mathcal{M}$. Thus using standard techniques for composing space-bounded transducers (see, e.g., [41, Proposition 8.2]), we can compute $\Pr_\iota(\mathcal{L}(\varphi))$ using polynomial space in φ and $\mathcal{M}$. □

Towards a proof of Theorem 19, observe that most steps of the algorithm from the proof of Theorem 3 can be implemented in NC in a straightforward way. The exception is the cut-computation algorithm from Lemma 18, which seems inherently sequential. Recall that we used this algorithm because a cut vector of an accepting recurrent SCC D yields a D-normalizer (Lemma 10(2)) and we need such a normalizer to set up the equation system (5) from Lemma 12. Note that any convex combination of normalizers is also a normalizer. In the following we show how to compute such a normalizer in NC.

Let $D \subseteq Q \times S$ be a recurrent SCC. Let $d_0 = \langle q_0 s_0 \rangle \in D$. Define $E := \{d \in D : \exists\, v \in S^* \,.\, d_0 \triangleright v \supseteq \{d_0, d\}\}$. Observe that E is fibered on s_0, i.e., there is $R \subseteq Q$ with $E = R \times \{s_0\}$. For any $q \in Q$ and $w \in S^*$ and $\alpha \subseteq Q$ with $\langle q s_0 \rangle \triangleright w = \alpha \times \{s_0\}$ we write $q \triangleright w = \alpha$ to avoid clutter. Hence

$$R \;=\; \{q \in Q \;:\; \exists\, v \in S^* . \, q_0 \triangleright v \supseteq \{q_0, q\}\}\,.$$

One can compute R in NC by a graph reachability analysis. Similarly, one can compute in NC for any $q \in R$ a word $v_q \in S^*$ such that $q_0 \triangleright v_q \supseteq \{q_0, q\}$. One can also compute in NC for any $q \in R$ a matrix $A(q) \in \{0,1\}^{R \times R}$ such that $A(q)_{r,r'} = 1$ if and only if $r \triangleright v_q \ni r'$. Define

$$A \;:=\; \frac{1}{|R|} \sum_{q \in R} A(q)\,.$$

In the following, for a set $\alpha \subseteq R$, we write $\vec{\chi}(\alpha) \in \{0,1\}^R$ for the characteristic vector of α. If α is a singleton set $\{q\}$ we may write $\vec{\chi}(q)$ for $\vec{\chi}(\alpha)$. The following lemma provides a D-normalizer:

Lemma 21. *Let $D \subseteq Q \times S$ be an accepting recurrent SCC. Let $d_0 = \langle q_0 s_0 \rangle \in D$. Define $R \subseteq Q$ and $A \in [0,1]^{R \times R}$ as above. Then the limit*

$$\vec{\eta}^\top \;:=\; \lim_{n \to \infty} \vec{\chi}(q_0)^\top A^n \;\in\; [0,1]^R$$

exists. The vector $\vec{\mu} \in [0,1]^D$ with

$$\vec{\mu}_{\langle qs \rangle} = \begin{cases} \vec{\eta}_q & \text{if } q \in R \text{ and } s = s_0 \\ 0 & \text{otherwise} \end{cases}$$

is a D-normalizer.

Proof. Observe that $A(q)_{q_0,q_0} = A(q)_{q_0,q} = 1$ for all $q \in R$. For any $q_1, \ldots, q_n \in R$ we have $q_0 \triangleright v_{q_1} \cdots v_{q_n} \supseteq \{q_0, q_n\}$. Since the automaton $\mathcal{A}$ is diamond-free, we have more generally:

$$\vec{\chi}(q_0)^\top A(q_1) \cdot \ldots \cdot A(q_n) \;=\; \vec{\chi}(q_0 \triangleright v_{q_1} \cdots v_{q_n})^\top \tag{12}$$

It follows that, for all n, the entries of $\vec{\chi}(q_0)^\top A^n$ are at most 1, and the q_0-entry equals 1. Since A is nonnegative, we obtain:

$$\vec{\chi}(q_0)^\top \leqslant \vec{\chi}(q_0)^\top A \leqslant \vec{\chi}(q_0)^\top A^2 \leqslant \cdots$$

So the limit $\vec{\eta}^\top \in [0,1]^R$ exists.

Define $\mathit{Cuts} := \{\alpha \subseteq R : \alpha \times \{s_0\} \text{ is a cut}\}$ and $V := \{v_q \in S^* : q \in R\}$. It follows from the definition of the v_q that we have $q_0 \triangleright v \ni q_0$ for all $v \in V^*$. By Lemma 17 there are $\bar{q}_1, \ldots, \bar{q}_k \in R$ such that

$$q_0 = q_0 \triangleright \varepsilon \subsetneq q_0 \triangleright v_{\bar{q}_1} \subsetneq q_0 \triangleright v_{\bar{q}_2} v_{\bar{q}_1} \subsetneq \ldots \subsetneq q_0 \triangleright v_{\bar{q}_k} \cdots v_{\bar{q}_2} v_{\bar{q}_1}$$

and $q_0 \triangleright \bar{v} \in \mathit{Cuts}$, where $\bar{v} = v_{\bar{q}_k} \cdots v_{\bar{q}_1}$. Thus, if $w \in V^* \cdot \{\bar{v}\} \cdot V^*$ then $q_0 \triangleright w \in \mathit{Cuts}$.

Define a function $\vec{\nu} : [0,1]^R \to [0,1]^D$ with

$$\vec{\nu}(\vec{x})_{\langle qs \rangle} = \begin{cases} \vec{x}_q & \text{if } q \in R \text{ and } s = s_0 \\ 0 & \text{otherwise.} \end{cases}$$

By Lemma 10(2) for all $\alpha \in \mathit{Cuts}$ the vector $\vec{\nu}(\vec{\chi}(\alpha))$ is a D-normalizer, i.e., $\vec{\nu}(\vec{\chi}(\alpha))^\top \vec{z}_D = 1$. Defining $f : [0,1]^R \to [0, D]$ with $f(\vec{x}) = \vec{\nu}(\vec{x})^\top \vec{z}_D$, we have $f(\vec{\chi}(\alpha)) = 1$ for all $\alpha \in \mathit{Cuts}$. Note that f is a linear function.

