RICE UNIVERSITY

# Büchi Automata as Specifications for Reactive Systems

by

**Seth Fogarty**

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

**Doctor of Philosophy**

APPROVED, THESIS COMMITTEE:

Moshe Vardi

Moshe Y. Vardi, Chair
Karen Ostrum George Professor
in Computational Engineering

[illegible]

Keith Cooper
L. John and Ann H. Doerr Professor of
Computational Engineering

[illegible]

Luay Nakhleh
Associate Professor of Computer Science
and Biochemistry and Cell Biology

[illegible]

Ray Simar
Professor in the Practice of Digital Signal
Processing Architecture
Departments of Electrical and Computer
Engineering and Computer Science.

Houston, Texas

June, 2012

Dedicated to the memory of Andrew Ladd.

ABSTRACT

# Büchi Automata as Specifications for Reactive Systems

by

Seth Fogarty

Computation is employed to incredible success in a massive variety of applications, and yet it is difficult to formally state what our computations are. Finding a way to model computations is not only valuable to understanding them, but central to automatic manipulations and formal verification. Often the most interesting computations are not functions with inputs and outputs, but ongoing systems that continuously react to user input. In the automata-theoretic approach, computations are modeled as words, a sequence of letters representing a trace of a computation. Each automaton accepts a set of words, called its language. To model reactive computation, we use Büchi automata: automata that operate over infinite words. Although the computations we are modeling are not infinite, they are unbounded, and we are interested in their ongoing properties. For thirty years, Büchi automata have been recognized as the right model for reactive computations.

In order to formally verify computations, however, we must also be able to create specifications that embody the properties we want to prove these systems possess. To date, challenging algorithmic problems have prevented Büchi automata from being used as specifications. I address two challenges to the use of Büchi automata as specifications in formal verification. The first, complementation, is required to check program adherence to a specification. The second, determination, is used in domains such as synthesis, probabilistic verification, and module checking. I present both empirical analysis of existing complementation constructions, and a new theoretical contribution that provides more deterministic complementation and a full determination construction.

# Acknowledgments

This work would not have been possible without my advisor, Professor Moshe Y. Vardi. Beyond the significant academic guidance Dr. Vardi has provided, he has helped me to find the enthusiasm, focus, and joy that have enabled this dissertation. I would like to thank my thesis committee, Dr. Keith Cooper, Dr. Luay K. Nakhleh, and Prof. Ray Simar, for support and feedback. I offer thanks to the past and present members of the Formal Verification group at Rice University, to Prof. Erickson at UIUC for inspiring my interest in computer science theory, and to my parents, Gail and Thomas.

# Contents

Abstract v

Acknowledgments vii

**1 Introduction 1**

1.1 Formal Verification for Reactive Systems . . . . . . . . . . . . . . . . . . 1

1.2 Büchi Specifications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.2.1 Complementation . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2.2 Determinization . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

1.4 Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

**2 Preliminaries 11**

2.1 Büchi Automata . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.2 Complementation Constructions . . . . . . . . . . . . . . . . . . . . . . 16

2.2.1 Rank-Based Complementation . . . . . . . . . . . . . . . . . . 16

2.2.2 Ramsey-Based Complementation . . . . . . . . . . . . . . . . 20

**3 Complementation Experiments 25**

3.1 Preliminaries: Tabakov-Vardi Random Automata . . . . . . . . . . . . 25

3.2 Empirical Comparisons: Size of Complemented Automata . . . . . . . 27

**4 Rank and Ramsey-Based Containment 31**

4.1 Preliminaries: Rank-Based Containment Checking . . . . . . . . . . . 31

4.2 Ramsey-Based Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . 33

4.2.1 Ramsey-Based Containment Checking . . . . . . . . . . . . . . 34

4.2.2 Composition of Supergraphs . . . . . . . . . . . . . . . . . . . . 39

4.2.3 Subsumption in Ramsey-Based Containment Checking . . . . . 44

4.3 Empirical Comparisons: Universality Experiments . . . . . . . . . . . . 54

4.3.1 Subsumption . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

4.3.2 Terrain Experiments . . . . . . . . . . . . . . . . . . . . . . . . 56

4.3.3 Scaling Experiments . . . . . . . . . . . . . . . . . . . . . . . . 57

4.3.4 Universal vs. Non-Universal Automata . . . . . . . . . . . . . . 61

4.3.5 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63

**5 Profile-Based Complementation 65**

5.1 Preliminaries: The Slice-Based Approach . . . . . . . . . . . . . . . . . 66

5.2 Analyzing DAGs With Profiles . . . . . . . . . . . . . . . . . . . . . . . 68

5.2.1 Profiles . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68

5.2.2 Complementing With Profiles . . . . . . . . . . . . . . . . . . . 75

5.3 Retrospection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79

5.3.1 Retrospective Labeling . . . . . . . . . . . . . . . . . . . . . . . 80

5.3.2 From Labelings to Rankings . . . . . . . . . . . . . . . . . . . . 82

5.4 Variations on the Retrospective Construction . . . . . . . . . . . . . . . 93

5.5 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97

**6 Profile-Based Determinization 101**

6.1 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102

6.1.1 Relations on Sets . . . . . . . . . . . . . . . . . . . . . . . . . . 102

6.1.2 Non-Büchi $\omega$-Automata . . . . . . . . . . . . . . . . . . . . . . . 103

6.1.3 Safra's Determinization Construction . . . . . . . . . . . . . . 103

6.2 Equivalence Classes Under Profiles . . . . . . . . . . . . . . . . . . . . . . . 106

6.2.1 The Tree of Equivalence Classes . . . . . . . . . . . . . . . . . . . 107

6.2.2 Labeling $T^{G'}$ . . . . . . . . . . . . . . . . . . . . . . . . . . 108

6.3 The Determinization Construction . . . . . . . . . . . . . . . . . . . . . . . 118

6.3.1 Correctness and Blow-up . . . . . . . . . . . . . . . . . . . . . . . . 120

6.3.2 Compact Determinization Constructions . . . . . . . . . . . . . 125

**7 Conclusion and Future Work 131**

**Bibliography 137**

**A Symbolic Subsumption 147**

A.1 Symbolic Subsumption for NFAs . . . . . . . . . . . . . . . . . . . . . . . 148

A.2 Implementation and Experimentation . . . . . . . . . . . . . . . . . . . . 152

**B Experimental Documentation 157**

B.1 Framework for Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . 157

B.2 Complementation of Automata . . . . . . . . . . . . . . . . . . . . . . . . 158

B.3 Büchi Universality and Containment Checking . . . . . . . . . . . . . 159

B.4 NFA Universality . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 159

B.5 File Formats . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 160

B.5.1 aut . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 160

B.5.2 svw . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 161

B.5.3 gff . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 162

B.5.4 NuSMV . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 164

# Chapter 1

# Introduction

## 1.1 Formal Verification for Reactive Systems

Computational systems, both hardware and software, are increasingly pervasive in society. The most interesting systems are often not functions with inputs and outputs, but ongoing systems that continuously react to user input. Because of the increasing complexity of these systems, we often want to verify their correctness. The traditional method of ensuring that a system behaves as expected is testing: manually or automatically run the system over a variety of test cases, and look for errors on those particular cases. For some systems, testing does not provide sufficient assurance that the system is correct. For instance, hardware systems are expensive to replace if a bug is found after release, embedded systems can control vehicles, medical devices, and other correctness-critical devices, and high-assurance software can be responsible for critical business operations [Har03,RvH93]. In these situations, we can seek a formal method for verifying that a system adheres to a specification. A proof of such adherence provides increased assurance that the system is correct.

There are three steps to formally verifying a computational system. First, we must abstract the system into a model, as proving any interesting property of on the system directly is undecidable. Second, we must construct a specification that embodies the properties we wish to prove the system possesses. Finally, we must prove the model adheres to the specification [VW86]. In the automata-theoretic approach, computations are modeled as *words*, a sequence of letters representing a trace of a computation. A system is thus a set of words, called an *language*. This approach then reduces checking adherence to checking language containment. To model reactive computation, we use infinite words. Although the computations we are modeling are not infinite, they are unbounded, and we are interested in their ongoing properties, properties like liveness, fairness, or termination.

To represent a set of words, we use *automata*: mathematical machines with a finite number of states. Each word defines a set of possible paths through the automaton, called the *runs* of the automaton on the word. Depending on the runs, the automaton will accept or reject the word. Thus each automaton accepts a set of words, called its *language*. Automata that operate over infinite words are classified by their acceptance condition and transition structure. This work primarily considers *nondeterministic Büchi automata on infinite words* (NBW). In an NBW, some of the states are designated as accepting, a run is accepting if it visits accepting states infinitely often, and a word is accepted if there exists a possible run that is accepting [Büc62]. For thirty years, Büchi automata have been recognized as the right model for reactive systems [Kur94].

## 1.2 Büchi Specifications

This dissertation addresses theoretical and practical limitations to the use of Büchi automata as *specifications* for reactive systems. Given a Büchi automaton as a specification, we address the manipulations and algorithms necessary to perform verification. When manipulating automata specifications, there are two canonical operations that have emerged over the last thirty years. *Complementation* is used to check that a program adheres to a specification, while *determinization* is employed in domains such as synthesis, probabilistic verification, and module checking. For automata on finite words, both complementation and determinization are done via the subset construction [RS59]. For Büchi automata the subset construction is not sufficient, and complementation and determinization constructions are more complicated [Var07a].

### 1.2.1 Complementation

When employing the automata-theoretic approach to formal verification, we construct one automaton $\mathcal{A}$ that models to the system, and another automaton $\mathcal{B}$ that corresponds to a property [VW86]. This reduces the question of the model adhering to the specification to the question of the language of $\mathcal{A}$ being contained in the language of $\mathcal{B}$. The standard approach to checking language containment employs complementation. First, obtain an automaton that accepts all and only the words that $\mathcal{B}$ rejects. We call this automaton the *complement* of $\mathcal{B}$, written $\overline{\mathcal{B}}$. The language of $\overline{\mathcal{B}}$ contains all possible counterexamples to adherence. To see if any of these counterexamples are valid, we compute the intersection of $\mathcal{A}$ with $\overline{\mathcal{B}}$, and check to see if there exists a word accepted by the intersection. If the language of the intersection is empty, then there is no counterexample and the model adheres to the specification. If the language of the intersection is not empty, then we have a counterexample which can be used to examine the system for bugs or to revise the model for increased fidelity to the system.

For a nondeterministic automaton $\mathcal{B}$, a word $w$ is rejected by $\mathcal{B}$ if *all* runs of $\mathcal{B}$ on $w$ reject the word. Thus, the complementary automaton has to consider all possible runs. Efforts to develop simple complementation constructions for Büchi automata started early in the 60's, motivated by decision problems of second-order logics. Büchi suggested a complementation construction for nondeterministic Büchi automata that involved a Ramsey-based combinatorial argument and a doubly-exponential blow-up in the state space [Büc62]. Thus, complementing an automaton with $n$ states resulted in an automaton with $2^{2^{O(n)}}$ states. In [SVW87], Sistla et al. suggested an improved implementation of Büchi's construction with only $2^{O(n^2)}$ states, which is still not optimal. Only in [Saf88] Safra introduced a determinization construction, based on *Safra trees*, which also enabled a $2^{O(n \log n)}$ complementation construction, matching a lower bound described by Michel [Mic88]. A careful analysis of the exact blow-up in Safra's and Michel's bounds, however, reveals an exponential gap in the constants hiding in the $O()$ notations: while the upper bound on the number of states in the complementary automaton constructed by Safra is $n^{2n}$, Michel's lower bound involves only an $n!$ blow up: roughly $(n/e)^n$. In addition, Safra's construction has been resistant to optimal implementations [ATW06, THB95], due to do with the complicated combinatorial structure of its states and transitions. In 2001, Kupferman and Vardi suggested a new complementation approach, called the *rank-based approach*, that circumvented Safra's determinization construction and the complicated data structure of Safra trees [KV01]. The resulting *rank-based construction* gave rise to a sequence of works improving the complexity [FKV06, Sch09a]. In an effort to unify Büchi complementation with other operations on automata, Kähler and Wilke introduced yet approach to complementing nondeterministic Büchi automata, called the *slice-based approach* [KW08].

In 2007, Tabakov and Vardi began investigating rank-based algorithms that checked language containment directly, without explicitly constructing the complemented automaton [TV07]. These algorithm explore the state space of the automaton on the fly, employing heuristic approaches to improve performance. Doyen and Raskin later introduced a useful subsumption technique for the rank-based approach [DR09], providing containment checking algorithms that scaled to problems an order of magnitude larger than previous approaches.

Despite the numerous approaches to complementation proposed over the years, practical application has remained elusive. No model checkers perform complementation for nondeterministic Büchi automata. The SPIN model checker accepts Büchi specifications, but it requires they be complemented ahead of time, given as a 'Never-Claims' automaton [Hol97]. By contrast, COSPAN accepts Büchi specifications only if they are deterministic [HHK96, Kur94], which reduces the complexity of complementation to polynomial [Kur87]. Some of the difficulty emerges from the nondeterminism inherent to complementation: complementation constructions must produce nondeterministic automata, which have been shown to perform poorly in practice [ST03].

### 1.2.2 Determinization

*Determinization* of nondeterministic automata is a fundamental problem in automata theory, going back to [RS59]. A deterministic automaton has only one run on a given word, greatly simplifying manipulations of the automaton. To determinize an automaton $\mathcal{B}$, we construct an automaton $\mathcal{B}'$ that accepts the same language as $\mathcal{B}$ but is deterministic. Determinization of NBW is employed in many applications, including decision procedures for branching-time logics [EJ88], synthesis of reactive systems [PR89], verification of probabilistic systems [CY95, Var85], and module checking [KVW01]. Determinization of NBW is complicated by the fact that they are not closed under determinization; deterministic Büchi automata are strictly less expressive than their nondeterministic counterparts [Lan69]. Thus, a determinization construction for Büchi automata must result in automata with a more powerful acceptance condition, such as Muller [McN66], Rabin [Saf88], or parity conditions [KC11, KW08, Pit06].

The first determinization construction for Büchi automata was presented by McNaughton, with a doubly-exponential blowup [McN66]. In 1988, Safra introduced a singly exponential construction [Saf88]. Safra's construction encodes a state of the deterministic automaton as a labeled tree, now called a *Safra tree*, of sets of states of the input Büchi automaton, generalizing the subset construction of [RS59]. In 2006, Piterman improved Safra trees by simplifying the use of labels [Pit06]. In 2009, Schewe offered a further improvement by moving the acceptance conditions from states to edges [Sch09b] (see also [LW09]). Karmarkar and Chakraborty, building on [Saf92], supply another take on Safra trees, providing a uniform way of determinizing automata with different acceptance conditions [KC11]. In a separate line of work, Muller and Schupp proposed a different determinization construction, based on *Muller–Schupp trees* [MS95]. In 2008, Kähler and Wilke proposed a simplification of the Muller–Schupp approach [KW08].

Unfortunately, both Safra trees and Muller–Schupp trees are combinatorially complex, and transitions between trees are defined as a sequence of operations on these trees. While Piterman and Schewe's improvements to Safra trees, and Kähler and Wilke's improvements to Muller–Schupp trees help separate the acceptance condition from the "mechanics" of the trees, they still rely on these trees as the underlying state structure. Because of this, determinization constructions have proven challenging to implement efficiently [ATW05, THB95]. In fact, the difficulty of determinization has motivated attempts to find determinization-free decision procedures [KV05]. The fundamental problem is that known determinization constructions lack crisp mathematical descriptions.

## 1.3 Results

This work attacks the problem of using Büchi automata as specifications on two fronts. One direction to develop concrete containment checking algorithms and empirically test their real-world performance. We also develop a new theoretical analysis of Büchi automata that results in new complementation and determinization constructions.

The empirical analysis focuses on two approaches to complementation: the $2^{O(n \log n)}$ rank-based approach, and the $2^{O(n^2)}$ Ramsey-based approach. Due to the massive gap in worst-case complexity, the Ramsey-based approach was discounted in the 1980's and neither studied much nor implemented. We perform an empirical comparison of the two approaches by examining the size of complemented automata. Due to a paucity of real-world containment problems, we compare the algorithms over a terrain of random automata [TV07]. The results hint that despite the exponential gap, the Ramsey-based approach may be competitive with the rank-based approach. Far more significantly, however, the results demonstrate that direct complementation using *any* approach is not feasible: we can only scale to automata with at most 10 states.

We therefore expand the Ramsey-based approach to a new on-the-fly containment checking algorithms. To improve the performance, we develop a subsumption technique modeled off [BAL07]. By doing so we provide a direct algorithm, derived from the Ramsey-based complementation construction, for checking the containment of Büchi automata. We note that subsumption is a heuristic technique and, even with this improvement, there is still an exponential gap between the $2^{O(n^2)}$ Ramsey-based approach and the $2^{O(n \log n)}$ rank-based approach. Therefore, we investigate the empirical performance of the Ramsey-based algorithm and the rank-based containment algorithm of Doyen and Raskin [DR07]. Our empirical results indicate that containment checking algorithms can scale to automata orders of magnitude larger than direct complementation. Further, the two algorithms exhibit significantly different behavior, with each scaling better on different configurations of problems. These investigations have thus provided a case study arguing that, while worst-case complexity is a useful metric, it is a poor predictor of performance and no substitute for empirical analysis.

One drawback of the rank-based approach is that it is inherently very nondeterministic. Restricting nondeterminism is necessary to use Büchi automata in several areas [CY95], and even when nondeterminism is allowed it can performs poorly in practice [ST03]. In the rank-based approach, the ranks bound the visits to accepting states yet to come. Thus, the ranks refer to the *future* of the run, making the construction inherently nondeterministic. In contrast, in the slice-based approach of Kähler and Wilke, states are partitioned based on previous visits to accepting states. Thus, the partition refers to the *past* of the run, not the future, and the slice-based approach has the potential to be less nondeterministic. We realize this potential. To do so, we introduce a new analysis based on a notion of *profiles* that unifies the rank and slice-based approach. In addition to revealing the theoretical connections between the two seemingly different approaches, profiles lead to a complementation construction with a transition function that is smaller and deterministic in the limit: every accepting run of the automaton is eventually deterministic. Determinism in the limit is sufficient for verification in probabilistic settings [CY95].

However, determinism in the limit is not sufficient in all situations, for instance in decision procedures for branching-time logics [EJ88], synthesis of reactive systems [PR89], or module checking [KVW01]. In these domains we require full determinization. We therefore extend the profile-based approach to full determinization. This approach provides a mathematically crisp Büchi determinization construction, in which a state of the deterministic automaton is a set of states of the input nondeterministic automaton, a preorder induced by profiles, a second preorder that encodes information about shared ancestry of nodes, and the labeling.

The work described in this dissertation has already had significant impact. For thirty years, the Ramsey-based approach languished. Now, the Ramsey-based approach has become an area of active research. Abdulla et al. built on the Ramsey-based subsumption relation and implement several heuristics to further improve Ramsey-based containment checking [ACC$^+$11]. Lange recently extended the Ramsey-based approach to checking the emptiness of alternating parity automata [Lan11]. Perhaps most excitingly, Breuers, Löding, and Olschewski just published a variant of the Ramsey-based complementation construction with $2^{O(n \log n)}$ worst-case bound, matching other constructions [BLO12].

## 1.4 Outline

Chapter 2 contains the preliminaries used throughout this dissertation, including Büchi automata, the run DAG, complementation, determinization, containment checking, and related topics. We also introduce two existing complementation constructions: the Ramsey-based approach of [SVW87], which has a worst-case complexity of $2^{O(n^2)}$, and the rank-based approach of [KV01], with a worst-case complexity of $2^{O(n \log n)}$.

Chapter 3 introduces the Tabakov-Vardi random automata that underlie the empirical analyses of this work. We then perform an empirical comparison of the two approaches by examining the size of complemented automata. The results demonstrate that for direct complementation neither approach scales to large problems, but that the Ramsey-based approach is surprisingly competitive. Chapter 4, presenting work published in [FV10], introduces an existing on-the-fly containment checking algorithm for the rank-based approach that makes heavy use of a subsumption heuristic [DR07]. We then expand the Ramsey-based approach to a new on-the-fly containment checking algorithms. Finally, we investigate the empirical performance of these two algorithms.

Chapter 5, presenting work published in [FKVW10], introduces another existing approach to complementation, the slice-based approach of Kähler and Wilke [KW08], and then defines a new approach to complementation we call the *profile-based* approach.

Chapter 6 extends the techniques of Chapter 5 to full determinization. We first introduce some definitions of relations over sets, and then defines a mathematically crisp determinization construction based on the notion of profiles. Finally, Chapter 7 concludes with a discussion on the use of Büchi automata as specifications, and presents future work.

# Chapter 2

# Preliminaries

This chapter provides the basic structures and notations used throughout this dissertation. We present Büchi automata, finite state machines that operate over infinite words. To analyze all possible runs of a Büchi automaton on a single word, we present the notion of a run DAG. We then describe in general terms how to check the containment of one Büchi automaton in another. Finally, we present two existing complementation constructions, the rank-based approach and the Ramsey-based approach.

## 2.1 Büchi Automata

A *nondeterministic Büchi automaton on infinite words* (NBW) is a five-tuple $\mathcal{A} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$, where $\Sigma$ is a finite alphabet, $Q$ a finite set of states, $Q^{in} \subseteq Q$ a set of initial states, $F \subseteq Q$ a set of accepting states, and $\rho \colon Q \times \Sigma \to 2^Q$ a nondeterministic transition relation. For convenience, we assume that no initial states are accepting: that $F \cap Q^{in} = \emptyset$. We lift the function $\rho$ to sets $R$ of states in the usual fashion: $\rho(R, \sigma) = \bigcup_{q \in R} \rho(q, \sigma)$. Further, we lift $\rho$ to words word $\sigma_0 \cdots \sigma_i$ by defining $\rho(R, \sigma_0 \cdots \sigma_i)$ = $\rho(\rho(R, \sigma_0), \sigma_1 \cdots \sigma_i)$. For completeness, let $\rho(R, \epsilon) = R$.

An infinite *run* of an NBW $\mathcal{A}$ on an infinite word $w = \sigma_0 \sigma_1 \cdots \in \Sigma^\omega$ is an infinite sequence of states $p_0, p_1, \ldots \in Q^\omega$ such that $p_0 \in Q^{in}$ and, for every $i \geq 0$, we have $p_{i+1} \in \rho(p_i, \sigma_i)$. Correspondingly, a *finite run of $\mathcal{A}$ from $p$ to $q$ on $w = \sigma_0 \cdots \sigma_{n-1}$* is a finite sequence of states $p_0, \ldots, p_n$ such that $p_0 = p$, $p_n = q$, and for every $0 \leq i \leq n$ we have $p_{i+1} \in \rho(p_i, \sigma_i)$. We say there is a *finite run of $\mathcal{A}$ from $p$ to $q$* when there is a

$w \in \Sigma^*$ and a finite run from $p$ to $q$ on $w$. Finally, say there is a *finite run of* $\mathcal{A}$ *to* $q$ *on* $w$ when there is a $p \in Q^{in}$ and a finite run from $p$ to $q$ on $w$. When unspecified, a run refers to an infinite run. A finite run is *accepting* when $p_i \in F$ for some $i$. An infinite run is *accepting* iff $p_i \in F$ for infinitely many $i \in \mathbb{N}$. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on $w$. The words accepted by $\mathcal{A}$ form the *language* of $\mathcal{A}$, denoted by $L(\mathcal{A})$. The complement of $L(\mathcal{A})$, denoted $\overline{L(\mathcal{A})}$, is $\Sigma^\omega \setminus L(\mathcal{A})$. To *complement* an automaton $\mathcal{A}$ is to create an automaton $\overline{\mathcal{A}}$ so that $L(\overline{\mathcal{A}}) = \overline{L(\mathcal{A})}$.

A state $q \in Q$ is *deterministic* if for every $\sigma \in \Sigma$ it holds that $|\rho(q, \sigma)| \leq 1$. An automaton is *deterministic* if every state is deterministic. To *determinize* an automaton $\mathcal{A}$ is to create a deterministic automaton $\mathcal{A}'$ so that $L(\mathcal{A}) = L(\mathcal{A}')$. For NBW, there is not always a deterministic equivalent, and determinization results in automata with different acceptance conditions. This is discussed in more detail in Chapter 6. However, there are intermediate levels of determinism. Say an automaton is *deterministic in the limit* if every state reachable from an accepting state is deterministic. Converting an NBW $\mathcal{A}$ to an equivalent deterministic in the limit automaton involves an exponential blowup [CY95, Saf88]. One can simultaneously complement and determinize in the limit, via co-determinization into a parity automaton [Pit06], and then converting that parity automaton to a deterministic-in-the-limit Büchi automaton, with a cost of $(n^2/e)^n$. Alternately, say an automaton $\mathcal{A}$ in *unambiguous* when, for every $w \in \Sigma^\omega$, there is at most one accepting run of $\mathcal{A}$ on $w$.

*Example* 2.1. In Figure 2.1 we describe an NBW $\mathcal{B}$ that accepts words with a finite but non-zero number of $b$'s. Intuitively, we wait in $p$ until we guess we have seen the last $b$, on which we transition to $q$. If we guess wrong, we can escape to $r$ and try again, transitioning to $t$ on the final $b$.

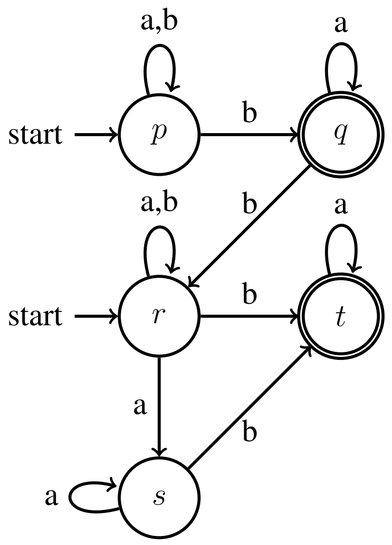


**Figure 2.1 :** The automaton $\mathcal{B}$, which has five states. $\mathcal{B}$ accepts words in which the letter $b$ occurs a finite, but non-zero, number of times.

**Run DAG:** Central to many of the constructions in this work is the notion of a run DAG. Consider an NBW $\mathcal{A}$ and an infinite word $w = \sigma_0\sigma_1\cdots$. The runs of $\mathcal{A}$ on $w$ can be arranged in an infinite DAG (directed acyclic graph) $G = \langle V, E\rangle$, where

- $V \subseteq Q \times \mathbb{N}$ is such that $\langle q, i\rangle \in V$ iff there is a finite run of $\mathcal{A}$ to $q$ on $w_0 \cdots w_{i-1}$.
- $E \subseteq \bigcup_{i\geq 0} (Q\times\{i\}) \times (Q\times\{i+1\})$ is s.t. $E(\langle q, i\rangle, \langle q', i+1\rangle)$ iff $\langle q, i\rangle \in V$ and $q' \in \rho(q, \sigma_i)$.

The DAG $G$, called the *run* DAG *of* $\mathcal{A}$ *on* $w$, embodies all possible runs of $\mathcal{A}$ on $w$. We are primarily concerned with *initial paths* in $G$: paths that start in $Q^{in} \times \{0\}$. Define a node $\langle q, i\rangle$ to be an $F$-node when $q \in F$, and a path in $G$ to be *accepting* when it is both initial and contains infinitely many $F$-nodes. An accepting path in $G$ corresponds to an accepting run of $\mathcal{A}$ on $w$. When $G$ contains an accepting path, call $G$ an accepting run DAG, otherwise call it a rejecting run DAG. We often consider DAGs $H$ that are subgraphs of $G$. A node $u$ is a *descendant* of $v$ in $H$ when $u$ is reachable from $v$ in $H$.

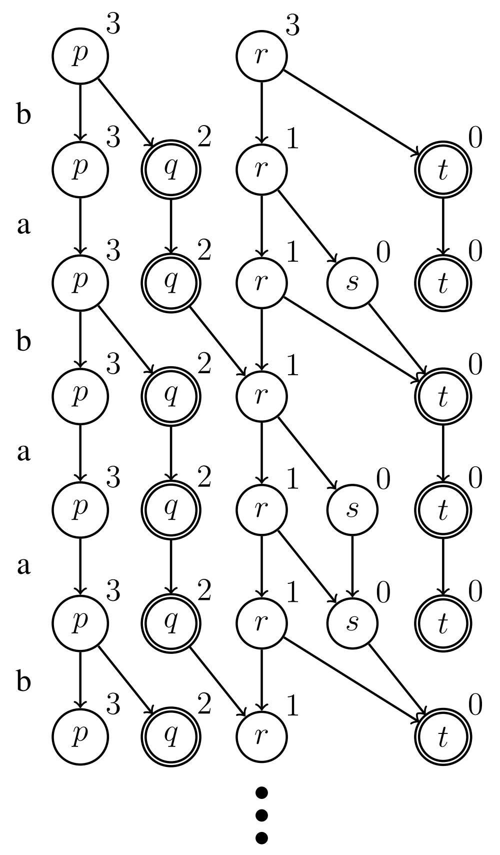


**Figure 2.2 :** The first seven levels of the rejecting run DAG $G$ for the automaton $\mathcal{B}$, from Figure 2.1, on $w = babaabaaabaaaa\cdots$. Nodes are superscripted with the prospective labels of Chapter 2.2.1.

A node $v$ is *finite* in $H$ if it has only finitely many descendants in $H$. A node $v$ is $F$*-free* in $H$ if it is not an $F$-node, and has no descendants in $H$ that are $F$-nodes. We say a node *splits* when it has at least two children, and conversely that two nodes *join* when they share a common child.

**Containment Checking:** We say that a Büchi automaton $\mathcal{A}$ is contained in a Büchi automaton $\mathcal{B}$ iff $L(\mathcal{A}) \subseteq L(\mathcal{B})$. To check containment, we verifying that the intersection of $\mathcal{A}$ with $\overline{\mathcal{B}}$ is empty: $L(\mathcal{A}) \cap L(\overline{\mathcal{B}}) = \emptyset$. Given $\mathcal{A}$ and $\overline{\mathcal{B}}$, we can compute their product: an automaton whose language is $L(\mathcal{A}) \cap L(\overline{\mathcal{B}})$, which has has a number of states proportional to the product of the number states of the original automaton [Cho74]. Further, we know that the language of an automaton is non-empty iff there are states $q \in Q^{in},\ r \in F$ such that there is a path from $q$ to $r$ and a path from $r$ to itself. The initial path is called the prefix, and the combination of the prefix and cycle is called a *lasso* [Var07b]. Checking for a lasso in an automaton can be done in linear time [CVWY92], although if a symbolic or semi-symbolic representation of states is used, a quadratic algorithm is often preferred [EL86, DR07]. Thus the most computationally demanding step is constructing the complement of $\mathcal{B}$. In the formal verification field, a paucity of real-world containment problems has limited empirical work. Existing empirical work has focused on the simplest form of containment testing, *universality* testing, where $\mathcal{A}$ is the universal automaton [DR09, TV07]. In universality testing, we answer if $\Sigma^\omega \subseteq L(\mathcal{B})$ by checking that $L(\overline{\mathcal{B}}) = \emptyset$.

In practice, we do not want to explicitly compute the complement of $\mathcal{B}$, none the less compute the product of $\mathcal{A}$ and the complement of $\mathcal{B}$. These automata can be very large, and it may not be necessary to search the entire automaton to find a counterexample. Instead, we can construct the automaton *on-the-fly* and search only a subset of states. An on the fly algorithm maintains one or more sets of reachable states, and performs tests on these sets to either search for a lasso or rule out the possibility of a lasso. Often, these algorithm employ heuristics. One of the most powerful heuristics for algorithms searching Büchi automata has been the use of a *subsumption* relation. A subsumption relation $\prec$ relates certain states in the automaton being searched, so that if a state $q$ subsumes a state $r$, written $q \prec r$, then $r$ can be discarded from a set that contains $q$. Intuitively, if $r$ is on a lasso, then $q$ must also be on a lasso, and there is no need to consider $r$. Thus a set needs to contain only the minimal elements under the subsumption relation.

## 2.2 Complementation Constructions

We present two complementation constructions central to this dissertation. The first is the *rank-based construction* from Kupferman and Vardi, proposed in 2001. This construction is based on mapping the nodes of the run DAG to ranks, where the rank of a node essentially indicates the progress made towards a suffix of the run with no accepting states. Further, all the runs of $\mathcal{B}$ on $w$ are rejecting iff there is a certain kind of ranking, called a *bounded odd ranking*, of the DAG. The second is the *Ramsey-based construction* from Sistla, Vardi, and Wolper, proposed in 1987. This construction is based on encoding the connectivity of the automaton in a set of arc-labeled graphs over states. Each graph describes a set of finite words that are processed by the automaton in the same way, and so partitions the infinite set of finite words into a finite set of equivalence classes.

### 2.2.1 Rank-Based Complementation

If an NBW $\mathcal{B}$ does not accept a word $w$, then every run of $\mathcal{B}$ on $w$ must eventually cease visiting accepting states. The notion of *ranking*s, foreshadowed in [Kla90] and introduced in [KV01], uses natural numbers to track the progress of each run in the DAG towards this point. A ranking for a DAG $G = \langle V, E\rangle$ is a mapping from $V$ to $\mathbb{N}$, in which no $F$-node is given an odd rank, and in which the ranks along all paths do not increase. Formally, a ranking is a function $\mathbf{r}\colon V \to \mathbb{N}$ such that if $u \in V$ is an $F$-node then $\mathbf{r}(u)$ is even; and for every $u, v \in V$, if $(u, v) \in E$ then $\mathbf{r}(u) \geq \mathbf{r}(v)$. Since each path starts at a finite rank and ranks cannot increase, every path eventually becomes trapped in a rank. A ranking is called an *odd ranking* if every path becomes trapped in an odd rank.

Since $F$-nodes cannot have odd ranks, if there exists an odd ranking $\mathbf{r}$, then every path in $G$ must stop visiting accepting nodes when it becomes trapped in its final, odd, rank, and $G$ must be a rejecting DAG.

**Lemma 2.2.** [KV01] *If a run* DAG *$G$ has an odd ranking, then $G$ is rejecting.*

A ranking is *bounded by* $l$ when its range is $\{0, ..., l\}$, and an NBW $\mathcal{B}$ is of rank $l$ when for every $w \notin L(\mathcal{B})$, the rejecting DAG $G$ has an odd ranking bounded by $l$. If we can prove that an NBW $\mathcal{B}$ is of rank $l$, we can use the notion of odd rankings to construct a complementary automaton. This complementary NBW, denoted $C^R_l(\mathcal{B})$, tracks the levels of the run DAG and attempts to guess an odd ranking bounded by $l$. An *$l$-bounded level ranking* for an NBW $\mathcal{B}$ is a function $f\colon Q_\mathcal{B} \to \{0, \ldots, l, \perp\}$, such that if $q \in F_\mathcal{B}$ then $f(q)$ is even or $\perp$. Let $\mathcal{R}^l$ be the set of all $l$-bounded level rankings. The state space of $C^R_l(\mathcal{B})$ is based on the set of $l$-bounded level rankings for $\mathcal{B}$. To define transitions of $C^R_l(\mathcal{B})$, we need the following notion: for $\sigma \in \Sigma$ and $f, f' \in \mathcal{R}^l$, say that *$f'$ follows $f$ under $\sigma$* when for every $q \in Q_\mathcal{B}$ and $q' \in \rho_\mathcal{B}(q, \sigma)$, if $f(q) \neq \perp$ then $f'(q') \neq \perp$ and $f'(q') \leq f(q)$: i.e. no transition between $f$ and $f'$ on $\sigma$ increases in rank. Finally, to ensure that the guessed ranking is an odd ranking, we employ the cut-point construction of Miyano and Hayashi, which maintains an obligation set of nodes along paths obliged to visit an odd rank [MH84]. For a level ranking $f$, let $even(f) = \{q \mid f(q) \text{ is even}\}$ and $odd(f) = \{q \mid f(q) \text{ is odd}\}$.

