Unambiguous and finitely ambiguous automata

Cas Widdershoven

St Edmund Hall

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

Acknowledgements

Firstly I would like to thank Stefan Kiefer for providing excellent guidance and advice. Under his supervision I have grown from merely a learner of computer science to a computer science communicator and teacher (while also still being a learner). I would also like to thank Pavel Semukhin for our collaboration on branching processes, and the insightful discussions we had during that time.

Secondly I am very grateful to the Department of Computer Science at Oxford University as well as Balliol College. In particular I would like to thank Ani Calinescu from the department, and Tom Melham from Balliol. Working under them did not only enable me financially to finish this degree, but also allowed me to discover my passion for teaching.

A massive thank you goes out to all my friends from the Oxford University Ice Hockey Club and ALTS Ice Hockey Club Oxford. Nathan Tree, for being a massive inspiration with the most goals of any blind guy on our team. Stefano Sacco, for many a game of pool during our time as secretary. Yuri Evdokimov, Abi Owen, Sasha Webb, and many others, for many late hours having an amazing time on the ice.

I also would like to thank St. Cross College and all the friends I made there for providing a home away from home.

Back in the Netherlands I am very thankful to the friends I made in my undergraduate. Without the infectious ambition from Ivor van der Hoog and Mark Kamsma I would not have gotten where I am now. I thank my oldest and closest friends, Saïd Henstra, Tjerk Hidde van der Werf, and Wilfried Damen, for always keeping me grounded with a deck of cards and a nice bonfire.

Last but most definitely not least, I would like to thank my parents and my sister for always being there for me, for always supporting me and always inspiring me.

Abstract

This thesis is concerned with unambiguous and finitely ambiguous automata, with a focus on automata on infinite words. In particular, it is concerned with the model checking question, which asks whether the traces of a (stochastic and / or non-deterministic) system satisfy a property given by an automaton. By studying these automata through the lens of matrix semigroups and making use of their spectral properties, we show that the model checking question can be solved efficiently in several different settings.

Our first result is an adaptation of Baier et al.'s method for model checking Markov chains against unambiguous Büchi automata. This method expresses the model checking problem as a system of linear equations, given by a product of the automaton and Markov chain combined with a normalising equation for each strongly connected component of this product. By replacing their combinatorial algorithm for computing normalisers with one based on linear algebra, we obtain a speedup in the (edge set) size of the automaton at the cost of a slowdown in the size of the Markov chain.

Next we introduce a type of weighted automata called image-binary automata, generalising unambiguous automata. We show that these automata are closed efficiently under the standard Boolean operations, implying an efficient algorithm to check whether a given $\mathbb{Q}$-weighted automaton is image-binary, and that they accept precisely the regular languages. We show that image-binary automata can be exponentially more succinct than deterministic automata, that an exponential blowup in the state size is sufficient and necessary for converting NFAs to image-binary automata, and that converting an image-binary automaton to an NFA may require a superpolynomial state size blowup. We then consider image-binary automata over infinite words, with a Büchi acceptance condition. We show that k-ambiguous Büchi automata can be translated to image-binary Büchi automata using a PSPACE transducer, and that the model checking question of Markov chains against image-binary Büchi automata can be done in NC using an adaptation of Baier et al.'s algorithm for UBAs. Combining these gives a PSPACE procedure for model checking k-ABAs against Markov chains, implying optimality of both the translation from k-ABAs to image-binary Büchi automata as well as the model checking procedure.

Lastly, we consider model checking of branching processes against LTL formulas using an automata theoretic approach. Branching processes are a model exhibiting both stochastic and non-deterministic branching, generalising both Markov chains and Kripke structures. We consider both the questions whether it's almost surely the case that a branching process generates a tree all of whose branches are accepted by a given automaton, and whether this is almost never the case, as well as the question whether a branching process almost surely generates a finite tree. We consider specifications given in terms of deterministic parity automata, NBAs, co-NBAs (where a tree is accepted iff all of its branches are rejected by a given NBA), co-UBAs (where all branches have to be rejected by a given UBA), and finally, LTL formulas. In general, this paints a picture where the question whether a branching process almost surely generates an accepted tree is easier than the question whether it almost never generates an accepted tree. By using the spectral properties of unambiguous automata, we show that the question whether a branching process almost surely generates a tree all of whose branches are rejected by a UBA is in NC. Combining this with a PSPACE transduction from an LTL formula to a UBA, we obtain an optimal procedure to check whether a branching process almost surely generates a tree satisfying an LTL formula.

Contents

1 Introduction

How do you make sure a system performs according to specification? This question is as old as there are systems, and there are many ways to approach this problem, also known as the verification problem. The two main branches of verification are dynamic verification and static verification, also known as testing and analysis. This thesis falls in the category of analysis.

Whereas testing takes a black box approach to verification, analysis takes a white box approach. This means that the problem assumes a formal model of the system and the specification, and asks whether the system meets the specification. Generally the system is given by a (possibly stochastic) finite-state model such as a Markov chain or a Kripke structure, while the specification is given by a logical formula, and the model checking question asks whether the formula is true in the given model.

A central issue in model checking lies in the complexity of the formal models. While more powerful modeling languages allow for a broader class of models to be represented, and may be able to describe certain models more concisely than less powerful languages can, they also increase the computational complexity of the model checking question, and it is well-known that for the most powerful logics the model checking question even becomes undecidable. This can be observed for instance through Gödel's incompleteness theorems or the halting problem.

Currently model checking techniques have evolved to the point where there are many robust and performant tools available. Examples of these include PRISM [47], SPIN [37], and UPPAAL [8]. PRISM is a probabilistic model checker, it checks probabilistic models against logical specifications. Some of the models it supports are (continuous- and discrete-time) Markov chains, Markov decision processes, and

probabilistic automata, while the specification language incorporates well-known languages such as PCTL* and LTL. SPIN focuses on the verification of multi-threaded software. The system is described in the input language PROMELA, which is based on Dijkstra's guarded command languages and Hoare's CSP language, and can be checked against specifications given in LTL. UPPAAL is a tool for verifying real-time systems. The models are networks of timed automata, extended with data types, and UPPAAL supports checking these models against safety and bonded-liveness properties. An overview comparing some model checking tools can be found at [52].

Instead of defining the specification in terms of a logical sentence, this thesis is mostly concerned with model checking finite-state models against a specification given in terms of an automaton. There are many different types of automata, specifying both finite-word languages and infinite-word languages. For languages over infinite words we generally consider automata with a Büchi acceptance condition, specifying a set of final states such that any accepted word has an accepting path containing infinitely many final states. In Chapters 3 to 5 we consider the problem of model checking automata against Markov chains, and in Chapter 6 we consider model checking automata against a model known as a branching process, that combines both stochastic and non-deterministic branching.

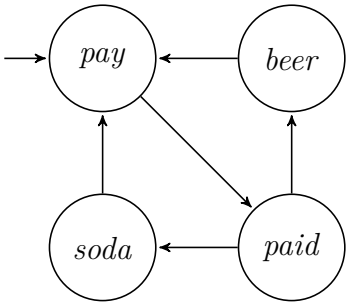
Example 1.1. *As an example of model checking in practice, we use the simple vending machine example from [5]. The transition system in Figure 1.1a describes a vending machine that dispenses a beverage after payment. After inserting a coin it can provide either a soda or a beer, this customer choice is modeled as non-deterministic branching in the system. One property that should hold true in this system is the following:*

the vending machine only delivers a soda or a beer after being paid.

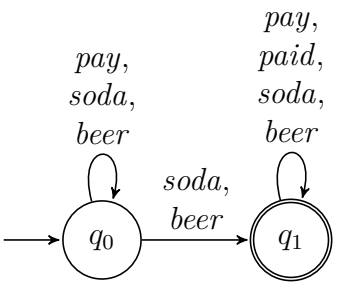
In LTL this is written as

$$(\neg(soda \vee beer)\mathcal{U}paid) \vee \mathcal{G}\neg(soda \vee beer).$$

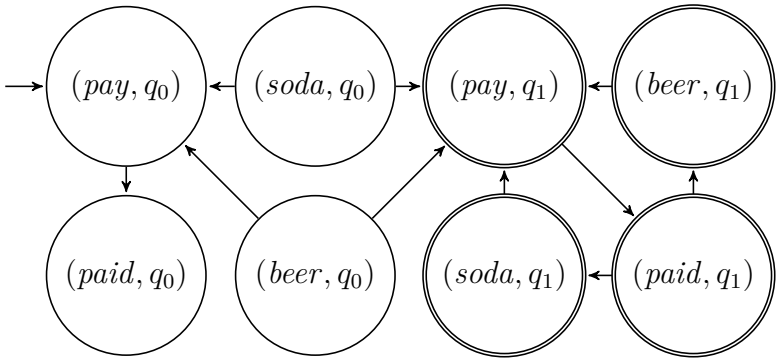
A common way of checking whether this formula holds true over all *traces in the system is by negating the formula, which results in* $\neg paid\mathcal{U}(soda \vee beer)$*, and checking whether* some *trace satisfies this formula. Expressing* $\neg paid\mathcal{U}(soda \vee beer)$ *in terms of a deterministic finite automaton results in the DFA in Figure 1.1b. To check whether some trace satisfies the specification given by this DFA, we take the product of the transition system and DFA (given in Figure 1.1c) and check if it is*



(a) A simple vending machine model Kripke structure. The initial state is indicated by an incoming arrow.



(b) A DFA expressing $\neg paid \mathcal{U} (soda \vee beer)$. The initial state is indicated by an incoming arrow and the final state by a double outline.



(c) The product of the vending machine model in Figure 1.1a and the DFA in Figure 1.1b. The initial state is indicated by an incoming arrow and the final states by a double outline.

possible to reach an accepting state. Since in this example there are no reachable accepting states, we know that there are no traces of the transition system satisfying the specification given by the DFA, and hence no trace satisfies $\neg paid \mathcal{U} (soda \vee beer)$*, i.e., all traces satisfy* $(\neg (soda \vee beer) \mathcal{U} paid) \vee \mathcal{G} \neg (soda \vee beer)$.

A central property in automata theory is non-determinism. Whereas a non-deterministic automaton can move into several possible successor states upon reading an input symbol, a deterministic automaton is restricted such that it can move into at most one successor state. This makes deterministic automata less powerful than their non-deterministic counterparts, but it also makes many questions computationally easier to solve. While the emptiness problem is in P for both deterministic finite-word automata and non-deterministic automata, the universality problem, and as a consequence the inclusion and equivalence problems, are in P for deterministic finite-word automata but PSPACE-complete for non-deterministic automata. Finally model checking Kripke structures and Markov chains against deterministic automata, i.e., computing whether all traces of the Kripke structure

are accepted by the automaton, or computing the probability that a trace generated by the Markov chain is accepted, are in P, but against non-deterministic automata these problems are PSPACE-complete. However, non-determinstic automata can be exponentially more concise than any deterministic automaton accepting the same language. All of these results can be found in any standard textbook on automata theory, see for instance [5].

For the infinite-word case we see a similar picture. Emptiness is in P both for deterministic Büchi automata and non-deterministic automata, but universality, inclusion, and equivalence are in P for deterministic Büchi automata and PSPACE-complete for non-deterministic Büchi automata[20]. We also have that model-checking Kripke structures and Markov chains against deterministic Büchi automata are in P, but against non-deterministic Büchi automata these problems are PSPACE-complete[20]. Moreover, while in the finite-word case, deterministic and non-deterministic automata are equi-expressive (through the subset construction), we have that deterministic Büchi automata are strictly less expressive than non-deterministic Büchi automata, and even for languages that can be expressed using deterministic Büchi automata, non-deterministic Büchi automata can be exponentially more concise, which follows from the finite-word case.

A lot of work has been done trying to find a middle ground between non-deterministic automata and deterministic automata, combining the desirable computational complexity results of deterministic automata with the conciseness of non-deterministic automata. This has resulted in a few models, such as limit-deterministic automata, which only allow non-deterministic transitions in an initial segment without final states, strongly-unambiguous automata, where for each word there exists exactly one accepting run (not necessarily starting in an initial state), and unambiguous automata, for which there exists for each word at most one accepting run starting in an initial state. Another model worth mentioning are synchronizing automata, where there exists a word w and a state q such that the automaton ends up in q upon reading w no matter which state it started in. However, this property does not imply better computational complexity results for questions such as universality or model checking.

Unambiguous automata allow for polynomial model checking against Markov chains[7], also implying a polynomial-time algorithm for the almost-unversality problem, which asks given a Markov chain and an unambiguous automaton, is it true that almost all words generated by the Markov chain are accepted by the automaton? However, this does not imply a polynomial-time algorithm for the universality problem, which asks if *all* words are accepted. To solve this question, one might try

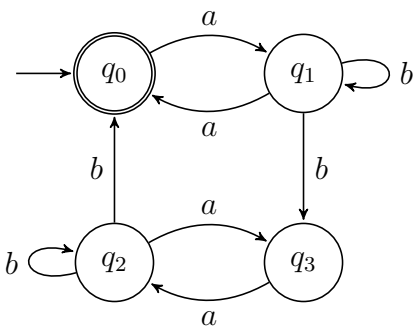


Figure 1.2: An unambiguous automaton $\mathcal{A}$.

complementing the automaton and checking the complement for emptiness. However, as it turns out complementation of an unambiguous automaton may result in an automaton with superpolynomially many states, even if the resulting automaton is not required to be unambiguous[40]. This thesis focuses on unambiguous automata as well as the related concept of finitely ambiguous automata.

1.1 Thesis overview

Several of the results in this thesis are based upon the model checking procedure for unambiguous Büchi automata against Markov chains by Baier et al[6]. For this reason, Chapter 3 is dedicated to explaining this procedure.

Given a finite automaton $\mathcal{A}$, one can specify a probability distribution on words, e.g., with a Markov chain, and ask for the probability that a word is accepted by $\mathcal{A}$. For instance, if $\Sigma = \{a, b\}$, one may generate a random word, letter by letter, by outputting a, b with probability 1/3 each, and ending the word with probability 1/3. For an NFA $\mathcal{A}$, determining whether the probability of generating an accepted word is 1 is equivalent to universality (is $L(\mathcal{A}) = \Sigma^*$?), which as we know is a PSPACE-complete problem[5]. However, if $\mathcal{A}$ is unambiguous, i.e., every accepted word has exactly one accepting path, then one can compute the probability of generating an accepted word in polynomial time by solving a linear system of equations. Clearly, the probability of $\mathcal{A}$ with initial state q accepting a word that starts with a character $c \in \Sigma$ is given by the probability of the union of words accepted by $\mathcal{A}$ with initial state q' for all c-successors q' of q. Unambiguousness allows us to express the probability of this union as the sum of probabilities of $\mathcal{A}$ with initial state q' accepting:

Example 1.2. *Consider the unambiguous automaton $\mathcal{A}$ in Figure 1.2, i.e., the automaton with state set $\{q_0, q_1, q_2, q_3\}$ on an alphabet $\{a, b\}$ where q_0 is the only initial state as well as the only final state, q_0 has a-successor q_1 and no b-successors, q_1 has a-successor q_0 and b-successors q_1 and q_3, q_2 has a-successor q_3 and b-successors q_0 and q_2, and q_3 has a-successor q_2 and no b-successors. If we generate a random word over $\{a, b\}$ according to the process described above, we have the following linear system for the vector $\vec{z}$ where $\vec{z}_q$ is, for each $q \in \{q_0, q_1, q_2, q_3\}$, the probability that the word is accepted when q is taken as initial state:*

$$\begin{aligned} \vec{z}_{q_0} &= \tfrac{1}{3}\vec{z}_{q_1} + \tfrac{1}{3} & \qquad \vec{z}_{q_1} &= \tfrac{1}{3}\vec{z}_{q_0} + \tfrac{1}{3}(\vec{z}_{q_1} + \vec{z}_{q_3}) \\ \vec{z}_{q_2} &= \tfrac{1}{3}\vec{z}_{q_3} + \tfrac{1}{3}(\vec{z}_{q_0} + \vec{z}_{q_2}) & \qquad \vec{z}_{q_3} &= \tfrac{1}{3}\vec{z}_{q_2} \end{aligned}$$

The constant term in the equation for $\vec{z}_{q_0}$ reflects the fact that q_0 is accepting. The (other) coefficients $\frac{1}{3}$ correspond to the production of either a or b. The brackets in the equations for $\vec{z}_{q_1}$ and $\vec{z}_{q_2}$ reflect the fact that q_1 and q_2 have 2 b-successors. The linear system has a unique solution.

One may view an NFA $\mathcal{A}$ as a Büchi automaton, so that its language $L(\mathcal{A}) \subseteq \Sigma^\omega$ is the set of those infinite words that have an accepting run in $\mathcal{A}$, i.e., a run that visits accepting states infinitely often. There is a natural notion of an infinite random word over Σ: in each step sample a letter from Σ uniformly at random, e.g., if $\Sigma = \{a, b\}$ then choose a and b with probability $1/2$ each. Perhaps more significantly, model checking Markov chains against Büchi automata, i.e., computing the probability that the random word generated by the Markov chain is accepted by the automaton, is a key problem in the verification of probabilistic systems:

Problem 1.3 (Model checking Markov chains against automata)**.** *Given an automaton $\mathcal{A}$ and a Markov chain $\mathcal{M}$, what is the probability that $\mathcal{M}$ generates a word accepted by $\mathcal{A}$?*

As we know, for general non-deterministic Büchi automata, this problem is PSPACE-complete. However, if the Büchi automaton is unambiguous, i.e., every accepted (infinite) word has exactly one accepting path, then one can compute the probability of generating an accepted word in polynomial time [6], both in the given Büchi automaton and in a given (discrete-time, finite-state) Markov chain. Since LTL specifications can be converted to unambiguous Büchi automata with a single-exponential blow-up, this leads to an LTL model-checking algorithm with single-exponential runtime, which is optimal. The polynomial-time algorithm from [6] for unambiguous Büchi automata is more involved than in the finite-word case.

Example 1.4. *In the following we view the automaton $\mathcal{A}$ from Figure 1.2 as an (unambiguous) Büchi automaton. If we generate a random word over $\{a, b\}$ according to the process described above, we have the following linear system for the vector $\vec{z}$ where $\vec{z}_q$ is, for each $q \in \{q_0, q_1, q_2, q_3\}$, the probability that the word is accepted when q is taken as initial state:*

$$\begin{aligned} \vec{z}_{q_0} &= \tfrac{1}{2}\vec{z}_{q_1} & \qquad \vec{z}_{q_1} &= \tfrac{1}{2}\vec{z}_{q_0} + \tfrac{1}{2}(\vec{z}_{q_1} + \vec{z}_{q_3}) \\ \vec{z}_{q_2} &= \tfrac{1}{2}\vec{z}_{q_3} + \tfrac{1}{2}(\vec{z}_{q_0} + \vec{z}_{q_2}) & \qquad \vec{z}_{q_3} &= \tfrac{1}{2}\vec{z}_{q_2} \end{aligned}$$

This linear system has multiple solutions: indeed, any scalar multiple $(1, 2, 2, 1)^\top$ is a solution.

In order to make such a linear system uniquely solvable, one needs to add further equations, and finding these further equations is where the real challenge lies. Assuming that the state space Q of $\mathcal{A}$ is strongly connected and the Markov chain generates letters uniformly at random as described above, a single additional equation $\vec{\mu}^\top \vec{z} = 1$ suffices (this can be shown with Perron-Frobenius theory: the eigenspace for the dominant eigenvalue of a non-negative irreducible matrix is one-dimensional). We call such a vector $\vec{\mu} \in \mathbb{R}^Q$ a *normaliser*: i.e., a *normaliser* is any vector $\vec{\mu}$ such that $\vec{\mu}^\top \vec{z} = 1$, where $\vec{z}_q$ is the probability that a word is accepted by the automaton starting in q.

In general, for model checking general Markov chains against Büchi automata, $\vec{z}$ needs to take into account the initial state of the Markov chain as well, and hence will be a vector in $\mathbb{R}^{Q \times S}$, where S is the state space of the Markov chain. In this case, the definition of normalisers needs to be changed accordingly. The formal definitions are given in Chapter 3.

The suggestion in [6] was to take as normaliser the characteristic vector $\llbracket c \rrbracket \in \{0, 1\}^Q$ of a so-called *cut* $c \subseteq Q$, i.e., the vector $\llbracket c \rrbracket$ such that $\llbracket c \rrbracket_q = 1$ whenever $q \in c$ and $\llbracket c \rrbracket_q = 0$ otherwise. To define cuts, let us write $\delta(q, w)$ for the set of states reachable from a state $q \in Q$ via the word $w \in \Sigma^*$. A *cut* is a set of states of the form $c = \delta(q, w)$ such that $\delta(q, wx) \neq \emptyset$ holds for all $x \in \Sigma^*$. If a cut does not exist or if $\mathcal{A}$ does not have accepting states, then one may take $\vec{z} = \vec{0}$. Chapter 3 contains a more formal definition of cuts taking into account the structure of the Markov chain.

Example 1.5. *In the automaton $\mathcal{A}$ from Figure 1.2, we have a cut $c = \delta(q_0, aba) = \{q_0, q_2\}$. Hence its characteristic vector $\vec{\mu} = (1, 0, 1, 0)^\top$ is a normaliser, allowing us to add the equation $\vec{\mu}^\top \vec{z} = \vec{z}_{q_0} + \vec{z}_{q_2} = 1$. Now the system is uniquely solvable: $\vec{z} = \frac{1}{3}(1, 2, 2, 1)^\top$. The equation $\vec{z}_{q_0} + \vec{z}_{q_2} = 1$ is valid by an ergodicity argument:*

intuitively, given a finite word that leads to q_0 and q_2*, a random infinite continuation will almost surely enable an accepting run. For instance,* $\vec{z}_{q_0} = \frac{1}{3}$ *is the probability that a random infinite word over* $\{a, b\}$ *has an odd number of as before the first* b*. (This holds despite the fact that the word* $abbb\ldots$ *is not accepted from* q_0*.)*

In Proposition 3.16 we show that an efficient implementation of the algorithm from [6] for computing a cut runs in time $O(|Q|^5)$. Let the constant $\kappa \in [2, 3)$ be such that one can multiply two $n \times n$ matrices in $O(n^\kappa)$. The precise value of the lowest such κ is unknown, and determining this κ is an active field of research. One can take, for instance, $\kappa = 2.37286$[2]. This results in the following theorem:

Theorem 1.6. *Given a Markov chain* $\mathcal{M} = (S, M)$ *with edge set* E*, an initial distribution* ι*, and a UBA* $\mathcal{A} = (Q, S, \delta, Q_0, F)$*, one can compute* $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ *in time* $O(|Q|^\kappa|S|^\kappa + |Q|^3|\delta||S| + |\delta|^2|E|)$*, where* $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ *denotes the probability that* $\mathcal{M}$ *with initial distribution* ι *generates a word in the language* $\mathcal{L}(\mathcal{A})$ *of words accepted by* $\mathcal{A}$*.*

In Chapter 4 we improve on this result by replacing the cut-based procedure by our own *pseudo-cut* based approach, obtaining a quadratic speedup in the size of the automaton. This chapter is based upon our results from [43].

The general idea is to move from a combinatorial problem, namely computing a set $c \subseteq Q$, to a continuous problem, namely computing a vector $\vec{\mu} \in \mathbb{R}^Q$. To illustrate this, note that since we can choose as $\vec{\mu}$ the characteristic vector of an arbitrary cut, we may also choose a convex combination of such vectors, leading to a normaliser $\vec{\mu}$ with entries other than 0 or 1.

The technical key ideas of Chapter 4 draw on the observation that for unambiguous automata with cuts, the transition matrices generate a semigroup of matrices whose spectral radii are all 1. (The spectral radius of a matrix is the largest absolute value of its eigenvalues.) This observation enables us to adopt techniques that have independently been devised by Protasov and Voynov [57] for the analysis of matrix semigroups with constant spectral radius. To the best of the authors' knowledge, such semigroups have not previously been connected to unambiguous automata.

To sketch the gist of this technique, for any $a \in \Sigma$ write $M(a) \in \{0, 1\}^{Q \times Q}$ for the transition matrix of the unambiguous automaton $\mathcal{A}$, define the average matrix $\overline{M} = \frac{1}{|\Sigma|} \sum_{a \in \Sigma} M(a)$, and let $\vec{y} = \overline{M}\vec{y} \in \mathbb{R}^Q$ be an eigenvector with eigenvalue 1 (the matrix $\overline{M}$ has such an eigenvector if $\mathcal{A}$ has a cut). Since the matrix semigroup, $\mathcal{S} \subseteq \{0, 1\}^{Q \times Q}$, generated by the transition matrices $M(a)$ has constant spectral radius, it follows from [57] that one can efficiently compute an affine space $\mathcal{F} \subseteq \mathbb{R}^Q$

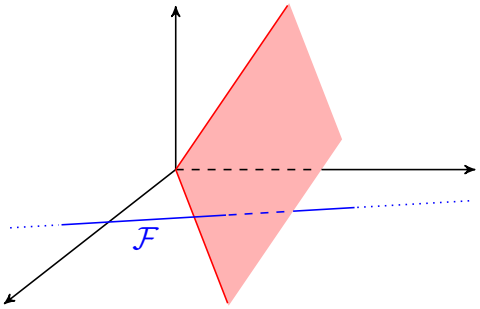


Figure 1.3: A visualisation of the affine space closed under the transition matrices of the automaton in Figure 1.2. Characteristic vectors of cuts lie in the plane shaded in red, which is orthogonal to the straight line $\mathcal{F}$ (blue).

with $\vec{y} \in \mathcal{F}$ and $\vec{0} \notin \mathcal{F}$ such that for any $\vec{v} \in \mathcal{F}$ and any $M \in \mathcal{S}$ we have $M\vec{v} \in \mathcal{F}$. Using the fact that $\delta(q, wx)$ is a cut (for all $x \in \Sigma^*$) whenever $\delta(q, w)$ is a cut, one can show that all characteristic vectors of cuts have the same scalar product with all $\vec{v} \in \mathcal{F}$, i.e., all characteristic vectors of cuts are in the vector space orthogonal to $\mathcal{F}$. Indeed, we choose as normaliser $\vec{\mu}$ a vector that is orthogonal to $\mathcal{F}$. This linear-algebra computation can be carried out in time $O(|Q|^3)$. In the visualisation in Figure 1.3, the characteristic vectors of cuts lie in the plane shaded in red, which is orthogonal to the straight line $\mathcal{F}$ (blue).

Example 1.7. *Recall the automaton $\mathcal{A}$ from Figure 1.2, let $M(a)$ and $M(b)$ denote the transition matrices of $\mathcal{A}$ upon reading a or b, respectively. The vector $\vec{y} = (1, 2, 2, 1)^\top$ satisfies $\overline{M}\vec{y} = \vec{y}$ where $\overline{M} = \frac{1}{2}(M(a) + M(b))$. The affine space $\mathcal{F} := \{\vec{y} + s(1, -1, -1, 1)^\top \mid s \in \mathbb{R}\}$ has the mentioned closure properties, i.e., $M(a)\mathcal{F} \subseteq \mathcal{F}$ and $M(b)\mathcal{F} \subseteq \mathcal{F}$. Note that the vector $\vec{\mu}$ from Example 1.5 is indeed orthogonal to $\mathcal{F}$, i.e., $\vec{\mu}^\top(1, -1, -1, 1)^\top = 0$.*

However, to ensure that $\vec{\mu}$ is a valid normaliser, we need to restrict it further. To this end, we compute, for some state $q \in Q$, the set $Co(q) \subseteq Q$ of *co-reachable* states, i.e., states $r \in Q$ such that $\delta(q, w) \supseteq \{q, r\}$ holds for some $w \in \Sigma^*$. This requires a combinatorial algorithm, which is similar to a straightforward algorithm that would verify the unambiguousness of $\mathcal{A}$. Its runtime is quadratic in the number of transitions of $\mathcal{A}$, i.e., $O(|Q|^4)$ in the worst case. Then we restrict $\vec{\mu}$ to be 1 in $\vec{\mu}_q$ and non-zero only in entries that correspond to $Co(q)$. In the visualisation in Figure 1.3, restricting some components of $\vec{\mu}$ to be 0 corresponds to intersecting the shaded plane with the coordinate plane spanned by $Co(q)$.

Example 1.8. *We have $Co(q_0) = \{q_0, q_2\}$. So we restrict $\vec{\mu}$ to be of the form $(1, 0, x, 0)^\top$. Together with the equation $\vec{\mu}^\top(1, -1, -1, 1)^\top = 0$ this implies $\vec{\mu} =$*

$(1,0,1,0)^\top$. *The point is that, although this is the same vector computed via a cut in Example 1.5, the linear-algebra based computation of* $\vec{\mu}$ *is more efficient.*

Using these techniques we show the following main theorem, showcasing a quadratic speedup in the size of the automaton for model checking UBAs against MCs using pseudo-cuts:

Theorem 1.9. *Given a Markov chain* $\mathcal{M} = (S, M)$*, an initial distribution* ι*, and a UBA* $\mathcal{A} = (Q, S, \delta, Q_0, F)$*, one can compute* $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ *in time* $O(|Q|^\kappa |S|^\kappa + |Q|^3|E| + |\delta|^2|E|)$*, where* $\kappa \in [2,3)$ *is such that one can multiply two* $n \times n$*-matrices in* $O(n^\kappa)$*.*

In Chapter 4 we will flesh out this theory in the more general case of model checking a Markov chain against a given unambiguous Büchi automaton. The efficiency gain we aim for with our technique can only be with respect to the automaton, not the Markov chain; nevertheless, we analyse in detail the runtime in terms of the numbers of states and transitions in both the automaton and the Markov chain. One might note that Theorem 1.9 only improves over Theorem 1.6 by a factor δ; however, in Section 4.3 we show that in general δ can be quadratic in Q.

In Chapter 5 we introduce a new type of weighted automata, based on our paper called Image-Binary Automata[44]. A weighted automaton assigns weights to words; i.e., it defines mappings of the form $f : \Sigma^* \to D$, where D is some domain of weights. Weighted automata are well-studied. Many variations have been discussed, such as max-plus automata [22] and probabilistic automata [54; 58], both over finite words and over infinite words, in the latter case often combined with ω-valuation monoids [23]. However, it has been shown that many natural questions are undecidable for many kinds of weighted automata [1; 28], including inclusion and equivalence. These problems become decidable for finitely ambiguous weighted automata [29].

In this chapter we consider only numerical weights, where D is a subfield of the reals. A *language* $L \subseteq \Sigma^*$ can be identified with its characteristic function $\chi_L : \Sigma^* \to \{0,1\}$. We explore weighted automata that encode characteristic functions of languages, i.e., weighted automata that map each word either to 0 or to 1. We call such automata *image-binary finite automata (IFAs)* and view them as acceptors of languages $L \subseteq \Sigma^*$. We do not require, however, that individual transitions have weight 0 or 1. This makes IFAs a "semantic" class: it may not be obvious from the transition weights whether a given weighted automaton over, say,

the rationals is image-binary. However, we will see that it can be checked efficiently whether a given $\mathbb{Q}$-weighted automaton is an IFA (Theorem 5.10).

An immediate question is on the expressive power of IFAs. Deterministic finite automata (DFAs) can be viewed as IFAs. On the other hand, in Section 5.2.2 we show that all languages accepted by IFAs are regular. It follows that IFAs accept exactly the regular languages. Moreover, IFAs are efficiently closed under Boolean operations; i.e., given two IFAs that accept L_1, L_2, respectively, one can compute in polynomial time IFAs accepting $L_1 \cup L_2$, $L_1 \cap L_2$, and $\Sigma^* \setminus L_1$.

The latter feature, efficient closure under complement, might be viewed as a key advantage of IFAs over unambiguous finite automata (UFAs). UFAs can be viewed as a special case of IFAs. Whereas UFAs are known to be not polynomially closed under complement [60], we have shown that IFAs can be complemented in polynomial time.

The next question is then on the succinctness of IFAs, and on the complexity of converting other types of finite automata to IFAs and vice versa. We study such questions in Section 5.2.4. In Section 5.2.5, we also study the relationship of IFAs to *mod-2 multiplicity automata*, which are weighted automata over $\mathbb{Z}_2$, the field $\{0, 1\}$ where $1 + 1 = 0$. Such automata [3] share various features with IFAs, in particular efficient closure under complement.

In the second part of the chapter we put IFAs "to work". Specifically, we consider an infinite-word version, which we call *image-binary Büchi automata (IBAs)*. Following the theme that image-binary automata naturally generalise and relax unambiguous automata, we show that IBAs can be used for model checking Markov chains in essentially the same way as *unambiguous Büchi automata (UBAs)* (Chapter 3, [7]). Specifically, we show in Section 5.3.3 that given an IBA and a Markov chain, one can compute in NC (hence in polynomial time) the probability that a random word produced by the Markov chain is accepted by the IBA.

It was shown in [51] that a non-deterministic Büchi automaton (NBA) with n states can be converted to an NBA with at most 3^n states whose ambiguity is bounded by n. Known conversions from NBAs to UBAs have a state blowup of roughly n^n, see, e.g., [41]. We show in Section 5.3.2 that NBAs with logarithmic ambiguity (as produced by the construction from [51]) can be converted to IBAs in polylogarithmic space. This suggests that in order to translate NBAs into an automaton model suitable for probabilistic model checking (such as IBAs), it is reasonable to first employ the partial disambiguation procedure from [51]. More specifically, by combining the partial disambiguation procedure from [51] with our translation to an IBA, we obtain a PSPACE transducer (i.e., a Turing machine whose work tape is polynomially bounded) that translates an NBA into an IBA.

For example, combining that with the mentioned probabilistic model checking procedure for IBAs we obtain an (optimal) PSPACE procedure for model checking Markov chains against NBA specifications.

In Chapter 6, based on our paper called Linear-Time Model Checking Branching Processes[42] (in collaboration with Pavel Semukhin), we consider an (unambiguous) automata-theoretic approach to model checking LTL formulas against branching processes, which are a kind of model that allows for both stochastic and non-deterministic branching. Although LTL captures only a subset of ω-regular languages, model-checking algorithms based on the automata-theoretic approach can be made optimal from the point of view of computational complexity. In particular, model checking finite Kripke structures against LTL specifications is PSPACE-complete [64], and the algorithm [70] that, loosely speaking, translates (the negation of) the LTL formula into a Büchi automaton and checks the product with the transition system for emptiness can indeed be implemented in PSPACE.

The same approach does not directly work for probabilistic systems modelled as finite Markov chains: intuitively, the non-determinism in a Büchi automaton causes issues in a stochastic setting where the specification should hold with probability 1, i.e., almost surely but not necessarily surely. A possible remedy is to translate the non-deterministic Büchi automaton further into a deterministic automaton, e.g., a deterministic Rabin automaton, with which the Markov chain can be naturally instrumented and subsequently analyzed. This determinization step causes a (second) exponential blowup and does not lead to algorithms that are optimal from a computational-complexity point of view. However, for *Markov decision processes (MDPs)*, which allow for non-determinism in the probabilistic system, this approach is adequate and leads to an optimal, double-exponential time, model-checking algorithm.

Checking whether a Markov chain satisfies an LTL specification with probability 1 is PSPACE-complete, but membership in PSPACE was proved only in [19; 20], not using the automata-theoretic approach but by a recursive procedure on the formula. An optimal automata-theoretic approach to model checking Markov chains against an LTL specification is given by translating the LTL formula to an *unambiguous* Büchi automaton (incurring an exponential blowup) and model checking the unambiguous automaton in polylogarithmic space, for instance using (a polylogarithmic space restricted version of) the algorithm from Chapter 3.

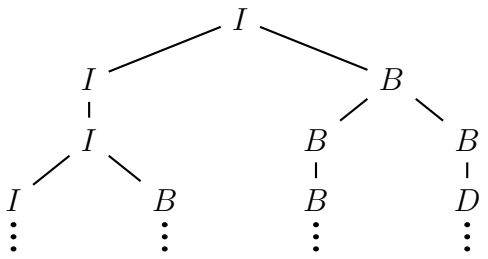
In Chapter 6 we exhibit a qualitative LTL model checking algorithm that has the following features: (1) it applies to (multi-type) branching processes, a well established model for random trees, generalizing both Kripke structures

and Markov chains; (2) it runs in PSPACE, which is the optimal complexity both for non-deterministic transition systems and Markov chains; and (3) it is based on the automata-theoretic approach (using unambiguous Büchi automata). The fact that there exists an algorithm with the first two features might seem surprising, as one might think that any system model that encompasses both non-determinism and probability will generalise MDPs, for which LTL model checking is 2EXPTIME-complete [20].

Branching processes (BPs) are a well-studied model in mathematics with applications in numerous fields including biology, physics and natural language processing; see, e.g., [4; 33; 34]. BPs randomly generate infinite trees, and, from a computer-science point of view, they might be the most natural model to do so: (multi-type) BPs can be thought of as a version of stochastic context-free grammars without terminal symbols, randomly generating infinite derivation trees. For example, consider the following BP, taken from [17], with 3 *types* I, B, D:

$$\begin{array}{lll} I \overset{0.9}{\hookrightarrow} I & B \overset{0.2}{\hookrightarrow} D & D \overset{1}{\hookrightarrow} D \\ I \overset{0.1}{\hookrightarrow} IB & B \overset{0.5}{\hookrightarrow} B & \\ & B \overset{0.3}{\hookrightarrow} BB & \end{array} \tag{1.1}$$

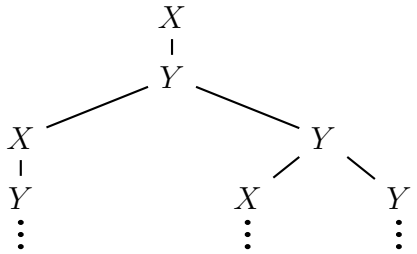
A random tree that this BP generates might have the following prefix:



The probability that the BP generates a tree with the shown prefix is the product of the probabilities of the fired transition rules, i.e., (in breadth-first order) $0.1 \cdot 0.9 \cdot 0.3 \cdot 0.1 \cdot 0.5 \cdot 0.2$.

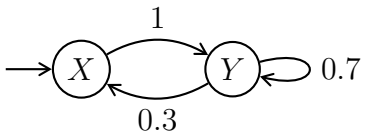
BPs generalise Kripke structures. Consider the following Kripke structure:

It is equivalent to the BP with $X \overset{1}{\hookrightarrow} Y$ and $Y \overset{1}{\hookrightarrow} XY$, which generates with probability 1 the following unique tree:

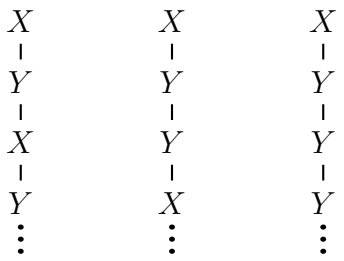


The branches of this unique tree are exactly the executions of the Kripke structure. As a consequence, any LTL formula holds on all executions of the Kripke structure if and only if it holds (with probability 1) on all branches of the generated tree.

BPs also generalise Markov chains. Consider the following Markov chain:



It is equivalent to the BP with $X \overset{1}{\hookrightarrow} Y$ and $Y \overset{0.3}{\hookrightarrow} X$ and $Y \overset{0.7}{\hookrightarrow} Y$, which generates, with probabilities 0.3, $0.7 \cdot 0.3$, $0.7 \cdot 0.7$, respectively, the following prefixes of (degenerated) trees:



Here, each possible "tree" has only a single branch, and the possible "trees" are distributed in the same way as the possible executions of the Markov chain. As a consequence, any LTL formula holds with probability 1 on a random execution of the Markov chain if and only if it holds with probability 1 on the (single) branch of the generated tree.

Hence, both for the Kripke structure and for the Markov chain, the respective model-checking question reduces to the *BP model-checking problem* which asks whether with probability 1 the property holds on *all* branches.

For LTL specifications, we refer to this BP model-checking problem as $\mathbb{P}(\text{LTL}) = 1$. The main result of Chapter 6 is that $\mathbb{P}(\text{LTL}) = 1$ is in PSPACE (in both the size of the formula and the size of the branching process), generalizing the corresponding classical results on Kripke structures and Markov chains. As mentioned, our model-checking algorithm is based on the automata-theoretic approach, in particular

on unambiguous Büchi automata. Another important technical ingredient is the algorithmic analysis of certain non-negative matrices in terms of their spectral radius.

The latter points to the fact that the numbers in the system generally matter, even though we only consider the qualitative problem of comparing the satisfaction probability with 1. For example, for the BP given in (1.1), one can show that the probability that all branches eventually hit a node of type D is less than 1 (in fact, it is 0). Intuitively, this is because the probability of "branching" via $B \overset{0.3}{\hookrightarrow} BB$ is larger than the probability of "dying" via $B \overset{0.2}{\hookrightarrow} D$. Were the probabilities 0.3 and 0.2 swapped, the probability that all branches eventually hit a node of type D would be 1; cf. [17, Section 1]. This shows that even for the qualitative properties considered in Chapter 6, the quantitative probabilities in the branching process matter.

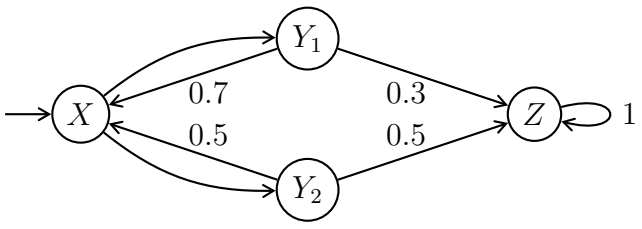
We also consider the problem $\mathbb{P}(\mathrm{LTL} = 0)$, which asks whether the probability that *all* branches satisfy a given LTL formula is 0. Even though it is trivial to negate an LTL formula, this problem is (unlike in Markov chains) not equivalent to the complement of $\mathbb{P}(\mathrm{LTL} = 1)$, because even when the probability is less than 1 that the formula holds on all branches, the probability may still be 0 that the negated formula holds on all branches. We will show that $\mathbb{P}(\mathrm{LTL} = 0)$ is much more computationally complex than $\mathbb{P}(\mathrm{LTL} = 1)$: it is 2EXPTIME-complete.

Besides LTL, we also consider automata-based specifications. Büchi automata are relevant from a verification point of view, as a way of specifying desired or undesired executions of the system. Unambiguous Büchi automata are useful from a technical point of view, in particular, to facilitate our main result on $\mathbb{P}(\mathrm{LTL} = 1)$. See Section 6.2 and table 6.1 for definitions of our problems and a map of our results.