Consider the stochastic process $r_1, r_2, \ldots$ where the r_i are chosen from R independently and uniformly at random. Write Pr and Ex for the associated probability measure and expectation. It follows from (12) that we have

$$\vec{\chi}(q_0)^\top A^n = \mathrm{Ex}\left(\vec{\chi}(q_0 \triangleright v_{r_1} \cdots v_{r_n})^\top\right) \quad \text{for all } n \in \mathbb{N}\,. \tag{13}$$

In the following, when we say 'almost surely' we mean with probability 1 with respect to Pr. Almost surely, $\bar{q}_k, \ldots, \bar{q}_1$ will eventually appear as a contiguous subsequence of $r_1, r_2, \ldots$. That is, $v_{r_1} v_{r_2} \cdots \in V^* \cdot \{\bar{v}\} \cdot V^\omega$ almost surely. Thus, almost surely there is $m \in \mathbb{N}$ such that $q_0 \triangleright v_{r_1} \cdots v_{r_n} \in \mathit{Cuts}$ holds for all $n \geqslant m$. Hence,

$$\Pr\left(\lim_{n\to\infty} f\left(\vec{\chi}(q_0 \triangleright v_{r_1} \cdots v_{r_n})\right) = 1\right) = 1\,. \tag{14}$$

So we have

$$\begin{aligned}
\vec{\mu}^\top \vec{z}_D &= \vec{\nu}(\vec{\eta})^\top \vec{z}_D && \text{definitions of } \vec{\mu}, \vec{\nu} \\
&= f(\vec{\eta}) && \text{definition of } f \\
&= f\left(\left(\lim_{n\to\infty} \vec{\chi}(q_0)^\top A^n\right)^\top\right) && \text{definition of } \vec{\eta} \\
&= f\left(\lim_{n\to\infty} \mathrm{Ex}\left(\vec{\chi}(q_0 \triangleright v_{r_1} \cdots v_{r_n})\right)\right) && \text{by (13)} \\
&= \lim_{n\to\infty} f\left(\mathrm{Ex}\left(\vec{\chi}(q_0 \triangleright v_{r_1} \cdots v_{r_n})\right)\right) && f \text{ is continuous} \\
&= \lim_{n\to\infty} \mathrm{Ex}\left(f\left(\vec{\chi}(q_0 \triangleright v_{r_1} \cdots v_{r_n})\right)\right) && f \text{ is linear} \\
&= \mathrm{Ex} \lim_{n\to\infty} \left(f\left(\vec{\chi}(q_0 \triangleright v_{r_1} \cdots v_{r_n})\right)\right) && \text{dominated convergence theorem} \\
&= 1 && \text{by (14)};
\end{aligned}$$

i.e., $\vec{\mu}$ is a D-normalizer. □

Lemma 22. *The vector $\vec{\eta}$ with $\vec{\eta}^\top = \lim_{n\to\infty} \vec{\chi}(q_0)^\top A^n$ from Lemma 21 is the unique solution of the linear system*

$$\vec{\xi}^\top = \vec{\xi}^\top (A - E) + \vec{\chi}(q_0)^\top \tag{15}$$

where $\vec{\xi}$ is a vector of variables indexed by R, and $E \in \{0,1\}^{R \times R}$ is the matrix with $E_{q,r} = 1$ if and only if $q = r = q_0$.

Proof. Recall from the proof of Lemma 21 that we have $\left(\vec{\chi}(q_0)^\top A^n\right)_{q_0} = 1$ for all n. It follows that $\vec{\eta}_{q_0} = 1$ and thus $\vec{\eta}^\top E = \vec{\chi}(q_0)^\top$. Hence we have:

$$\vec{\eta}^\top = \vec{\eta}^\top A = \vec{\eta}^\top (A - E) + \vec{\eta}^\top E = \vec{\eta}^\top (A - E) + \vec{\chi}(q_0)^\top$$

So $\vec{\eta}$ solves (15). Towards uniqueness, since

$$\begin{aligned}\vec{\eta}^\top &= \vec{\eta}^\top(A-E) + \vec{\chi}(q_0)^\top \\ &= \left(\vec{\eta}^\top(A-E) + \vec{\chi}(q_0)^\top\right)(A-E) + \vec{\chi}(q_0)^\top,\end{aligned}$$

we have:

$$\begin{aligned}\vec{\eta}^\top(A-E)^2 &= \vec{\eta}^\top - \vec{\chi}(q_0)^\top A + \vec{\chi}(q_0)^\top E - \vec{\chi}(q_0)^\top \\ &= \vec{\eta}^\top - \vec{\chi}(q_0)^\top A\end{aligned}$$

Since $A_{q_0,q} > 0$ holds for all $q \in R$, it follows that $\vec{\eta}^\top(A-E)^2$ is less than $\vec{\eta}^\top$ in all entries. Hence $\rho(A-E) < 1$. Let $\vec{x}$ be any solution of (15). Then $\vec{x}^\top - \vec{\eta}^\top = \left(\vec{x}^\top - \vec{\eta}^\top\right)(A-E)$. Since $\rho(A-E) < 1$, it follows $\vec{x} = \vec{\eta}$. □

Now we can prove Theorem 19.

Proof (of Theorem 19). We follow the same approach as in the proof of Theorem 3. Most steps can easily be carried out in NC. Instead of step 4, we compute, in NC [10, Theorem 5], the vector $\vec{\eta}$ by solving the linear system (15) in Lemma 22. From $\vec{\eta}$ we easily obtain a D-normalizer by Lemma 21. □

8. Implementation and experiments

We implemented a probabilistic model checking procedure for Markov chains and UBA specifications using the algorithm detailed in Section 4 as an extension to the probabilistic model checker `PRISM` [35] version 4.4 beta.[5] All experiments were carried out on a computer with two Intel E5-2680 8-core CPUs at 2.70 GHz with 384 GB of RAM running Linux, a time limit of 30 minutes and a memory limit of 10 GB. Our implementation is based on the `explicit` engine of `PRISM`, where the Markov chain is represented explicitly. Our implementation supports UBA-based model checking for handling the LTL fragment of `PRISM`'s PCTL*-like specification language as well as direct verification against a path specification given by a UBA provided in the HOA format [2]. For LTL formulas, we rely on external LTL-to-UBA translators. For the purpose of the benchmarks we employ the `ltl2tgba` tool from `SPOT` [20] version 2.7 to generate a UBA for a given LTL formula. For the linear algebra parts of the algorithms, we use the `COLT` library [29].