**Definition 2.3.** For an NBW $\mathcal{B} = \langle \Sigma, Q_\mathcal{B}, Q^{in}_\mathcal{B}, \rho_\mathcal{B}, F_\mathcal{B} \rangle$ and $l \in \mathbb{N}$, define $C^R_l(\mathcal{B})$ to be the NBW $\langle \Sigma, \mathcal{R}^l \times 2^{Q_\mathcal{B}}, \langle f^{in}, \emptyset \rangle, \rho_R, \mathcal{R}^l \times \{\emptyset\} \rangle$, where:

- $f^{in}(q) = l$ for each $q \in Q^{in}_\mathcal{B}$, $\perp$ otherwise.
- $\rho_R(\langle f, O \rangle, \sigma) = \begin{cases} \{\langle f', \rho_\mathcal{B}(O, \sigma) \setminus odd(f') \rangle \mid f' \text{ follows } f \text{ under } \sigma\} & \text{if } O \neq \emptyset, \\ \{\langle f', even(f') \rangle \mid f' \text{ follows } f \text{ under } \sigma\} & \text{if } O = \emptyset. \end{cases}$

**Lemma 2.4.** [KV01] *For every $l \in N$, it holds that $L(C^R_l(\mathcal{B})) \subseteq \overline{L(\mathcal{B})}$.*

Lemma 2.4 states that for every $l \in \mathbb{N}$, the NBW $C^R_l(\mathcal{B})$ accepts only words rejected by $\mathcal{B}$ — exactly all words for which there exists an odd ranking with maximal rank $l$. In addition, [KV01] prove that for every rejecting run DAG there exists a bounded odd ranking. Below we sketch the derivation of this ranking. Given a rejecting run DAG $G$, we inductively define a sequence of subgraphs by eliminating nodes that cannot be part of accepting runs. At odd steps we remove finite nodes, while in even steps we remove nodes that are $F$-free. Formally, define a sequence of subgraphs as follows:

- $G_0 = G$.
- $G_{2i+1} = G_{2i} \setminus \{v \mid v \text{ is finite in } G_{2i}\}$.
- $G_{2i+2} = G_{2i+1} \setminus \{v \mid v \text{ is } F\text{-free in } G_{2i+1}\}$.

It is shown in [GKSV03, KV01] that only $m = 2|Q_\mathcal{B} \setminus F_\mathcal{B}|$ steps are necessary to remove all nodes from a rejecting run DAG: $G_m$ is empty. Nodes can be ranked by the last graph in which they appear: for every node $u \in G$, the *prospective rank* of $u$ is the index $i$ such that $u \in G_i$ but $u \notin G_{i+1}$. The *prospective ranking* of $G$ assigns every node its prospective rank. Paths through $G$ cannot increase in prospective rank, and no $F$-node can be given an odd rank: thus the prospective ranking abides by the requirements for rankings. We call these rankings prospective because the rank of a node depends solely on its descendants. By [KV01], if $G$ is a rejecting run DAG, then the prospective ranking of $G$ is an odd ranking bounded by $m$. By the above, we thus have the following.

**Theorem 2.5.** [KV01] *For every NBW $\mathcal{B}$, it holds that $L(C^R_m(\mathcal{B})) = \overline{L(\mathcal{B})}$.*

*Example* 2.6. In Figure 2.2, nodes for states $s$ and $t$ are finite in $G_0$. With these nodes removed, $r$-nodes are $F$-free in $G_1$. Without $r$-nodes, $q$-nodes are finite in $G_2$. Finally,

$p$-nodes are $F$-free in $G_3$.

A number of improvements have been proposed since the initial introduction of the rank-based construction. The most notable improvements are the introduction of tight rankings [FKV06], presented below, and Schewe's improved cut-point construction [Sch09a]. These improvements tightened the $(6n)^n$ upper bound of [KV01] to $(0.76n)^n$. Together with recent work on a tighter lower bound [Yan06], the gap between the upper and lower bound is now a quadratic term. Karmarkar and Chakraborty have derived both theoretical and practical benefits from exploiting properties of this prospective ranking: they demonstrated an unambiguous complementary automaton that, for certain classes of problems, is exponentially smaller than $C^R_m(\mathcal{B})$ [KC09].

**Tight Rankings:** For an odd ranking **r** and $l \in \mathbb{N}$, let $max\_rank(\mathbf{r}, l)$ be the maximum rank that **r** assigns a vertex on level $l$ of the run DAG. We say that **r** is *tight*[1] if there exists an $i \in \mathbb{N}$ such that, for every level $l \geq i$, all odd ranks below $max\_rank(\mathbf{r}, l)$ appear on level $l$. It is shown in [FKV06] that the retrospective ranking is tight. This observation suggests two improvements to $C^R_m(\mathcal{B})$. First, we can postpone, in an unbounded manner, the level in which it starts to guess the level ranking. Until this point, $C^R_m(\mathcal{B})$ may use sets of states to deterministically track only the levels of the run DAG, with no attempt to guess the ranks. Second, after this point, $C^R_m(\mathcal{B})$ can restrict attention to *tight level rankings* – ones in which all the odd ranks below the maximal rank appear. Formally, say a level ranking $f$ with a maximum rank $mr = \max\{f(q) \mid q \in Q,\ f(q) \neq \bot\}$ is tight when, for every odd $i \leq mr$, there exists a $q \in Q$ such that $f(q) = i$. Let $\mathcal{R}^m_T$ be the subset of $\mathcal{R}^m$ that contains only tight level rankings. The size of $\mathcal{R}^m_T$ is at most

[1]This definition of tightness for an odd ranking is weaker that of [FKV06], but does not affect the resulting bounds.

$(0.76n)^n$ [FKV06]. Including the cost of the cut-point construction, this reduces the state space of $C^R_m(\mathcal{B})$ to $(0.96n)^n$.

### 2.2.2 Ramsey-Based Complementation

When Büchi introduced these automata in 1962, he described a complementation construction involving a Ramsey-based combinatorial argument [Büc62]. We describe an optimized implementation presented in 1987 [SVW87]. To construct the complement of $\mathcal{B} = \langle \Sigma, Q_\mathcal{B}, Q^{in}_\mathcal{B}, \rho_\mathcal{B}, F_\mathcal{B}\rangle$ we define a set $\widetilde{Q}_\mathcal{B}$ whose elements capture the essential behavior of $\mathcal{B}$. Each element corresponds to an answer to the following question:

Given a finite nonempty word $w$, for every two states $q, r \in Q_\mathcal{B}$:

(1) Is there a finite run of $\mathcal{B}$ from $q$ to $r$ over $w$?

(2) If so, is some such finite run accepting?

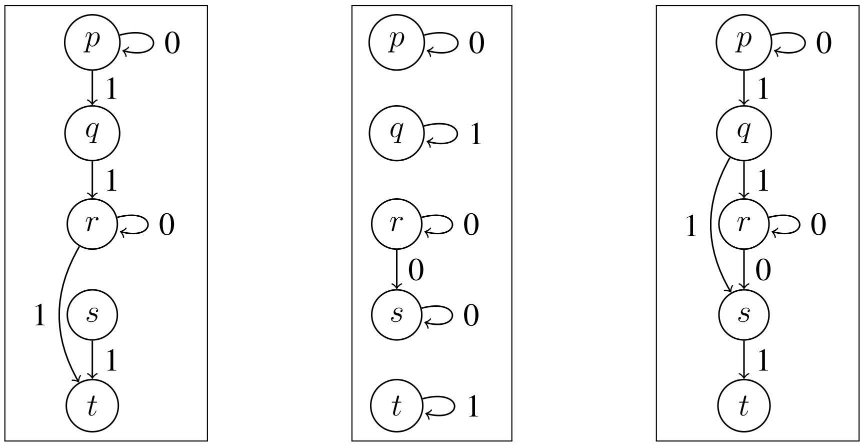
Define $Q' = Q_\mathcal{B} \times \{0, 1\} \times Q_\mathcal{B}$, and $\widetilde{Q}_\mathcal{B}$ to be the subset of $2^{Q'}$ whose elements, for every $q, r \in Q_\mathcal{B}$, do not contain both $\langle q, 0, r\rangle$ and $\langle q, 1, r\rangle$. Each element of $\widetilde{Q}_\mathcal{B}$ is a $\{0, 1\}$-arc-labeled graph on $Q_\mathcal{B}$. An arc represents a finite run of $\mathcal{B}$, and the label is 1 if the finite run is accepting. Note that there are $3^{n^2}$ such graphs. With each graph $\widetilde{g} \in \widetilde{Q}_\mathcal{B}$ we associate a language $L(\widetilde{g})$, the set of words for which the answer to the posed question is the graph encoded by $\widetilde{g}$.

**Definition 2.7.** Let $\widetilde{g} \in \widetilde{Q}_\mathcal{B}$ and $w \in \Sigma^+$. Then $w \in L(\widetilde{g})$ iff, for all pairs of states $q, r \in Q_\mathcal{B}$:

(1) $\langle q, a, r\rangle \in \widetilde{g},\ a \in \{0, 1\}$, iff there is a finite run of $\mathcal{B}$ from $q$ to $r$ over $w$.

(2) $\langle q, 1, r\rangle \in \widetilde{g}$ iff there is an accepting finite run of $\mathcal{B}$ from $q$ to $r$ over $w$.

*Example* 2.8. Three graphs from $\widetilde{Q}_\mathcal{B}$ are shown in Figure 2.1. All graphs have a nonempty language. The word $b$ is in the language of the first graph, the word $a$ is in the



**(a)** $\widetilde{q}_b$ **(b)** $\widetilde{q}_a$ **(c)** $\widetilde{q}_{ba}$

**Figure 2.3 :** Three graphs from $\widetilde{Q}_{\mathcal{B}}$, where $\mathcal{B}$ is from Figure 2.1. All graphs have a non-empty language. The word $b$ is in the language of the first graph, the word $a$ is in the language of the second graph, and the word $ba$ is in the language of the third graph. The arcs of $\widetilde{q}_{ba}$ indicate that there is an accepting finite run from $q$ to $s$ on $ba$, and a finite run from $r$ to $s$ on $ba$, but no accepting finite run from $r$ to $s$ on $ba$.

language of the second graph, and the word $ba$ is in the language of the third graph.

**Lemma 2.9.** [Büc62, SVW87]

*(1)* $\{L(\widetilde{g}) \mid \widetilde{g} \in \widetilde{Q}_{\mathcal{B}}\}$ *is a partition of* $\Sigma^+$

*(2) If* $u \in L(\widetilde{g}),\ v \in L(\widetilde{h}),$ *and* $uv \in L(\widetilde{k}),$ *then* $L(\widetilde{g}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{k})$

The languages $L(\widetilde{g})$, for the graphs $\widetilde{g} \in \widetilde{Q}_{\mathcal{B}}$, form a partition of $\Sigma^+$. With this partition of $\Sigma^+$ we can devise a finite family of $\omega$-languages that cover $\Sigma^\omega$. For every $\widetilde{g},\ \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$, let $Y(\widetilde{g}, \widetilde{h})$ be the $\omega$-language $L(\widetilde{g}) \cdot L(\widetilde{h})^\omega$. We can restrict our attention to a subset of these languages through a notion of properness.

**Definition 2.10.** A pair of graphs $\langle \widetilde{g}, \widetilde{h} \rangle$ is *proper* if $Y(\widetilde{g}, \widetilde{h})$ is non-empty, $L(\widetilde{g}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{g})$, and $L(\widetilde{h}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{h})$.

We say that $Y(\widetilde{g}, \widetilde{h})$ is proper when $\langle \widetilde{g}, \widetilde{h} \rangle$ is proper. There are a finite, if exponential, number of such languages. A Ramsey-based argument shows that every infinite string belongs to a language of this form, and that $\overline{L(\mathcal{B})}$ can be expressed as the union of languages of this form.

**Lemma 2.11.** [Büc62, SVW87] $\Sigma^\omega = \bigcup\{Y(\widetilde{g}, \widetilde{h}) \mid Y(\widetilde{g}, \widetilde{h})$ *is proper*$\}$

*Proof.* The proof is based on Ramsey's Theorem. Consider an infinite word $w = \sigma_0\sigma_1...$ By Lemma 2.9, every prefix of the word $w$ is in the language of a unique graph $\widetilde{g}_i$. Let $k = 3^{n^2}$ be the number of graphs. Thus $w$ defines a partition of $\mathbb{N}$ into $k$ sets $D_1, ..., D_k$ such that $i \in D_l$ iff $\sigma_0...\sigma_{i-1} \in L(\widetilde{g}_l)$. Clearly there is some $m$ such that $D_m$ is infinite.

Similarly, by Lemma 2.9 we can use the word $w$ to define a partition of all *pairs* of elements $(i, j)$ from $D_m$, where $i < j$. This partition consists of $k$ sets $C_1, ...C_k$, such that $\langle i, j \rangle \in C_l$ iff $\sigma_i...\sigma_{j-1} \in L(\widetilde{g}_l)$. Ramsey's Theorem tells us that, given such a partition, there exists an infinite subset $\{i_1, i_2, ...\}$ of $D_m$ and a $C_n$ such that for all pairs of distinct elements $i_j, i_k$, it holds that $\langle i_j, i_k \rangle \in C_n$.

This implies that the word $w$ can be partitioned into

$$w_0 = \sigma_0...\sigma_{i_1-1}$$
$$w_1 = \sigma_{i_1}...\sigma_{i_2-1}$$
$$w_2 = \sigma_{i_2}...\sigma_{i_3-1}$$
$$\cdots$$

where $w_0 \in L(\widetilde{g}_m)$ and $w_i \in L(\widetilde{g}_n)$ for $i > 0$. By construction, $\sigma_0...\sigma_{i_j-1} \in L(\widetilde{g}_m)$ for every $i_j$, and thus we have that $w_0w_1 \in L(\widetilde{g}_m)$. In addition, as $\sigma_{i_j}...\sigma_{i_k-1} \in L(\widetilde{g}_n)$ for every pair $i_j, i_k$, we have that $w_1w_2 \in L(\widetilde{g}_n)$. By Lemma 2.9, it follows that $L(\widetilde{g}_m) \cdot L(\widetilde{g}_n) \subseteq L(\widetilde{g}_m)$, and that $L(\widetilde{g}_n) \cdot L(\widetilde{g}_n) \subseteq L(\widetilde{g}_n)$, and thus $Y(\widetilde{g_m}, \widetilde{g_n})$ is proper. $\square$

Furthermore, each proper language is entirely contained or entirely disjoint from $L(\mathcal{B})$. This provides a way to construct the complement of $L(\mathcal{B})$: take the union every proper language that is disjoint from $L(\mathcal{B})$.

**Lemma 2.12.** [Büc62, SVW87]

*(1) For $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$, either $Y(\widetilde{g}, \widetilde{h}) \cap L(\mathcal{B}) = \emptyset$ or $Y(\widetilde{g}, \widetilde{h}) \subseteq L(\mathcal{B})$*

*(2) For a proper $Y(\widetilde{g}, \widetilde{h})$, it holds that $Y(\widetilde{g}, \widetilde{h}) \subseteq L(\mathcal{B})$ iff there exists $q \in Q^{in}_{\mathcal{B}},\ r \in Q_{\mathcal{B}},\ a \in \{0,1\}$ where $\langle q, a, r\rangle \in \widetilde{g}$ and $\langle r, 1, r\rangle \in \widetilde{h}$.*

*(3) $\overline{L(\mathcal{B})} = \bigcup\{Y(\widetilde{g}, \widetilde{h}) \mid Y(\widetilde{g}, \widetilde{h})$ is proper and $Y(\widetilde{g}, \widetilde{h}) \cap L(\mathcal{B}) = \emptyset\}$*

To obtain the complementary Büchi automaton $\overline{\mathcal{B}}$, Sistla et al. construct a family of deterministic automata on finite words that accept, for each $\widetilde{g} \in \widetilde{Q}_{\mathcal{B}}$, $L(\widetilde{g})$. From these automata and Lemma 2.11, we can construct the complementary automaton $\overline{\mathcal{B}}$. The state space of these automata is $\widetilde{Q}_{\mathcal{B}} \cup \{p_0\}$, the set of graphs with the addition of a start state. $\mathcal{A}_q$ is then $\langle \Sigma, \widetilde{Q}_{\mathcal{B}} \cup \{p_0\}, \widetilde{\rho}, p_0, \{\widetilde{q}\}\rangle$ where the deterministic transition function $\widetilde{\rho} : \widetilde{Q}_{\mathcal{B}} \cup \{p_0\} \times \Sigma \to \widetilde{Q}_{\mathcal{B}}$ is:

$$\begin{aligned}
\widetilde{\rho}(p_0, a) &= \{\langle q, 0, r\rangle \mid q \in Q_{\mathcal{B}},\ r \in Q_{\mathcal{B}} \setminus F_{\mathcal{B}},\ r \in \rho_{\mathcal{B}}(q, a)\} \\
&\cup \{\langle q, 1, r\rangle \mid q \in Q_{\mathcal{B}},\ r \in F_{\mathcal{B}},\ r \in \rho_{\mathcal{B}}(q, a)\} \\
\widetilde{\rho}(\widetilde{q}, a) &= \{\langle q, 0, s\rangle \mid \langle q, 0, r\rangle \in \widetilde{q},\ s \in \rho_{\mathcal{B}}(r, a) \setminus F_{\mathcal{B}}\} \\
&\cup \{\langle q, 1, s\rangle \mid \langle q, 0, r\rangle \in \widetilde{q},\ s \in \rho_{\mathcal{B}}(r, a) \cap F_{\mathcal{B}}\} \\
&\cup \{\langle q, 1, s\rangle \mid \langle q, 1, r\rangle \in \widetilde{q},\ s \in \rho_{\mathcal{B}}(r, a)\}
\end{aligned}$$

**Lemma 2.13.** [SVW87] $L(A_q) = L(\widetilde{q})$

Knowing how to construct an automaton for each $L(\widetilde{q})$, one can create the complementary automaton $C^{\text{Ramsey}}(\mathcal{B})$. For each proper $Y_{qr}$, create from $A_q$ and $A_r$ a Büchi automaton $A_{qr}$ that accepts $Y_{qr}$. Simply find all $Y_{qr}$ disjoint from $L(A)$ and construct the union automaton of the corresponding $A_{qr}$.

We pause to note that we can avoid an explicit lasso search over the complementary automaton by employing the rich structure of the graphs in $\widetilde{Q}_{\mathcal{B}}$. For every two graphs $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$, determine if $Y(\widetilde{g}, \widetilde{h})$ is proper. If $Y(\widetilde{g}, \widetilde{h})$ is proper, test if it is contained in $L(\mathcal{B})$. In order to test if a proper language $Y(\widetilde{g}, \widetilde{h})$ is contained in $L(\mathcal{B})$, search for a $q \in Q^{in}_{\mathcal{B}},\ r \in Q_{\mathcal{B}},\ a \in \{0, 1\}$ such that the arc $\langle q, a, r\rangle \in \widetilde{g}$ and the arc $\langle r, 1, r\rangle \in \widetilde{h}$. We call this test of a pair of graphs the *two-arc test*. $\mathcal{B}$ is universal if every proper $Y(\widetilde{g}, \widetilde{h})$ is so contained.

**Lemma 2.14.** [SVW87] *A Büchi automaton $\mathcal{B}$ is universal iff every proper pair $\langle \widetilde{g}, \widetilde{h}\rangle$ of graphs from $\widetilde{Q}_{\mathcal{B}}$ passes the two-arc test.*

Lemma 2.14 yields a PSPACE algorithm to determine universality [SVW87]. Simply check each $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$. If $Y(\widetilde{g}, \widetilde{h})$ is both proper and not contained in $L(\mathcal{B})$, then the pair $(\widetilde{g}, \widetilde{h})$ provide a counterexample to the universality of $\mathcal{B}$. If no such pair exists, the automaton must be universal.

# Chapter 3

# Complementation Experiments

This chapter performs some experiments on the rank and Ramsey-based complementation constructions. Because Büchi specifications are not used in practice, there is a paucity of real-world problems on which to evaluate approaches. When specifications are given as logical formulas, Büchi automata are used as an intermediate step. However, because formulas have a constant-size negation, automata derived from formulas often have a linear-sized complement. This makes them a poor suite on which to test complementation problems. Therefore we employ a random model of automata that has seen significant use in the literature. We first present the model of random automata.

## 3.1 Preliminaries: Tabakov-Vardi Random Automata

Random automata were first proposed by Tabakov and Vardi as a method for testing tools over automata on finite words, and later for tools over automata on infinite words [TV05, WDHR06, TV07, DR07]. This model fixes the input alphabet as $\Sigma = \{0, 1\}$. Each automaton $\mathcal{B} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$ is constructed according to three parameters: a *size* $n$, the *transition density* $r$, and the *acceptance density* $f$. The set of states $Q$ is the set $\{0...n-1\}$. Only the first state is initial, and thus $Q^{in} = \{0\}$. For each letter $\sigma \in \Sigma$, we choose $\lceil n * r \rceil$ pairs of states $(s, s') \in Q^2$ uniformly at random and add the transition $\langle s, \sigma, s' \rangle$ are included in $\rho$. Thus the transition density reflects the expected number

This chapter contains unpublished work, building on that published in [TFVT11].

of transitions from each state on each letter, and is related to Karp's model of random directed graphs [Kar90]. Karp shows that when the transition density is greater than 1, it is very likely that a graph contains a strongly connected component: a necessary condition for a Büchi automaton to accept a word. We impose one exception to avoid trivial cases of non-universality: the initial node must have at least one outgoing transition for each letter of the alphabet. The set of accepting states is likewise constructed as a linear function of the number of states: $F$ comprises $\lceil n * f \rceil$ states chosen uniformly at random.

When devising a random model, we want to know that the parameters can generate a variety of interesting problems. Figure 3.1, taken from [TV07], demonstrates that the chance of a random automaton of size 6 being universal. The behavior indicates that the parameters do generate an interesting space of problems, and that universality is highly correlated with the parameters. Automata with a high transition density tend to be universal, while automata with low transition density tend to be non-universal. With a transition density of 3, nearly all automata are universal. Correspondingly, with a transition density of 1, nearly all automata are non-universal. Acceptance density has a smaller, but still noticeable, affect on universality [TV07].

Experiments using the Tabakov-Vardi model can be broadly classes into two categories. *Terrain* experiments hold the size of the automata constant, and vary the acceptance and transition densities. These produce results like Figure 3.1, that demonstrate behavior over a variety of configurations. Terrain experiments are good for discovering which configurations of problems are hard for a given approach. They provide an empirical description of how transition and acceptance density might affect the performance of an approach. However, terrain experiments are not a good way to judge how well an approach performs, as they give no indication of the scalability of the approach.

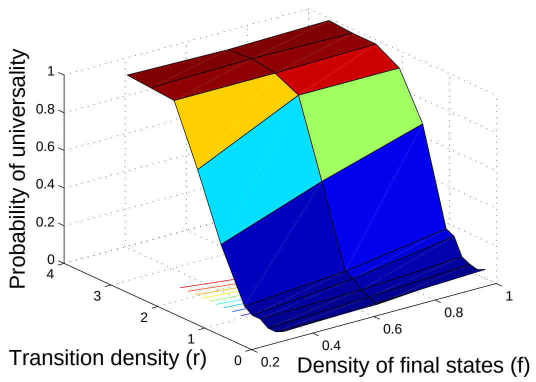


**Figure 3.1 :** Probability that a random automaton of size 6 is universal, varying by transition density and acceptance density.

Thus when comparing the suitability of two approaches, it is inappropriate to use terrain experiments.

To test scalability, we use *scaling* experiments. In a scaling experiment, we pick an interesting configuration of transition and acceptance densities, and only use automata with that configuration. We then examine how an approach performs as we increase the size of the automata. By plotting the results, we can estimate the real-world scalability of the approach. In many cases, algorithms with exponential worst cast demonstrate less-than-exponential real world performance. Thus scaling experiments allow us to directly compare two approaches on a single configuration of random automata.

## 3.2 Empirical Comparisons: Size of Complemented Automata

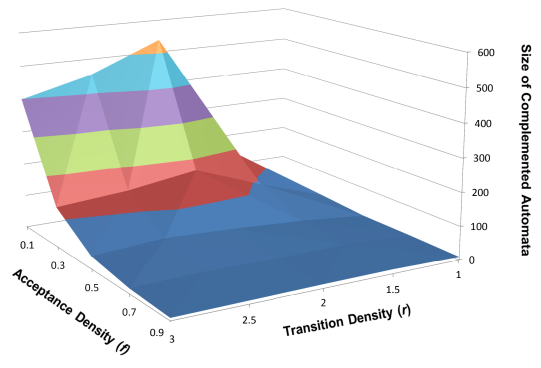
We compare the size of complemented automata using both the rank and Ramsey-based complementation constructions. Complemented automata are derived from the GOAL tool. The rank-based construction employs several optimizations, including the use of the tight level rankings described in Chapter 2.2.1 and Schewe's turn-wise cut-point construction. Further, we note that the number of accepting states serves to both limit the maximum rank, and to reduce the number of legal level rankings. Thus we heuristically attempt to increase the number of accepting states. If all paths from a state must lead to an accepting state, then that state can be marked as accepting. There are no effective heuristics for the Ramsey-based approach: in fact, maximizing the size of the accepting set ends up generating larger automata using the Ramsey-based construction. The results are not minimized, but the GOAL tool only computes the reachable states in a construction [TFVT11].

Data points are derived from 100 or more random automata with the given $n$, $r$, and $f$. All experiments were performed on the Shared University Grid at Rice (SUG@R)[1], a cluster of Sunfire x4150 nodes, each with two 2.83GHz Intel Xeon quad-core processors and 16GB of RAM. Each tool is given a dedicated node.

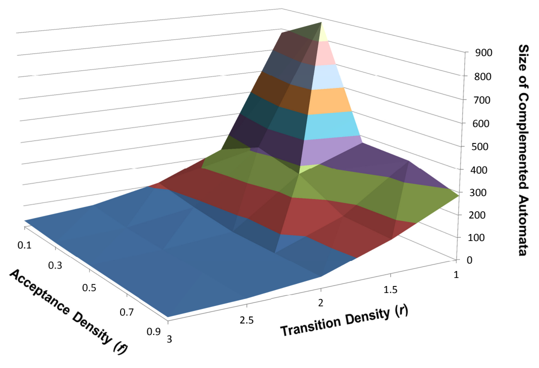
The first experiments were terrain experiments, on automata of size 6. As stated above, randomly generated automata of a given size can be configured by transition and acceptance density. To map out the behavior of the two approaches over the terrain of configurations, we hold size constant at $n = 6$, and examined a variety of configurations. We generate data points for each combination of transition density $r \in \{1, 1.5, 2, 2.5, 3\}$ and acceptance density $f \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. We attempted to run the tool to completion on all problems. When this was not possible, we discarded the case. Out of 2500 cases, only eight had to be discarded.

Figure 3.2(a) displays the median size for rank-based complementation. As expected, the rank-based complementation is very sensitive to the number of accepting states. This illustrates the utility of maximizing the accepting set of the input automaton. However, it remains to be seen if this heuristic is equally effective on more structured,

[1] `http://rcsg.rice.edu/sugar/`, funded by NSF under Grant EIA-0216467, and a partnership between Rice University, Sun Microsystems, and Sigma Solutions, Inc.



**(a)** Rank-based Complementation



**(b)** Ramsey-based Complementation

**Figure 3.2 :** Differences in behavior between Rank and Ramsey-based complementation constructions over a terrain of random automata, measured as the median number of states of the resulting automata, when size $n = 6$.

human-generated, problems. Figure 3.2(b) displays the median size for Ramsey-based complementation. Ramsey-based complementation is very sensitive to the transition density. A higher transition density results in denser graphs, and seems to reduce the number of different graphs required to describe all words.

Simply glancing at the terrain graphs, it appears that Ramsey-based complementation is competitive with rank-based complementation for automata of size 6. Despite the massive gap in worst-case complexity, when the transition density is high and the acceptance density is low the Ramsey-based approach produces smaller automata for this input size. In other cases, the rank-based approach produces smaller automata. We also see that neither construction approaches the worst case behavior. This provides the first hint that the two approaches are not directly comparable, a fact that we will explore in more depth in the next chapter. We did observe a higher variance in the Ramsey-based approach, and the worst-case for Ramsey can be very bad indeed. The largest rank-based complementary automaton had less than 1500 states, while there were 117 Ramsey-based complementary automata with over 10,000 states.

We hasten to note that these results should not be used to indicate that one approach *scales* better than the other. We can only conclude that the traits that make a problem hard in the Ramsey-based approach are different from those traits that make a problem hard in the rank-based approach. To compare the scalability of the approaches, we would have to test the complementation constructions on automata of increasing size and compare the number of states in each construction as a function of the number of states in the input. However, such a scaling experiment is not feasible here. Smaller automata are already uninteresting, and in experiments on automata of size 10 the Ramsey-based approach was so slow to complement it had to be removed from the study entirely [TFVT11]. Thus we cannot generate enough data points to make a scaling experiment worthwhile.

The real conclusion of these experiments is then that, when checking if a Büchi automaton $\mathcal{A}$ is contained in a Büchi automaton $\mathcal{B}$, it is not feasible to explicitly construct the complement of $\mathcal{B}$. With input automata of six states, are are already seeing results with hundreds to thousands of states. If we wish to scale to input automata with fifty or a hundred states, the cost of direct complementation is prohibitive. To develop a system that allows Büchi specifications, we must instead develop algorithms that explore the complemented automata on the fly, and that avoid searching as much of the state space as possible.

# Chapter 4

# Rank and Ramsey-Based Containment

As we saw in Chapter 3.2, no approach to complementation scaled well: we could handle around ten states. By avoiding direct complementation, we can scale orders of magnitude larger. To do so, we develop algorithms that explore complemented automata on-the-fly, often using heuristics to trim the search space. In this chapter, we present a rank-based algorithm from Doyen and Raskin, which uses the Emerson-Lei fixpoint and a subsumption heuristic. We then develop a new Ramsey-based algorithm and subsumption heuristic.

## 4.1 Preliminaries: Rank-Based Containment Checking

The simplest form of containment checking is universality. An algorithm seeking to refute the universality of $\mathcal{B}$ can look for a lasso in the state-space of the rank-based complement of $\mathcal{B}$. A classical approach is Emerson-Lei backward-traversal nested fixpoint. Given a set of states $X \subset Q_\mathcal{B}$, we define the predecessor operation as $Pre_\mathcal{B}(X) = \{y \mid x \in X,\ a \in \Sigma,\ x \in \rho_\mathcal{B}(y, a)\}$. The Emerson-Lei universality testing fixpoint is $\nu Y.\mu X.(Pre_\mathcal{B}(X) \cup (Pre_\mathcal{B}(Y) \cap F_\mathcal{B}))$ [EL86]. This nested fixpoint employs the observation that a state in a lasso can reach an arbitrary number of accepting states. The outer fixpoint iteratively computes sets $Y_0, Y_1, \ldots$ such that $Y_i$ contains all states with a path to $i$ accepting states. Universality is checked by testing if $Y_\infty$, the set of all states with a

This chapter contains work published in [FV10].

path to arbitrarily many accepting states, intersects $Q^{in}_{\mathcal{B}}$. The running time of this algorithm is $O(n^2)$, based on the nested fix points. While theoretically quicker algorithms exist, the Emerson-Lei approach is popular because it requires few operations: intersection, union, and the predecessor operation. The strongest algorithm implementing this approach, from Doyen and Raskin, takes advantage of the presence of a subsumption relation in the rank-based construction.

**Definition 4.1.** For an NBW $\mathcal{B}$ and its rank-based complement $C^R_m(\mathcal{B})$, define the relation $\prec \ \subseteq \ (\mathcal{R}^m \times 2^{Q_{\mathcal{B}}}) \times (\mathcal{R}^m \times 2^{Q_{\mathcal{B}}})$ so that $\langle f', o'\rangle \preceq \langle f, o\rangle$ iff:

(1) For every $q \in Q_{\mathcal{B}}$, if $f(q) \neq \bot$, then $f'(q) \neq \bot$ and $f'(q') \leq f(q)$

(2) $o' \subseteq o$

(3) $o = \emptyset$ iff $o' = \emptyset$

When computing sets in the Emerson-Lei approach, it is sufficient to store only the minimal elements under this relation. Let `Min` be an function that removes all non-minimal elements from a set. Furthermore, the predecessor operation for a single state and letter results in at most two incomparable elements. For details, see [DR07]. With this predecessor operation `Pre` in hand, we can define a universality testing algorithm. This algorithm has scaled to automata an order of magnitude larger than other approaches [DR07].

Algorithm 4.1 can terminate early on some universal cases, when we have already computed that no state can reach $k$ accepting states. To lift this algorithm to check the containment of $\mathcal{A}$ in $\mathcal{B}$, we operate over pairs $\langle q, \langle f, o\rangle\rangle$ where $q \in Q_{\mathcal{A}}$ and $\langle f, o\rangle$ is a state in $C^R_m(\mathcal{B})$. These pairs correspond to states in the product of $\mathcal{A}$ and $C^R_m(\mathcal{B})$. Lifting subsumption to these pairs is simple: say that $\langle q', \langle f', o'\rangle\rangle \preceq \langle q, \langle f, o\rangle\rangle$ when $q' = q$ and $\langle f', o'\rangle \preceq \langle f, o\rangle$. The predecessor operation is similar. For a pair $\langle q, \langle f, o\rangle\rangle$, let $\texttt{Pre}(\langle q, \langle f, o\rangle\rangle, a) = \{r \mid q \in \rho_{\mathcal{A}}(r, a)\} \times \texttt{Pre}(\langle f, o\rangle, a)$. Finally, we define two sets of

**Algorithm 4.1:** `RankUniversality`($\mathcal{B}$)

```
$\langle \Sigma, Q_R, Q_R^{in}, \rho_R, F_R \rangle \Leftarrow (C_m^R(\mathcal{B}))$
$\mathsf{Y} \Leftarrow F_R$
$\mathsf{X} \Leftarrow \emptyset$
repeat
    if $\mathsf{Y} \cap Q_R^{in} = \emptyset$ then return Universal
    $\mathsf{X} \Leftarrow \mathtt{Min}(\mathsf{Y} \cap F_R)$
    repeat
        $\mathsf{Preds} \Leftarrow \bigcup_{\langle f,o \rangle \in \mathsf{X}} \bigcup_{a \in \Sigma} \mathtt{Pre}(\langle f, o \rangle, a)$
        $\mathsf{X} \Leftarrow \mathtt{Min}(\mathsf{X} \cup \mathsf{Preds})$
    until $\mathsf{X}$ reaches fixpoint
    $\mathsf{Y} \Leftarrow \mathsf{X}$
until $\mathsf{Y}$ reaches fixpoint
if $\mathsf{Y} \cap Q_R^{in} = \emptyset$ then return Universal
return Not Universal
```

pairs corresponding to accepting states of $\mathcal{A}$ and $C_m^R(\mathcal{B})$. Let $\mathcal{F}_{\mathcal{A}} = \{\langle q, \langle f, o \rangle \rangle \mid q \in F_{\mathcal{A}}\}$, and $\mathcal{F}_R = \{\langle q, \langle f, \emptyset \rangle \rangle\}$. We can now define Algorithm 4.2, which checks containment by using the rank-based construction. This algorithm has three fixpoints, but two are run in parallel, and thus the running time is still quadratic in the number of states of $\mathcal{A}$ and the complement of $\mathcal{B}$. Like Algorithm 4.1, Algorithm 4.2 can terminate early in cases where containment can already be shown.