Remark 1.10. *Readers familiar with MDPs may wonder how the problem* $\mathbb{P}(\mathrm{LTL}) = 1$ *can have lower computational complexity than the problem whether all schedulers of an MDP satisfy an LTL specification almost surely. Consider the BP*

$$X \overset{1}{\hookrightarrow} Y_1 Y_2 \qquad Y_1 \overset{0.7}{\hookrightarrow} X \qquad Y_1 \overset{0.3}{\hookrightarrow} Z \qquad Y_2 \overset{0.5}{\hookrightarrow} X \qquad Y_2 \overset{0.5}{\hookrightarrow} Z \qquad Z \overset{1}{\hookrightarrow} Z\,,$$

which might be depicted graphically as follows:



One might view this BP as an MDP where in an X-node the scheduler non-deterministically picks either the Y_1- or the Y_2-successor, and in an Y_i-node, the X- or the Z-successor is chosen randomly. In such an MDP, regardless of the scheduler, a random run reaches with probability 1 *a Z-node. However, in the BP above, the probability is positive that* some *branch of a random tree never reaches a Z-node. Although each branch of a random tree could be thought of as being witnessed by at least one scheduler, this is not a contradiction, as there are uncountably many schedulers (over which one cannot take a sum). Hence, if an MDP is interpreted as a BP in the way sketched above, then the requirement that the BP satisfy an LTL formula almost surely on* all *branches is* stronger*, and computationally less complex to check, than the requirement that the MDP satisfy, for* each *scheduler, the formula almost surely.*

The rest of this thesis is structured as follows: in Chapter 3 we explain the procedure for model checking Markov chains against unambiguous Büchi automata by Baier et al.[7] and give our own complexity analysis of this method. This procedure underlies several of the results later in this thesis. Chapter 4 is based on [43]. In this chapter we replace the cut-based procedure for calculating normalisers from Chapter 3 with our own method based on pseudo-cuts and matrix semigroups, improving the runtime for model checking Markov chains against unambiguous automata by a factor linear in the state set size of the automaton. In Chapter 5 we introduce a new type of weighted automata called image-binary automata. Based on [44], we give several succinctness and closure properties, and show that an infinite-word version is amenable for model checking using a modified method of the model checking technique in Chapter 3. Finally, in Chapter 6, we consider qualitative model checking of branching processes based on [42]. Using spectral properties of unambiguous automata, we show that one can check in PSPACE whether a branching process almost surely generates a tree all of whose branches adhere to an LTL specification.

2
Preliminaries

We assume the reader to be familiar with basic notions of finite automata over infinite words and Markov chains, see, e.g., [32; 46]. In the following we provide a brief summary of our notation and a few facts related to linear algebra.

Finite automata. A *Büchi automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ where Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, Σ is the finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states. 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. For $q \in Q$ we write $\mathcal{A}_q$ for the automaton obtained from $\mathcal{A}$ by making q the only initial state.

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 $w = a_0 a_1 a_2 \cdots \in \Sigma^\omega$ 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 $\inf(\rho) \cap F \neq \emptyset$ 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. $\mathcal{A}$ is called *unambiguous* if each word $w \in \Sigma^\omega$ has at most one accepting run. We use the acronym UBA for unambiguous Büchi automaton.

We define $|\delta| := |\{(q, r) \mid \exists\, a \in \Sigma : r \in \delta(q, a)\}|$, i.e., $|\delta| \leq |Q|^2$ is the number of transitions in $\mathcal{A}$ when allowing for multiple labels per transition. In Section 4.3 we give an example that shows that the number of transitions can be quadratic in $|Q|$, even for UBAs with strongly connected state space. We assume $|Q| \leq |\delta|$,

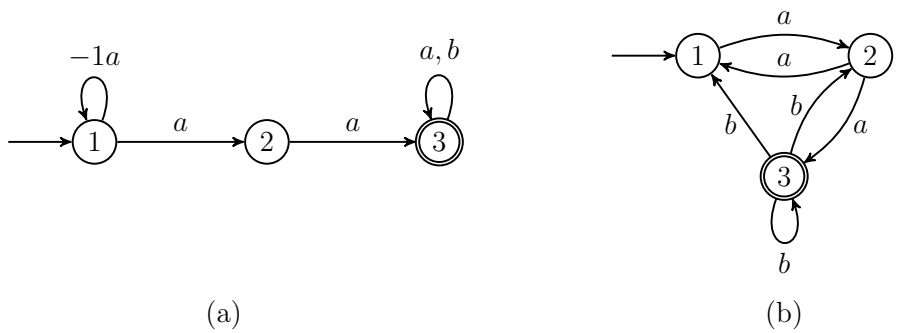


Figure 2.1: The weighted automaton in (a) is a forward conjugate of, and hence equivalent to, the UFA in (b). Unless indicated otherwise, edges in (a) have weight 1.

as states without outgoing transitions can be removed. In this thesis, Σ may be a large set (of states in a Markov chain), so it is imperative to allow for multiple labels per transition. We use a lookup table to check in constant time whether $r \in \delta(q, a)$ holds for given r, q, a.

A *diamond* is given by two states $q, r \in Q$ and a finite word w such that there exist at least two distinct runs for w from q to r. One can remove diamonds:

Lemma 2.1. *Given a UBA, one can compute in time $O(|\delta|^2|\Sigma|)$ a UBA of at most the same size, with the same language and without diamonds.*

Let $\mathbb{F}$ be one of the fields $\mathbb{Q}$ or $\mathbb{R}$ (with ordinary addition and multiplication). An $\mathbb{F}$-*weighted automaton* $\mathcal{A} = (Q, \Sigma, M, \alpha, \eta)$ consists of a set of states Q, a finite alphabet Σ, a map $M : \Sigma \to \mathbb{F}^{Q\times Q}$, an initial (row) vector $\alpha \in \mathbb{F}^Q$, and a final (column) vector $\eta \in \mathbb{F}^Q$. Extend M to Σ^* by setting $M(a_1 \cdots a_k) \stackrel{\text{def}}{=} M(a_1) \cdots M(a_k)$. The *language* $L_\mathcal{A}$ of an $\mathbb{F}$-weighted automaton $\mathcal{A}$ is the map $L_\mathcal{A} : \Sigma^* \to \mathbb{F}$ with $L_\mathcal{A}(w) = \alpha M(w)\eta$. Automata $\mathcal{A}, \mathcal{B}$ over the same alphabet Σ are said to be *equivalent* if $L_\mathcal{A} = L_\mathcal{B}$.

Let $\mathcal{A} = (Q, \Sigma, M, \alpha, \eta)$ be an $\mathbb{R}$-weighted automaton. We call $\overrightarrow{\mathcal{A}} := (\overrightarrow{Q}, \Sigma, \overrightarrow{M}, \overrightarrow{\alpha}, F\eta)$ a *forward conjugate* of $\mathcal{A}$ with base F if $F \in \mathbb{R}^{\overrightarrow{Q}\times Q}$ and $FM(a) = \overrightarrow{M}(a)F$ for all $a \in \Sigma$ and $\alpha = \overrightarrow{\alpha}F$. Such $\mathcal{A}$ and $\overrightarrow{\mathcal{A}}$ are equivalent: indeed, let $w \in \Sigma^*$; by induction we have $FM(w) = \overrightarrow{M}(w)F$ and hence

$$L_\mathcal{A}(w) \;=\; \alpha M(w)\eta \;=\; \overrightarrow{\alpha}FM(w)\eta \;=\; \overrightarrow{\alpha}\overrightarrow{M}(w)F\eta \;=\; L_{\overrightarrow{\mathcal{A}}}(w)\,.$$

A backward conjugate can be defined analogously.

Example 2.2. *The weighted automaton $\mathcal{A}$ on the left of Figure 2.1 is a forward conjugate of the UFA on the right with base*

$$F = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

Indeed, we have $\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} F = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$ *and* $\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T = F \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$*, where* $\vec{v}^T$ *denotes the transpose of a vector* $\vec{v}$*, and*

$$F \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} F \quad \textit{and} \quad F \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} F.$$

For some proofs we need the following definition. Let $L : \Sigma^* \to \mathbb{F}$, where $\mathbb{F}$ is any field. Then the *Hankel matrix* of L is the infinite matrix $H_L \in \mathbb{F}^{\Sigma^* \times \Sigma^*}$ with $H_L[x, y] = L(xy)$. It was shown by Carlyle and Paz [15] and Fliess [30] that the rank of H_L is equal to the number of states of the minimal (in number of states) $\mathbb{F}$-weighted automaton $\mathcal{A}$ with $L_{\mathcal{A}} = L$.

Proposition 2.3 ([15; 30])**.** *Let L be an $\mathbb{F}$-weighted regular language, i.e. a function $L : \Sigma^* \to \mathbb{F}$ that can be represented by an $\mathbb{F}$-weighted automaton. Let $\mathcal{A} = (Q, \Sigma, M, \alpha, \eta)$ be a minimal $\mathbb{F}$-weighted automaton such that $L_{\mathcal{A}} = L$. Then* $\operatorname{rank} H_L = |Q|$.

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 $\iota : S \to [0, 1] \subset \mathbb{R}$ 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, see e.g. [5, Chapter 10]. We may write $\Pr_\iota$ for $\Pr_\iota^{\mathcal{M}}$ if $\mathcal{M}$ is understood. If ι is a Dirac distribution on a state s, i.e. $\iota(s) = 1$, we may write $\Pr_s^{\mathcal{M}}$ for $\Pr_\iota^{\mathcal{M}}$. We write E for the set of edges in the graph of M. Note that $|S| \leq |E| \leq |S|^2$, as M is stochastic.

Given a regular language with positive probability according to the probability measure given by a Markov chain, we see that there exists a prefix such that the conditional probability is 1 of the language restricted to those words starting with that prefix, see for instance [7]:

Lemma 2.4. *Let $M = (S, P)$ be a Markov chain, and $\mathcal{L} \subseteq S^\omega$ an ω-regular language. Suppose $s_0 \in S$ such that $\Pr_{s_0}(\mathcal{L}) > 0$. Then there exists $s_0 \dots s_n \in Paths_{s_0}(P)$ such that $\Pr_{s_n}(\{w \in s_n S^\omega \mid s_0 \dots s_{n-1} w \in \mathcal{L}\}) = 1$.*

Branching processes. A *(multi-type) branching process (BP)* is a tuple $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$, where Γ is a finite set of types, $\hookrightarrow \subseteq \Gamma \times \Gamma^+$ is a finite set of transition rules, *Prob* is a function assigning positive rational probabilities to transition rules so that for every $X \in \Gamma$ we have $\sum_{X \hookrightarrow w} \mathit{Prob}(X \hookrightarrow w) = 1$, and $X_0 \in \Gamma$ is the start type. We write $X \overset{p}{\hookrightarrow} w$ to denote that $\mathit{Prob}(X \hookrightarrow w) = p$. Given a BP $\mathcal{B}$ and a type $X \in \Gamma$ we write $\mathcal{B}[X]$ for the BP obtained from $\mathcal{B}$ by making X the start type. For $X, Y \in \Gamma$ we call Y a *successor* of X if there is a rule $X \hookrightarrow uYv$ for some $u, v \in \Gamma^*$.

A BP *with ε-rules allowed* relaxes the requirement $\hookrightarrow \subseteq \Gamma \times \Gamma^+$ to $\hookrightarrow \subseteq \Gamma \times \Gamma^*$, i.e., there may be rules of the form $X \hookrightarrow \varepsilon$, where ε denotes the empty word. In the following, we disallow ε-rules unless specified otherwise; but the definitions generalise in a natural way.

Fix a BP $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ for the rest of the chapter.

Trees generated by branching processes. Write $[\![\mathcal{B}]\!]$ for the set of trees *generated* by $\mathcal{B}$; i.e., $[\![\mathcal{B}]\!]$ denotes the set of ordered Γ-labelled trees t such that for each $X \in \Gamma$ and each X-labelled node v in t, there is a rule $X \hookrightarrow X_1 \cdots X_k$, denoted by $\mathit{rule}(v)$, such that the k ordered children of v are labelled with $X_1, \ldots, X_k$, respectively. We say a node *has type* $X \in \Gamma$ if the node is labelled with X. A *finite prefix* of a tree $t \in [\![\mathcal{B}]\!]$ is an ordered Γ-labelled *finite* tree obtained from t by designating some nodes as leaves, and removing all their children, grandchildren, etc. Write $(\!|\mathcal{B}|\!)$ for the set of finite prefixes of trees generated by $\mathcal{B}$. For $t \in (\!|\mathcal{B}|\!)$ write $\mathit{Cylt} \subseteq [\![\mathcal{B}]\!]$ for the ("cylinder") set of trees $t' \in [\![\mathcal{B}]\!]$ such that t is a finite prefix of t'. For $X \in \Gamma$ write $[\![\mathcal{B}]\!]_X \subseteq [\![\mathcal{B}]\!]$ and $(\!|\mathcal{B}|\!)_X \subseteq (\!|\mathcal{B}|\!)$ for the subsets of trees whose root has type X; the trees in $[\![\mathcal{B}]\!]_X$ are called *X-trees*. A *branch* of a tree t is a sequence $v_0 v_1 \cdots$ of nodes in t, where v_0 is the root of t and v_{i+1} is a child of v_i for all $i \in \mathbb{N}_0$. See [17] for equivalent, more formal tree-related definitions.

Probability measure induced by branching processes. For each $X \in \Gamma$ we define the probability space $([\![\mathcal{B}]\!]_X, \Sigma_X, \mathbb{P}_X)$, where Σ_X is the σ-algebra generated by $\{\mathit{Cyl}(t) \mid t \in (\!|\mathcal{B}|\!)_X\}$, and $\mathbb{P}_X$ is the probability measure defined by $\mathbb{P}_X(\mathit{Cyl}(t)) := \prod_v \mathit{Prob}(\mathit{rule}(v))$ for all $t \in (\!|\mathcal{B}|\!)_X$, where the product extends over all non-leaf nodes v in t. This is analogous to the standard definition of the probability space of a Markov chain. We may write $\mathbb{P}_{\mathcal{B}}$ for $\mathbb{P}_{X_0}$, omitting the subscript when $\mathcal{B}$ is understood. We often talk about events (i.e., measurable sets of trees) and their probability in text form. For example, by saying "a $\mathcal{B}$-tree has with positive

probability infinitely many nodes of type X" we mean that $\mathbb{P}_{\mathcal{B}}(E) > 0$ where $E \subseteq [\![\mathcal{B}]\!]_{X_0}$ is the set of X_0-trees with infinitely many nodes of type X.

Alternating Turing machines An *alternating Turing machine* is a 6-tuple $(S_\exists, S_\forall, \Sigma, T, s_0, s_{acc})$, where $S = S_\exists \cup S_\forall \cup \{s_{acc}\}$ is a finite set of (control) states partitioned into existential states $S_\exists$ and universal states $S_\forall$ and the (only) accepting state s_{acc}, Σ is a finite alphabet, $T \subseteq (S_\exists \cup S_\forall) \times \Sigma \times \Sigma \times \{-1, +1\} \times S$ is a transition relation, and s_0 is the initial state. A transition $(s, a, a', D, s') \in T$ means that if M is in state s and its head reads letter a, then it rewrites the content of the current cell with the letter a', it moves the head in direction D (either left if $D = -1$, or right if $D = +1$), and it changes its state to s'. We assume that for all $s \in S_\exists \cup S_\forall$ and $a \in \Sigma$ there is at least one outgoing transition. A configuration of an alternating Turing machine is given by a 3-tuple (i, s, w) where $i \in \mathbb{N}$ indicates the header position, $s \in S$ is the current state of the Turing machine, and $w \in \Sigma^*$ is the contents of the memory tape.

For a word $w \in \Sigma^*$ we will write $w(i)$ to denote its i'th component, and $w[i = a]$ to denote the string $w(1) \dots w(i-1)aw(i+1) \dots w(n)$. The initial configuration of an alternating Turing machine on a word $w \in \Sigma^*$ is $(1, s_0, w)$, and given a transition $t = (s, a, a', D, s')$ and a configuration $c = (i, s'', w)$ we will use $c \triangleright t$ to denote the configuration $(i + D, s', w[i = a'])$ if $s'' = s$ and $w_i = a$ (and leave it undefined otherwise). We extend this notation to sequences of transitions $t_1 \dots t_n \in T^*$ by writing $c \triangleright t_1 \dots t_n = (c \triangleright t_1) \triangleright t_2 \dots t_n$. We will use $\pi_{\mathbb{N}}$, π_S, and π_{Σ^*} to denote the projections of a configuration (i, s, w) to i, s, and w, respectively.

For any $(s, a) \in (S_\exists \cup S_\forall) \times \Sigma$, let $T_{s,a} := \{(s, a, a', D, s') \in T \mid a' \in \Sigma,\ D \in \{-1, +1\},\ s' \in S\}$. A string $r = r_1 r_2 \dots \in T^* \cup T^\omega$ is called a *run* of M on w if for all i, $r_{i+1} \in T_{\pi_S(c_i), \pi_{\Sigma^*}(c_i)(\pi_{\mathbb{N}}(c_i))}$, where $c_i = c_0 \triangleright r_1 \dots r_i$ whenever r_{i+1} exists. Any run r is called an *accepting run* if $r \in T^*$ and $\pi_S(c_0 \triangleright r) = s_{acc}$. A strategy is a function $\sigma : \{(i, s, w, r) \in \mathbb{N} \times S \times \Sigma^* \times T^* \mid s \in S_\exists\} \to T$ such that $\sigma(i, s, w, r) \in T_{s,w_i}$ for any i, s, w, r. For a configuration c and a run r, by abuse of notation, we will write $\sigma(c, r)$ to mean $\sigma(\pi_{\mathbb{N}}(c), \pi_S(c), \pi_{\Sigma^*}(c), r)$. A run is *consistent* with a strategy σ if for all i, $r_{i+1} = \sigma(c_0 \triangleright r_1 \dots r_i, r_1 \dots r_i)$ whenever the latter is defined. The *behaviour* of σ is the set of runs consistent with σ. A *winning strategy* is a strategy σ such that every run r consistent with σ is an accepting run. Note that whenever σ is winning, its behaviour is finite.

Let σ be a strategy, then we call σ *N-bounded* if for any r in the behavior of σ, and any prefix $r_1 \dots r_k$ of r, $|\pi_{\Sigma^*}(c_0 \triangleright r_1 \dots r_k)| < N$. A Turing machine is N-bounded if it has an N-bounded winning strategy.

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 $f(x)$. We say that f is 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 polylogarithmic time) in parallel (i.e., with a polynomial number of processors) (see, e.g., [53, Chapter 15]).

A standard result of complexity theory is that a function computable in NC is computable by a Turing machine using polylogarithmic space [12, 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 [53].

For many of our PSPACE results we will use a polylogarithmic space algorithm on an exponentially sized input that can be computed using a PSPACE transducer. For this we need the following lemma:

Lemma 2.5. *Given two problems P_1, P_2, where P_2 is in NC. Suppose there is a reduction from P_1 to P_2 implemented by a PSPACE transducer, i.e., a Turing machine whose work tape (but not necessarily its output tape) is PSPACE-bounded. Then P_1 is in PSPACE.*

Proof. Note that the output of the transducer is (at most) exponential. Problems in NC can be decided in polylogarithmic space [12, Theorem 4]. Using standard techniques for composing space-bounded transducers (see, e.g., [53, Proposition 8.2]), it follows that P1 is in PSPACE. □

Vectors and matrices. We consider vectors and square matrices indexed by a finite set S. We write (column) vectors $\vec{v} \in \mathbb{R}^S$ with arrows on top, and $\vec{v}^\top$ 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. For a set $T \subseteq S$ we write $[\![T]\!] \in \{0,1\}^S$ for the characteristic vector of T, i.e., $[\![T]\!]_s = 1$ if $s \in T$ and $[\![T]\!]_s = 0$ otherwise. A matrix $M \in [0,1]^{S\times S}$ is called *(right-)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 non-negative 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 in M.

Solving linear systems. Let $\kappa \in [2,3]$ be such that one can multiply two $n \times n$-matrices in time $O(n^\kappa)$ (in other literature, κ is often denoted by ω). We assume that arithmetic operations cost constant time. One can choose $\kappa = 2.4$, see [48] for a recent result. One can check whether an $n \times n$ matrix is invertible in time $O(n^\kappa)$ [13]. Finally, one can solve a linear system with n equations using the Moore-Penrose pseudo-inverse [39] in time $O(n^\kappa)$ [55].

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 results summarise some facts in the spectral theory of non-negative matrices that will be used in this thesis.

Theorem 2.6. *Let $M \in \mathbb{R}^{S \times S}$ be a non-negative matrix. Then the following all hold:*

1. *The spectral radius $\rho(M)$ is an eigenvalue of M and there is a non-negative eigenvector $\vec{x}$ with $M\vec{x} = \rho(M)\vec{x}$. Such a vector $\vec{x}$ is called* dominant.
2. *If $0 \leq M' \leq M$ then $\rho(M') \leq \rho(M)$.*
3. *There is $C \subseteq S$ such that $M_{C,C}$ is strongly connected and $\rho(M_{C,C}) = \rho(M)$.*

Theorem 2.7. *Let $M \in \mathbb{R}^{S \times S}$ be a strongly connected non-negative matrix. Then the following all hold:*

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 $0 \leq M' < M$ (i.e., $M'_{i,j} < M_{i,j}$ for some $i,j \in S$) then $\rho(M') < \rho(M)$.*
4. *If $\vec{x} \geq 0$ and $M\vec{x} \leq \rho(M)\vec{x}$ then $M\vec{x} = \rho(M)\vec{x}$.*
5. *Any non-negative eigenvector $\vec{x}$ of M has eigenvalue $\rho(M)$.*

These results can all be found in [9, Chapter 2]. Specifically, Theorem 2.6.1, Theorem 2.7.1, and Theorem 2.7.2 are part of the Perron-Frobenius theorem, see [9, Theorems 2.1.1, 2.1.4]; Theorems 2.6.2 and 2.7.3 are [9, Corollary 2.1.5]; Theorem 2.6.3 follows from [9, Corollary 2.1.6(b)]; Theorem 2.7.4 follows from [9, Corollary 2.1.11]; Theorem 2.7.5 follows from [9, Theorem 2.1.4].

We will need to check whether powers of non-negative matrices grow, shrink, or stay stable.

Lemma 2.8. *Given a non-negative rational matrix M, one can determine in NC whether $\rho < 1$ or $\rho = 1$ or $\rho > 1$, where ρ denotes the spectral radius of M.*

Proof. Use the algorithm from [24, Proposition 2.2], but not with Gaussian elimination as suggested there, but by solving the systems of linear equations described in [24, Proposition 2.2] in NC. The latter is possible in NC [11, Theorem 5]. □

3

Model-checking UBAs against MCs using cuts

Contents

Many of the original results of this thesis are based on or inspired by the model checking procedure for unambiguous automata by Baier et al.[7]. For this reason, we dedicate Chapter 3 to explaining the results from this paper.

3.1 Introduction

Consider the question "Given a Markov chain inducing a probability measure Pr and an automaton $\mathcal{A}$ accepting a language L, what is the value of $\Pr(L)$?" This problem is PSPACE-complete [14; 20], but solvable in polynomial time if $\mathcal{A}$ is deterministic. Baier et al. give a polynomial time procedure for answering this question in the case where the automaton is unambiguous. The main idea is to exploit the unambiguity of the automaton in order to express the probability of a union as a sum of probabilities. This allows one to write the probability of the language given by the automaton starting in a fixed state q as the sum of the

probabilities of the automaton starting in the successor states of q after reading a single character from the alphabet. However, because we are considering automata over infinite words, these expressions alone are not sufficient to uniquely express the probability $\Pr(L)$. For this reason, certain normalising equations given by *normalisers* are required. Baier et al. give a combinatoric procedure using so-called *cut vectors* to compute normalisers.

The main theorem of this chapter, using the more detailed complexity analysis from [43], is the following:

Theorem 1.6. *Given a Markov chain $\mathcal{M} = (S, M)$ with edge set E, an initial distribution ι, and a UBA $\mathcal{A} = (Q, S, \delta, Q_0, F)$, one can compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ in time $O(|Q|^\kappa|S|^\kappa + |Q|^3|\delta||S| + |\delta|^2|E|)$, where $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ denotes the probability that $\mathcal{M}$ with initial distribution ι generates a word in the language $\mathcal{L}(\mathcal{A})$ of words accepted by $\mathcal{A}$.*

(Recall that the constant $\kappa \in [2, 3)$ is such that one can multiply two $n \times n$ matrices in $O(n^\kappa)$.) This chapter is structured as follows: in Section 3.2, the basic system of equations is given. Section 3.3 outlines the procedure for calculating normalisers based on cuts. The procedure is based on two technical lemmas, Lemma 3.8 and Lemma 3.13, which are proved in Section 3.4 and Section 3.5, respectively. In Section 3.6 we put it all together and conclude.

3.2 The System Of Equations

Let $\mathcal{M} = (S, M)$ be a Markov chain, ι an initial distribution, and $\mathcal{A} = (Q, S, \delta, Q_0, F)$ an automaton. Let $B \in \mathbb{R}^{(Q\times S)\times(Q\times S)}$ be the following matrix:

$$B_{\langle q,s\rangle,\langle q',s'\rangle} = \begin{cases} M_{s,s'} & \text{if } q' \in \delta(q,s) \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Define the vector $\vec{z} \in \mathbb{R}^{Q\times S}$ by $\vec{z}_{\langle q,s\rangle} = \Pr_s^{\mathcal{M}}(\mathcal{L}(\mathcal{A}_q))$, i.e., the probability of $\mathcal{A}$ starting in q of accepting a word generated by $\mathcal{M}$ starting in s.

Lemma 3.1 (Lemma 4, [7])**.** *The following equations hold:*

$$\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A})) = \sum_{s\in S} \iota(s) \sum_{q\in Q_0} \Pr_s^{\mathcal{M}}(\mathcal{L}(\mathcal{A}_q)) \tag{3.2}$$

$$\textit{for all } q \in Q \textit{ and } s \in S\textit{:}\ \Pr_s^{\mathcal{M}}(\mathcal{L}(\mathcal{A}_q)) = \sum_{s'\in S} \sum_{q'\in\delta(q,s)} M_{s,s'} \Pr_{s'}^{\mathcal{M}}(\mathcal{L}(\mathcal{A}_{q'})). \tag{3.3}$$

Hence, $\vec{z} = B\vec{z}$ and $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A})) = \sum_{q\in Q_0} \sum_{s\in S} \iota(s)\vec{z}_{q,s}$.

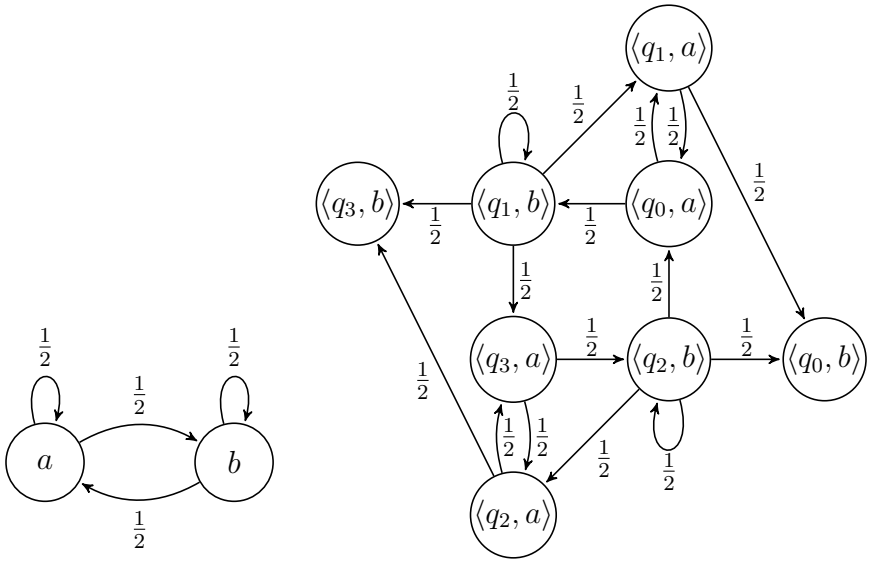


Figure 3.1: Markov chain $\mathcal{M}$, and its product, B, with the UBA from Figure 1.2

Proof. Since $\mathcal{L}(\mathcal{A}_q) \cap \mathcal{L}(\mathcal{A}_{q'}) = \emptyset$ for any $q, q' \in Q_0$ by unambiguity of $\mathcal{A}$, Equation (3.2) holds. For Equation (3.3), we have that for any $q', q'' \in \delta(q, s)$, $\mathcal{L}(\mathcal{A}_{q'}) \cap \mathcal{L}(\mathcal{A}_{q''}) = \emptyset$ by unambiguity. Hence,

$$\begin{aligned}\Pr_s^{\mathcal{M}}(\mathcal{L}(\mathcal{A}_q)) &= \Pr_s^{\mathcal{M}}(\mathcal{L}(\mathcal{A}_q) \cap sS^\omega) = \Pr_s^{\mathcal{M}}(s \bigcup_{q' \in \delta(q,s)} \mathcal{L}(\mathcal{A}_{q'})) \\ &= \sum_{s' \in S} M_{s,s'} \Pr_{s'}^{\mathcal{M}}(\bigcup_{q' \in \delta(q,s)} \mathcal{L}(\mathcal{A}_{q'})) = \sum_{s' \in S} \sum_{q' \in \delta(q,s)} M_{s,s'} \Pr_{s'}^{\mathcal{M}}(\mathcal{L}(\mathcal{A}_{q'})).\end{aligned}$$

□

Example 3.2. *Consider the UBA $\mathcal{A}$ from Figure 1.2 and the two-state Markov chain $\mathcal{M}$ shown on the left of Figure 3.1. The weighted graph on the right of Figure 3.1 represents the matrix B, obtained from $\mathcal{A}$ and $\mathcal{M}$ according to Equation* (3.1)*. 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) is not always one.*

Although $\vec{z}$ is a solution the system of equations $\vec{\zeta} = B\vec{\zeta}$, this system does not uniquely identify $\vec{z}$. Indeed, any scalar multiple of $\vec{z}$ is a solution for these equations. To uniquely identify $\vec{z}$ by a system of linear equations, we need to analyse the strongly connected components of B. To be precise, we are concerned with *recurrent* and *accepting* strongly connected components:

Definition 3.3. *Let $D \subseteq Q \times S$ be a strongly connected component of B. We then call D* recurrent *if the spectral radius of $B_{D,D}$ is 1. We call D* accepting *if $D \cap (Q \times S) \neq \emptyset$.*

For each accepting recurrent strongly connected component $D \subseteq Q \times S$, we need to add a normalising equation given by a *D-normaliser*:

Definition 3.4. *Let $D \subseteq Q \times S$ be an accepting recurrent strongly connected component of B. We call a vector $\vec{\mu} \in \mathbb{R}^{Q \times S}$ a D-*normaliser *if $\vec{\mu}_D^\top \vec{z}_D = 1$.*

The powers of (submatrices of) B allow for a probabilistic interpretation:

Lemma 3.5 (Lemma 6, [7])**.** *Let $C \subseteq Q \times S$ and $\langle q, s\rangle, \langle q', s'\rangle \in C$. Let $n \in \mathbb{N}$. Define $A := B_{C,C}$ and*

$$E^{C,n}_{\langle q,s\rangle,\langle q',s'\rangle} := \left\{ s_0 s_1 \ldots \in sS^\omega \mid \exists q_0 q_1 \ldots q_n \,.\, \langle q_0, s_0\rangle \ldots \langle q_n, s_n\rangle \in Paths_{\langle q,s\rangle,\langle q',s'\rangle}(A) \right\}.$$

Then $(A^n)_{\langle q,s\rangle,\langle q',s'\rangle} = \Pr_s(E^{C,n}_{\langle q,s\rangle,\langle q',s'\rangle})$. In particular,

$$(B^n)_{\langle q,s\rangle,\langle q',s'\rangle} = \Pr_s(\{s_0 s_1 \ldots \in sS^\omega \mid q' \in \delta(q, s_0 \ldots s_{n-1}), s_n = s'\}).$$

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

$$\Pr_s(s_0 \ldots s_n S^\omega) = M_{s_0,s_1} \cdot \ldots \cdot M_{s_{n-1},s_n} = A_{s_0,s_1} \cdot \cdots \cdot \ldots \cdot A_{s_{n-1},s_n}.$$

The proposition follows. □

Unambiguity and finite ambiguity often imply spectral properties that will be used throughout this thesis. The following corollary is an example of that, showing that strongly connected components of B have a spectral radius of at most one:

Corollary 3.6. *Let $D \subseteq Q \times S$ be a strongly connected component of B. Then $\rho(B_{D,D}) \leq 1$.*

Proof. By Lemma 3.5. □

Example 3.7. *The matrix B from Figure 3.1 has three strongly connected components, namely the two singleton sets $\{\langle q_0, b\rangle\}$ and $\{\langle q_3, b\rangle\}$, and $D = \{\langle q_0, a\rangle, \langle q_1, a\rangle, \langle q_1, b\rangle, \langle q_2, a\rangle, \langle q_2, b\rangle, \langle q_3, a\rangle\}$. Only D is recurrent; indeed, $\vec{y} = (\vec{y}_{\langle q_0,a\rangle}, \vec{y}_{\langle q_1,a\rangle}, \vec{y}_{\langle q_1,b\rangle}, \vec{y}_{\langle q_2,a\rangle}, \vec{y}_{\langle q_2,b\rangle}, \vec{y}_{\langle q_3,a\rangle})^\top = (2, 1, 3, 1, 3, 2)^\top$ is a dominant eigenvector with $B_{D,D}\vec{y} = \vec{y}$. Since q_0 is accepting, D is accepting recurrent.*

In order to prove the existence of D-normalisers, and to show that D-normalisers are sufficient to uniquely identify $\vec{z}$, we will need the following technical lemma:

Lemma 3.8. *[Lemma 8, [7]] Let $D \subseteq Q \times S$ be a recurrent strongly connected component.*

1. *For every $\vec{x}$ such that $\vec{x} = B\vec{x}$, we have that $\vec{x}_D = B_{D,D}\vec{x}_D$.*
2. *For every $d \in D$ we have that $\vec{z}_d > 0$ iff D is accepting.*

The proof can be found in Section 3.4.

Denote the set of accepting recurrent strongly connected components by $\mathcal{D}_+$ and the set of non-accepting recurrent strongly connected components by $\mathcal{D}_0$. By Lemma 3.8, for $D \in \mathcal{D}_+$ we have $\vec{z}_d > 0$ for all $d \in D$, and for $D \in \mathcal{D}_0$ we have $\vec{z}_D = \vec{0}$. Hence, for $D \in \mathcal{D}_+$, there exists a *D-normaliser*, i.e., a vector $\vec{\mu} \in \mathbb{R}^D$ such that $\vec{\mu}^\top \vec{z}_D = 1$. Using D-normalisers allows us to set up a system of linear equations that uniquely identifies $\vec{z}$:

Lemma 3.9 (Lemma 12, [7])**.** *Let $\mathcal{D}_+$ be the set of accepting recurrent strongly connected components, and $\mathcal{D}_0$ the set of non-accepting recurrent strongly connected components. For each $D \in \mathcal{D}_+$ let $\vec{\mu}_D$ be a D-normaliser. Then $\vec{z}$ is the unique solution of the following linear system:*

$$\begin{aligned} \vec{\zeta} &= B\vec{\zeta} \\ \text{for all } D \in \mathcal{D}_+ : \quad \vec{\mu}_D^\top \vec{\zeta}_D &= 1 \\ \text{for all } D \in \mathcal{D}_0 : \quad \vec{\zeta}_D &= \vec{0} \end{aligned} \tag{3.4}$$

Proof. The vector $\vec{z}$ solves Equation (3.4) by Lemma 3.1 and Lemma 3.8 and the definition of a D-normaliser. It remains to show uniqueness. Let $\vec{x}$ solve Equation (3.4), we will then show that $\vec{x} = \vec{z}$:

- Let $D \in \mathcal{D}_0$. Then, $\vec{x}_D = \vec{0} = \vec{z}_D$.
- Let $D \in \mathcal{D}_+$. By Lemma 3.8, we have $\vec{x}_D = B_{D,D}\vec{x}_D$ and $\vec{z}_D = B_{D,D}\vec{z}_D$. By the Perron-Frobenius theorem (Theorem 2.7.2) and since D is strongly connected, the eigenspace corresponding to eigenvalue 1 of $B_{D,D}$ is one-dimensional, and since $\mu_D\vec{x}_D = 1 = \mu_D\vec{z}_D$ we have that $\vec{x}_D = \vec{z}_D$.
- Let $D \in \bigcup \mathcal{D}_+ \cup \bigcup \mathcal{D}_0$ be the union of all recurrent strongly connected components and let $\bar{D} = (Q \times S) \setminus D$. We have $\vec{x} - \vec{z} = B(\vec{x} - \vec{z})$. It

follows

$$\begin{aligned}\vec{x}_{\bar{D}} - \vec{z}_{\bar{D}} &= B_{\bar{D},\bar{D}}(\vec{x}_{\bar{D}} - \vec{z}_{\bar{D}}) + B_{\bar{D},D}(\vec{x}_D - \vec{z}_D) \\ &= B_{\bar{D},\bar{D}}(\vec{x}_{\bar{D}} - \vec{z}_{\bar{D}})\end{aligned}$$

by the previous two items. By definition of $\bar{D}$, for any strongly connected component C in $B_{\bar{D},\bar{D}}$ we have that $\rho(C) < 1$. By Theorem 2.6.3, this implies that $\rho(B_{\bar{D},\bar{D}}) < 1$ and hence $\vec{x}_{\bar{D}} - \vec{z}_{\bar{D}} = \vec{0}$. It follows that $\vec{x}_{\bar{D}} = \vec{z}_{\bar{D}}$ and $\vec{x} = \vec{z}$.

□

This leads to the following result:

Proposition 3.10. *Suppose N is the runtime of an algorithm to calculate a normaliser for each accepting recurrent strongly connected component. Then one can compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ in time $O(|Q|^\kappa|S|^\kappa) + N$.*

Proof. Lemma 3.9 implies correctness of the following procedure to calculate $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$:

1. Set up the matrix B from Equation (3.1).
2. Compute the strongly connected components of B.
3. For each strongly connected component C, check whether C is recurrent.
4. For each accepting recurrent strongly connected component D, compute its D-normaliser $\vec{\mu}(D)$.
5. Compute $\vec{z}$ by solving the linear system (3.4) in Lemma 3.9.
6. Compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A})) = \sum_{s\in S}\sum_{q\in Q_0}\iota(s)\vec{z}_{q,s}$.

One can set up B in time $O(|Q|^2|S|^2)$. Using Tarjan's algorithm one can compute the strongly connected components of B in time linear in the vertices and edges of B, hence in $O(|Q|^2|S|^2)$. One can find those strongly connected components D which are recurrent in time $O(|Q|^\kappa|S|^\kappa)$ by checking if $I - B_{D,D}$ is invertible. The linear system (3.4) has $O(|Q||S|)$ equations, and thus can be solved in time $O(|Q|^\kappa|S|^\kappa)$. Hence the total runtime is $O(|Q|^\kappa|S|^\kappa) + N$. □

3.3 Calculating D-Normalisers Using Cuts

We now have to find a way to calculate D-normalisers. In this chapter, we will explain the *cut*-based approach from [7], and give a detailed complexity analysis of this approach.

For the remainder of this section, let D be an accepting recurrent strongly connected component.

Definition 3.11. *A* fibre over $s \in S$ *is a subset of* D *of the form* $\alpha \times \{s\}$ *for some* $\alpha \subseteq Q$.

Given a fibre $f = \alpha \times \{s\}$ and a state $s' \in S$, if $M_{s,s'} > 0$ we define the fibre $f \triangleright s'$ as follows:

$$f \triangleright s' := \{\langle q, s' \rangle \mid q \in \delta(\alpha, s)\} \cap D.$$

If $M_{s,s'} = 0$, then $f \triangleright s'$ is undefined, and for $w \in S^*$ we define $f \triangleright w = f$ if $w = \varepsilon$ and $f \triangleright ws' = (f \triangleright w) \triangleright s'$. If $f = \{d\}$ for some $d \in D$ we may write $d \triangleright s'$ for $f \triangleright s'$.

Definition 3.12. *We call a fibre* c *a* cut *if* $c = d \triangleright v$ *for some* $v \in S^*$ *and* $d \in D$, *and* $c \triangleright w \neq \emptyset$ *for all* $w \in S^*$ *whenever* $c \triangleright w$ *is defined. Given a cut* $c \subseteq D$ *we use the term* cut vector *to describe its characteristic vector* $[\![c]\!] \in \{0,1\}^D$, *i.e., the vector* $[\![c]\!]$ *such that* $[\![c]\!]_d = 1$ *whenever* $d \in c$ *and* $[\![c]\!]_d = 0$ *otherwise.*

Note that if c is a cut then so is $c \triangleright w$ whenever it is defined.

Lemma 3.13. *[Lemma 10, [7]] Let* $D \subseteq Q \times S$ *be a strongly connected component, then:*

1. *D is recurrent if and only if it has a cut.*
2. *Any cut vector $\vec{\mu}$ is a normaliser, i.e., $\vec{\mu}^\top \vec{z}_D = 1$.*

The proof of this lemma is given in Section 3.5. The following lemma is the basis for the cut computation algorithm in [6; 7]:

Lemma 3.14 (Lemma 17, [7])**.** *Let* $D \subseteq Q \times S$ *be a recurrent strongly connected component. 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* $d' \triangleright w \neq \emptyset$. *For any such* d', $d \triangleright w \cap d' \triangleright w = \emptyset$. *Hence* $d \triangleright vw \supseteq \{d, d'\} \triangleright w \supsetneq d \triangleright w$.

Proof. Since D is recurrent, by Lemma 3.13 it has a cut, say $\alpha = d_1 \triangleright v_1$. Since D is an strongly connected component, there exists v_0 with $d_1 \in d \triangleright v_0$. Hence, $d \triangleright v_0 v_1 \supseteq \alpha$ is also a cut. Again, since D is an strongly connected component, there exists v_2 with $d \in d \triangleright v_0 v_1 v_2$. Define $v := v_0 v_1 v_2$. Since $d \triangleright v_0 v_1$ is a cut, so is $d \triangleright v$. Moreover, we have $d \triangleright vw \supseteq d \triangleright w \ni d$. Since $d \triangleright vw$ is a cut and $d \triangleright w$ is not, $d \triangleright vw \supsetneq d \triangleright w$. Therefore there exists $d' \in d \triangleright vw$ with $d \neq d'$ and $d' \triangleright w \neq \emptyset$. □

This suggests a way of generating an increasing sequence of fibres, culminating in a cut. Define, for some $d = \langle q, s \rangle \in D$, its *co-reachability* set $Co(d) \subseteq D$: it consists of those $d' \in D$ such that there exists a word w with $\{d, d'\} \subseteq d \triangleright w$. We can use co-reachability sets in order to efficiently calculate this above-mentioned increasing sequence of fibres. Note that $Co(d)$ is a fibre over s. The following lemma establishes a bound on the time to compute $Co(d)$:

Lemma 3.15. *Let $D \subseteq Q \times S$ be a recurrent strongly connected component. Denote by T the set of edges in $B_{D,D}$. One can compute $Co(d)$ in time $O(|Q||D| + |\delta||T|)$. Moreover, one can compute in time $O(|Q||D| + |\delta||T|)$ a list $(CoPath(d)(d'))_{d' \in Co(d)}$ such that $CoPath(d)(d') \in S^*$ and $\{d, d'\} \subseteq d \triangleright CoPath(d)(d')$ and $|CoPath(d)(d')| \leq |Q||D|$.*

Proof. Let $P = (V, A)$ denote the graph with vertices

$$V = \{(q, q', s) \in Q \times Q \times S \mid (q, s), (q', s) \in D\}$$

and edges

$$A = \left\{((q, q', s), (r, r', t)) \in V \times V \mid (B_{D,D})_{\langle q,s\rangle,\langle r,t\rangle} > 0, (B_{D,D})_{\langle q',s\rangle,\langle r',t\rangle} > 0\right\},$$

i.e. there is an edge from (q, q', s) to (r, r', t) in P iff there are edges from $\langle q, s\rangle$ to $\langle r, t\rangle$ and from $\langle q', s\rangle$ to $\langle r', t\rangle$ in $B_{D,D}$. We have that $|V| \leq |Q||D|$ and $|A| \leq |\delta||T|$.