8.1. Analyzing SCCs

In our experiments we solved the linear system (5) SCC-wise, bottom-up. We call accepting recurrent SCCs *positive*. We considered two different variants for checking positivity of an SCC. The first variant relies on `COLT` to perform a QR decomposition of the matrix for the SCC to compute the rank, which allows for deciding the positivity of the SCC. The second approach is based on a variant of the power iteration method for iteratively computing an eigenvector.

Recurrence check via rank computation Let $D \subseteq Q \times S$ be an SCC. Using Theorem 1(1) we can check whether D is recurrent by seeing if the linear system $B_{D,D}\vec{x} = \vec{x}$ has a nonzero solution. It is equivalent to check whether the matrix $B_{D,D} - I$ has full rank, where I is the $D \times D$ identity matrix. Indeed, if $B_{D,D} - I$ has full rank, then it has a trivial kernel $\{0\}^D$, so $B_{D,D}\vec{x} = \vec{x}$ does not have a nonzero solution. Conversely, if $B_{D,D} - I$ does not have full rank, then there is a nonzero $\vec{x}$ with $B_{D,D}\vec{x} = \vec{x}$.

Iterative algorithm Consider again an SCC $D \subseteq Q \times S$, and define $\overline{B} = (I + B_{D,D})/2$. Denote by $\vec{1} \in \{1\}^D$ the column vector all whose components are 1. For $i \geqslant 0$ define $\vec{y}(i) = \overline{B}^i\vec{1}$. Our algorithm is as follows. Exploiting the recurrence $\vec{y}(i+1) = \overline{B}\vec{y}(i)$ compute the sequence $\vec{y}(0), \vec{y}(1), \ldots$ until we find $i > 0$ with either $\vec{y}(i+1) < \vec{y}(i)$ (by this inequality we mean strict inequality in all components) or $\vec{y}(i+1) = \vec{y}(i)$. In the first case we conclude that D is not recurrent.

In the second case we conclude that D is recurrent. If D is, in addition, accepting, then we can use the result of our iterative computation to simplify the linear system (5): we compute a cut vector $\vec{\mu}$ and a scalar $c > 0$ so that $c\vec{\mu}^\top\vec{y}(i) = 1$, and replace all equations in (5) with variables from $\vec{\zeta}_D$ on the left-hand side by $\vec{\zeta}_D = c\vec{y}(i)$. This algorithm is justified by the following two lemmas, combined with Lemma 10(2).

Lemma 23. *D is recurrent if and only if there is no $i \geqslant 0$ with $\vec{y}(i+1) < \vec{y}(i)$.*

Lemma 24. *If D is recurrent, then $\vec{y}(\infty) := \lim_{i\to\infty}\vec{y}(i) > \vec{0}$ exists, and $B\vec{y}(\infty) = \vec{y}(\infty)$, and $\vec{z}_D$ is a scalar multiple of $\vec{y}(\infty)$.*

For the proofs we need the following two auxiliary lemmas:

[5] More details are available at https://wwwtcs.inf.tu-dresden.de/ALGI/TR/JCSS19/.

Lemma 25. *For any $\vec{y} \in \mathbb{C}^D$ and any $c \in \mathbb{C}$ we have $B_{D,D}\vec{y} = c\vec{y}$ if and only if $\overline{B}\vec{y} = \frac{1+c}{2}\vec{y}$. In particular, $B_{D,D}$ and $\overline{B}$ have the same eigenvectors with eigenvalue* 1.

Proof. Immediate. □

Lemma 26. *Let $\rho > 0$ denote the spectral radius of $\overline{B}$. Then the matrix limit $\lim_{i\to\infty} \left(\overline{B}/\rho\right)^i$ exists and is strictly positive in all entries.*

Proof. Since $B_{D,D}$ is irreducible, $\overline{B}^{|D|}$ is strictly positive (in all entries). By Theorem 2(5) the matrix limit

$$\lim_{i\to\infty} \left(\overline{B}/\rho\right)^i = \lim_{i\to\infty} \left(\left(\overline{B}/\rho\right)^{|D|}\right)^i$$

exists and is strictly positive. □

Proof (of Lemma 23). Let $i \geqslant 0$ with $\vec{y}(i+1) < \vec{y}(i)$. It follows from Theorem 2(4) that the spectral radius of $\overline{B}$ is < 1. Hence by Lemma 25 the spectral radius of $B_{D,D}$ is also < 1, i.e., D is not recurrent.

For the converse, suppose D is not recurrent, i.e., the spectral radius of $B_{D,D}$ is < 1. Let ρ denote the spectral radius of $\overline{B}$. By Lemma 25, we have $\rho < 1$. If $\rho = 0$ then $\overline{B}$ is the zero matrix and we have $\vec{y}(1) = \vec{0} < \vec{1} = \vec{y}(0)$. Let $\rho > 0$. It follows from Lemma 26 that there is $i \geqslant 0$ such that $\rho\left(\overline{B}/\rho\right)^{i+1} < \left(\overline{B}/\rho\right)^i$ (with the inequality strict in all components). Hence $\overline{B}^{i+1} < \overline{B}^i$ and $\vec{y}(i+1) = \overline{B}^{i+1}\vec{1} < \overline{B}^i\vec{1} = \vec{y}(i)$. □

Proof (of Lemma 24). Let D be recurrent, i.e., the spectral radius of $B_{D,D}$ is 1. So, with Lemma 25 the spectral radius of $\overline{B}$ is 1. By Lemma 26 the limit $\vec{y}(\infty) = \lim_{i\to\infty} \overline{B}^i\vec{1}$ exists and is positive. From the definition of $\vec{y}(\infty)$ we have $\vec{y}(\infty) = \overline{B}\vec{y}(\infty)$. By Lemma 25 also $\vec{y}(\infty) = B_{D,D}\vec{y}(\infty)$. Lemma 8(1) states $\vec{z}_D = B_{D,D}\vec{z}_D$. By Theorem 2(2), the eigenspace of $B_{D,D}$ associated with the spectral radius is one-dimensional, implying that $\vec{z}_D$ is a scalar multiple of $\vec{y}(\infty)$. □

In our implementation we replace the check whether $\vec{y}(i+1) = \vec{y}(i)$ by a check whether $\vec{y}(i+1)$ and $\vec{y}(i)$ are approximately equal, up to a convergence threshold of 10^{-10}. Thus, our implementation is sound only up to these numerical issues.

8.2. Evaluation of the rank computation

The iterative (power iteration) algorithm from the previous subsection has the benefit that, in addition to deciding positivity of an SCC, the computed eigenvector can be directly used to compute the values of $\vec{z}$ corresponding to a positive SCC, once a cut has been found. We have evaluated the performance and scalability of the cut generation algorithm together with both approaches for treating SCCs with selected automata specifications that are challenging for our UBA-based model checking approach.