**Theorem 4.2.** *For every two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, it holds that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff* `RankContainment`*($\mathcal{A}$,$\mathcal{B}$) returns Contained.*

## 4.2 Ramsey-Based Algorithms

To test the containment of a Büchi automaton $\mathcal{A}$ in a Büchi automaton $\mathcal{B}$, we could construct the Ramsey-based complement of $\mathcal{B}$, compute the intersection automaton of $\mathcal{A}$ and $\overline{\mathcal{B}}$, and search this intersection automaton for a lasso. With universality, however, we avoided directly constructing $\overline{\mathcal{B}}$ by exploiting the structure of states in the Ramsey-based

**Algorithm 4.2:** `RankContainment` $(\mathcal{A}, \mathcal{B})$

```
$\langle \Sigma, Q_R, Q_R^{in}, \rho_R, F_R \rangle \Leftarrow (C_m^R(\mathcal{B}))$
$\mathsf{Y} \Leftarrow Q_{\mathcal{A}} \times Q_R$
$\mathsf{X}_1 \Leftarrow \emptyset$
$\mathsf{X}_2 \Leftarrow \emptyset$
repeat
    if $\mathsf{Y} \cap (Q_{\mathcal{A}}^{in} \times Q_R^{in}) = \emptyset$ then return Contained
    $\mathsf{X}_1 \Leftarrow \mathtt{Min}(\mathsf{Y} \cap \mathcal{F}_{\mathcal{A}})$
    repeat
        $\mathsf{Preds} \Leftarrow \bigcup_{\mathbf{q} \in \mathsf{X}_1} \bigcup_{a \in \Sigma} \mathtt{Pre}(\mathbf{q}, a)$
        $\mathsf{X}_1 \Leftarrow \mathtt{Min}(\mathsf{X}_1 \cup \mathsf{Preds})$
    until $\mathsf{X}_1$ reaches fixpoint
    $\mathsf{X}_2 \Leftarrow \mathtt{Min}(\mathsf{Y} \cap \mathcal{F}_R)$
    repeat
        $\mathsf{Preds} \Leftarrow \bigcup_{\mathbf{q} \in \mathsf{X}_2} \bigcup_{a \in \Sigma} \mathtt{Pre}(\mathbf{q}, a)$
        $\mathsf{X}_2 \Leftarrow \mathtt{Min}(\mathsf{X}_2 \cup \mathsf{Preds})$
    until $\mathsf{X}_2$ reaches fixpoint
    $\mathsf{Y} \Leftarrow \mathsf{X}_1 \cap \mathsf{X}_2$
until $\mathsf{Y}$ reaches fixpoint
if $\mathsf{Y} \cap (Q_{\mathcal{A}}^{in} \times Q_R^{in}) = \emptyset$ then return Contained
return Not Contained
```

construction (see Lemma 2.14). We demonstrate a similar test for containment. We then introduce an on-the-fly approach to containment checking, that avoids searching graphs with empty languages. Finally, we define a subsumption relation, similar that that used for rank-based containment checking.

### 4.2.1 Ramsey-Based Containment Checking

Consider two automata, $\mathcal{A} = \langle \Sigma, Q_{\mathcal{A}}, Q_{\mathcal{A}}^{in}, \rho_{\mathcal{A}}, F_{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, Q_{\mathcal{B}}^{in}, \rho_{\mathcal{B}}, F_{\mathcal{B}} \rangle$. When testing the universality of $\mathcal{B}$, any word not in $L(\mathcal{B})$ is a sufficient counterexample. To test $L(\mathcal{A}) \subseteq L(\mathcal{B})$ we must restrict our search to the subset of $\Sigma^\omega$ accepted by $\mathcal{A}$. In Section 2.2.2, we defined a set $\widetilde{Q}_{\mathcal{B}}$ of 0-1 arc-labeled graphs, whose elements provide a family of $\omega$-languages that covers $\Sigma^\omega$ (see Lemma 2.11). We now define a set, $\widehat{Q}_{\mathcal{A},\mathcal{B}}$, which provides a family of $\omega$-languages covering $L(\mathcal{A})$.

We first define $\bar{Q}_{\mathcal{A}} = Q_{\mathcal{A}} \times Q_{\mathcal{A}}$ to capture the connectivity in $Q_{\mathcal{A}}$. An element $\bar{g} = \langle q, r \rangle \in \bar{Q}_{\mathcal{A}}$ is a single arc asserting the existence of a finite run of $\mathcal{A}$ from $q$ to $r$. With each arc we associate a language, $L(\bar{g})$.

**Definition 4.3.** Given $w \in \Sigma^+$, say that $w \in L(\langle q, r \rangle)$ iff there is a finite run of $\mathcal{A}$ from $q$ to $r$ over $w$.

Define $\widehat{Q}_{\mathcal{A},\mathcal{B}}$ as $\bar{Q}_{\mathcal{A}} \times \widetilde{Q}_{\mathcal{B}}$. The elements of $\widehat{Q}_{\mathcal{A},\mathcal{B}}$, called *supergraphs*, are pairs consisting of an arc from $\bar{Q}_{\mathcal{A}}$ and a graph from $\widetilde{Q}_{\mathcal{B}}$. Each element simultaneously captures all finite runs of $\mathcal{B}$ and a single finite run of $\mathcal{A}$. The language $L(\langle \bar{g}, \widetilde{g} \rangle)$ is then $L(\bar{g}) \cap L(\widetilde{g})$. For convenience, we implicitly take $\widehat{g} = \langle \bar{g}, \widetilde{g} \rangle$, and say $\langle q, a, r \rangle \in \widehat{g}$ when $\langle q, a, r \rangle \in \widetilde{g}$. Since the language of each graph consists of finite words, we employ the concatenation of languages to characterize infinite runs. To do so, we first prove Lemma 4.4, which simplifies the concatenation of entire languages by demonstrating an equivalence to the concatenation of arbitrary words from these languages.

**Lemma 4.4.** *If* $u \in L(\widehat{g})$, $v \in L(\widehat{h})$, $uv \in L(\widehat{k})$*, and* $L(\bar{g}) \cdot L(\bar{h}) \subseteq L(\bar{k})$*, then* $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{k})$

*Proof.* Assume we have such an $u$ and $v$. We demonstrate every word $w \in L(\widehat{g}) \cdot L(\widehat{h})$ must be in $L(\widehat{k})$. If we expand the premise, we obtain $w \in (L(\bar{g}) \cap L(\widetilde{g})) \cdot (L(\bar{h}) \cap L(\widetilde{h}))$. This implies $w$ must be in $L(\bar{g}) \cdot L(\bar{h})$ and in $L(\widetilde{g}) \cdot L(\widetilde{h})$. Next, we know that $u \in L(\widetilde{g})$, $v \in L(\widetilde{h})$, and $uv \in L(\widetilde{k})$. Thus by Lemma 2.9, $L(\widetilde{g}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{k})$, and $w \in L(\widetilde{k})$. Along with the premise $L(\bar{g}) \cdot L(\bar{h}) \subseteq L(\bar{k})$, we can now conclude $w \in L(\bar{k}) \cap L(\widetilde{k})$, which is $L(\widehat{k})$. □

The languages $L(\widehat{g}),\ \widehat{g} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, cover all finite subwords of $L(\mathcal{A})$. A subword of $L(\mathcal{A})$ has at least one finite run between two states in $Q_{\mathcal{A}}$, and thus is in the language of an arc in $\bar{Q}_{\mathcal{A}}$. Furthermore, by Lemma 2.9 this word is described by some graph, and the pair of the arc and the graph makes a supergraph. Unlike the case of graphs and $\Sigma^+$, the languages of supergraphs do not form a partition of $L(\mathcal{A})$: a word might have finite runs between multiple pairs of states in $\mathcal{A}$, and so be described by more than one arc in $\bar{Q}_{\mathcal{A}}$. With them we construct the finite family of $\omega$-languages that cover $L(\mathcal{A})$. Given $\widehat{g},\ \widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, let $Z(\widehat{g}, \widehat{h})$ be the $\omega$-language $L(\widehat{g}) \cdot L(\widehat{h})^\omega$. In analogy to Section 2.2.2, we define a notion of properness for supergraphs.

**Definition 4.5.** Say a pair of supergraphs $\langle \widehat{g}, \widehat{h} \rangle$ is proper when (1) $Z(\widehat{g}, \widehat{h})$ is non-empty; (2) $\bar{g} = \langle q, r \rangle$ and $\bar{h} = \langle r, r \rangle$ where $q \in Q^{in}_{\mathcal{A}}$ and $r \in F_{\mathcal{A}}$; (3) $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{g})$ and $L(\widehat{h}) \cdot L(\widehat{h}) \subseteq L(\widehat{h})$.

Call a language $Z(\widehat{g}, \widehat{h})$ proper when $\langle \widehat{g}, \widehat{h} \rangle$ is proper. We note that $Z(\widehat{g}, \widehat{h})$ is non-empty if $L(\widehat{g})$ and $L(\widehat{h})$ are non-empty, and that, by the second condition, every proper $Z(\widehat{g}, \widehat{h})$ is contained in $L(\mathcal{A})$.

**Lemma 4.6.** $L(\mathcal{A}) = \bigcup\{Z(\widehat{g}, \widehat{h}) \mid, \widehat{g}, \widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}},\ Z(\widehat{g}, \widehat{h})$ *is proper*$\}$.

*Proof.* We extend the Ramsey argument of Lemma 2.11 to supergraphs.

Consider an infinite word $w = \sigma_0\sigma_1...$ with an accepting run $p = p_0p_1...$ in $\mathcal{A}$. As $p$ is accepting, we know that $p_0 \in Q^{in}_{\mathcal{A}}$ and $p_i \in F_{\mathcal{A}}$ for infinitely many $i$. Since $F_{\mathcal{A}}$ is finite, at least one accepting state $q$ must appear infinitely often. Let $D \subseteq \mathbb{N}$ be the set of indices $i$ such that $p_i = q$.

We pause to observe that, by the definition of the languages of arcs, for every $i \in D$ the word $\sigma_0...\sigma_{i-1}$ is in $L(\langle p_0, q \rangle)$, and for every $i, j \in D$, $i < j$, the word $\sigma_i...\sigma_{j-1} \in L(\langle q, q \rangle)$. Every language $Z(\widehat{g}, \widehat{h})$ where $\bar{g} = \langle p_0, q \rangle$ and $\bar{g} = \langle q, q \rangle$ also satisfies the second requirement of properness.

In addition to restricting our attention the subset of nodes where $p_i = q$, we further partition $D$ into $k = 3^{n^2}$ sets $D_1, ..., D_k$ based on the prefix of $w$ until that point, where there is a $D_l$ associated with each possible graph $\widetilde{g}_l$. By Lemma 2.9, every finite word is in the language of some graph $\widetilde{g}$. Say that $i \in D_l$ iff $\sigma_0...\sigma_{i-1} \in L(\widetilde{g}_l)$. As $k$ is finite, for some $m$ the $D_m$ must be infinite. Let $\widetilde{g} = \widetilde{g}_m$.

Similarly, by Lemma 2.9 we can use the word $w$ to define a partition of all unordered *pairs* of elements from $D_m$. This partition consists of $k$ sets $C_1, ...C_k$, such that $(i, j) \in C_l$ iff $\sigma_i...\sigma_{j-1} \in L(\widetilde{g}_l)$. Without loss of generality, for $(i, j) \in C_l$, assume $i < j$. Ramsey's Theorem tells us that, given such a partition, there exists an infinite subset $\{i_1, i_2, ...\}$ of $D_m$ and a $C_n$ such that $(i_j, i_k) \in C_n$ for all pairs of distinct elements $i_j, i_k$.

This is precisely to say there is a graph $\widetilde{h}$ so that, for every $(i_j, i_k) \in C_n$, it holds that $\sigma_{i_j}...\sigma_{i_k-1} \in L(\widetilde{h})$. $C_n$ thus partitions the word $w$ into

$$w_0 = \sigma_0...\sigma_{i_1-1}, \quad w_1 = \sigma_{i_1}...\sigma_{i_2-1}, \quad w_2 = \sigma_{i_2}...\sigma_{i_3-1}, \quad ...,$$

such that $w_0 \in L(\widetilde{g})$ and $w_i \in L(\widetilde{h})$ for $i > 0$. Let $\widehat{g} = \langle\langle p_0, q\rangle, \widetilde{g}\rangle$ and let $\widehat{h} = \langle\langle q, q\rangle, \widetilde{h}\rangle$. By the above partition of $w$, we know that $w \in Z(\widehat{g}, \widehat{h})$.

We now show that $Z(\widehat{g}, \widehat{h})$ is proper. First, as $w \in Z(\widehat{g}, \widehat{h})$, we know $Z(\widehat{g}, \widehat{h})$ is non-empty. Second, as noted above, the second requirement is satisfied by the arcs $\langle p_0, q\rangle$ and $\langle q, q\rangle$. Finally, we demonstrate the third condition holds. As $\sigma_0...\sigma_{i-1} \in L(\widetilde{g})$ for every $i \in C_n$, we have that $w_0w_1 \in L(\widetilde{g})$. Both $w_0$ and $w_0w_1$ are in $L(\langle p_0, q\rangle)$ and so $w_0,\ w_0w_1 \in L(\widehat{g})$. By the definition of the language of arcs, $L(\langle p_0, q\rangle) \cdot L(\langle q, q\rangle) \subseteq L(\langle p_0, q\rangle)$. Thus by Lemma 4.4, we can conclude that $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{g})$. Next observe that as $\sigma_i...\sigma_{j-1} \in L(\widetilde{h})$ for every pair $i, j \in C_n$, we have that $w_1w_2 \in L(\widetilde{h})$. As $w_1,\ w_1w_2$ are both in $\langle q, q\rangle$, it holds that $w_1,\ w_1w_2 \in L(\widehat{h})$. By the definition of the language of arcs, $L(\langle q, q\rangle) \cdot L(\langle q, q\rangle) \subseteq L(\langle q, q\rangle)$. By Lemma 4.4 we can now conclude

$L(\widehat{h}) \cdot L(\widehat{h}) \subseteq L(\widehat{h})$. Therefore $Z(\widehat{g}, \widehat{h})$ is a proper language containing $w$. $\square$

**Lemma 4.7.**

*(1) For all $Z(\widehat{g}, \widehat{h})$, either $Z(\widehat{g}, \widehat{h}) \cap L(\mathcal{B}) = \emptyset$ or $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$.*

*(2) $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff every proper language $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$.*

*(3) Let $\widehat{g}, \widehat{h}$ be a proper pair of supergraphs such that. $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$ iff there exists $q \in Q^{in}_{\mathcal{B}},\ r \in Q_{\mathcal{B}},\ a \in \{0,1\}$ such that $\langle q, a, r\rangle \in \widehat{g}$ and $\langle r, 1, r\rangle \in \widehat{h}$.*

*Proof.* Given two supergraphs $\widehat{g} = \langle \bar{g}, \widetilde{g}\rangle$ and $\widehat{h} = \langle \bar{h}, \widetilde{h}\rangle$, recall that $Y(\widetilde{g}, \widetilde{h})$ is the $\omega$-language $L(\widetilde{g}) \cdot L(\widetilde{h})^{\omega}$. Further note that $L(\widehat{g}) \subseteq L(\widetilde{g})$ and $L(\widehat{h}) \subseteq L(\widetilde{h})$, and therefore $Z(\widehat{g}, \widehat{h}) \subseteq Y(\widetilde{g}, \widetilde{h})$.

(1) Consider two supergraphs $\widehat{g}, \widehat{h}$. By Lemma 2.12 either $Y(\widetilde{g}, \widetilde{h}) \cap L(\mathcal{B}) = \emptyset$ or $Y(\widetilde{g}, \widetilde{h}) \subseteq L(\mathcal{B})$. Since $Z(\widehat{g}, \widehat{h}) \subseteq Y(\widetilde{g}, \widetilde{h})$, it holds that $Z(\widehat{g}, \widehat{h}) \cap L(\mathcal{B}) = \emptyset$ or $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$.

(2) Immediate from Lemma 4.6 and clause (1).

(3) By Lemma 2.12 either $Y(\widetilde{g}, \widetilde{h}) \subseteq L(\mathcal{B})$ or $Y(\widetilde{g}, \widetilde{h}) \cap L(\mathcal{B}) = \emptyset$. By Lemma 2.14 $Y(\widetilde{g}, \widetilde{h}) \subseteq L(\mathcal{B})$ iff a $q, r$ and $a$ exist such that $\langle q, a, r\rangle \in \widetilde{g}$ and $\langle r, 1, r\rangle \in \widetilde{h}$. Since $Z(\widehat{g}, \widehat{h}) \subseteq Y(\widetilde{g}, \widetilde{h})$, $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$ iff such a $q, r$ and $a$ exist.

$\square$

In an analogous fashion to Section 2.2.2, we can use supergraphs to test the containment of two automata, $\mathcal{A}$ and $\mathcal{B}$. Search all pairs of supergraphs, $\widehat{g}, \widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$ for a pair that is both proper and for which there does not exist a $q \in Q^{in}_{\mathcal{B}}$, $r \in Q_{\mathcal{B}}, a \in \{0,1\}$ such that $\langle q, a, r\rangle \in \widehat{g}$ and $\langle r, 1, r\rangle \in \widehat{h}$. Such a pair is a counterexample to containment. If no such pair exists, then $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

### 4.2.2 Composition of Supergraphs

Employing supergraphs to test containment faces difficulty on two fronts. First, the number of supergraphs is very large: $|Q_{\mathcal{A}}|^2 3^{|Q_{\mathcal{B}}|^2}$. Second, verifying properness requires checking language nonemptiness and containment: PSPACE-hard problems. To address these problems we construct only supergraphs with non-empty languages. Borrowing a notion of composition from [LJBA01] allows us to use exponential space to compute exactly the needed supergraphs. Along the way we develop a polynomial-time test for the containment of supergraph languages. Our plan is to start with graphs corresponding to single letters and compose them until we reach closure. The resulting subset of $\widehat{Q}_{\mathcal{A},\mathcal{B}}$, written $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$, contains exactly the supergraphs with non-empty languages. In addition to removing the need to check for emptiness, composition allows us to test the sole remaining aspect of properness, language containment, in time polynomial in the size of the supergraphs. We begin by defining the composition of simple graphs.

**Definition 4.8.** Given $\widetilde{g}$ and $\widetilde{h}$ define their composition, written $\widetilde{g}; \widetilde{h}$, as the graph

$$\begin{aligned} & \{\langle q,1,r\rangle \mid q,r,s \in Q_{\mathcal{B}},\ \langle q,b,s\rangle \in \widetilde{g},\ \langle s,c,r\rangle \in \widetilde{h},\ b=1 \text{ or } c=1\} \\ \cup\ & \{\langle q,0,r\rangle \mid q,r,s \in Q_{\mathcal{B}},\ \langle q,0,s\rangle \in \widetilde{g},\ \langle s,0,r\rangle \in \widetilde{h},\ \text{and} \\ & \text{for all } t \in Q_{\mathcal{B}},\ b,c \in \{0,1\} \text{ if } \langle q,a,t\rangle \in \widetilde{g} \text{ and } \langle t,b,r\rangle \in \widetilde{h} \text{ then } a=b=0\} \end{aligned}$$

*Example* 4.9. Figure 4.1 shows the composition of a graph with itself. These graphs do not describe the automaton from Figure 2.1, but demonstrate new behavior. Note that there are ways to compose an arc from $p$ to $s$: by following either $\langle p,0,r\rangle$ and $\langle r,0,s\rangle$, or by following $\langle p,1,q\rangle$ and $\langle q,0,s\rangle$. Since the second pair of arcs contains a 1-labeled arc, the arc from $p$ to $s$ in the composition is 1-labeled. However, note there is only one pair of arcs linking $q$ to $s$: $\langle q,0,r\rangle$ and $\langle r,0,s\rangle$. Therefore the arc from $q$ to $s$ in

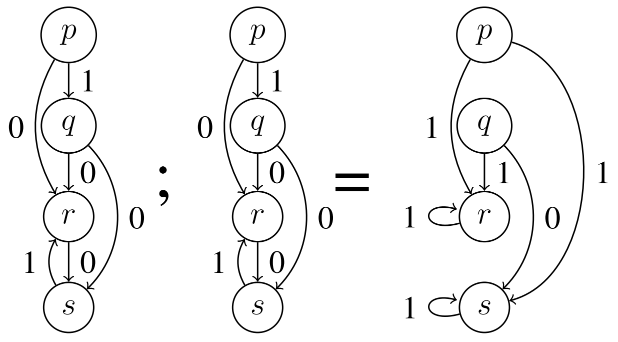


**Figure 4.1 :** The composition of a graph with itself.

the composition is not $0$ labeled. Figure 2.3 is also illustrative, as the third graph is the composition of the first two.

We can then define the composition of two supergraphs $\widehat{g} = \langle\langle q, r\rangle, \widetilde{g}\rangle$ and $\widehat{h} = \langle\langle r, s\rangle, \widetilde{h}\rangle$, written $\widehat{g}; \widehat{h}$, as the supergraph $\langle\langle q, s\rangle, \widetilde{g}; \widetilde{h}\rangle$. To generate exactly the set of supergraphs with non-empty languages, we start with supergraphs describing single letters. For a containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$, define the subset of $\widehat{Q}_{\mathcal{A},\mathcal{B}}$ corresponding to single letters to be $\widehat{Q}^1_{\mathcal{A},\mathcal{B}} = \{\widehat{g} \mid \widehat{g} \in \widehat{Q}_{\mathcal{A},\mathcal{B}},\ a \in \Sigma,\ a \in L(\widehat{g})\}$. For completeness, we present a constructive definition of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$.

**Definition 4.10.**

$$\begin{aligned}\widehat{Q}^1_{\mathcal{A},\mathcal{B}} = \big\{\langle\langle q, r\rangle, \widetilde{g}\rangle \mid\ & a \in \Sigma,\ q \in Q_{\mathcal{A}},\ r \in \rho_{\mathcal{A}}(q, a),\\ & \widetilde{g} = \ \{\langle q', 0, r'\rangle \mid q' \in Q_{\mathcal{B}} \setminus F_{\mathcal{B}},\ r' \in (\rho_{\mathcal{B}}(q', a) \setminus F_{\mathcal{B}})\} \cup\\ & \qquad \{\langle q', 1, r'\rangle \mid q' \in Q_{\mathcal{B}},\ r' \in \rho_{\mathcal{B}}(q', a),\ q' \text{ or } r' \in F_{\mathcal{B}})\}\big\}\end{aligned}$$

We then define $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ to be the closure of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ under composition. Algorithm `RamseyContainment`, which we prove correct below, employs composition to check the containment of two automata. It first generates the set of initial supergraphs, and then computes the closure of this set under composition. Along the way it tests properness by using composition. Every time it encounters a proper pair of supergraphs, it either verifies that a satisfying pair of arcs exist, or halts with a counterexample to containment. For convenience, we define the two-arc test for supergraphs.

**Definition 4.11.** A proper pair $\langle \widehat{g}, \widehat{h} \rangle$ of supergraphs passes the two-arc test when there exists a $q \in Q^{in}$, $r \in Q$, and $a \in \{0, 1\}$ such that $\langle q, a, r \rangle \in \widehat{g}$ and $\langle r, 1, r \rangle \in \widehat{h}$.

**Algorithm 4.3:** `RamseyContainment(`$\mathcal{A}$`,`$\mathcal{B}$`)`

```
Data: Two Büchi automata, $\mathcal{A}$ and $\mathcal{B}$.
Result: Whether $L(\mathcal{A})$ is contained in $L(\mathcal{B})$.
Initialize $\widehat{Q}^f \Leftarrow \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$
repeat
    for all pairs $\widehat{g}, \widehat{h} \in \widehat{Q}^f$ where $\bar{g} = \langle q, r \rangle$ and $\bar{h} = \langle r, s \rangle$ do
        Add $\widehat{g}; \widehat{h}$ to $\widehat{Q}^f$
        if $q \in Q^{in}_{\mathcal{A}}$, $r \in F_{\mathcal{A}}$, $s = r$, $\widetilde{g}; \widetilde{h} = \widetilde{g}$ and $\widetilde{h}; \widetilde{h} = \widetilde{h}$ then
            if $\langle \widehat{g}, \widehat{h} \rangle$ fails the two-arc test then
                return Not Contained
until $\widehat{Q}^f$ reaches closure
return Contained
```

To begin proving our algorithm correct, we link composition and the concatenation of languages, first for simple graphs and then for supergraphs.

**Lemma 4.12.** *For every two graphs $\widetilde{g}$ and $\widetilde{h}$, it holds that $L(\widetilde{g}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{g}; \widetilde{h})$.*

*Proof.* Consider two words $w_1 \in L(\widetilde{g}),\ w_2 \in L(\widetilde{h})$. By Definition 2.7, to prove $w_1 w_2 \in L(\widetilde{g}; \widetilde{h})$ we must show that for every $q, r \in Q$: both (1) $\langle q, a, r \rangle \in \widetilde{g}; \widetilde{h}$ iff there is a finite run of $\mathcal{B}$ from $q$ to $r$ over $w_1 w_2$, and (2) that $a = 1$ iff there is an accepting finite run.

If an arc $\langle q, a, r \rangle \in \widetilde{g}; \widetilde{h}$ exists, then there is an $s \in Q$ such that $\langle q, b, s \rangle \in \widetilde{g}$ and $\langle s, c, r \rangle \in \widetilde{h}$. By Definition 2.7, this implies the existence of a finite run $x_1 s$ from $q$ to $s$

over $w_1$, and a finite run $sx_2$ from $s$ to $r$ over $w_2$. Thus $x_1sx_2$ is a finite run from $q$ to $r$ over $w_1w_2$.

If $a$ is 1, then either $b$ or $c$ must be 1. By Definition 2.7, $b$ (resp., $c$) is 1 iff there is an accepting finite run $x'_1s$ (resp., $sx'_2$) over $w_1$ (resp.,$w_2$) from $q$ to $s$ (resp., $s$ to $r$). In this case $x'_1sx_2$ (resp., $x_1sx'_2$) is an accepting finite run of $\mathcal{B}$ from $q$ to $r$ over $w_1w_2$.

Symmetrically, if there is a finite run $x$ from $q$ to $r$ over $w_1w_2$, then after reading $w_1$ we are in some state $s$ and have split $x$ into $x_1sx_2$, so that $x_1s$ is a finite run from $q$ to $s$ and $sx_2$ a finite run from $s$ to $r$. Thus by Definition 2.7 $\langle q, b, s\rangle \in \widetilde{g}$, $\langle s, c, r\rangle \in \widetilde{h}$, and $\langle q, a, s\rangle \in \widetilde{g}; \widetilde{h}$.

Furthermore, if there is an accepting finite run from $q$ to $r$ over $w_1w_2$, then after reading $w_1$ we are in some state $s$ and have split the finite run into $x_1sx_2$, so that $x_1s$ is a finite run from $q$ to $s$, and $sx_2$ a finite run from $s$ to $r$. Either $x_1s$ or $sx_2$ must be accepting, and thus by Definition 2.7 $\langle q, b, s\rangle \in \widetilde{g}$, $\langle s, c, r\rangle \in \widetilde{h}$, and either $b$ or $c$ must be 1. Therefore $a$ must be 1. □

**Lemma 4.13.** *Let $\widehat{g}, \widehat{h}, \widehat{k}$ be supergraphs in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ such that $\bar{g} = \langle q, r\rangle$, $\bar{h} = \langle r, s\rangle$, and $\bar{k} = \langle q, s\rangle$. Then $\widehat{g}; \widehat{h} = \widehat{k}$ iff $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{k})$.*

*Proof.* Assume $\widehat{g}; \widehat{h} = \widehat{k}$ as a premise. This implies $\widehat{k} = \langle\langle q, s\rangle, \widetilde{g}; \widetilde{h}\rangle)$. If either $L(\widehat{g})$ or $L(\widehat{h})$ are empty, then $L(\widehat{g}) \cdot L(\widehat{h})$ is empty and this direction holds trivially. Otherwise, take two words $u \in L(\widehat{g}),\ v \in L(\widehat{h})$. By construction, $u \in L(\langle q, r\rangle)$ and $v \in L(\langle r, s\rangle)$. The definition of the languages of arcs therefore implies the existence of a finite run of $\mathcal{B}$ from $q$ to $r$ over $u$ and a finite run from $r$ to $s$ over $v$. Thus $uv \in L(\langle q, s\rangle)$. Similarly, $u \in L(\widetilde{g}),\ v \in L(\widetilde{h})$, and Lemma 4.12 implies that $uv \in L(\widetilde{k})$. Thus $uv$ is in $L(\langle\langle q, r\rangle, \widetilde{k}\rangle)$ and by Lemma 4.4 $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{k})$.

In the other direction, if $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{k})$, we show that $\widehat{g}; \widehat{h} = \widehat{k}$. By definition, $\widehat{g}; \widehat{h}$ is $\langle\langle q, s\rangle, \widetilde{g}; \widetilde{h}\rangle$. As $\widehat{g}, \widehat{h} \in \widehat{Q}^f_{\mathcal{A},\mathcal{B}}$, they are the composition of a finite number of graphs from $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$. The above direction then demonstrates that they are non-empty, and there is a word $w \in L(\widehat{g}) \cdot L(\widehat{h})$. This expands to $w \in (L(\bar{g}) \cap L(\widetilde{g})) \cdot (L(\bar{h}) \cap L(\widetilde{h}))$, which implies $w \in L(\widetilde{g}) \cdot L(\widetilde{h})$. By Lemma 4.12, $w$ is then in $L(\widetilde{g}; \widetilde{h})$. Since, by Lemma 2.9, $w$ is the language of exactly one graph, we have that $\widetilde{g}; \widetilde{h} = \widetilde{k}$, which proves $\widehat{g}; \widehat{h} = \widehat{k}$.

□

Lemma 4.13 provides the polynomial time test for properness employed in Algorithm `RamseyContainment`. Namely, given two supergraphs $\widehat{g} = \langle \langle q, r \rangle, \widetilde{g} \rangle$ and $\widehat{h} = \langle \langle r, r \rangle, \widetilde{h} \rangle$ from $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$, the pair $\langle \widehat{g}, \widehat{h} \rangle$ is proper exactly when $q \in Q^{in}_{\mathcal{A}}$, $r \in F_{\mathcal{A}}$, $\widetilde{g}; \widetilde{h} = \widetilde{g}$ and $\widetilde{h}; \widetilde{h} = \widetilde{h}$. We now provide the final piece of our puzzle: proving that the closure of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ under composition contains every non-empty supergraph.

**Lemma 4.14.** *For two NBW $\mathcal{A}$ and $\mathcal{B}$, every $\widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, where $L(\widehat{h}) \neq \emptyset$, is in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$.*

*Proof.* Let $\widehat{h} = \langle \langle q, r \rangle, \widetilde{h} \rangle$ where $L(\widehat{h}) \neq \emptyset$. Then there is at least one word $w = \sigma_0 \ldots \sigma_{n-1} \in L(\widehat{h})$, which is to say $w \in L(\langle q, r \rangle) \cap L(\widetilde{h})$. By the definition of the languages of arcs, there is a finite run $p = p_0 \ldots p_n$ in $\mathcal{A}$ over $w$ such that $p_0 = q$ and $p_n = r$.

Define $\widetilde{g}_{\sigma_i}$ to be the graph in $\widetilde{Q}^1_{\mathcal{B}}$ containing $\sigma_i$. Let $\widehat{g}_{\sigma_i}$ be $\langle \langle p_i, p_{i+1} \rangle, \widetilde{g}_{\sigma_i} \rangle$, and let $\widehat{g}_w$ be $\widehat{g}_{\sigma_0}; \widehat{g}_{\sigma_1}; \ldots; \widehat{g}_{\sigma_{n-1}}$. Note that each $\widehat{g}_{\sigma_i} \in \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$. By Lemma 4.13 $w \in \widetilde{g}_w$. By Lemma 2.9, $w$ is in only one graph and $\widetilde{g}_w = \widetilde{h}$. By construction, $\bar{g}_w = \langle q, r \rangle$. Therefore $\langle \bar{g}_w, \widetilde{g}_w \rangle = \langle \langle q, r \rangle, \widetilde{h} \rangle = \widehat{h}$, and $\widehat{h}$ is in the closure of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ under composition. □

We can now show the correctness of Algorithm `RamseyContainment`, using Lemma 4.13 to justify testing properness with composition, and Lemma 4.15 below to justify the correctness and completeness of our search for a counterexample.

**Lemma 4.15.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two Büchi automata. $L(\mathcal{A})$ is* not *contained in $L(\mathcal{B})$ iff $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a pair of supergraphs $\widehat{g}$, $\widehat{h}$ such that $\langle \widehat{g}, \widehat{h} \rangle$ is proper and there do* not *exist arcs $\langle q, a, r \rangle \in \widehat{g}$ and $\langle r, 1, r \rangle \in \widehat{h}$, $q \in Q^{in}_{\mathcal{B}}$.*

*Proof.* As all proper graphs are non-empty, this follows Lemma 4.7 parts (2) and (3) and Lemma 4.14. □

**Theorem 4.16.** *For every two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, it holds that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff* `RamseyContainment` *($\mathcal{A}$,$\mathcal{B}$) returns Contained.*

*Proof.* By Lemma 4.13, testing for composition is equivalent to testing for language containment, and the outer conditional in Algorithm `RamseyContainment` holds only for proper pairs of supergraphs. By Lemma 4.15, the inner conditional checks if a proper pair of supergraphs is a counterexample, and if no such proper pair in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ is a counterexample then containment must hold. □

### 4.2.3 Subsumption in Ramsey-Based Containment Checking

Subsumption has proven to be very effective in the rank-based approach [DR09]. In [FV09], we demonstrated that the Lee, Jones, and Ben-Amram algorithm for size-change termination is a specialized realization of the Ramsey-based containment test. In [BAL07], Ben-Amram and Lee demonstrated the effectiveness of subsumption for this specialized algorithm. We now show how to employ Ben-Amram and Lee's subsumption relation for the general case of Büchi universality. Doing so allows us to ignore graphs when they are approximated by other graphs.

In the special case of size-change termination, Ben-Amram and Lee replaced a test for an arc in idempotent graphs with a test for strongly-connected components in all graphs. To use subsumption in the general Ramsey-based approach, we need to replace the two-arc test over proper pairs of supergraphs. We simplify Algorithm 4.3 by relaxing the requirement that pairs of supergraphs be proper. First, we introduce the subsumption relation. Then, we demonstrate how to change the algorithm to allow for discarding subsumed supergraphs. Finally, we present a proof of correctness.

Intuitively, a graph $\widetilde{g}$ approximates another graph $\widetilde{h}$ when the arcs of $\widetilde{g}$ are a subset of, or less strict than, the arcs of $\widetilde{h}$. In this case, finding an arc in $\widetilde{g}$ is strictly harder than finding one in than $\widetilde{h}$. When the right arc can be found in $\widetilde{g}$, then it also occurs in $\widetilde{h}$. When $\widetilde{g}$ does not have a satisfying arc, then we already have a counterexample. Thus we need not consider $\widetilde{h}$.