For any $d = \langle q, s\rangle \in D$, note that any state $(r, r', t) \in V$ is reachable from $(q, q, s) \in V$ iff there exists a word $s_0 \dots s_n$ such that $s_n = t$, $\langle r, t\rangle \in d \triangleright s_0 \dots s_n$, and $\langle r', t\rangle \in d \triangleright s_0 \dots s_n$. Hence, any $d' = \langle q', s\rangle \in D$ is in $Co(d)$ iff $(q, q', s) \in V$ is reachable from (q, q, s). Hence, by doing breadth-first search in P, one can compute $Co(d)$ and record $CoPath(d)(d')_{d' \in D}$ in $O(|Q||D| + |\delta||T|)$. □

We use Lemma 3.15 to obtain an efficient method for finding cuts:

Proposition 3.16. *Let $D \subseteq Q \times S$ be a recurrent strongly connected component. Denote by T the set of edges in $B_{D,D}$. One can compute a cut in time $O(|Q|^2|\delta||D| + |\delta||T|)$.*

Proof sketch of Proposition 3.16. Starting from a singleton fibre $\{d\}$, where $d = \langle q, s\rangle \in D$ is chosen arbitrarily, we keep looking for words $v \in S^*$ that have the properties described in Lemma 3.14 to generate larger fibres $d \triangleright w$:

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

By Lemma 3.14, the algorithm returns a cut. In every loop iteration the fibre $d \triangleright w$ increases, so the loop terminates after at most $|Q|$ iterations. For efficiency we calculate $Co(d)$ and $CoPath(d)$ using Lemma 3.15, and we use dynamic programming to maintain the set, *Survives*, of those $d' \in D$ for which $d' \triangleright w \neq \emptyset$ holds. Whenever a prefix v is added to w, we update *Survives* by processing v backwards. This leads to the following algorithm:

1. Calculate $Co(d)$ and $CoPath(d)$ using Lemma 3.15
2. $w := \varepsilon$; $\mathit{Survives} := (Q \times \{s\}) \cap D$
3. while $\exists\, d' \in Co(d) \setminus \{d\}$ such that $d' \in \mathit{Survives}$:
   $v_0 = s$; $v_1 \dots v_n := CoPath(d)(d')$
   for $i = n, n-1, \dots, 1$:
   $\mathit{Survives} := \{\langle p, v_{i-1}\rangle \in D \mid (\delta(p, v_{i-1}) \times \{v_i\}) \cap \mathit{Survives} \neq \emptyset\}$
   $w := v_1 \dots v_n w$
4. return $d \triangleright w$

Step 1 of the algorithm computes $Co(d)$ and $CoPath(d)$ in time $O(|Q||D| + |\delta||T|)$ using Lemma 3.15. We have $|CoPath(d)(d')| \leq |Q||D|$, so the (final) word w has length at most $|Q|^2|D|$. For each state v_{i-1} in w one can update the set *Survives* in the inner loop in time $O(|\delta|)$ by backwards traversing the transitions labelled by v_{i-1}. Hence step 3 of the algorithm takes time $O(|Q|^2|\delta||D|)$. Finally calculating $d \triangleright w$ can also be done in time $O(|Q|^2|\delta||D|)$. The total runtime is $O(|Q|^2|\delta||D| + |\delta||T|)$. □

Example 3.17. *We will calculate a cut in the example from Figure 3.1. Letting* $d = \langle q_0, a\rangle$ *and* $d' = \langle q_2, a\rangle$ *we have* $Co(d) = \{d, d'\}$ *with* $CoPath(d)(d) = \varepsilon$ *and* $CoPath(d)(d') = baa$. *Initially we have* $Survives = Q \times \{a\}$. *In the first iteration the algorithm can only pick* d'. *The inner loop updates Survives first to*

$\{q_0, q_1, q_2, q_3\} \times \{a\}$ *(i.e., to itself), then to* $\{q_1, q_2\} \times \{b\}$*, and finally to* $\{q_0, q_3\} \times \{a\}$*. Now* $(Co(d) \setminus d) \cap Survives$ *is empty and the loop terminates. The algorithm returns the cut* $d \triangleright baa = \{d, d'\}$*.*

The general procedure in Proposition 3.10 parameterises the model checking algorithm in terms of the complexity of calculating normalisers. Plugging in the complexity from Proposition 3.16 for the cut-based calculation of normalisers leads to the following result:

Theorem 1.6. *Given a Markov chain* $\mathcal{M} = (S, M)$ *with edge set* E*, an initial distribution* ι*, and a UBA* $\mathcal{A} = (Q, S, \delta, Q_0, F)$*, one can compute* $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ *in time* $O(|Q|^\kappa |S|^\kappa + |Q|^3 |\delta| |S| + |\delta|^2 |E|)$*, where* $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ *denotes the probability that* $\mathcal{M}$ *with initial distribution* ι *generates a word in the language* $\mathcal{L}(\mathcal{A})$ *of words accepted by* $\mathcal{A}$*.*

In [7], Baier et al. also give a procedure to compute normalisers in NC, resulting in the following:

Theorem 3.18 (Theorem 19, [7])**.** *Given a Markov chain* $\mathcal{M} = (S, M)$*, an initial distribution* ι*, and a UBA* $\mathcal{A} = (Q, S, \delta, Q_0, F)$*, one can compute* $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ *in NC.*

3.4 Proof of Lemma 3.8

We have the following:

Lemma 3.8. *[Lemma 8, [7]] Let* $D \subseteq Q \times S$ *be a recurrent strongly connected component.*

1. *For every* $\vec{x}$ *such that* $\vec{x} = B\vec{x}$*, we have that* $\vec{x}_D = B_{D,D}\vec{x}_D$*.*
2. *For every* $d \in D$ *we have that* $\vec{z}_d > 0$ *iff* D *is accepting.*

Proof of Lemma 3.8.1. If $\vec{x_D} = \vec{0}$, then clearly $\vec{x_D} = B\vec{x_D}$. Otherwise, observe that $\vec{x_D} = (B\vec{x})_D \geq B_{D,D}\vec{x}_D$. Since $\rho(B_{D,D}) = 1$ and D is strongly connected, it follows by the Perron-Frobenius theorem that $\vec{x_D} = B_{D,D}\vec{x}_D$. □

The proof of Lemma 3.8.2 requires some auxiliary definitions and results. First we will show that for any strongly connected component C reachable from a recurrent strongly connected component D, $\vec{z_C} = \vec{0}$. Then we will show that for any recurrent strongly connected component D with positive probability, the

Markov chain generates a word for which there exists a path that stays in D, but almost surely this path visits every state in D infinitely often. Using these facts, we will prove Lemma 3.8.2.

Let $C, D \subseteq Q \times S$ be two strongly connected components of matrix B. We write $C \preceq D$ if C is reachable from D. We have the following result:

Lemma 3.19. *Let $D \subseteq Q \times S$ be a recurrent strongly connected component. Then all strongly connected components $C \prec D$ are such that $\vec{z_C} = \vec{0}$.*

Proof. By Lemma 3.8.1 we have $\vec{z_D} = B_{D,D}\vec{z_D}$. Since $\vec{z_D} = (B\vec{z})_D$, 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 $\vec{z_c} = 0$. Because $\vec{z_c} = \sum B_{c,c'}\vec{z_{c'}}$, we also have $\vec{z_{c'}} = 0$ whenever $B_{c,c'} \neq 0$ for any c'. It follows that $\vec{z_C} = \vec{0}$ for all strongly connected components C directly below D and hence $\vec{z_C} = \vec{0}$ for all strongly connected components $C \prec D$. □

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 $\pi_Q : (Q \times S)* \to Q^*$.

Lemma 3.20. *Let D be a recurrent strongly connected component, let $C \subseteq D$ be strongly connected, and let $c = \langle qs \rangle \in C$. Define $A := B_{C,C}$, and write*

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

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

(Intuitively, the probability of generating a word such that there exists a path starting from c staying in C is greater than 0 if and only if $C = D$.)

Proof. Let $\vec{x} \geq 0$ be a dominant eigenvector of A and write $\vec{x}_{\min}$ and $\vec{x}_{\max}$ for the respective minimum and maximum entries of $\vec{x}$. Since C is strongly connected, we have $\vec{x}_{\min} > 0$.

For $n \in \mathbb{N}$ and $d \in D$, recall the notation $E_{c,d}^{C,n}$ from Lemma 3.5:

$$E_{\langle q,s\rangle,\langle q',s'\rangle}^{C,n} := \left\{ s_0 s_1 \ldots \in sS^\omega \mid \exists q_0 q_1 \ldots q_n \,.\, \langle q_0, s_0\rangle \ldots \langle q_n, s_n\rangle \in Paths_{\langle q,s\rangle,\langle q',s'\rangle}(A) \right\}.$$

Write also $E_c^{C,n} := \bigcup_{d \in C} E_{c,d}^{C,n}$. Then $E_c^C = \bigcap_{n\in\mathbb{N}} E_c^{C,n}$ by König's Lemma - inclusion of E_c^C in $\bigcap_{n\in\mathbb{N}} E_c^{C,n}$ is obvious, and for the other direction König's Lemma shows that there are infinite paths in A for each word in $\bigcap_{n\in\mathbb{N}} E_c^{C,n}$, and hence each such word is in $E_c^{C,n}$. We have for any $d \in C$:

$$(A^n)_{c,d} \overset{Lemma\ 3.5}{=} \Pr_s(E_{c,d}^{C,n}) \leq \Pr_s(E_c^{C,n}) \leq \sum_{d' \in C} \Pr_s(E_{c,d'}^{C,n}) = \sum_{d' \in C} (A^n)_{c,d'}. \quad (3.5)$$

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

$$\begin{aligned} |C|\Pr_s(E_c^{C,n}) &\geq \sum_{d\in C}(A^n)_{c,d} && \text{by Equation (3.5)} \\ &= (A^n\vec{1})_c \\ &\geq \frac{1}{\vec{x}_{\max}}(A^n\vec{x})_c \\ &= \frac{\vec{x}_c}{\vec{x}_{\max}}. \end{aligned}$$

Since $E_c^C = E_c^{C,n}$ and by continuity of measures it follows that $\Pr_s(E_c^C) > 0$.

Conversely, suppose that $C \neq D$. Then by Theorem 2.7.3, $\rho(A) < 1$. We have

$$\begin{aligned} \Pr_s(E_c^{C,n}) &\leq \sum_{d\in C}(A^n)_{c,d} && \text{by Equation (3.5)} \\ &= (A^n\vec{1})_c \\ &\leq \frac{1}{\vec{x}_{\min}}(A^n\vec{x})_c \\ &= \rho(A)^n\frac{\vec{x}_c}{\vec{x}_{\min}}. \end{aligned}$$

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

Lemma 3.21. *Let D be a recurrent strongly connected component. Write*

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

where the second union ranges over strongly connected subsets of D. Then $\Pr_s(E^{<D}) = 0$ for all $s \in S$.

(Intuitively, the probability of generating a word in the Markov chain for which there exists a path that eventually stays in a strict subset of D, is 0.)

Proof. Let $s \in S$ and $u \in S^*$, and let $C \subsetneq D$ be strongly connected and $\langle q', s'\rangle = d \in C$. Then $\Pr_s(uE_d^C) \leq \Pr_{s'}(E_d^C) = 0$ by Lemma 3.20. Since $E^{<D}$ is a countable union of sets of measure 0 w.r.t. $\Pr_s$, it follows that $\Pr_s(E^{<D}) = 0$. □

Now we have everything in place to prove Lemma 3.8.2:

Proof of Lemma 3.8.2. Let $d = \langle q, s\rangle \in D$. We show that $\vec{z}_d > 0$ iff D is accepting.

Suppose that D is accepting. Recall the definitions of E_d^C and $E^{<D}$ from above. Since D is accepting we have $E_d^D \setminus E^{<D} \subseteq \mathcal{L}(\mathcal{A}_q)$. Hence

$$\begin{aligned} \vec{z}_d &= \Pr_s(\mathcal{L}(\mathcal{A})_q) \\ &\geq \Pr_s(E_d^D \setminus E^{<D}) \\ &\geq \Pr_s(E_d^D) - \Pr_s(E^{<D}) \\ &> 0 && \text{by Lemmas 3.20 and 3.21.} \end{aligned}$$

Suppose now that D is not accepting. Then any accepted word must have an accepted path outside D. Hence we have

$$\mathcal{L}(\mathcal{A}_q) \subseteq \bigcup_{u \in S^*} \bigcup_{C \prec D} \bigcup_{\langle q', s' \rangle \in C} u\mathcal{L}(\mathcal{A}_{q'}).$$

However, by Lemma 3.19, $\mathcal{L}(\mathcal{A}_{q'})$ has measure 0 w.r.t. $\Pr_{s'}$. Hence, $\mathcal{L}(\mathcal{A}_q)$ is a subset of a countable union of sets with measure 0, and thus $\vec{z}_d = \Pr_s(\mathcal{L}(\mathcal{A}_q)) = 0$. □

3.5 Proof of Lemma 3.13

We have the following:

Lemma 3.13. *[Lemma 10, [7]] Let $D \subseteq Q \times S$ be a strongly connected component, then:*

1. *D is recurrent if and only if it has a cut.*
2. *Any cut vector $\vec{\mu}$ is a normaliser, i.e., $\vec{\mu}^\top \vec{z}_D = 1$.*

Proof of Lemma 3.13.1. Let $D \subseteq Q \times S$ be a strongly connected component. We have for all $\langle q, s \rangle, \langle q', s' \rangle \in D$:

$$\Pr_s(E^{D,n}_{\langle q,s \rangle, \langle q', s' \rangle}) = \Pr(\{s_0 s_1 \ldots \in Paths^\omega_s(M) \mid \langle q', s' \rangle \in \langle q, s \rangle \triangleright s_1 \ldots s_n\}). \quad (3.6)$$

Define a matrix $A := B_{D,D}$. By Theorem 2.7.1, A has a dominant eigenvector $\vec{x}$, positive in all entries, with $A\vec{x} = \rho(A)\vec{x}$. Write $\vec{x}_{\min} > 0$ 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 $X^{d_0}_n : Paths^\omega_{s_0}(M) \to \mathbb{R}_{\geq 0}$ by

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

Note that $X^{d_0}_n(s_0 s_1 \ldots)$ is nonnegative since $\vec{x}$ is positive in all entries, and $X^{d_0}_n(s_0 s_1 \ldots)$ is zero only if the sum is over an empty set. We have:

$$\mathbb{E}_{s_0}(X^{d_0}_n) \overset{(3.6)}{=} \sum_{d \in D} \Pr_{s_0}(E^{D,n}_{d_0,d})\vec{x}_d \overset{Lemma\ 3.5}{=} (A^n \vec{x})_{d_0} = \rho(A)^n \vec{x}_{d_0}, \quad (3.7)$$

where $\mathbb{E}_{s_0}$ denotes expectation with respect to $\Pr_{s_0}$.

Towards the direction "$\Longleftarrow$", let $d_0 \triangleright s_1 \ldots s_k$ be a cut, where $d_0 = \langle q_0, s_0 \rangle$. Then we have for all $n \geq k$, using Markov's inequality:

$$\begin{aligned} 0 < \Pr_{s_0}(s_0 \ldots s_k S^\omega)\vec{x}_{\min} &\leq \Pr_{s_0}(X^{d_0}_n \geq \vec{x}_{\min})\vec{x}_{\min} \\ &\leq \mathbb{E}_{s_0}(X^{d_0}_n) \overset{(3.7)}{=} \rho(A)^n \vec{x}_{d_0}. \end{aligned}$$

This gives a uniform lower bound on $\rho(A)^n$ for all $n \geq k$ and hence $\rho(A) \geq 1$ and 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 $d_0 \triangleright vw = \emptyset$. It follows that there are $l \in \mathbb{N}$ and $y > 0$ such that for all $s_0 \dots s_n \in Paths_{s_0}(M)$:

$$\Pr_{s_0}(0 = X^{d_0}_{n+l} = X^{d_0}_{n+l+1} = \dots \mid s_0 \dots s_n S^\omega) \geq y.$$

Thus, $X^{d_0}_0, X^{d_0}_1, \dots$ converges to 0 almost surely with respect to $\Pr_{s_0}$. Since $X^{d_0}_0, X^{d_0}_1, \dots$ is bounded (by $\sum_{d \in D} \vec{x}_d$), it follows that $\lim_{n\to\infty} \mathbb{E}_{s_0}(X^{d_0}_n) = 0$. By Equation (3.7), we conclude $\rho(A) < 1$, i.e., D is not recurrent. □

Proof of Lemma 3.13.2. Let D be a recurrent strongly connected component, and let $\alpha = \alpha' \times \{s_n\} = \langle q_0, s_0 \rangle \triangleright s_1 \dots s_n \subseteq D$ be a cut. Recall that $\alpha' \subseteq \delta(q_0, s_0 \dots s_n)$. Let $\vec{\mu} \in \{0,1\}^D$ be the cut vector associated with α. Then we have:

$$\begin{aligned}\vec{\mu}^T \vec{z}_D = \sum_{d \in \alpha} \vec{z}_d = \sum_{q \in \alpha'} \Pr_{s_n}(\mathcal{L}(\mathcal{A}_q)) &\leq \sum_{q \in \delta(q_0, s_0 \dots s_n)} \Pr_{s_n}(\mathcal{L}(\mathcal{A}_q)) \\ &= \Pr_{s_n}(\mathcal{L}(\mathcal{A}[\delta(q_0, s_0 \dots s_n)])),\end{aligned}$$

where the last equality holds as the sets $\mathcal{L}(\mathcal{A}_q)$ are disjoint by unambiguousness. Hence, $\vec{\mu}^T \vec{z} \leq 1$. Suppose $\vec{\mu}^T \vec{z} < 1$. Then $\Pr_{s_n}(\mathcal{L}(\mathcal{A}[\delta(q_0, s_0 \dots s_n)])) < 1$ and by Lemma 2.4 there exists $s_n \dots s_m \in Paths_{s_n}(M)$ such that

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

Equivalently,

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

But since α is a cut, we have $\langle q_0, s_0 \rangle \triangleright s_1 \dots s_m \neq \emptyset$, i.e. there exists $q \in \delta(q_0, s_0 \dots s_{m-1})$ with $\langle q, s_m \rangle \in D$. By the above, we have $\Pr_{s_m}(\mathcal{L}(\mathcal{A}_q)) = 0$, and hence by Lemma 3.8.2 we have that D is not accepting. □

3.6 Conclusion

In this chapter we looked at the procedure for model checking Markov chains against unambiguous Büchi automata by Baier et al. They show that the model checking problem can be solved by expressing $\Pr^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ as the solution of a system of equations, consisting of a basic system of equations combined with normalisers. In Proposition 3.10 we show that the complexity of solving this system of equations is

given by $O(|Q|^\kappa|S|^\kappa) + N$, where N is the complexity of calculating normalisers. Their algorithm for computing normalisers using cuts is given in Section 3.3, where in Proposition 3.16 we show that by using this algorithm one can calculate normalisers in $O(|Q|^3|\delta||S| + |\delta|^2|E|)$. Finally, this results in Theorem 1.6, a model checking algorithm for Markov chains against unambiguous automata that runs in $O(|Q|^5)$.

4

Model checking UBAs against MCs using pseudo-cuts

Contents

4.1 Introduction

This chapter is based on [43]. In this paper we improve on the results from Baier et al.[7] by replacing their cut based procedure for calculating normalisers with a procedure for computing *pseudo-cuts*. Pseudo-cuts are vectors that are orthogonal to the matrices in the semigroup induced by the transition matrices from the automaton. A method of computing such orthogonal vectors was also independently devised by Protasov and Voynov[57] in their paper on matrix semigroups with constant spectral radius. Our results adapt this method for the case where the underlying graph of the Markov chain is not a complete graph (and hence where not all elements of the matrix semigroup are valid), and apply it in the context of model checking automata against Markov chains. Ultimately this results in a speedup in terms of the size of the automaton, at a cost in terms of the size of the Markov chain. At several places throughout this thesis we will base our results on

the cut-based model checking algorithm - often a speedup can be achieved there by using the pseudo-cut algorithm described in this chapter when the automaton is big.

4.2 Calculating D-Normalisers Using Linear Algebra

Recall from Chapter 3 the definition of the product of the automaton and Markov chain denoted by $B \in \mathbb{R}^{(Q\times S)\times(Q\times S)}$. Let $D \subseteq Q \times S$ be a strongly connected component of B. For $t \in S$ define the matrix $\Delta(t) \in \{0,1\}^{D\times D}$ as follows:

$$\Delta(t)_{\langle q,s\rangle,\langle q',s'\rangle} := \begin{cases} 1 & \text{if } s' = t,\ M_{s,t} > 0, \text{ and } q' \in \delta(q,s) \\ 0 & \text{otherwise} \end{cases}$$

Note that the graph of $\Delta(t)$ contains exactly the edges of the graph of $B_{D,D}$ that end in vertices in $Q \times \{t\}$. If $M_{s,t} > 0$ holds for all pairs (s,t), then the matrices $(\Delta(t))_{t\in S}$ generate a semigroup of matrices, all of which have spectral radius 1. Such semigroups were recently studied by Protasov and Voynov [57]. Specifically, Theorem 5 in [57] shows that there exists an affine subspace $\mathcal{F}$ of $\mathbb{R}^D$ which excludes $\vec{0}$ and is invariant under multiplication by matrices from the semigroup. Moreover, they provide a way to compute this affine subspace efficiently. One can show that cut vectors are orthogonal to $\mathcal{F}$. The key idea of our contribution is to generalise cut vectors to *pseudo-cuts*, which are vectors $\vec{\mu} \in \mathbb{R}^D$ that are orthogonal to $\mathcal{F}$. We will show (in Lemma 4.2 below) how to derive a D-normaliser based on a pseudo-cut that is non-zero only in components that are in a co-reachability set $Co(d)$ (from Lemma 3.15).

If $M_{s,t} = 0$ holds for some s,t (which will often be the case in model checking), then $\Delta(s)\cdot\Delta(t)$ is the zero matrix, which has spectral radius 0 not 1. Therefore, the results of [57] are not directly applicable and we have to move away from matrix semigroups. In the following we re-develop and generalise parts of the theory of [57] so that this chapter is self-contained and products of $\Delta(s)\Delta(t)$ with $M_{s,t} = 0$ are not considered.

Let $w = s_1 s_2 \dots s_n \in S^*$. Define $\Delta(w) = \Delta(s_1)\Delta(s_2)\cdots\Delta(s_n)$. We say w is *enabled* if $M_{s_i,s_{i+1}} > 0$ holds for all $i \in \{1,\dots,n-1\}$. If $f \subseteq D$ is a fibre over s such that sw is enabled, we have $\lceil f \triangleright w\rceil^\top = \lceil f\rceil^\top\Delta(w)$. (Recall that $\lceil F\rceil$ denotes the characteristic vector of a subset $F \subseteq D$.) We overload the term *fibre over s* to describe any vector $\vec{\mu} \in \mathbb{R}^D$ such that $\vec{\mu}_{\langle q,s'\rangle} = 0$ whenever $s' \neq s$. We define *pseudo-cuts over s* to be fibres $\vec{\mu}$ over s such that $\vec{\mu}^\top\Delta(w)\vec{z} = \vec{\mu}^\top\vec{z}$ holds for all $w \in S^*$ such that sw is enabled. Let $c \subseteq Q \times \{s\}$ be a cut with sw enabled.

Then $c \triangleright w$ is a cut, and $[\![c]\!]^\top \Delta(w)\vec{z} = 1 = [\![c]\!]^\top \vec{z}$ holds by Lemma 3.13. It follows that cut vectors are pseudo-cuts.

Example 4.1. *Since $c = \{\langle q_0, a\rangle, \langle q_2, a\rangle\}$ from Example 3.17 is a cut, $[\![c]\!]$ is a pseudo-cut over a. Pseudo-cuts do not need to be combinations of cut vectors: although the fibre $f = \{\langle q_0, a\rangle, \langle q_1, a\rangle\}$ is not a cut, $[\![f]\!]$ is a pseudo-cut over a.*

Fix some $d = \langle q, s\rangle \in D$. Recall that $Co(d)$ consists of those $e \in D$ such that there exists a word w with $\{d, e\} \subseteq d \triangleright w$. We define *$Co(d)$-pseudo-cuts* to be pseudo-cuts $\vec{\mu}$ over s such that $\vec{\mu}_d \neq 0$ and $\vec{\mu}_e = 0$ holds for all $e \notin Co(d)$. From a $Co(d)$-pseudo-cut we can easily derive a D-normaliser:

Lemma 4.2. *Let $\vec{\mu} \in \mathbb{R}^D$ be a $Co(d)$-pseudo-cut. Then $\frac{1}{\vec{\mu}_d}\vec{\mu}$ is a D-normaliser.*

Proof. Let w be an enabled word in M such that $d \triangleright w$ is a cut containing d. Such a word exists (see the proof sketch of Proposition 3.16). Since $([\![d]\!]^\top \Delta(w))^\top = [\![d \triangleright w]\!]$ is a D-normaliser (by Lemma 3.13), it suffices to prove that $\frac{1}{\vec{\mu}_d}\vec{\mu}^\top \vec{z} = [\![d]\!]^\top \Delta(w)\vec{z}$.

We can write $\vec{\mu}$ as $\sum_{d' \in Co(d)} \vec{\mu}_{d'}[\![d']\!]$, so $\vec{\mu}^\top \Delta(w) = \sum_{d' \in Co(d)} \vec{\mu}_{d'}[\![d']\!]^\top \Delta(w)$. For any $d' \in Co(d) \setminus \{d\}$, let w' be such that $\{d, d'\} \subseteq d \triangleright w'$. Now we see that $d' \in d \triangleright ww'$, and since $d \triangleright w$ is a cut so are $d \triangleright ww'$ and $d \triangleright ww'w$. Thus,

$$[\![d]\!]^\top \Delta(w)\vec{z} \;=\; [\![d]\!]^\top \Delta(ww'w)\vec{z} \;\geq\; [\![d]\!]^\top \Delta(w)\vec{z} + [\![d']\!]^\top \Delta(w)\vec{z},$$

which implies $[\![d']\!]^\top \Delta(w)\vec{z} = 0$ for every $d' \in Co(d) \setminus \{d\}$. This means that

$$\vec{\mu}^\top \Delta(w)\vec{z} = \sum_{d' \in Co(d)} \vec{\mu}_{d'}[\![d']\!]^\top \Delta(w)\vec{z} = \vec{\mu}_d[\![d]\!]^\top \Delta(w)\vec{z}.$$

Since $\vec{\mu}$ is a pseudo-cut, this implies that $\frac{1}{\vec{\mu}_d}\vec{\mu}^\top \vec{z} = \frac{1}{\vec{\mu}_d}\vec{\mu}^\top \Delta(w)\vec{z} = [\![d]\!]^\top \Delta(w)\vec{z}$. □

By Lemma 4.2, to find a D-normaliser it suffices to find a $Co(d)$-pseudo-cut. Fix a dominant eigenvector $\vec{y}$ of $B_{D,D}$ so that $\vec{y}$ is strictly positive in all components. One can compute such $\vec{y}$ in time $O(|D|^\kappa)$. By [6, Lemma 8] the vector $\vec{z}_D$ is also a dominant eigenvector of $B_{D,D}$, hence $\vec{y}$ and $\vec{z}_D$ (the latter of which is yet unknown) are scalar multiples. In order to compute a $Co(d)$-pseudo-cut, we compute a basis for the space spanned by $\Delta(w)\vec{y}$ for all enabled words w. We use a technique similar to the one employed by Tzeng in [68] for checking equivalence of probabilistic automata. To make this more efficient, we compute separate basis vectors for each $s \in S$. Define $\Delta'(t) \in \{0,1\}^{D \times D}$ as $\Delta'(t)_{\langle q_1, s_1\rangle, \langle q_2, s_2\rangle} = 1$ if $q_1 = q_2$ and $s_1 = s_2 = t$ and 0 otherwise. Note that $\Delta(s)\Delta'(s) = \Delta(s)$ holds for all $s \in S$.

Lemma 4.3. *Suppose $\vec{y} = B_{D,D}\vec{y}$ is given. Denote by $V(s) \subseteq \mathbb{R}^D$ the vector space spanned by the vectors $\Delta'(s)\Delta(w)\vec{y}$ for $w \in S^*$ and $s \in S$. Let $Q_{D,t} = (Q \times \{t\}) \cap D$ and let $E(t) = \{(s,t) \mid M_{s,t} > 0\}$ be the set of edges in M that end in t. One can compute a basis $R(s)$ of $V(s)$ for all $s \in S$ in time $O(|Q|^2 \sum_{t\in S} |Q_{D,t}||E(t)|)$, where for each $\vec{r} \in R(s)$ we have $\vec{r} = \Delta'(s)\Delta(w)\vec{y}$ for some enabled word sw.*

Proof. Fix an arbitrary total order $<_S$ on S. We define a total order $\ll_S$ on S^* as the "shortlex" order but with words read from right to left. That is, the empty word ε is the smallest element, and for $v, w \in S^*$ and $s, t \in S$, we have $vs \ll_S wt$ if (1) $|vs| < |wt|$ or (2) $|vs| = |wt|$ and $s <_S t$ or (3) $|vs| = |wt|$ and $s = t$ and $v \ll_S w$.

We use a technique similar to the one by Tzeng in [68]. At every step in the algorithm, *worklist* is a set of pairs $(sw, \Delta'(s)\Delta(w)\vec{y})$. We write $\min_{\ll_S}(\mathit{worklist})$ to denote the pair in *worklist* where sw is minimal with respect to $\ll_S$.

1. for each $s \in S$, let $R(s) := \{\Delta'(s)\vec{y}\}$ and $R(s)_\perp := \{\Delta'(s)\vec{y}\}$
2. $\mathit{worklist} := \{(st, \Delta'(s)\Delta(t)\vec{y}) \mid M_{s,t} > 0\}$
3. while $\mathit{worklist} \neq \emptyset$:
   $(tw, \vec{u}) := \min_{\ll_S}(\mathit{worklist})$; $\mathit{worklist} := \mathit{worklist} \setminus \{(tw, \vec{u})\}$
   Using the Gram-Schmidt process[1], let $\vec{u}_\perp$ be the orthogonalisation of $\vec{u}$ against $R_\perp(t)$
   if $\vec{u}_\perp \neq \vec{0}$, i.e., if $\vec{u}$ is linearly independent of $R_\perp(t)$:
   $R(t) := R(t) \cup \{\vec{u}\}$ and $R_\perp(t) := R_\perp(t) \cup \{\vec{u}_\perp\}$
   $\mathit{worklist} := \mathit{worklist} \cup \{(stw, \Delta'(s)\Delta(t)\vec{u}) \mid M_{s,t} > 0\}$
4. return $R(s)$ for all $s \in S$

At any point and for all $s \in S$, the sets $R(s)$ and $R(s)_\perp$ span the same vector space, and this space is a subspace of $V(s)$. The sets $R(s)$ and $R(s)_\perp$ consist of linearly independent fibres over s, and these fibres are possibly non-zero only in the $Q_{D,s}$-components. Hence $|\bigcup_{s\in S} R(s)| \leq |D|$ and thus there are at most $|D|$ iterations of the while loop that increase *worklist*. At every iteration where $\vec{u}$ is dependent on $R(t)_\perp$ the set *worklist* decreases by one, and therefore the algorithm terminates.

In order to prove that $R(s)$ spans $V(s)$ we need the following lemma:

[1]For good numerical stability, one should use the so-called Modified Gram-Schmidt process [35, Chapter 19].

Lemma 4.4. *Let $t \in S$ and $x \in S^*$. Let $U \subseteq S^*$ be a set of words with $u \ll_S x$ for all $u \in U$ such that $\Delta'(t)\Delta(x)\vec{y}$ is linearly dependent on $\{\Delta'(t)\Delta(u)\vec{y} \mid u \in U\}$. Let $s \in S$ and $w \in S^*$. Then there exists a set U' of words $u' \ll_S wtx$ such that $\Delta'(s)\Delta(wtx)\vec{y}$ is linearly dependent on $\{\Delta'(s)\Delta(u')\vec{y} \mid u' \in U'\}$.*

Proof. We have:

$$\begin{aligned}
\Delta'(s)\Delta(wtx)\vec{y} &= \Delta'(s)\Delta(w)\Delta(t)\Delta(x)\vec{y} \\
&= \Delta'(s)\Delta(w)\Delta(t)\Delta'(t)\Delta(x)\vec{y} \\
&= \Delta'(s)\Delta(w)\Delta(t)\Big(\sum_{u\in U} \gamma_u \Delta'(t)\Delta(u)\vec{y}\Big) \\
&= \sum_{u\in U} \gamma_u \Delta'(s)\Delta(wtu)\vec{y}\,.
\end{aligned}$$

Hence choose $U' = \{wtu \mid u \in U\}$. □

We now use induction to prove that for each s the span of $R(s)$ contains $V(s)$. Denote all non-empty words in S^* by $v^{(1)}, v^{(2)}, \ldots$ such that $v^{(i)} \ll_S v^{(j)}$ if and only if $i < j$. For each $v^{(i)}$, let $s^{(i)} \in S$ and $w^{(i)} \in S^*$ be such that $v^{(i)} = s^{(i)}w^{(i)}$. For each $v^{(i)}$ such that $w^{(i)}$ is the empty word, $\Delta'(s^{(i)})\vec{y}$ is linearly dependent on $R(s^{(i)})$. Our induction hypothesis is that for each $v^{(i)}$ for $i \leq n$, we have that $\Delta'(s^{(i)})\Delta(w^{(i)})\vec{y}$ is dependent on $R(s^{(i)})$. For the inductive step, consider $v^{(n+1)}$ and distinguish two cases:

- Either $\Delta'(s^{(n+1)})\Delta(w^{(n+1)})\vec{y}$ was visited by the algorithm:
  Then either $\Delta'(s^{(n+1)})\Delta(w^{(n+1)})\vec{y}$ was shown to be dependent on $R(s^{(n+1)})$ at that time, or $\Delta'(s^{(n+1)})\Delta(w^{(n+1)})\vec{y}$ was independent of $R(s^{(n+1)})$ at that time, after which $\Delta'(s^{(n+1)})\Delta(w^{(n+1)})\vec{y}$ was added to $R(s^{(n+1)})$.

- Or $\Delta'(s^{(n+1)})\Delta(w^{(n+1)})\vec{y}$ was not visited by the algorithm:
  Then for some suffix $v^{(q)}$ of $v^{(n+1)}$ the vector $\Delta'(s^{(q)})\Delta(w^{(q)})\vec{y}$ was visited by the algorithm and was found to be dependent on $R(s^{(q)})$ at the time, which was dependent on vectors of the form $\Delta'(s^{(q)})\Delta(w^{(p)})\vec{y}$, with $w^{(p)} \ll_S w^{(q)}$. By Lemma 4.4 the vector $\Delta'(s^{(n+1)})\Delta(w^{(n+1)})\vec{y}$ is dependent on vectors of the form $\Delta'(s^{(n+1)})\Delta(w^{(m)})\vec{y}$ with $w^{(m)} \ll_S w^{(n+1)}$. Since $s^{(n+1)}w^{(m)} \ll_S v^{(n+1)}$, by the induction hypothesis all of these vectors are dependent on $R(s^{(n+1)})$. Therefore, $\Delta'(s^{(n+1)})\Delta(w^{(n+1)})\vec{y}$ is dependent on $R(s^{(n+1)})$.

Concerning runtime, for every vector $\vec{u} = \Delta'(t)\Delta(w)\vec{y}$ visited by the algorithm, $\vec{u}$ is orthogonalised against $R(t)$. This costs $O(|Q|^2)$ time, as $\vec{u}$ and every vector in $R(t)$ is a fibre over t. Moreover, one can calculate $\Delta'(s)\Delta(t)\vec{u}$ from $\vec{u}$ in time

$O(|Q|^2)$, as $\Delta'(s)\Delta(t)$ is possibly non-zero only in a $|Q| \times |Q|$ submatrix. The algorithm adds $|E(t)|$ vectors to *worklist* each time a vector gets added to $R(t)$, and the latter happens at most $|Q_{D,t}|$ times since the non-zero elements only occur in $\langle q, t\rangle$ with $\langle q, t\rangle \in D$. This gives us the bound of $O(|Q|^2 \sum_{t\in S} |Q_{D,t}||E(t)|)$. □

Example 4.5. *Let us return to our running example. We see that the vector* $\vec{y} = (\vec{y}_{\langle q_0,a\rangle}, \vec{y}_{\langle q_1,a\rangle}, \vec{y}_{\langle q_1,b\rangle}, \vec{y}_{\langle q_2,a\rangle}, \vec{y}_{\langle q_2,b\rangle}, \vec{y}_{\langle q_3,a\rangle})^\top = (2,1,3,1,3,2)^\top$ *is a dominant eigenvector of* $B_{D,D}$*. Fix the order* $a <_S b$*. Step 1 initialises* $R(a)$ *to* $\{\Delta'(a)\vec{y}\}$ *and* $R(b)$ *to* $\{\Delta'(b)\vec{y}\}$*, where* $\Delta'(a)\vec{y} = (2,1,0,1,0,2)^\top$ *and* $\Delta'(b)\vec{y} = (0,0,3,0,3,0)^\top$*. Step 2 computes* $\Delta'(a)\Delta(a)\vec{y} = (1,2,0,2,0,1)^\top$*, which is linearly independent of* $\Delta'(a)\vec{y}$*. However,* $\Delta'(b)\Delta(a)\vec{y} = (0,0,3,0,3,0)^\top = \Delta'(b)\vec{y}$*. Also,* $\Delta'(a)\Delta(b)\vec{y} = (3,0,0,0,0,3)^\top = 2\Delta'(a)\vec{y} - \Delta'(a)\Delta(a)\vec{y}$ *and* $\Delta'(b)\Delta(b)\vec{y} = (0,0,3,0,3,0)^\top = \Delta'(b)\vec{y}$*. One can check that* $\Delta'(a)\Delta(aa)\vec{y} = \Delta'(a)\vec{y}$ *and* $\Delta'(b)\Delta(aa)\vec{y} = \Delta'(b)\vec{y}$*. Hence the algorithm returns* $R(a) = \{(2,1,0,1,0,2)^\top, (1,2,0,2,0,1)^\top\}$ *and* $R(b) = \{(0,0,3,0,3,0)^\top\}$*.*

Fix $d = \langle q, s\rangle \in D$ for the rest of this chapter. The following lemma characterises $Co(d)$-pseudo-cuts in a way that is efficiently computable:

Lemma 4.6. *A vector* $\vec{\mu} \in \mathbb{R}^D$ *with* $\vec{\mu}_d = 1$ *and* $\vec{\mu}_e = 0$ *for all* $e \notin Co(d)$ *is a* $Co(d)$*-pseudo-cut if and only if* $\vec{\mu}^\top \vec{r} = \vec{\mu}^\top \vec{y}$ *holds for all* $\vec{r} \in R(s)$*.*

For an intuition of the proof, consider the affine space, $\mathcal{F} \subseteq \mathbb{R}^D$, affinely spanned by those $\Delta'(s)\Delta(w)\vec{y}$ for which sw is enabled. This affine space was alluded to in the beginning of this chapter and is visualised as a blue straight line in Figure 1.3. The shaded plane in this figure is the vector space of pseudo-cuts over s. This space is orthogonal to $\mathcal{F}$. The following lemma says that $\mathcal{F}$ is affinely spanned by the points in $R(s)$. This strengthens the property of $R(s)$ in Lemma 4.3 where $R(s)$ was defined to span a *vector* space.

Lemma 4.7. *Let* $w \in S^*$ *be such that* sw *is enabled. By the definition of* $R(s)$ *there are* $\gamma_{\vec{r}} \in \mathbb{R}^D$ *for each* $\vec{r} \in R(s)$ *such that* $\Delta'(s)\Delta(w)\vec{y} = \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{r}$*. We have* $\sum_{\vec{r}\in R(s)} \gamma_{\vec{r}} = 1$*.*

Proof. Let c be a cut containing d. Since $R(s)$ is a basis, for any $\vec{r} = \Delta'(s)\Delta(w_{\vec{r}})\vec{y} \in R(s)$ the word $sw_{\vec{r}}$ is enabled. Therefore, $c \triangleright w_{\vec{r}}$ is a cut and by Lemma 3.13 we have $[\![c \triangleright w_{\vec{r}}]\!]^\top \vec{y} = [\![c]\!]^\top \vec{y}$. Hence $[\![c]\!]^\top \vec{r} = [\![c]\!]^\top \Delta'(s)\Delta(w_{\vec{r}})\vec{y} = [\![c]\!]^\top \Delta(w_{\vec{r}})\vec{y} = [\![c \triangleright w_{\vec{r}}]\!]^\top \vec{y} =$

$[c]^\top \vec{y}$. Moreover, we have:

$$\begin{aligned}
[c]^\top \vec{y} &= [c]^\top \Delta(w)\vec{y} && \text{since } sw \text{ is enabled and by Lemma 3.13}\\
&= [c]^\top \Delta'(s)\Delta(w)\vec{y} && \text{since } [c] \text{ is a fibre over } s\\
&= [c]^\top \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{r} && \text{by the definition of } \gamma_{\vec{r}}\\
&= [c]^\top \vec{y} \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}} && \text{as argued above.}
\end{aligned}$$

Therefore, $\sum_{\vec{r}\in R(s)} \gamma_{\vec{r}} = 1$. □

Now we can prove Lemma 4.6:

Proof of Lemma 4.6. For the "if" direction, let w be such that sw is enabled, and it suffices to show that $\vec{\mu}^\top \Delta(w)\vec{y} = \vec{\mu}^\top \vec{y}$. By Lemma 4.7 there are $\gamma_{\vec{r}}$ such that $\Delta'(s)\Delta(w)\vec{y} = \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{r}$ and $\sum_{\vec{r}\in R(s)} \gamma_{\vec{r}} = 1$. We have:

$$\vec{\mu}^\top \Delta(w)\vec{y} \;=\; \vec{\mu}^\top \Delta'(s)\Delta(w)\vec{y} \;=\; \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{\mu}^\top\vec{r} \;=\; \sum_{\vec{r}\in R(s)} \gamma_{\vec{r}}\vec{\mu}^\top\vec{y} \;=\; \vec{\mu}^\top\vec{y}\,,$$

where the last equality is from Lemma 4.7.

For the "only if" direction, suppose $\vec{\mu}$ is a $Co(d)$-pseudo-cut. Let $\vec{r} = \Delta'(s)\Delta(w_{\vec{r}})\vec{y} \in R(s)$. Then $sw_{\vec{r}}$ is enabled and $\vec{\mu}^\top\vec{r} = \vec{\mu}^\top\Delta'(s)\Delta(w_{\vec{r}})\vec{y} = \vec{\mu}^\top\Delta(w_{\vec{r}})\vec{y} = \vec{\mu}^\top\vec{y}$. □

Example 4.8. *In Example 3.17 we derived that* $\vec{y} = (2,1,3,1,3,2)^\top$ *and* $R(a) = \{(2,1,0,1,0,2)^\top, (1,2,0,2,0,1)^\top\}$. *The cut vector* $\vec{\mu} = (1,0,0,1,0,0)^\top$ *from Example 4.1 satisfies* $\vec{\mu}^\top\vec{r} = 3 = \vec{\mu}^\top\vec{y}$ *for both* $\vec{r} \in R(a)$.