To assess the scalability of our implementation in the face of particularly difficult UBA, we considered two families of parametrized UBA. Both have an alphabet defined over a single atomic proposition resulting in a two-element alphabet that we use to represent either a 0 (meaning the negated atomic proposition) or a 1 bit (meaning the positive atomic proposition). The first automaton ("complete automaton"), depicted in Fig. 3 on the left for $k = 2$, is a complete automaton, i.e., recognizes Σ^ω. It consists of a single, accepting starting state that nondeterministically branches to one of 2^k states, each one leading after a further step to a k-state chain that only lets a particular k-bit bit-string pass, subsequently returning to the initial state. As all the k-bit bit-strings that can occur have a chain, the automaton is complete. Likewise, the automaton is unambiguous as each of the bit-strings can only pass via one of the chains.

Our second automaton ("nearly complete automaton"), depicted in Fig. 3 on the right for $k = 2$, arises from the first automaton by a modification of the chain for the "all zero" bit-string, preventing the return to the initial state. Clearly, the automaton is not complete.

We use both kinds of automata in an experiment using our extension of `PRISM` against a simple, two-state DTMC that encodes a uniform distribution between the two "bits". This allows us to determine whether the given automaton is almost universal. As the `PRISM` implementation requires the explicit specification of a DTMC, we end up with a product that is slightly larger than the UBA, even though we are essentially performing the UBA computations for the uniform probability distribution. In particular, this experiment serves to investigate the scalability of our implementation in practice for determining whether an SCC is positive, for the cut generation and for computing $\vec{z}$. It should be noted that equivalent deterministic automata, e.g., obtained by determinizing the UBA using the `ltl2dstar` tool, are significantly smaller (in the range of tens of states) due to the fact that the UBA in question are constructed inefficiently on purpose.

Table 1 presents statistics for our experiments with the "complete automaton" with various parameter values k, resulting in increasing sizes of the UBA and the SCC (number of states). We list the time spent for generating a cut (t_{cut}), the number of iterations in the cut generation algorithm of Lemma 18, and the size of the cut. In all cases, the cut generation requires 2 iterations. Then we compare the SCC handling based on power iteration with the SCC handling relying on a rank

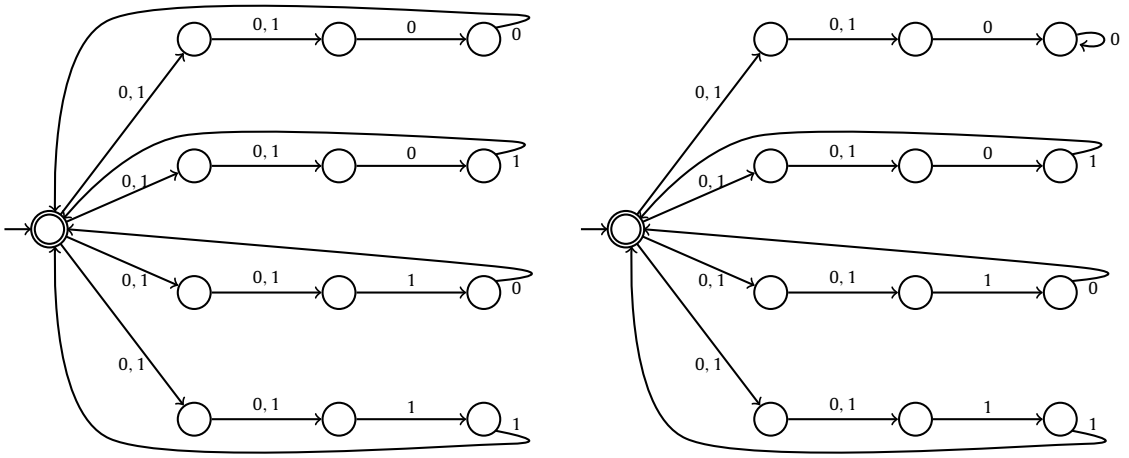

Fig. 3. UBA "complete automaton" (left) and "nearly complete automaton" (right) for $k = 2$.

Table 1
Benchmark results for "complete automaton" with parameter k. – stands for time-out.

k	$\lvert\mathcal{A}\rvert$	SCC size	cut generation			power iter.		rank-based	
			t_{cut}	ext. checks	cut size	t_{eigen}	iter.	t_{positive}	t_{values}
5	193	258	0.1 s	10124	32	$<$0.1 s	215	0.5 s	0.4 s
6	449	578	0.1 s	40717	64	$<$0.1 s	282	4.3 s	4.3 s
7	1025	1282	0.9 s	172102	128	0.1 s	358	56.5 s	56.9 s
8	2305	2818	1.8 s	929413	256	0.1 s	443	830.8 s	835.1 s
9	5121	6146	17.9 s	6818124	512	0.1 s	537	–	–

Table 2
Benchmark results for "nearly complete automaton" with parameter k.

k	$\lvert\mathcal{A}\rvert$	SCC size	power iteration		rank-based
			t_{eigen}	iter.	t_{positive}
5	193	250	$<$0.1 s	52	0.4 s
6	449	569	$<$0.1 s	78	4.1 s
7	1025	1272	$<$0.1 s	112	54.4 s
8	2305	2807	0.1 s	155	844.0 s
9	5121	6134	0.1 s	205	–

computation for determining positivity of the SCC and a subsequent computation of the values. For the power iteration method, we provide the time spent for iteratively computing an eigenvector (t_{eigen}) and the number of iterations (iter.). For the other method, we provide the time spent for the positivity check by a rank computation with a QR decomposition from the `COLT` library (t_{positive}) and for the subsequent computation of the values via solving the linear equation system (t_{values}). We used an overall timeout of 60 minutes for each `PRISM` invocation and an epsilon value of 10^{-10} as the convergence threshold.

As can be seen, the power iteration method for the numeric SCC handling performs well, with a modest increase in the number of iterations for rising k until converging on an eigenvector, as it can fully exploit the sparseness of the matrix. The QR decomposition for rank computation performs worse. The time for cut generation exhibits a super-linear relation with k, which is reflected in the larger number of words that were checked to determine that they are an extension. Note that our example was chosen in particular to put stress on the cut generation.