**Definition 4.17.**

(1) For two $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$, say that $\widetilde{g}$ *approximates* $\widetilde{h}$, written $\widetilde{g} \preceq \widetilde{h}$, when for every arc $\langle q, a, r\rangle \in \widetilde{g}$ there is an arc $\langle q, a', r\rangle \in \widetilde{h}$ where $a \leq a'$.

(2) For two supergraphs $\widehat{g}, \widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, say that $\widehat{g} \preceq \widehat{h}$ when $\bar{g} = \bar{h}$ and $\widetilde{g} \preceq \widetilde{h}$.

We note that approximation is a transitive relation. Using the notion of approximation, we can compute a subset of $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$, called $\widehat{Q}^{\preceq}$, that contains only minimal elements under the $\preceq$ relation. A set $\widehat{Q}$ of supergraphs is *$\preceq$-closed under composition* when for every $\widehat{g}, \widehat{h} \in \widehat{Q}$, there exists $\widehat{k} \in \widehat{Q}$ such that $\widehat{k} \preceq \widehat{g}; \widehat{h}$.

However, once we limit our search to supergraphs in $\widehat{Q}^{\preceq}$ the two-arc test is no longer sufficient. Since we are now removing elements from $\widehat{Q}^{\preceq}$, it is possibly that the proper pair of supergraphs in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ that fails the two-arc test may never be computed: a graph in the pair may be approximated by another graph, one that does not satisfy the conditions of properness. As an example, consider the set containing the single supergraph $\widehat{g} = \langle\langle s, s\rangle, \{\langle q, 0, q\rangle, \langle q, 0, r\rangle, \langle r, 0, q\rangle\}\rangle$ when $s \in Q^{in}_{\mathcal{A}}$ and $s \in F_{\mathcal{A}}$. This set is $\preceq$-closed under composition, but $\langle \widehat{g}, \widehat{g}\rangle$ is not proper. We must relax our notion of properness.

Recall that a pair $\langle \widehat{g}, \widehat{h}\rangle$ of supergraphs is proper when three conditions hold. First,

$L(\widehat{g})$ and $L(\widehat{h})$ must be nonempty, Second, it must hold that $\bar{g} = \langle q, r\rangle$ and $\bar{h} = \langle r, r\rangle$ where $q \in Q^{in}_{\mathcal{A}}$ and $r \in F_{\mathcal{A}}$. Finally, it must hold that $\widehat{g}; \widehat{h} = \widehat{g}$ and $\widehat{h}; \widehat{h} = \widehat{h}$. It is this third requirement that we remove.

**Definition 4.18.** A pair $\langle \widehat{g}, \widehat{h}\rangle$ of supergraphs is *weakly proper* when $L(\widehat{g}) \neq \emptyset$, $L(\widehat{h}) \neq \emptyset$, and $\bar{g} = \langle q, r\rangle$ and $\bar{h} = \langle r, r\rangle$ where $q \in Q^{in}_{\mathcal{A}}$ and $r \in F_{\mathcal{A}}$.

When examining a proper pair of supergraphs, we used the two-arc test: search for a $q \in Q^{in},\ r \in Q,\ a \in \{0, 1\}$ such that $\langle q, a, r\rangle \in \widehat{g}$ and $\langle r, 1, r\rangle \in \widehat{h}$. When examining a weakly proper pair of supergraphs, we cannot limit our search to single arcs. Consider the supergraph $\widehat{g} = \langle \langle s, s\rangle, \{\langle q, 1, r\rangle, \langle r, 1, q\rangle\}\rangle$: while this supergraph does represent an accepting finite run in $\mathcal{B}$, note that $\langle \widehat{g}, \widehat{g}\rangle$ fails the two-arc test We must search the graph for a path from $q$ to $r$, and a path from $r$ to itself. We test for this path by computing the strongly connected components of $\widehat{h}$, and testing if some strongly connected component of $\widehat{h}$ both contains a 1-labeled arc and is reachable from a start state in $\widehat{g}$.

A *strongly connected component* (SCC) of a graph $\widetilde{g}$ is a maximal set $S$ of nodes, so that for every $q, r \in S$ there is a path from $q$ to $r$, and a path from $r$ to $q$. Computing the strongly connected components of a graph can be done in linear time with a depth-first search [CLR90]. An SCC $S$ in a graph $\widetilde{g}$ is 1-labeled when there are $q, r \in S$ with an arc $\langle q, 1, r\rangle \in \widetilde{g}$. We say there is a path from a state $q$ to an SCC $S$ when there is a path from $q$ to an element of $S$. Say that $S$ is an SCC of a supergraph $\widehat{g} = \langle \bar{g}, \widetilde{g}\rangle$ when $S$ is an SCC of $\widetilde{g}$. Once we partition the nodes into strongly connected components, we can simply search for a reachable 1-labeled SCC. We codify this as the lasso-finding test. Lemma 4.20 is proven correct later.

**Definition 4.19.** A weakly proper pair $\langle \widehat{g}, \widehat{h}\rangle$ of supergraphs passes the *lasso-finding test* when there exists: $q \in Q^{in}_{\mathcal{B}}$, $r \in Q_{\mathcal{B}}$, $a \in \{0, 1\}$, and $S \subseteq Q_{\mathcal{B}}$ such that, $\langle q, a, r\rangle \in \widehat{g}$, there is a path from $r$ to $S$ in $\widehat{h}$, and $S$ is a 1-labeled SCC of $\widehat{h}$.

**Lemma 4.20.** *$L(\mathcal{A}) \not\subseteq L(\mathcal{B})$ iff $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a weakly proper pair $\langle \widehat{g}, \widehat{h} \rangle$ that fails the lasso-finding test.*

We now have the machinery for Algorithm 4.4. This algorithm extends Algorithm 4.3 to exploit subsumption and avoid computing the entirety of $\widetilde{Q}^f_{\mathcal{B}}$. Along the way, we look for a counterexample to containment by checking every weakly proper pair of supergraphs with the lasso-finding test. To make the algorithm more concrete, a worklist is used to keep track of which graphs have yet to be considered. Further, instead of composing arbitrary pairs of graphs, we compose each graph only with graphs from $\widetilde{Q}^1_{\mathcal{B}}$. Since every composition can be phrased as a sequence of compositions of graphs from $\widetilde{Q}^1_{\mathcal{B}}$, this is sufficient to generate the entirety of $\widetilde{Q}^{\preceq}_{\mathcal{B}}$ while reducing the size of the worklist considerably.

To achieve reasonable performance, our implementation utilizes two optimizations not detailed in Algorithm 4.4. First, we memoize the strongly connected components of graphs. More explicitly, we associate each graph with the set of states that lead to 1-labeled SCCs, and the set of states reachable from a state in $Q^{in}_{\mathcal{B}}$. This allows us to implement the lasso-finding test by checking for an intersection between the sets. While both computing SCCs and testing intersection are linear-time operations, we found the constant-time speedup to be an order of magnitude. Second, note that we iterate through supergraphs based on their arcs. Thus an effective further optimization is to index supergraphs by the two elements of their arcs. This can improve the running time by a factor of $|\mathcal{A}|^2$, much smaller than $3^{|B|^2}$, but still significant[1].

**Proof of Correctness:** To Algorithm 4.4 correct, we begin with Lemma 4.20. We first

[1]Neither the algorithm nor our implementation prunes $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ for subsumed graphs. As our alphabet consists of two letters, this is acceptable for our use. For larger alphabets, pruning the set of initial supergraphs is suggested.

**Algorithm 4.4:** `RamseySubsumptionContainment(`$\mathcal{B}$`)`

```
Construct the set $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ of all single-letter graphs
Initialize the worklist $\widehat{W} \Leftarrow \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$
Initialize the set $\widehat{Q}^{\preceq} \Leftarrow \emptyset$
while $\widehat{W} \neq \emptyset$ do
    Remove an element $\widehat{g} = \langle\langle q, r\rangle, \widetilde{g}\rangle$ from $\widehat{W}$
    for $\widehat{h} \in \widehat{Q}^{\preceq}$ where $\bar{h} = \langle q, r\rangle$ do
1       if $\widehat{h} \preceq \widehat{g}$ then Discard $\widehat{g}$ and exit for
2       else if $\widehat{g} \preceq \widehat{h}$ then Remove $\widehat{h}$ from $\widehat{Q}^{\preceq}$
    if $\widehat{g}$ has not been discarded then
        if $q \in Q^{in}_{\mathcal{A}}$ and $r \in F_A$ then
            for $\widehat{h} \in \widehat{Q}^{\preceq}$ where $\bar{h} = \langle r, r\rangle$ do
                if $\langle\widehat{g}, \widehat{h}\rangle$ fails the lasso-finding test then return Not Contained
        if $q = r$ and $r \in F_A$ then
            for $\widehat{h} \in \widehat{Q}^{\preceq}$ where $\bar{h} = \langle s, r\rangle$ and $s \in Q^{in}_{\mathcal{A}}$ do
                if $\langle\widehat{h}, \widehat{g}\rangle$ fails the lasso-finding test then return Not Contained
        Add $\widehat{g}$ to $\widehat{Q}^{\preceq}$
        for $\widehat{h} \in \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ where $\bar{h} = \langle r, s\rangle$ do
            Add $\widehat{g}; \widehat{h}$ to $\widehat{W}$
return Contained
```

prove that for every weakly proper pair $\langle\widehat{g}, \widehat{H}\rangle$ of supergraphs it holds that the lasso-finding test determines if $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$, and that the languages of weakly proper pairs of supergraphs covers $L(\mathcal{A})$.

**Lemma 4.21.** *For a pair $\langle\widehat{g}, \widehat{h}\rangle$ of supergraphs, $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$ iff $\langle\widehat{g}, \widehat{h}\rangle$ passes the lasso-finding test.*

*Proof.* Recall that, By Lemma 4.7.(1), either $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$ or $Z(\widehat{g}, \widehat{h}) \cap L(\mathcal{B}) = \emptyset$. Thus it suffices to consider a single word $w = \sigma_0\sigma_1 \cdots \in Z(\widehat{g}, \widehat{h})$. As $Z(\widehat{g}, \widehat{h}) =$

$L(\widehat{g}) \cdot L(\widehat{h})^\omega$, $w$ can be partitioned by indices $i_1, i_2, \ldots$ into

$$w_0 = \sigma_0 ... \sigma_{i_1 - 1}$$
$$w_1 = \sigma_{i_1} ... \sigma_{i_2 - 1}$$
$$w_2 = \sigma_{i_2} ... \sigma_{i_3 - 1}$$
$$\ldots$$

where $w_0 \in L(\widehat{g})$ and for $i > 0$, $w_i \in L(\widehat{h})$.

In one direction, assume $\langle \widehat{g}, \widehat{h} \rangle$ passes the lasso-finding test. This implies there is a $q \in Q^{in}_{\mathcal{B}}$, $r \in Q_{\mathcal{B}}$, $a \in \{0,1\}$, and $S \subseteq Q_{\mathcal{B}}$ such that: $\langle q, a, r \rangle \in \widehat{g}$, there is a path from $r$ to $S$ in $\widehat{h}$, and $S$ is a 1-labeled SCC of $\widehat{h}$. First, since there is a path from $r$ to $S$ in $\widehat{h}$, and since $s$ is a 1-labeled SCC of $\widehat{h}$, we can constructing a sequence of states $r_1, r_2, r_3, \ldots$ so that: $r = r_1$; for every $i > 0$: $\langle r_i, a_i, r_{i+1} \rangle \in \widehat{h}$; and for infinitely many $i > 0$, it holds that $a_i = 1$. Second, since $w_0 \in L(\widehat{g})$, there is a finite run $p_0$ of $\mathcal{B}$ from $q$ to $r_1$ over $w_0$. Third, for every $i > 0$, since $w_i \in L(\widehat{h})$, there is a finite run $p_i$ of $\mathcal{B}$ from $r_i$ to $r_{i+1}$ over $w_i$. If $a_i = 1$, then there is an accepting finite run. By concatenating the finite runs $p_0, p_1, \ldots$, we form an accepting infinite run of $\mathcal{B}$ on $w$.

In the other direction, assume $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{B})$. This implies there is an accepting run $p$ of $w$ on $\mathcal{B}$. Since the run is accepting, $p_0 \in Q^{in}_{\mathcal{B}}$. Since there is a finite run between $p_0$ and $p_{i_1}$ on $w_0$, there is an arc $\langle p_0, a_0, p_{i_1} \rangle \in \widehat{g}$. Similarly, for every $k > 0$, there is a finite run between $p_{i_k}$ and $p_{i_{k+1}}$ on $w_k$, and therefore an arc $\langle p_{i_k}, a_k, p_{i_{k+1}} \rangle \in \widehat{h}$. Since there are a finite number of states, there is some state $t \in Q_{\mathcal{B}}$ so that for infinite many $k > 0$, $p_{i_k} = t$. Let $i'$ and $i''$ be the first two indices such that $p_{i'} = p_{i''} = t$. The arcs $\langle p_{i'}, a', p_{i'+1} \rangle, \ldots, \langle p_{i''-1}, a'', p_{i''} \rangle$ are a path in $\widetilde{h}$ from $t$ to itself, and thus $t$ is part of an SCC $S$. Further, the arcs $\langle p_{i_1}, a_1, p_{i_2} \rangle, \ldots, \langle p_{i'-1}, a', p_{i'} \rangle$ are a path in $\widetilde{h}$ from $p_{i_1}$ to the SCC $S$. Further, as the run is accepting, for an infinite number of $k$, $a_k = 1$. Thus some arc in a path from $t$ to itself is labeled $1$, and $S$ is a 1-labeled SCC. These are the conditions for $\langle \widehat{g}, \widehat{h} \rangle$ to pass the lasso-finding test. □

**Lemma 4.22.** *$L(\mathcal{A}) = \bigcup \{Z(\widehat{g}, \widehat{h}) \mid Z(\widehat{g}, \widehat{h})$ is weakly proper$\}$.*

*Proof.* Since every proper pair of supergraphs is also weakly proper, Lemma 4.6 implies $L(\mathcal{A}) \subseteq \bigcup \{Z(\widehat{g}, \widehat{h}) \mid Z(\widehat{g}, \widehat{h})$ is weakly proper$\}$. Note that the second property of weak properness is that $\bar{g} = \langle q, r \rangle$ and $\bar{h} = \langle r, r \rangle$ where $q \in Q^{in}_{\mathcal{A}}$ and $r \in F_{\mathcal{A}}$. By the definition of the languages of arcs, we thus have that every weakly proper $Z(\widehat{g}, \widehat{h}) \subseteq L(\mathcal{A})$. □

**Lemma 4.20.** *$L(\mathcal{A}) \not\subseteq L(\mathcal{B})$ iff $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a weakly proper pair $\langle \widehat{g}, \widehat{h} \rangle$ that fails the lasso-finding test.*

*Proof.* This follows from Lemma 4.22, Lemma 4.7.(1), and Lemma 4.21. □

We have now shown that it is sufficient to search $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ for pairs of graphs that fail the lasso finding test. In the final step of proving Algorithm 4.4 correct, we show that it is safe to restrict our search to graphs in $\widehat{Q}^{\preceq}$: that when using the lasso finding test $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a counterexample if and only if $\widehat{Q}^{\preceq}$ contains a counterexample. We do so by showing that if $\widehat{Q}^f_{\mathcal{B}}$ contains a pair of graphs that fails the lasso-finding test, then $\widehat{Q}^{\preceq}$ contains a pair of graphs approximating the pair in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$. This pair of graphs also fails the lasso-finding test. Recall that we say $\widehat{g} \preceq \widehat{h}$ when, for every arc $\langle q, a, r \rangle \in \widehat{g}$, there is an arc $\langle q, a', r \rangle \in \widehat{h}$, where if $a = 1$ then $a' = 1$. We first state some basic properties of approximation.

**Lemma 4.23.** *Let $\widehat{g}, \widehat{h}$ be two supergraphs where $\widehat{g} \preceq \widehat{h}$:*

*(1) For $q, r \in Q$, if $\widehat{g}$ has a path from $q$ to $r$ then $\widehat{h}$ has a path from $q$ to $r$*

*(2) If $S$ is a 1-labeled SCC in $\widehat{g}$, then there is 1-labeled SCC $T$ of $\widehat{h}$ where $S \subseteq T$*

*Proof.*

(1) Every arc in $\widehat{g}$ has a corresponding arc in $\widehat{h}$, so a path in $\widehat{g}$ remains a path in $\widehat{h}$.

(2) Let $S$ be a 1-labeled SCC in $\widehat{g}$. By part (1) above, all nodes in $S$ are in the same SCC $T$ of $\widehat{g}$. Furthermore, there must be $q, r \in S$ with an arc $\langle q, 1, r\rangle \in \widehat{g}$. This arc must also appear in $\widehat{h}$. Therefore $T$ is a 1-labeled SCC of $\widehat{h}$.

□

**Lemma 4.24.** *Let $\widehat{g}, \widehat{h}, \widehat{k}, \widehat{l}$ be four supergraphs. If $\widehat{g} \preceq \widehat{h}$ and $\widehat{k} \preceq \widehat{l}$, then $\widehat{g}; \widehat{k} \preceq \widehat{h}; \widehat{l}$.*

*Proof.* Take an arc $\langle q, a, s\rangle \in \widehat{g}; \widehat{k}$. First assume $a$ is 0. This arc exists because there is state $r$, an arc $\langle q, 0, r\rangle \in \widehat{g}$, and an arc $\langle r, 0, s\rangle \in \widehat{k}$. As $\widehat{g} \preceq \widehat{h}$, there is an arc $\langle q, a_1, r\rangle \in \widehat{h}$. As $\widehat{k} \preceq \widehat{l}$, there is an arc $\langle r, a_2, s\rangle$. Therefore the arc $\langle q, a_3, s\rangle \in \widehat{h}; \widehat{l}$, $a_3 \geq \max(a_1, a_2)$.

If $a$ is 1, there is state $r$, an arc $\langle q, a_1, r\rangle \in \widehat{g}$ and an arc $\langle r, a_2, s\rangle \in \widehat{k}$ where $a_1 = 1$ or $a_2 = 1$. As $\widehat{g} \preceq \widehat{h}$, there is an arc $\langle q, a_1', r\rangle \in \widehat{h}$. As $\widehat{k} \preceq \widehat{l}$, there is an arc $\langle r, a_2', s\rangle \in \widehat{l}$. If $a_1 = 1$, then $a_1'$ must be 1 and $\langle q, 1, s\rangle \in \widehat{h}; \widehat{l}$. If $a_2 = 1$, then $a_2' = 1$ and the arc $\langle q, 1, s\rangle \in \widehat{h}; \widehat{l}$. We can conclude that $\widehat{g}; \widehat{k} \preceq \widehat{h}; \widehat{l}$. □

Recall that a set, $\widehat{Q}_{\mathcal{A},\mathcal{B}}$, of supergraphs is $\preceq$*-closed under composition* when, for every two supergraphs, $\widehat{g}, \widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, there is a supergraph $\widehat{k} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$ so that $\widehat{k} \preceq \widehat{g}; \widehat{h}$. Given a set $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ of supergraphs we show that it is sufficient to test a $\preceq$-closed subset of $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$, its closure under composition, so long as the subset still contains $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$. Let $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ be a set of supergraphs, $\widehat{Q}^f_{\mathcal{B}}$ the closure of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ under composition, and $\widehat{Q}^{\preceq}$ a set computed during Algorithm 4.4.

**Lemma 4.25.**

*(1) $\widehat{Q}^{\preceq}$ is $\preceq$-closed*

*(2) $\widehat{Q}^{\preceq} \subseteq \widehat{Q}^f_{\mathcal{A},\mathcal{B}}$.*

*(3) For every $\widehat{g}_\sigma \in \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$, at every point in the computation of Algorithm 4.4, there is a $\widehat{h} \in \widehat{Q}^{\preceq} \cup \widehat{W}$ such that $\widehat{h} \preceq \widehat{g}_\sigma$.*

*Proof.*

(1) Note that the composition of two elements is only omitted from $\widehat{Q}^{\preceq}$ on lines 1 or 2 when it is approximated by an element already in $\widehat{Q}^{\preceq}$. Thus $\widehat{Q}^{\preceq}$ is indeed $\preceq$-closed.

(2) As every element in $\widehat{Q}^{\preceq}$ is the composition of a finite number of elements of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$, $\widehat{Q}^{\preceq}$ is indeed a subset of $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$,

(3) At the beginning of the computation, $\widehat{g}_\sigma$ is in the worklist $\widehat{W}$. Since the worklist is empty at the end of the computation, at some point we know $\widehat{g}_\sigma$ was removed from the worklist. In order for $\widehat{g}_\sigma$ to be discarded on line 1 or removed on line 2 of Algorithm 4.4, there must be a $\widehat{h}_0$ included in $\widehat{Q}^{\preceq}$ such that $\widehat{h}_0 \prec \widehat{g}_\sigma$. Similarly, $\widehat{h}_0$ can only be removed if a supergraph $\widehat{h}_1$ that approximates it is included, and so on. This allows us to conclude that at every point in the computation of Algorithm 4.4 there will be a supergraph $\widehat{h}_i \in \widehat{Q}^{\preceq}$ such that:

$$\widehat{h}_i \preceq \widehat{h}_{i-1} \preceq \cdots \preceq \widehat{h}_0 \preceq \widehat{g}$$

As approximation is transitive, it holds that $\widehat{h}_i \preceq \widehat{g}$.

□

**Lemma 4.26.** *For every $\widehat{g} \in \widehat{Q}^f_{\mathcal{A},\mathcal{B}}$, there is $\widehat{g}' \in \widehat{Q}^{\preceq}$ such that $\widehat{g}' \preceq \widehat{g}$.*

*Proof.* The supergraph $\widehat{g}$ is the finite composition $\widehat{g}_0; \widehat{g}_1; ...; \widehat{g}_{n-1}$ of $n$ elements of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$. Let $\widehat{g}_{0...i}$ be the sequence of compositions $\widehat{g}_0; ...; \widehat{g}_i$. We prove by induction that, for each $i$, there is a supergraph $\widehat{h}_i \in \widehat{Q}^{\preceq}$ so that $\widehat{h}_i \preceq \widehat{g}_{0...i}$. Since $\widehat{g} = \widehat{g}_{1...(n-1)}$, this implies

$\widehat{h}_{n-1} \preceq \widehat{g}$. The base case follows immediately from the third clause of Lemma 4.25 and the fact that the worklist is empty at the end of the computation.

Inductively, assume there is a supergraph $\widehat{h}_i \in \widehat{Q}^{\preceq}$ so that $\widehat{h}_i \preceq \widehat{g}_{0\dots i}$. First, as $\widehat{g}_{i+1} \in \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$, the third clause of Lemma 4.25 implies a $\widehat{g}'_{i+1} \in \widehat{Q}^{\preceq}$ exists so that $\widehat{g}'_{i+1} \preceq \widehat{g}_{i+1}$. Second, note that by Lemma 4.24 it holds that $\widehat{h}_i; \widehat{g}'_{i+1} \preceq \widehat{g}_{0\dots i}; \widehat{g}_{i+1}$. Finally, as $\widehat{Q}^{\preceq}$ is $\preceq$-closed under composition, either $\widehat{h}_i; \widehat{g}'_{i+1}$ is in $\widehat{Q}^{\preceq}$, or it is discarded/removed on line 1/2 when it is approximated by some $\widehat{k} \in \widehat{Q}^{\preceq}$. In the former case, take $\widehat{h}_{i+1}$ to be $\widehat{h}_i; \widehat{g}'_{i+1}$. In the later case, recall that approximation is transitive and take $\widehat{h}_{i+1}$ to be $\widehat{k}$.

□

**Lemma 4.27.** *$\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a pair of supergraphs that fails the lasso-finding test iff $\widehat{Q}^{\preceq}$ contains a pair of supergraphs that fails the lasso-finding test.*

*Proof.* In one direction, assume $\widehat{Q}^{\preceq}$ contains a pair $\langle \widehat{g}, \widehat{h} \rangle$ of supergraphs that fails the lasso-finding test. By Lemma 4.25, $\widehat{Q}^{\preceq} \subseteq \widehat{Q}^f_{\mathcal{A},\mathcal{B}}$. Therefore $\langle \widehat{g}, \widehat{h} \rangle$ is such a pair of supergraphs in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$.

In the other direction, assume $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a pair $\langle \widehat{g}, \widehat{h} \rangle$ of supergraphs that fails the lasso-finding test. By Lemma 4.26 there must be a $\widehat{g}', \widehat{h}' \in \widehat{Q}^{\preceq}$ such that $\widehat{g}' \preceq \widehat{g}$ and $\widehat{h}' \preceq \widehat{h}$. We show by way of contradiction that $\langle \widehat{g}', \widehat{h}' \rangle$ must fail the lasso-finding test. Presume that $\langle \widehat{g}', \widehat{h}' \rangle$ passes the lasso-finding test. This implies the existence of $q \in Q^{in},\ r \in Q,\ a, b \in \{0, 1\}$ and $S \subseteq Q$ such that $\langle q, a, r \rangle \in \widehat{g}'$, there is a path from $r$ to $S$ in $\widehat{h}'$, and $S$ is a 1-labeled SCC of $\widehat{h}'$. As $\widehat{g}' \preceq \widehat{g}$, there must be an arc $\langle q, a', r \rangle \in \widehat{g}$. Further, by the second half of Lemma 4.23 there must be a $T \supseteq S$ that is a 1-labeled SCC of $\widehat{h}$. By the first half of Lemma 4.23, there must be a path from $r$ to $S$, and thus to $T$, in $\widehat{h}$. However, as $\langle \widehat{g}, \widehat{h} \rangle$ fails the lasso-finding test, there can be no such $q$, $r$, and $T$.

□

**Theorem 4.28.** *For every two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, it holds that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff* `RamseySubsumptionContainment` *($\mathcal{A}$,$\mathcal{B}$) returns Contained.*

*Proof.* This follows Lemmas 4.20 and 4.27. $\square$

## 4.3 Empirical Comparisons: Universality Experiments

The subsumption technique employed in Algorithm 4.4 is purely a heuristic improvement: the worst-case complexity of the algorithm does not change. Thus the Ramsey-based algorithm has a worst-case running time exponentially slower than that of the rank-based algorithm. Motivated by the strong performance of Ramsey-based algorithms on SCT problems [FV09] and the competitive sizes of complemented automata in Chapter 3, we compare Ramsey and rank based solvers on a terrain of random automata.

To evaluate the performance of various tools on Büchi universality problems, we employ the random model describe in Chapter 3.1. Data points are derived from 100 or more[2] random automata with the given $n$, $r$, and $f$. When possible, we compute the median running time [DR09, TV07]. This allows us to plot the data on a logarithmic scale and easily judge exponential behavior. However, in many cases interesting behavior emerges after a significant percentage of the runs time out. In these cases we measure the timeout percentage instead of median running time.

Our rank-based tool, simply called RANK, is our modified version of the **mh** tool developed by Doyen and Raskin [DR09]. Our Ramsey-based tool, called RAMSEY, is based on the **sct/scp** program– an optimized C implementation of the SCT algorithm from Ben-Amram and Lee [BAL07]. We have rewritten the RAMSEY tool to solve

[2]When the results from 100 automata appear anomalous, additional automata are generated and tested to improve the fidelity of the results. No data are ever excluded.

arbitrary Büchi universality problems by implementing Algorithm 4.4. Both tools can be configured to not employ their subsumption techniques. In this case, we append (ns) to the program name.

All experiments were performed on the Shared University Grid at Rice (SUG@R), a cluster of Sunfire x4150 nodes, each with two 2.83GHz Intel Xeon quad-core processors and 16GB of RAM. Each run is given a dedicated node, and each tool is given one hour to solve each problem.

### 4.3.1 Subsumption

We know that subsumption is vital to the performance of rank-based solvers [DR09]. Further, we have observed subsumption's utility on the domain of SCT problems [FV09]. This motivates us to extend the subsumption technique of [BAL07] to the case of general Büchi universality, resulting in Algorithm 4.4. Employing observations from Section 4.3.2 below, we check the practical utility of subsumption on the most difficult terrain point for RAMSEY, where transition density $r = 1.5$ and acceptance density $f = 0.5$. Figure 4.2 displays RAMSEY's performance as size increases, on a logarithmic scale. If more than $50\%$ of the problems timed out, the median is displayed at $3600$ seconds, which flattens the RAMSEY (ns) line at the last data point. We observe that the RAMSEY (ns) line has a higher slope than the RAMSEY line. As this graph uses a logarithmic scale, this difference in the slope indicates an exponential improvement in scalability when subsumption is used. Similar results held for every terrain point we measured, demonstrating that although a heuristic technique, subsumption is required for the scalability of our Ramsey-based approach. We also note that the curves appear to be linear on the logarithmic scale, suggesting that the median running time for this terrain point is $2^{O(n)}$, rather than the $2^{O(n^2)}$ of the worst-case complexity bound.

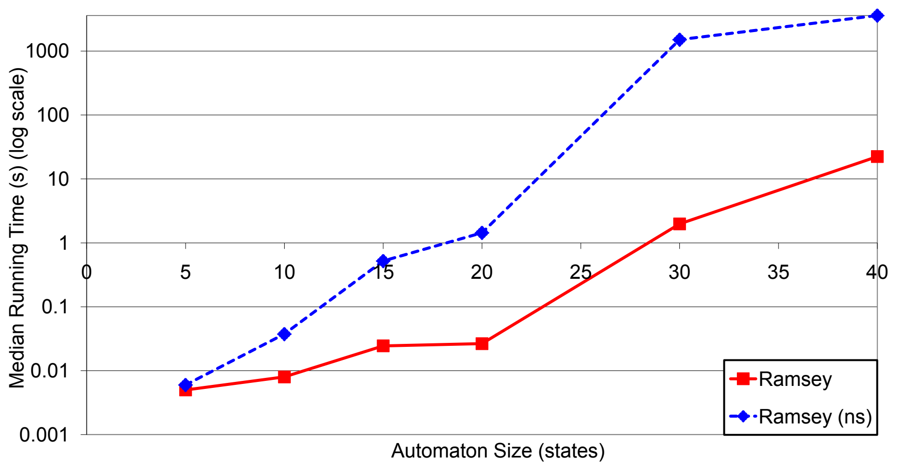
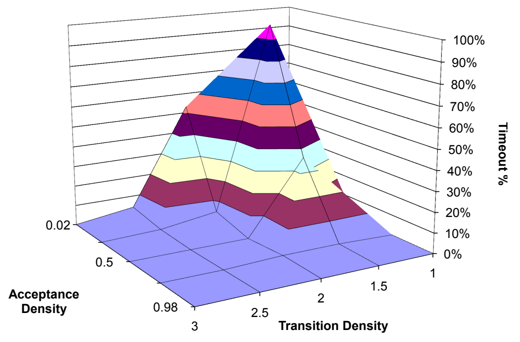


**Figure 4.2 :** Subsumption exponentially improves RAMSEY median running times ($r = 1.5, f = 0.5$) Note that when more than $50\%$ of the problems timed out, the median is displayed at $3600$ seconds, which flattens the RAMSEY (ns) line at the last data point.

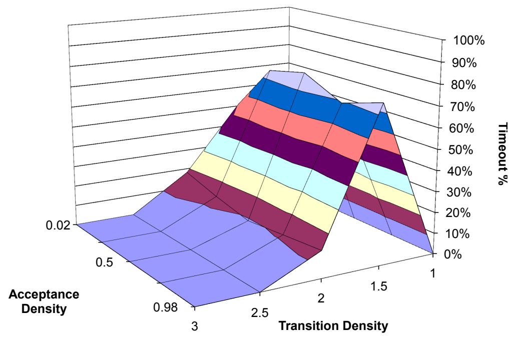
### 4.3.2 Terrain Experiments

As stated above, randomly generated automata of a given size can configured in two parameters. To map out the behavior of the two tools over this terrain, we hold size constant at $n = 100$, and examined a variety of terrain points. We generate data points for each combination of transition density $r \in \{0.02, 0.26, 0.50, 0.74, 0.98\}$ and acceptance density $f \in \{0.5, 1.5, 2.0, 2.5, 3.0\}$.

Figure 4.3(a) displays the percentage of cases in which the RANK tool timed out in each terrain point. As observed in [DR09], there is a sharp spike in timeouts at transitions density $r = 1.5$, acceptance density of $0.26$. This spike trails off quickly as transition density changes, and only slightly more gradually as acceptance density changes. There is a subtler high point at $r = 2.0$, $f = 0.02$, where the timeouts rise to $50\%$. This is consistent with other rank-based tools, even those using different lasso-finding algorithms [TV07]. Figure 4.3(b) displays the percentage of cases in which the



**(a)** RANK



**(b)** RAMSEY

**Figure 4.3 :** Differences in behavior between RANK and RAMSEY over problem terrain, measured as percentage of problems that timeout when size $n = 100$

RAMSEY tool timed out in each terrain point. Like RANK, $r = 1.5$, $f = 0.26$ is a difficult terrain point for RAMSEY. However, RAMSEY continues to time out frequently along all terrain points with transition density $r = 1.5$, and has no significant timeouts at other terrain points.

Simply glancing at the terrain graphs, it appears that RANK may perform better than RAMSEY in most terrain points. On the other hand, RAMSEY does not exhibit a second high point at $r = 2.0, f = 0.02$, and at least for this size of automata RAMSEY beats RANK at the hardest point for RANK. What these graphs clearly show is that those attributes that make a problem hard for RANK to handle are not necessarily the same as those attributes of a problem that cause difficulty for RAMSEY.

### 4.3.3 Scaling Experiments

We explore some interesting terrain points by measuring the scalability of each algorithm: we hold the transition and acceptance densities constant, and increase size. We choose to investigate three terrain points: a point $r = 1.5$, $f = 0.5$, where RANK seems

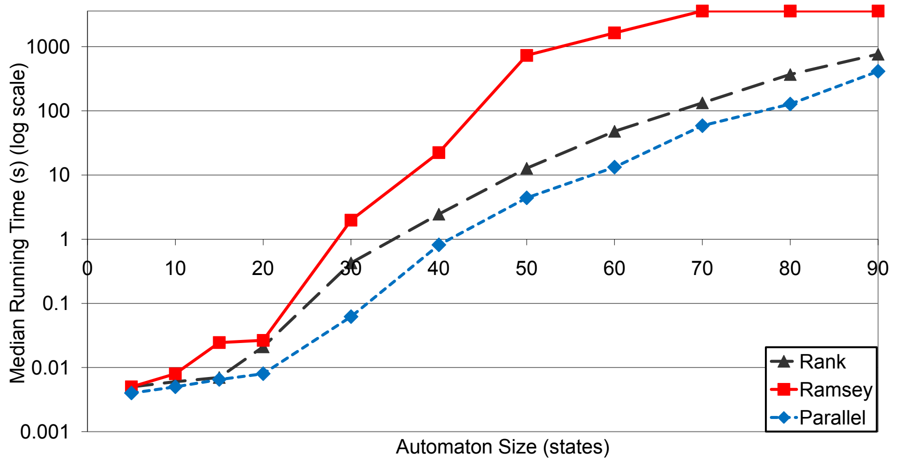


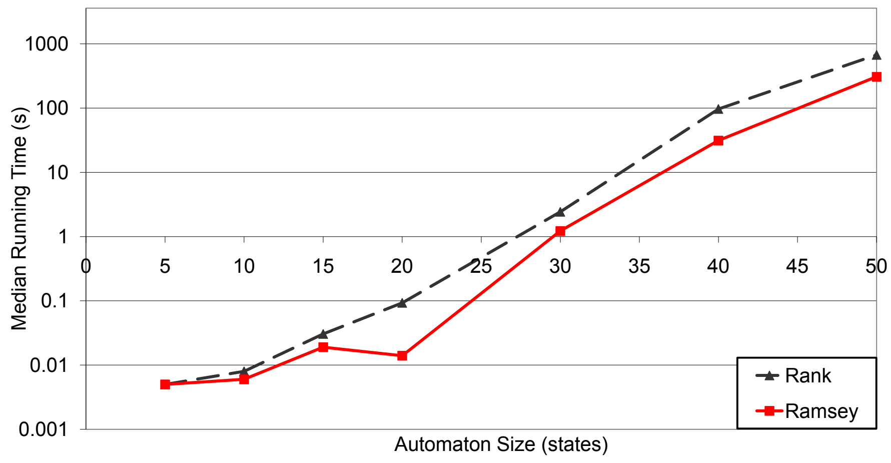
**Figure 4.4 :** RANK scales exponentially better than RAMSEY at $r = 1.5$, $f = 0.5$.

to perform better than RAMSEY; the main spike $r = 1.5$, $f = 0.26$, where both tools exhibited difficulty solving problems; and a final point $r = 2.0$, $f = 0.05$ near RANK's second high point, where RAMSEY seems to perform better.