Using Lemmas 3.15, 4.3 and 4.6 we obtain:

Proposition 4.9. *Let* $D \subseteq Q \times S$ *be a recurrent strongly connected component. Denote by* T_D *the set of edges of* $B_{D,D}$. *For* $t \in S$, *let* $E(t)$ *denote the set of edges of* M *that end in* t, *and let* $Q_{D,t} = (Q \times \{t\}) \cap D$. *Let* $d = \langle q, s\rangle \in D$. *One can compute a* $Co(d)$*-pseudo-cut in time* $O(|D|^\kappa + |Q||D| + |\delta||T_D| + |Q|^2 \sum_{t\in S} |Q_{D,t}||E(t)|)$.

Proof. One can compute $\vec{y}$ in time $O(|D|^\kappa)$. By Lemma 3.15, one can calculate $Co(d)$ in time $O(|Q||D| + |\delta||T_D|)$. By Lemma 4.3, one can compute $R(s)$ in time $O(|Q|^2 \sum_{t\in S} |Q_{D,t}||E(t)|)$. The $O(|D|)$ equations from Lemma 4.6 can be solved in time $O(|D|^\kappa)$. The total runtime is $O(|D|^\kappa + |Q||D| + |\delta||T_D| + |Q|^2 \sum_{t\in S} |Q_{D,t}||E(t)|)$. □

Now our main result follows, which we restate here:

Theorem 1.9. *Given a Markov chain $\mathcal{M} = (S, M)$, an initial distribution ι, and a UBA $\mathcal{A} = (Q, S, \delta, Q_0, F)$, one can compute $\Pr_\iota^{\mathcal{M}}(\mathcal{L}(\mathcal{A}))$ in time $O(|Q|^\kappa |S|^\kappa + |Q|^3|E| + |\delta|^2|E|)$, where $\kappa \in [2, 3)$ is such that one can multiply two $n \times n$-matrices in $O(n^\kappa)$.*

Proof. Denote by $\mathcal{D}$ the set of accepting recurrent strongly connected components. Using Proposition 4.9 we compute a normaliser for each of them. Since there are at most $|\delta||E|$ edges in B, we have $\sum_{D\in\mathcal{D}} |T_D| \leq |\delta||E|$. Hence, $\sum_{D\in\mathcal{D}} |D|^\kappa + |Q||D| + |\delta||T_D|$ is $O(|Q|^\kappa|S|^\kappa + \delta^2|E|)$. For any $t \in S$ and different strongly connected components $D, D' \in \mathcal{D}$, the sets $Q_{D,t}$ and $Q_{D',t}$ are disjoint. Thus,

$$\sum_{t\in S}\sum_{D\in\mathcal{D}} |Q_{D,t}||E(t)| \;\leq\; \sum_{t\in S} |Q||E(t)| \;=\; |Q||E|\,.$$

Hence by Proposition 4.9, one can compute normalisers for all accepting recurrent strongly connected components in time $O(|Q|^\kappa|S|^\kappa + |Q|^3|E| + |\delta|^2|E|)$. Using Proposition 3.10 we get a total runtime of $O(|Q|^\kappa|S|^\kappa + |Q|^3|E| + |\delta|^2|E|)$. □

Note that the procedure for computing pseudo-cuts can also be placed in NC, giving an NC model checking procedure overall.

4.3 The Number of Transitions Can Be Quadratic

Since the complexity of model checking using pseudo-cuts improves on the complexity using cuts by a factor $|\delta||S|$ at the cost of a factor $|E|$, one may ask about upper bounds on $|\delta|$ for unambiguous automata. Clearly, $|\delta|$ can be $O(|Q|^2)$ in the case where $\mathcal{A}$ is a non-deterministic, not necessarily unambiguous automaton. We show that $|\delta|$ can also be $O(|Q|^2)$ when $\mathcal{A}$ is unambiguous.

The example in Figure 4.1 shows a strongly connected UBA where the number of transitions is quadratic in $|Q|$ for a fixed alphabet $\Sigma = \{a, b\}$. Indeed, for any $n \geq 2$ we can create a similar strongly connected UBA with $|Q| = O(2^n)$ states and at least $(2^n)^2$ transitions.

Such a UBA can be created as follows: take two (directed) complete binary trees T_1 and T_2 where each node except for the leaves has 2 outgoing edges labelled by a, b, respectively. In T_2, flip the directions of all edges and label them all with both a and b. From the former root of T_2, add an a-labelled self-loop and a b-labelled transition to the root of T_1. From each leaf of T_1 add transitions to each former leaf of T_2, all labelled by both a and b. The resulting graph is a strongly connected UBA with a number of arrows that is quadratic in the number of states, similar to the one in Figure 4.1.

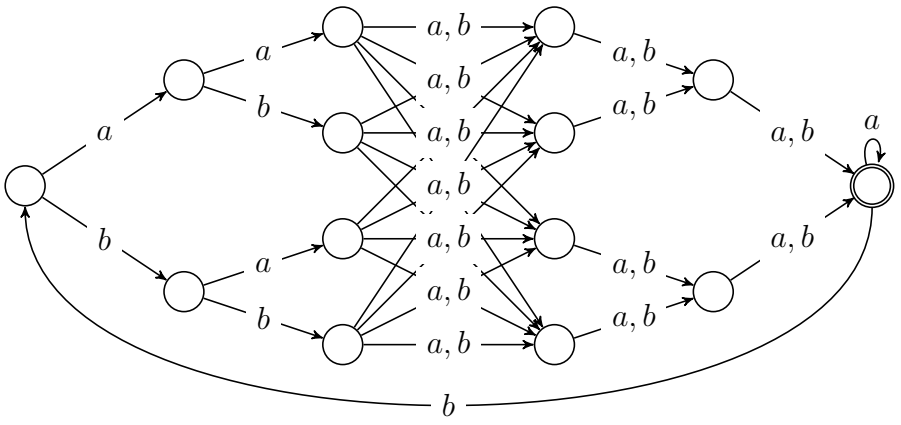


Figure 4.1: An unambiguous automaton with a single recurrent strongly connected component and a quadratic number of edges.

4.4 Conclusion

In this chapter we introduced a novel algorithm for computing normalisers. This approach for computing normalisers combines a linear-algebra component to compute $R(s)$ with a combinatorial algorithm to compute the co-reachability set $Co(d)$. In terms of the automaton, the linear-algebra component runs in time $O(|Q|^3)$ (Lemma 4.3), while the combinatorial part runs in time $O(|\delta|^2)$, leading to an overall runtime of $O(|Q|^3 + |\delta|^2)$. Recall from Chapter 3 that the cut based approach runs in time $O(|Q|^3|\delta| + |\delta|^2) = O(|Q|^3|\delta|)$ (Proposition 3.16). Note that for all $r \in [1, 2]$, if $|\delta| = \Theta(|Q|^r)$ then the second approach is faster by at least a factor of $|Q|$.

We have also analysed the complexity of both the cut based algorithm and the pseudo-cut based algorithm in terms of the size of the Markov chain. The cut based algorithm runs in time $O(|Q|^\kappa|S|^\kappa + |Q|^3|\delta||S| + |\delta|^2|E|)$, while the pseudo-cut based algorithm runs in time $O(|Q|^\kappa|S|^\kappa + |Q|^3|E| + |\delta|^2|E|)$. There are cases in which the latter is asymptotically worse, but not if $\kappa = 3$ (i.e., solving linear systems in a normal way such as Gaussian elimination) or if $|E|$ is $O(|S|)$.

It is perhaps unsurprising that a factor of $|\delta|^2$ from the computation of $Co(d)$ occurs in the runtime, as it also occurs when one merely verifies the unambiguousness of the automaton, by searching the product of the automaton with itself. Can the factor $|\delta|^2$ (which may be quartic in $|Q|$) be avoided?

5 Image-binary automata

Contents

5.1 Introduction

In this chapter we introduce a new variant of weighted automata called *image-binary automata*. Image-binary automata are weighted automata (over $\mathbb{Q}$ or $\mathbb{R}$) such that every word has either weight 0 or weight 1. This makes it a semantic class; however, we show that it can be easily decided whether a given $\mathbb{Q}$-weighted automaton is image-binary.

We show that image-binary automata are efficiently closed under the Boolean operations, which is a key advantage over other automata such as unambiguous automata, and that they can be exponentially more succinct than deterministic

automata. We also compare them against a different class of automata that allows for efficient complementation, namely mod-2-multiplicity automata. We show that IFAs can be converted to mod-2-multiplicity automata, while converting a mod-2-multiplicity automaton to an IFA may incur an exponential state set blowup. Finally we consider an infinite word version of image-binary automata, using a Büchi acceptance condition. We show that k-ambiguous Büchi automata can be translated into image-binary automata with an exponential blowup in k (which itself may be exponential in the state set size), and we show that model checking Markov chains against image-binary Büchi automata can be done in NC. Combining the translation and model checking procedure gives a PSPACE model checking procedure for k-ABAs, implying optimality of both the translation and the model checking procedure.

In Section 5.2 we consider image-binary finite-word automata. In Section 5.2.2 we show that IFAs accept the regular languages, in Section 5.2.3 we show their efficient closure under Boolean operations, and in Section 5.2.4 we give succinctness results for IFAs. In Section 5.2.5 we compare IFAs against mod-2-multiplicity automata, as described by Angluin et al.[3]. After that we look at image-binary Büchi automata in Section 5.3. In Section 5.3.2 we give a translation from k-ABAs to IBAs, resulting in an exponential blowup in the state size. In Section 5.3.3 we adapt the procedure for model checking Markov chains against UBAs to IBAs.

5.2 Image-Binary Finite Automata

5.2.1 Definitions

Let $\mathcal{A} = (Q, \Sigma, M, \alpha, \eta)$ be a $\mathbb{Q}$-weighted automaton. We call $\mathcal{A}$ an *image-binary (weighted) finite automaton (IFA)* if $L_{\mathcal{A}}(\Sigma^*) \subseteq \{0, 1\}$, i.e., $L_{\mathcal{A}}(w) \in \{0, 1\}$ holds for all $w \in \Sigma^*$. An $\mathbb{R}$-IFA is defined like an IFA, but with $\mathbb{Q}$ replaced by $\mathbb{R}$. An ($\mathbb{R}$)-IFA $\mathcal{A}$ defines a language $L(\mathcal{A}) := \{w \in \Sigma^* \mid L_{\mathcal{A}}(w) = 1\}$. Note that we call both $L_{\mathcal{A}}$ and $L(\mathcal{A})$ the language of A; strictly speaking, the former is the characteristic function of the latter.

If an IFA $\mathcal{A} = (Q, \Sigma, M, \alpha, \eta)$ is such that $\alpha \in \{0, 1\}^Q$ and $\eta \in \{0, 1\}^Q$ and $M(a) \in \{0, 1\}^{Q \times Q}$ for all $a \in \Sigma$, then $\mathcal{A}$ is an unambiguous finite automaton (UFA). Note that this definition of a UFA is essentially equivalent to the classical one used throughout this thesis. Similarly, a deterministic finite automaton (DFA) is essentially a special case of a UFA, and hence of an IFA.

Example 5.1. *Figure 2.1 shows an IFA and a UFA in a graphical notation. Formally, the IFA on the left is* $\mathcal{A} = (Q_\mathcal{A}, \Sigma, M_\mathcal{A}, \alpha_\mathcal{A}, \eta_\mathcal{A})$ *with* $Q = \{1, 2, 3\}$ *and* $\Sigma = \{a, b\}$ *and*

$$M_\mathcal{A}(a) = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad M_\mathcal{A}(b) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and $\alpha_\mathcal{A} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$ *and* $\eta_\mathcal{A} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$*. Both automata recognise the language of words that start in an even (positive) number of as.*

5.2.2 Regularity

Since a DFA is an IFA, for each regular language there is an IFA that defines it. We show that IFAs accept precisely the regular languages:

Theorem 5.2. *Let* $L(\mathcal{A})$ *be any regular language, then there exists an* $\mathbb{R}$*-IFA* $\mathcal{A}$ *accepting* $L(\mathcal{A})$*. Conversely, let* $\mathcal{A} = (Q, \Sigma, M, \alpha, \eta)$ *be an* $\mathbb{R}$*-IFA. Then* $L(\mathcal{A})$ *is regular, and there is a DFA* $\mathcal{B}$ *with at most* $2^{|Q|}$ *states and* $L(\mathcal{A}) = L(\mathcal{B})$*.*

In order to prove this, we need to show that the Hankel matrix of $L(\mathcal{A})$ has at most exponentially many different rows. For this, we will need some auxiliary lemmas. View $\mathbb{Z}_2 = \{0, 1\}$ as the field with two elements. In the proof of Theorem 5.2 we consider vector spaces *over* $\mathbb{Z}_2$, i.e., where the scalars are from $\mathbb{Z}_2$. In particular, we will argue with the vector space $\mathbb{Z}_2^\mathbb{N} \cong \mathbb{Z}_2^{\Sigma^*}$ over $\mathbb{Z}_2$. We first show:

Lemma 5.3. *Let* V *be a set of* n *vectors. Consider the vector space* $\langle V \rangle$ *spanned by* V *over* $\mathbb{Z}_2$*. Then* $|\langle V \rangle| \leq 2^n$*.*

Proof. Let $V = \{v_1, \ldots, v_n\}$. Then $\langle V \rangle = \{\sum_{i=1}^n \lambda_i v_i \mid \lambda_i \in \mathbb{Z}_2\}$. □

Corollary 5.4. *Let* V *be a vector space over* $\mathbb{Z}_2$*. For any* $n \in \mathbb{N}$*, if* $\dim \mathsf{V} \leq n$ *then* $|\mathsf{V}| \leq 2^n$*.*

The following two lemmas show that if an $\mathbb{R}$-weighted automaton is image-binary, then the rank over $\mathbb{R}$ of its Hankel matrix H is at least the rank of H over $\mathbb{Z}_2$.

Lemma 5.5. *Let* $V \subseteq \{0, 1\}^\mathbb{N}$ *be a set of vectors. If* V *is linearly dependent over* $\mathbb{R}$ *then* V *is linearly dependent over* $\mathbb{Q}$*. Hence* $\dim\langle V \rangle_\mathbb{Q} \leq \dim\langle V \rangle_\mathbb{R}$*.*

Proof. Let V be linearly dependent (over $\mathbb{R}$), and let $n = \dim\langle V\rangle$. Then there are $v_0, v_1, \ldots, v_n \in V$ such that $V' := \{v_1, \ldots, v_n\}$ is linearly independent and $v_0 \notin V'$ but $v_0 \in \langle V'\rangle$. So there are unique $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ with $v_0 = \sum_{i=1}^n \lambda_i v_i$. It suffices to show that $\lambda_1, \ldots, \lambda_n \in \mathbb{Q}$. Since V' is linearly independent, there exists $J \subseteq \mathbb{N}$ with $|J| = n$ such that $\{v_1[J], \ldots, v_n[J]\}$ is linearly independent, where $v_i[J] \in \{0,1\}^n$ is the restriction of v_i to the entries indexed by J. Hence the λ_i are the unique solution of a linear system of equations

$$v_0[j] \;=\; \sum_{i=1}^{n} \lambda_i v_i[j] \qquad \text{for all } j \in J.$$

Linear systems of equations with rational coefficients and constants have rational solutions. Hence $\lambda_1, \ldots, \lambda_n \in \mathbb{Q}$. □

Lemma 5.6. *Let $V \subseteq \{0,1\}^{\mathbb{N}}$ be a set of vectors. If V is linearly dependent over $\mathbb{Q}$ then V is linearly dependent over $\mathbb{Z}_2$. Hence* $\dim\langle V\rangle_{\mathbb{Z}_2} \leq \dim\langle V\rangle_{\mathbb{Q}}$.

Proof. Let $V = \{v_1, \ldots, v_n\}$ and $\sum_{i=1}^n \lambda_i v_i = \vec{0}$, where $\lambda_i \in \mathbb{Q}$ are not all zero. By multiplying all λ_i with a common denominator, we can assume without loss of generality that $\lambda_i \in \mathbb{Z}$ where λ_i are not all zero. By dividing all λ_i with the largest power of 2 that divides all λ_i, we can assume without loss of generality that the λ_i are not all even. In the equation $\sum_{i=1}^n \lambda_i v_i = \vec{0}$, by regarding every λ_i and every entry of every v_i modulo 2, we have $\sum_{i=1}^n \lambda_i v_i = \vec{0}$ over $\mathbb{Z}_2$, i.e., V is linearly dependent over $\mathbb{Z}_2$. □

Hence we can prove Theorem 5.2:

Proof of Theorem 5.2. Since any DFA is also an $\mathbb{R}$-IFA, the first claim is obvious. For the second, write $n = |Q|$. Let H be the Hankel matrix of $L_{\mathcal{A}}$. We have $H \in \{0,1\}^{\Sigma^* \times \Sigma^*}$. By proposition 2.3 we have $\operatorname{rank} H \leq n$, where the rank is over $\mathbb{R}$. By lemmas 5.5 and 5.6 it follows that $\operatorname{rank} H \leq n$, where the rank is over $\mathbb{Z}_2$. By corollary 5.4 it follows that H has at most 2^n different rows, say $H[w_1, \cdot], \ldots, H[w_\ell, \cdot]$ with $\ell \leq 2^n$. So every word is Myhill-Nerode equivalent to a word in $\{w_1, \ldots, w_\ell\}$. Thus, there is an equivalent DFA with ℓ states.

Explicitly, the following DFA $\mathcal{B} = (Q', \Sigma, \delta, q_0, F)$ is equivalent to $\mathcal{A}$:

$$\begin{aligned} Q' &= \{H[w_1, \cdot], \ldots, H[w_\ell, \cdot]\} \\ \delta(H[w_i, \cdot], a) &= H[w_i a, \cdot] \\ q_0 &= H[\varepsilon, \cdot] \\ F &= \{H[w_i, \cdot] \mid 1 \leq i \leq \ell,\ H[w_i, \varepsilon] = 1\}. \end{aligned}$$

□

5.2.3 Boolean Operations and Checking Image-Binariness

IFAs are *polynomially closed* under all boolean operations, by which we mean:

Theorem 5.7. *Let $\mathcal{A}_1, \mathcal{A}_2$ be IFAs over Σ. One can compute in polynomial time IFAs $\mathcal{B}_\neg, \mathcal{B}_\cap, \mathcal{B}_\cup$ with $L(\mathcal{B}_\neg) = \Sigma^* \setminus L(\mathcal{A}_1)$ and $L(\mathcal{B}_\cap) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ and $L(\mathcal{B}_\cup) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$.*

By De Morgan's laws it suffices to construct $\mathcal{B}_\neg$ and $\mathcal{B}_\cap$. Since $\mathcal{B}_\neg$ and $\mathcal{B}_\cap$ need to satisfy only $L_{\mathcal{B}_\neg} = 1 + (-L_{\mathcal{A}_1})$ (where $1 : \Sigma^* \to \{1\}$ denotes the constant 1 function) and $L_{\mathcal{B}_\cap} = L_{\mathcal{A}_1} \cdot L_{\mathcal{A}_2}$, it suffices to know that $\mathbb{Q}$-weighted automata are polynomially closed under negation and pointwise addition and multiplication:

Proposition 5.8 (see, e.g., [10, Chapter 1])**.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be $\mathbb{Q}$-weighted automata. One can compute in polynomial time $\mathbb{Q}$-weighted automata $\mathcal{B}_-, \mathcal{B}_+, \mathcal{B}_\times$ with $L_{\mathcal{B}_-} = -L_{\mathcal{A}_1}$ and $L_{\mathcal{B}_+} = L_{\mathcal{A}_1} + L_{\mathcal{A}_2}$ and $L_{\mathcal{B}_\times} = L_{\mathcal{A}_1} \cdot L_{\mathcal{A}_2}$.*

Proof. For $i \in \{1, 2\}$ let $\mathcal{A}_i = (Q_i, \Sigma, M_i, \alpha_i, \eta_i)$. For $\mathcal{B}_-$, replace α_1 by $-\alpha_1$. For $\mathcal{B}_+$, assume $Q_1 \cap Q_2 = \emptyset$, take $Q = Q_1 \cup Q_2$, and $M(a) = \begin{pmatrix} M_1(a) & 0 \\ 0 & M_2(a) \end{pmatrix}$ for all $a \in \Sigma$, and $\alpha = \begin{pmatrix} \alpha_1 & \alpha_2 \end{pmatrix}$, and $\eta = \begin{pmatrix} \eta_1 \\ \eta_2 \end{pmatrix}$. For $\mathcal{B}_\times$, take $Q = Q_1 \times Q_2$, and $M(a) = M_1(a) \otimes M_2(a)$ for all $a \in \Sigma$, and $\alpha = \alpha_1 \otimes \alpha_2$, and $\eta = \eta_1 \otimes \eta_2$, where $\otimes$ stands for the Kronecker product. It follows from the mixed-product property of $\otimes$ (i.e., $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$) that indeed $L_{\mathcal{B}_\times} = L_{\mathcal{A}_1} \cdot L_{\mathcal{A}_2}$. □

While DFAs are also polynomially closed under complement (switch accepting and non-accepting states), NFAs and UFAs are not. For NFAs, it was shown in [36] that the (worst-case) blowup in the number n of states is $\Theta(2^n)$. For UFAs, it was shown recently:

Proposition 5.9 ([60])**.** *For any $n \in \mathbb{N}$ there exists a unary (i.e., on an alphabet Σ with $|\Sigma| = 1$) UFA $\mathcal{A}_n$ with n states such that any NFA for the complement language has at least $n^{(\log\log\log n)^{\Theta(1)}}$ states.*

This super-polynomial blowup (even for unary alphabet and even if the output automaton is allowed to be ambiguous) refuted a conjecture that it may be possible to complement UFAs with a polynomial blowup [18]. An upper bound (for general alphabets and requiring the output to be a UFA) of $O(2^{0.79n})$ was shown in [40]; see also [38] for an (unpublished) improvement.

The author believes Proposition 5.9 shows the strength of Theorem 5.7: while UFAs cannot be complemented efficiently, the more general IFAs are polynomially closed under all boolean operations.

Proposition 5.8 can be used to show:

Theorem 5.10. *Given a $\mathbb{Q}$-weighted automaton, one can check in polynomial time if it is an IFA.*

Proof. Let $\mathcal{A}$ be a $\mathbb{Q}$-weighted automaton. By Proposition 5.8 one can compute in polynomial time a $\mathbb{Q}$-weighted automaton $\mathcal{B}$ with $L_{\mathcal{B}} = L_{\mathcal{A}} \cdot L_{\mathcal{A}}$ (pointwise multiplication). Then $\mathcal{A}$ is an IFA if and only if $\mathcal{A}$ and $\mathcal{B}$ are equivalent. Equivalence of $\mathbb{Q}$-weighted automata can be checked in polynomial time, see [63; 67]. □

5.2.4 Succinctness

It is known that UFAs can be exponentially more succinct than DFAs: for each $n \in \mathbb{N}$ with $n \geq 3$ there is a UFA with n states such that the smallest equivalent DFA has 2^n states, see [50, Theorem 1]. Since UFAs are IFAs, Theorem 5.2 is optimal:

Corollary 5.11. *For converting IFAs to DFAs, a state blowup of 2^n is sufficient and necessary.*

It is also known from [50] that converting NFAs to UFAs can require $2^n - 1$ states. The argument carries over to IFAs:

Proposition 5.12. *For converting NFAs to IFAs, a state blowup of $\Theta(2^n)$ is sufficient and necessary in the worst case.*

Proof. Sufficiency is clear via the subset construction. For necessity, Leung [50, Theorem 3] considers for each $n \geq 3$ an NFA (even an MDFA, which is a DFA with multiple initial states) such that its Hankel matrix has rank $2^n - 1$, which he proved in [49]. He then invokes an analogue of Proposition 2.3, due to Schmidt [62], to show that any equivalent UFA needs at least $2^n - 1$ states. But by Proposition 2.3 this holds also for IFAs. □

It follows from Theorem 5.7 and Proposition 5.9 that IFAs cannot be converted to NFAs in polynomial time:

Proposition 5.13. *Converting IFAs to NFAs requires a super-polynomial state blowup.*

Proof. Let $n \in \mathbb{N}$, and let $\mathcal{A}_n$ be the UFA from proposition 5.9. By theorem 5.7 there is a polynomial-size IFA, say $\mathcal{B}_n$, for the complement of $L(\mathcal{A}_n)$. If converting IFAs to NFAs required only a polynomial state blowup, there would exist an NFA, say $\mathcal{B}'_n$, with $L(\mathcal{B}'_n) = L(\mathcal{B}_n) = \Sigma^* \setminus L(\mathcal{A}_n)$, of size polynomial in $\mathcal{A}_n$, contradicting proposition 5.9. □

5.2.5 Mod-2-Multiplicity Automata

We compare IFAs with the mod-2-multiplicity automata (mod-2-MAs) as introduced in [3], which are weighted automata over the field $\mathbb{Z}_2$. Given a mod-2-MA $\mathcal{A}$ and a word w, w is accepted iff $\mathcal{A}(w) = 1$, i.e., the weight of the word assigned by $\mathcal{A}$ is 1. Like with IFAs, mod-2-MAs are exponentially more succinct than DFAs [3, Lemma 6]. Converting NFAs to mod-2-MAs requires a super-polynomial state blowup [3, Lemma 10] while (under the assumption that there are infinitely many Mersenne primes) converting mod-2-MAs to NFAs requires an exponential blowup [3, Lemma 11].

We can convert IFAs to mod-2-MAs without incurring a blowup:

Proposition 5.14. *For any IFA $\mathcal{A}$ with n states there exists a mod-2-MA $\mathcal{A}'$ of at most n states with $L_{\mathcal{A}} = L_{\mathcal{A}'}$.*

Proof. Let H be the Hankel matrix of $L_{\mathcal{A}}$. By Proposition 2.3, $\operatorname{rank} H \leq n$, where the rank is taken over $\mathbb{R}$. Invoking Lemma 5.5 and Lemma 5.6 then shows that $\operatorname{rank} H \leq n$ also when the rank is taken over $\mathbb{Z}_2$. Then by Proposition 2.3 there exists a mod-2-MA with $\operatorname{rank} H \leq n$ (over $\mathbb{Z}_2$) states that accepts the same language as $\mathcal{A}$. □

However, the converse requires an exponential blowup. Inspired by Angluin et al.'s [3] proof that mod-2 automata can be exponentially more succinct than NFAs, this proof makes use of shift register sequences. However, note that this proof does not require the assumption that there are infinitely many Mersenne primes. For further information on shift register sequences, see [31]. A *shift register sequence* of dimension d is an infinite periodic sequence $\{a_n\}$ of bits defined by initial conditions $a_i = b_i$ for $i = 0, \ldots, d-1$ and $b_i \in \{0, 1\}$, and a linear recurrence over $\mathbb{Z}_2$

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_d a_{n-d},$$

for all $n \geq d$, i.e., each c_i is in $\{0, 1\}$ and addition is done modulo 2. The *minimum period* of a periodic sequence $\{a_n\}$ is the lowest $p \in \mathbb{N}$ such that $a_n = a_{n \bmod p}$ for every n. The maximum possible minimum period of a shift register sequence is $2^d - 1$,

and it is known that for each positive integer d there are shift register sequences of maximum period. These are known as *maximal length* or *pseudo-noise* sequences [31].

Given $d > 0$, let $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_d a_{n-d}$ define a maximum period shift register sequence. Let L_d be the language over a unary alphabet $\{\#\}$ defined by $\#^n \in L_d$ if and only if $a_n = 1$. We have the following:

Proposition 5.15. *The language L_d is accepted by a mod-2-MA with d states, but not by any IFA with fewer than $2^d - 1$ states.*

Proof. The existence of a mod-2-MA with d states accepting L_d is shown in [3, Lemma 11].

Consider the Hankel matrix over $\mathbb{R}$ of L_d. Since $\{a_n\}$ is $2^d - 1$ periodic, we have that the i'th row of the Hankel matrix is equal to the $i + 2^d - 1$'st row for any i, and similar for the columns. Hence, the rank (over $\mathbb{R}$) of the Hankel matrix is equal to the rank of the submatrix of size $(2^d - 1) \times (2^d - 1)$ in the top left corner. We will call this matrix H. Notice that $H_{i,j} = a_{i+j}$. The *auto-correlation* of $\{a_n\}$ is defined as

$$C(\tau) = \sum_{k=1}^{2^d-1} a_k a_{k+\tau},$$

for $\tau \geq 0$. By Equation 10 on page 82 in [31], we have that $C(\tau) = 2^{d-1}$ if $\tau = 0$ and $C(\tau) = 2^{d-2}$ otherwise. Consider H^2. If H^2 has full rank, then so does H. We have that

$$\begin{aligned}(H^2)_{i,j} &= \sum_{k=1}^{2^d-1} H_{i,k} H_{k,j} \\ &= \sum_{k=1}^{2^d-1} a_{i+k} a_{k+j} \\ &= \sum_{k=1}^{2^d-1} a_k a_{k+|j-i|} = C(|j-i|),\end{aligned}$$

where the indices are taken modulo $2^d - 1$. Hence, H^2 is the matrix with 2^{d-1} on the diagonal and 2^{d-2} elsewhere. We show that the matrix with $2^{-d+2} - 2^{-2d+2}$ on the diagonal and -2^{-2d+2} elsewhere is an inverse of H^2. Let $H' \in \mathbb{R}^{(2^d-1)\times(2^d-1)}$ as follows:

$$H'_{i,j} = \begin{cases} 2^{-d+2} - 2^{-2d+2} & \text{if } i = j \\ 2^{-2d+2} & \text{otherwise} \end{cases}$$

Now we have that $(H^2H')_{i,j} = 1$ if $i = j$:

$$\begin{aligned}(H^2H')_{i,i} &= \sum_{k=1}^{2^d-1} H^2_{i,k}H'_{k,i} \\ &= 2^{d-1}(2^{-d+2} - 2^{-2d+2}) + (2^d - 2)(2^{d-2}(-2^{-2d+2}) \\ &= 2^1 - 2^{-d+1} - 2^0 + 2^{-d+1} \\ &= 1\end{aligned}$$

We also have $(H^2H')_{i,j} = 0$ if $i \neq j$:

$$\begin{aligned}(H^2H')_{i,j} &= \sum_{k=1}^{2^d-1} H^2_{i,k}H'_{k,j} \\ &= 2^{d-1}(-2^{-2d+2}) + 2^{d-2}(2^{-d+2} - 2^{-2d+2}) + (2^d - 3)(2^{d-2}(-2^{-2d+2})) \\ &= -2^{-d+1} + 2^0 - 2^{-d} - 2^0 + 3 * 2^{-d} \\ &= 0\end{aligned}$$

Thus, H' is an inverse of H^2. Hence, H^2 and H have full rank and thus the Hankel matrix of L_d has rank $2^d - 1$. This means that the smallest IFA accepting L_d has $2^d - 1$ states. □

5.3 Image-binary Büchi Automata

5.3.1 Definitions

Let $\mathcal{A} = (Q, \Sigma, M, \alpha)$ be as in a weighted automaton over a field $\mathbb{F}$ and let F be a set of final states. We call $\mathcal{A}$ *ultimately stable* if for any $q, q' \in Q$ and $a \in \Sigma$ such that there exists a word w with $M(w)_{q',q} \neq 0$ (i.e., there is a path from q' to q over some word w), $M(a)_{q,q'} = 0$ or $M(a)_{q,q'} = 1$, meaning that any edges in a loop have weight 1. For any infinite word $w = w_0w_1 \ldots$ we call $q_0q_1 \ldots \in Q^\omega$ a *path* over w if $\alpha(q_0) \neq 0$ and for all i, $M(w_i)_{q_i,q_{i+1}} \neq 0$. We call $q_0q_1 \ldots$ a *final path* if $\mathit{infinite}(q_0q_1 \ldots) \cap F \neq \emptyset$, where $\mathit{infinite}(q_0q_1 \ldots)$ denotes the set of states in Q that occur infinitely often in the path. We will write $\mathit{FinalPaths}_{\mathcal{A}}(w)$ to denote the set of final paths of an automaton $\mathcal{A}$ over a word w.

It is clear that for any path $q_0q_1 \ldots$ over a word $w = w_0w_1 \ldots$ there exists an i such that for any $j \geq i$, q_j lies on a loop, and therefore we can define the weight of the path $q_0q_1 \ldots$ over w to be $\lim_{i\to\infty} \prod_{n\leq i} M(w_n)_{q_n,q_{n+1}}$, denoted by $\mathit{weight}(q_0q_1 \ldots, w)$. If w is clear from context, we may simply write $\mathit{weight}(q_0q_1 \ldots)$. For any word w with finitely many final paths, we define the weight of w to be

the sum of the weights of the final paths over w, denoted by $L_{\mathcal{A}}(w)$. We call $\mathcal{A} = (Q, \Sigma, M, \alpha, F)$ an *image-binary (weighted) Büchi automaton (IBA)* if it is ultimately stable, there exists a bound $N \in \mathbb{N}$ such that $|FinalPaths_{\mathcal{A}}(w)| \leq N$ for any word w, and $L_{\mathcal{A}}(\Sigma^\omega) \subseteq \{0, 1\}$, i.e. for all $w \in \Sigma^\omega$, $L_{\mathcal{A}}(w) \in \{0, 1\}$. As in the finite word case, we view image-binary Büchi automata as language acceptors and say that a word w is accepted if $L_{\mathcal{A}}(w) = 1$.

If an IBA $\mathcal{A}$ is such that $\alpha \in \{0, 1\}^Q$ and $M(a) \in \{0, 1\}^{Q \times Q}$ for all $a \in \Sigma$, then $\mathcal{A}$ is called an *unambiguous Büchi automaton (UBA)*. Similarly to the finite word case, we note that this definition of a UBA is essentially equivalent to the classical one, which says that a UBA is an NBA (non-deterministic Büchi automaton) where each word has at most 1 final path. We also see that a *deterministic Büchi automaton (DBA)* essentially is a special case of a UBA, and hence of an IBA.

We will use the following notation: given a finite sequence $a = a_1 a_2 \dots a_n$, we will write $last(a)$ to denote a_n. Given a (possibly finite) sequence $a = a_1 a_2 \dots$ and a character a_0 we will write $a_0 \cdot a$ to denote the concatenation $a_0 a_1 a_2 \dots$. We will write $\mathbb{B}$ to denote the two element set $\{\bot, \top\}$ and for any set S we will write $\mathcal{P}^{\leq k}(S)$ to denote the set $\{S' \subseteq S \mid |S'| \leq k\}$. For ease of notation we will treat $\mathbb{B}$ as the set of true ($\top$) and false ($\bot$) predicates, on which the standard Boolean operations apply.

5.3.2 IBAs and k-Ambiguous NBAs

In this section we will compare image-binary Büchi automata to k-ambiguous NBAs. This will allow us later to employ partial disambiguation techniques in order to do model checking on general NBAs. First observe that k-ABAs can be exponentially more succinct than equivalent IBAs:

Lemma 5.16. *Let $\mathcal{A}$ be a k-ABA with n states. The minimal IBA accepting the same language as $\mathcal{A}$ may require at least 2^n states, even if $k = n$.*

Proof. By Proposition 5.12, there exists a family of n-ambiguous NFAs with exponentially fewer states than the minimal IFAs accepting the same languages. Let $\mathcal{A}$ be such an NFA on n states, and let $L \subseteq \Sigma^*$ be its language. Consider the language $(L)_\$$ given by $w \in (L)_\$$ if and only if w can be decomposed as $u\$v$, where \$ does not occur in Σ and $u \in L$. We can create an n-ABA that accepts this language with $n + 1$ states by adding a \$-labeled arrow from the final states of $\mathcal{A}$ to an accepting sink state. However, let $\mathcal{A}'$ be any IBA accepting $(L)_\$$. Let $\$w$ be any infinite word starting in \$, and let $\eta_q = L_{\mathcal{A}'[q]}(\$w)$. Let $\mathcal{A}''$ be IFA defined as follows: the state set of $\mathcal{A}''$ is equal to the state set of $\mathcal{A}'$, the alphabet is the alphabet of $\mathcal{A}$

(i.e. the alphabet of $\mathcal{A}'$ without \$), there exists an a-edge between two states q, q' if and only if there exists an a-edge between q and q' in $\mathcal{A}'$, and the final vector is given by η. We claim that $\mathcal{A}''$ accepts L. Indeed, we have $L_{\mathcal{A}''}(u) = L_{\mathcal{A}'}(u\$w)$. Hence, $\mathcal{A}'$ has at least 2^n states, because otherwise $\mathcal{A}''$ would be an IFA with fewer than 2^n states accepting L. □

The rest of this section will be dedicated to converting k-ABAs to equivalent IBAs, resulting in an IBA with at most a singly exponential state set size blowup.

By the binomial theorem, $(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^i$, and hence, setting $x = -1$, $1 = 1 - \sum_{i=0}^{n} (-1)^i \binom{n}{i} = 1 - (-1)^0 \binom{n}{0} - \sum_{i=1}^{n} (-1)^i \binom{n}{i} = \sum_{i=1}^{n} (-1)^{i-1} \binom{n}{i}$. Hence, for any set S, $\sum_{S' \in \mathcal{P}(S) \setminus \{\emptyset\}} (-1)^{|S'|-1} = \sum_{i=1}^{|S|} (-1)^{i-1} \binom{|S|}{i} = 1$. Let R be the set of final paths of $\mathcal{A}_k$ over a word w. We can design an (infinite state) IBA $\mathcal{A}'_k = (Q', \Sigma, \Delta', \alpha, F')$ where final paths correspond to subsets of R, and where for each $R' \subseteq R$, the final path corresponding to R' has weight $(-1)^{|R'|-1}$. This IBA is given as follows:

- $Q' = \mathcal{P}^{\leq k}(Q^* \times \mathbb{B}) \setminus \{\emptyset\}$,
- $\alpha_P = (-1)^{|P|-1}$ for each $P \in \mathcal{P}(Q_0 \times \{\bot\}) \setminus \{\emptyset\}$ and $\alpha_P = 0$ otherwise,
- $F' = \mathcal{P}^{\leq k}(Q^* \times \{\top\}) \setminus \{\emptyset\}$, and
- for any $P \in Q'$, let $b'' = \bot$ if for all $(r, b) \in P$, $b = \top$, and let $b'' = \top$ otherwise. Let P' be such that:
  - For any $(r, b) \in P$, there exists $(r', b') \in P'$ and $q \in \delta(last(r), a)$ such that $r' = r \cdot q$ and $b' = ((last(r) \in F) \vee b) \wedge b''$, and
  - For any $(r', b') \in P'$, there exists $(r, b) \in P$ and $q \in \delta(last(r), a)$ such that $r' = r \cdot q$ and $b' = ((last(r) \in F) \vee b) \wedge b''$.

  Then $\Delta(a)_{P,P'} = (-1)^{|P'|-|P|}$. For any other P', $\Delta(a)_{P,P'} = 0$.

Intuitively, the bit b in a (r, b)-tuple flips to true every time r reaches a final state, and back to false every time all the prefixes have reached a final state. This ensures that every sequence of prefixes in a final path of $\mathcal{A}'_k$ visits final states infinitely often. This technique mirrors for instance Safra's determinisation construction [61].

In Figure 5.1 we give an example of a 4-ambiguous automaton over a unary alphabet, together with the corresponding infinite $\mathcal{A}'_k$.

Lemma 5.17. *$\mathcal{A}'_k$ is an infinite-state IBA equivalent to $\mathcal{A}_k$.*

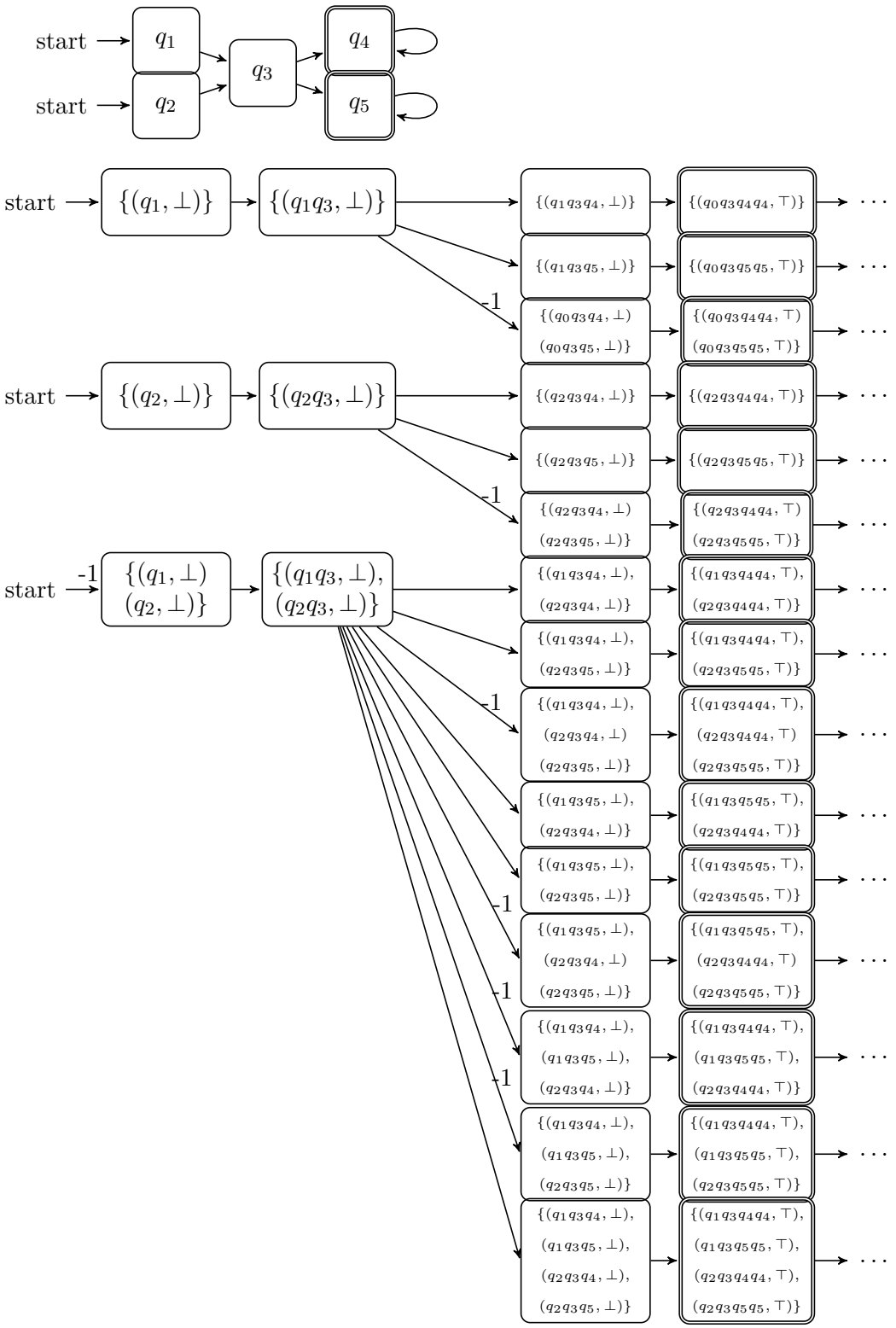


Figure 5.1: An example 4-ambiguous automaton (above) and its infinite IBA counterpart (below). In the IBA, the weights of the unlabeled edges are 1.

Proof. For $\mathcal{A}'_k$ to be an IBA equivalent to $\mathcal{A}_k$, it needs to satisfy three properties - the edges of $\mathcal{A}'_k$ over loops have weight 1, any word w has at most N final paths for some global bound N, and the sum of the weights of final paths over w is 1 if w is accepted by $\mathcal{A}_k$ and 0 otherwise. Since states in a path of $\mathcal{A}'_k$ consist of sets of prefixes of increasing length, we see that $\mathcal{A}'_k$ does not have any loops and hence the first property is trivially true. For the second and third properties, let w be any fixed word.