The results for the "nearly complete automaton", shown in Table 2, focus on the computation in the "dominant SCC", i.e., the one containing all the chains that return to the initial state. For the other SCC, containing the self-loop, non-positivity is immediately clear as it does not contain a final state. In contrast to the "complete automaton", no cut generation takes place, as the SCC is not positive. The results roughly mirror the ones for the "complete automata", i.e., the power iteration method is quite efficient in determining that the SCC is not positive, while the QR decomposition for the rank computation needs significantly more time and scales worse.

As the power iteration method performed better, our benchmark results presented in the following subsections use this method for the SCC handling.

Table 3
Statistics for DBA/DRA- and UBA-based model checking of the BRP case study, a DTMC with 29358 states, showing the number of states for the automata and the product and the time for model checking (t_{MC}). For $\mathcal{B}$, the time for checking positivity (t_{Pos}) is included in t_{MC}. The mark – stands for "not available" or timeout (30 minutes).

	PRISM standard			PRISM UBA			
	$\lvert\mathcal{A}^k_{DRA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}^k_{DRA}\rvert$	t_{MC}	$\lvert\mathcal{A}^k_{UBA}\rvert$	$\lvert\mathcal{M}\otimes\mathcal{A}^k_{UBA}\rvert$	t_{MC}	t_{Pos}
$k=4$, $\mathcal{A}^4$	33	61,025	0.4 s	6	34,118	0.3 s	
$\mathcal{B}^4$	33	75,026	0.4 s	6	68,474	1.3 s	1.0 s
$k=6$, $\mathcal{A}^6$	129	62,428	0.5 s	8	36,164	0.2 s	
$\mathcal{B}^6$	129	97,754	0.5 s	8	99,460	1.7 s	1.3 s
$k=8$, $\mathcal{A}^8$	513	64,715	0.6 s	10	38,207	0.3 s	
$\mathcal{B}^8$	513	134,943	0.7 s	10	136,427	2.6 s	2.1 s
$k=14$, $\mathcal{A}^{14}$	32,769	83,845	4.2 s	16	44,340	0.3 s	
$\mathcal{B}^{14}$	32,769	444,653	4.9 s	16	246,346	6.8 s	6.1 s
$k=16$, $\mathcal{A}^{16}$	131,073	–	–	18	46,390	0.3 s	
$\mathcal{B}^{16}$	131,037	–	–	18	282,699	8.9 s	8.0 s
$k=48$, $\mathcal{A}^{48}$	–	–	–	50	79,206	0.8 s	
$\mathcal{B}^{48}$	–	–	–	50	843,414	72.4 s	70.3 s

8.3. Case study: bounded retransmission protocol

Next we report on benchmarks using the bounded retransmission protocol (BRP) case study of the PRISM benchmark suite [36]. The model from the benchmark suite covers a single message transmission, retrying for a bounded number of times in case of an error. In this protocol the message is split into several so-called hunks. The number of hunks and the number of allowed retransmissions are parameters in the model. We set the number of hunks to 16 and the maximal of retransmissions to 128.

We have slightly modified the model to allow the transmission of an infinite number of messages by restarting the protocol once a message has been successfully delivered or the bound for retransmissions has been reached. We include benchmarks with pre-generated automata, as well as benchmarks with LTL as starting point. We include also the evaluation for deterministic Rabin automata generated by Rabinizer from [33].

Automata based specifications We consider the property "the message was retransmitted k steps before an acknowledgment." To remove the effect of selecting specific tools for the LTL to automaton translation (ltl2tgba for UBA, the Java-based PRISM reimplementation of ltl2dstar [31] to obtain a deterministic Rabin automaton (DRA) for the PRISM standard approach), we first consider model checking directly against automata specifications. As the language of the property is equivalent to the UBA depicted in Fig. 1 (on the left) where a stands for a retransmission, b for an acknowledgment, and c for no acknowledgment, we use this automaton and the minimal DBA for the language (this case is denoted by $\mathcal{A}^k$). We additionally consider the UBA and DBA obtained by replacing the self-loop in the last state with a switch back to the initial state (denoted by $\mathcal{B}^k$), i.e., roughly applying the ω-operator to $\mathcal{A}^k$.

Table 3 shows results for selected k (with a timeout of 30 minutes), demonstrating that for this case study and properties our UBA-based implementation is generally competitive with the standard approach of PRISM based on deterministic automata. For $\mathcal{A}^k$, our implementation detects that the UBA has a special shape where all final states have a true-self loop which allows skipping the SCC handling. If we execute the positivity check nevertheless, t_{Pos} is in the sub-second range for all considered $\mathcal{A}^k$. At a certain point, the implementation of the standard approach in PRISM becomes unsuccessful, due to PRISM size limitations in the product construction of the Markov chain and the deterministic automaton ($\mathcal{A}^k/\mathcal{B}^k$: $k \geqslant 16$). As can be seen, using the UBA approach we can scale the parameter k beyond 48 when dealing directly with the automata-based specifications ($\mathcal{A}^k/\mathcal{B}^k$) and within reasonable time required for model checking.

LTL based specifications We consider two LTL properties: The first one is:

$$\varphi^k = (\neg \texttt{ack\_received})\, \mathcal{U}\, \big(\texttt{retransmit} \wedge (\neg \texttt{ack\_received}\, \mathcal{U}^{=k}\, \texttt{ack\_received})\big),$$

where $a\mathcal{U}^{=k}b$ stands for $a \wedge \neg b \wedge \bigcirc(a \wedge \neg b) \wedge \ldots \wedge \bigcirc^{k-1}(a \wedge \neg b) \wedge \bigcirc^k b$. The formula φ^k ensures that k steps before an acknowledgment the message was retransmitted. Hence, it is equivalent to the property described by the automaton $\mathcal{A}^k$. For the LTL-to-automaton translation we included the Java-based PRISM reimplementation of ltl2dstar [31] to obtain a deterministic Rabin automaton (DRA) for the PRISM standard approach as well as the tool Rabinizer (version 3.1) from [23]. For the generation of UBA, we relied on SPOT (version 2.7), as it is the only tool that is capable of generating UBA explicitly.