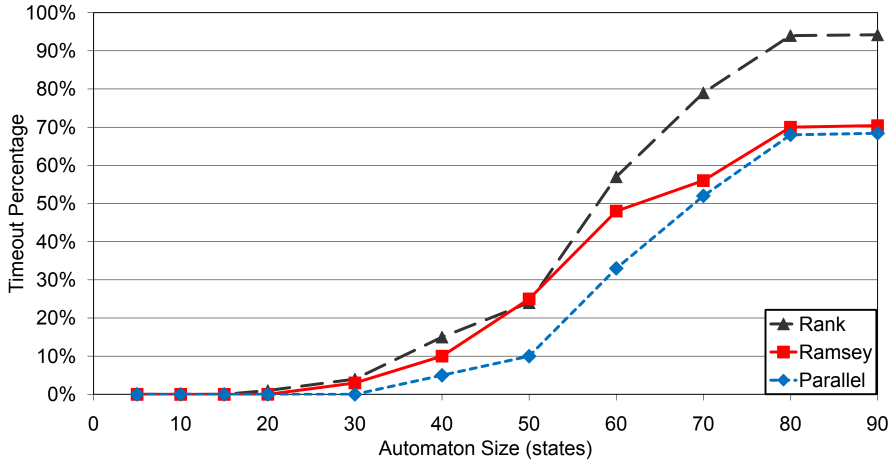
Figure 4.4 displays the median running time for problems with the transition density at $r = 1.5$ and the acceptance density at $f = 0.5$, on a logarithmic scale. If more than $50\%$ of the problems timed out, the median is displayed at $3600$ seconds, cutting off RAMSEY's line. As the scale is logarithmic, the difference in the slope between RANK's line and RAMSEY's indicates that, on this terrain point, RANK clearly scales exponentially better than RAMSEY. The third line, labeled "Parallel", displays the behavior of running both tools in parallel on separate machines, and terminating as soon as either tool gives an answer. Is is notable that this line, while having the same slope as RANK's, is lower; indicating there are a number of cases even at this terrain point where RAMSEY terminates before RANK.

The most difficult terrain points for both tools lie near $r = 1.5$, $f = 0.26$. Figure 4.5(a) shows the median running time for each size $n$ at this terrain point. Up to

$n = 50$, RAMSEY performs better than RANK only by a constant factor . Past this this size, the percentage of timeouts is too high for median measurements to be meaningful. However, a gap in the timeout percentage appears as the automata grow larger than 50 states. Figure 4.5(b) displays the percentage of runs that timed out for each size $n$ at this terrain point. It does appear that, past $n = 50$, RAMSEY begins to scale significantly better than RANK. The behavior of running both tools in parallel on separate machines is shown using the third line, labeled “Parallel.” We again find that even at a terrain point that favors one tool, RAMSEY, we benefit from running both tools simultaneously.



**(a)** Up to inputs of 50 states, RAMSEY performs a constant factor better than RANK (median running time, log scale).



**(b)** As inputs grow past 50 states, RAMSEY scales better than RANK (timeout percentage).

**Figure 4.5 :** RAMSEY and RANK on the most difficult terrain point ($r = 1.5$, $f = 0.26$)

At size $n = 100$, RANK exhibited difficulty when the transition density was $2.0$ and the acceptance density was low. We measured the scalability of RAMSEY and RANK on problems with $r = 2.0$ and $f = 0.05$. At this terrain point the median running times do not increase exponentially for either RANK or RAMSEY. As a large number of problems still did not complete, Figure 4.6 displays the timeout percentages as size grows. At this terrain point, RAMSEY does appear to scale better than RANK. However the gap is not the exponential improvement we observed when RANK performed better than RAMSEY. At this configuration, running the tools in parallel was only a slight improvement over running RAMSEY alone.

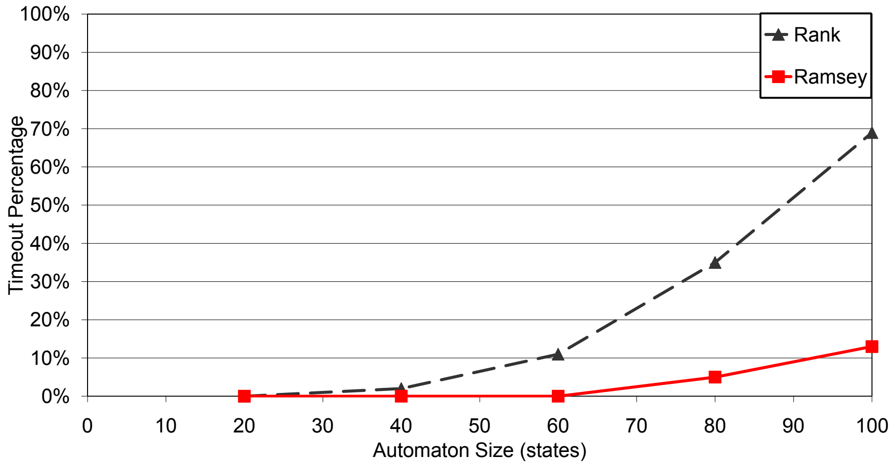


**Figure 4.6 :** RAMSEY scales much better than RANK when $r = 2$ and $f = 0.05$

### 4.3.4 Universal vs. Non-Universal Automata

When we reviewed the algorithms used by RANK and RAMSEY to explore the state space of the complemented automaton, we noted that in certain cases each tools can terminate before computing the entirety of their fixpoints: RANK on universal automata, and RAMSEY on non-universal automata. This suggests that RANK may perform better on universal automata, and RAMSEY may perform better on non-universal automata.

To confirm this hypothesis, we compare RANK and RAMSEY on a corpus of universal and non-universal automata. Our corpus is derived from 1000 automata with size $n = 50$, transition density $r = 1.8$, and acceptance density $f = 0.2$. This point was chosen because of the relatively equal proportion of universal and non-universal automata. Table 4.1 summarizes the results. RANK does indeed perform better on universal automata. Universal automata were solved in a median time of 108.3 seconds, while on non-universal automata, the median running time was 177.8 seconds. We observe the inverse behavior in RAMSEY: on non-universal automata RAMSEY had a median running time of only 33.1 seconds, while on universal automata the median running time was 253.4 seconds. The universality or non-universality of a problem does affect the performance of each approach.

| | Count | RANK | RAMSEY |
|---|---|---|---|
| Universal | 460 | 108.3 | 253.4 |
| Non-Universal | 527 | 177.8 | 33.1 |
| Unknown | 13 | | |

**Table 4.1 :** RANK performs better on universal problems, RAMSEY on non-universal problems, measured by median running time ($n = 50, r = 1.8, f = 0.2$)

The question naturally arises: does the difference in performance on universal vs. non-universal automata fully explain the different behaviors of RAMSEY and RANK. This is not the case. As previously noted in Figure 4.4, RANK performs exponentially better than RAMSEY on automata with a transition density of $1.5$ and an acceptance density of $0.5$. More than $80\%$ of the solved automata at this terrain point are non-universal: a distribution that should favor RAMSEY. Further, Figure 4.6 shows a terrain point where RAMSEY scales significantly better than RANK. At this terrain point, more than two-thirds of solved automata with $n > 50$ were universal, and should have favored RANK. Therefore we cannot conclude that the difference in behavior between RANK and RAMSEY is truly attributed to the gap in performance between universal and non-universal automata.

### 4.3.5 Discussion

These experiments tell two stories. The first story is about subsumption. In general, subsumption is a trade off: there is a benefit to reducing the working sets of the algorithms, but checking for subsumed states can be computationally expensive. In the domain of CNF satisfiability solvers, subsumption is generally regarded as an ineffective technique: the overhead of checking for subsumed clauses outweighs any benefit gained from removing them. For checking Büchi automata universality, it has previously been shown that subsumption is not only useful, but vital for the scalability of the rank-based approach [DR09]. Here, we demonstrate that this also holds for the Ramsey-based approach, which use not only a different construction but also a different algorithm to explore the state space of this construction.

The second story is that neither the rank-based approach nor the Ramsey-based approach to Büchi universality testing is clearly superior. This is true despite the massive gap in worst-case complexity between the two approaches. Each approach exhibits distinct behavior on the terrain of random universality problems. Due to these differences, we do not believe a winner takes all approach is best for universality checking. The current best approach is to run both tools in parallel, and see which terminates first. Doing so improves performance by a constant factor, relative to the best tool for a given terrain point.

One point that is well demonstrated in our investigation is that theoretical worst-case analysis often yields little information on actual algorithmic performance; an algorithm running in $2^{O(n^2)}$ can perform better in practice than an algorithm running in $2^{O(n \log n)}$. We do note RAMSEY, the program running in $2^{O(n^2)}$ time and space, sometimes consumed on the order of 20 GB of memory, where RANK rarely consumed more than 300 megabytes.

# Chapter 5

# Profile-Based Complementation

In an effort to unify Büchi complementation with other operations on automata, Kähler and Wilke introduced yet another analysis of runs of nondeterministic Büchi automata [KW08]. The analysis is based on *reduced split trees*, which are related to the Müller-Schupp trees used for determinization [MS95]. A reduced split tree is a binary tree whose nodes are sets of states as follows: the root is the set of initial states; and given a node associated with a set of states, its left child is the set of successors that are accepting, while the right child is the set of successors that are not accepting. In addition, each state of the automaton appears at most once in each level of the binary tree: if it would appear in more than one set, it occurs only in the leftmost one. The construction that follows from the analysis, termed the *slice-based construction*, is simpler than Safra's determinization, but its implementation suffers from similar difficulties: the need to refer to leftmost children requires encoding of a preorder, and working with reduced split trees makes the transition relation between states awkward. Thus, as has been the case with Safra's construction, it is not clear how the slice-based approach can be implemented symbolically. This is unfortunate, as the slice-based approach does offer a very clean and intuitive analysis, suggesting that a better construction is hidden in it.

In this chapter we reveal such a hidden, elegant, construction, and we do so by unifying the rank-based and the slice-based approaches. Before we turn to describe our construction, let us point to a key conceptual difference between the two approaches.

This chapter contains work published in [FKVW10].

This difference has made their relation of special interest and challenge. In the rank-based approach, the ranks assigned to a node bound the visits to accepting states yet to come. Thus, the ranks refer to the *future* of the run, making the rank-based approach inherently nondeterministic. In contrast, in the slice-based approach, the partition of the states of the automaton to the different sets in the tree is based on previous visits to accepting states. Thus, the partition refers to the *past* of the run, and does not depend on its future.

In order to draw parallels between the two approaches, we present a formulation of the slice-based approach in terms of run DAGs. A careful analysis of the slice-based approach then enables us to reduce the nondeterminism in the construction. We can then employ this improved slice-based approach in order to define a particular odd ranking of rejecting run DAGs, called a *retrospective ranking*. In addition to revealing the theoretical connections between the two seemingly different approaches, the new ranks lead to a complementation construction with a transition function that is smaller and deterministic in the limit: every accepting run of the automaton is eventually deterministic. This presents the first deterministic-in-the-limit complementation construction that does not use determinization. Determinism in the limit is central to verification in probabilistic settings [CY95] and has proven useful in experimental results [ST03]. Phrasing slice-based complementation as an odd ranking also immediately affords us the improved cut-point of Schewe, the subsumption operation of Doyen and Raskin, and provides an easy symbolic encoding.

## 5.1 Preliminaries: The Slice-Based Approach

The paper of Kähler et al. introduces the notion of the split tree, reduced split tree, and skeleton tree for a run of an automaton $\mathcal{B}$ on a word $w$. A binary tree is a prefix-closed non-empty subsets of $\{0 \dots 1\}^*$. The elements $v \in V$ are called *nodes*. The root is $\epsilon$. For a node $v \in V$, the node $v0$ is called the left child of $v$, and $v1$ the right child of $v$. The node $v$ is said to be on level $i$ when $|v| = i$. For a set $L$, an $L$-labeled tree is a pair $\langle V, l\rangle$ where $V$ is a tree and $l : V \to L$ is a label function. By abuse of notation, for an $L$-labeled tree $T = \langle V, l\rangle$ and vertex $v$, say $v \in T$ when $v \in V$, and let $T(v) = l(v)$. For two nodes $v$ and $v'$, say that $v' > v$ when $|v| = |v'|$ and $v'$ is to the right of, i.e. lexicographically larger than, $v$. A tree is of *bounded width* when there is a constant $l$ such that for every $i$ there are at most $l$ nodes on level $i$.

The *split tree*, written $T^{sp}$, is the $2^Q$-labeled tree defined inductively as follows[1]. As a base case, $\epsilon \in T^{sp}$ and $T^{sp}(\epsilon) = Q^{in}$. Inductively, given a node $v$ on level $i$, let $P = T^{sp}(v)$. If $\rho_{\mathcal{B}}(P, w_i) \setminus F_{\mathcal{B}} \neq \emptyset$ then $v0 \in T^{sp}$ and $T^{sp}(v0) = \rho_{\mathcal{B}}(P, w_i) \setminus F_{\mathcal{B}}$. Similarly, if $\rho_{\mathcal{B}}(P, w_i) \cap F_{\mathcal{B}} \neq \emptyset$, then $v1 \in T^{sp}$ and $T^{sp}(v1) = \rho_{\mathcal{B}}(P, w_i) \cap F_{\mathcal{B}}$. As argued in [KW08], branches in $T^{sp}$ correspond to runs of $\mathcal{B}$ on $w$.

**Lemma 5.1.** *$w \in L(\mathcal{B})$ iff $T^{sp}(\mathcal{B}, w)$ has a branch that goes right infinitely often.*

The *reduced split tree*, written $T^{rs}$, keeps only the rightmost instance of each state at each level of the tree. This bounds the width of $T^{rs}$ to $n$. Formally, we define $T^{rs}$ inductively as follows. As a base case, the root $\epsilon \in T^{rs}$, and $T^{rs}(\epsilon) = Q^{in}$. Inductively, given a node $v \in T^{rs}$ on level $i$, let $P = T^{rs}(v)$ and let $P' = \bigcup\{\rho_{\mathcal{B}}(T^{rs}(v') \mid v' \in T^{rs} \text{ and } v' < v\}$. If $(\rho_{\mathcal{B}}(P, w_i) \setminus F_{\mathcal{B}}) \setminus P' \neq \emptyset$ then $v0 \in T^{rs}$ and $T^{rs}(v0) = (\rho_{\mathcal{B}}(P, w_i) \setminus F_{\mathcal{B}}) \setminus P'$. Similarly, if $(\rho_{\mathcal{B}}(P, w_i) \cap F_{\mathcal{B}}) \setminus P' \neq \emptyset$, then $v1 \in T^{rs}$ and $T^{rs}(v1) = (\rho_{\mathcal{B}}(P, w_i) \cap F_{\mathcal{B}}) \setminus P'$.

**Lemma 5.2.** *$T^{rs}$ has a branch that goes right infinitely often iff $T^{sp}$ has a branch that goes right infinitely often.*

[1]Compared to [KW08], these definitions reverse the left and right children. This was done to match below.

Finally, the *skeleton* $T^{sp}$ is obtained by removing from the reduced split tree all nodes that are finite. The slice automaton of Kähler and Wilke proceeds by tracking the levels of $T^{rs}$ and guessing which nodes occur in $T^{sp}$. Each level $i$ of $T^{rs}$ is encoded as a *slice*, a sequence $\langle P_0, \ldots, P_m \rangle$ of pairwise disjoint subsets of $Q$.

## 5.2 Analyzing DAGs With Profiles

In this section we present an alternate formulation of the slice-based complementation construction of Kähler and Wilke [KW08]. Whereas Kähler and Wilke approached the problem through reduced split trees, we derive the slice-based construction directly from an analysis of the run DAG. This analysis proceeds by pruning $G$ in two steps: the first removes edges, and the second removes vertices.

### 5.2.1 Profiles

Consider a run DAG $G = \langle V, E \rangle$. Let $l \colon V \to \{0, 1\}$ be such that $l(\langle q, i \rangle) = 1$ if $q \in F$ and $l(\langle q, i \rangle) = 0$ otherwise. Thus, $l$ labels $F$-nodes by $1$ and all other nodes by $0$. The *profile* of a path in $G$ is the sequence of labels of nodes in the path. The profile of a node is then the lexicographically maximal profile of all initial paths to that node. Formally, let $\leq$ be the lexicographic ordering on $\{0,1\}^* \cup \{0,1\}^\omega$. The profile of a finite path $b = v_0, v_1, \ldots, v_n$ in $G$, written $h_b$, is $l(v_0)l(v_1)\cdots l(v_n)$, and the profile of an infinite path $b = v_0, v_1, \ldots$ is $h_b = l(v_0)l(v_1)\cdots$. Finally, the profile of a node $v$, written $h_v$, is the lexicographically maximal element of $\{h_b \mid b \text{ is an initial path to } v\}$. The lexicographic order of profiles induces a preorder over nodes.

We define the sequence of preorders $\preceq_i$ over the nodes on each level of the run DAG as follows[2]. For every two nodes $u$ and $v$ on a level $i$, we have that $u \prec_i v$ if $h_u < h_v$,

[2]In this and the following chapter, $\preceq$ is not a subsumption relation.

and $u \approx_i v$ if $h_u = h_v$. For convenience, we conflate nodes on the $i$th level of the run DAG with their states when employing this preorder, and say $q \preceq_i r$ when $\langle q, i\rangle \preceq_i \langle r, i\rangle$. Note that $\approx_i$ is an equivalence relation. Since the final element of a node's profile is $1$ iff the node is an $F$-node, all nodes in an equivalence class must agree on membership in $F$. We call an equivalence class an $F$-class when all its members are $F$-nodes, and a non-$F$-class when none of its members is an $F$-node.

We now use profiles in order to remove from $G$ edges that are not on lexicographically maximal paths. Let $G'$ be the subgraph of $G$ obtained by removing all edges $\langle u, v\rangle$ for which there is another edge $\langle u', v\rangle$ such that $u \prec_{|u|} u'$. Formally, $G' = \langle V, E'\rangle$ where $E' = E \setminus \{\langle u, v\rangle \mid \text{there exists } u' \in V \text{ such that } \langle u', v\rangle \in E \text{ and } u \prec_{|u|} u'\}$.

**Lemma 5.3.** *For every two nodes $u$ and $v$, if $(u, v) \in E'$, then $h_v \in \{h_u 0, h_u 1\}$.*

*Proof.* Assume by way of contradiction that $h_v \notin \{h_u 0, h_u 1\}$. Recall that $h_v$ is the lexicographically maximal element of $\{h_b \mid b \text{ is an initial path to } v\}$. Thus our assumption entails an initial path $b$ to $v$ so that $h_b > h_u 1$. Let $u'$ be $b_{|u|}$: the node on the same level of $G$ as $u$. Since $b$ is a path to $v$, it holds that $(u', v) \in E$. Further, $h_b > h_u 1$, it must be that $h_{u'} > h_u$. By definition of $E'$, the presence of $(u', v)$ where $h_{u'} > h_u$ precludes the edge $(u, v)$ from being in $E'$ — a contradiction. □

Note that while it is possible for two nodes with different profiles to share a child in $G$, Lemma 5.3 precludes this possibility in $G'$. If two nodes join in $G'$, they must have the same profile and be in the same equivalence class.

**Lemma 5.4.** *Consider nodes $u$ and $v$ on level $i$ and nodes $u'$ and $v'$ on level $i + 1$. If $\langle u, u'\rangle \in E'$, $\langle v, v'\rangle \in E'$, and $u' \approx_{i+1} v'$, then $u \approx_i v$.*

*Proof.* We have that $u' \approx_{i+1} v'$ iff $h_{u'} = h_{v'}$. If $u'$ is an $F$-node, then $v'$ must also be an $F$-node, and then the last letter in both $h_{u'}$ and $h_{v'}$ is $1$. Thus by Lemma 5.3 we have $h_u1 = h_{u'} = h_{v'} = h_v1$. If $u'$ is a non-$F$-node, then by Lemma 5.3 we have $h_u0 = h_{u'} = h_{v'} = h_v0$. In either case, $h_u = h_v$ and $u \approx_i v$. □

Lemma 5.4 allows us to conflate nodes and equivalence classes, and for every edge $(u, v) \in E'$, consider $[v]$ to be the child of $[u]$. Lemma 5.3 then entails that the class $[u]$ can have at most two children: the class of $F$-nodes with profile $h_u1$, and the class of non-$F$-nodes with profile $h_u0$. We call the first class the $F$-child of $[u]$, and the second class the non-$F$-child of $[u]$.

By using lexicographic ordering we can derive the preorder for each level $i+1$ of the run DAG solely from the preorder for the previous level $i$. To determine the relation between two nodes, we need only know whether the nodes are $F$-nodes and the relation between the parents of those nodes. Formally, we have the following.

**Lemma 5.5.** *For all nodes $u, v$ on level $i$, and nodes $u', v'$ where $E'(u, u')$ and $E'(v, v')$:*

- *If $u \prec_i v$, then $u' \prec_{i+1} v'$.*
- *If $u \approx_i v$ and both $u'$ and $v'$ are $F$-nodes, then $u' \approx_{i+1} v'$.*
- *If $u \approx_i v$ and neither $u'$ nor $v'$ are $F$-nodes, then $u' \approx_{i+1} v'$.*
- *If $u \approx_i v$ and $v'$ is an $F$-node while $u'$ is not, then $u' \prec_{i+1} v'$.*

*Proof.* If $u \prec_i v$, then $h_u < h_v$ and, by Lemma 5.3, we know that $h_{u'} \in \{h_u0, h_u1\}$ must be smaller than $h_{v'} \in \{h_v0, h_v1\}$, implying that $u' \prec_{i+1} v'$. If $u \approx_i v$, we have three sub-cases. If $v'$ is an $F$-node and $u'$ is not, then $h_{u'} = h_u0 = h_v0 < h_v1 = h_{v'}$ and $u' \prec_{i+1} v'$. If both $u'$ and $v'$ are $F$-nodes, then $h_{u'} = h_u1 = h_v1 = h_{v'}$ and $u' \approx_i v'$. Finally, if neither $u'$ nor $v'$ are $F$-nodes, then $h_{u'} = h_u0 = h_v0 = h_{v'}$ and $u' \approx_i v'$. □

We now demonstrate that by keeping only edges associated with lexicographically maximal profiles, $G'$ captures an accepting path from $G$.

**Lemma 5.6.** *$G'$ has an accepting path iff $G$ has an accepting path.*

*Proof.* If $G'$ has an accepting path, then its superset $G$ has the same path.

In the other direction, assume $G$ has an accepting path. Consider the set $P$ of accepting paths in $G$. We prove that there is a lexicographically maximal element $\pi \in P$. To begin, we construct an infinite sequence, $P_0, P_1, \ldots$, of subsets of $P$ such that the elements of $P_i$ are lexicographically maximal in the first $i+1$ positions. Since no initial states are accepting, no nodes on level $0$ on $G$ will be $F$ nodes. Therefore let $P_0 = P$. Inductively, if $P_i$ contains an element $b$ such that $b_{i+1}$ is an $F$-node, then $P_{i+1} = \{b \mid b \in P_i,\ b_{i+1} \text{ is an } F\text{-node}\}$. Otherwise $P_{i+1} = P_i$. Note that since $G$ has an accepting path, $P$ is non-empty. Further, every set $P_i$ is not equal to its predecessor $P'$ only when there is a path in $P'$ with an $F$-node in the $i$th position. In this case, that path is in $P_i$. Thus every $P_i$ is non-empty.

First, we prove that there is a path $\pi \in \bigcap_{i \geq 0} P_i$. Consider the sequence $U_0, U_1, U_2, \ldots$ where $U_i$ is the set of nodes that occur at position $i$ in runs in $P_i$. Formally, $U_i = \{u \mid u \in G,\ b \in P_i,\ u = b_i\}$. Each node in $U_{i+1}$ has a parent in $U_i$, although it may not have a child in $U_{i+2}$. We can thus connect the nodes in $\bigcup_{i>0} U_i$ to their parents, forming a sub-DAG of $G$. As every $P_i$ is non-empty, every $U_i$ is non-empty, and this DAG has infinitely many nodes. Since each node has at most $n$ children, by Koñig's Lemma there is an initial path $\pi$ through this DAG, and thus through $G$. We now show by induction that $\pi \in P_i$ for every $i$. As a base case, $\pi \in P = P_0$. Inductively, assume $\pi \in P_i$. The set $P_{i+1}$ is either $P_i$, in which case $\pi \in P_{i+1}$, or the set $\{b \mid b \in P_i,\ b_{i+1} \text{ is an } F\text{-node}\}$. In this latter case, as $U_{i+1}$ consists only of $F$-nodes, the node $\pi_{i+1}$ must be an $F$-node, and $\pi \in P_{i+1}$.

Second, having established that there must be an element $\pi \in \bigcap_{i \geq 0} P_i$, we prove $\pi$ is lexicographically maximal in $P$. Assume by way of contradiction that there exists an accepting path $\pi'$ so that $h_{\pi'} > h_\pi$. Let $k$ be the first point where $h_{\pi'}$ differs from $h_\pi$. At this point, it must be that $\pi_k$ is not an $F$ node, while $\pi'_k$ is an $F$ node. Therefore it must be that $k > 0$. Further, $\pi'$ is an accepting path that shares a profile with $\pi$ up until this point. As $\pi \in P_{k-1}$ it must also be that $\pi'$ is in $P_{k-1}$. By definition, $P_k$ then would be $\{b \mid b \in P_{k-1},\ b_k$ is an $F$-node$\}$. This would imply $\pi \notin P_k$, a contradiction.

Finally, we demonstrate that every edge in $\pi$ occurs in $G'$. Assume by way of contradiction that some edge $(\pi_i, \pi_{i+1})$ is in $E$ but not in $E'$. This implies there is a node $u$ on level $i$ such that $(u, \pi_{i+1})$ is in $E$ and $\pi_i \prec_i u$. Since $u \in G$, there is an initial path $b$ to $u$. Thus, the path $b, u, \pi_{i+1}, \pi_{i+2} \ldots$ is an accepting path in $G$. This path would be lexicographically larger than $\pi$, contradicting the second claim above. Hence, we conclude $\pi$ is an accepting path in $G'$. □

In the next stage, we remove from $G'$ finite nodes. Let $G'' = G' \setminus \{v \mid v \text{ is finite in } G'\}$. Note there may be nodes that are not finite in $G$, but are finite in $G'$. It is not hard to see that $G$ may have infinitely many $F$-nodes and still not contain a path with infinitely many $F$-nodes. Indeed, $G$ may have infinitely many paths each with finitely many $F$-nodes. We now show that the transition from $G$ via $G'$ to $G''$ removes this possibility, and the presence of infinitely many $F$-nodes in $G''$ does imply the existence of a path with infinitely many $F$-nodes.

**Lemma 5.7.** *$G$ has an accepting path iff $G''$ has infinitely many $F$-nodes.*

*Proof.* If $G$ has an accepting path, then by Lemma 5.6 the DAG $G'$ contains an accepting path. Every node in this path is infinite in $G'$, and thus this path is preserved in $G''$. This path contains infinitely many $F$-nodes, and thus $G''$ contains infinitely many $F$-nodes.

In the other direction, we consider the DAG over equivalence classes induced by $G''$. Given a node $u$ in $G''$, recall that its equivalence class in $G''$ contains all states $v$ such that $v \in G''$ and $h_u = h_v$. Given two equivalence classes $U$ and $V$, recall that $V$ is a child of $U$ when there are $u \in U$, $v \in V$, and $E''(u, v)$. As mentioned above, once we have pruned edges not in $G'$, two nodes of different classes cannot join. Thus this DAG is a tree. Further, as every node $u$ in $G''$ is infinite and has a child, its equivalence class must also have a child. Thus the DAG of classes in $G''$ is a leafless tree. The width of this tree must monotonically increase and is bounded by $n$. It follows that at some level $j$ the tree reaches a stable width. We call this level $j$ the *stabilization level* of $G$.

After the stabilization level, each class $U$ has exactly one child: as noted above, $U$ cannot have zero children, and if $U$ had two children the width of the tree would increase. Therefore, we identify each equivalence class on level $j$ of $G''$ with its unique branch of children in $G''$, which we term its *pipe*. These pipes form a partition of nodes in $G''$ after $j$. Every node in these pipes has an ancestor, or it would not be in the DAG, and has a child, or it would not be infinite and in $G''$. Therefore each node is part of an infinite path in this pipe. Thus, the pipe with infinitely many $F$-classes contains only accepting paths. These paths are accepting in $G$. □

In the proof above we demonstrated there is a *stabilization level* $j$ at which the number of equivalence classes in $G''$ stabilized, and discussed the *pipes* of $G''$: the single chain of descendants from each equivalence class on the stabilization level $j$ of $G''$.

*Example* 5.8. Figure 5.1 displays $G''$ for the example of Figure 2.1. Edges removed from $G'$ are dotted: at levels 2 and 5 where both $q$ and $r$ transition to $r$. When both $r$ and $s$ transition to $t$, they have the same profile and both edges remain. The removed edges render all but the first two $q$-node finite in $G'$. The stabilization level is 1.

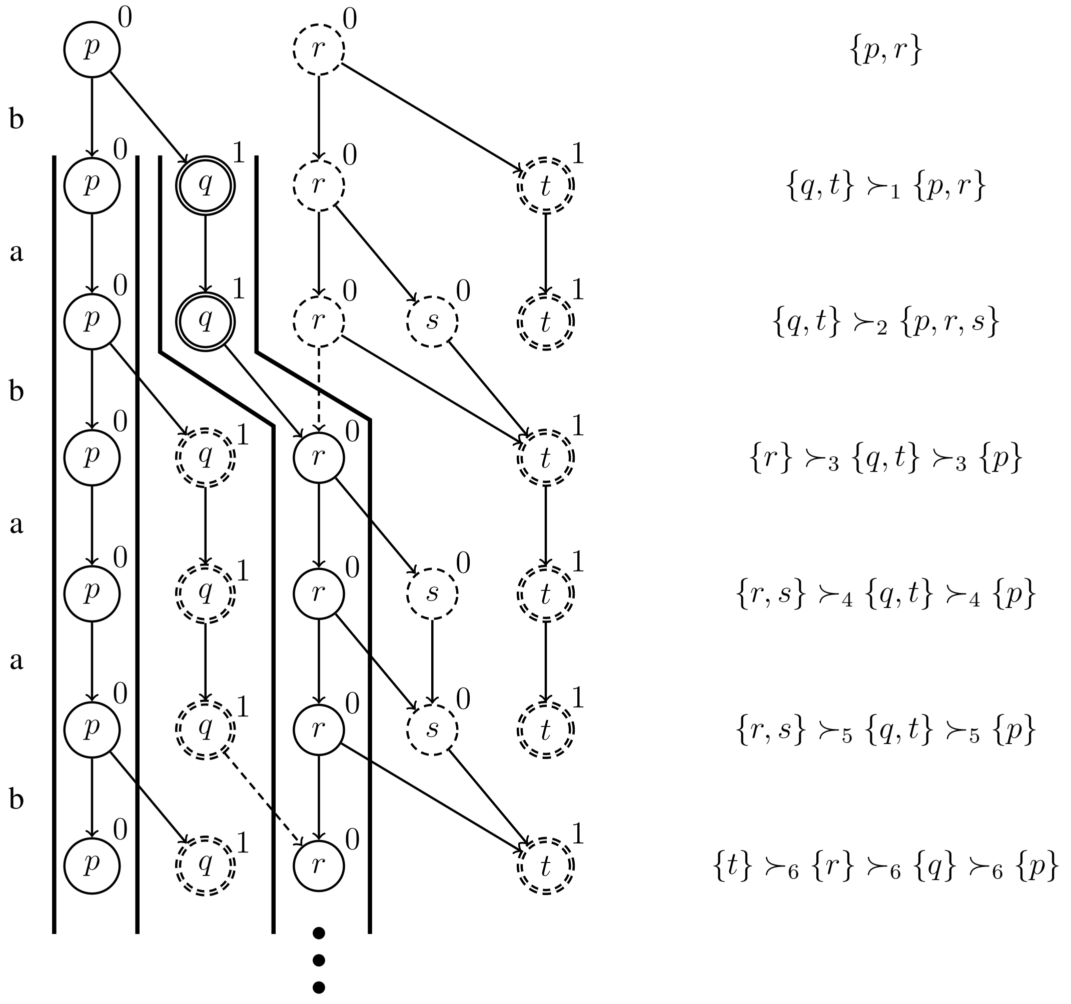

b
a
b
a
a
b
$\{p, r\}$
$\{q, t\} \succ_1 \{p, r\}$
$\{q, t\} \succ_2 \{p, r, s\}$
$\{r\} \succ_3 \{q, t\} \succ_3 \{p\}$
$\{r, s\} \succ_4 \{q, t\} \succ_4 \{p\}$
$\{r, s\} \succ_5 \{q, t\} \succ_5 \{p\}$
$\{t\} \succ_6 \{r\} \succ_6 \{q\} \succ_6 \{p\}$

**Figure 5.1 :** The first seven levels of $G''$, where dashed edges were removed from $G$ and dashed states were removed from $G'$. Nodes are superscripted with their $l$-labels. Bold lines denote the pipes of $G''$. The lexicographic order of equivalence classes for each level of $G'$ is to the right. Edges are removed from $G'$ at levels 2 and 5 where both $q$ and $r$ transition to $r$. When both $r$ and $s$ transition to the same state, they have the same profile and both edges remain. The removed edges render all but the first two $q$-node finite in $G'$. The stabilization level is $1$.

### 5.2.2 Complementing With Profiles

We now complement $\mathcal{B}$ by constructing an NBW, $C^S(\mathcal{B})$, that employs Lemma 5.7 to determine if a word is in $L(\mathcal{B})$. This construction is a reformulation of the slice-based approach of [KW08] in the framework of run DAGs. The NBW $C^S(\mathcal{B})$ tracks the levels of $G'$ and guesses which nodes are finite in $G'$ and therefore do not occur in $G''$. To track $G'$, the automaton $C^S(\mathcal{B})$ stores at each point in time a set $S$ of states that occurs on each level. The sets $S$ are labeled with a guess of which nodes are finite and which are infinite. States that are guessed to be infinite, and thus correspond to nodes in $G''$, are labeled $\top$, and states that are guessed to be finite, and thus omitted from $G''$, are labeled $\bot$. In order to track the edges of $G'$, and thus maintain this labeling, $C^S(\mathcal{B})$ needs to know the lexicographic order of nodes. Thus $C^S(\mathcal{B})$ also maintains the preorder $\preceq_i$ over states on the corresponding level of the run DAG. To enforce that states labeled $\bot$ are indeed finite, $C^S(\mathcal{B})$ employs the cut-point construction of Miyano and Hayashi [MH84], keeping an "obligation set" of states currently being verified as finite. Finally, to ensure the word is rejected, $C^S(\mathcal{B})$ must enforce that there are finitely many $F$-nodes in $G''$. To do so, $S_A$ uses a bit $b$ to guess the level from which no more $F$-nodes appear in $G''$. After this point, $F$-nodes must be labeled $\bot$.