Suppose w is not accepted by $\mathcal{A}_k$, then by König's lemma there exists a bound m such that any path of $\mathcal{A}_k$ over w has at most m occurences of final states. Let $\rho_1\rho_2\dots\rho_i$ be any prefix of a path of $\mathcal{A}'_k$ over w. Since for any $(r,b) \in \rho_i$, r has at most m occurrences of final states, and all the bits are flipped to false any time every prefix in $\rho_1\rho_2\dots\rho_i$ has visited a final state, we see that $\rho_1\rho_2\dots\rho_i$ also visits final states at most m times. Therefore, $\mathcal{A}'_k$ has no final paths (and hence the sum of the weights is 0) and the second and third properties are true.

Suppose, then, that w is accepted by $\mathcal{A}_k$. We will show that final paths of $\mathcal{A}'_k$ over w correspond exactly to sequences of prefixes of final paths of $\mathcal{A}_k$ over w. Let $\rho_1\rho_2\dots$ be a path of $\mathcal{A}'_k$ over w. Suppose that for some i there exists $(r,b) \in \rho_i$ such that r is not a prefix of a final path of $\mathcal{A}_k$ over w. Let r be the shortest such prefix. Then for large enough m and any $m' > m$ there exists $(r', \bot) \in \rho_{m'}$ where r is a prefix of r'. Hence, $\rho_{m'} \notin F'$ and $\rho_1\rho_2\dots$ is not a final path.

Finally, let $\rho_1\rho_2\dots$ be a path of $\mathcal{A}'_k$ over w such that for every i and every $(r,b) \in \rho_i$, r is a prefix of a final path of $\mathcal{A}_k$ over w. Let R be the set of those paths ρ' of $\mathcal{A}_k$ over w such that for all i there exists a tuple $(r,b) \in \rho_i$ where r is a prefix of ρ'. We have that for any i and any $(r,b) \in \rho_i$ there exists $(r',b') \in \rho_{i+1}$ such that $r' = r \cdot q$ with $q \in \delta(last(r), w_i)$ and for any $(r',b') \in \rho_{i+1}$ there exists $(r,b) \in \rho_i$ such that $r' = r \cdot q$ with $q \in \delta(last(r), w_i)$. Therefore R is non-empty and ρ_i is precisely the set of prefixes of length i of paths in R (paired with some boolean b). Hence, paths $\rho_1\rho_2\dots$ in $\mathcal{A}'_k$ over w correspond uniquely to sets of paths of $\mathcal{A}_k$ over w.

We claim that R is a set of final paths, and that the weight of $\rho_1\rho_2\dots$ is $(-1)^{|R|-1}$. Indeed, suppose $\rho'_1\rho'_2\dots \in R$ is not a final path. Since for any i, $\rho'_1\rho'_2\dots\rho'_i$ prefixes a final path, there exists $j > i$ such that $\rho'_1\rho'_2\dots\rho'_i$ is a prefix of a final path $\rho''_1\rho''_2\dots$, and $\rho''_j \neq \rho'_j$. Hence $\mathcal{A}_k$ has infinitely many final paths over w, which contradicts its k-ambiguity. Moreover, the weight of any prefix of length i of $\rho_1\rho_2\dots$ is $(-1)^{|\rho_1|-1}\prod_j(-1)^{|\rho_{j+1}|-|\rho_j|} = (-1)^{|\rho_i|-1}$. Since the size of ρ_i is non-decreasing and bounded from above by k, and for any i, ρ_i contains prefixes of length i of paths in

R, for large enough i we have that $|\rho_i| = |R|$, and hence the weight of $\rho_1\rho_2\ldots$ is $(-1)^{|R|-1}$.

Thus, final paths in $\mathcal{A}'_k$ over w correspond uniquely to sets of final paths in $\mathcal{A}_k$ over w, and hence there are at most 2^k final paths in $\mathcal{A}'_k$ over w. Moreover, since by the binomial theorem we have $\sum_{i=1}^{|S|}(-1)^{i-1}\binom{|S|}{i} = 1$ for any set S, and any path in $\mathcal{A}'_k$ corresponding to a set R of final paths in $\mathcal{A}_k$ has weight $(-1)^{|R|-1}$, we see that the sum of the weights of final paths of $\mathcal{A}'_k$ over w is precisely equal to 1 if and only if $\mathcal{A}_k$ has an accepting path over w. □

We will construct a finite IBA called the k-disambiguation of $\mathcal{A}_k$ based on $\mathcal{A}'_k$ that accepts the same language as $\mathcal{A}_k$.

Let $\pi_{last} : Q' \to [k]^{Q\times\mathbb{B}}$ be defined as $\pi_{last}(P)_{q,b} = |\{(r,b) \in P \mid last(r) = q\}|$. We extend π_{last} over finite and infinite sequences of elements of Q' in the natural way: $\pi_{last}(P_1P_2\ldots) = \pi_{last}(P_1)\cdot\pi_{last}(P_2\ldots)$.

Lemma 5.18. *Let $\rho = \rho_1\rho_2\ldots \in (Q')^\omega$ be a path of $\mathcal{A}'_k$ over a word w and let $\rho'_1\rho'_2\ldots = \pi_{last}(\rho)$. Then $\rho_1\rho_2\ldots$ is final if and only if for infinitely i, $(\rho'_i)_{q,\bot} = 0$ for all $q \in Q$.*

Proof. Trivial. □

Paths in our k-disambiguation will be sequences in $([k]^{Q\times\mathbb{B}})^\omega$ such that there exist paths in $\mathcal{A}'_k$ that map to that sequence. However, there is not a one-to-one correspondence between sequences over $[k]^{Q\times\mathbb{B}}$ and paths in $\mathcal{A}'_k$: in Figure 5.1, for instance, both $\{(q_1,\bot),(q_2,\bot)\}\ \{(q_1q_3,\bot),(q_2q_3,\bot)\}\ \{(q_1q_3q_4,\bot),(q_2q_3q_5,\bot)\}$ $\{(q_1q_3q_4q_4,\top),(q_2q_3q_5q_5,\top)\}\ldots$ and $\{(q_1,\bot),(q_2,\bot)\}\ \{(q_1q_3,\bot),(q_2q_3,\bot)\}$ $\{(q_1q_3q_5,\bot),(q_2q_3q_4,\bot)\}\ \{(q_1q_3q_5q_5,\top),(q_2q_3q_4q_4,\top)\}\ldots$ map to the sequence $(1,1,0,0,0,0,0,0,0,0)^\top\ (0,0,2,0,0,0,0,0,0,0)^\top\ ((0,0,0,1,1,0,0,0,0,0)^\top$ $(0,0,0,0,0,0,0,0,1,1)^\top)^\omega$, where the order of the vector components is given by $((q_1,\bot),(q_2,\bot),(q_3,\bot),(q_4,\bot),(q_5,\bot),(q_1,\top),(q_2,\top),(q_3,\top),(q_4,\top),(q_5,\top))$.

Fix any $\vec{r},\vec{r'} \in [k]^{Q\times\mathbb{B}}$ and $a \in \Sigma$. Let P be any state in Q' such that $\pi_{last}(P) = \vec{r}$. As it turns out, the number of states P' with $\pi_{last}(P') = \vec{r'}$ where P' is an a-successor of P only depends on $\vec{r}$, a, and $\vec{r'}$. We call this number $w(\vec{r},a,\vec{r'})$.

Lemma 5.19. *The number $w(\vec{r},a,\vec{r'})$ is unique and at most exponential in k.*

Proof. Given $\vec{r},\vec{r'} \in [k]^{Q\times\mathcal{B}}$ and $a \in \Sigma$, let

$$C = \{f : Q\times\mathcal{B}\times[k] \to \mathcal{P}(Q) \mid \forall(q,b,i) \in Q\times\mathcal{B}\times[k] : ((i > \vec{r}_{q,b}) \to f(q,b,i) = \emptyset) \wedge ((i \le \vec{r}_{q,b}) \to f(q,b,i) \in \mathcal{P}(\delta(q,a)) \setminus \{\emptyset\})\}.$$

Intuitively, functions in C map the final states of prefixes in a state P with $\pi_{last}(P) = \vec{r}$ to non-empty sets of successor states. Let $b'' = \bot$ if $b = \top$ for all $(r, b) \in P$, and let $b'' = \top$ otherwise. Let C' be the set of those $f \in C$ such that for any $(q', b') \in Q \times \mathcal{B}$, $|\{(q, b, i) \in Q \times \mathcal{B} \times [k] \mid (b' = (q \in F \vee b) \wedge b'') \wedge q' \in f(q, b, i)\}| = \vec{r'}_{q',b'}$. That is, C' consists of those $f \in C$ for which $\vec{r'}$ can be considered to represent a multiset containing the image of f. We claim that $w(\vec{r}, a, \vec{r'}) = |C'|$.

Fix an ordering on Q^*. Since $\pi_{last}(P) = \vec{r}$, we see that for any $q \in Q, b \in \mathcal{B}$ there are $\vec{r}_{q,b}$ tuples (s, b) in P where $last(s) = q$. Let $s_{q,b,i}$ denote the i'th (under the ordering on Q^*) path in P that is paired with bit b and ends in state q. Then for any $f \in C$, $f(q, b, i)$ is a non-empty set of a-successors of q. Let $b'' = \bot$ if $b' = \top$ for every $(s', b') \in P$ and let $b'' = \bot$ otherwise. If $q \in F$, let $b' = \top \wedge b''$, and otherwise let $b' = b \wedge b''$. Hence, for any $f \in C$ the following is an a-successor of P in $\mathcal{A}'_k$:

$$P' = \{(s_{q,b,i} \cdot q', b') \mid (s_{q,b,i}, b) \in P \wedge q' \in f(q, b, i)\}.$$

If $f \in C'$, then by definition of C', for any q', b', $\pi_{last}(P')_{q',b'} = \vec{r'}_{q',b'}$, and hence $\pi_{last}(P') = \vec{r'}$. Since each $f \in C$ gives rise to a unique successor, this means P has $|C'|$ a-successors P' with $\pi_{last}(P') = \vec{r'}$. Moreover, $|C'| \leq |C| \leq 2^{2k|Q|^2}$. □

For a state $\vec{r} \in Q''$ we will write $size(\vec{r}) = \sum_{(q,b)} \vec{r}_{q,b}$ to denote the size of $\vec{r}$. We define the IBA $k\text{-dis}'(\mathcal{A}_k) = (Q'', \Sigma, \Delta'', \alpha', F'')$ where:

- $Q'' = ([k]^{Q \times \mathbb{B}}) \setminus \{\vec{0}\}$,
- $\Delta''(a)_{\vec{r},\vec{r'}} = (-1)^{size(\vec{r'}) - size(\vec{r})} w(\vec{r}, a, \vec{r'})$
- $\alpha'_{\vec{r}} = (-1)^{size(\vec{r})-1}$ for every $\vec{r}$ with:
  - $\vec{r}_{q,b} = 0$ if $q \notin Q_0$ or $b = \top$, and
  - $\vec{r}_{q,b} \leq 1$ otherwise.

  $\alpha'_{\vec{r}} = 0$ otherwise.
- $F'' = \{\vec{r} \in Q'' \mid \forall (q, \bot) \in Q'' : \vec{r}_{q,\bot} = 0\}$.

The IBA $k\text{-dis}(\mathcal{A}_k)$ (the k-disambiguation of $\mathcal{A}_k$) is then defined as $k\text{-dis}'(\mathcal{A}_k)$ restricted to those reachable states in Q'' that can reach a loop over a final state. This trimness condition helps the proofs later on, but could be omitted. In Figure 5.2 we see the k-disambiguation of the automaton in Figure 5.1.

The weights in $k\text{-dis}(\mathcal{A}_k)$ count the number of equivalent paths in $\mathcal{A}'_k$, where two paths $\rho, \rho' \in (Q')^\omega$ are equivalent if $\pi_{last}(\rho) = \pi_{last}(\rho')$:

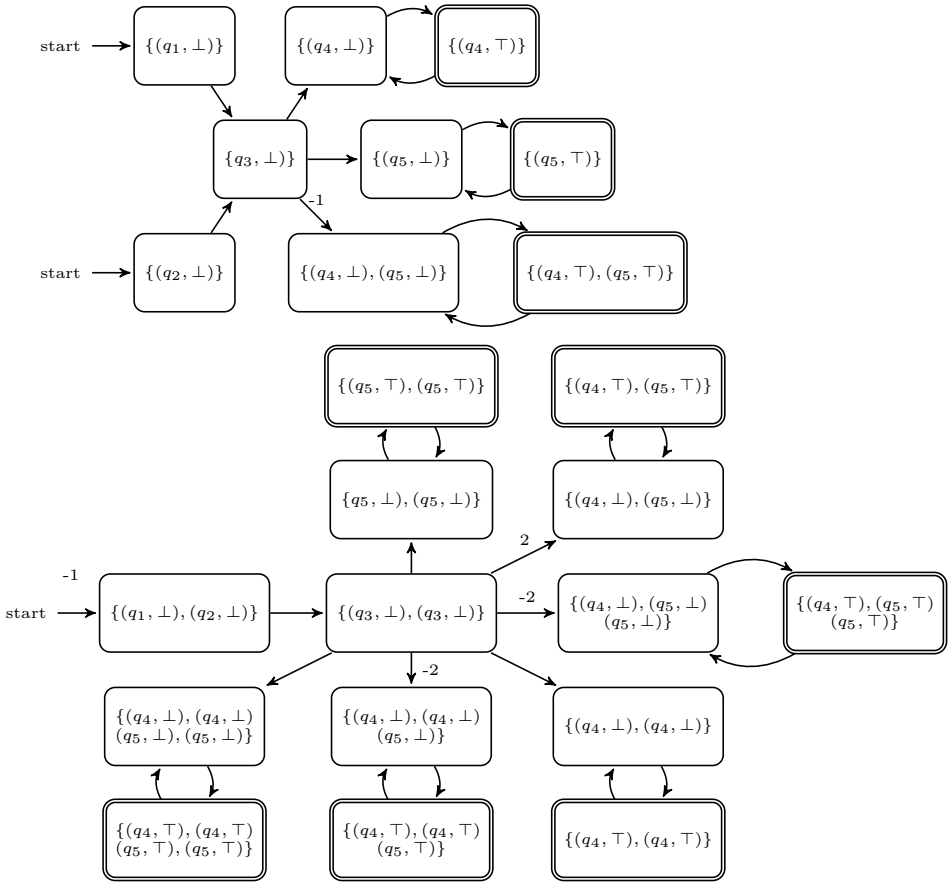


Figure 5.2: The k-disambiguation of the automaton in Figure 5.1. The elements of $[k]^{Q\times\mathbb{B}}$ are represented by multisets. Unlabeled edges have weight 1.

Lemma 5.20. *Let $\rho \in (Q'')^\omega$ be a path in k-dis$(\mathcal{A}_k)$ over a word w. Let R be the set of those paths ρ' in $\mathcal{A}'_k$ over w such that $\pi_{last}(\rho') = \rho$, and let $n = \max_i size(\rho_i)$. Then $weight(\rho) = (-1)^{n-1}|R|$.*

Proof. Let R_i be the set of prefixes of length i of paths in R. Let ρ'_i be any path in R_i. By Lemma 5.19, ρ'_i has $w(\rho_i, w_i, \rho_{i+1})$ w_i-successors that map to ρ_{i+1} under π_{last}. Hence, $|R_{i+1}| = w(\rho_i, w_i, \rho_{i+1})|R_i|$ and in particular, $|R| = \lim_{n\to\infty} \prod_{i=1}^{n} w(\rho_i, w_i, \rho_{i+1})$. Moreover, $\lim_{n\to\infty}(-1)^{size(\rho_1)-1} \prod_{i=1}^{n}(-1)^{size(\rho_{i+1})-size(\rho_i)} = (-1)^{n-1}$, and hence $weight(\rho) = \lim_{n\to\infty}(-1)^{size(\rho_1)-1} \prod_{i=1}^{n}(-1)^{size(\rho_{i+1})-size(\rho_i)} w(\rho_i, w_i, \rho_{i+1}) = (-1)^{n-1}|R|$. □

Since by Lemma 5.18, a path ρ in k-dis$(\mathcal{A}_k)$ is final whenever any path ρ' in $\mathcal{A}'_k$ with $\pi_{last}(\rho') = \rho$ is, this means $L_{k\text{-dis}(\mathcal{A}_k)} = L_{\mathcal{A}'_k}$. This proves the final result:

Theorem 5.21. *k-dis$(\mathcal{A}_k)$ is an IBA that accepts the same language as $\mathcal{A}_k$.*

Proof. We will prove equivalence between $\mathcal{A}'_k$ and k-dis$(\mathcal{A}_k)$. Combined with Lemma 5.17, this gives us the required result. As before, we need to show that the edges

of k-dis($\mathcal{A}_k$) over loops have weight 1, any word w has at most N final paths for some global bound N, and the sum of the weights of final paths over w is 1 if w is accepted by $\mathcal{A}'_k$ and 0 otherwise.

Pick any loop in k-dis($\mathcal{A}_k$) and any a-edge from any $\vec{r}$ to any $\vec{r'}$ on that loop. Note that $\Delta(a)_{\vec{r},\vec{r'}} > 0$ implies that $size(\vec{r}) \leq size(\vec{r'})$ and hence since they are on a loop, $size(\vec{r}) = size(\vec{r'})$. Let $w_1 \dots w_m$ be such that there exists a path from an initial state to $\vec{r}$, and let $w_{m+1} \dots w_n$ be a word that traverses the loop starting with this edge from $\vec{r}$ to $\vec{r'}$ (hence, $w_{m+1} = a$). Finally let $w_{n+1}w_{n+2} \dots$ be a word such that $w_1 \dots w_m(w_{m+1} \dots w_n)^C w_{n+1} \dots$ has a final path that traverses the loop over $\vec{r}$ at least C times, such a word exists since every state can reach a loop over a final state. Note that weights in k-dis($\mathcal{A}_k$) are integers. Then the absolute weight of this path is at least $w(\vec{r}, w_{m+1}, \vec{r'})^C$. Using Lemmas 5.20 and 5.18, then, we see that there exist at least $w(\vec{r}, w_{m+1}, \vec{r'})^C$ final paths over $w_1 \dots w_m(w_{m+1} \dots w_n)^C w_{n+1} \dots$ in $\mathcal{A}'_k$. Since this is the case for any C, and the number of final paths over any word in $\mathcal{A}'_k$ is bounded above, we must have that $w(\vec{r}, w_{m+1}, \vec{r'}) = 1$ and hence $\Delta(a)_{\vec{r},\vec{r'}} = 1$.

Now let w be any word. By Lemma 5.18 a path in $\mathcal{A}'_k$ is final if and only if the path in k-dis($\mathcal{A}_k$) it maps to under π_{last} is final, and by Lemma 5.20 any path with non-zero weight (i.e. any path) has at least one path in $\mathcal{A}'_k$ mapping to it. Hence we see that k-dis($\mathcal{A}_k$) has at most as many final paths over w as $\mathcal{A}'_k$ does, and since the number of final paths in $\mathcal{A}'_k$ is globally bounded, so is the number of final paths in k-dis($\mathcal{A}_k$).

Let w be any word and define the equivalence relation $=_\pi \subseteq Q'^\omega \times Q'^\omega$ as $\rho =_\pi \rho'$ iff $\pi_{last}(\rho) = \pi_{last}(\rho')$ (note that $\pi_{last}(\rho) \in ([k]^{Q \times \mathbb{B}})^\omega$). Note that this equivalence relation partitions the final paths of $\mathcal{A}'_k$ over w, and for any ρ, ρ' with $\rho =_\pi \rho'$, $weight(\rho) = weight(\rho') = (-1)^{\lim_{i\to\infty} |\rho_i|}$. Each equivalence class can be represented by a path in k-dis($\mathcal{A}_k$). Let ρ' be the representative of the equivalence class containing ρ, and let R be the size of this equivalence class. By Lemma 5.20, $weight(\rho') = weight(\rho)R$. Hence, the sum of the weights of final paths in k-dis($\mathcal{A}_k$) is equal to the sum of the weights of all the equivalence classes of $=_\pi$ where the representative is a final path, which by Lemma 5.18 is precisely the sum of the weights of final paths in $\mathcal{A}'_k$.

Hence, the weight of a word w in k-dis($\mathcal{A}_k$) is equal to the weight of w in $\mathcal{A}'_k$. This means k-dis($\mathcal{A}_k$) is equivalent to $\mathcal{A}'_k$ which is in turn equivalent to $\mathcal{A}_k$, concluding the proof. □

Theorem 5.22. *Given a k-ambiguous automaton $\mathcal{A}_k$ with n states, the disambiguation k-dis$(\mathcal{A}_k)$ has at most k^{2n} states. Moreover, k-dis$(\mathcal{A}_k)$ can be calculated using a PSPACE transducer.*

Proof. The number of states follows from the size of $[k]^{Q\times\mathcal{B}}$. Note that k is at most singly exponential in the size of the state set of the automaton, which follows from for instance [71, Theorem 2.1], or the fact that finite ambiguity implies the non-existence of diamonds on a loop. This means that k^{2n} is also singly exponential in n. Hence, we can iterate over every element in Q'', each of which is a vector in $[k]^{Q\times\mathcal{B}}$ which can be represented in polynomial space. Given $\vec{r}, \vec{r'} \in Q''$ and $a \in \Sigma$, to calculate $\Delta''(a)_{\vec{r},\vec{r'}}$ we need to calculate $(-1)^{size(\vec{r'})-size(\vec{r})}$ and $w(\vec{r}, a, \vec{r'})$. Clearly the former can be calculated in polynomial space. For the latter, notice that $w(\vec{r}, a, \vec{r'})$ can be doubly exponential, meaning that there can be doubly exponentially many edges between any P and P' with $\pi_{last}(P) = \vec{r}$ and $\pi_{last}(P') = \vec{r'}$. Hence, we cannot simply list the edges between states in $\mathcal{A}'_k$ to calculate $w(\vec{r}, a, \vec{r'})$. However, we have the following lemma:

Lemma 5.23. *Let $w(\vec{r}, a, \vec{r'})$ as defined above, let $b'' = \bot$ if $\vec{r}_{q,\bot} = 0$ for every $q \in Q$ and $b'' = \top$ otherwise. For each $(q, b) \in Q \times \mathcal{B}$ let $C_{q,b}$ denote the set of those $\vec{r'}[q, b] \in [k]^{Q\times\mathcal{B}}$ such that*

$$size(\vec{r'}[q,b]) \geq \vec{r}_{q,b} \quad \wedge \quad \forall (q',b') \in Q \times \mathcal{B} \,.\, \vec{r'}[q,b]_{q',b'} \leq \vec{r}_{q,b} \quad \wedge$$
$$\forall (q',b') \in Q \times \mathcal{B} \,.\, ((b' \neq (q \in F \vee b) \wedge b'') \vee q' \notin \delta(q,a)) \rightarrow \vec{r'}[q,b]_{q',b'} = 0\,.$$

Intuitively, $C_{q,b}$ contains the possible multisets of successors of the $\vec{r}_{q,b}$ prefixes ending in q with bit b. Let $\#succ(q, n, \vec{r'}[q, b]) = \sum_{j=0}^{n}(-1)^j \binom{n}{j} \left(\prod_{q',b'} \binom{n-j}{\vec{r'}[q,b]_{q',b'}}\right)$. Intuitively, $\#succ$ counts the number of ways $\vec{r'}[q, b]$ can succeed the $\vec{r}_{q,b}$ prefixes ending in q with bit b. Then

$$w(\vec{r}, a, \vec{r'}) = \sum_{f:Q\times\mathcal{B}\to C_{q,b} \mid \sum_{q,b} f(q,b) = \vec{r'}} \prod_{q,b} \#succ(q, \vec{r}_{q,b}, f(q,b)).$$

Proof. We have that $w(\vec{r}, a, \vec{r'}) = |C'|$, where C' is defined as in Lemma 5.19. We define the following equivalence relation: two elements $f, f' \in C'$ are equivalent if for every $(q, b) \in Q \times \mathcal{B}$ we have that $\sum_i [\![f(q, b, i)]\!] = \sum_i [\![f'(q, b, i)]\!]$, where for any set S, $[\![S]\!]$ denotes the characteristic vector of S.

Let $\mathcal{G} := \{g : Q \times \mathcal{B} \to C_{q,b} \mid \sum_{q,b} g(q, b) = \vec{r'}\}$. Clearly the equivalence classes of C' can be characterised by elements of $\mathcal{G}$. We can view any function $f \in C'$ as a way of distributing non-empty successor sets over the numbered elements in $\vec{r}$ in

such a way that it adds up to $\vec{r'}$. Hence, for any $g \in \mathcal{G}$, the number of elements in the equivalence class of g is given by those functions that, for every $(q, b) \in Q \times \mathcal{B}$, distribute the elements in $g(q, b)$ to the $\vec{r}_{q,b}$ paths ending in (q, b). These distributions are independent, and hence the number of elements in the equivalence class of g is the product over all (q, b) of the number of ways to distribute the elements in $g(q, b)$.

This number is captured by $\#succ(q, \vec{r}, g(q, b))$. We will explain $\#succ$ as a balls-and-bins problem. We need to assign a non-empty successor set to each of the $\vec{r}_{q,b}$ paths ending in (q, b) such that for each $(q', b') \in Q \times \mathcal{B}$ there are $g(q, b)_{q',b'}$ sets containing (q', b'). This is equivalent to dividing $\sum_{q',b'} g(q, b)_{q',b'}$ coloured balls, with colours in $Q \times \mathcal{B}$, over $\vec{r}_{q,b}$ bins in such a way that no bin has two balls of the same colour and no bin is empty. We can express the number of distributions where bins may be empty by distributing the balls per colour independently, this is given by $\prod_{q',b'} \binom{n}{g(q,b)_{q',b'}}$ where n is the number of bins (i.e. $\vec{r}_{q,b}$). Using the inclusion-exclusion principle, then, we can add the restriction that no bins may be empty by including/excluding sets of empty bins. This gives us the formula $\#succ(q, \vec{r}, g(q, b)) = \sum_{j=0}^{n} (-1)^j \binom{n}{j} \left(\prod_{q',b'} \binom{n-j}{\vec{r'}[q,b]_{q',b'}} \right)$.

Hence, the number of elements in the equivalence class of g is given by $\prod_{q,b} \#succ(q, \vec{r}_{q,b}, g(q, b))$ and therefore we have that

$$w(\vec{r}, a, \vec{r'}) = |C'| = \sum_{g \in \mathcal{G}} \prod_{q,b} \#succ(q, \vec{r}_{q,b}, g(q, b)).$$

□

Using this lemma,, we can enumerate every vector in $C_{q,b}$ for every (q, b)-pair, which is a polynomial combination of polynomially sized objects. We can then calculate $\#succ$ with a PSPACE transducer to find $w(\vec{r}, a, \vec{r'})$. Hence, we can construct k-dis$(\mathcal{A}_k)$ with a PSPACE transducer. □

By Lemma 5.16, we have an $O(2^n)$ lower bound even if $k = n$. From Theorem 5.22 we already have a $2^{O(n \log n)}$ upper bound, leaving only a small gap.

When the ambiguity is comparatively low, we can do even better:

Theorem 5.24. *Given a k-ambiguous automaton $\mathcal{A}_k$ with n states where $k = O(\log n)$, the disambiguation k-dis$(\mathcal{A}_k)$ has at most $(2n)^{O(\log n)}$ states. Moreover, k-dis$(\mathcal{A}_k)$ can be calculated using a POLYLOGSPACE transducer.*

Proof. While in general, elements in $[\log n]^{[n] \times \mathbb{B}}$ take $2n \log n$ space to represent, we know that for any $\vec{r} \in Q''$, $\sum_i \vec{r}_i \leq \log n$ and hence we can instead represent

elements in Q'' as vectors in $([n] \times \mathbb{B})^{\log n}$, which take $\log(2n^{\log n}) = O((\log n)^2)$ space. Moreover, $w(\vec{r}, a, \vec{r'})$ is at most exponential in the ambiguity of $\mathcal{A}_k$ and hence we can simply enumerate the successors of any $P \in Q'$ with $\pi_{last}(P) = \vec{r}$ in POLYLOGSPACE. Thus, we can calculate k-dis$(\mathcal{A}_k)$ in POLYLOGSPACE. □

5.3.3 Model Checking IBA

In this section we will consider the problem of model checking IBAs against Markov chains. Since IBAs associate with each word $w \in \Sigma^\omega$ a weight in $\{0, 1\}$, we can again consider the problem of calculating the probability that an infinite word generated by a Markov chain is accepted by a given IBA. This is the model checking problem for IBAs against Markov chains.

We show that model checking IBAs against MCs can be done in NC using a modified procedure for model checking UBAs from Chapter 3 and [7]. This is the main theorem:

Theorem 5.25. *Let $\mathcal{M}$ be an MC and let $\mathcal{A}$ be an IBA. The probability that a random word sampled from $\mathcal{M}$ is in $L_\mathcal{A}$ can be computed in NC.*

We will now outline the model checking procedure. As in Chapter 3 the algorithm will consist of solving a system of linear equations, split up in two parts. The first part is the basic linear system, which we will describe below, while the second part will add normalising equations in the form of cuts (or pseudo-cuts, if Chapter 4 is followed). Since the weights of the edges within strongly connected components are all on loops, and therefore all have weight 1, this part can be copied verbatim from Chapter 3. The difference is in the basic system of equations.

Given an automaton $\mathcal{A} = (Q, \Sigma, M, \alpha, F)$ and Markov chain $\mathcal{M} = (S, P)$, we will write $\mathcal{A}[q]$ (resp. $\mathcal{M}[s]$) to denote the weighted automaton (resp. Markov chain) starting in $q \in Q$ (resp. $s \in S$). W.l.o.g. we will assume that every $q \in Q$ is reachable from some q' with $\alpha(q') \neq 0$ and every $q \in Q$ can reach a loop over a final state. Note that while $\mathcal{A}$ is an IBA, $\mathcal{A}[q]$ does not necessarily have to be. We will write $\mathbb{E}_\mathcal{M} L_\mathcal{A}$ for the expected weight of a word in $\mathcal{A}$ for a random word generated by $\mathcal{M}$. Since the weight of a word in an IBA is either 0 or 1, we see that $\mathbb{E}_\mathcal{M} L_\mathcal{A} = Pr_\mathcal{M}(L_\mathcal{A})$ if $\mathcal{A}$ is an IBA.

Lemma 5.26. *Let $\mathcal{A} = (Q, \Sigma, M, \alpha, F)$ be an IBA w.r.t. Markov chain $\mathcal{M} = (S, P)$ with initial distribution ι. The following equations hold:*

$$Pr_{\mathcal{M}}(L_{\mathcal{A}}) = \sum_{s \in S} \iota(s) \sum_{q \in Q} \alpha(q)\, \mathbb{E}_{\mathcal{M}[s]}\, L_{\mathcal{A}[q]} \tag{5.1}$$

$$\mathbb{E}_{\mathcal{M}[s]}\, L_{\mathcal{A}[q]} = \sum_{s' \in S} \sum_{q' \in Q} P_{s,s'} M(s)_{q,q'}\, \mathbb{E}_{\mathcal{M}[s']}\, L_{\mathcal{A}[q']} \tag{5.2}$$

Proof. The first equation holds because of the fact that $\mathcal{A}$ is an IBA. Note that for any $q \in Q$, $|\mathit{FinalPaths}_{\mathcal{A}[q]}(w)| \leq N$ for all $w \in S^\omega$. For the second equation we have the following:

$$\begin{aligned}
& \mathbb{E}_{\mathcal{M}[s]}\, L_{\mathcal{A}[q]} \\
= & \int_{w \in Paths(\mathcal{M}[s])} L_{\mathcal{A}[q]}(w) d\, Pr(w) \\
= & \sum_{s' \in S} \int_{w \in Paths(\mathcal{M}[s]) \cap s' S^\omega} L_{\mathcal{A}[q]}(w) d\, Pr(w) \quad \text{since the cylinder sets } sS^\omega \text{ partition } S^\omega \\
= & \sum_{s' \in S} P_{s,s'} \int_{w \in Paths(\mathcal{M}[s'])} L_{\mathcal{A}[q]}(sw) d\, Pr(w) \quad \text{by definition of } Pr(w) \\
= & \sum_{s' \in S} P_{s,s'} \int_{w \in Paths(\mathcal{M}[s'])} \sum_{p \in \mathit{FinalPaths}_{\mathcal{A}[q]}(sw)} \mathit{weight}(p) d\, Pr(w) \quad \text{definition of } L_{\mathcal{A}[q]} \\
= & \sum_{s' \in S} P_{s,s'} \int_{w \in Paths(\mathcal{M}[s'])} \sum_{q' \in Q} M(s)_{q,q'} \sum_{p \in \mathit{FinalPaths}_{\mathcal{A}[q']}(w)} \mathit{weight}(p) d\, Pr(w) \\
& \qquad \text{definition of } \mathit{FinalPaths} \\
= & \sum_{s' \in S} \sum_{q' \in Q} P_{s,s'} M(s)_{q,q'} \int_{w \in Paths(\mathcal{M}[s'])} \sum_{p \in \mathit{FinalPaths}_{\mathcal{A}[q']}(w)} \mathit{weight}(p) d\, Pr(w) \\
= & \sum_{s' \in S} \sum_{q' \in Q} P_{s,s'} M(s)_{q,q'} \int_{w \in Paths(\mathcal{M}[s'])} L_{\mathcal{A}[q']}(w) d\, Pr(w) \quad \text{by definition of } L \\
= & \sum_{s' \in S} \sum_{q' \in Q} P_{s,s'} M(s)_{q,q'}\, \mathbb{E}_{\mathcal{M}[s']}\, L_{\mathcal{A}[q']}
\end{aligned}$$

□

Let $\mathcal{M} = (S, P)$ be a Markov chain with state set S, and let ι be an initial distribution on S. Let $B \in \mathbb{R}^{(Q \times S) \times (Q \times S)}$ be the following matrix:

$$B_{\langle q,s \rangle, \langle q',s' \rangle} = P_{s,s'} M(s)_{q,q'}. \tag{5.3}$$

Define $\vec{z} \in \mathbb{R}^{Q \times S}$ by $\vec{z}_{q,s} = \mathbb{E}_{\mathcal{M}[s]}\, L_{\mathcal{A}[q]}$, i.e., the expected weight of a word for the Markov chain starting in s and the automaton starting in q. Similar to Lemma 3.1 we have the following:

Lemma 5.27. *Let B and $\vec{z}$ as defined above. We have that $\vec{z} = B\vec{z}$.*

Proof. Using Lemma 5.26:

$$\begin{aligned}\vec{z}_{q,s} &= \mathbb{E}_{\mathcal{M}[s]}\, L_{\mathcal{A}[q]} \\ &= \sum_{s' \in S} \sum_{q' \in Q} P_{s,s'} M(s)_{q,q'}\, \mathbb{E}_{\mathcal{M}[s']}\, L_{\mathcal{A}[q']} \\ &= (B\vec{z})_{q,s}\end{aligned}$$

□

W.l.o.g. we can assume that for every $\langle q, s\rangle$ in B we can reach an accepting loop. We need to show that strongly connected components in B have a spectral radius of at most 1. In fact, we can give a probabilistic interpretation to values in $(B_{C,C})^n$, where $C \subseteq Q \times S$ is a strongly connected component in B. This mirrors Lemma 3.5.

Lemma 5.28. *Let $C \subseteq Q \times S$ and $\langle q, s\rangle, \langle r, t\rangle \in C$. Let $n \in \mathbb{N}$. Define $A := B_{C,C}$. Then $(A^n)_{\langle q,s\rangle,\langle r,t\rangle} = \mathbb{E}_{\mathcal{M}[s]}\, weight^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}(w)$, where*

$$weight^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}(w) = \sum_{\langle q_0,s_0\rangle \ldots \langle q_n,s_n\rangle \in Paths_{\langle q,s\rangle,\langle r,t\rangle}(A), w \in s_0 \ldots s_n S^\omega} \prod_{0 \le i < n} M(s_i)_{q_i,q_{i+1}}.$$

(Intuitively, $weight^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}(w)$ is the weight of the prefixes of length n of paths over w in C starting in $\langle q, s\rangle$ that reach $\langle r, t\rangle$ at time n.)

Define

$$E^{C,n}_{\langle q,s\rangle,\langle r,t\rangle} := \{s_0 s_1 \ldots \in sS^\omega \mid \exists q_1 \ldots q_n . \langle q_0, s_0\rangle \ldots \langle q_n, s_n\rangle \in Paths_{\langle q,s\rangle,\langle r,t\rangle}(A)\}.$$

In particular, if C is (a subset of) a strongly connected component, then $(A^n)_{\langle q,s\rangle,\langle r,t\rangle} = Pr_{\mathcal{M}[s]}\left(E^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}\right)$ and hence $\rho(A) \le 1$.

Proof.

$$
\begin{aligned}
& \mathbb{E}_{\mathcal{M}[s]}\, weight^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}(w) \\
= & \int_{w\in Paths(\mathcal{M}[s_0])} weight^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}(w) d\, Pr(w) \\
= & \sum_{s_0\ldots s_n\in S^n} \int_{w\in Paths(\mathcal{M}[s])\cap s_0\ldots s_n S^\omega} weight^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}(w) d\, Pr(w) \\
= & \sum_{s_0\ldots s_n\in S^n} \int_{w\in Paths(\mathcal{M}[s_n])} \prod_{0\le i<n} P_{s_i,s_{i+1}}\, weight^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}(s_0\ldots s_n w) d\, Pr(w) \\
= & \sum_{s_0\ldots s_n\in S^n} \int_{w\in Paths(\mathcal{M}[s_n])} \prod_{0\le i<n} P_{s_i,s_{i+1}} \\
& \qquad \sum_{\langle q_0,s_0\rangle\ldots\langle q_n,s_n\rangle\in Paths_{\langle q,s\rangle,\langle r,t\rangle}(A), w\in s_0\ldots s_n S^\omega} \prod_{0\le i<n} M(s_i)_{q_i,q_{i+1}} d\, Pr(w) \\
= & \sum_{s_0\ldots s_n\in S^n} \int_{w\in Paths(\mathcal{M}[s_n])} \\
& \qquad \sum_{\langle q_0,s_0\rangle\ldots\langle q_n,s_n\rangle\in Paths_{\langle q,s\rangle,\langle r,t\rangle}(A)), w\in s_0\ldots s_n S^\omega} \prod_{0\le i<n} P_{s_i,s_{i+1}} M(s_i)_{q_i,q_{i+1}} d\, Pr(w) \\
= & \sum_{s_0\ldots s_n\in S^n} \sum_{\langle q_0,s_0\rangle\ldots\langle q_n,s_n\rangle\in Paths_{\langle q,s\rangle,\langle r,t\rangle}(A)} \prod_{0\le i<n} P_{s_i,s_{i+1}} M(s_i)_{q_i,q_{i+1}} \\
= & \sum_{s_0\ldots s_n\in S^n} \sum_{\langle q_0,s_0\rangle\ldots\langle q_n,s_n\rangle\in Paths_{\langle q,s\rangle,\langle r,t\rangle}(A)} \prod_{0\le i<n} A_{\langle q_i,s_i\rangle,\langle q_{i+1},s_{i+1}\rangle} \\
= & (A^n)_{\langle q,s\rangle,\langle r,t\rangle}
\end{aligned}
$$

Note that since an accepting loop can be reached from every state in B, no strongly connected component in B can have diamonds: otherwise, one could traverse a loop over that diamond $\log N + 1$ times to get a word that has more than N final paths. Since any path that stays in a strongly connected component has weight 1, and there are no diamonds in strongly connected components, we see that if C is (a subset of) a strongly connected component, then $\mathbb{E}_{\mathcal{M}[s]}\, weight^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}(w) = Pr_{\mathcal{M}[s]}\left(E^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}\right)$ and hence $(A^n)_{\langle q,s\rangle,\langle r,t\rangle} = Pr_{\mathcal{M}[s]}\left(E^{C,n}_{\langle q,s\rangle,\langle r,t\rangle}\right)$. □

The definitions of recurrent strongly connected components and cuts are completely equivalent to those in Chapter 3 and are copied here verbatim: a *recurrent strongly connected component* is an strongly connected component with spectral radius 1. We call a recurrent strongly connected component D *accepting* if for some $\langle q,t\rangle \in D$ we have $q \in F$. Let $D \subseteq Q \times S$ be a strongly connected component of B. A set $\alpha \subseteq D$ is called a *fiber* if it can be written as $\alpha = A \times \{s\}$ for some $A \subseteq Q, s \in S$. Given such a fiber α and $t \in S$, if $M_{s,t} > 0$ we then define a fiber

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

If $M_{s,t} = 0$ then $\alpha \triangleright t$ is left undefined. We extend this definition inductively by writing $\alpha \triangleright \epsilon = \alpha$ and $\alpha \triangleright wt = (\alpha \triangleright w) \triangleright t$ for $t \in S, w \in S^*$. 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*.

Let D be a recurrent strongly connected component. A vector $\vec{\mu} \in [0,1]^D$ is called a *D-normaliser* if $\vec{\mu}^T \vec{z} = 1$.

We will need versions of Lemma 3.8 and Lemma 3.13 as well. Lemma 3.13.1 only relies on the probabilistic interpretation of submatrices in strongly connected components as shown in Lemma 5.28 and the non-negativity of those matrices, and hence we can invoke its proof verbatim:

Lemma 5.29 (Lemma 3.13.1 for IBAs)**.** *Let $D \subseteq Q \times S$ be a strongly connected component. Then D is recurrent if and only if it has a cut.*

Proof. Verbatim from Lemma 3.13. □

For our proof of Lemma 3.8.1 for IBAs we will first show something stronger:

Lemma 5.30. *Let D be a recurrent strongly connected component, and let $C \subseteq Q \times S \setminus D$ be the set of states outside of D reachable from D. Then $\rho(B_{C,C}) < 1$.*

Proof. By Lemma 5.28, we have that $(B^n_{C,C})_{\langle q,s\rangle,\langle q',s'\rangle}$ is the expected weight of paths of length n from $\langle q, s\rangle$ to $\langle q', s'\rangle$. Since the number of such paths is bounded from above by N, and the weight of each individual path is bounded by the product of the weights in the automaton (since all the edges on loops have weight 1), this means that $(B^n_{C,C})_{\langle q,s\rangle,\langle q',s'\rangle}$ is bounded from above and below by a constant, and hence $\rho(B_{C,C}) \leq 1$. Towards a contradiction, assume $\rho(B_{C,C}) = 1$. Then by the boundedness of every (product of) submatrices of $B_{C,C}$ there must exist a recurrent strongly connected component D' such that D' is reachable from D. We will create a word consisting of five parts such that $\mathcal{A}$ has more than N final paths over that word. Since N is the global bound on the number of final paths in $\mathcal{A}$, this leads to a contradiction.

1. Firstly, pick $d = \langle q, s_0\rangle \in D$. Let $s_1 \dots s_i$ be such that $d \triangleright s_1 \dots s_i$ is a cut, such a word exists by Lemma 5.29.

2. Let $s_{i+1} \dots s_j$ be such that there exists a path from d to some $d' \in D'$ over $s_1 \dots s_j$.

3. Let $s_{j+1} \dots s_k$ be such that $d' \triangleright s_{j+1} \dots s_k$ is a cut. Again this exists by Lemma 5.29.

4. Let $s_{k+1} \dots s_l$ be such that $d \in d \triangleright s_1 \dots s_l$. This exists because $d \triangleright s_1 \dots s_i$ is a cut and for any cut c and any $s \in S$ we have that $c \triangleright s$ is also a cut.