Table 4
Statistics for automata-based (standard, `Rabinizer`, and UBA) model checking of the BRP model and φ^k. For every approach the corresponding automata sizes and product sizes are depicted. The overall model checking times (t_{MC}) are listed, which includes the time for automata translation.

k	PRISM standard			PRISM Rabinizer			PRISM UBA		
	$\mathcal{A}_{DRA}$	$\lvert\mathcal{M}\otimes\mathcal{A}_{DRA}\rvert$	t_{MC}	$\mathcal{A}_{Rab}$	$\lvert\mathcal{M}\otimes\mathcal{A}_{Rab}\rvert$	t_{MC}	$\mathcal{A}_{UBA}$	$\lvert\mathcal{M}\otimes\mathcal{A}_{UBA}\rvert$	t_{MC}
4	122	62,162	1.7 s	18	60,642	0.6 s	6	34,118	0.6 s
6	4,602	72,313	3.3 s	66	61,790	0.6 s	8	36,164	0.5 s
8	–	–	–	258	63,698	1.0 s	10	38,207	0.6 s
10	–	–	–	1,026	66,739	3.8 s	12	40,249	0.7 s
12	–	–	–	4,098	71,660	38.5 s	14	42,293	1.0 s
14	–	–	–	16,386	79,576	925.5 s	16	44,340	5.8 s
16	–	–	–	–	–	–	18	46,390	132.9 s

Table 5
Statistics for automata-based (standard, `Rabinizer`, and UBA) of the BRP model and ψ^k. The structure of this table corresponds to Table 4, but with additional listing of the time for the positivity checks t_{pos} and cut calculation time t_{cut}. n/a means not available.

k	PRISM standard			PRISM Rabinizer			PRISM UBA				
	$\mathcal{A}_{DRA}$	$\lvert\mathcal{M}\otimes\mathcal{A}_{DRA}\rvert$	t_{MC}	$\mathcal{A}_{Rab}$	$\lvert\mathcal{M}\otimes\mathcal{A}_{Rab}\rvert$	t_{MC}	$\mathcal{A}_{UBA}$	$\lvert\mathcal{M}\otimes\mathcal{A}_{UBA}\rvert$	t_{pos}	t_{Cut}	t_{MC}
1	6	29,358	0.7 s	5	29,358	0.4 s	4	31,422	<0.1 s	n/a	0.3 s
2	17	37,678	0.9 s	7	35,630	0.5 s	8	41,822	4.8 s	0.2 s	5.4 s
3	65	39,726	1.1 s	11	37,678	0.5 s	14	45,934	5.2 s	0.2 s	5.8 s
4	314	43,806	1.5 s	23	41,758	0.6 s	22	54,126	5.8 s	0.3 s	6.6 s
5	1,443	47,902	2.3 s	59	45,854	0.9 s	32	62,334	6.5 s	0.3 s	7.3 s
6	9,016	56,029	5.3 s	167	53,997	2.1 s	44	78,669	9.5 s	0.2 s	10.3 s
7	67,964	–	–	491	58,081	9.6 s	58	86,853	10.4 s	0.2 s	11.3 s
8	–	–	–	1,463	66,217	76.1 s	74	103,157	13.9 s	0.3 s	15.0 s
9	–	–	–	4,379	70,291	783.7 s	92	111,321	15.2 s	0.3 s	16.8 s
10	–	–	–	–	–	–	112	127,562	20.1 s	0.3 s	22.7 s

Table 4 lists the results for model checking φ^k. From a certain point on, the implementation of the standard approach in `PRISM` is unsuccessful, due to `PRISM`'s restrictions in the DRA construction ($k \geqslant 8$). Concerning automata sizes and model checking times, `SPOT` shows the best behavior among `PRISM` standard and `Rabinizer`. `SPOT` actually generates a UFA for φ^k which is recognized by our implementation and handled as explained in [6]. The sizes of the unambiguous automata output by `SPOT` grow linearly in k, whereas the sizes of the deterministic automata output by `Rabinizer` grow exponentially. Thus, `PRISM` with `Rabinizer` times out after 30 minutes for $k = 15$. However, `SPOT` produces for φ^k an exponential-sized intermediate automaton, which is then shrunk via bisimulation to an automaton of linear size. Thus, our implementation `PRISM` UBA times out for $k = 18$.

As a second formula we investigate

$$\psi^k = \Box(\texttt{msg\_send} \to \Diamond(\texttt{ack\_send} \wedge \Diamond^{\leqslant k}\texttt{ack\_received})),$$

where $\Diamond^{\leqslant k}a$ denotes $a \vee \underbrace{\bigcirc(a \vee \bigcirc(\ldots \vee \bigcirc a))}_{k \text{ times}}$. This formula requires that every request (sending a message and waiting for an acknowledgment) is eventually responded to by an answer (the receiver of the message sends an acknowledgment and this acknowledgment is received within the next k steps).

Table 5 summarizes the results of the benchmark for ψ^k. Here, the `PRISM` standard approach with its own implementation of `ltl2dstar` finishes the calculations until $k = 6$. The sizes of the DRA produced by `PRISM`'s `ltl2dstar` increase rapidly with k. For $k = 7$, `PRISM` standard can construct the DRA (with 67,964 states and within 37.0 seconds), but cannot construct the product anymore. Similarly, the sizes of the DRA produced by `Rabinizer` grow rapidly in k. The sizes of the DRA of `Rabinizer` are smaller than the size of the UBA for $k \geqslant 4$.

In contrast to the deterministic automata, the UBA sizes increase moderately with k. In the UBA approach the positivity check is the most time consuming part of the calculation, whereas the cut generation is always below 0.4 seconds. For $k = 1$ there is no positive SCC, so the cut calculation is omitted. The model checking process consumes more time in the UBA case in comparison with `PRISM` standard until $k = 6$, but for bigger k the performance turns around. Even if `PRISM` standard were to complete the calculation for $k = 7$, it would be slower, as the creation of the DRA takes 37.0 seconds. Similarly, `PRISM Rabinizer` outperforms `PRISM` UBA for $k \leqslant 8$, which is due to the time-consuming positivity check. For bigger k, both `PRISM` standard and `PRISM Rabinizer` cannot finish their calculation within the given time bound, whereas `PRISM` UBA finishes the calculations for all tested $k \leqslant 12$.

Table 6
Number of formulas where the (standard) NBA and UBA has a number of states $\leqslant x$.

Number of states $\leqslant x$	$\leqslant 1$	$\leqslant 2$	$\leqslant 3$	$\leqslant 4$	$\leqslant 5$	$\leqslant 7$	$\leqslant 10$	$\leqslant 20$	$\geqslant 20$
`ltl2tgba` NBA	12	49	103	145	158	176	181	188	0
`ltl2tgba` UBA	12	42	74	108	123	153	168	180	8

8.4. NBA versus UBA

To gain some understanding of the cost of requiring unambiguity for an NBA, we compare the sizes of NBA and UBA generated by the `ltl2tgba` tool of `SPOT` for the formulas of [24,45,22], which have been used for benchmarking, e.g., in [31]. We consider both the "normal" formulas and their negations, yielding 188 formulas.