Before we define $C^S(\mathcal{B})$, we formalize *preordered subsets* and operations over them. For a set $Q_\mathcal{B}$ of states, define $\mathbf{Q} = \{\langle S, \preceq\rangle \mid S \subseteq Q_\mathcal{B} \text{ and } \preceq \text{ is a preorder over } S\}$ to be the set of preordered subsets of $Q_\mathcal{B}$. Let $\langle S, \preceq\rangle$ be an element in $\mathbf{Q}$. When considering the successors of a state, we want to consider edges that remain in $G'$. For every state $q \in S$ and $\sigma \in \Sigma$, define

$$\rho_{S,\preceq}(q,\sigma) = \{r \in \rho_\mathcal{B}(q,\sigma) \mid \text{for every } q' \in S, \text{ if } r \in \rho_\mathcal{B}(q',\sigma) \text{ then } q' \preceq q\}$$

Now define the *$\sigma$-successor of* $\langle S, \preceq\rangle$ as the tuple $\langle \rho_{\mathcal{B}}(S,\sigma), \preceq'\rangle$, where for every $q, r \in S$, $q' \in \rho_{S,\preceq}(q,\sigma)$, and $r' \in \rho_{S,\preceq}(r,\sigma)$:

- If $q \prec r$, then $q' \prec' r'$.
- If $q \approx r$ and $q', r' \in F_{\mathcal{B}}$, then $q' \approx' r'$.
- If $q \approx r$ and $q', r' \notin F_{\mathcal{B}}$, then $q' \approx' r'$.
- If $q \approx r$, $q' \notin F_{\mathcal{B}}$, and $r' \in F_{\mathcal{B}}$, then $q' \prec' r'$.

We can now define $C^S(\mathcal{B})$. The states of $C^S(\mathcal{B})$ are tuples $\langle S, \preceq, \lambda, O, b\rangle$ where: $\langle S, \preceq\rangle \in \mathbf{Q}$ is preordered subset of $Q_{\mathcal{B}}$; $\lambda\colon S \to \{\top, \bot\}$ is a labeling indicating which states are guessed to be finite ($\bot$) or infinite ($\top$); $O \subseteq S$ is the obligation set; and $b \in \{0,1\}$ is a bit indicating whether we have seen the last $F$-node in $G''$. To transition between states of $C^S(B)$, say that ***t'*** $= \langle S', \preceq', \lambda', O', b'\rangle$ *follows* ***t*** $= \langle S, \preceq, \lambda, O, b\rangle$ *under* $\sigma$ when:

(1) $\langle S', \preceq'\rangle$ is the $\sigma$-successor of $\langle S, \preceq\rangle$.

(2) $\lambda'$ is such that for every $q \in S$:

- If $\lambda(q) = \top$, then there exists $r \in \rho_{S,\preceq}(q,\sigma)$ such that $\lambda'(r) = \top$,
- If $\lambda(q) = \bot$, then for every $r \in \rho_{S,\preceq}(q,\sigma)$, it holds that $\lambda'(r) = \bot$.

(3) $O' = \begin{cases} \bigcup_{q\in O} \rho_{S,\preceq}(q,\sigma) & O \neq \emptyset, \\ \{q \mid q \in S' \text{ and } \lambda'(q) = \bot\} & O = \emptyset. \end{cases}$

(4) $b' \geq b$.

We want to ensure that runs of $A_S$ reach a suffix where all $F$-nodes are labeled finite. To this end, given a state of $C^S(\mathcal{B})$ $\langle S, \preceq, \lambda, O, b\rangle$, we say that $\lambda$ is *$F$-free* if for every $q \in S \cap F$ we have $\lambda(q) = \bot$.

**Definition 5.9.** For an NBW $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, Q^{in}_{\mathcal{B}}, \rho_{\mathcal{B}}, F_{\mathcal{B}}\rangle$, define $C^S(\mathcal{B})$ be the NBW $\langle \Sigma, Q_S, Q^{in}_S, \rho_S, F_S\rangle$ where:

- $Q_S = \{\langle S, \preceq, \lambda, O, b\rangle \mid \text{if } b = 1 \text{ then } \lambda \text{ is } F\text{-free}\}$,

- $Q^{in}_S = \{\langle Q^{in}_{\mathcal{B}}, \preceq, \lambda, \emptyset, 0\rangle \mid$ for all $q, r \in Q^{in}$, $q \preceq r$ iff $q \notin F_{\mathcal{B}}$ or $r \in F_{\mathcal{B}}\}$,
- $\rho_S(\mathbf{t}, \sigma) = \{\mathbf{t}' \mid \mathbf{t}'$ follows $\mathbf{t}$ under $\sigma\}$, and
- $F_S = \{\langle S, \preceq, \lambda, \emptyset, 1\rangle\}$.

We divide runs of $C^S(\mathcal{B})$ into two parts. The *prefix* of a run is the initial sequence of states in which $b_i$ is $0$, and the *suffix* is the remaining sequence states, in which $b_i$ is $1$. A run without a suffix, where $b$ stays $0$ for the entire run, has no accepting states.

**Lemma 5.10.** *Consider a word $w \in \Sigma^\omega$, the run* DAG *$G$, and an infinite run $\mathbf{t}_0, \mathbf{t}_1, \ldots$ of $C^S(\mathcal{B})$ on $w$. For every $i$, let $\mathbf{t}_i = \langle S_i, \preceq_i, \lambda_i, O_i, b_i\rangle$ and let $\mathcal{S}_i = \langle S_i, \preceq_i\rangle$.*

*(1) The states in $S_i$ are precisely $\{q \mid \langle q, i\rangle \in G\}$.*

*(2) The preorder $\preceq_i$ is the projection of $\preceq$ onto states occurring at level $i$.*

*(3) For every $p \in S_i$, $q \in S_{i+1}$, it holds that $q \in \rho_{\mathcal{S}_i}(p, \sigma_i)$ iff $E'(\langle p, i\rangle, \langle q, i+1\rangle)$.*

*(4) $O_i$ is empty for infinitely many $i$'s iff every state labeled $\perp$ is not in $G''$.*

*(5) Every state labeled $\top$ is in $G''$.*

*This follows from the definition of transitions between states: every $\top$-labeled state must have a $\top$-labeled child, and thus is infinite in $G'$ and in $G''$.*

*Proof.*

(1) This follows from the definition of $\sigma$-successors.

(2) This follows from Lemma 5.5 and the definition of one state in $C^S(\mathcal{B})$ following another.

(3) This follows from the definitions of $E'$ and $\rho_{\mathcal{S}}$.

(4) This follows from the cut-point construction of Miyano and Hayashi. [MH84].

□

**Theorem 5.11.** *For every NBW $\mathcal{B}$, it holds that $L(C^S(\mathcal{B})) = \overline{L(\mathcal{B})}$.*

*Proof.* Consider a word $w \in \Sigma^\omega$ and the run DAG $G$. We first make the following claims about every infinite run $\mathbf{t}_0, \mathbf{t}_1, \ldots$, where $\mathbf{t}_i = \langle S_i, \preceq_i, \lambda_i, O_i, b_i \rangle$. For convenience, define $\mathcal{S}_i = \langle S_i, \preceq_i \rangle$. We exploit Lemma 5.10.(1) to conflate a state $q$ in the $i$th state with the node $\langle q, i \rangle$, and speak of states in $S_i$ being in, being finite in, and being infinite in a graph $G$.

We can now prove the theorem. In one direction, assume there is an accepting run $\mathbf{t}_0, \mathbf{t}_1, \ldots$. As this run is accepting, infinitely often $O_i = \emptyset$. By Lemma 5.10.(4) and (5), this implies the states in $S_i$ are correctly labeled $\top$ when and only when they occur in $G''$. Further, for this run to be accepting we must be able to divide the run into a prefix, and suffix as specified above. In the suffix no state in $F$ can be labeled $\top$, and thus no $F$-nodes occur in $G''$ past this point. As only finitely many $F$-nodes can occur before this point, by Lemma 5.6 $G$ does not have an accepting path and $w \notin L(\mathcal{B})$.

In the other direction, assume $w \notin L(\mathcal{B})$. This implies there are finitely many $F$-nodes in $G''$, and thus a level $j$ where the last $F$-node occurs. We construct an accepting run $\mathbf{t}_0, \mathbf{t}_1, \ldots$, demonstrating along the way that we satisfy the requirements for $\mathbf{t}_{i+1}$ to be in $\rho_S(\mathbf{t}_i, \sigma_i)$. Given $w$, the sequence $\langle S_0, \preceq_0 \rangle, \langle S_1, \preceq_1 \rangle, \ldots$ of preordered subsets is uniquely defined by $\rho_S$. There are many possible labelings $\lambda$. For every $i$, select $\lambda_i$ so that a state $q \in S_i$ is labeled with $\top$ when $\langle q, i \rangle \in G''$, and $\perp$ when it is not. Since every node in $G''$ has a child, by Lemma 5.10.(3), for every $p \in S_i$ where $\lambda_i(p) = \top$, there exist a $q \in \rho_{\mathcal{S}_i}(p, \sigma_i)$ so that $\lambda_{i+1}(q) = \top$. Further, every node labeled $\perp$ has only finitely many descendants, and so for every $p \in S_i$ where $\lambda_i(p) = \perp$ and $q \in \rho_{\mathcal{S}_i}(p, \sigma_i)$, it holds that $\lambda_{i+1}(q) = \perp$. Therefore the transition from $\lambda_i$ to $\lambda_{i+1}$ satisfies the requirements of $\rho_S$. The set $O_0 = \emptyset$, and given the sets $S_i$ and labelings $\lambda_i$, the sets $O_{i+1}$, $i \geq 0$ are again uniquely defined by $\rho_S$. Finally, we choose $b_i = 0$ when $i < j$, and $b_i = 1$ for $i \geq j$. Since there are no $F$-nodes in $G''$ past $j$, no $F$-node will be labeled $\top$ and all states past

$j$ will be $F$-free. We have satisfied the last requirement for the transitions from every $\mathbf{t}_i$ to $\mathbf{t}_{i+1}$ to be valid, rendering this sequence a run. By Lemma 5.10.(4), infinitely often $O_i = \emptyset$, including infinitely often after $j$, thus there are infinitely many states $\mathbf{t}_i$ where $b_i = 1$ and $O_i = \emptyset$, and this run is accepting. $\square$

If $n = |Q_{\mathcal{B}}|$, the number of preordered subsets is roughly $(0.53n)^n$ [Var80]. As there are $2^n$ labelings, and a further $2^n$ obligation sets, the state space of $\mathcal{B}_s$ is at most $(2n)^n$. The slice-based automaton obtained in [KW08] coincides with $C^S(\mathcal{B})$, modulo the details of labeling states and the cut-point construction. Whereas the correctness proof in [KW08] is given by means of reduced split trees, here we proceed directly on the run DAG.

## 5.3 Retrospection

Consider an NBW $\mathcal{B}$. So far, we presented two complementation constructions for $\mathcal{B}$, generating the NBWs $C^R_m(\mathcal{B})$ and $C^S(\mathcal{B})$. In this section we present a third construction, generating an NBW that combines the benefits of the two constructions above. Both constructions refer to the run DAG of $\mathcal{B}$. In the rank-based approach applied in $C^R_m(\mathcal{B})$, the ranks assigned to a node bound the visits in accepting states yet to come. Thus, the ranks refer to the future, making $C^R_m(\mathcal{B})$ inherently nondeterministic. On the other hand, the NBW $C^S(\mathcal{B})$ refers to both the past, using profiles to prune edges from $G$, as well as to the future, by keeping in $G''$ only nodes that are infinite in $G'$. Guessing which nodes are infinite and labeling them $\top$ inherently introduces nondeterminism into the automaton.

Our first goal in the combined construction is to reduce this latter nondeterminism. Recall that a labeling is $F$-free if all the states in $F$ are labeled $\perp$. Observe that the fewer labels of $\perp$ (finite nodes) we have, the more difficult it is for a labeling to be $F$-free and, consequently, the more difficult it is for a run of $C^S(\mathcal{B})$ to proceed to the $F$-free suffix in which $b = 1$. It is therefore safe for $C^S(\mathcal{B})$ to underestimate which nodes to label $\bot$, as long as the requirement to reach an $F$-free suffix is maintained. We use this observation in order to introduce a purely retrospective construction.

For a run DAG $G$, say that a level $k$ is an *$F$-finite level* of $G$ when all $F$-nodes *after* level $k$ (i.e. on a level $k'$ where $k' > k$) are finite in $G'$. By Lemma 5.7, $G$ is rejecting iff there is a level after which $G''$ has no $F$-nodes. As finite nodes in $G'$ are removed from $G''$, we have:

**Corollary 5.12.** *A run* DAG *$G$ is rejecting iff it has an $F$-finite level.*

### 5.3.1 Retrospective Labeling

The labeling function $\lambda$ used in the construction of $C^S(\mathcal{B})$ labels nodes by $\{\top, \bot\}$, with $\bot$ standing for "finite" and $\top$ standing for "infinite". In this section we introduce a variant of $\lambda$ that again maps nodes to $\{\top, \bot\}$ except that now $\top$ stands for "unrestricted", allowing us to underestimate which nodes to label $\bot$. To capture the relaxed requirements on labelings, say that a labeling $\lambda$ is *legal* when every $\bot$-labeled node is finite in $G'$. This enables the automaton to track the labeling and its effect on $F$-nodes only after it guesses that an $F$-finite level $k$ has been reached: all nodes *at or before* level $k$ (i.e. on a level $k'$ where $k' \leq k$) are unrestricted, whereas $F$-nodes after level $k$ and their descendants are required to be finite. The only nondeterminism in the automaton lies in guessing when the $F$-finite level has been reached. This reduces the branching degree of the automaton to 2, and renders it deterministic in the limit.

The suggested new labeling is parametrized by the $F$-finite level $k$. The labeling $\lambda^k$ is defined inductively over the levels of $G$. Let $S_i$ be the set of nodes on level $i$ of $G$. For $i \geq 0$, the function $\lambda^k \colon S_i \to \{\top, \bot\}$ is defined as follows:

- If $i \leq k$, then for every $u \in S_i$ we define $\lambda^k(u) = \top$.
- If $i > k$, then for every $u \in S_i$:
  - If $u$ is an $F$-node, then $\lambda^k(u) = \bot$.
  - Otherwise, $\lambda^k(u) = \lambda^k(v)$, for a node $v$ where $E'(v, u)$.

For $\lambda^k$ to be well defined when $i > k$ and $u$ is not an $F$-node, we need to show that $\lambda^k(u)$ does not depend on the choice of the node $v$ where $E'(v, u)$ holds. By Lemma 5.3, all parents of a node in $G'$ belong to the same equivalence class. Therefore, it suffices to prove that all nodes in the same class share a label: for all nodes $u$ and $u'$, if $u' \approx_{|u|} u$ then $\lambda^k(u) = \lambda^k(u')$. The proof proceeds by an induction on $i = |u|$. Consider two nodes $u$ and $u'$ on level $i$ where $u' \approx_i u$. As a base case, if $i \leq k$, then $u$ and $u'$ are labeled $\top$. For $i > k$, if $u$ is an $F$-node, then $u'$ is also an $F$-node and $\lambda^k(u) = \lambda^k(u') = \bot$. Finally, if $u$ and $u'$ are both non-$F$-nodes, recall that all parents of $u$ are in the same equivalence class $V$. As $u \approx_i u'$, Lemma 5.3 implies that all parents of $u'$ are also in $V$ By the induction hypothesis, all nodes in $V$ share a label, and thus $\lambda^k(u) = \lambda^k(u')$.

**Lemma 5.13.** *For a run* DAG *$G$ and $k \in \mathbb{N}$, the labeling $\lambda^k$ is legal iff $k$ is an $F$-finite level for $G$.*

*Proof.* If $\lambda^k$ is legal, then every $\bot$-labeled node is finite in $G'$. Every $F$-node after level $k$ (i.e. on a level $i$ where $i > k$) is labeled $\bot$, and thus $k$ is an $F$-finite level for $G$. If $\lambda^k$ is not legal, then there is a $\bot$-labeled node $u$ that is infinite in $G'$. Every ancestor of $u$ is also infinite. Let $u'$ be the earliest ancestor of $u$ (possibly $u$ itself) so that $\lambda^k(u') = \bot$. Observe that only nodes after level $k$ can be $\bot$-labeled, and so $u'$ is on a level $i > k$. It must be that $u'$ is an $F$-node: otherwise it would inherit its parents' label, and by assumption the parents of $u'$ are $\top$-labeled. Thus, $u'$ is an $F$-node after level $k$ that is infinite in $G'$, and $k$ is not an $F$-finite level for $G$. $\square$

**Corollary 5.14.** *A run* DAG *$G$ is rejecting iff, for some $k$, the labeling $\lambda^k$ is legal.*

### 5.3.2 From Labelings to Rankings

In this section we derive an odd ranking for $G$ from the function $\lambda^k$, thus unifying the retrospective analysis behind $\lambda^k$ with the rank-based analysis of [KV01]. Consider again the DAG $G'$ and the function $\lambda^k$. Recall that every equivalence class $U$ has at most two child equivalence classes, one $F$-class and one non-$F$-class. After the $F$-finite level $k$, only non-$F$-classes can be labeled $\top$. Hence, after level $k$, every $\top$-labeled equivalence class $U$ can only have a one child that is $\top$-labeled. For every class $U$ on level $k$, we consider this possibly infinite sequence of $\top$-labeled non-$F$-children. The odd ranking we are going to define, termed the *retrospective ranking*, gives these sequences of $\top$-labeled children odd ranks. The $\perp$-labeled classes, which lie between these sequences of $\top$-labeled classes, are assigned even ranks. The ranks increase in inverse lexicographic order, i.e. the maximal $\top$-labeled class in a level is given rank 1. As with $\lambda^k$, the retrospective ranking is parametrized by $k$. The primary insight that allows this ranking is that there is no need to distinguish between two adjacent $\perp$-labeled classes. Formally, we have the following.

**Definition 5.15** ($k$-retrospective ranking)**.** Consider a run DAG $G$, $k \in \mathbb{N}$, and a labeling $\lambda^k \colon G \to \{\top, \perp\}$. Let $m = 2|Q_{\mathbb{B}} \setminus F_{\mathbb{B}}|$. For a node $u$ on level $i$ of $G$, define $\alpha(u)$ to be the number of $\top$-labeled classes on level $i$ lexicographically larger than $u$: $\alpha(u) = |\{[v] \mid \lambda^k(v) = \top \text{ and } u \prec_i v\}|$. The *k-retrospective ranking* of $G'$ is the function $\mathbf{r}^k \colon V \to \{0..m\}$ defined for every node $u$ on level $i$ as follows.

$$\mathbf{r}^k(u) = \begin{cases} m & \text{if } i \leq k, \\ 2\alpha(u) & \text{if } i > k \text{ and } \lambda^k(u) = \perp, \\ 2\alpha(u) + 1 & \text{if } i > k \text{ and } \lambda^k(u) = \top. \end{cases}$$

Note that $\mathbf{r}^k$ is tight. As defined in Section 2.2.1, a ranking is tight if there exists an $i \in \mathbb{N}$ such that, for every level $l \geq i$, all odd ranks below $max\_rank(\mathbf{r}, l)$ appear on level $l$. For $\mathbf{r}^k$ this level is $k + 1$, after which each $\top$-labeled class is given the odd rank greater by two than the rank of the next lexicographically larger $\top$-labeled class.

**Lemma 5.16.** *For every $k \in \mathbb{N}$, the following hold:*

*(1) If $u \prec_{|u|} u'$ then $\mathbf{r}^k(u) \geq \mathbf{r}^k(u')$.*

*(2) If $(u, v) \in E'$, then $\mathbf{r}^k(u) \geq \mathbf{r}^k(v)$.*

*Proof.* As both claims are trivial when $u$ is at or before level $k$, assume $u$ is on level $i > k$. To prove the first claim, note that $\alpha(u) \geq \alpha(u')$: every class, $\top$-labeled or not, that is larger than $u'$ must also be larger than $u$. If $\alpha(u) > \alpha(u')$, then (1) follows immediately. Otherwise $\alpha(u) = \alpha(u')$, which implies that $\lambda^k(u') = \bot$: otherwise $[u']$ would be a $\top$-labeled equivalence class larger than $u$, but not larger than itself. Thus $\mathbf{r}^k(u') = 2\alpha(u)$, and $\mathbf{r}^k(u) \in \{2\alpha(u),\ 2\alpha(u)+1\}$ is at least $\mathbf{r}^k(u')$.

As a step towards proving the second claim, we show that $\alpha(u) \geq \alpha(v)$. Consider every $\top$-labeled class $[v']$ where $v \prec_{i+1} v'$. The class $[v']$ must have a $\top$-labeled parent $[u']$. Since $v \prec_{i+1} v'$, the contrapositive of Lemma 5.5, part 1, entails that $u \preceq_i u'$. By the definition of $\lambda^k$, the class $[u']$ can only have one $\top$-labeled child class: $[v']$. We have thus established that for every $\top$-labeled class larger than $v$, there is a unique $\top$-labeled class larger than $u$, and can conclude that $\alpha(u) \geq \alpha(v)$. We now show by contradiction that $\mathbf{r}^k(u) \geq \mathbf{r}^k(v)$. For $\mathbf{r}^k(u) < \mathbf{r}^k(v)$, it must be that $\alpha(u) = \alpha(v)$, that $\mathbf{r}^k(u) = 2\alpha(u)$, and that $\mathbf{r}^k(v) = 2\alpha(u) + 1$. In this case, $\lambda^k(u) = \bot$ and $\lambda^k(v) = \top$. Since a $\bot$-labeled node cannot have a $\top$-labeled child in $G'$, this is impossible. □

When $k$ is an $F$-finite level of $G$, the $k$-retrospective ranking is an $m$-bounded odd ranking.

**Lemma 5.17.** *For a run* DAG $G$ *and* $k \in \mathbb{N}$*, the function* $\mathbf{r}^k$ *is a ranking bounded by* $m$*. Further, if the labeling* $\lambda^k$ *is legal then* $\mathbf{r}^k$ *is an odd ranking.*

*Proof.* There are three requirements for $\mathbf{r}^k$ to be a ranking bounded by $m$:

(1) *Every $F$-node must have an even rank.* At or before level $k$, every node has rank $m$, which is even. After $k$ only $\top$-labeled nodes are given odd ranks, and every $F$-node is labeled $\bot$.

(2) *For every $(u, v) \in E$, it must hold that $\mathbf{r}^k(u) \geq \mathbf{r}^k(v)$.* If $u$ is at or before level $k$, then it has the maximal rank of $m$. If $u$ is after level $k$, we consider two cases: edges in $E'$, and edges in $E \setminus E'$. For edges in $E'$, this follows from Lemma 5.16 (2). For edges $(u, v) \in E \setminus E'$, we know there exists a $u'$ where $u \prec_{|u|} u'$ and $(u', v) \in E'$. By Lemma 5.16, $\mathbf{r}^k(u) \geq \mathbf{r}^k(u') \geq \mathbf{r}^k(v)$.

(3) *The rank is bounded by $m$.* No $F$-node can be $\top$-labeled. Thus the maximum number of $\top$-labeled classes on every level is $|Q_{\mathcal{B}} \setminus F_{\mathcal{B}}|$. The largest possible rank is given to a node smaller than all $\top$-labeled classes, which must be be a $F$-node and $\bot$-labeled. Thus, this node is given a rank of at most $m = 2|Q_{\mathcal{B}} \setminus F_{\mathcal{B}}|$.

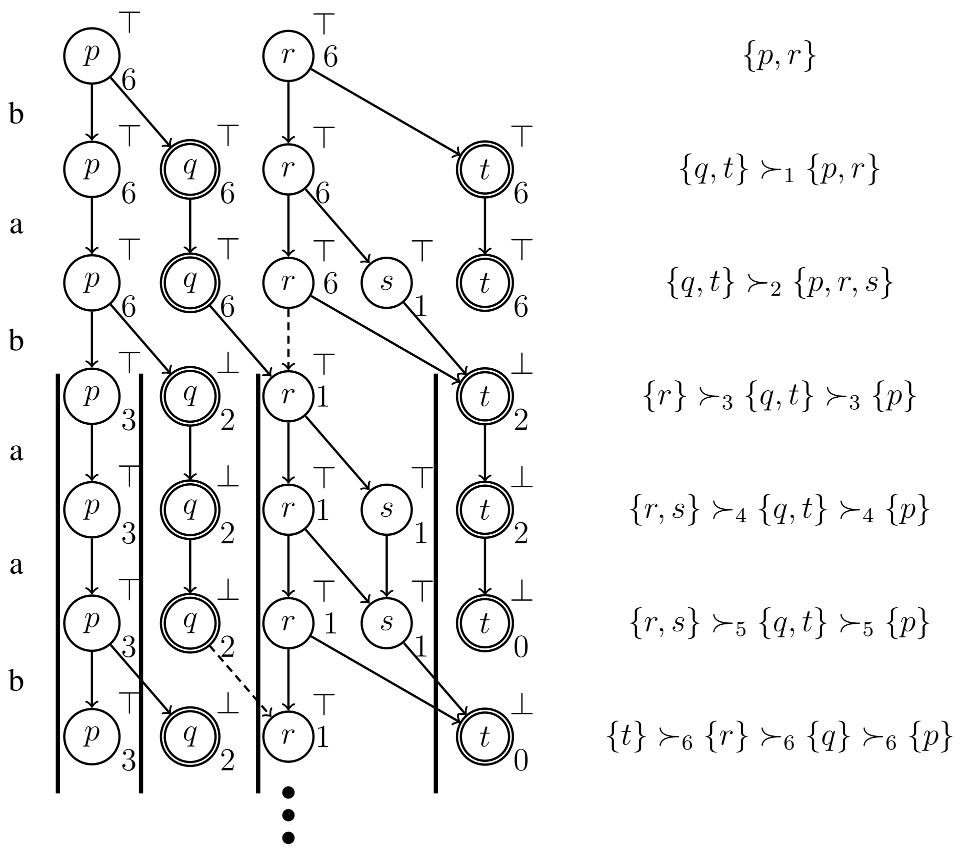
It remains to show that if $\lambda^k$ is legal, then $\mathbf{r}^k$ is an odd ranking. Consider an infinite path $u_0, u_1, \ldots$ in $G$. We demonstrate that for every $i > k$ such that $\mathbf{r}^k(u_i)$ is an even rank $e$, there exists $i' > i$ such that $\mathbf{r}^k(u_{i'}) \neq e$. Since a path cannot increase in rank, this implies $\mathbf{r}^k(u_{i'}) < e$. To do so, define the sequence $U_i, U_{i+1}, \ldots$, of sets of nodes inductively as follows. Let $U_i = \{v \mid \mathbf{r}^k(v) = e\}$. For every $j \geq i$, let $U_{j+1} = \{v \mid v' \in U_j,\ (v', v) \in E'\}$. As $\mathbf{r}^k(v)$ is even only when $\lambda^k(v) = \bot$, if $\lambda^k$ is legal then every node given an even rank (such as $e$) must be finite in $G'$. Therefore every element of $U_i$ is finite in $G'$, and thus at some $i' > i$, the set $U_{i'}$ is empty. Since $U_{i'}$ is empty, to establish that $\mathbf{r}^k(u_{i'}) \neq e$, it is sufficient to prove that for every $j$, if $\mathbf{r}^k(u_j) = e$, then $u_j \in U_j$.

To show that $\mathbf{r}^k(u_j) = e$ entails $u_j \in U_j$, we prove a stronger claim: for every $j \geq i$ and $v$ on level $j$, if $u_j \preceq_j v$ and $\mathbf{r}^k(v) = e$, then $v \in U_j$. We proceed by induction over $j$. For the base case of $j = i$, this follows from the definition of $U_i$. For the inductive step, take a node $v$ on level $j+1$ where $\mathbf{r}^k(v) = e$ and $u_{j+1} \preceq_{j+1} v$. We consider two cases. If $\mathbf{r}^k(u_{j+1}) \neq e$ then the path from $u_i$ to $u_{j+1}$ entails that $\mathbf{r}^k(u_{j+1}) < e$, and this case of the subclaim follows from Lemma 5.16 (1). Otherwise, it holds that $\mathbf{r}^k(u_{j+1}) = e$, and thus $\mathbf{r}^k(u_j) = e$. Let $u'$ and $v'$ be nodes on level $j$ so that $(u', u_{j+1}) \in E'$ and $(v', v) \in E'$. As $u_{j+1} \preceq_{j+1} v$, the contrapositive of Lemma 5.5, part 1, entails that $u' \preceq_j v'$. Further, since $(u', u_{j+1}) \in E'$ and $(u_j, u_{j+1}) \in E$, we know $u_j \preceq_j u'$. By transitivity we can thus conclude that $u_j \preceq_j v'$, which along with Lemma 5.16 (1) entails $\mathbf{r}^k(u') = e \geq \mathbf{r}^k(v')$. As $(v', v) \in E$, Lemma 5.16 (2) entails that $\mathbf{r}^k(v') \geq \mathbf{r}^k(v) = e$. Thus $\mathbf{r}^k(v') = e$, and by the inductive hypothesis $v' \in U_j$. As $E'(v', v)$ holds, by definition $v \in U_{j+1}$, and our subclaim is proven. $\square$

The ranking of Definition 5.15 is termed *retrospective* as it relies on the relative lexicographic order of equivalence classes; this order is determined purely by the history of nodes in the run DAG, not by looking forward to see which descendants are infinite or $F$-free in some subgraph of $G$.

*Example* 5.18. Figure 5.2 displays $\lambda^2$ and the 2-retrospective ranking of our running example. In the prospective ranking (Figure 2.2), the nodes for state $t$ on levels $3$ and $4$ are given rank $0$, like other $t$-nodes. In the absence of a path forcing this rank, their retrospective rank is $2$.

We are now ready to define a new construction, generating an NBW $C^U(\mathcal{B})$, which combines the benefits of the previous two constructions. The automaton $C^U(\mathcal{B})$ guesses the $F$-finite level $k$, and uses level rankings to check if the $k$-retrospective ranking is an odd ranking. We partition the operation of $C^U(\mathcal{B})$ into two stages. Until the level $k$, the



b
a
b
a
a
b
$\{p, r\}$
$\{q, t\} \succ_1 \{p, r\}$
$\{q, t\} \succ_2 \{p, r, s\}$
$\{r\} \succ_3 \{q, t\} \succ_3 \{p\}$
$\{r, s\} \succ_4 \{q, t\} \succ_4 \{p\}$
$\{r, s\} \succ_5 \{q, t\} \succ_5 \{p\}$
$\{t\} \succ_6 \{r\} \succ_6 \{q\} \succ_6 \{p\}$

**Figure 5.2 :** The run DAG $G'$, where $2$ is an $F$-finite level. Dashed edges were removed from $G'$. The labels of $\lambda^2$ and ranks in $\mathbf{r}^2$ are displayed as superscripts and subscripts, respectively. The bold lines display the sequences of $\top$-labeled classes in $G'$. The lexicographic order of states is repeated on the right.

NBW $C^U(\mathcal{B})$ is in the first stage, where it deterministically tracks preordered subsets. After level $k$, the NBW $C^U(\mathcal{B})$ moves to the second stage, where it tracks ranks. This stage is also deterministic. Consequently, the only nondeterminism in $C^U(\mathcal{B})$ is indeed the guess of $k$. Before defining $C^U(\mathcal{B})$, we need some definitions and notations.

Lemma 5.17 gives us an alternative odd ranking to the one of [KV01]. We now provide an NBW that uses level rankings to guess this retrospective ranking. In comparison to the slice-based construction in Definition 5.9, employing level rankings gives us a tighter bound and affords further improvements, discussed later. In comparison to the rank-based construction in Definition 2.3, this automaton needs make only a single guess on each accepting run: what is the $F$-finite level $k$. Before and after this guess we proceed deterministically. Before $k$, our automaton tracks preordered subsets. At level $k$ our automaton moves to ranks, and these ranks proceed stably. We thus obtain an automaton with a linear-sized transition relation. We now introduce the machinery to define the automaton.

Recall that $\mathbf{Q}$ denotes the set of preordered subsets of $Q_\mathcal{B}$, and $\mathcal{R}^m_T$ the set of tight level rankings bounded by $m$. We distinguish between three types of transitions of $C^U(\mathcal{B})$: transitions within the first stage, transitions from the first stage to the second, and transitions within the second stage. The first type of transition is similar to the one taken in $C^S(\mathcal{B})$, by means of the $\sigma$-successor relation between preordered subsets. Below we explain in detail the other two types of transitions. Recall that in the retrospective ranking $\mathbf{r}^k$, each class in $G'$ labeled $\top$ by $\lambda^k$ is given a unique odd rank. Thus the rank of a node $u$ depends on the number of $\top$-labeled classes larger than it, denoted $\alpha(u)$.

We begin with transitions where $C^U(\mathcal{B})$ moves between the stages: from a preordered subset $\langle S, \preceq \rangle$ to a level ranking. On level $k+1$, a node is labeled $\top$ iff it is an non-$F$-node. Thus for every $q \in S$, let $\beta(q) = |\{[v] \mid v \in S \setminus F_{\mathcal{B}}, u \prec v\}|$ be the number of non-$F$-classes larger than $q$. We now define $\texttt{torank}\colon \mathbf{Q} \to \mathcal{R}^m_T$. Let $\texttt{torank}(\langle S, \preceq\rangle)$ be the tight level ranking $f$ where for every $q$:

$$f(q) = \begin{cases} \bot & \text{if } q \notin S, \\ 2\beta(q) & \text{if } q \in S \cap F_{\mathcal{B}}, \\ 2\beta(q)+1 & \text{if } q \in S \setminus F_{\mathcal{B}}. \end{cases}$$

We now turn to transitions within the second stage, between level rankings. The rank of a node $v$ is inherited from its predecessor $u$ in $G'$. However, $\lambda^k$ may label a finite class $\top$. If a $\top$-labeled class larger than $u$ has no children, then $\alpha(u) \geq \alpha(v)$. In this case the rank of $v$ decreases. Given a level ranking $f$, for every $q \in Q_{\mathcal{B}}$ where $f(q) \neq \bot$, let $\gamma(q) = |\{f(q') \mid q' \in Q_{\mathcal{B}},\ f(q') \text{ is odd},\ f(q') < f(q)\}|$ be the number of odd ranks in the range of $f$ lower than $f(q)$. We define the function $\texttt{tighten}\colon \mathcal{R}^m \to \mathcal{R}^m_T$. Let $\texttt{tighten}(f)$ be the tight level ranking $f'$ where for every $q$:

$$f'(q') = \begin{cases} \bot & \text{if } f(q) = \bot, \\ 2\gamma(q) & \text{if } f(q) \neq \bot \text{ and } q \in F_{\mathcal{B}}, \\ 2\gamma(q)+1 & \text{if } f(q) \neq \bot \text{ and } q \notin F_{\mathcal{B}}. \end{cases}$$

Note that if $f$ is tight, then $f' = f$, and that while $\texttt{tighten}$ may merge two even ranks, it cannot merge two odd ranks.