Then consider the word $(s_1 \dots s_l)^{|D'|(N+1)}$. Because $d \in d \triangleright s_1 \dots s_l$ and we reach a cut in D' from d over $s_1 \dots s_l$, this means at least $|D'|(N+1)$ cuts in D' are reached after reading $(s_1 \dots s_l)^{|D'|(N+1)}$ from D. By the pigeonhole principle, there exists a $d' \in D'$ such that there are at least $N+1$ paths from d to d' over $(s_1 \dots s_l)^{|D'|(N+1)}$. Hence, if we pick $s_{l+1} \dots$ such that there is a final path from d' over $s_{l+1} \dots$, we see that there are more than N final paths from d over $(s_1 \dots s_l)^{|D'|(N+1)} s_{l+1} \dots$, contradicting the global bound on the number of final paths in $\mathcal{A}$. Thus, $\rho(D') < 1$ and therefore $\rho(B_{C,C}) < 1$. □

Now we can show Lemma 3.8.1 for IBAs:

Lemma 5.31 (Lemma 3.8.1 for IBAs)**.** *Let D be a recurrent strongly connected component, and let $\vec{x}$ be any vector satisfying $\vec{x} = B\vec{x}$. Then $\vec{x}_D = B_{D,D}\vec{x}_D$.*

Proof. Let $\bar{D} = Q \times S \setminus D$ and let $C \subseteq \bar{D}$ be the set of states reachable from D. We have that $\vec{x}_D = (B\vec{x})_D = B_{D,D}\vec{x}_D + B_{D,\bar{D}}\vec{x}_{\bar{D}} = B_{D,D}\vec{x}_D + B_{D,C}\vec{x}_C$. We claim that $\vec{x}_C = \vec{0}$. Indeed,

$$\begin{aligned} \vec{x}_C &= (B\vec{x})_C \\ &= B_{C,C}\vec{x}_C + B_{C,D}\vec{x}_C, \end{aligned}$$

but since C is reachable from D and D is a strongly connected component, $B_{C,D}$ is a zero matrix, and by Lemma 5.30, $\rho(B_{C,C}) < 1$. Hence, $\vec{x}_C = B_{C,C}\vec{x}_C$ iff $\vec{x}_C = \vec{0}$ and therefore $\vec{x}_D = B_{D,D}\vec{x}_D + B_{D,C}\vec{x}_C = B_{D,D}\vec{x}_D$. □

The proof of Lemma 3.8.2 relies on the following lemma:

Lemma 5.32. *Let D be a recurrent strongly connected component in B, and let $\langle q, s\rangle \in D$. Then almost surely all final paths of $\mathcal{A}[q]$ are contained in D, and for any $\langle q', s'\rangle \notin D$ reachable from $\langle q, s\rangle$, $z_{\langle q', s'\rangle} = 0$.*

Proof. Suppose that with positive probability a word starting in s is generated such that $\mathcal{A}[q]$ has a final path outside D. We will construct a word consisting of three repeating parts that admits more than N final paths:

1. Let $s_0 \dots s_l \in S^*$ be such that $s_0 = s$ and $\langle q, s\rangle \triangleright s_1 \dots s_l$ is a cut containing $\langle q, s\rangle$. Such a word exists by Lemma 5.29.

2. The language of words that have a final path outside D is regular. The probability of generating a word with a final path outside D is bounded from above by the sum of probabilities of reaching some $\langle q', s'\rangle$ outside D times the probability of generating a word starting in s' over which there exists a final path from q', and the probability of generating a word with a final path outside D is non-zero. Therefore, there must exist $\langle q', s'\rangle \notin D$ reachable from $\langle q, s\rangle$ such that with non-zero probability, a word starting in s' is generated over which $\mathcal{A}[q']$ has a final path. Let $s_l \dots s_{m'}$ be such that $s_{m'} = s'$ and $M(s_l \dots s_{m'})_{q,q'} > 0$. By Lemma 2.4 there exists a word $s_{m'} \dots s_m$ such that $Pr_{s_m}(\{w \in s_m S^\omega \mid \text{there exists a final path from } q' \text{ over } s_{m'} \dots s_m w\}) = 1$. Hence, $s_l \dots s_m$ is such that $Pr_{s_l}(s_l \dots s_m S^\omega) > 0$ and any word prefixed by $s_l \dots s_m$ almost surely has a final path outside D.

3. Let $\langle q'', s''\rangle \in \langle q, s\rangle \triangleright s_0 \dots s_m$ (which exists since $\langle q, s\rangle \triangleright s_0 \dots s_l$ is a cut and by extension so is $\langle q, s\rangle \triangleright s_0 \dots s_m$). By strongly connectedness of D there exists $s_m \dots s_n \in S^*$ such that $s_m = s''$ and $\langle q, s\rangle \in \langle q'', s''\rangle \triangleright s_{m+1} \dots s_n$.

Now consider the word $(s_0 \dots s_n)^{N+1}$. By construction, $\langle q, s\rangle \in \langle q, s\rangle \triangleright s_0 \dots s_n$ and there exists $\langle q', s'\rangle \notin D$ such that q reaches q' over $s_0 \dots s_{m-1}$ and $Pr_{s_m}(\{w \in s_m S^\omega \mid \text{there exists a final path from } q' \text{ over } s_{m'} \dots s_m w\}) = 1$. Hence, after every iteration of $s_0 \dots s_n$, the automaton can non-deterministically choose to stay in D or move to a set of states out of D after which it will almost surely have a final path. Thus, for almost any word prefixed with $(s_0 \dots s_n)^{N+1}$, there are at least $N+1$ final paths, which contradicts the fact that $\mathcal{A}$ is an IBA. This shows that almost surely any final path from $\langle q, s\rangle \in D$ stays in D, and hence for any $\langle q', s'\rangle \notin D$ reachable from $\langle q, s\rangle$, $z_{\langle q', s'\rangle} = 0$. □

With this we can invoke the proof of Lemma 3.8.2 verbatim:

Lemma 5.33 (Lemma 3.8.2 for IBAs)**.** *Let D be a recurrent strongly connected component. For all $d \in D$, we have $\vec{z}_d > 0$ iff D is accepting.*

Proof. Verbatim from Lemma 3.8.2, relying only on Lemma 5.32 and the probabilistic interpretation in Lemma 5.28. □

The proof of Lemma 3.13.2 only relies on Lemma 5.33 and hence can be invoked verbatim. This shows that cut vectors are D-normalisers.

Lemma 5.34 (Lemma 3.13.2 for IBAs)**.** *Let D be an accepting recurrent strongly connected component and let $\vec{\mu}$ be a cut vector in D. Then $\vec{\mu}^\top \vec{z}_D = 1$.*

Proof. Verbatim. □

Finally we can express $\vec{z}$ as the unique solution of a system of equations, mirroring Lemma 3.9:

Lemma 5.35. *Let $\mathcal{D}_+$ be the set of accepting recurrent strongly connected components, and $\mathcal{D}_0$ the set of non-accepting recurrent strongly connected components. For each $D \in \mathcal{D}_+$, let $\vec{\mu}_D \in [0,1]^D$ be a D-normaliser (which exists by Lemma 5.34). Then $\vec{z}$ is the unique solution of the following linear system:*

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

Proof. This proof is equivalent to the one of Lemma 3.9 and is included for completeness. The vector $\vec{z}$ solves the system of equations by Lemma 5.27, Lemma 5.33, and the definition of a D-normaliser. To show uniqueness, let $\vec{x}$ solve the system of equations. We show that $\vec{x} = \vec{z}$.

- Let $D \in \mathcal{D}_0$. Then $\vec{x}_D = \vec{0} = \vec{z}_D$.
- Let $D \in \mathcal{D}_+$. By Lemma 5.31, we have $\vec{x}_D = B_{D,D}\vec{x}_D$ and $\vec{z}_D = B_{D,D}\vec{z}_D$. By Theorem 2.7.2, the eigenspace of $B_{D,D}$ 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 $\vec{x}_D = \vec{z}_D$.
- Let $D := \bigcup \mathcal{D}_+ \cup \bigcup \mathcal{D}_0$ be the union of all recurrent strongly connected components, and let $\bar{D} := (Q \times S) \setminus D$. We have $\vec{x} - \vec{z} = B(\vec{x} - \vec{z})$. It follows

$$\begin{aligned} \vec{x}_{\bar{D}} - \vec{z}_{\bar{D}} &= B_{\bar{D},\bar{D}}(\vec{x}_{\bar{D}} - \vec{z}_{\bar{D}}) + B_{\bar{D},D}(\vec{x}_D - \vec{z}_D) \\ &= B_{\bar{D},\bar{D}}(\vec{x}_{\bar{D}} - \vec{z}_{\bar{D}}) + B_{\bar{D},D}(\vec{0}) \\ &= B_{\bar{D},\bar{D}}(\vec{x}_{\bar{D}} - \vec{z}_{\bar{D}}) \end{aligned}$$

  by the previous two items. By Lemma 5.30, $\rho(B_{\bar{D},\bar{D}}) < 1$. Thus, $\vec{x}_{\bar{D}} = \vec{z}_{\bar{D}}$.

□

This enables us to prove Theorem 5.25:

Proof of Theorem 5.25. It suffices to calculate D-normalisers and solve the system of equations described in Lemma 5.35. Solving a system of equations can be done in NC, and we can find a D-normaliser using Theorem 3.18 in NC, proving the theorem. □

Corollary 5.36. *Let $\mathcal{M}$ be an MC and let $\mathcal{A}_k$ be a k-ABA with n states, where $k = O(\log n)$. The probability that a random word sampled from $\mathcal{M}$ is in $L_{\mathcal{A}}$ can be computed in POLYLOGSPACE.*

Proof. This combines Theorems 5.24 and 5.25. Since NC is contained in POLYLOGSPACE [53], this proves the corollary. □

PSPACE-hardness of model checking k-ABAs against Markov chains can be shown easily from the *finite intersection problem* introduced by D. Kozen [45]. This problem asks, given a set of deterministic automata, whether the intersection of the languages is empty. By considering the union of the complements of the automata as a k-ABA we see that the probabilistic universality problem for k-ABAs is PSPACE-hard. Membership in PSPACE is shown by translating the k-ABA into an equivalent IBA and model checking this IBA against the Markov chain.

Theorem 5.37. *The model checking problem for k-ABAs is PSPACE-complete.*

Corollary 5.38. *Let $\mathcal{M}$ be an MC and let $\mathcal{A}$ be an NBA with n states. The probability that a word in $L_{\mathcal{A}}$ is accepted by $\mathcal{M}$ can be computed in PSPACE.*

Proof. By Löding et al. [51], $\mathcal{A}$ can be converted in a k-ABA $\mathcal{A}_k$ with at most 3^n states where $k = n$. Hence, using Corollary 5.36, we can calculate $\Pr(L_{\mathcal{A}_k}) = \Pr(L_{\mathcal{A}})$ in $\mathrm{POLYLOG}(2^n) = \mathrm{POLY}(n)$ space. □

5.4 Conclusion

In this chapter we explored a new class of automata called image-binary automata. We showed that these automata exhibit certain desirable properties, in particular efficient closure under Boolean operations. In addition we showed that the model checking procedure for unambiguous Büchi automata, which is a subclass of image-binary automata, still holds for IBAs.

However, many questions are still open about image-binary automata. While for unambiguous and non-deterministic Büchi automata the emptiness question can be solved trivially in polynomial time, this is not the case for image-binary automata. In fact, the complexity of the universality problem for UBAs is an important open

question, and the efficient closure under complement of image-binary automata means that the emptiness problem for image-binary automata is as hard as the universality problem for unambiguous automata (as a subclass of image-binary automata). Beyond that, we did not consider the model checking question of non-deterministic models such as Kripke structures or even Markov decision processes against specifications given by image-binary automata.

One might also consider natural generalisations of image-binary automata. One such generalisation one might consider, is image-finite automata: automata that map words to a finite domain. By labeling some elements of that domain to be accepting, one can again view such image-finite automata as language acceptors. With similar arguments as those in Section 5.2.2, one can show that the languages accepted by such automata are again the regular languages. However, open questions remain as to the succinctness of these image-finite automata. Similarly, one might consider the relation with mod-k-multiplicity automata, for which it is not even clear if the class of languages accepted is the class of regular languages.

6
Branching processes

Contents

6.1 Introduction

This chapter is about qualitative model checking of branching processes against LTL formulas using an automata theoretic approach. Branching processes are a model exhibiting both stochastic and non-deterministic branching. As such, a branching process defines a probability measure over (infinite) trees. Branching processes generalise both Markov chains and Kripke structures, but not Markov decision processes which also have both stochastic and non-deterministic branching. In this

chapter we only consider qualitative model checking, meaning we only consider the question whether a branching process generates a tree all of whose branches are accepted by an LTL formula with probability 0 or 1.

To answer this question we will use an automata theoretic approach. First we will consider branching processes with ϵ-rules allowed, and show that the question whether a branching process almost surely generates a finite tree can be solved in NC. We will use this to show that the qualitative model checking procedure by Chen et al.[17], asking whether the a branching process almost surely generates trees accepted by a deterministic parity automaton, can be run in NC. By using a PSPACE transduction from non-deterministic Büchi automata to deterministic parity automata, we show that this question is in PSPACE for NBAs, and we show that this is tight. We then consider the question whether almost surely a tree is generated where all branches are rejected by NBAs, and we show that this is PSPACE-complete too. However, by making novel use of the spectral properties of unambiguous automata, we show that this question is again in NC when the automaton is a UBA. This finally allows us to solve the qualitative model checking question for branching processes against LTL specifications in PSPACE.

In Section 6.3 we give NC procedures to check whether a branching process almost surely generates a finite tree or a tree accepted by a DPA specification, and a PSPACE procedure for specifications given by an NBA. In Section 6.4 we give a PSPACE procedure to check whether it almost surely generates a tree where all branches are rejected by an NBA specification. This could be done through a transduction into DPA and complementation of the DPA, but the construction in Section 6.4 paves the way for Section 6.5, where we place this problem in NC for specifications given by a UBA. Section 6.6 then uses this procedure, combined with a PSPACE transduction from LTL into UBAs, to show membership in PSPACE of the question whether almost surely all branches are accepted by an LTL specification. In Section 6.7 we complement the results for the question whether the branching process almost surely generates accepted trees with hardness results for the question whether it almost never generates accepted trees. For these questions, membership is easily shown through translation into DPAs and the polynomial time model checking procedure from [17]. Finally we conclude in Section 6.8.

6.2 Problem statement and results

In this chapter we consider the following computational problems:

Problem 6.1. *Let $\mathbb{P}_{\mathcal{B}}(\varphi)$ denote the probability that $\mathcal{B}$ generates a tree where all branches satisfy φ, and similarly let $\mathbb{P}_{\mathcal{B}}(\mathcal{A}$ accepts$)$ (resp. $\mathbb{P}_{\mathcal{B}}(\mathcal{A}$ rejects$)$) denote the probability that $\mathcal{B}$ generates a tree where all branches are accepted (resp. rejected) by an automaton $\mathcal{A}$. The qualitative model checking problems for branching processes are as follows:*

1. *The problem $\mathbb{P}($finite$) = 1$ asks, given a BP $\mathcal{B}$ with ε-rules allowed, whether the probability that a $\mathcal{B}$-tree is finite is 1.*
2. *The problem $\mathbb{P}(\mathrm{LTL}) = 1$ asks, given a BP $\mathcal{B}$ and an LTL formula φ, whether $\mathbb{P}_{\mathcal{B}}(\varphi) = 1$.*
3. *The problems $\mathbb{P}(\mathrm{DPA}) = 1$ (resp., $\mathbb{P}(\mathrm{NBA}) = 1$) ask, given a BP $\mathcal{B}$ and a DPA (resp., NBA) $\mathcal{A}$, whether $\mathbb{P}_{\mathcal{B}}(\mathcal{A}$ accepts$) = 1$.*
4. *The problems $\mathbb{P}(\mathrm{coNBA}) = 1$ (resp., $\mathbb{P}(\mathrm{coUBA}) = 1$)[1] ask, given a BP $\mathcal{B}$ and an NBA (resp., UBA) $\mathcal{A}$, whether $\mathbb{P}_{\mathcal{B}}(\mathcal{A}$ rejects$) = 1$.*
5. *The problems $\mathbb{P}(\mathrm{LTL}) = 0, \mathbb{P}(\mathrm{DPA}) = 0, \ldots$ are defined similarly, where "$= 1$" is replaced with "$= 0$".*

We may write $\mathbb{P}(\_\_) = 1$ to collectively refer to the problems $\mathbb{P}(\mathrm{LTL}) = 1, \mathbb{P}(\mathrm{DPA}) = 1, \ldots$, and similarly we may write $\mathbb{P}(\_\_) = 0$ to refer to the problems $\mathbb{P}(\mathrm{LTL}) = 0, \mathbb{P}(\mathrm{DPA}) = 0, \ldots$.

See Table 6.1 for a map of our results in those terms, as well as for an overview of the rest of the chapter. As explained in the introduction, the problem $\mathbb{P}(\mathrm{LTL}) = 1$ is of particular interest from a model-checking point of view, and the technically most challenging one.

6.3 Basic Results

In this section we develop the more basic results indicated in Table 6.1, on finiteness (Section 6.3.1), deterministic parity automata (Section 6.3.2), and Büchi automata (Section 6.3.3), on the one hand rounding off the complexity map in Table 6.1, and on the other hand building the foundation for more challenging results in the following sections. In particular, Proposition 6.4 is indirectly used throughout the paper.

[1] We do not explicitly define or use a notion of "co-Büchi automata" to avoid possible confusion about accepting/rejecting. If one were to do so, one would define a "co-NBA" $\mathcal{A}$ like an NBA $\mathcal{A}$, but the "co-NBA" $\mathcal{A}$ would accept a word $w \in \Gamma^\omega$ if and only if $\mathcal{A}$ viewed as an NBA rejects w. Similarly for "co-UBAs".

	$= 1$	$= 0$
$\mathbb{P}$(finite) Section 6.3.1	in NC Proposition 6.4	-
$\mathbb{P}$(DPA) Section 6.3.2	in NC Theorem 6.6	P Theorem 6.7
$\mathbb{P}$(NBA) Section 6.3.3	PSPACE Theorem 6.8	EXPTIME Theorem 6.9
$\mathbb{P}$(coNBA) Section 6.4	PSPACE Theorem 6.12	EXPTIME Theorem 6.13
$\mathbb{P}$(coUBA) Section 6.5	in NC Theorem 6.16	-
$\mathbb{P}$(LTL) Section 6.6	PSPACE Theorem 6.18	2EXPTIME Theorem 6.19

Table 6.1: Results and organization of the paper. The complexity classes indicate completeness results, except "in NC", which only means membership in NC. The problems $\mathbb{P}(\text{finite}) = 0$ and $\mathbb{P}(\text{coUBA}) = 0$ were not considered in this work.

6.3.1 Finiteness

In this section we consider BPs with ε-rules allowed, i.e., rules of the form $X \hookrightarrow \varepsilon$. Such BPs may generate finite trees. We are interested in the *almost-sure finiteness* problem, also denoted as $\mathbb{P}(\text{finite}) = 1$, i.e., the problem whether the probability that a given BP with ε-rules allowed generates a finite tree is equal to 1. In Proposition 6.4 below we show that this problem is in NC. All upper bounds on the complexity of $\mathbb{P}(\cdot) = 1$ problems in this paper build directly or indirectly on this result.

While the almost-sure finiteness (or "extinction") problem has often been studied and is known to be in (strongly) polynomial time [24; 26], its membership in NC is, to the best of the authors' knowledge, new. For instance, since linear programming is P-complete, one cannot use linear rogramming (as in [26]) to show membership in NC. Nor can one directly use the strongly polynomial-time algorithm of [24], as it computes, in a sub-procedure, the set of types X for which there *exists* a finite X-tree. But the latter problem is P-complete.

For the rest of the section, fix a BP $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ with ε-rules allowed. Define a directed graph $G = (\Gamma, E)$ (i.e., the types of $\mathcal{B}$ are the vertices of G) with an edge $(X, Y) \in E$ if and only if Y is a successor of X (i.e., there is a rule $X \hookrightarrow uYv$ for some $u, v \in \Gamma^*$). Given a strongly connected component $S \subseteq \Gamma$ of G and $X \in S$, define a BP $\mathcal{B}[S, X] = (S, \hookrightarrow_S, \mathit{Prob}_S, X)$ obtained from $\mathcal{B}$ by

restricting the types to S and deleting on all right-hand sides of the rules those types not in S. The following lemma is straightforward:

Lemma 6.2. *A $\mathcal{B}$-tree is infinite with positive probability if and only if there exist a strongly connected component $S \subseteq \Gamma$ of G and $X \in S$ such that X is reachable from X_0 in G and a $\mathcal{B}[S, X]$-tree is infinite with positive probability.*

Let $M \in \mathbb{Q}^{\Gamma\times\Gamma}$ be the non-negative $\Gamma \times \Gamma$-matrix with $M_{X,Y} = \sum_{X \overset{p}{\hookrightarrow} w} p|w|_Y$, where $|w|_Y \in \mathbb{N}_0$ is the number of occurrences of Y in w. That is, $M_{X,Y}$ is the expected number of direct Y-successors of the root of a $\mathcal{B}[X]$-tree. By induction, M^i, the ith power of M, is such that $(M^i)_{X,Y}$ is the expected number of Y-nodes that are exactly i levels under the root of a $\mathcal{B}[X]$-tree. The graph of M is exactly the previously defined graph G.

Let $S \subseteq \Gamma$ be a strongly connected component of G. Denote by $M_S \in \mathbb{Q}^{S\times S}$ the (square) principal submatrix obtained from M by restricting it to the rows and columns indexed by elements of S. Let ρ_S denote the spectral radius of M_S. Call S *supercritical* if $\rho_S > 1$. Call S *linear* if for all rules $X \hookrightarrow w$ with $X \in S$ there is exactly one occurrence in w of a type in S. Observe that if S is linear then M_S is *stochastic*, i.e., $M_S\vec{1} = \vec{1}$ where $\vec{1}$ is the all-1 vector, i.e., the element of $\{1\}^S$. In that case, by Theorem 2.7.5, we have $\rho_S = 1$ and, thus, S is not supercritical.

The following characterization can be proved using [24, Section 3] (which builds on [26, Section 8.1]):

Lemma 6.3. *A $\mathcal{B}$-tree is infinite with positive probability if and only if there exist a strongly connected component $S \subseteq \Gamma$ of G and $X \in S$ such that X is reachable from X_0 in G and S is supercritical or linear.*

Proof. By Lemma 6.2 it suffices to show that if G is strongly connected then a $\mathcal{B}$-tree is infinite with positive probability if and only if Γ is supercritical or linear. So, let G be strongly connected.

Call a type $X \in \Gamma$ *immortal* if there is no finite X-tree. If a type X is immortal, then the probability that a $\mathcal{B}[X]$-tree is finite is 0, and for all $Y \in \Gamma$ the probability that a $\mathcal{B}[Y]$-tree is finite is less than 1 (as G is strongly connected). If Γ is linear then all types are immortal. So we can assume in the following that Γ is not linear.

Since Γ is not linear and G is strongly connected, for all $X \in \Gamma$ there is a finite prefix of an X-tree that has at least two leaves of type X; and an X-tree has, with positive probability, that finite prefix. Suppose some type, say X, is immortal. Then every X-tree that has a finite prefix with k ($k \in \mathbb{N}$) leaves of type X has at least k (infinite) branches that go through those k nodes of type X.

By strong connectedness, it follows that an X-tree that has a finite prefix with k leaves of type X has, with probability 1 (i.e., $\mathbb{P}_X$-almost surely), a finite prefix with $k+1$ leaves of type X. Hence, with probability 1 we have $\lim_{i\to\infty} Z_i = \infty$, where Z_i is the number of nodes that are exactly i levels under the root of an X-tree. Denoting by Ex the expectation with respect to $\mathbb{P}_X$, it follows with Fatou's lemma that $\lim_{i\to\infty} \mathrm{Ex} Z_i = \infty$. We can write $\mathrm{Ex} Z_i = (M^i \vec{1})_X$, where $\vec{1}$ denotes the vector in $\{1\}^\Gamma$. Then

$$\lim_{i\to\infty} \left\| M^i \vec{1} \right\| \;=\; \infty\,,$$

where $\|\cdot\|$ denotes the 1-norm. Since G is strongly connected, by Theorem 2.7.1, M has an eigenvector $v \in \mathbb{R}^\Gamma$ with $v_Y > 0$ for all $Y \in \Gamma$ and $Mv = \rho v$, where ρ denotes the spectral radius of M. Since $M^i v = \rho^i v$, it follows that

$$\lim_{i\to\infty} \rho^i \, \|v\| \;=\; \lim_{i\to\infty} \left\| M^i v \right\| \;=\; \infty\,.$$

Hence $\rho > 1$. We conclude that if some type is immortal, then we have $\rho > 1$ and the probability that a $\mathcal{B}$-tree is finite is less than 1.

On the other hand, if no type is immortal, then it follows from [24, Section 3] (building on [26, Section 8.1]) that a $\mathcal{B}$-tree is infinite with positive probability if and only if $\rho > 1$. We conclude that, regardless of whether there exists an immortal type, a $\mathcal{B}$-tree is infinite with positive probability if and only if Γ is supercritical. □

It follows:

Proposition 6.4. *The problem $\mathbb{P}(\textit{finite}) = 1$ is in NC.*

Proof. We use the characterization from Lemma 6.3. One can compute in NL, and hence in NC, the directed acyclic graph of strongly connected components of G and the set of types that are reachable from X_0. Therefore, we assume without loss of generality that G is strongly connected. By Lemma 2.8 one can check in NC whether Γ is supercritical. Whether Γ is linear can be checked in logarithmic space, hence in NC. □

6.3.2 Deterministic Parity Automata

In this section we consider deterministic parity automata (DPAs) on words. In [17, Section 3] it was shown that the problem $\mathbb{P}(\mathrm{DPA}) = 1$ can be decided in polynomial time. We improve this to membership in NC.

By the following lemma we can check in NC whether a $\mathcal{B}$-tree almost surely has a finite prefix all whose leaves have types in a given set T. The proof is by reduction to almost-sure finiteness.

Lemma 6.5. *Given a BP $\mathcal{B} = (\Gamma, \hookrightarrow, Prob, X_0)$ and a set of types $T \subseteq \Gamma$, the problem whether $\mathbb{P}_{X_0}(\Diamond T) = 1$ is in NC.*

Proof. The problem can be rephrased as almost-sure finiteness of BPs with ε-rules allowed. Indeed, given $\mathcal{B}$ and T, one can eliminate all occurrences of types in T from all right-hand sides; then, $\Diamond T$ in the original BP corresponds to finiteness in the new BP. Hence, the result follows from Proposition 6.4. □

By combining Lemma 6.5 with results from [17] we obtain:

Theorem 6.6. *The problem $\mathbb{P}(\mathrm{DPA}) = 1$ is in NC.*

Proof. In [17, Section 3] it was shown that the problem $\mathbb{P}(\mathrm{DPA}) = 1$ can be decided in polynomial time. We show how to implement this approach in NC. In [17, Section 3] a product of the BP and the DPA is computed and analyzed; the product is a BP whose types are coloured with *priorities* (natural numbers). The product, call it $\mathcal{B} = (\Gamma, \hookrightarrow, Prob, X_0)$, can be computed in logarithmic space. The question is then whether the probability is 1 that all branches of a $\mathcal{B}$-tree are such that the highest priority that appears infinitely often is even. It is shown in [17, Section 3.1] that this is the case if and only if all types X that are reachable from X_0 and are associated with an odd priority satisfy $\mathbb{P}_X(\Diamond N_X) = 1$, where $N_X \subseteq \Gamma$ is a certain set of types that can be computed in NL by a simple reachability analysis in $\mathcal{B}$. By Lemma 6.5 one can determine in NC whether $\mathbb{P}_X(\Diamond N_X) = 1$. The theorem follows. □

The hardness result in the following theorem highlights the different complexities of $\mathbb{P}(\cdot) = 0$ and $\mathbb{P}(\cdot) = 1$ problems in this chapter.

Theorem 6.7. *The problem $\mathbb{P}(\mathrm{DPA}) = 0$ is P-complete. It is P-hard even for deterministic Büchi automata with two states, the accepting state being a sink.*

The proof can be found in Section 6.7.

6.3.3 Büchi Automata

Theorem 6.8. *The problem $\mathbb{P}(\mathrm{NBA}) = 1$ is PSPACE-complete.*

Proof. PSPACE-hardness is immediate in two different ways. It follows from the PSPACE-hardness of model checking Markov chains against NBAs [69]. It also follows from the PSPACE-hardness of model checking transition systems against NBAs. (The latter follows easily from the PSPACE-hardness of NBA universality [64].) Both model-checking problems are special cases of $\mathbb{P}(\mathrm{NBA}) = 1$.

Towards membership in PSPACE, we use a translation from NBA to DPA [56]. This translation causes an exponential blow-up, but an inspection of the construction [56, Section 3.2] reveals that it can be computed by a PSPACE transducer. By Theorem 6.6 the problem $\mathbb{P}(\text{DPA}) = 1$ is in NC. By Lemma 2.5 it follows that $\mathbb{P}(\text{NBA}) = 1$ is in PSPACE. □

Theorem 6.9. *The problem* $\mathbb{P}(\text{NBA}) = 0$ *is EXPTIME-complete. It is EXPTIME-hard even for NBAs whose only accepting state is a sink.*

The proof can be found in Section 6.7.

6.4 Co-Büchi Automata

In this section we consider the problem $\mathbb{P}(\text{coNBA}) = 1$, which asks, given a BP $\mathcal{B}$ and a Büchi automaton $\mathcal{A}$, whether $\mathcal{B}$ almost surely generates a tree whose branches are all rejected by $\mathcal{A}$; i.e., whether $\mathbb{P}_{\mathcal{B}}(\mathcal{A} \text{ rejects}) = 1$. Dually, one might ask whether the probability is positive that a $\mathcal{B}$-tree has a branch accepted by $\mathcal{A}$. Intuitively, we view the Büchi automaton $\mathcal{A}$ as specifying "bad" branches, and we would like the tree almost surely not to have any bad branches.

This problem is in PSPACE, which can be shown via a translation to DPAs, as in Theorem 6.8. However, with a view on the following sections, in particular on LTL specifications, we pursue a different approach to the problem $\mathbb{P}(\text{coNBA}) = 1$. In this section we lay the groundwork for arbitrary Büchi automata $\mathcal{A}$. By building on these results, we will show in the next section that if $\mathcal{A}$ is *unambiguous* then the problem is in NC, which will allow us to derive our headline result, namely that $\mathbb{P}(\text{LTL}) = 1$ is in PSPACE.

Let $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ be a BP and $\mathcal{A} = (Q, \Gamma, \delta, Q_0, F)$ a (not necessarily unambiguous) Büchi automaton.

Define a Büchi automaton, $\mathcal{A} \times \mathcal{B}$, by $\mathcal{A} \times \mathcal{B} := (Q \times \Gamma, \Gamma, \delta_{\mathcal{A}\times\mathcal{B}}, Q_0 \times \{X_0\}, F \times \Gamma)$, where

$$\delta_{\mathcal{A}\times\mathcal{B}}((q_1, X_1), X_2) = \begin{cases} \delta(q_1, X_1) \times \{X_2\} & \text{if } X_2 \text{ is a successor of } X_1 \\ \emptyset & \text{otherwise.} \end{cases}$$

The remainder of the section is organised as follows. In Section 6.4.1 we show that the problem $\mathbb{P}(\text{coNBA}) = 1$ reduces to the analysis of certain strongly connected components within $\mathcal{A} \times \mathcal{B}$. In Section 6.4.2 we introduce a key lemma, Lemma 6.11, which allows us to "forget" about the distinction between accepting and non-accepting states: the lemma reduces $\mathbb{P}(\text{coNBA}) = 1$ to a pure reachability problem

in an exponential-sized BP, $\mathcal{B}_{det}$. This leads us to prove PSPACE-completeness of $\mathbb{P}(\text{coNBA}) = 1$, but more importantly, Lemma 6.11 plays a key role in the rest of the chapter. We prove it in Section 6.4.3.

6.4.1 The Automaton $\mathcal{A}[f, X_f]$

For any $(f, X_f) \in F \times \Gamma$ on a cycle of the transition graph of $\mathcal{A} \times \mathcal{B}$, define the Büchi automaton

$$\mathcal{A}[f, X_f] \;:=\; (\{\bar{q}_0\} \cup Q[f, X_f], \Gamma, \delta[f, X_f], \{\bar{q}_0\}, \{(f, X_f)\})$$

as the Büchi automaton obtained from $\mathcal{A} \times \mathcal{B}$ by

1. making (f, X_f) the only accepting state,
2. restricting the set of states, $Q[f, X_f] \subseteq Q \times \Gamma$, to those (q, X) that, in the transition graph of $\mathcal{A} \times \mathcal{B}$, are reachable from (f, X_f) and can reach (f, X_f), i.e., those (q, X) in the strongly connected component containing (f, X_f),
3. restricting the transition function $\delta[f, X_f]$ accordingly, i.e.,
$$\delta[f, X_f]((q, X), Y) \;:=\; \delta_{\mathcal{A}\times\mathcal{B}}((q, X), Y) \cap Q[f, X_f]\,,$$
4. making $\bar{q}_0$ the only initial state, and
5. setting $\delta[f, X_f](\bar{q}_0, X_f) := \{(f, X_f)\}$ and $\delta[f, X_f](\bar{q}_0, X) := \emptyset$ for all $X \in \Gamma \setminus \{X_f\}$.

The following lemma follows from the pigeonhole principle and basic probability arguments:

Lemma 6.10. *The probability that some branch of a $\mathcal{B}$-tree is accepted by $\mathcal{A}$ is positive if and only if there are $q_0 \in Q_0$ and $f \in F$ and $X_f \in \Gamma$ such that (f, X_f) is reachable from (q_0, X_0) in the transition graph of $\mathcal{A} \times \mathcal{B}$ and the probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive.*

Proof. Consider a branch $X_0X_1X_2\cdots$ of a $\mathcal{B}$-tree accepted by $\mathcal{A}$. Then $\mathcal{A}$ has an accepting run

$$q_0 \xrightarrow{X_0} q_1 \xrightarrow{X_1} q_2 \xrightarrow{X_2} \cdots$$

with $q_0 \in Q_0$. By the pigeonhole principle, there are $f \in F$ and $X_f \in \Gamma$ such that this run contains the segment $f \xrightarrow{X_f}$ infinitely often. By its construction, the Büchi automaton $\mathcal{A} \times \mathcal{B}$ has the accepting run

$$(q_0, X_0) \xrightarrow{X_1} (q_1, X_1) \xrightarrow{X_2} \cdots\,,$$

which contains the state (f, X_f) infinitely often. Let $k \geq 1$ be such that $(q_k, X_k) = (f, X_f)$. Then $\mathcal{A}[f, X_f]$ accepts $X_k X_{k+1} \cdots$ via the run

$$\bar{q}_0 \xrightarrow{X_k} (q_k, X_k) \xrightarrow{X_{k+1}} (q_{k+1}, X_{k+1}) \cdots .$$

Therefore, denoting by $E(f, X_f, n)$ for $n \in \mathbb{N}$ the event that there exists a branch $X_k X_{k+1} \cdots$ emanating from the nth (in a breadth-first order) node in the tree (necessarily a node of type $X_k = X_f$) such that $\mathcal{A}[f, X_f]$ has an accepting run

$$\bar{q}_0 \xrightarrow{X_k} (q_k, X_k) \xrightarrow{X_{k+1}} (q_{k+1}, X_{k+1}) \cdots ,$$

we have

$$\mathbb{P}_{X_0}(\mathcal{A} \text{ accepts some branch}) \leq \sum_{f \in F} \sum_{X_f \in \Gamma} \sum_{n \in \mathbb{N}} \mathbb{P}_{X_0}(E(f, X_f, n)) .$$

Further,

$$\begin{aligned} & \mathbb{P}_{X_0}(E(f, X_f, n)) \\ = \; & \mathbb{P}_{X_0}(\text{the } n\text{th node has type } X_f) \cdot \mathbb{P}_{X_f}(\mathcal{A}[f, X_f] \text{ accepts some branch}) \\ \leq \; & \mathbb{P}_{X_f}(\mathcal{A}[f, X_f] \text{ accepts some branch}) . \end{aligned}$$

The "only if" direction follows.

Towards the "if" direction, suppose that $\mathcal{A} \times \mathcal{B}$ has a path $(q_0, X_0) \xrightarrow{w}{}^* (f, X_f)$ with $q_0 \in Q_0$ and $f \in F$ such that the probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive. Then there is a successor, say X'_f, of X_f such that the probability that some branch of a $\mathcal{B}[X'_f]$-tree is accepted by $\mathcal{A} \times \mathcal{B}$ when started in (f, X_f) is positive. Thus, the probability that some branch of a $\mathcal{B}$-tree (starts with $X_0 w X'_f$ and) is accepted by $\mathcal{A}$ is positive. □

For the rest of the section let $(f, X_f) \in F \times \Gamma$ be on a cycle of the transition graph of $\mathcal{A} \times \mathcal{B}$.

6.4.2 The Determinization $\mathcal{A}_{det}$ and the BP $\mathcal{B}_{det}$

Let

$$\mathcal{A}_{det} := (2^{\{\bar{q}_0\} \cup Q[f, X_f]}, \Gamma, \delta_{det}, \{\bar{q}_0\}, 2^{\{\bar{q}_0\} \cup Q[f, X_f]} \setminus \{\emptyset\})$$

be the determinization of $\mathcal{A}[f, X_f]$ obtained by the standard subset construction. Which states are accepting will not actually be relevant. Note that every state reachable via a non-empty path from $\{\bar{q}_0\}$ is of the form $P \times \{X\}$ with $P \subseteq Q$ and $X \in \Gamma$.

Define a BP $\mathcal{B}_{det}$ based on $\mathcal{A}_{det}$ as

$$\mathcal{B}_{det} := (\Gamma', \hookrightarrow', Prob', \{(f, X_f)\}),$$

where the set of types $\Gamma' \subseteq 2^{Q[f,X_f]}$ is the set of those states in $\mathcal{A}_{det}$ that are reachable (in $\mathcal{A}_{det}$) from $\{\bar{q}_0\}$ via a non-empty path (recall that they are of the form $P \times \{X\}$ with $P \subseteq Q$ and $X \in \Gamma$), and

$$X' \overset{p}{\hookrightarrow}' \delta_{det}(X', X_1) \cdots \delta_{det}(X', X_k)$$

for all $X' = P \times \{X\} \in \Gamma'$ with $P \neq \emptyset$ and all $X \overset{p}{\hookrightarrow} X_1 \cdots X_k$, and $\emptyset \overset{1}{\hookrightarrow}' \emptyset$. Here is the key lemma of this section:

Lemma 6.11. *The following statements are equivalent:*

(i) The probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive.

(ii) The probability that some branch of a $\mathcal{B}_{det}$-tree does not have any nodes of type $\emptyset$ is positive.

We prove Lemma 6.11 in Section 6.4.3. It will be used in the proof of Theorem 6.12 below; but more importantly, Lemma 6.11 is the foundation of Section 6.5.

Given that Lemma 6.11 reflects the key insight of this section, let us comment further. Considering that condition (ii) does not mention a notion of acceptance, one might have two concerns at this point:

(a) Condition (ii) does not obviously imply that with positive probability there is even a branch with infinitely many nodes of types containing (f, X_f).

(b) Even if with positive probability there is such a branch, it is not obvious that such branches would necessarily correspond to branches of $\mathcal{B}[X_f]$ that are accepted by $\mathcal{A}[f, X_f]$.

Even for the special case of Markov chains (i.e., every tree has only a single branch), Lemma 6.11 is not at all obvious, and both concerns (a) and (b) apply. Indeed, for Markov chains, Courcoubetis and Yannakakis prove a statement related to Lemma 6.11, namely [20, Proposition 4.1.4], with a proof related to ours and dealing explicitly with concern (b) above. For the special case of transition systems (i.e., the BP generates exactly one tree), Lemma 6.11 is simple though: consider the branch that follows a cycle around (f, X_f). For the general case, we need a result on BPs from [17], dealing with concern (a) above. The high-level principle

behind the proof of Lemma 6.11 is often used in the analysis of Markov chains: if it is possible, infinitely often, to reach a state with a probability bounded away from 0, then this state is almost surely reached infinitely often. See Section 6.4.3 for a full proof of Lemma 6.11.

We can now derive a PSPACE procedure for the problem $\mathbb{P}(\text{coNBA}) = 1$ without resorting to DPAs:

Theorem 6.12. *The problem* $\mathbb{P}(\text{coNBA}) = 1$ *is PSPACE-complete.*

Proof. Towards membership in PSPACE, fix a BP $\mathcal{B}$ and an NBA $\mathcal{A}$. Since PSPACE is closed under complement, we can focus on the problem whether the probability is positive that a $\mathcal{B}$-tree has a branch accepted by $\mathcal{A}$. We use Lemma 6.10. Since reachability in a graph is in NL and, hence, in PSPACE, it suffices to decide in PSPACE whether the probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive. In order to check that, by Lemma 6.11 it suffices to construct the BP $\mathcal{B}_{det}$ and then invoke the NC procedure of Lemma 6.5 to check if $\mathbb{P}_{\mathcal{B}_{det}}(\Diamond\{\emptyset\}) < 1$. The BP $\mathcal{B}_{det}$ has exponential size but can be computed with a PSPACE transducer. (In particular, whether a state in $\mathcal{A}_{det}$ is reachable from $\{\bar{q}\}$ via a non-empty path can be determined in NPSPACE = PSPACE.) By Lemma 2.5 it follows that $\mathbb{P}(\text{coNBA}) = 1$ is in PSPACE.

PSPACE-hardness follows from the PSPACE-hardness [69] of the *probabilistic emptiness problem*, which, given a Markov chain and an NBA, asks if the probability is 0 that the Markov chain generates a word accepted by the NBA. (We remark that, in contrast, the problem whether a given transition system has a run accepted by a given NBA is in NL: search the product for an accepting cycle). □

Theorem 6.9 (for NBAs) has a coNBA-analogue:

Theorem 6.13. *The problem* $\mathbb{P}(\text{coNBA}) = 0$ *is EXPTIME-complete. It is EXPTIME-hard even for NBAs all whose states are accepting.*

The proof can be found in Section 6.7.

6.4.3 Proof of Lemma 6.11

In this subsection we prove Lemma 6.11, which is instrumental for the main results of the chapter.

Lemma 6.11. *The following statements are equivalent:*

(i) The probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive.

(ii) The probability that some branch of a $\mathcal{B}_{det}$-tree does not have any nodes of type $\emptyset$ is positive.

Fix a word $w \in \Gamma^+$ such that $\delta[f, X_f]((f, X_f), w)$ is maximal, i.e., there is no $w' \in \Gamma^+$ such that $\delta[f, X_f]((f, X_f), w') \supsetneq \delta[f, X_f]((f, X_f), w)$.