As can be seen in Table 6, both the NBA and UBA are of reasonable size. Most of the generated UBA (102) have the same size as the NBA and for 166 of the formulas the UBA is at most twice the size as the corresponding NBA. The largest UBA has 112 states, the second largest has 45 states.

9. Conclusion

We have presented a polynomial-time algorithm for Markov chain analysis against properties given as unambiguous automata. The algorithm is based on the analysis of nonnegative matrices and exploits in particular their spectral theory.

As LTL formulas can be transformed into UBA with a single exponential blow-up, our algorithm yields a procedure for model checking LTL formulas on Markov chains that is singly exponential in the formula size. We have moreover refined the process of UBA generation and Markov chain analysis to achieve the optimal PSPACE upper bound for computing the exact probability that a given Markov chain satisfies a given LTL formula.

We have developed an extension of `PRISM` that supports our approach. Its experimental evaluation shows that the UBA-based method is very competitive with the approach using deterministic automata, outperforming the latter in certain cases.

For the other singly exponential approaches to LTL model checking, such as using separated automata [18] or weak alternating automata [12], we are not aware of any available implementation to compare our approach against.[6] Our algorithm for arbitrary UBA can be seen as a generalization of the approach of [18], which requires separated UBA [40].

Markov chain analysis via general UBA, rather than separated UBA or deterministic automata, offers additional flexibility that can be exploited when building automata from LTL formulas. In particular, it facilitates use of state-reduction techniques, such as simulation, that may not preserve the separatedness property. As our experiments (specifically with the bounded retransmission protocol) suggest, the eigenvalue algorithm can deduce non-positiveness of an SCC very efficiently in practice. For the generation of UBA we have used the tool `SPOT`, which implements a simple and straightforward way to produce unambiguous Büchi automata [21]. Alternatively, `Tulip` contains an LTL-to-UBA translator; but this tool is not available anymore. As with the approach using deterministic automata, the performance of the UBA-based method depends strongly on the availability of small UBA. In contrast to nondeterministic or deterministic automata, the generation of small UBA and their simplification has not yet been explored thoroughly.

CRediT authorship contribution statement

The authors contributed equally.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Supplemental material concerning our implementation and experiments is available at https://wwwtcs.inf.tu-dresden.de/ALGI/TR/JCSS19/

References

[1] André Arnold, Deterministic and non ambiguous rational omega-languages, in: Automata on Infinite Words, Ecole de Printemps d'Informatique Théorique, Le Mont Dore, May 1984, in: Lecture Notes in Computer Science, vol. 192, Springer, 1985, pp. 18–27.

[6] The paper [18] addresses experiments with a prototype implementation, but this implementation seems not to be available anymore.