For a level ranking $f$, letter $\sigma \in \Sigma$, and state $q' \in Q_{\mathcal{B}}$, define $\texttt{pred}(q', \sigma, f) = \{q \mid f(q) \neq \bot,\ q' \in \rho_{\mathcal{B}}(q, \sigma)\}$ be the predecessors of $q'$ given a non-$\bot$ rank by $f$. The lowest ranked element in this set corresponds to the predecessor in $G$ with the maximal profile. With two exceptions, $q'$ will inherit this lowest rank. First, $\texttt{tighten}$ might shift the rank down. Second, if $q'$ is in $F$, it cannot be given an odd rank. For $n \in \mathbb{N}$, let $\lfloor n \rfloor_{even}$ be: $n$ when $n$ is even; and $n-1$ when $n$ is odd. Define the *$\sigma$-successor of $f$* to be $\texttt{tighten}(f')$ where for every $q' \in Q_{\mathcal{B}}$:

$$f'(q') = \begin{cases} \bot & \text{if } \texttt{pred}(q', \sigma, f) = \emptyset, \\ \lfloor \texttt{min}(\{f(q) \mid q \in \texttt{pred}(q', \sigma, f)\}) \rfloor_{even} & \text{if } \texttt{pred}(q', \sigma, f) \neq \emptyset \text{ and } q' \in F, \\ \texttt{min}(\{f(q) \mid q \in \texttt{pred}(q', \sigma, f)\}) & \text{if } \texttt{pred}(q', \sigma, f) \neq \emptyset \text{ and } q' \notin F. \end{cases}$$

**Definition 5.19.** For an NBW $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, Q^{in}_{\mathcal{B}}, \rho_{\mathcal{B}}, F_{\mathcal{B}} \rangle$, let $C^U(\mathcal{B})$ be the NBW $\langle \Sigma, \mathbf{Q} \cup (\mathcal{R}^m_T \times 2^Q), Q^{in}_U, \rho_U, \mathcal{R}^m_T \times \{\emptyset\} \rangle$, where

- $Q^{in}_L = \{\langle Q^{in}_{\mathcal{B}}, \preceq^{in} \rangle\}$ where $\preceq^{in}$ is such that for all $q, r \in Q^{in}_{\mathcal{B}}$, $q \preceq r$ iff $q \notin F_{\mathcal{B}}$ or $r \in F_{\mathcal{B}}$.
- $\rho_L(\mathcal{S}, \sigma) = \{\mathcal{S}'\} \cup \{\langle \texttt{torank}(\mathcal{S}'), \emptyset \rangle\}$, where $\mathcal{S}'$ is the $\sigma$-successor of $\mathcal{S}$.
- $\rho_L(\langle f, O \rangle, \sigma) = \{\langle f', O' \rangle\}$ where $f'$ is the $\sigma$-successor of $f$

$$\text{and } O' = \begin{cases} \rho_{\mathcal{B}}(O, \sigma) \setminus odd(f') & \text{if } O \neq \emptyset, \\ even(f') & \text{if } O = \emptyset. \end{cases}$$

**Lemma 5.20.** *Consider a word $w \in \Sigma^\omega$, the run $\dagger G$ of $\mathcal{B}$ on $w$, and an infinite run $\langle S_0, \preceq_0 \rangle, \ldots, \langle S_k, \preceq_k \rangle, \langle f_{k+1}, O_{k+1} \rangle, \langle f_{k+2}, O_{k+2} \rangle, \ldots$ of $C^U(\mathcal{B})$ on $w$. For $i > k$, define $S_i = \{q \mid f_i(q) \neq \bot\}$. The following hold:*

*(1) The states in $S_i$ are precisely $\{q \mid \langle q, i \rangle \in G\}$.*

*(2) The preorder $\preceq_i$ is the projection of $\preceq$ onto states occurring at level $i$.*

*(3) For every $i \leq k$, state $q \in S_i$, and $s \in S_{i+1}$, it holds that $s \in \rho_{\langle S_i, \preceq_i \rangle}(q, \sigma_i)$ iff $E'(\langle q, i \rangle, \langle s, i+1 \rangle)$.*

*(4) For every $i > k$ and $q, s \in S_i$, if $f_i(q) > f_i(s)$, then $\langle q, i \rangle \prec_i \langle s, i \rangle$.*

*(5) For every $i > k$ and $q, s \in S_i$, if $f_i(s)$ is odd and $\langle q, i \rangle \prec_i \langle s, i \rangle$, then $f_i(q) >$*

$f_i(s)$.

(6) *For every $i \geq k$ and $q \in S_i$, it holds that $f_i(q)$ is even iff $\lambda^k(\langle q, i\rangle) = \perp$.*

*Proof.*

(1) This follows by the definitions of $\sigma$-successors of preordered subsets and $\sigma$-successors of level rankings.

(2) This follows from Lemma 5.5 and the definition of $\sigma$-successors.

(3) This follows from the definitions of $E'$ and $\rho_{\langle S_i, \preceq_i\rangle}$.

(4),(5) We simultaneously prove (4) and (5) by induction. As a base case, both hold from the definition of `torank`. As the inductive step, assume both hold for level $i$. To prove step (4), take two states $q, s \in S_{i+1}$ where $f_{i+1}(q) > f_{i+1}(s)$. Each state has a parent in $G'$, i.e. a $q'$ and $s'$ so that $E'(q', q)$ and $E'(s', s)$. By the inductive hypothesis, this implies $f_i(q') = \mathtt{min}(\{f_i(q') \mid q \in \rho_{\mathcal{B}}(q', \sigma_i)\})$ and $f_i(s') = \mathtt{min}(\{f_i(s') \mid s \in \rho_{\mathcal{B}}(s', \sigma_i)\})$. We analyze two cases. When $f_i(q') > f_i(s')$, by the inductive hypothesis we have $\langle q', i\rangle \prec_i \langle s', i\rangle$. Since $E'(q', q)$ and $E'(s', s)$, by Lemma 5.3 this implies $\langle q, i+1\rangle \prec_{i+1} \langle s, i+1\rangle$. Alternately, when $f_i(q') = f_i(s')$, then for $f_{i+1}(q) > f_{i+1}(s)$ to hold, it must be that $f_i(q')$ is odd, $s \in F$, and $q \notin F$. Since $f_i(q') = f_i(s')$ is odd, by the inductive hypothesis we have that $\langle q', i\rangle \equiv \langle s', i\rangle$. By Lemma 5.3 we then have $h_{\langle q, i+1\rangle} = h_{\langle q', i\rangle}0 < h_{\langle s, i+1\rangle} = h_{\langle s', i\rangle}1$.

To prove step (5), consider when $f_{i+1}(s)$ is odd and $\langle q, i+1\rangle \prec \langle s, i+1\rangle$. This implies that $h_{\langle s, i+1\rangle} = h_{\langle s', i\rangle}0$. Thus in order for $\langle q, i+1\rangle \prec_{i+1} \langle s, i+1\rangle$ to hold, $\langle q', i\rangle \prec_i \langle s', i\rangle$ must hold. By the inductive hypothesis, this implies $f_i(q') > f_i(s')$. Before the `tighten` function reduces ranks, since $f_{i+1}(q) = \lfloor f_i(q')\rfloor_{even}$, and $f_{i+1}(s)$ is odd, it must be that $f_{i+1}(q) > f_{i+1}(s)$. The `tighten` function can shift $f_{i+1}(q)$ down more than $f_{i+1}(s)$ only when an odd rank between $f_{i+1}(s)$ and $f_{i+1}(q)$ becomes empty. Since this odd rank must be two greater than $f_{i+1}(s)$, reducing $f_{i+1}(q)$ by 2 cannot change that $f_{i+1}(q) > f_{i+1}(s)$. We now proceed with the proof of Theorem 5.21.

(6) This follows from the definition of $\lambda^k$, which assigns $\perp$ to $F$-nodes and their descendants in $G'$, and $f_i$, which assigns even ranks to states in $F$. By (4), the parent of a node in $G'$ will be the parent with the lowest rank. Thus the descendants of $F$-nodes in $G'$ will inherit the even rank of their parent.

□

The proof of Theorem 5.21 is based on from Lemmas 2.2, 5.17, 5.20 and Corollary 5.14.

**Theorem 5.21.** *For every NBW $\mathcal{B}$, it holds that $L(C^U(\mathcal{B})) = \overline{L(\mathcal{B})}$.*

*Proof.* Consider a word $w \in \Sigma^\omega$ and the run DAG $G$. In one direction, assume there exists an accepting infinite run $\langle S_0, \preceq_0\rangle, \ldots, \langle S_k, \preceq_k\rangle, \langle f_{k+1}, O_{k+1}\rangle, \langle f_{k+2}, O_{k+2}\rangle, \ldots$ of $C^U(\mathcal{B})$ on $w$. For every $i > k$, define $S_i = \{q \mid f_i(q) \neq \perp\}$. We construct a ranking $\mathbf{r}$ of $G$ as follows. For all nodes $u$ on level $i \leq k$, $\mathbf{r}(u) = m$. For all nodes $\langle q, i\rangle$ where $i > k$, $\mathbf{r}(\langle q, i\rangle) = f_i(q)$. We note that each state is given at most the minimum rank of all its parents, and that no state in $F$ is given an odd rank, thus $\mathbf{r}$ is in fact a ranking. That $\mathbf{r}$ is an odd ranking follows from the cut-point construction.

In the other direction, assume $G$ is a rejecting run DAG. By Lemma 5.17 there exists a $k$ so that $\mathbf{r}^k$ is an odd ranking. We construct a run $\mathcal{S}_0, \ldots, \mathcal{S}_k, \langle f_{k+1}, O_{k+1}\rangle, \langle f_{k+2}, O_{k+2}\rangle, \ldots$, which is uniquely defined by the transition relation of Definition 5.19. Further, the transition relation of Definition 5.19 is total, so this run is infinite. To demonstrate that this run is accepting, we will prove below that for every $i > k$ and $q \in S_i$, it holds that

$f_i(q) = \mathbf{r}^k(\langle q, i\rangle)$. Since $\mathbf{r}^k$ is an odd ranking and the cut-point construction is identical to that of Definition 2.3, this is sufficient to show the run is accepting.

Recall that if $\lambda(\langle q, i\rangle) = \bot$, then $\mathbf{r}^k(\langle q, i\rangle) = 2\alpha(\langle q, i\rangle)$, and otherwise $\mathbf{r}^k(\langle q, i\rangle) = 2\alpha(\langle s, i\rangle) + 1$. We can thus use Lemma 5.20.(6) to simplify our claim. It suffices to show that for every $i > k$ and $q \in S_i$, we have $\alpha(\langle q, i\rangle) = \lfloor f_i(q)/2\rfloor$. We proceed by induction over $i > k$. As the base case, consider a node $\langle q, k\rangle$. Recall that $\alpha(\langle q, k\rangle) = |\{[v] \mid \lambda^k(v) = \top, \langle q, k\rangle \prec_k v\}|$. By the definition of $\lambda^k$, a node on level $k$ is labeled $\bot$ only when it is an $F$-node. All other nodes inherit the label of their parents, and every node on level $k$ is $\top$-labeled. From Lemma 5.20.(2), we then have that $\alpha(\langle q, k+1\rangle) = |\{[v] \mid v \in S \setminus F, u \prec v\}|$, which is the definition of $\beta(q) = \lfloor f_i(q)/2\rfloor$.

Inductively, assume the claim holds for every $q \in S_i$. We show for every $s \in S_{i+1}$, it holds that $\alpha(\langle s, i+1\rangle) = \lfloor f_{i+1}(s)/2\rfloor$. Let $q$ be $s$'s parent in $G'$, i.e. $E'(q, s)$. Take the set $P = \{[v] \mid \lambda^k(v) = \top, \langle q, i\rangle \prec_i v\}$ of $\top$-labeled equivalence classes greater than $q$, By the inductive hypothesis, $\lfloor f_i(q)/2\rfloor = \alpha(\langle q, i\rangle) = |P|$. By the definition of $\mathbf{r}^k$, each $[v] \in P$ has a unique odd rank assigned to each of its elements. By Lemma 5.20.(5), for each $[v]$ this odd rank is smaller than $f_i(q)$. Consider $P$'s subset $P_s = \{[v] \mid [v] \in P,\ [v] \text{ has } \top\text{-labeled child class on level } i+1\}$. Define $P_e = P \setminus P_s$ to be the complementary set: pipes that die on level $i$. By Lemma 5.20.(5), before the `tighten` operation is applied, every element of $P_e$ has a corresponding odd rank that is unoccupied on level $i+1$. Since $q$ is clearly not in an element of $P_e$, this odd rank must be less than $\lfloor f_i(q)\rfloor_{even}$. Thus the final rank assigned to $s$, after `tighten`, is either $f_i(q) - 2|P_e|$ or $\lfloor f_i(q) - 2|P_e|\rfloor_{even}$. In both cases $\lfloor f_{i+1}(s)/2\rfloor = \lfloor f_i(q)/2\rfloor - |P_e|$. By the inductive hypothesis this is equivalent to $\alpha(\langle q, i\rangle) - |P_e| = |P| - |P_e|$. By the definition of $P_s$ and $P_e$, $|P| - |P_e| = |P_s|$. By Lemma 5.3, every $\top$-labeled child of a class in $P_s$ is lexicographically larger than $\langle s, i+1\rangle$. As every $\top$-labeled child must have a unique parent in $P_s$, we conclude that $|P_s| = \alpha(\langle s, i+1\rangle)$. $\square$

**Analysis:** Like the tight-ranking construction in Section 2.2.1, the automaton $C^U(\mathcal{B})$ operates in two stages. In both, the second stage is the set of tight level rankings and obligation sets. The tight-ranking construction uses sets of states in the first stage, and is bounded by the size of the second stage: $(0.96n)^n$ [FKV06]. The automaton $C^U(\mathcal{B})$ replaces the first stage with preordered subsets. As the number of preordered subsets is $O((\frac{n}{e\ln 2})^n) \approx (0.53n)^n$ [Var80], the size of $C^U(\mathcal{B})$ remains bounded by $(0.96n)^n$. This can be improved to $(0.76n)^n$: see below. Further, $C^U(\mathcal{B})$ has a very restricted transition relation: states in the first stage only guess whether to remain in the first stage or move to the second, and have nondeterminism of degree 2. States in the second stage are deterministic. Thus the transition relation is linear in the number of states and size of the alphabet, and $C^U(\mathcal{B})$ is deterministic in the limit.

## 5.4 Variations on the Retrospective Construction

In this section we present two variations of $C^U(\mathcal{B})$: one based on Schewe's variant of the rank-based construction that achieves a tighter bound, and one the is amenable to Tabakov and Vardi's symbolic implementation of the rank-based construction. Schewe's construction alters the cut-point of the rank-based construction to check only one even rank at a time. Doing so drastically reduces the size of the cut-point: intuitively, we can avoid carrying the obligation set explicitly. Instead we could carry the current rank $i$ we are checking, and add to the domain of our ranking function a single extra symbol $c$ that indicates the state is currently being checked, and thus is of rank $i$. For an analysis of the resulting state space, please see [Sch09a]. For clarity , we do not remove the obligation set from the construction. Instead, states in this variant of the automaton carry with them the index $i$, and in a state $\langle f, O, i\rangle$, it holds that $O \subseteq \{q \mid f(q) = i\}$. For a level ranking

$f$, let $\mathtt{mr}(f)$ be the largest rank in $f$. Note that $\mathtt{mr}(f)$, for a tight ranking, is always odd.

**Definition 5.22.** For an NBW $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, Q^{in}_{\mathcal{B}}, \rho_{\mathcal{B}}, F_{\mathcal{B}} \rangle$, let $C^U_{\text{Schewe}}(\mathcal{B})$ be the NBW $\langle \Sigma, \mathbf{Q} \cup (\mathcal{R}^m \times 2^{Q_{\mathcal{B}}} \times N), Q^{in}_U, \rho_{Sch}, F_{Sch} \rangle$, where

- $\rho_{Sch}(\mathcal{S}, \sigma) = \{\langle \mathtt{torank}(\mathcal{S}'), \emptyset, 0 \rangle\} \cup \{\mathcal{S}'\}$, where $\mathcal{S}'$ is the $\sigma$-successor of $\mathcal{S}$.
- $\rho_{Sch}(\langle f, O, i \rangle, \sigma) = \{\langle f', O', i' \rangle\}$ where

$$f' \text{ is the } \sigma\text{-successor of } f$$
$$i' = \begin{cases} i & \text{if } O \neq \emptyset, \\ (i+2) \bmod (\mathtt{mr}(f')+1) & \text{if } O = \emptyset, \end{cases}$$
$$\text{and } O' = \begin{cases} \rho_{\mathcal{B}}(O, \sigma) \setminus \mathit{odd}(f') & \text{if } O \neq \emptyset, \\ \{q \mid f'(q) = i'\} & \text{if } O = \emptyset. \end{cases}$$

- $F_{Sch} = \mathcal{R}^m \times \{\emptyset\} \times \{0\}$

**Theorem 5.23.** *For every NBW $\mathcal{B}$, it holds that $L(C^U_{Schewe}(\mathcal{B})) = \overline{L(\mathcal{B})}$.*

*Proof.* Given a word $w$, we relate the runs of $C^U_{\text{Schewe}}(\mathcal{B})$ and $C^U(\mathcal{B})$. As both automata are comprised of two internally deterministic stages, with a nondeterministic transition, each index $k$ defines a unique run for each automaton. As the first stage of both automata are identical, and the second stage is deterministic, given a fixed $k$ let $p_L = \langle S_0, \preceq_0 \rangle, \ldots, \langle S_k, \preceq_k \rangle, \langle f_{k+1}, O_{k+1} \rangle, \langle f_{k+2}, O_{k+2} \rangle, \ldots$ be the run of $C^U(\mathcal{B})$ on $w$ that moves to the second stage on the $k$th transition, and let the corresponding run of $C^U_{\text{Schewe}}(\mathcal{B})$ be $p_{Sch} \langle S_0, \preceq_0 \rangle, \ldots, \langle S_k, \preceq_k \rangle, \langle f'_{k+1}, O'_{k+1}, n_{k+1} \rangle, \langle f'_{k+2}, O'_{k+2}, n_{k+2} \rangle, \ldots$ We show that $p_L$ is accepting iff $p_{Sch}$ is accepting, or more precisely that $p_L$ is rejecting iff $p_{Sch}$ is rejecting. First, we note that the level rankings $f_{k+1}, f_{k+2}, \ldots$ and $f'_{k+1}, f'_{k+2}, \ldots$ in both automata are defined by $\mathtt{torank}(\langle S_k, \preceq_k \rangle)$ and the $\sigma$-successor relation, and thus for every $j > k$, it holds $f_j = f'_j$.

In one direction, assume that $p_{Sch}$ is rejecting. This implies there is some $j > k$ so that for every $j' > j$, $O'_{j'}$ is non-empty. In turn, this implies that there is a sequence $q_j, q_{j+1}, \ldots$ of states so that, for every $j' \geq j$, we have that $q_{j'} \in O'_{j'}$, that $f_{j'}(q_{j'}) = n_j$, and that $q_{j'+1} \in \rho_{\mathcal{B}}(q_{j'}, w_{j'})$. If there is no $l > j$ where $O_l = \emptyset$, then we have that $p_L$ is rejecting. Alternately, if there is such a $l > j$, then $q_{l+1} \in O_{l+1}$, and for every $l' > l$ we have $q_{l'} \in O_{l'}$. Again, this implies $p_L$ is rejecting.

In the other direction, assume that $p_L$ is rejecting. This implies there is some $j > k$ so that for every $j' > j$ the set $O_{j'}$ is non-empty. In turn, this implies that there is an even rank $i$ and sequence $q_j, q_{j+1}, \ldots$ of states so that, for every $j' \geq j$, we have that $q_{j'} \in O_{j'}$, that $f_{j'}(q_{j'}) = i$, and that $q_{j'+1} \in \rho_{\mathcal{B}}(q_{j'}, w_{j'})$. We now consider the indexes $n_{j'}$ in $p_{Sch}$. If there is some $j' > j$ where $n_{j'} = i$, then for every $l \geq j'$, it will hold that $q_l \in O'_l$, and $p_{Sch}$ will be rejecting. Alternately, if there is no $j' > j$ where $n_{j'} = i$, then it must be that the indexes $n_{j'}$ stops cycling through the even ranks. This implies the obligation set stops emptying, and therefore that $p_{Sch}$ must be rejecting. $\square$

To symbolically encode a deterministic-in-the-limit automaton, we avoid storing the preorders. To encode the preorder in a BDD as a relation would require a quadratic number of variables, increasing the size unacceptably. Alternately, we could associate each state with its index in the preorder. Unfortunately, calculating the index of each state in the succeeding preorder would require a global compacting step, to remove indices that had become empty. To handle this difficulty, we simply store only the subset in the first stage, and transition to an arbitrary level ranking when we move to the second stage. This maintains determinism in the limit, and cannot result in false accepting run: we can always construct an odd ranking from the sequence of level rankings. The construction and a small example encoding are provided below.

**Definition 5.24.** For an NBW $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, Q^{in}_{\mathcal{B}}, \rho_{\mathcal{B}}, F_{\mathcal{B}} \rangle$, let $C^U_{\text{Symb}}(\mathcal{B})$ be the NBW $\langle \Sigma, 2^{Q_{\mathcal{B}}} \cup (\mathcal{R}^m \times 2^{Q_{\mathcal{B}}}), Q^{in}_{\mathcal{B}}, \rho_{Symb}, \mathcal{R}^m \times \{\emptyset\} \rangle$, where

- $\rho_{Symb}(S,\sigma) = \{\rho_{\mathcal{B}}(S,\sigma)\} \cup$
  $$\{\langle f, \emptyset\rangle \mid f \in \mathcal{R}^m \text{ and for all } q \in Q_{\mathcal{B}},\ f(q) \neq \bot \text{ iff } q \in \rho_{\mathcal{B}}(S,\sigma)\}.$$
- $\rho_{Symb}(\langle f, O\rangle, \sigma) = \rho_L(\langle f, O\rangle, \sigma)$

**Theorem 5.25.** *For every NBW $\mathcal{B}$, it holds that $L(C^U_{\text{Symb}}(\mathcal{B})) = \overline{L(\mathcal{B})}$.*

*Proof.* In one direction, assume $w \in \overline{L(\mathcal{B})}$. This implies $w \in L(C^U(\mathcal{B}))$, and thus there exists an accepting run $\langle S_0, \preceq_0\rangle, \dots, \langle S_k, \preceq_k\rangle, \langle f_{k+1}, O_{k+1}\rangle, \langle f_{k+2}, O_{k+2}\rangle, \dots$ of $C^U(\mathcal{B})$ on $w$. We show that $S_0, \dots, S_k, \langle f_{k+1}, O_{k+1}\rangle, \langle f_{k+2}, O_{k+2}\rangle, \dots$ is an accepting run of $C^U_{\text{Symb}}(\mathcal{B})$ on $w$. We note that in the second stage transitions and accepting states in $C^U_{\text{Symb}}(\mathcal{B})$ are identical to $C^U(\mathcal{B})$. Thus to show that this is an accepting run $C^U_{\text{Symb}}(\mathcal{B})$, we only need show that the run is valid from 0 to $k+1$,

By definition, $S_0 = Q^{in}_{\mathcal{B}}$ is the initial state of $C^U_{\text{Symb}}(\mathcal{B})$. For every $i,\ 0 \leq i < k$, it holds that $S_{i+1} = \rho_{\mathcal{B}}(S_i, w_i) \in \rho_{Symb}(S_i, w_i)$. Finally, consider the transition from $S_k$ to $\langle f_{k+1}, O_{k+1}\rangle$. Let $\langle S_{k+1}, \preceq_{k+1}\rangle$ be the $\sigma$-successor of $\langle S_k, \preceq_k\rangle$. By definition, $S_{k+1} = \rho_{\mathcal{B}}(S_k, w_k)$. By the transition relation of $C^U(\mathcal{B})$, we have $f_{k+1} = \texttt{torank}(\langle S_{k+1}, \preceq_{k+1}\rangle)$ and $O_{k+1} = \emptyset$. By the definition of $\texttt{torank}$, for every $q \in Q_{\mathcal{B}}$ it holds that $f_{k+1}(q) = \bot$ iff $q \notin S_{k+1}$. Thus $\langle f_{k+1}, O_{k+1}\rangle \in \rho_{Symb}(S_k)$, and $S_0, \dots, S_k, \langle f_{k+1}, O_{k+1}\rangle, \langle f_{k+2}, O_{k+2}\rangle, \dots$ is an accepting run of $C^U_{\text{Symb}}(\mathcal{B})$ on $w$.

In the other direction, if $w \in L(C^U_{\text{Symb}}(\mathcal{B}))$, this implies the existence of an accepting run $S_0, \dots, S_k, \langle f_{k+1}, O_{k+1}\rangle,\ \langle f_{k+2}, O_{k+2}\rangle, \dots$ of $C^U_{\text{Symb}}(\mathcal{B})$ on $w$. From this run we construct an odd ranking of $G$, which implies $w \in \overline{L(\mathcal{B})}$. Define the ranking function $f$ so that for every $\langle q, i\rangle \in G$: if $i \leq k$ then $\mathbf{r}(\langle q, i\rangle) = m = 2|Q_{\mathcal{B}} \setminus F_{\mathcal{B}}|$; and if $i > k$ then $\mathbf{r}(\langle q, i\rangle) = f_i(q)$. As demonstrated in the proof of Theorem 5.21, the definition of $\sigma$-successors and $G$ implies that when $i > k$, it holds that $f_i(q) \neq \bot$. Similarly, by the definition of $\sigma$-successors no path in $G$ can increase in rank under $\mathbf{r}$. We conclude that $\mathbf{r}$ is a valid ranking function.

To demonstrate that **r** is an odd ranking, assume by way of contradiction that there is a path $\langle q_0, 0\rangle, \langle q_1, 1\rangle, \ldots$ in $G$ that gets trapped in an even rank. Let $j$ be the point at which this path gets trapped, or $k+1$, whichever is later. If there is no $j' > j$ such that $O_{j'} = \emptyset$, then there is no accepting state after $j$, and the run would not be accepting. If there is such a $j'$, then $O_{j'+1}$ would contain $q_{j'+1}$, as $f_{j'+1}(q_{j'+1})$ is even. At every point $j'' > j' + 1$, the obligation set will contain $q_{j''}$, and thus there will be no accepting state after $j'$, and the run would not be accepting. However, we have that the run is accepting as a premise. Therefore no path in $G$ gets trapped in an even rank, **r** is an odd ranking, and by Lemma 2.2 $w \in \overline{L(\mathcal{B})}$. □

As an example, Figure 5.3 is the SMV encoding of the complement of a two-state automaton.

## 5.5 Discussion

We have unified the slice-based and rank-based approaches by phrasing the former in the language of run DAGs. This enables us to define and exploit a retrospective ranking, providing a deterministic-in-the-limit complementation construction that does not employ determinization. Experiments show that the more deterministic automata are, the better they perform in practice [ST03]. By avoiding determinization, we reduce the cost of such a construction from $(n^2/e)^n$ to $(0.76n)^n$ [Pit06].

In addition, our transition generates a transition relation that is linear in the number of states and size of the alphabet. Schewe demonstrated how to achieve a similar linear bound on the transition relation, but the resulting relation is larger and is not deterministic in the limit [Sch09a].

The use of level rankings affords several improvements from existing research on the rank-based approach. First, the cut-point construction of Miyano and Hayashi [MH84]

```
typedef STATE 0..1;  /* Size for complemented automaton: 2, maximum allowed rank = 2*/
module main() {
 letter: {a,b};                   /* The transition letter */
 rank: array STATE of 0..3;       /* The value 3 represents bottom */
 phase : 0..1;                    /* The automaton phase, ranks 2 or 3 in phase 0*/
 subset: array STATE of boolean;  /* The obligation set vector */
 init(rank) := [2,2,2,2];         /* 2 to initial states, 3 to others */
 init(subset) := [1,1,1,1];       /* initially rejecting */
 init(phase) := 0;
 next(phase) := {i : i=0..1, i >= phase};

 /* Define the rank of states in the next time step. Cases fall through. */
 /* state 0 has transition from 0 on a and b */
 next(rank[0]) := case {
     rank[0]=3 : 3;
     next(phase)=0 : 2;
     phase=0 & next(phase)=1 : {i : i=0..2, i <= rank[0]};
     phase=1 : rank[0];
 };

 /* 1 has transition from 1 on a and from 0 on b. 1 is accepting */
 next(rank[1]) := case {
    letter=a & rank[1]=3 : 3;
    letter=a & next(phase)=0 : 2;
    letter=a & phase=0 & next(phase)=1 : {i : i=0..2, i <= rank[1] & i in {0,2}};
    letter=a & phase=1 : {i : i=0..2, i in {rank[1], rank[1]-1} & i in {0,2}};
    letter=b & rank[0]=3 : 3;
    letter=b & next(phase)=0 : 2;
    letter=b & phase=0 & next(phase)=1 : {i : i=0..2, i <= rank[0] & i in {0,2}};
    letter=b & phase=1 : {i : i=0..2, i in {rank[0], rank[0]-1} & i in {0,2}};
 };


 /* Defining the transitions of the P-set */
 if (next(phase)=0)  {
     forall (i in STATE) next(subset[i]) := 1;
 } else {
   if (subset=[0,0,0,0]) { /* The P-set is empty */
     forall (i in STATE) next(subset[i]) := next(rank[i]) in {0,2};
   } else { /* The P-set is non-empty */
     if (letter=a) {
       next(subset[0]) := (subset[0]) & next(rank[0]) in {0,2};
       next(subset[1]) := (subset[1]) & next(rank[1]) in {0,2};
     } else { /* letter=b */
       next(subset[0]) := (subset[0]) & next(rank[0]) in {0,2};
       next(subset[1]) := (subset[0]) & next(rank[1]) in {0,2};
 }}}
SPEC 0;
 FAIRNESS subset=[0,0,0,0];
}
```

**Figure 5.3 :** The SMV encoding of the $C^U_{\text{Symb}}(\mathcal{B})$, for the two-state automaton consisting of states $p$ and $q$ of Figure 2.1.

can be improved. Schewe's construction only checks one even rank at a time, reducing the size of the state space to $(0.76n)^n$, only an $n^2$ factor from the lower bound [Sch09a]. As Schewe's approach does not alter the progression of the level rankings, it could be applied directly to the second stage of Definition 5.19. The resulting construction inherits the asymptotic state-space complexity of [Sch09a]. Second, symbolically encoding a preorder is complicated. In contrast, ranks are easily encoded, and the transition between ranks is nearly trivial to implement in SMV [TV07]. By changing the states in first stage of $C^U(\mathcal{B})$ from preordered subsets to simple subsets, and guessing the appropriate transition to the second stage, we obtain a symbolic representation while maintaining determinism in the limit. This approach sacrifices the linear-sized transition relation, but this is less important in a symbolic encoding. Finally, the subsumption relations of Doyen and Raskin [DR09] could be applied to the second stage of the automaton, while it is unclear if it could be applied at all to the slice-based construction.

From a broader perspective, we find it very interesting that the prospective and retrospective approaches are so strongly related. Odd rankings seem to be inherently "prospective," depending on the descendants of nodes in the run DAG. By investigating the slice-based approach, we are able to pinpoint the dependency on the future to a single component: the $F$-free level.

# Chapter 6

# Profile-Based Determinization

This chapter presents a mathematically crisp determinization construction, based on the notion of *profiles* from the previous chapter. In this chapter , we consider the equivalence classes of nodes induced by profiles, in which two nodes are in the same class if they have the same profile. We show that profiles turn the run DAG into a binary tree, with bounded width, over the equivalence classes. (In contrast to Safra trees, this is not a sequence of trees, each encoding the nodes of a single level of the run DAG. Rather, this is a single tree that captures the accepting or rejecting nature of the run DAG.) We introduce a labeling, in which labels follow lexicographically minimal infinite branches of this binary tree. This labeling provides a Büchi determinization construction, where a state of the deterministic automaton is a set of states of the input nondeterministic automaton, the lexicographic preorder induced by profiles, a second preorder that encodes information about shared ancestry of nodes, and the labeling. This yields a mathematically crisp description of the deterministic automaton.

We first present some additional background material. We then describe how to describe $G'$ as a tree of equivalence classes. The tree is then labeled in a way that we can deterministically track acceptance. Finally, we demonstrate how to construct this labeling with a bounded amount of space.

---

This chapter contains unpublished work done with Orna Kupferman, Moshe Y. Vardi, and Thomas Wilke.

## 6.1 Preliminaries

### 6.1.1 Relations on Sets

Given a set $R$, a binary relation $\leq$ over $R$ is a *preorder* if $\leq$ is reflexive and transitive. If for every $r_1, r_2 \in R$ either $r_1 \leq r_2$ or $r_2 \leq r_1$, then $\leq$ is a *linear preorder*. If a preorder $\leq$ is antisymmetric, that is, $r_1 \leq r_2$ and $r_2 \leq r_1$ implies $r_1 = r_2$, then it is a *partial order*. A linear partial order is a *total order*. Consider a partial order $\leq$. If for every $r \in R$, the set $\{r' \mid r' \leq r\}$ is totally ordered by $\leq$, then we say that $\leq$ is a *tree order*. The equivalence class of $r \in R$ under $\leq$, written $[r]$, is $\{r' \mid r' \leq r \text{ and } r \leq r'\}$. The equivalence classes under a linear preorder form a totally ordered partition of $R$. Given a set $R$ and linear preorder $\leq$ over $R$, define the minimal elements of $R$ as $\mathtt{min}_{\leq}(R) = \{r_1 \in R \mid \; r_1 \leq r_2 \text{ for all } r_2 \in R\}$. Note that $\mathtt{min}_{\leq}(R)$ is either empty or an equivalence class under $\leq$. Given a set $R$ and a total order $\leq$, we instead define $\mathtt{min}_{\leq}$ as the partial function that maps $R$ to its unique minimal element, if exists.

Given two sets $R$ and $R'$ where $|R| \leq |R'|$, a linear preorder $\leq$ over $R$, and a total order $<'$ over $R'$, define the $\langle \leq, <' \rangle$*-minjection* from $R$ to $R'$ to be the function $\mathtt{mj}$ that maps all the elements in the $k$-th equivalence class of $R$ to the $k$-th element of $R'$. The number of equivalence classes is at most $|R|$, and thus at most $|R'|$. If $\leq$ is also a total order, than the $\langle \leq, <' \rangle$-minjection is also an injection.

*Example* 6.1. Let $R = \mathbb{Q}$ and $R' = \mathbb{N}$ be the sets of rational numbers and integers, respectively. Define the linear preorder $\leq_1$ over $\mathbb{Q}$ by $x \leq_1 x'$ iff $\lfloor x \rfloor \leq \lfloor x' \rfloor$, and the total order $<_2$ over $\mathbb{N}$ by $x <_2 x'$ if $x < x'$. Then, the $\langle \leq_1, <_2 \rangle$-minjection from $\mathbb{Q}$ to $\mathbb{N}$ maps a rational number $x$ to $\lfloor x \rfloor$.