Lemma 6.14. *Let $v \in \Gamma^*$ be a word such that $\delta[f, X_f]((f, X_f), v) \ni (f, X_f)$. Then, $\delta[f, X_f]((f, X_f), vw) = \delta[f, X_f]((f, X_f), w)$; i.e., for any path $(f, X_f) \xrightarrow{vw}{}^* (q, X)$ there is a path $(f, X_f) \xrightarrow{v}{}^* (f, X_f) \xrightarrow{w}{}^* (q, X)$.*

Proof. Since $\delta[f, X_f]((f, X_f), v) \ni (f, X_f)$, we have

$$\delta[f, X_f]((f, X_f), vw) \;\supseteq\; \delta[f, X_f]((f, X_f), w)\,.$$

But w is maximal. □

We enrich $\mathcal{A}_{det}$ to obtain a DBA, $\mathcal{A}'_{det}$, whose states have an additional component keeping track of whether the word $w \in \Gamma^+$ from above is being seen. The accepting runs of $\mathcal{A}'_{det}$ contain infinitely many w-labelled segments that, loosely speaking, "start from (f, X_f)". Formally, let

$$W = \{0\} \cup \{v \in \Gamma^* \mid v \text{ is a suffix of } w\}\,,$$

including w and the empty word ε. We assume $0 \notin \Gamma$. Define

$$\mathcal{A}'_{det} \;:=\; (2^{\{\bar{q}_0\} \cup Q[f,X_f]} \times W, \delta'_{det}, (\{\bar{q}_0\}, 0), (2^{Q[f,X_f]} \setminus \{\emptyset\}) \times \{\varepsilon\})\,,$$

where

$$\begin{aligned}
\delta'_{det}((U, Xv), X) &= (\delta_{det}(U, X), v) \\
\delta'_{det}((U, Xv), Y) &= (\delta_{det}(U, X), 0) && \text{for } X \neq Y \\
\delta'_{det}((U, v), X) &= (\delta_{det}(U, X), w) && \text{for } v \in \{0, \varepsilon\},\ (f, X_f) \in \delta_{det}(U, X) \\
\delta'_{det}((U, v), X) &= (\delta_{det}(U, X), 0) && \text{for } v \in \{0, \varepsilon\},\ (f, X_f) \notin \delta_{det}(U, X)\,.
\end{aligned}$$

It follows from Lemma 6.14 and the construction of $\mathcal{A}'_{det}$ that $\mathcal{A}'_{det}$ has a single accepting state reachable from $(\{\bar{q}_0\}, 0)$, namely $(\delta_{det}(\{(f, X_f)\}, w), \varepsilon)$.

Lemma 6.15. *The following statements are equivalent:*

(i) The probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive.

(ii) The probability that some branch of a $\mathcal{B}[X_f]$-tree has a run (accepting or not) in $\mathcal{A}[f, X_f]$ is positive.

(iii) The probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}'_{det}$ is positive.

Lemma 6.15 implies Lemma 6.11, as it follows from the definition of $\mathcal{B}_{det}$ that the probability that some branch of a $\mathcal{B}[X_f]$-tree has a run in $\mathcal{A}[f, X_f]$ (cf. condition (ii) in Lemma 6.15) equals the probability that some branch of a $\mathcal{B}_{det}$-tree does not have any nodes of type $\emptyset$ (cf. condition (ii) of Lemma 6.11). So it remains to prove Lemma 6.15.

Proof of Lemma 6.15. (i) $\Longrightarrow$ (ii). Trivial.

(iii) $\Longrightarrow$ (i). Let $X_f X_1 X_2 \cdots$ be accepted by $\mathcal{A}'_{det}$. Then $X_1 X_2 \cdots$ can be decomposed in $v_1 w v_2 w \cdots$ with $v_1, v_2, \ldots \in \Gamma^*$ such that $\mathcal{A}[f, X_f]$ has paths

$$(f, X_f) \xrightarrow{v_1 w v_2 w \cdots v_i}{}^* (f, X_f)$$

for all $i \geq 1$. Let $i \geq 2$. There is (q, X) with

$$(f, X_f) \xrightarrow{v_1 w v_2 w \cdots v_{i-1} w}{}^* (q, X) \xrightarrow{v_i}{}^* (f, X_f)\,.$$

By Lemma 6.14 there is a path

$$(f, X_f) \xrightarrow{v_1 w v_2 w \cdots v_{i-1}}{}^* (f, X_f) \xrightarrow{w}{}^* (q, X)\,.$$

Thus also $(f, X_f) \xrightarrow{w v_i}{}^* (f, X_f)$. Since $i \geq 2$ was arbitrary, it follows that $\mathcal{A}[f, X_f]$ has an accepting run

$$\bar{q}_0 \xrightarrow{X_f} (f, X_f) \xrightarrow{v_1}{}^* (f, X_f) \xrightarrow{w v_2}{}^* (f, X_f) \xrightarrow{w v_3}{}^* \cdots\,.$$

(ii) $\Longrightarrow$ (iii). For this part we use results from [17], which considers the problem of model checking BPs against DPAs. We need only a special case of such automata: (a) the same (word) automaton is run on every branch of the tree, and (b) our automaton $\mathcal{A}'_{det}$ is a DBA, which can be viewed as a DPA whose states are labelled with only 2 priorities: priority 0 for non-accepting states and priority 1 for accepting states. The paper [17] constructs a product BP from the BP and the automaton, and subsequently considers BPs whose types are coloured with a priority. In this way the model-checking problem reduces to computing the probability that there is a branch on which the highest priority that occurs infinitely often is odd. Since our

automaton $\mathcal{A}'_{det}$ already embeds a BP, instead of taking another product, we define a BP, $\mathcal{B}'_{det}$, more directly based on $\mathcal{A}'_{det}$, in the same way that $\mathcal{B}_{det}$ was defined based on $\mathcal{A}_{det}$ in Section 6.4.2. More explicitly,

$$\mathcal{B}'_{det} \;:=\; (\Gamma', \hookrightarrow', \mathit{Prob}', (\{(f, X_f)\}, w))\,,$$

where the set of types $\Gamma' \subseteq 2^{Q[f,X_f]} \times W$ is the set of those states in $\mathcal{A}'_{det}$ that are reachable (in $\mathcal{A}'_{det}$) from $(\{\bar{q}\}, 0)$ via a non-empty path (recall that they are of the form $(P \times \{X\}, v)$ with $P \subseteq Q$ and $X \in \Gamma$ and $v \in W$), and

$$X' \overset{p}{\hookrightarrow}' \delta'_{det}(X', X_1) \cdots \delta'_{det}(X', X_k)$$

for all $X' = (P \times \{X\}, v) \in \Gamma'$ with $P \neq \emptyset$ and all $X \overset{p}{\hookrightarrow} X_1 \cdots X_k$, and

$$(\emptyset, v) \overset{1}{\hookrightarrow}' (\emptyset, v)$$

for all $v \in W$. Recall that

$$(\delta_{det}(\{(f, X_f)\}, w), \varepsilon) =: X'_w$$

is the single accepting state in $\mathcal{A}'_{det}$ that is reachable from $(\{\bar{q}_0\}, 0)$. We call a branch of $\mathcal{B}'_{det}$ *accepting* if it contains X'_w infinitely often.

Suppose "(ii)", i.e., the probability that some branch of a $\mathcal{B}[X_f]$-tree has a (non-accepting or accepting) run in $\mathcal{A}[f, X_f]$ is positive. Thus, the probability that some branch of a $\mathcal{B}[X_f]$-tree has a run in $\mathcal{A}_{det}$ that does not enter the state $\emptyset$ is positive. Let $X_1 w' \in \Gamma^+$ be such that

$$(f, X_f) \in \delta_{det}(\{(f, X_f)\}, w X_1 w')\,.$$

Then, also the probability that some branch of a $\mathcal{B}[X_f]$-tree starts with $X_f w X_1 w'$ and has a run in $\mathcal{A}_{det}$ that does not enter $\emptyset$ is positive. Thus, the probability that some branch of a $\mathcal{B}[X_1]$-tree starts with $X_1 w'$ and has a run in $\mathcal{A}_{det}$, started in $\delta_{det}(\{(f, X_f)\}, w)$, that does not enter $\emptyset$ is positive. Hence, the probability that some branch of a $\mathcal{B}[X_1]$-tree has a run in $\mathcal{A}'_{det}$, started in $(\delta_{det}(\{(f, X_f)\}, w), \varepsilon)$, that does not enter a state of the form $(\emptyset, v)$ is positive. From the construction of $\mathcal{B}'_{det}$ it follows that the probability that some branch of a $\mathcal{B}'_{det}[X'_w]$-tree (i.e., with $X'_w = (\delta_{det}(\{(f, X_f)\}, w), \varepsilon)$ as start type) does not have a node of a type of the form $(\emptyset, v)$ is positive.

Consider any type (U_0, v_0) in $\mathcal{B}'_{det}$ with $U_0 \neq \emptyset$ and view it as a state in $\mathcal{A}'_{det}$. Then there is $u_1 \in \Gamma^*$ with

$$(U_0, v_0) \overset{u_1}{\rightarrow}{}^* (U_1, v_1)$$

where $U_1 \neq \emptyset$ and $v_1 \in \{0, \varepsilon\}$. Let $u_2 \in \Gamma^*$ be a shortest word such that there is $(q_1, X_1) \in U_1$ with

$$(q_1, X_1) \xrightarrow{u_2}{}^* (f, X_f)$$

in $\mathcal{A}[f, X_f]$. It follows that in $\mathcal{A}'_{det}$ we have

$$(U_0, v_0) \xrightarrow{u_1 u_2}{}^* (U_2, w) \xrightarrow{w}{}^* (U_3, \varepsilon)$$

with $(f, X_f) \in U_2$. By Lemma 6.14 we have

$$U_3 = \delta_{det}(\{(f, X_f)\}, w)\,.$$

Hence $(U_0, v_0) \to^* X'_w$.

Combining this reachability fact with the previous argument, we infer that the probability that some branch of a $\mathcal{B}'_{det}[X'_w]$-tree has only nodes of types from which X'_w is reachable (in $\mathcal{B}'_{det}$) is positive. It follows from [17, Lemma 11] that the probability that some branch of a $\mathcal{B}'_{det}[X'_w]$-tree is accepting is positive.[2] Hence, the probability that some branch of a $\mathcal{B}'_{det}$-tree is accepting is positive. From the construction of $\mathcal{B}'_{det}$ it follows that the probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}'_{det}$ is positive, i.e., "(iii)". □

6.5 Co-Unambiguous Büchi Automata

In this section we build on the previous section, in particular on Lemma 6.11, to derive our main technical result: given a BP $\mathcal{B}$ and an *unambiguous* Büchi automaton (UBA) $\mathcal{A}$, one can decide in NC whether $\mathcal{B}$ almost surely generates a tree all whose branches are rejected by $\mathcal{A}$:

Theorem 6.16. *The problem* $\mathbb{P}(\text{coUBA}) = 1$ *is in NC.*

The rest of the section is devoted to the proof of this theorem. Fix a BP $\mathcal{B}$ and a UBA $\mathcal{A}$. Since NC is closed under complement, we can focus on the problem whether the probability is positive that a $\mathcal{B}$-tree has some branch accepted by $\mathcal{A}$. We use Lemma 6.10. Since reachability in a graph is in NL and, hence, in NC, it suffices to decide in NC whether the probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive. By Lemma 6.11 it suffices to decide in NC whether

[2] In terms of the notation therein, we instantiate [17, Lemma 11] with $X := X'_w$. By the reachability argument above, $N_X = N_{X'_w}$ does not include types of the form (U, v) with $U \neq \emptyset$. Thus, we have argued that the probability of $\Diamond N_{X'_w}$ is $p < 1$. Hence, [17, Lemma 11] asserts that the probability that a $\mathcal{B}'_{det}[X'_w]$-tree has an X'_w-*branch* equals $1 - p > 0$, where X'_w-*branch* means accepting path in terms of our definition.

the probability that some branch of a $\mathcal{B}_{det}$-tree does not have any nodes of type $\emptyset$ is positive. The challenge is that $\mathcal{B}_{det}$ may be exponentially larger than $\mathcal{A}$, so we need to exploit the unambiguousness of $\mathcal{A}$ and the regular structure it gives to $\mathcal{B}_{det}$.

Let $\mathcal{B}''_{det}$ be the BP (with ε-rules allowed) obtained from $\mathcal{B}_{det}$ by removing the type $\emptyset$ and eliminating all occurrences of type $\emptyset$ from all right-hand sides. The probability that a $\mathcal{B}_{det}$-tree has an infinite branch of non-$\emptyset$ nodes is equal to the probability that a $\mathcal{B}''_{det}$-tree is infinite. Hence, it remains to show that one can decide in NC whether the probability that a $\mathcal{B}''_{det}$-tree is infinite is positive.

Define a matrix $M \in \mathbb{Q}^{Q[f,X_f]\times Q[f,X_f]}$ whose rows and columns are indexed with the non-$\bar{q}_0$ states of $\mathcal{A}[f,X_f]$:

$$M_{(q,X),(r,Y)} := \begin{cases} \sum_{X \overset{p}{\hookrightarrow} u} p|u|_Y & \text{if } (q,X) \overset{Y}{\to} (r,Y) \text{ in } \mathcal{A}[f,X_f] \\ 0 & \text{otherwise}, \end{cases}$$

where $|u|_Y \in \mathbb{N}_0$ is the number of occurrences of Y in u. (Think of $M_{(q,X),(r,Y)}$ as the expected number of (r,Y)-"successors" of (q,X).) The graph of M is equal to the transition graph of $\mathcal{A}[f,X_f]$ (excluding $\bar{q}_0$), which is strongly connected.

Say that $\mathcal{A}[f,X_f]$ has *proper branching* if there exist $(q,Y) \overset{Z_1}{\to} (r_1,Z_1)$ and $(q,Y) \overset{Z_2}{\to} (r_2,Z_2)$ in $\mathcal{A}[f,X_f]$ and a rule $Y \overset{p}{\hookrightarrow} u_1Z_1u_2Z_2u_3$ in $\mathcal{B}$ with $u_1,u_2,u_3 \in \Gamma^*$. Now we can state the key lemma:

Lemma 6.17. *Let ρ be the spectral radius of M. The probability that a $\mathcal{B}''_{det}$-tree is infinite is positive if and only if either $\rho > 1$ or $\rho = 1$ and $\mathcal{A}[f,X_f]$ does not have proper branching.*

Observe the similarity between Lemmas 6.3 and 6.17. In fact, the proof of Lemma 6.17, given below, is based on Lemma 6.3. Lemma 6.17 shows that properties of $\mathcal{A}[f,X_f]$ and M (which are polynomial-sized objects) determine a property of the exponential-sized BP $\mathcal{B}''_{det}$. Unambiguousness of $\mathcal{A}[f,X_f]$ is crucial for that connection.

Proof of Lemma 6.17. The automaton $\mathcal{A}[f,X_f]$ is unambiguous, as $\mathcal{A}$ is unambiguous, and has (f,X_f) as (the only) accepting state. Recall also that (f,X_f) is reachable from all states in $\mathcal{A}[f,X_f]$. It follows that $\mathcal{A}[f,X_f]$ does not have *diamonds*, i.e., $\mathcal{A}[f,X_f]$ does not have states $(q,X),(q',X')$ and a word $u \in \Gamma^*$ such that $\mathcal{A}[f,X_f]$ has two different paths $(q,X) \overset{u}{\to}{}^* (q',X')$.

By Theorem 2.7.1, M has an eigenvector $v \in (0,\infty)^{Q[f,X_f]}$ (all entries positive) with $Mv = \rho v$. (Think of the entries of v as "weights" of the states in $\mathcal{A}[f,X_f]$. Loosely speaking, the equality $(Mv)_{(q,X)} = \rho v_{(q,X)}$ expresses that the expected

combined weight of the "successors" of (q, X) is equal to the weight of (q, X) multiplied by ρ.)

We "lift" v to define a vector $\bar{v} \in [0, \infty)^{\Gamma'}$ where Γ' is the set of types in $\mathcal{B}_{det}$ (recall that they are of the form $P \times \{X\}$ with $P \subseteq Q$ and $X \in \Gamma$):

$$\bar{v}_{P\times\{X\}} \;:=\; \sum_{q\in P} v_{(q,X)}$$

Denote by $\bar{M} \in \mathbb{Q}^{\Gamma'\times\Gamma'}$ the matrix defined before Lemma 6.3, but for $\mathcal{B}_{det}$. Then we have for all $P \times \{X\} \in \Gamma'$:

$$\begin{aligned}
(\bar{M}\bar{v})_{P\times\{X\}} \;&=\; \sum_{X \overset{p}{\hookrightarrow} X_1\cdots X_k} p \sum_{i=1}^{k} \bar{v}_{\delta_{det}(P\times\{X\},X_i)} \\
&=\; \sum_{X \overset{p}{\hookrightarrow} X_1\cdots X_k} p \sum_{i=1}^{k} \sum_{(r,X_i)\in\delta_{det}(P\times\{X\},X_i)} v_{(r,X_i)} \\
&\overset{*}{=}\; \sum_{X \overset{p}{\hookrightarrow} X_1\cdots X_k} p \sum_{i=1}^{k} \sum_{q\in P} \sum_{r:(q,X)\xrightarrow{X_i}(r,X_i)} v_{(r,X_i)} \\
&=\; \sum_{q\in P} \sum_{X \overset{p}{\hookrightarrow} X_1\cdots X_k} p \sum_{i=1}^{k} \sum_{r:(q,X)\xrightarrow{X_i}(r,X_i)} v_{(r,X_i)} \\
&=\; \sum_{q\in P} (Mv)_{(q,X)} \\
&=\; \sum_{q\in P} \rho v_{(q,X)} \\
&=\; \rho \bar{v}_{P\times\{X\}} \;,
\end{aligned}$$

where the third equality (marked with $\overset{*}{=}$) holds as for any $(r, X_i) \in \delta_{det}(P\times\{X\}, X_i)$ there is exactly one $q \in P$ with $(q, X) \xrightarrow{X_i} (r, X_i)$ in $\mathcal{A}[f, X_f]$. Indeed, towards a contradiction, suppose there are $q_1, q_2 \in Q$ with $q_1 \neq q_2$ and $(q_j, X) \xrightarrow{X_i} (r, X_i)$ for both $j \in \{1, 2\}$. Since $P \times \{X\}$ is reachable in $\mathcal{A}_{det}$ from $\{(f, X_f)\}$, there is $u \in \Gamma^*$ with

$$(f, X_f) \xrightarrow{u}{}^{*} (q_j, X) \xrightarrow{X_i} (r, X_i)$$

in $\mathcal{A}[f, X_f]$ for both $j \in \{1, 2\}$, contradicting the absence of diamonds in $\mathcal{A}[f, X_f]$. We conclude from the above computation that $\bar{M}\bar{v} = \rho\bar{v}$; i.e., $\bar{v}$ is an eigenvector of $\bar{M}$ with eigenvalue ρ.

In the following, for subsets $\Delta \subseteq \Gamma'$, we write $\bar{M}_\Delta \in \mathbb{Q}^{\Delta\times\Delta}$ for the (square) principal submatrix obtained from $\bar{M}$ by restricting it to the rows and columns indexed by elements of Δ. Similarly, define $\bar{v}_\Delta \in [0, \infty)^\Delta$ by restricting $\bar{v}$ to

the entries indexed by elements of Δ. Since $\bar{M}$ and $\bar{v}$ are non-negative, we have $\bar{M}_\Delta \bar{v}_\Delta \leq \rho \bar{v}_\Delta$ (the inequality is meant componentwise).

Note that $\bar{v}_\emptyset = 0$. Define $\Gamma'' := \Gamma' \setminus \{\emptyset\}$. Thus we have $\bar{M}_{\Gamma''} \bar{v}_{\Gamma''} = \rho \bar{v}_{\Gamma''}$. All entries of $\bar{v}_{\Gamma''}$ are positive, as all entries of v are. By Theorem 2.7.5 it follows that ρ is the spectral radius of $\bar{M}_{\Gamma''}$. The matrix $\bar{M}_{\Gamma''}$ is equal to the matrix defined before Lemma 6.3, but for $\mathcal{B}''_{det}$. We complete the proof with the following case distinction.

- Suppose $\rho > 1$. Let $\Delta \subseteq \Gamma''$ be a bottom strongly connected component of the graph of $\bar{M}_{\Gamma''}$. As Δ is bottom, $\bar{M}_\Delta \bar{v}_\Delta = \rho \bar{v}_\Delta$. So the spectral radius of $\bar{M}_\Delta$ is at least $\rho > 1$ (in fact, it must be equal to ρ). It follows that Δ is supercritical in $\mathcal{B}''_{det}$. Thus, by Lemma 6.3, a $\mathcal{B}''_{det}$-tree is infinite with positive probability.

- Suppose that $\rho = 1$ and that $\mathcal{A}[f, X_f]$ does not have proper branching. Let $\Delta \subseteq \Gamma''$ be a bottom strongly connected component of the graph of $\bar{M}_{\Gamma''}$. Then $\bar{M}_\Delta \bar{v}_\Delta = \bar{v}_\Delta$, so by Theorem 2.7.5 the spectral radius of $\bar{M}_\Delta$ is 1. By the absence of proper branching we also have $\bar{M}_\Delta \vec{1} \leq \vec{1}$, where $\vec{1}$ denotes the all-1 vector, i.e., the element of $\{1\}^\Delta$. By Theorem 2.7.4 it follows that $\bar{M}_\Delta \vec{1} = \vec{1}$. Thus, Δ is linear in $\mathcal{B}''_{det}$. Hence, by Lemma 6.3, a $\mathcal{B}''_{det}$-tree is infinite with positive probability.

- Suppose that $\rho = 1$ and that $\mathcal{A}[f, X_f]$ has proper branching, i.e., there exist

$$(q, Y) \xrightarrow{Z_1} (r_1, Z_1) \text{ and } (q, Y) \xrightarrow{Z_2} (r_2, Z_2) \text{ in } \mathcal{A}[f, X_f]$$

  and a rule $Y \overset{p}{\hookrightarrow} u_1 Z_1 u_2 Z_2 u_3$ with $u_1, u_2, u_3 \in \Gamma^*$. Consider any strongly connected component $\Delta \subseteq \Gamma''$ of the graph of $\bar{M}_{\Gamma''}$. Denote by ρ_Δ the spectral radius of $\bar{M}_\Delta$. As $\bar{M}_\Delta$ is a principal submatrix of $\bar{M}$, we have $\rho_\Delta \leq \rho = 1$ by Theorem 2.6.2. So Δ is not supercritical.

  (i) $\rho_\Delta < 1$. We have argued before Lemma 6.3 that Δ being linear would imply $\rho_\Delta = 1$. Hence, Δ is not linear.

  (ii) $\rho_\Delta = 1$. Recall that $\bar{M}_\Delta \bar{v}_\Delta \leq \bar{v}_\Delta$, so by Theorem 2.7.4 we must have $\bar{M}_\Delta \bar{v}_\Delta = \bar{v}_\Delta$. Let $P \times \{X\} \in \Delta$ and $p \in P$. Let $u_0 \in \Gamma^*$ with $(p, X) \xrightarrow{u_0}{}^* (q, Y)$ in $\mathcal{A}[f, X_f]$. Hence

$$(p, X) \xrightarrow{u_0}{}^* (q, Y) \xrightarrow{Z_j} (r_j, Z_j)$$

  in $\mathcal{A}[f, X_f]$ for both $j \in \{1, 2\}$. Towards a contradiction, suppose Δ is linear. If $\delta_{det}(P \times \{X\}, u_0)$ is not in Δ, then neither are $\delta_{det}(P \times$

$\{X\}, u_0 Z_1)$ or $\delta_{det}(P \times \{X\}, u_0 Z_2)$. Otherwise (i.e., $\delta_{det}(P \times \{X\}, u_0) \in \Delta$) there is $j \in \{1,2\}$ such that $\delta_{det}(P \times \{X\}, u_0 Z_j) \notin \Delta$, as Δ is linear. Either way there is $j \in \{1,2\}$ such that

$$\delta_{det}(P \times \{X\}, u_0 Z_j) \notin \Delta\,.$$

Note that $(r_j, Z_j) \in \delta_{det}(P \times \{X\}, u_0 Z_j) \neq \emptyset$. Let xZ (with $x \in \Gamma^*$ and $Z \in \Gamma$) be the shortest prefix of $u_0 Z_j$ such that $U := \delta_{det}(P \times \{X\}, x) \in \Delta$ but $\delta_{det}(U, Z) \notin \Delta$. Then we have:

$$\begin{aligned} \bar{v}_U &= (\bar{M}_\Delta \bar{v}_\Delta)_U \\ &< (\bar{M}_\Delta \bar{v}_\Delta)_U + \bar{M}_{U,\delta_{det}(U,Z)} \bar{v}_{\delta_{det}(U,Z)} \\ &\leq (\bar{M}\bar{v})_U \\ &= \bar{v}_U\,, \end{aligned}$$

a contradiction. Thus, Δ is not linear.

We conclude that in both cases Δ is neither linear nor supercritical. Since Δ was an arbitrary strongly connected component, we conclude from Lemma 6.3 that a $\mathcal{B}''_{det}$-tree is almost surely finite.

- Suppose $\rho < 1$. Then, for any strongly connected component Δ the spectral radius of $\bar{M}_\Delta$ is also less than 1 by Theorem 2.6.2, so Δ is neither supercritical nor linear. Thus, by Lemma 6.3, a $\mathcal{B}''_{det}$-tree is almost surely finite.

Hence, the probability that a $\mathcal{B}''_{det}$-tree is infinite is positive if and only if either $\rho > 1$ or $\rho = 1$ and $\mathcal{A}[f, X_f]$ does not have proper branching. □

Given that Lemma 6.17 reflects the key insight of this section (if not of this chapter), let us comment further. Suppose $\mathcal{A}[f, X_f]$ has two outgoing transitions in a state (q, Y), say $(q, Y) \xrightarrow{Z_1} (r_1, Z_1)$ and $(q, Y) \xrightarrow{Z_2} (r_2, Z_2)$. This branching could be "proper branching" as defined before Lemma 6.17, or the original UBA $\mathcal{A}$ could be non-deterministic when reading Y in q and have transitions $q \xrightarrow{Y} r_1$ and $q \xrightarrow{Y} r_2$. Either type of branching causes non-0 entries in the matrix M and, intuitively, increases its spectral radius ρ. Lemma 6.17 tells us that the probability that a $\mathcal{B}''_{det}$-tree is infinite is governed by the *combined* effect on ρ of both types of branching: if $\rho > 1$ then a $\mathcal{B}''_{det}$-tree is infinite with positive probability; only in the borderline case, $\rho = 1$, the type of branching matters. Again, this characterization is only correct if the non-determinism in $\mathcal{A}$ does not cause ambiguousness.

Let us consider what Lemma 6.17 states for the special case of Markov chains. In that case, clearly there is no proper branching. One can show, using unambiguousness, that for Markov chains the spectral radius ρ of the matrix M is at most 1. Hence, Lemma 6.17 states for Markov chains that the probability that a $\mathcal{B}''_{det}$-tree (consisting of a single branch) is infinite is positive if and only if $\rho = 1$. Indeed, a related statement can be found in [6, Lemma 6].

To finish the proof of Theorem 6.16 it suffices to show that we can check the conditions of Lemma 6.17 in NC. Indeed, for comparing the spectral radius with 1, we employ Lemma 2.8. One can check for proper branching in logarithmic space, hence in NC. This completes the proof of Theorem 6.16.

6.6 Linear temporal logic

We can now translate LTL formulas to unambiguous automata and invoke Theorem 6.16 to show our headline result:

Theorem 6.18. *The problem* $\mathbb{P}(\mathrm{LTL}) = 1$ *is PSPACE-complete.*

Proof. PSPACE-hardness is immediate in two different ways. It follows both from the PSPACE-hardness of model checking Markov chains against LTL and from the PSPACE-hardness of model checking transition systems against LTL [65]. Both model-checking problems are special cases of $\mathbb{P}(\mathrm{LTL}) = 1$.

Towards membership in PSPACE, there is a classical PSPACE procedure that translates an LTL formula into an (exponential-sized) Büchi automaton [70]. As noted by several authors (e.g., [16; 21]), this procedure can easily be adapted to ensure that the Büchi automaton be a UBA. By applying this translation to the negation $\neg\varphi$ of the input formula φ, we obtain a UBA that rejects exactly those words that satisfy φ. By Theorem 6.16 the problem $\mathbb{P}(\mathrm{coUBA}) = 1$ is in NC. By Lemma 2.5 it follows that $\mathbb{P}(\mathrm{LTL}) = 1$ is in PSPACE. □

Finally we show the following result, exhibiting a big complexity gap between the problems $\mathbb{P}(\mathrm{LTL}) = 1$ and $\mathbb{P}(\mathrm{LTL}) = 0$.

Theorem 6.19. *The problem* $\mathbb{P}(\mathrm{LTL}) = 0$ *is 2EXPTIME-complete.*

The proof can be found in Section 6.7.

6.7 Proofs of $\mathbb{P}(\_\_) = 0$

This section contains the proofs of the $\mathbb{P}(\_\_) = 0$ variants on the right hand side of Table 6.1.

Theorem 6.7. *The problem* $\mathbb{P}(\mathrm{DPA}) = 0$ *is P-complete. It is P-hard even for deterministic Büchi automata with two states, the accepting state being a sink.*

Proof. Membership in P was shown in [17, Theorem 15]. For P-hardness we reduce from the monotone circuit value problem. Given a monotone circuit with output gate g_{out}, we construct a BP $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_{g_{out}})$ and a DBA $\mathcal{A}$ such that $\mathbb{P}_{\mathcal{B}}(\mathcal{A} \text{ accepts}) > 0$ if and only if g_{out} evaluates to 1.

For each $\wedge$- and each $\vee$-gate g include a type $X_g \in \Gamma$. Also include types $X_0, X_1 \in \Gamma$ for the inputs $0, 1$, respectively. For each $\wedge$-gate g with children $g_1, \ldots, g_k$ include a rule

$$X_g \overset{1}{\hookrightarrow} X_{g_1} \cdots X_{g_k} .$$

For each $\vee$-gate g with children $g_1, \ldots, g_k$ include rules

$$X_g \overset{1/k}{\hookrightarrow} X_{g_1}, \ \ldots, \ X_g \overset{1/k}{\hookrightarrow} X_{g_k} .$$

Include rules

$$X_0 \overset{1}{\hookrightarrow} X_0 \text{ and } X_1 \overset{1}{\hookrightarrow} X_1 .$$

Construct a two-state DBA $\mathcal{A}$ that accepts exactly those $w \in \Gamma^\omega$ that contain X_1. It follows from a straightforward induction over the gate height (longest distance to an input) that, for any gate g, it evaluates to 1 if and only if $\mathbb{P}_{X_g}(\mathcal{A} \text{ accepts}) > 0$. □

Theorem 6.9. *The problem* $\mathbb{P}(\mathrm{NBA}) = 0$ *is EXPTIME-complete. It is EXPTIME-hard even for NBAs whose only accepting state is a sink.*

Proof. Towards membership in EXPTIME, an NBA can be translated, in exponential time, to a DPA of exponential size; see, e.g., [56]. Since $\mathbb{P}(\mathrm{DPA}) = 0$ is in P by Theorem 6.7, it follows that $\mathbb{P}(\mathrm{NBA}) = 0$ is in EXPTIME.

Concerning EXPTIME-hardness, we adapt the proof of [27, Theorem 17] on model checking *recursive Markov chains* against NBAs. Recall that alternating PSPACE equals EXPTIME. We give a polynomial-time reduction from the problem of acceptance of a word by a PSPACE-bounded alternating Turing machine. Without loss of generality, we can assume that the Turing machine is linear-bounded, i.e., uses only the space occupied by the input word.

Let $M = (S_\exists, S_\forall, \Sigma, T, s_0, s_{acc})$ be a linear-bounded alternating Turing machine. Let $w = a_1 \cdots a_n \in \Sigma^*$ be the input word. As mentioned before, we can assume that M uses exactly n tape cells. We construct a BP $\mathcal{B}$ and an NBA $\mathcal{A}$ such that $\mathbb{P}_\mathcal{B}(\mathcal{A} \text{ accepts}) > 0$ if and only M accepts w.

The BP $\mathcal{B}$ has the following set of types:

$$\Gamma \;=\; (\{1,\ldots,n\} \times S \times \Sigma) \;\cup\; (\{1,\ldots,n\} \times T) \;\cup\; (\{1,\ldots,n\} \times \{chk\}) \;\cup\; \{E\}$$

Intuitively, a type (i, s, a) means that the head is at position i, the current state is s, and the head is reading letter a; a type (i, t) means that the head is at position i, and transition t is being executed; a type (i, chk) means that the accepting state has been reached, and cell i is being "checked" (in a sense to be explained later); type E indicates an error. The type $(1, s_0, a_1)$ is the start type of $\mathcal{B}$.

For all $(i, s, a) \in \Gamma$ with $s \in S_\exists$, include rules

$$(i, s, a) \xhookrightarrow{1/k} (i, t_j) \quad (1 \le j \le k)\,,$$

where $\{t_1, \ldots, t_k\} = T_{s,a}$. (Intuitively, a transition going out of an existential state is chosen as the only child, uniformly at random.) For all $(i, s, a) \in \Gamma$ with $s \in S_\forall$, include a rule

$$(i, s, a) \xhookrightarrow{1} (i, t_1) \cdots (i, t_k)\,,$$

where $\{t_1, \ldots, t_k\} = T_{s,a}$. (Intuitively, all possible transitions going out of a universal state are children.) For all $(i, (s, a, a', D, s')) \in \Gamma$ with $1 \le i{+}D \le n$, include rules

$$(i, (s, a, a', D, s')) \xhookrightarrow{1/|\Sigma|} (i{+}D, s', b_j) \quad (1 \le j \le |\Sigma|)\,,$$

where $\{b_1, \ldots, b_{|\Sigma|}\} = \Sigma$. (Intuitively, the letter in the cell at the new head position $i{+}D$ is guessed uniformly at random.) For all $(i, (s, a, a', D, s')) \in \Gamma$ with $i{+}D \in \{0, n{+}1\}$, include a rule

$$(i, (s, a, a', D, s')) \xhookrightarrow{1} E\,.$$

(Intuitively, when the space bound is exceeded, move to the error type E.) Include a rule $E \xhookrightarrow{1} E$ (i.e., a self-loop). For all $(i, s_{acc}, a) \in \Gamma$, include a rule

$$(i, s_{acc}, a) \xhookrightarrow{1} (1, chk) \cdots (n, chk)\,.$$

(Intuitively, after reaching s_{acc} all n cells are "checked".) For all $(i, chk) \in \Gamma$, include a rule $(i, chk) \xhookrightarrow{1} (i, chk)$ (i.e., a self-loop).

The NBA $\mathcal{A} = (Q, \Gamma, \delta, Q_0, \{f\})$ has the following set of states:

$$Q \;=\; (\{1, \ldots, n\} \times \Sigma) \;\cup\; \{f\}$$

The set of initial states is $Q_0 = \{(1, a_1), \ldots, (n, a_n)\}$ (recall that $a_1 \cdots a_n$ is the input word). The idea is that if a prefix $X_1 \cdots X_k \in \Gamma^*$ of a tree branch corresponds to a prefix of a correct computation of M, then the set of automaton states in $\delta(Q_0, X_1 \cdots X_k)$ corresponds to the tape after this computation prefix. In fact, the transition relation $\delta \subseteq Q \times \Gamma \times Q$ is *deterministic*, i.e., for all $q \in Q$ and $X \in \Gamma$ there is at most one q' with $(q, X, q') \in \delta$. Moreover, for any transition $((i, a), X, (j, a')) \in \delta$ we will have $i = j$.

For $(i, a) \in Q$ and all $(j, s, b) \in \Gamma$ with $i \neq j$ or $a = b$, include a self-loop

$$((i, a), (j, s, b), (i, a)) \in \delta\,.$$

(Intuitively, letter a in cell i is compatible with the head being on cell j and reading letter b.) For all $(i, a) \in Q$ and all $(j, t) \in \Gamma$ with $i \neq j$, include a self-loop

$$((i, a), (j, t), (i, a)) \in \delta\,.$$

(Intuitively, the content of cell i stays unchanged when the head is at position j.) For all i, s, a, a', D, s' with $(i, a) \in Q$ and $(i, (s, a, a', D, s')) \in \Gamma$, include a transition

$$((i, a), (i, (s, a, a', D, s')), (i, a')) \in \delta\,.$$

(Intuitively, the transition (s, a, a', D, s') changes the content of cell i from a to a'.) For all $(i, a) \in Q$, include a transition

$$((i, a), (i, \mathit{chk}), f) \in \delta\,.$$

(Intuitively, the type (i, chk) checks if the computation has been consistent in cell i.) For all $X \in \Gamma$, include a self-loop $(f, X, f) \in \delta$.

We will show that in this case, $\mathbb{P}_{\mathcal{B}}(\mathcal{A} \text{ accepts}) > 0$ if and only if M accepts w. Firstly note that f is a sink state in $\mathcal{A}$, and hence, if $\mathcal{A}$ reaches f after reading some prefix of a branch, it will accept any branch with this prefix. This means that a tree t is accepted by $\mathcal{A}$ if and only if there exists a finite prefix of t such that $\mathcal{A}$ reaches f on all of its branches.

Assume that M accepts w. Then there exists a linear-bounded winning strategy σ for the existential player. We will write c_0 for the initial configuration of M on w, R for the behaviour of σ, and N for the length of the longest run in R. We will show that there exists a finite prefix generated by $\mathcal{B}$ with non-zero probability that

precisely models this strategy, and that is such that $\mathcal{A}$ reaches f on all of its branches. Since any tree with this prefix is accepted, this implies that $\mathbb{P}_{\mathcal{B}}(\mathcal{A} \text{ accepts}) > 0$ if M accepts w.

Let t be the tree defined as follows:

- the root of t is $(1, s_0, a_1)$,
- for any node (p_i, s_i, b_i) on level $2i$ with $s_i \in S_\exists$, let $t_1 \dots t_i$ be such that the branch from the root to (p_i, s_i, b_i) has nodes (p_j, t_{j+1}) on level $2j+1$ for each $0 \leq j < i$. Then (p_i, s_i, b_i) has a child $(p_i, \sigma(c_0 \triangleright t_1 \dots t_i, t_1 \dots t_i))$,
- for any node (p_i, s_i, b_i) on level $2i$ with $s_i \in S_\forall$, let $t_1 \dots t_i$ be such that the branch from the root to (p_i, s_i, b_i) has nodes (p_j, t_{j+1}) on level $2j+1$ for each $0 \leq j < i$. Let $w_i = \pi_{\Sigma^*}(c_0 \triangleright t_1 \dots t_i)$. Then (p_i, s_i, b_i) has children (p_i, t_{i+1}) for each $t_{i+1} \in T_{s_i, w_i(p_i)}$,
- for any node (p_i, t_{i+1}) with $t_{i+1} = (s, a, a', D, s')$ on level $2i+1$, let $t_1 \dots t_i$ be such that the branch from the root to (p_i, t_{i+1}) has nodes (p_j, t_{j+1}) on level $2j+1$ for each $0 \leq j < i$. Let $c_{i+1} = c_0 \triangleright t_1 \dots t_{i+1}$. Then (p_i, t_{i+1}) has a child $(p_i + D, s', \pi_{\Sigma^*}(c_{i+1})(p_i + D))$,
- any node (p_i, s_{acc}, b_i) has children (j, chk) for each $1 \leq j \leq n$,
- and any node (j, chk) has a child (j, chk).

By construction, t is now such that for any node (p_i, t_{i+1}) with $t_{i+1} = (s, a, a', D, s')$ on level $2i+1$, there exist runs in R prefixed by $t_1 \dots t_{i+1}$, where $t_1 \dots t_i$ are such that the branch from the root to (p_i, t_{i+1}) has nodes (p_j, t_{j+1}) on level $2j+1$ for each $0 \leq j < i$. Hence, by the fact that σ is linear-bounded, $1 \leq p_i + D \leq n$ and thus all the transitions in t are according to the rules of $\mathcal{B}$. Moreover, since the length of runs in R is bounded by N, all the nodes at level $2N+1$ must be of the form (i, chk), and since all states (i, chk) are sink states, t is generated with a non-zero probability. Finally pick any state (i, chk) in t. The only types from which $(i, a) \in Q$ does not have an outgoing edge are of the form (i, s, b). However, for any state (i, s, b) at level $2j$ on the branch to (i, chk), let $t_1 \dots t_j$ be such that the branch from the root to (i, s, b) has nodes (p_k, t_{k+1}) on level $2k+1$ for each $0 \leq k < j$. Then $(i, b) = (i, \pi_{\Sigma^*}(c_0 \triangleright t_1 \dots t_j)(i)) = \delta((i, w(i)), (p_0, t_1) \dots (p_{j-1}, t_j))$ and hence (i, a) survives, and reaches f upon reading (i, chk).

For the other direction, assume that $\mathbb{P}_{\mathcal{B}}(\mathcal{A} \text{ accepts}) > 0$. Then there exists a prefix t generated by $\mathcal{B}$ with branches of length N for some N such that $\mathcal{A}$ reaches f

on all of its branches. W.l.o.g. we can assume that $\pi_S(c_0) \neq s_{acc}$, because otherwise any strategy is winning. Note that for any accepted tree t and any branch prefix $(1, s_0, a_1)(1, t_1) \dots (p_i, s_i, b_i)(p_i, t_{i+1})$ in t, the set of reachable states in the automaton from Q_0 reflects the tape contents of M, ie. $\delta(Q_0, (1, s_0, a_1)(1, t_1) \dots (p_i, s_i, b_i)(p_i, t_{i+1})) = \{(j, w(j)) \mid 1 \leq j \leq n, w = \pi_{\Sigma^*}(c_0 \triangleright t_1 \dots t_{i+1})\}$. Also $s_i = \pi_S(c_0 \triangleright t_1 \dots t_i)$. If this was not the case, then there would exist j such that $\delta((j, a_j), (1, s_0, a_1)(1, t_1) \dots (p_i, s_i, b_i)(p_i, t_{i+1})) = \emptyset$ and hence the branch reaching (j, chk) prefixed by $(1, s_0, a_1)(1, t_1) \dots (p_i, s_i, b_i)(p_i, t_{i+1})$ does not reach f. Let $n_0 n_1 \dots$ be any branch in t. Pick m such that $n_{2i+1} = (a_i, t_{i+1})$ for all $0 \leq i < m$ and $n_{2m+1} = (a_m, t_{m+1})$. Let $c = c_0 \triangleright t_1 \dots t_m$. Then we define σ to be any strategy such that for any such c, if $\pi_S(c) \in S_\exists$, then $\sigma(c, t_1 \dots t_m) = t_{m+1}$. This is a valid strategy since for any branch in t, the set of reachable states of the automaton after reading a branch prefix reflects the tape contents of M and the alphabet characters contained in the nodes of the branch prefix have to reflect the automaton states. Note that σ is n-bounded. We claim that σ is a winning strategy and hence M accepts w.

Let $r = t_1 t_2 \dots$ be a run consistent with σ. We will show that there exists a branch $(1, s_0, a_1)(1, t_1) \dots (p_i, t_{i+1})(p_{i+1}, s_{acc}, w_{i+1}(p_{i+1}))$ in t with $p_i = \pi_{\mathbb{N}}(c_0 \triangleright t_1 \dots t_i)$, $s_i = \pi_S(c_0 \triangleright t_1 \dots t_i)$, and $w_i = \pi_{\Sigma^*}(c_0 \triangleright t_1 \dots t_i)$ (and hence $|r| = i + 1$ and $\pi_S(c_0 \triangleright r) = s_{acc}$).