[2] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, Jan Strejček, The Hanoi omega-automata format, in: 27th International Conference on Computer Aided Verification (CAV), in: Lecture Notes in Computer Science, vol. 9206, Springer, 2015, pp. 479–486.
[3] Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, James Worrell, Markov chains and unambiguous Büchi automata, in: Proc. of the 28th International Conference on Computer Aided Verification (CAV) - Part I, in: Lecture Notes in Computer Science, vol. 9779, Springer, 2016, pp. 23–42.
[4] Christel Baier, Stefan Kiefer, Joachim Klein, Sascha Klüppelholz, David Müller, James Worrell, Markov chains and unambiguous Büchi automata (extended version), arXiv:1605.00950, 2016, http://arxiv.org/abs/1605.00950.
[5] Michael Benedikt, Rastislav Lenhardt, James Worrell, LTL model checking of interval Markov chains, in: 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), in: Lecture Notes in Computer Science, vol. 7795, Springer, 2013, pp. 32–46.
[6] Michael Benedikt, Rastislav Lenhardt, James Worrell, Model checking Markov chains against unambiguous Büchi automata, arXiv:1405.4560, 2014.
[7] Michael Benedikt, Ratislav Lenhardt, James Worrell, Two variable vs. linear temporal logic in model checking and games, Log. Methods Comput. Sci. 9 (2) (2013).
[8] Abraham Berman, Robert J. Plemmons, Nonnegative Matrices in the Mathematical Sciences, SIAM, 1994.
[9] Allan Borodin, On relating time and space to size and depth, SIAM J. Comput. 6 (4) (1977) 733–744.
[10] Allan Borodin, Joachim von zur Gathen, John Hopcroft, Fast parallel matrix and GCD computations, Inf. Control 52 (3) (1982) 241–256.
[11] Nicolas Bousquet, Christof Löding, Equivalence and inclusion problem for strongly unambiguous Büchi automata, in: 4th International Conference on Language and Automata Theory and Applications (LATA), in: Lecture Notes in Computer Science, vol. 6031, Springer, 2010, pp. 118–129.
[12] Doron Bustan, Sasha Rubin, Moshe Y. Vardi, Verifying ω-regular properties of Markov chains, in: 16th International Conference on Computer Aided Verification (CAV), in: Lecture Notes in Computer Science, vol. 3114, Springer, 2004, pp. 189–201.
[13] Souymodip Chakraborty, Joost-Pieter Katoen, Parametric LTL on Markov chains, in: 8th IFIP TC 1/WG 2.2 International Conference on Theoretical Computer Science, in: Lecture Notes in Computer Science, vol. 8705, Springer, 2014, pp. 207–221.
[14] Thomas Colcombet, Forms of determinism for automata (invited talk), in: 29th International Symposium on Theoretical Aspects of Computer Science (STACS), in: LIPIcs, vol. 14, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 1–23.
[15] Thomas Colcombet, Unambiguity in automata theory, in: 17th International Workshop on Descriptional Complexity of Formal Systems (DCFS), in: Lecture Notes in Computer Science, vol. 9118, Springer, 2015, pp. 3–18.
[16] Costas Courcoubetis, Mihalis Yannakakis, Verifying temporal properties of finite-state probabilistic programs, in: 29th Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1988, pp. 338–345.
[17] Costas Courcoubetis, Mihalis Yannakakis, The complexity of probabilistic verification, J. ACM 42 (4) (1995) 857–907.
[18] Jean-Michel Couvreur, Nasser Saheb, Grégoire Sutre, An optimal automata approach to LTL model checking of probabilistic systems, in: 10th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR), in: Lecture Notes in Computer Science, vol. 2850, Springer, 2003, pp. 361–375.
[19] Alexandre Duret-Lutz, Manipulating LTL formulas using Spot 1.0, in: 11th International Symposium on Automated Technology for Verification and Analysis (ATVA), in: Lecture Notes in Computer Science, vol. 8172, Springer, 2013, pp. 442–445.
[20] Alexandre Duret-Lutz, LTL translation improvements in Spot 1.0, Int. J. Crit. Compt.-Based Syst. 5 (1/2) (2014) 31–54.
[21] Alexandre Duret-Lutz, Contributions to LTL and ω-Automata for Model Checking, Habilitation thesis, Université Pierre et Marie Curie (Paris 6), 2017.
[22] Matthew B. Dwyer, George S. Avrunin, James C. Corbett, Patterns in property specifications for finite-state verification, in: 21st International Conference on Software Engineering (ICSE), ACM, 1999, pp. 411–420.
[23] Javier Esparza, Jan Křetínský, Salomon Sickert, From LTL to deterministic automata - a Safraless compositional approach, Form. Methods Syst. Des. 49 (3) (2016) 219–271.
[24] Kousha Etessami, Gerard Holzmann, Optimizing Büchi automata, in: 11th International Conference on Concurrency Theory (CONCUR), in: Lecture Notes in Computer Science, vol. 1877, Springer, 2000, pp. 153–167.
[25] Erich Grädel, Wolfgang Thomas, Thomas Wilke (Eds.), Automata, Logics, and Infinite Games: A Guide to Current Research, Lecture Notes in Computer Science, vol. 2500, Springer, 2002.
[26] Yo-Sub Han, Arto Salomaa, Kai Salomaa, Ambiguity, nondeterminism and state complexity of finite automata, Acta Cybern. 23 (1) (2017) 141–157.
[27] Thomas A. Henzinger, Nir Piterman, Solving games without determinization, in: 20th International Workshop on Computer Science Logic (CSL), in: Lecture Notes in Computer Science, vol. 4207, Springer, 2006, pp. 395–410.
[28] Roger A. Horn, Charles R. Johnson, Matrix Analysis, 2nd edition, Cambridge University Press, 2013.
[29] Wolfgang Hoschek, The Colt distribution: open source libraries for high performance scientific and technical computing in Java, 2004.
[30] Dimitri Isaak, Christof Löding, Efficient inclusion testing for simple classes of unambiguous ω-automata, Inf. Process. Lett. 112 (14–15) (2012) 578–582.
[31] Joachim Klein, Christel Baier, Experiments with deterministic ω-automata for formulas of linear temporal logic, Theor. Comput. Sci. 363 (2) (2006) 182–195.
[32] Joachim Klein, David Müller, Christel Baier, Sascha Klüppelholz, Are good-for-games automata good for probabilistic model checking?, in: 8th International Conference on Language and Automata Theory and Applications (LATA), in: Lecture Notes in Computer Science, vol. 8370, Springer, 2014, pp. 453–465.
[33] Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, Christopher Ziegler, Rabinizer 4: from LTL to your favourite deterministic automaton, in: 30th International Conference on Computer Aided Verification (CAV), Part I, in: Lecture Notes in Computer Science, vol. 10981, Springer, 2018, pp. 567–577.
[34] Vidyadhar G. Kulkarni, Modeling and Analysis of Stochastic Systems, Chapman & Hall, 1995.
[35] Marta Z. Kwiatkowska, Gethin Norman, David Parker, PRISM 4.0: verification of probabilistic real-time systems, in: 23rd International Conference on Computer Aided Verification (CAV), in: Lecture Notes in Computer Science, vol. 6806, Springer, 2011, pp. 585–591.
[36] Marta Z. Kwiatkowska, Gethin Norman, David Parker, The PRISM benchmark suite, in: 9th International Conference on Quantitative Evaluation of SysTems (QEST), IEEE Computer Society, 2012, pp. 203–204.
[37] Rastislav Lenhardt Tulip, Model checking probabilistic systems using expectation maximisation algorithm, in: 10th International Conference on Quantitative Evaluation of Systems (QEST), in: Lecture Notes in Computer Science, vol. 8054, Springer, 2013, pp. 155–159.
[38] Rastislav Lenhardt, Two Variable and Linear Temporal Logic in Model Checking and Games, PhD thesis, University of Oxford, 2013.
[39] Andreas Morgenstern, Symbolic Controller Synthesis for LTL Specifications, PhD thesis, Technische Universität Kaiserslautern, 2010.
[40] David Müller, Alternative Automata-Based Approaches to Probabilistic Model Checking, PhD thesis, Technische Universität Dresden, 2018.
[41] Christos M. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, Massachusetts, 1994.
[42] Alexander Rabinovich, Complementation of finitely ambiguous Büchi automata, in: 22nd International Conference on Developments in Language Theory (DLT), in: Lecture Notes in Computer Science, Springer, 2018, pp. 541–552.
[43] Ravikumar Bala, Oscar H. Ibarra, Relating the type of ambiguity of finite automata to the succinctness of their representation, SIAM J. Comput. 18 (6) (1989) 1263–1282.
[44] Salomon Sickert, Javier Esparza, Stefan Jaax, Jan Křetínský, Limit-deterministic Büchi automata for linear temporal logic, in: 28th International Conference on Computer Aided Verification (CAV), in: Lecture Notes in Computer Science, vol. 9780, Springer, 2016, pp. 312–332.

[45] Fabio Somenzi, Roderick Bloem, Efficient Büchi automata from LTL formulae, in: 12th International Conference on Computer Aided Verification (CAV), in: Lecture Notes in Computer Science, vol. 1855, Springer, 2000, pp. 248–263.
[46] Moshe Y. Vardi, Automatic verification of probabilistic concurrent finite-state programs, in: 26th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1985, pp. 327–338.
[47] Moshe Y. Vardi, Pierre Wolper, An automata-theoretic approach to automatic program verification (preliminary report), in: 1st Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 1986, pp. 332–344.
[48] Pierre Wolper, Moshe Y. Vardi, A. Prasad Sistla, Reasoning about infinite computation paths (extended abstract), in: 24th Annual Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1983, pp. 185–194.
[49] Martin Zimmermann, Optimal bounds in parametric LTL games, Theor. Comput. Sci. 493 (2013) 30–45.