- For any branch prefix $(1, s_0, a_1)(1, t_1) \dots (p_j, t_{j+1})$ where $t_{j+1} = (s, a, a', D, s')$, according to the rules of $\mathcal{B}$, (p_j, t_{j+1}) has a single child $(p_i + D, s', b)$. Let $c_{j+1} = c_0 \triangleright t_1 \dots t_{j+1}$. Since trees prefixed by t are accepted, $p_i + D = \pi_{\mathbb{N}}(c_{j+1})$, $s' = \pi_S(c_{j+1})$, and $b = \pi_{\Sigma^*}(c_{j+1})(\pi_{\mathbb{N}}(c_{j+1}))$.

- For any branch prefix $(1, s_0, a_1)(1, t_1) \dots (p_{j-1}, t_j)(p_j, s_j, b_j)$ where $s_j \in S_\forall$, according to the rules of $\mathcal{B}$, (p_j, s_j, b_j) has children (p_j, t') for each $t' \in T_{s_j, b_j}$. Let $c_j = c_0 \triangleright t_1 \dots t_j$. Since trees prefixed by t are accepted, $s_j = \pi_S(c_j)$ and $b_j = \pi_{\Sigma^*}(c_j)(\pi_{\mathbb{N}}(c_j))$. Hence, $t_{j+1} \in T_{s_j, b_j}$ and (p_j, s_j, b_j) has a child (p_j, t_{j+1}).

- For any branch prefix $(1, s_0, a_1)(1, t_1) \dots (p_{j-1}, t_j)(p_j, s_j, b_j)$ where $s_j \in S_\exists$, according to the rules of $\mathcal{B}$, (p_j, s_j, b_j) has a single child (p_j, t'). By definition of σ, $\sigma(c_0 \triangleright t_1 \dots t_j, t_1 \dots t_j) = t'$.

Since t is finite, every branch reaches states of the form (p, s_{acc}, b) in a finite number of steps. Hence, r is finite and reaches s_{acc} and since r is any run consistent with σ, σ is an winning strategy. Thus, M accepts w.

□

Theorem 6.13. *The problem* $\mathbb{P}(\text{coNBA}) = 0$ *is EXPTIME-complete. It is EXPTIME-hard even for NBAs all whose states are accepting.*

Proof. Towards membership in EXPTIME, an NBA can be translated, in exponential time, to a DPA of exponential size; see, e.g., [56]. By shifting the priorities (colours) in the DPA by 1, we can make the DPA accept exactly those words that are rejected by the NBA. Since $\mathbb{P}(\text{DPA}) = 0$ is in P by Theorem 6.7, it follows that $\mathbb{P}(\text{coNBA}) = 0$ is in EXPTIME.

Concerning EXPTIME-hardness, we adapt the construction of the proof of Theorem 6.9. As in that proof, let $M = (S_\exists, S_\forall, \Sigma, T, s_0, s_{acc})$ be a linear-bounded alternating Turing machine. Let $w = a_1 \cdots a_n \in \Sigma^*$ be the input word, and assume again that M uses exactly n tape cells. We construct a BP $\mathcal{B}$ and an NBA $\mathcal{A}$ such that the probability that all branches of the random tree are rejected by $\mathcal{A}$ is positive if and only M accepts w.

For $\mathcal{B}$ we use almost the same construction as in Theorem 6.9, except that we do not need the types $(1, \mathit{chk}), \ldots, (n, \mathit{chk})$. We replace them by a single type *end*. Accordingly, for all $(i, s_{acc}, a) \in \Gamma$, we replace the rule

$$(i, s_{acc}, a) \overset{1}{\hookrightarrow} (1, \mathit{chk}) \cdots (n, \mathit{chk})$$

by a rule

$$(i, s_{acc}, a) \overset{1}{\hookrightarrow} \mathit{end}\,,$$

and include a rule $\mathit{end} \overset{1}{\hookrightarrow} \mathit{end}$ (i.e., a self-loop).

We want to construct the NBA $\mathcal{A}$ so that it accepts exactly those branches that correspond to infinite computations that do not arrive at s_{acc}, or to "non-computations", i.e., where the "guessing" in a rule

$$(i, (s, a, a', D, s')) \xhookrightarrow{1/|\Sigma|} (i{+}D, s', b_j) \quad (1 \leq j \leq |\Sigma|)\,,$$

has been wrong.

The NBA $\mathcal{A} = (Q, \Gamma, \delta, Q_0, Q)$ has the same set of states as in Theorem 6.9:

$$Q \;=\; (\{1, \ldots, n\} \times \Sigma) \;\cup\; \{f\}$$

As in Theorem 6.9, the set of initial states is $Q_0 = \{(1, a_1), \ldots, (n, a_n)\}$ (recall that $a_1 \cdots a_n$ is the input word). As in Theorem 6.9, the idea is that if a prefix $X_1 \cdots X_k \in \Gamma^*$ of a tree branch corresponds to a prefix of a correct computation of M, then the set of automaton states in $\delta(Q_0, X_1 \cdots X_k)$ corresponds to the tape

after this computation prefix. In fact, the transition relation $\delta \subseteq Q \times \Gamma \times Q$ is *deterministic*, i.e., for all $q \in Q$ and $X \in \Gamma$ there is at most one q' with $(q, X, q') \in \delta$. Moreover, for any transition $((i, a), X, (j, a')) \in \delta$ we will have $i = j$. Unlike in Theorem 6.9, all states are accepting.

For $(i, a) \in Q$ and all $(j, s, b) \in \Gamma$ with $i \neq j$ or $a = b$, include a self-loop

$$((i, a), (j, s, b), (i, a)) \in \delta\,.$$

(Intuitively, letter a in cell i is compatible with the head being on cell j and reading letter b.) For $(i, a) \in Q$ and all $(i, s, b) \in \Gamma$ with $a \neq b$, include a transition

$$((i, a), (i, s, b), f) \in \delta\,.$$

(Intuitively, letter a in cell i is *not* compatible with the head being on cell i and reading letter b; i.e., the prefix of the branch does not correspond to a correct computation.) For all $(i, a) \in Q$ and all $(j, t) \in \Gamma$ with $i \neq j$, include a self-loop

$$((i, a), (j, t), (i, a)) \in \delta\,.$$

(Intuitively, the content of cell i stays unchanged when the head is at position j.) For all i, s, a, a', D, s' with $(i, a) \in Q$ and $(i, (s, a, a', D, s')) \in \Gamma$, include a transition

$$((i, a), (i, (s, a, a', D, s')), (i, a')) \in \delta\,.$$

(Intuitively, the transition (s, a, a', D, s') changes the content of cell i from a to a'.) For all $X \in \Gamma$, include a self-loop $(f, X, f) \in \delta$. Note that (f, end, f) is the only transition labeled with *end*.

In this way:

- If a tree branch does not correspond to a correct computation, the automaton $\mathcal{A}$ enters the state f, remains there forever, and, thus, accepts.
- If a tree branch corresponds to an infinite computation not entering s_{acc}, the set of states that $\mathcal{A}$ can be in always reflects the tape. Thus, $\mathcal{A}$ accepts.
- If a tree branch corresponds to a computation entering s_{acc}, the branch also enters *end*, and $\mathcal{A}$ does not enter f. Thus, $\mathcal{A}$ rejects.

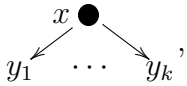
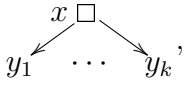
It follows that the probability that all branches of the random tree are rejected by $\mathcal{A}$ is positive if and only M accepts w. A more detailed argument would follow very similar lines as the proof of Theorem 6.9. □

Theorem 6.19. *The problem* $\mathbb{P}(\mathrm{LTL}) = 0$ *is 2EXPTIME-complete.*

Proof. For membership in 2EXPTIME, we use again the classical procedure that translates an LTL formula into an exponential-sized Büchi automaton [70] and then invoke Theorem 6.9.

For 2EXPTIME-hardness, our construction is inspired by the proof of 2EXPTIME-hardness of model-checking a concurrent probabilistic program (another name for MDP) against an LTL formula, see Theorem 3.2.1 in [20].

We will use the fact that 2EXPTIME is equal to alternating EXPSPACE. Let $M = (S_\exists, S_\forall, \Sigma, T, s_0, s_{acc})$ be an alternating Turing machine whose work tape usage is bounded by 2^n on any input of length n. Without loss of generality, we can assume that the machine has two possible next moves for each configuration and that it halts when it reaches the accepting state s_{acc}. For a given alternating TM M and an input w of length n, we will construct a BP $\mathcal{B}$ and an LTL formula φ both of size $\mathcal{O}(n)$ such that $\mathbb{P}_\mathcal{B}(\varphi) > 0$ if and only if M accepts w.

The branching process $\mathcal{B}$ is defined by the diagram in Figure 6.1. Every node in the diagram has a unique label that corresponds to a type of $\mathcal{B}$, although not all nodes are explicitly labelled. The ●-nodes represent randomising branching and □-nodes represent tree branching. Namely, if a ●-node x has k successors $y_1, \ldots, y_k$:  , then $\mathcal{B}$ has the rules $x \overset{1/k}{\hookrightarrow} y_i$ for $i = 1, \ldots, k$. On the other hand, if a □-node x has k successors $y_1, \ldots, y_k$:  , then $\mathcal{B}$ has the rule $x \overset{1}{\hookrightarrow} y_1 \cdots y_k$.

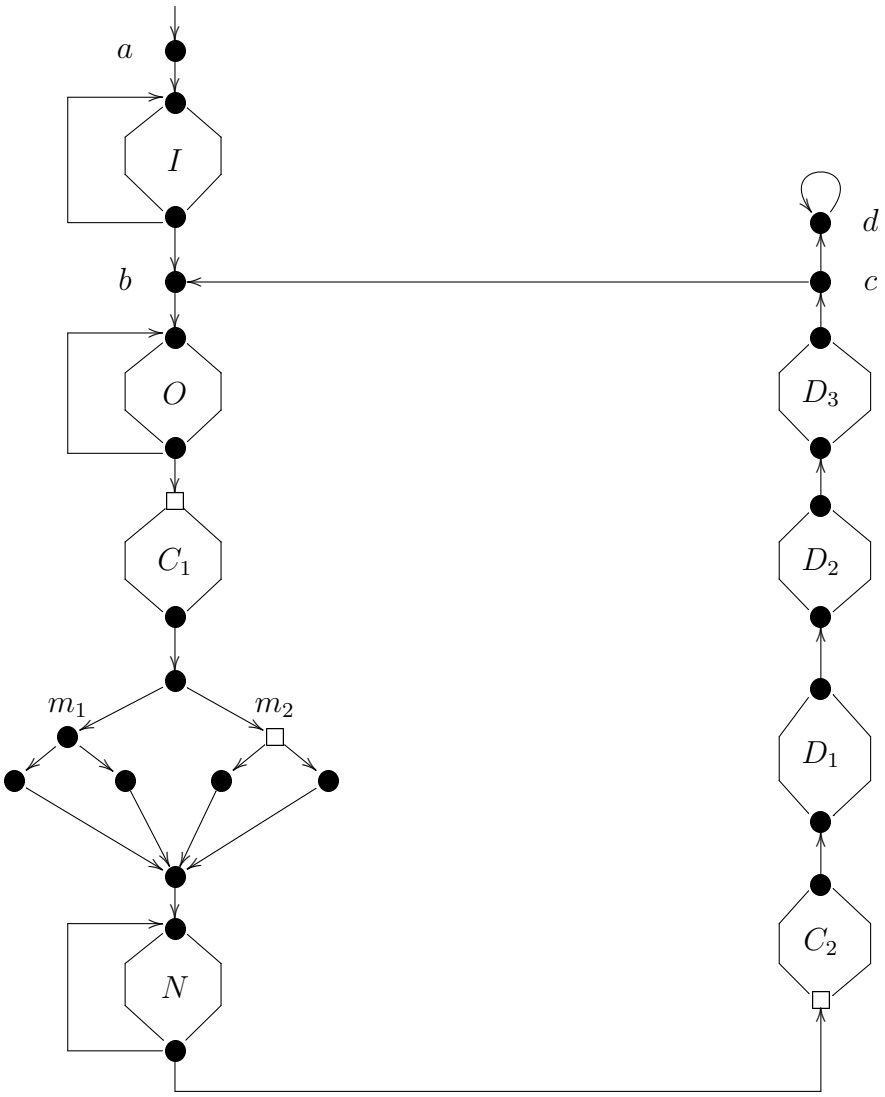


Figure 6.1: Diagram of the branching process $\mathcal{B}$.

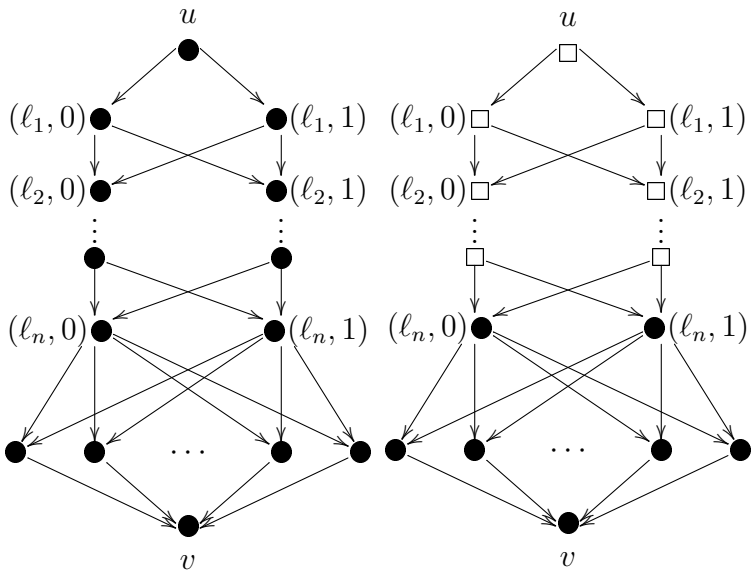


Figure 6.2: Diagrams of the randomising (left) and tree (right) branching blocks.

The start type of $\mathcal{B}$ is a. The detailed diagram of the blocks I, O, N, D_1, D_2 and D_3 is shown in Figure 6.2 on the left. All nodes in these blocks are ●-nodes. The diagram of the blocks C_1 and C_2 is shown in Figure 6.2 on the right. These blocks have □-nodes in the first $n+1$ levels, and the rest are ●-nodes. The initial and final nodes are labelled by u and v, respectively. The nodes below u are labelled with $(\ell_i, 0)$ and $(\ell_i, 1)$, $i = 1, \ldots, n$, as shown in the picture. Every block has its own unique set of labels u, v, $(\ell_i, 0)$ and $(\ell_i, 1)$, $i = 1, \ldots, n$; however we do not distinguish them in the diagram for simplicity.

In every block, except for I, the nodes above v are labelled by (ℓ_{n+1}, δ), where $\delta \in \Sigma \cup (S_\exists \cup S_\forall) \times \Sigma$ ranges over the symbols of the extended work tape alphabet. In block I, there are $n+1$ nodes above v which are labelled by (ℓ_{n+1}, γ_i) for $i = 1, \ldots, n+1$ such that for an input word $w = w_1 \ldots w_n$ we have $\gamma_1 = (s_0, w_1)$, $\gamma_i = w_i$ for $1 < i \leq n$, and $\gamma_{n+1} = \square$ is the blank symbol. These nodes will define the initial configuration of the Turing machine M.

The intended behaviour of process $\mathcal{B}$ is as follows. It starts generating a tree T with a root node a. Then it tries to constructs the initial configuration of the Turing machine M on input $w = w_1 \ldots w_n$ by looping through block I for 2^n many times. Each iteration of the block I produces a string of the form $u(\ell_1, b_1) \ldots (\ell_n, b_n)(\ell_{n+1}, z)v$, where $b_1 \ldots b_n$ is the address of a work tape cell in binary, and z is the content of that cell. $\mathcal{B}$ is supposed to construct the initial configuration by specifying the content of the work tape starting with 0 and ending with cell $2^n - 1$.

Each iteration of the loop, closed by the arc $c \to b$, corresponds to a move from one configuration of the Turing machine to the next. First, in block O, process $\mathcal{B}$ tries to reproduce the current configuration (in the first iteration of the loop, it is the initial configuration) by making 2^n iterations. Then, using tree branching in block C_1, the process produces a full binary tree of height n, each branch of which looks like $u(\ell_1, b_1) \dots (\ell_n, b_n)$. Note that the last type (ℓ_n, b_n) is randomising, and after it $\mathcal{B}$ tries to correctly reproduce the content of the cell $b_1 \dots b_n$ in the current configuration. If the current state of M is existential, then $\mathcal{B}$ is expected to move to m_1; if it is universal, it is expected to move to m_2. Two successors of m_1 and m_2 correspond to two possible moves out of the current configuration. In m_1, $\mathcal{B}$ randomly chooses the next move; in m_2, $\mathcal{B}$ makes a tree branching with two children corresponding to two possible next moves. Then, in block N, $\mathcal{B}$ tries to reproduce the next configuration of M in the same way as it produced the current configuration in block O. In block C_2, the process produces a full binary tree of height n (in the same way as it does it in block C_1), and after each branch of the form $u(\ell_1, b_1) \dots (\ell_n, b_n)$ it tries to correctly reproduce the content of the cell $b_1 \dots b_n$ in the next configuration. Then, on every branch $u(\ell_1, b_1) \dots (\ell_n, b_n)(\ell_{n+1}, z)v$ produced by C_2, in blocks D_1, D_2, D_3, the process tries to reproduce the content of the cell $b_1 \dots b_n$ from the old configuration (block O) and its two adjacent cells. Finally, in state c, $\mathcal{B}$ is expected to move from c to the sink state d if the new configuration if accepting. Otherwise, $\mathcal{B}$ is expected to move to b.

We now define an LTL formula φ that describes the expected behaviour of process $\mathcal{B}$. The formula φ is the conjunction of the following parts:

1. In blocks I, O and N, the process constructs a configuration of M cell-by-cell in order starting from cell 0 and ending with cell $2^n - 1$.

2. In block I, the process constructs the initial configuration.

3. On a branch produced by C_1 that corresponds to index k, the cell content specified by C_1 is equal to that of cell k defined in block O, in block I (if this is the first iteration of block C_1) and in the previous iteration of block N (if there was any).

4. If the current configuration if existential, then $\mathcal{B}$ moves to m_1. Otherwise, it moves to m_2.

5. On a branch produced by C_2 that corresponds to index k, the cell content specified by C_2 is equal to that of cell k defined in block N, and the indices in blocks D_1, D_2, D_3 are $k - 1$, k, $k + 1$, respectively.

6. The cell contents in blocks D_1, D_2, D_3 are equal to those defined in block O.

7. The cell content specified by C_2 follows directly from the contents of the cells specified by D_1, D_2, D_3 and the rule of M that was chosen on the current branch.

8. If the new configuration is accepting, then $\mathcal{B}$ moves from c to d. Otherwise, it moves to b.

9. $\mathsf{FG}\, d$, that is, eventually d always holds.

The above properties can be expressed using LTL formulas. We will not explicitly write them down but the details of these formulas are very similar to those defined in the proof of Theorem 3.2.1 from [20].

Now suppose that Turing machine M accepts an input w. Hence there is a 2^n-bounded winning strategy for the existential player. In this case, with some positive probability, $\mathcal{B}$ can generate a tree T all whose branches satisfy φ as follows:

- First, it generates the initial configuration in blocks I and O.
- On every branch produced by C_1, process $\mathcal{B}$ generates the correct cell content.
- Then it moves to either m_1, if the current configuration is existential, or to m_2, if it is universal.
- In the former case, $\mathcal{B}$ chooses the next move that agrees with the winning strategy of the existential player.
- Then in block N, it generates a new configuration of M that follows from O according to the chosen move. (If the tree branching node m_2 was chosen, then $\mathcal{B}$ generates correct new configurations on every branch.)
- Next, $\mathcal{B}$ generates correct cell content on each branch produced by C_2 and chooses correct indices and cell contents in blocks D_1, D_2, D_3.
- Finally, $\mathcal{B}$ moves from c to d, if an accepting configuration is reached, or it moves back to b otherwise. In the latter case, $\mathcal{B}$ generates in block O the same configuration that was generated in N and continues the process.

Note that since the existential player has a winning strategy, state d will eventually appear on every branch of T. Since T is finitely branching, it follows by König's lemma that the above process will reach d on every branch of T in a finite number

of steps. After that, $\mathcal{B}$ repeats the rule $d \overset{1}{\hookrightarrow} d$ forever. Clearly, all these events can happen with some positive probability. Hence, $\mathbb{P}_{\mathcal{B}}(\varphi) > 0$.

Conversely, suppose there is a positive probability that $\mathcal{B}$ generates a tree T all whose branches satisfy φ. Hence every branch of T eventually reaches state d (and then always repeats it). Since T is finitely branching, it follows by König's lemma that there exists a finite prefix of T whose every leaf is labelled by d. This prefix encodes the moves of the existential player (after each appearance of state m_1 in the prefix) that allow him to reach the accepting configuration, no matter what the universal player does. In other words, the existential player has a winning strategy, and hence M accepts w. Formally, this can be shown along similar lines as in the proof of Theorem 6.9.

Therefore, we proved that M accepts w if and only if $\mathbb{P}_{\mathcal{B}}(\varphi) > 0$. □

6.8 Conclusion

We have devised a PSPACE procedure for $\mathbb{P}(\text{LTL}) = 1$, i.e., qualitative LTL model checking of BPs. The best previously known procedure ran in 2EXPTIME [17]. Since BPs naturally generalise both transition systems and Markov chains (for both of which LTL model checking is PSPACE-complete), one might view our model-checking algorithm as an optimal general procedure. The same holds for NBA-specifications instead of LTL.

The main technical ingredients have been the automata-theoretic approach and the algorithmic analysis of UBAs, non-negative matrices, and finiteness of BPs. Our proofs were inspired by the observation that the model checking methods for Markov chains against UBAs from Chapter 3 and Chapter 4 make heavy use of the spectral radius of the transition matrices of the automata, while the spectral radii of adjacency graphs in branching processes also determine fundamental properties of those BPs. Very loosely speaking, when model checking Markov chains against UBAs, the spectral radius measures the amount of non-deterministic branching in the UBA, whereas when analyzing BPs, the spectral radius measures the amount of tree branching. The "general case", i.e., model checking BPs, features both kinds of branching. Serendipitously, an analysis of spectral radii still leads, as we have seen, to optimal algorithms.

We have also established the complexities of related problems, partially as a tool for the mentioned LTL and NBA problems and partially to map out the landscape. We have shown that the $\mathbb{P}(\_\_) = 0$ variants are more complex than their $\mathbb{P}(\_\_) = 1$ counterparts. An intuitive explanation of this phenomenon is

that for an instance of an $\mathbb{P}(\llcorner\!\lrcorner) = 1$ problems to be negative, tree branching and probabilistic branching "work together" to falsify the specification on some branch. In contrast, for $\mathbb{P}(\llcorner\!\lrcorner) = 0$ problems, tree branching and probabilistic branching are "adversaries", like in MDPs. Indeed, for lower bounds on $\mathbb{P}(\llcorner\!\lrcorner) = 0$ problems we have encoded alternation in various forms.

One might ask about the complexity of $\mathbb{P}(\text{UBA}) = 1$. Indeed, in trying to solve $\mathbb{P}(\text{LTL}) = 1$ efficiently, the authors set out to solve $\mathbb{P}(\text{UBA}) = 1$ efficiently (perhaps in P or even NC), with the PSPACE transduction from LTL to UBA in mind. However, the complexity of UBA universality is an open problem [59]; only membership in PSPACE is known. So even for the fixed transition system with $a \overset{1}{\hookrightarrow} ab$ and $b \overset{1}{\hookrightarrow} ab$ the problem $\mathbb{P}(\text{UBA}) = 1$ cannot be placed in P without improving the complexity of UBA universality. A PSPACE-hardness proof of $\mathbb{P}(\text{UBA}) = 1$ might have to make use of both types of branching in BPs, as $\mathbb{P}(\text{UBA}) = 1$ is in NC for Markov chains, as we've seen in Chapter 3.

Model checking BPs quantitatively, i.e., computing the satisfaction probability, comparing it with a threshold, or approximating it, is left for future work. Exact versions of these problems are computationally complex, as they are at least as hard as the corresponding $\mathbb{P}(\llcorner\!\lrcorner) = 0$ problem. The paper [17] describes, for DPAs, non-linear equation systems whose least non-negative solution characterises the satisfaction probabilities. Newton's method is efficient for approximating the solution of such equation systems; see [25; 66].

7 Conclusion

In this thesis we considered model checking of both Markov chains and branching processes against unambiguous automata, as well as a type of weighted automata we call image-binary automata. In particular, we exploited the spectral properties of unambiguous automata to construct efficient, linear algebra based model checking algorithms. One central idea that these algorithms are based on is the observation that we can use the spectral radius of a product of the automaton and the model in order to check whether the probability of the model generating a word accepted by the automaton is positive. In the case of qualitative model checking in Chapter 6, we can use this fact to reduce the model checking problem of branching processes against UBAs to a question about the degree of branching in the branching process. In order to do quantitative model checking, we need to compute vectors that are in a certain sense invariant to actions of the automata, such vectors are called normalisers. Chapter 4 concerns an efficient way of calculating these normalisers when model checking unambiguous Büchi automata against Markov chains, and Chapter 5 describes a more general model of automata which allows for a similar NC model checking procedure.

In Chapter 4 we describe an improvement on the model checking procedure of UBAs against Markov chains by Baier et al.[7] as described in Chapter 3. The main idea of Baier et al.'s model checking procedure is to describe the model checking question as a system of linear equations, allowing us to solve it using linear algebra. However, the complexity of setting up this system of linear equations is bounded by the complexity of calculating normalisers, which they do in a combinatoric procedure to compute so-called cuts. In Chapter 4 we replace these cuts with a linear algebra

based procedure to calculate different normalisers called $Co(d)$-pseudo-cuts. By doing so, we managed to improve on the algorithmic complexity of the model checking question by a linear factor in the size of the automaton, at the cost of a linear factor in the size of the Markov chain. The computation of co-reachability sets in this algorithm is the current bottleneck in terms of computational complexity, and one might ask if this bottleneck can be avoided.

Our efforts to generalise this linear algebra based procedure to finitely ambiguous automata led to the discovery of a new type of automata called image-binary automata in Chapter 5. For image-binary automata over finite words, we show that these automata have several desirable properties. We show that they accept the regular languages and that one can in polynomial time determine whether a given $\mathbb{Q}$-weighted automaton is image-binary. We also show that they are efficiently (linearly) closed under all boolean operations, which shows an advantage of image-binary automata over unambiguous automata, and that they can be exponentially more succinct than DFAs. We also compare IFAs against mod-2-multiplicity automata, and show that while for every IFA with n states there exists an equivalent mod-2-MA with n states, converting a mod-2-MA to an IFA may require an exponential blowup in the number of states. Lastly we consider image-binary Büchi automata. We give an optimal procedure for converting finitely ambiguous automata into IBAs using a PSPACE transducer, and we show that the model checking procedure for unambiguous automata works even in the image-binary setting. By combining these we obtain an optimal PSPACE model checking procedure of finitely ambiguous automata against Markov chains, showing optimality of both the translation from finitely ambiguous automata to IBAs and the model checking procedure for IBAs. Many standard questions for automata remain open for image-binary automata, and one may consider natural generalisations of image-binary automata such as image-finite automata.

In Chapter 6 we consider qualitative model checking of LTL formulas against branching processes using an automata-theoretic approach. Branching processes are a model for generating trees, thereby exhibiting both stochastic and non-deterministic branching. We show that the question whether a BP almost surely generates a finite tree, is in NC, and we consider the questions whether it's almost surely the case and whether it's almost never the case that a BP generates a tree where all branches are accepted by a specification given by deterministic parity automata, NBAs, coNBAs (where a branch is rejected if and only if the Büchi acceptance condition is satisfied), an coUBAs (see coNBAs). We see that the question whether almost surely all branches are accepted by NBAs or coNBAs is

PSPACE-complete. However, we use the spectral properties of UBAs to show that the question whether almost surely all branches are accepted by a coUBA is in NC. We also show that the question whether the BP almost never generates a tree where all the branches are accepted by various kinds of automata is much harder than the almost surely variants. This is because for the BP to almost surely generate an accepted tree, the stochastic branching and non-deterministic branching "work together" to disprove it, whereas for it to almost never generate an accepted tree, the two types of branching can be considered adversaries. Questions remain open on quantitative model checking of branching processes.

Throughout this thesis we see that we can use spectral properties of finitely ambiguous automata in order to do model checking in probabilistic settings. However, open questions remain, especially in non-stochastic settings. An important open question is the complexity of the universality question for unambiguous Büchi automata:

Problem 7.1. *Given an unambiguous Büchi automaton $\mathcal{A}$ on an alphabet Σ, does $\mathcal{A}$ accept all words in Σ^ω?*

A common way of checking universality of automata is by complementing the automaton and checking the resulting automaton for emptiness. However, we know that complementation of unambiguous (finite-word or Büchi) automata may require a superpolynomial blowup. Since the only known upper bounds for the complementation of unambiguous automata are exponential, there is still a gap to be closed between the lower and upper bound. While we have shown hardness results for qualitative model checking of several types of automata as well as LTL against branching processes, one might ask whether these problems are still hard if the branching process is considered to be fixed. Since the universality problem can be viewed as a qualitative model checking problem of automata against the fixed branching problem that has a single derivation rule from every type to all types, we know that this problem is at least as hard as the universality problem, which is hard for NBAs and unknown for UBAs. However, the exact complexity is unknown. Another open question to be solved is the quantitative model checking question of branching processes. Since we already know that the question whether a BP almost never generates a tree where all branches satisfy an LTL formula is 2EXPTIME-complete, we cannot expect to find any algorithm solving the quantitative model checking question faster than that. However, to the best of our knowledge, no procedure is known for quantitative model checking of branching processes. Finally there is the question of the parameterised complexity of model checking k-ambiguous

automata against Markov chains. From a reduction from the finite intersection problem for DFAs we know that this question is PSPACE-hard even when k is as low as the number of states in the automaton. However, for low or constant k the complexity is not known, and since a naive disambiguation method would have to complement the automaton, this problem may be hard even for constant k.

Bibliography

[1] Almagor, Shaull ; Boker, Udi ; Kupferman, Orna: What's decidable about weighted automata? In: *Information and Computation* (2020)

[2] Alman, Josh ; Williams, Virginia V.: A refined laser method and faster matrix multiplication. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)* SIAM (Veranst.), 2021, S. 522–539

[3] Angluin, Dana ; Antonopoulos, Timos ; Fisman, Dana: Strongly Unambiguous Büchi Automata Are Polynomially Predictable With Membership Queries. In: *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, 2020

[4] Athreya, K.B. ; Ney, P.E.: *Branching Processes*. Springer, 1972

[5] Baier, Christel ; Katoen, Joost-Pieter: *Principles of model checking*. MIT press, 2008

[6] Baier, Christel ; Kiefer, Stefan ; Klein, Joachim ; Klüppelholz, Sascha ; Müller, David ; Worrell, James: Markov Chains and Unambiguous Büchi Automata. In: *Proceedings of Computer Aided Verification (CAV)* Bd. 9779, 2016, S. 23–42

[7] Baier, Christel ; Kiefer, Stefan ; Klein, Joachim ; Klüppelholz, Sascha ; Müller, David ; Worrell, James: *Markov Chains and Unambiguous Büchi Automata (extended version)*. arXiv:1605.00950. 2016. – `http://arxiv.org/abs/1605.00950`

[8] Bengtsson, Johan ; Larsen, Kim ; Larsson, Fredrik ; Pettersson, Paul ; Yi, Wang: UPPAAL—a tool suite for automatic verification of real-time systems. In: *International hybrid systems workshop* Springer (Veranst.), 1995, S. 232–243

[9] Berman, Abraham ; Plemmons, Robert J.: *Nonnegative matrices in the mathematical sciences*. SIAM, 1994

[10] Berstel, J. ; Reutenauer, C.: *Rational Series and Their Languages*. Springer, 1988

[11] Borodin, A. ; Gathen, J. von zur ; Hopcroft, J.E.: Fast Parallel Matrix and GCD Computations. In: *Information and Control* 52 (1982), Nr. 3, S. 241–256. – URL https://doi.org/10.1016/S0019-9958(82)90766-5

[12] Borodin, Allan: On Relating Time and Space to Size and Depth. In: *SIAM J. Comput.* 6 (1977), Nr. 4, S. 733–744

[13] Bunch, James R. ; Hopcroft, John E.: Triangular factorization and inversion by fast matrix multiplication. In: *Mathematics of Computation* 28 (1974), S. 231–236

[14] Bustan, Doron ; Rubin, Sasha ; Vardi, Moshe Y.: Verifying ω-Regular Properties of Markov Chains. In: *16th International Conference on Computer Aided Verification (CAV)* Bd. 3114, Springer, 2004, S. 189–201

[15] Carlyle, J. W. ; Paz, A.: Realizations by stochastic finite automata. In: *Journal of Computer and System Sciences* 5 (1971), Nr. 1, S. 26–40

[16] Chakraborty, Souymodip ; Katoen, Joost-Pieter: Parametric LTL on Markov Chains. In: *8th IFIP TC 1/WG 2.2 International Conference on Theoretical Computer Science* Bd. 8705, Springer, 2014, S. 207–221

[17] Chen, T. ; Dräger, K. ; Kiefer, S.: Model Checking Stochastic Branching Processes. In: *Mathematical Foundations of Computer Science (MFCS)* Bd. 7464, Springer, 2012, S. 271–282

[18] Colcombet, Thomas: Unambiguity in Automata Theory. In: *17th International Workshop on Descriptional Complexity of Formal Systems (DCFS)* Bd. 9118, Springer, 2015, S. 3–18

[19] Courcoubetis, C. ; Yannakakis, M.: Verifying temporal properties of finite-state probabilistic programs. In: *Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, 1988, S. 338–345

[20] Courcoubetis, Costas ; Yannakakis, Mihalis: The Complexity of Probabilistic Verification. In: *Journal of the ACM* 42 (1995), Nr. 4, S. 857–907

[21] Couvreur, Jean-Michel ; Saheb, Nasser ; Sutre, Grégoire: An optimal automata approach to LTL model checking of probabilistic systems. In: *10th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR)* Bd. 2850, Springer, 2003, S. 361–375

[22] Droste, Manfred ; Kuich, Werner ; Vogler, Heiko: *Handbook of Weighted Automata*. 1st. Springer Publishing Company, Incorporated, 2009. – ISBN 3642014917

[23] Droste, Manfred ; Meinecke, Ingmar: Weighted automata and regular expressions over valuation monoids. In: *International Journal of Foundations of Computer Science* (2011)

[24] Esparza, J. ; Gaiser, A. ; Kiefer, S.: A strongly polynomial algorithm for criticality of branching processes and consistency of stochastic context-free grammars. In: *Information Processing Letters* 113 (2013), Nr. 10-11, S. 381–385

[25] Etessami, K. ; Stewart, A. ; Yannakakis, M.: A Polynomial Time Algorithm for Computing Extinction Probabilities of Multitype Branching Processes. In: *SIAM Journal of Computing* 46 (2017), Nr. 5, S. 1515–1553

[26] Etessami, K. ; Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. In: *Journal of the ACM* 56 (2009), Nr. 1, S. 1:1–1:66

[27] Etessami, K. ; Yannakakis, M.: Model Checking of Recursive Probabilistic Systems. In: *ACM Transactions on Computational Logic* 13 (2012), Nr. 2, S. 12:1–12:40. – URL `https://doi.org/10.1145/2159531.2159534`

[28] Fijalkow, Nathanaël: Undecidability results for probabilistic automata. In: *ACM SIGLOG News* 4 (2017), Nr. 4, S. 10–17

[29] Filiot, Emmanuel ; Gentilini, Raffaella ; Raskin, Jean-Francois: Finite-Valued Weighted Automata. In: *34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, 2014

[30] Fliess, M.: Matrices de Hankel. In: *Journal de Mathématiques Pures et Appliquées* 53 (1974), S. 197–222

[31] Golomb, Solomon W. ; Goldstein, Robert M. ; Hales, A. W. ; Welch, L. R.: *Shift register sequences*. San Francisco : Holden-Day, 1967 (Holden-Day series in information systems)

[32] Grädel, Erich (Hrsg.) ; Thomas, Wolfgang (Hrsg.) ; Wilke, Thomas (Hrsg.): *Automata, Logics, and Infinite Games: A Guide to Current Research*. Bd. *2500*. Springer, 2002. (Lecture Notes in Computer Science)

[33] Haccou, P. ; Jagers, P. ; Vatutin, V.A.: *Branching Processes: Variation, Growth, and Extinction of Populations*. Cambridge University Press, 2005

[34] Harris, T.E.: *The Theory of Branching Processes*. Springer, 1963

[35] Higham, Nicholas J.: *Accuracy and Stability of Numerical Algorithms*. Second. SIAM, 2002

[36] Holzer, M. ; Kutrib, M.: Nondeterministic Descriptional Complexity Of Regular Languages. In: *International Journal of Foundations of Computer Science* (2003)

[37] Holzmann, Gerard J.: The model checker SPIN. In: *IEEE Transactions on software engineering* 23 (1997), Nr. 5, S. 279–295

[38] Indzhev, Emil ; Kiefer, Stefan: On Complementing Unambiguous Automata and Graphs With Many Cliques and Cocliques. In: *arXiv preprint arXiv:2105.07470* (2021)

[39] James, M.: The generalised inverse. In: *The Mathematical Gazette* 62 (1978), S. 109–114

[40] Jirásek, J. ; Jirásková, G. ; Sebej, J.: Operations on Unambiguous Finite Automata. In: *International Journal of Foundations of Computer Science* 29 (2018), Nr. 5, S. 861–876

[41] Karmarkar, Hrishikesh ; Joglekar, Manas ; Chakraborty, Supratik: Improved Upper and Lower Bounds for Büchi Disambiguation. In: *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Proceedings*, 2013

[42] Kiefer, Stefan ; Semukhin, Pavel ; Widdershoven, Cas: Linear-Time Model Checking Branching Processes. In: *32nd International Conference on Concurrency Theory*, 2021

[43] Kiefer, Stefan ; Widdershoven, Cas: Efficient Analysis of Unambiguous Automata Using Matrix Semigroup Techniques. In: *44th International Symposium on Mathematical Foundations of Computer Science*, 2019, S. 82:1–82:13

[44] Kiefer, Stefan ; Widdershoven, Cas: Image-Binary Automata. In: *International Conference on Descriptional Complexity of Formal Systems* Springer (Veranst.), 2021, S. 176–187

[45] Kozen, Dexter: Lower bounds for natural proof systems. In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)* IEEE (Veranst.), 1977, S. 254–266

[46] Kulkarni, Vidyadhar G.: *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, 1995

[47] Kwiatkowska, Marta Z. ; Norman, Gethin ; Parker, David: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: *23rd International Conference on Computer Aided Verification (CAV)* Bd. 6806, Springer, 2011, S. 585–591

[48] Le Gall, François: Powers of Tensors and Fast Matrix Multiplication. In: *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ACM, 2014 (ISSAC'14), S. 296–303

[49] Leung, H.: Separating Exponentially Ambiguous Finite Automata from Polynomially Ambiguous Finite Automata. In: *SIAM Journal of Computing* 27 (1998), Nr. 4, S. 1073–1082

[50] Leung, H.: Descriptional complexity of NFA of different ambiguity. In: *International Journal of Foundations of Computer Science* 16 (2005), Nr. 5, S. 975–984

[51] Löding, Christof ; Pirogov, Anton: On finitely ambiguous Büchi automata. In: *International Conference on Developments in Language Theory* Springer (Veranst.), 2018, S. 503–515

[52] Mazzanti, Franco ; Ferrari, Alessio: Ten Diverse Formal Models for a CBTC Automatic Train Supervision System. In: Gallagher, John P. (Hrsg.) ; Glabbeek, Rob van (Hrsg.) ; Serwe, Wendelin (Hrsg.): *Proceedings Third Workshop on Models for Formal Analysis of Real Systems and*

Sixth International Workshop on Verification and Program Transformation, MARS/VPT@ETAPS 2018, Thessaloniki, Greece, 20th April 2018 Bd. 268, URL `https://doi.org/10.4204/EPTCS.268.4`, 2018, S. 104–149

[53] Papadimitriou, Christos M.: *Computational complexity.* Reading, Massachusetts : Addison-Wesley, 1994. – ISBN 0201530821

[54] Paz, Azaria: *Introduction to probabilistic automata.* Academic Press, 2014

[55] Petković, Marko D. ; Stanimirović, Predrag S.: Generalised matrix inversion is not harder than matrix multiplication. In: *Journal of Computational and Applied Mathematics* 230 (2009), S. 270–282

[56] Piterman, N.: From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. In: *Logical Methods Computer Science* 3 (2007), Nr. 3. – URL `https://doi.org/10.2168/LMCS-3(3:5)2007`

[57] Protasov, V.Yu. ; Voynov, A.S.: Matrix semigroups with constant spectral radius. In: *Linear Algebra and its Applications* 513 (2017), S. 376–408

[58] Rabin, Michael O.: Probabilistic automata. In: *Information and control* 6 (1963), Nr. 3, S. 230–245

[59] Rabinovich, A.: Complementation of Finitely Ambiguous Büchi Automata. In: *Developments in Language Theory (DLT)* Bd. 11088, Springer, 2018, S. 541–552

[60] Raskin, M.: A Superpolynomial Lower Bound for the Size of Non-Deterministic Complement of an Unambiguous Automaton. In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, 2018

[61] Safra, Shmuel: On the complexity of omega-automata. In: *Proc. 29th IEEE Symp. Found. of Comp. Sci*, 1988, S. 319–327

[62] Schmidt, E.M.: *Succinctness of descriptions of context-free, regular and finite languages.* Ithaca, NY, Cornell University, Dissertation, 1978

[63] Schützenberger, M.P.: On the definition of a family of automata. In: *Information and Control* 4 (1961), Nr. 2, S. 245–270

[64] Sistla, A. P. ; Vardi, Moshe Y. ; Wolper, Pierre: The complementation problem for Büchi automata with applications to temporal logic. In: *International Colloquium on Automata, Languages, and Programming* Springer (Veranst.), 1985, S. 465–474

[65] Sistla, A.P ; Clarke, E.M.: The Complexity of Propositional Linear Temporal Logics. In: *Journal of the ACM* 32 (1985), Nr. 3, S. 733–749. – URL `https://doi.org/10.1145/3828.3837`

[66] Stewart, A. ; Etessami, K. ; Yannakakis, M.: Upper Bounds for Newton's Method on Monotone Polynomial Systems, and P-Time Model Checking of Probabilistic One-Counter Automata. In: *Journal of the ACM* 62 (2015), Nr. 4, S. 30:1–30:33

[67] Tzeng, W.-G.: On Path Equivalence of Nondeterministic Finite Automata. In: *Information Processing Letters* 58 (1996), Nr. 1, S. 43–46

[68] Tzeng, Wen-Guey: A polynomial-time algorithm for the equivalence of probabilistic automata. In: *SIAM Journal on Computing* 21 (1992), Nr. 2, S. 216–227

[69] Vardi, Moshe Y.: Automatic verification of probabilistic concurrent finite-state programs. In: *26th IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE Computer Society, 1985, S. 327–338

[70] Vardi, Moshe Y. ; Wolper, Pierre: An automata-theoretic approach to automatic program verification (preliminary report). In: *1st Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society Press, 1986, S. 332–344

[71] Weber, Andreas ; Seidl, Helmut: On the degree of ambiguity of finite automata. In: *Theoretical Computer Science* 88 (1991), Nr. 2, S. 325–349