### 6.1.2 Non-Büchi $\omega$-Automata

While nondeterministic Büchi automata are $\omega$-regular, their deterministic counterparts are not. We thus introduce more powerful acceptance conditions that generalize Büchi automata. A *nondeterministic $\omega$-automaton* is a tuple $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, Q^{in}_{\mathcal{B}}, \rho_{\mathcal{B}}, \alpha_{\mathcal{B}} \rangle$, where $\Sigma$ is a finite alphabet, $Q_{\mathcal{B}}$ is a finite set of states, $Q^{in}_{\mathcal{B}} \subseteq Q_{\mathcal{B}}$ is a set of initial states, $\rho_{\mathcal{B}} \colon Q_{\mathcal{B}} \times \Sigma \to 2^{Q_{\mathcal{B}}}$ is a nondeterministic transition relation, and $\alpha$ is an acceptance condition defined below.

The acceptance condition $\alpha_{\mathcal{B}}$ determines if a run is accepting. Recall that for a *Büchi automaton*, the acceptance condition is a set of states $F_{\mathcal{B}} \subseteq Q_{\mathcal{B}}$, and a run $q_0, q_1, \ldots$ is accepting iff $q_i \in F_{\mathcal{B}}$ for infinitely many $i$'s. For a *Rabin-edge automaton*, the acceptance condition is a set $\langle G_0, B_0 \rangle, \ldots, \langle G_k, B_k \rangle$ of pairs of sets of transitions: thus $G_j, B_j \subseteq Q^2_{\mathcal{B}}$ for $0 \leq j \leq k$. A run is accepting iff there exists $0 \leq j \leq k$ so that $\langle q_i, q_{i+1} \rangle \in G_j$ for infinitely many $i$'s, while $\langle q_i, q_{i+1} \rangle \in B_j$ for only finitely many $i$'s. For a *parity-edge automaton*, the acceptance condition is a parity function $\gamma \colon Q^2_{\mathcal{B}} \to \{0, \ldots, k\}$, and a run is accepting if the smallest element of $\{j \mid j = \gamma(q_i, q_{i+1})$ for infinitely many $i$'s$\}$ is even. We are particularly interested in nondeterministic Büchi automata (NBW), deterministic Rabin-edge automata (DREW) and deterministic parity-edge automata (DPEW).

### 6.1.3 Safra's Determinization Construction

This section presents a short description of Safra's determinization construction, modeled after the exposition in [Pit06]. This construction takes a nondeterministic Büchi automaton $\mathcal{B}$ and constructions an equivalent deterministic Rabin automaton $D^{Safra}(\mathcal{B})i$. Safra's construction is based on Safra trees. Each Safra tree store one level of the run DAG $G$ of $\mathcal{B}$ on a word $w$, along with additional information encoding a finite amount of the history of $G$. Because nodes are moved and shifted frequently in Safra trees, we will not define them as prefix-closed sets of strings. Instead, we define $V = \{1 \dots |Q|_{\mathcal{B}}\}$ and use $V$ to label nodes.

**Definition 6.2.** A *Safra tree* is a tuple $T = \langle N, r, p, \prec, l \rangle$ where:

- $N \subseteq V$ is a set of nodes.
- $r \in N$ is the root node.
- $p : (N \setminus r) \to N$ is the parent function.
- $\prec$ is a partial order representing 'older than' over siblings: for $v, v' \in N,\ v \neq v'$, it holds that $p(v) = p(v')$ iff either $v' \prec v$ or $v \prec v'$.
- $l : N \to 2^{Q_{\mathcal{B}}}$ is a labeling function from nodes to sets of states.

Further, the following three conditions hold:

(1) The label of every $v \in N$ is non-empty: $l(v) \neq \emptyset$.

(2) The label of every $v \in N$ is a strict superset of the union of labels of its children: $l(v) \supset \bigcup \{l(w) \mid p(w) = v\}$.

(3) The labels of the siblings are disjoint: for $v, v' \in N$, if $v \neq v'$ and $p(v) = p(v')$ then $l(v) \cap l(v') = \emptyset$.

Let $\mathcal{T}$ denote the set of Safra trees. Since every run begins in the initial set of states, define the initial Safra tree to be $T^{in} = \langle \{1\}, 1, \emptyset, \emptyset, l(1) = Q^{in} \rangle$. Given a Safra tree $T$ and a character $\sigma$, we define the unique $\sigma$-successor of $T$, written $T' = \langle N', r', p', \prec', l \rangle$, operationally. Intuitively, we first change the label of each node to the successors of the current label under $\rho_{\mathcal{B}}$. Then, for every node with $F$-states in its label, we create a new, youngest, child of that node labeled with those $F$-states. Finally, we restore the three conditions of Safra trees stated above: if the labels of two siblings share a state, we remove the state from the label of the younger sibling; if the label of a node is no longer a strict superset of the labels of its children, we remove all children; and if the label of a node is empty, we remove the node. During intermediate steps, there may be more than $|Q_{\mathcal{B}}|$ nodes. Thus we temporarily allow additional nodes from a set $V'$ disjoint from $V$. Formally, define $T'$ as the result of the following sequence of operations:

(1) Initially, let $N' = N$, $p' = p$, and $\prec' = \prec$.

(2) For every $v \in N'$, initially let $l'(v) = \rho_{\mathcal{B}}(l(v), \sigma)$

(3) For every $v \in N'$ where $l'(v) \cap F \neq \emptyset$, create a new node $w \in V'$ where: $p'(w) = v$; $l'(w) = l'(v) \cap F$; and for every $w' \in N'$ where $p(w') = v$, $w \prec' w'$.

(4) For every node $v \in N'$ and state $s \in l'(v)$, if there exists $v' \in N'$, $s \in l'(v')$ where $v \prec v'$, then remove $s$ from $l'(v)$ and, for every descendant $w$ of $v$ under $p'$, remove $s$ from $l'(w)$.

(5) For every node $v \in N'$, if $l'(v) = \bigcup\{l'(w) \mid p'(w) = v\}$ then remove all descendants of $v$.

(6) For every node $v \in N'$, if $l'(v) = \emptyset$, then remove $v$.

(7) Change all nodes in $V'$ to nodes in $V$.

We do not formally prove that the $\sigma$-successor of a Safra tree is a Safra tree. Intuitively, we note that the first three steps define the elements, and the second three steps restore the properties we demand of a Safra tree. To see that the final step is allowable, we note that since every node must be labeled with a strict superset of the labels of its children, there are at most $n$ nodes. Thus we can fit all the new nodes into the set $V$. Safra's construction allows us to check the sequence of Safra trees for properties that are necessary and sufficient for acceptance. Given a Safra tree $T$ and it's $\sigma$-successor $T'$, say a node $v \in N'$ is *successful in* $T'$ if the descendants of $v$ were removed in step (5). If a node is successful infinitely often, then there is an accepting path through the states that label that node. However, because node names are reused, we have to take care not to consider successes on nodes that share a name, but represent different paths.

Thus say a node $v \in V$ *dies in* $T'$ if $v \notin N'$. We use success and death to define a Rabin condition in our deterministic automaton.

**Definition 6.3.** Define the DREW automaton $D^{Safra}(\mathcal{B})$ to be:
$\langle \Sigma, \mathcal{T} \times \{1, \ldots, |Q|_B\}^2, \langle T^{in}, \emptyset, \{2, \ldots |Q|_B\}\rangle, \rho_{\mathbf{T}}, \alpha\rangle$, where:

- For $\langle T, G, B\rangle$ and $\sigma \in \Sigma$, let $\rho_{\mathbf{T}}(\langle T, G, B\rangle\sigma) = \langle T', G', B'\rangle$ where:
  - $T'$ is the $\sigma$-successor of $T$
  - $G' = \{v \in N' \mid v \text{ is successful in } T'\}$
  - $B' = \{v \in V \mid v \text{ dies in } T'\}$
- $\alpha = \langle G_1, B_1\rangle, \ldots, \langle G_{|Q_{\mathcal{B}}|}, B_{|Q_{\mathcal{B}}|}\rangle$, where for every $i \in \{1, \ldots, |Q_{\mathcal{B}}|\}$:
  - $G_i = \{\langle T, G, B\rangle \mid i \in G\}$.
  - $B_i = \{\langle T, G, B\rangle \mid I \in B\}$.

**Theorem 6.4.** [Saf88] *For a Büchi automaton $\mathcal{B}$ with $n$ states, $L(D^{Safra}(\mathcal{B})) = L(\mathcal{B})$ and $D^{Safra}(\mathcal{B})$ has $n^{O(n)}$ states.*

## 6.2 Equivalence Classes Under Profiles

In this section we define $T^{G'}$, a binary tree of bounded width that captures the accepting or rejecting nature of $G$. Nodes of this tree are equivalence classes of nodes in the run DAG under the profiles of Chapter 5.2. We then present a method of deterministically labeling the classes in $T^{G'}$, so we can determine if $G$ is accepting by examining the labels. The labels are integers, and we first allow the use of an unbounded number of labels. Finally, we show how to determine the labeling using bounded information, and how to use a fixed set of labels.

### 6.2.1 The Tree of Equivalence Classes

The nodes of $T^{G'}$ are the equivalence classes $\{[u] \mid u \in G'\}$ of $G'$. To reduce confusion, we refer to the nodes of $T^{G'}$ as *classes*, and use $u$ and $v$ for nodes in $G$, and $U$ and $V$ for classes in $T^{G'}$. As all members of a class share a profile, we define the profile $h_U$ of a class $U$ to be $h_u$ for some $u \in U$. Recall that by Lemma 5.3, a class $U$ can have at most two children: the $F$-child with profile $h_U 1$, and the non-$F$-child with profile $h_U 0$. A class $V$ is a *descendant* of a class $U$ if there is a, possibly empty, path from $U$ to $V$ in $T^{G'}$.

**Lemma 6.5.** *$T^{G'}$ is a binary tree of bounded width of degree $|Q_{\mathcal{B}}|$.*

*Proof.* That $T^{G'}$ has bounded width follows from the fact that a class on level $i$ contains at least one node on level $i$ of $G$, and $G$ is of bounded width of degree $|Q_{\mathcal{B}}|$. To show that $T^{G'}$ is a tree, note that as $Q^{in}_{\mathcal{B}} \cap F_{\mathcal{B}} = \emptyset$, all nodes on the first level of $G$ have profile $0$, and every class descends from this class of nodes with profile $0$. For a class $V$, let $U = \{u \mid \text{ there is } v \in V \text{ such that } \langle u, v \rangle \in E'\}$. Lemma 5.4 implies that $U$ is an equivalence class, and is the sole parent of $V$. Finally, Lemma 5.3 entails that a class $U$ can have at most two children: the class with profile $h_U 1$, and the class with profile $h_U 0$. Thus $T^{G'}$ is binary. □

A *branch* of $T^{G'}$ is a finite or infinite initial path in $T^{G'}$. Since $T^{G'}$ is a tree, two branches share a prefix until they *split*. An infinite branch is *accepting* if it contains infinitely many $F$-classes, and is *rejecting* otherwise. An infinite rejecting branch must reach a suffix consisting only of non-$F$-classes. Note that if $U'$ is a descendant of both $U$ and $V$, either $U$ is a descendant of $V$, or $V$ is a descendant of $U$. A class $U$ is called *finite* if it has finitely many descendants, and a finite class $U$ *dies out* on level $k$ if it has a descendant on level $k-1$, but none on level $k$.

We extend the function $f$ to classes, so that $f(U) = 1$ if $U$ is an $F$-class, and $f(U) = 0$ otherwise. We define the profile of an infinite branch $b = U_0, U_1, \ldots$ to be $h_b = f(U_0), f(U_1), \ldots$. For two classes $U$ and $V$ on level $i$, we say that $U \prec_i V$ if $h_U < h_V$. For two infinite branches $b$ and $b'$, we say that $b \prec b'$ if $h_b < h_{b'}$. Note that $\prec_i$ is a total order over the classes on level $i$, and that $\prec$ is a total order over infinite branches. We now show that while $G'$ can have infinitely many infinite branches, this is not possible for a tree.

**Lemma 6.6.** *$T^{G'}$ has a finite number of infinite branches.*

*Proof.* As $T^{G'}$ has bounded width $|Q_{\mathcal{B}}|$, there are at most $|Q_{\mathcal{B}}|$ infinite branches. $\square$

**Theorem 6.7.** *$G$ has an accepting path iff $T^{G'}$ has an accepting branch.*

*Proof.* If $G$ is accepting, then by Lemma 5.6 we have that $G'$ contains an accepting path $u_0, u_1, \ldots$. This path gives rise to an accepting branch, $[u_0], [u_1], \ldots$ in $T^{G'}$. In the other direction, if $T^{G'}$ has an accepting branch $U_0, U_1, \ldots$, consider the infinite subgraph of $G'$ consisting only of the nodes in $U_i$, for $i > 0$. For every $i$, there exists $u_i \in U_i$ and $u_{i+1} \in U_{i+1}$ so that $E'(u_i, u_{i+1})$. Because no node is orphaned in $G'$, Lemma 5.4 implies that every node in $U_{i+1}$ has a parent in $U_i$, thus this subgraph is connected. As each node has degree of as most $n$, Koňig's Lemma implies that there is an initial path $u_0, u_1, \ldots$ through this subgraph. Further, at every level $i$ where $U_i$ is an $F$-class, we have that $u_i \in U_i$, and thus this path is accepting in $G'$ and in $G$. $\square$

### 6.2.2 Labeling $T^{G'}$

We first present a labeling that uses an unbounded number of labels and global information about $T^{G'}$. We call this labeling the *global labeling*, and denote it with $gl$. For a class $U$ on level $i$ of $T^{G'}$, and a class $V$, we say that $V$ is *before* $U$ if $V$ is on level $j < i$

or $V \prec_i U$. For each label $m$, we refer to the first class labeled $m$ as $\texttt{first}(m)$. That is, $U = \texttt{first}(m)$ if $U$ is labeled $m$ and for all classes $V$ before $U$, the label of $V$ is not $m$. For a label $m$ and level $i$, let the lexicographical minimal descendant of $m$ on level $i$, written $\texttt{lmd}(m, i)$, be $\texttt{min}_{\preceq}(\{V \mid V \text{ is a descendant of } \texttt{first}(m) \text{ on level } i.\})$. That is, $\texttt{lmd}(m, i)$ is the class with the minimal profile among all the descendants of $\texttt{first}(m)$ on level $i$. For a class $U$ on level $i$, we define $\texttt{labels}(U) = \{m \mid U = \texttt{lmd}(m, i)\}$, and define $\texttt{ml}(U) = \texttt{max}(\{gl(V) \mid V \text{ is before } U\})$ as the maximum label occurring in $T^{G'}$ before $U$. We define the labeling function $gl$ as follows. For the initial class $U_0 = \{\langle q, 0\rangle \mid q \in Q^{in}_{\mathcal{B}}\}$ with profile $0$, let $gl(U_0) = 0$. Inductively, we proceed lexicographically through each level of $T^{G'}$.

**Definition 6.8.** $gl(U) = \begin{cases} \texttt{min}(\texttt{labels}(U)) & \text{if } \texttt{labels}(U) \neq \emptyset, \\ \texttt{ml}(U) + 1 & \text{if } \texttt{labels}(U) = \emptyset. \end{cases}$

Lemma 6.9 demonstrates that the labeling has some very nice properties:

**Lemma 6.9.** *For classes $U$ and $V$ on level $i$ of $T^{G'}$, it holds that:*

*(1) If $U \neq V$ then $gl(U) \neq gl(V)$.*

*(2) $U$ is a descendant of $\texttt{first}(gl(U))$.*

*(3) If $U$ is a descendant of $\texttt{first}(gl(V))$, then $V \preceq_i U$. Consequently, if $U \prec_i V$, then $U$ is not a descendant of $\texttt{first}(gl(V))$.*

*(4) $\texttt{first}(gl(U))$ is an $F$-class with a sibling.*

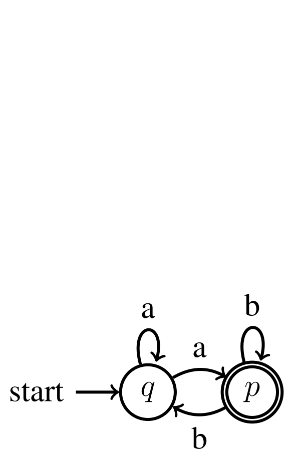
*(5) If $gl(U) < gl(V)$ then $\texttt{first}(gl(U))$ is before $\texttt{first}(gl(V))$.*

*(6) If $U \neq \texttt{first}(gl(U))$, then there is a class $V$ on level $i - 1$ that has label $m$.*

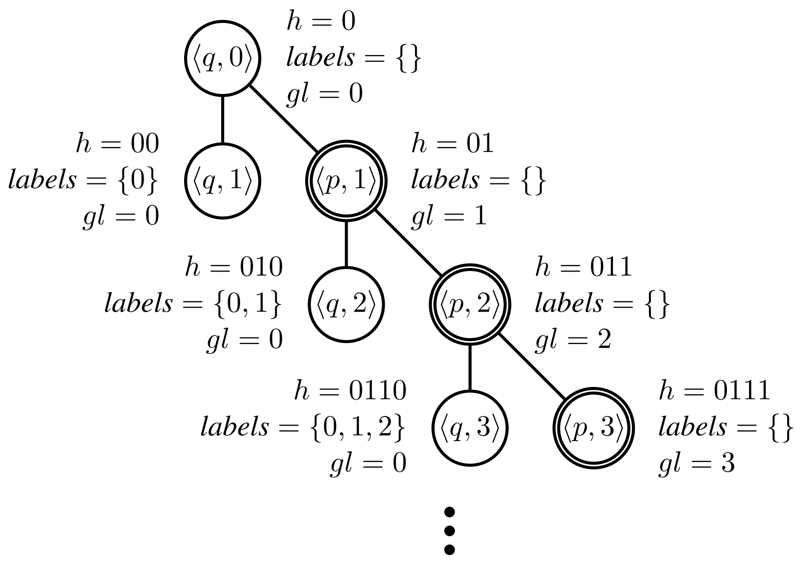
Intuitively, the label $m$ is going to follow the lexicographical minimal child of $\texttt{first}(m)$ on every level. Consider the lexicographically minimal branch from $\texttt{first}(m)$. Recall that if this branch is infinite and visits infinitely many $F$-classes, then $T^{G'}$ is accepting. The label $m$ stands for the proposition that the lexicographically minimal infinite branch going through $\texttt{first}(m)$ is accepting. If a class $U$ with label $m$ is on the lexicographical minimal infinite branch from $\texttt{first}(m)$, then in order for $m$ to "survive", it is waiting for the branch to reach an $F$-class. If $U$ is not on the lexicographically minimal infinite branch from $\texttt{first}(m)$, then we know that in $T^{G'}$ all the paths from $U$ are going to be finite, Accordingly, in order for $m$ to survive, it is waiting for $U$ to die out. Note that when a path leaves the lexicographically minimal infinite branch, it must go to a non-$F$-child, which implies that the lexicographically minimal infinite branch, does go through an $F$-child. Thus, when the non-$F$-child dies out, this is a supporting evidence to the fact that the lexicographically minimal infinite branch from $\texttt{first}(m)$ is accepting.

Formally, we say that a label $m$ is *gl-successful on level $i$* if there is a class $U$ on level $i - 1$ and a class $U'$ on level $i$ such that $gl(U) = gl(U') = m$, and either $U'$ is the $F$-child of $U$ or $U'$ is not a child $U$. Recall that the label $m$ represents the proposition that the lexicographically minimal infinite branch going through $\texttt{first}(m)$ is accepting. When a class with label $m$ has two children, we are not certain which, if either, is part of an infinite branch. We are thus conservative, and follow the non-$F$-child. If the non-$F$-child dies out, we revise our guess and move to a descendant of the $F$-child.

Consider for example the NBW in Figure 6.1 (a), and the tree of equivalence classes that corresponds to a run of it in the word $ab^{\omega}$. There is only one infinite branch, $\{\langle q, 0\rangle\}, \{\langle p, 1\rangle\}, \{\langle p, 2\rangle\}, \ldots$, which is accepting. At level 0 this branch is labeled with 0. At a level $i > 0$, we conservatively assume that an infinite branch that starts in $\langle q, 0\rangle$ goes through $\{\langle q, i\rangle\}$, and we thus label $\{\langle q, i\rangle\}$ by 0. As $\{\langle q, i\rangle\}$ is proven finite on level $i + 1$, we revise our assumption and continue to follow the path through $\{\langle p, i\rangle\}$. Since $\{\langle q, i + 1\rangle\}$ is a non-$F$-class, the label 0 is $gl$-successful on every level



**(a)** An automaton



**(b)** $T^{G'}$ for automaton (a) on $ab^\omega$.

**Figure 6.1 :** An automaton and tree of classes. Each class is a singleton set, brackets are omitted for brevity. Each class is labeled with its profile $h$, the set `labels`, and its label under $gl$. $F$-classes are circled twice.

$i+1$. Although the infinite branch is not labeled $0$ after the first level, the label $0$ asymptotically approaches the infinite branch, checking along the way that the branch is in fact lexicographically minimal among the infinite branches through the root.

**Theorem 6.10.** *$T^{G'}$ has an accepting branch iff there is a label $m$ that is $gl$-successful infinitely often.*

*Proof.* In one direction, let $m$ be a label that is $gl$-successful infinitely often. As $m$ can be successful only when it occurs, $m$ occurs infinitely often. This implies `first`$(m)$ has infinitely many descendants and there is at least one infinite branch through `first`$(m)$. Let $b = U_0, U_1, \ldots$ be the lexicographically minimal infinite branch that goes through `first`$(m)$. We demonstrate that $b$ cannot have a suffix consisting solely of non-$F$-classes, and therefore is an accepting branch. By way of contradiction, assume there is an index $j$ so that for every $k > j$, the class $U_k$ is a non-$F$-class. By Lemma 6.9.(4), $\texttt{first}(m)$ is an $F$-class, and thus occurs before level $j$.

Let $\mathcal{U} = \{V \mid V \prec_j U_j,\ V \text{ is a descendant of } \texttt{first}(m)\}$ be the set of descendants of $\texttt{first}(m)$ on level $j$ that are lexicographically smaller than $U_j$. Since $b$ is the lexicographically minimal infinite branch through $\texttt{first}(m)$, every class in $\mathcal{U}$ must be finite. Let $j' \geq j$ be the level at which the last class in $\mathcal{U}$ dies out. At this point, $U_{j'}$ is the lexicographically minimal descendant of $\texttt{first}(m)$. Since $m$ occurs infinitely often, it must be that $gl(U_{j'}) = m$. Indeed, otherwise there is no class on level $j'$ with label $m$, and, by Lemma 6.9.(6), $m$ would not occur after level $j'$. Further, for every $k > j'$, the class $U_k$ is the lexicographically minimal descendant of $U_{j'}$ on level $k$, and so $gl(U_k) = m$. This implies that $m$ is not successful after level $j'$, a contradiction. Therefore, there is no such suffix of $b$, and $b$ must be an accepting branch.

For the other direction, if there is an infinite accepting branch, then let $b = U_0, U_1, \ldots$ be the lexicographically minimal infinite accepting branch. Let $B'$ be the set of infinite branches that are lexicographically smaller than $b$. As $b$ is the minimal infinite accepting branch, every branch in $B'$ must be rejecting. Let $j$ be the first index after which the last branch in $B'$ splits from $b$. Note that either $j = 0$, or $U_{j-1}$ is part of an infinite rejecting branch $U_0, \ldots, U_{j-1}, V_j, V_{j+1}, \ldots$ smaller than $b$. In both cases, we show that $U_j$ is the candidate class for a new label $m$ that occurs on every level $k > j$ of $T^{G'}$.

In the first case, where $j = 0$, let $m = 0$. Since $m$ is the smallest label, and there is a descendant of $U_j$ on every level of $T^{G'}$, it holds that $m$ will occur on every level of $T^{G'}$. In the second case, where $j > 0$, then $V_j$ must be the non-$F$-child of $U_{j-1}$, and so $U_j$ is the $F$-child. Thus, $U_j$ is given a new label $m$ where $U_j = \texttt{first}(m)$. For a label $m' < m$ and level $k > j$, it cannot be that $\texttt{lmd}(m', k)$ is a descendant of $U_j$, as $V_k$ will be a descendant of $\texttt{first}(m')$, and for every descendant $U'$ of $U_j$, it holds that

$V_k \preceq_k U'$. Thus, on every level $k > j$, the lexicographically minimal descendant of $U_j$ will be labeled $m$, and $m$ occurs on every level of $T^{G'}$.

We show that $m$ is $gl$-successful infinitely often by defining an infinite sequence of levels, $j_0, j_1, j_2, \ldots$, so that $m$ is $gl$-successful on $j_i$ for all $i > 0$. As a base case, let $j_0 = j$. Inductively, at level $j_i$, let $U'$ be the class on level $j_i$ labeled with $m$. We have two cases. If $U' \neq U_{j_i}$, then as all infinite branches smaller than $b$ have already split from $b$, $U'$ must be finite in $T^{G'}$. Let $j_{i+1}$ be the level at which $U'$ dies out. At level $j_{i+1}$, $m$ will return to a descendant of $U_{j_0}$, and $m$ will be $gl$-successful. In the second case, $U' = U_{j_i}$. Take the first $k > j_i$ so that $U_k$ is an $F$-class. As $b$ is an accepting branch, such $k$ must exist. If $U_k$ is the only child of $U_{k-1}$ then let $j_{i+1} = k$: since $gl(U_k) = m$ and $U_k$ is not the non-$F$-child of $U_{k-1}$, it holds that $m$ is $gl$-successful on level $k$. Otherwise let $U'_k$ be the non-$F$-child of $U_{k-1}$, so that $gl(U'_k) = m$. Again, $U'_k$ is finite. Let $j_{i+1}$ be the level at which $U'_k$ dies out. At level $j_{i+1}$, the label $m$ will return to a descendant of $U_k$, and $m$ will be $gl$-successful. □

## Determining the Lexicographically Minimal Descendant

Recall that the definition of the labeling $gl$ involves the computation of the call $\texttt{lmd}(m, i)$, which is the class with the minimal profile among all the descendants of $\texttt{first}(m)$ on level $i$. Finding $\texttt{lmd}(m, i)$ requires knowing the descendants of $\texttt{first}(m)$ on level $i$. We show how to store this information with a partial order over classes. Using this partial order, we can determine, for every label $m$ that occurs on level $i$, the class $\texttt{lmd}(m, i + 1)$ using only information about levels $i$ and $i + 1$ of $T^{G'}$. Lemma 6.9. (6) implies that we can safely restrict ourselves to labels that occur on level $i$. The partial order is denoted $\lessdot_i$ and is defined below.

**Definition 6.11.** For two classes $U$ and $V$ on level $i$ of $T^{G'}$, we have that $U \lessdot_i V$ iff $V$

is a descendant of $\texttt{first}(gl(U))$. Also, $U \lessdot_i V$ when $U \lesseqdot_i V$ and $U \neq V$.

**Lemma 6.12.** *For a class $U$ on level $i$,*

$$\texttt{lmd}(gl(U), i+1) = \texttt{min}_{\preceq_{i+1}}(\{V' \mid \textit{for some } V,\ U \lesseqdot_i V,\ V' \textit{ is a child of } V\}).$$

*Proof.* We prove that the set $\{V' \mid V'$ is a child of V, $U \lesseqdot_i V\}$ contains every descendant of $\texttt{first}(gl(U))$ on level $i+1$, and thus that its minimal element is $\texttt{lmd}(gl(U), i+1)$. Let $V'$ be a class on level $i+1$, with parent $V$ on level $i$. If $U \lesseqdot_i V$, then $V$ is a descendant of $\texttt{first}(gl(U))$ and $V'$ is likewise a descendant of $\texttt{first}(gl(U))$. Conversely, as $gl(U)$ exists on level $i$, if $V'$ is a descendant of $\texttt{first}(gl(U))$, then its parent $V$ must also be a descendant of $\texttt{first}(gl(U))$ and $U \lesseqdot_i V$. □

We have thus shown that for a label $m$ and level $i$, we can determine $\texttt{lmd}(m, i+1)$ given only the classes on levels $i$ and $i+1$ and the partial order $\lessdot_i$ . We now demonstrate how to determine the partial order $\lesseqdot_{i+1}$ from the same information.

**Lemma 6.13.** *Let $U'$ and $V'$ be two classes on level $i+1$, so that $U' \neq V'$. Let $V$ be the parent of $V'$. We have that $U' \lesseqdot_{i+1} V'$ iff there exists a class $U$ on level $i$ so that $gl(U) = gl(U')$ and $U \lesseqdot_i V$.*

*Proof.* If there is no class $U$ on level $i$ so that $gl(U) = gl(U')$, then $U' = \texttt{first}(gl(U'))$. Since $V'$ is not a descendant of $U'$, this would preclude $U' \lesseqdot_{i+1} V'$. Therefore such a class $U$ must exist, and $U \lesseqdot_i V$ iff $V$ is a descendant of $\texttt{first}(gl(U))$, which is true iff $V'$ is a descendant of $\texttt{first}(gl(U'))$: the definition of $U' \lesseqdot_{i+1} V'$. □

## Using A Fixed Set of Labels

As defined, the labeling function $l$ uses an unbounded number of labels. We here inductively define a sequence of labelings, $l_i$, each from the $i$th level of $T^{G'}$ to $\{0, \ldots, 2|Q_{\mathbb{B}}|\}$.

We thus demonstrate how to use a bounded number of labels, each defined with respect to a bounded amount of information about $T^{G'}$. Formally, we define a sequence of functions $l_i$ from the classes on level $i$ of $T^{G'}$ to $\{0, \ldots, 2|Q_{\mathcal{B}}|\}$, exploiting the relation $\lessdot_i$ between classes. As a base case, there is only one equivalence class $U$ on level 0 of $T^{G'}$, and define $l_0(U) = 0$.

Inductively, given the set of classes $\mathcal{U}_i$ on level $i$, the function $l_i$, and the set of classes $\mathcal{U}_{i+1}$ on level $i+1$, we define $l_{i+1}$ as follows. Since labels can be reused, we can no longer simply rely on numerical order among labels. For $U \in \mathcal{U}_i$, define the $\lessdot_i$-nephew of $U$ as $\mathtt{neph}_i(U) = \mathtt{min}_{\preceq_{i+1}}(\{V' \mid \text{for some } V,\ U \lessdot_i V, \text{ and } V' \text{ is a child of } V\})$. Note that, by Lemma 6.12, it holds that $\mathtt{lmd}(gl(U), i+1) = \mathtt{neph}_i(U)$. For $U' \in \mathcal{U}_{i+1}$, we define the $\lessdot_i$-uncles of $U'$ as $\mathtt{unc}_i(U') = \{U \mid U' = \mathtt{neph}_i(U)\}$. As we prove below, $\mathtt{unc}_i$ corresponds to $\mathtt{labels}$. Recall that the set of unused labels $\mathrm{FL}(l_i)$ is $\{m \mid m \text{ is not in the range of } l_i\}$. As $T^{G'}$ has bounded width $|Q_{\mathcal{B}}|$, we have that $|Q_{\mathcal{B}}| \leq |\mathrm{FL}(l_i)|$. Let $\mathtt{mj}_{i+1}$ be the $\langle \preceq_{i+1}, < \rangle$-minjection from $\{U' \text{ on level i+1} \mid \mathtt{unc}_i(U') = \emptyset\}$ to $\mathrm{FL}(l_i)$. Finally, define the labeling $l_{i+1}$ as

$$l_{i+1}(U') = \begin{cases} l_i(\mathtt{min}_{\preceq_i}(\mathtt{unc}_i(U'))) & \text{if } \mathtt{unc}_i(U') \neq \emptyset, \\ \mathtt{mj}_{i+1}(U') & \text{if } \mathtt{unc}_i(U') = \emptyset. \end{cases}$$

Say that a label $m$ *dies in* $l_i$ if $m \notin \mathrm{FL}(l_{i-1})$, but $m \in \mathrm{FL}(l_i)$. Adjusting the definition of success to the new labeling, we say that a label $m$ *succeeds in* $l_i$ if there is a class $U$ on level $i-1$ and a class $U'$ on level $i$ such that $l_{i-1}(U) = l_i(U') = m$ and $U'$ is either the $F$-child of $U$ or is not a child of $U$.

To show a correlation between the labeling in Section 6.2.2 and the labeling here, we define a mapping, $g$, from the labels of $l$ to $\{0, \ldots, 2|Q_{\mathcal{B}}|\}$. For a label $m$, where $\mathtt{first}(m)$ occurs on level $i$, let $g(m) = l_i(\mathtt{first}(m))$.

**Lemma 6.14.** *Consider a class $U'$ on level $i+1$. The following hold:*

*(1)* $\texttt{labels}(U') \cap \{gl(V) \mid V \textit{ on level } i\} = \{gl(U) \mid U \in \texttt{unc}_i(U')\}$.

*(2)* $\texttt{labels}(U') = \emptyset$ *iff* $\texttt{unc}_i(U') = \emptyset$.

*Proof.* (1) Let $U$ be a class on level $i$. By definition, $gl(U) \in \texttt{labels}(U')$ iff $U' = \texttt{lmd}(gl(U), i+1)$. By Lemma 6.12, it holds that $\texttt{lmd}(gl(U), i+1) = \texttt{neph}_i(U)$. By the definition of $\texttt{unc}_i$, we have that $U' = \texttt{neph}_i(U)$ iff $U \in \texttt{unc}_i(U')$. Thus every label in $\texttt{labels}(U')$ that occurs on level $i$ labels some node in $\texttt{unc}_i(U')$.

(2) If $\texttt{unc}_i(U') \neq \emptyset$, then (1) implies $\texttt{labels}(U') \neq \emptyset$. In other direction, let $m = \texttt{min}(\texttt{labels}(U'))$. By Lemma 6.9(6), there is a $U$ on level $i$ so that $gl(U) = m$, and by part (1) $U \in \texttt{unc}_i(U')$. □

**Lemma 6.15.** *For classes $U$ on level $i$ and $U'$ on level $i+1$, if $gl(U) = gl(U')$, then $l_i(U) = l_{i+1}(U') = g(gl(U))$.*

*Proof.* Let $k$ be the number of levels between $U$ and $\texttt{first}(gl(U))$. We prove this lemma by induction over $k$. As a base case, if $k = 0$, then $U = \texttt{first}(gl(U))$ and by definition $g(gl(U)) = l_i(U)$. Inductively, assume $k > 0$, and assume this lemma holds for every $V$ at most $k-1$ steps removed from $\texttt{first}(gl(V))$. Since $k > 0$, then $U \neq \texttt{first}(gl(U))$. Let $V$ be the node on level $i-1$ such that $gl(V) = gl(U)$. By the inductive hypothesis, $l_{i-1}(V) = l_i(U)$. Further, since $\texttt{first}(gl(V)) = \texttt{first}(gl(U))$, we have $l_i(U) = g(gl(U))$. We now show that $U = \texttt{min}_{\preceq_i}(\texttt{unc}_i(U'))$.

As $gl(U) = gl(U')$, we have that $gl(U) \in \texttt{labels}(U')$. By Lemma 6.14, this implies $U \in \texttt{unc}_i(U')$. To prove that $U = \texttt{min}_{\preceq_i}(\texttt{unc}_i(U'))$, let $V \in \texttt{unc}_i(U')$ be another class on level $i$. By Lemma 6.14, this implies $gl(V) \in \texttt{labels}(U')$, and thus $gl(U) < gl(V)$. As $U'$ is a descendant of both $\texttt{first}(gl(U))$ and $\texttt{first}(gl(V))$, one is a descendant of the other. Since $gl(U) < gl(V)$, by Lemma 6.9.(5) it must be that $\texttt{first}(gl(V))$ is a descendant of $\texttt{first}(gl(U))$. Thus $V$ is a descendant of $\texttt{first}(gl(U))$, and by Lemma 6.9.(3) we have $U \preceq V$. Therefore $U = \texttt{min}_{\preceq_i}(\texttt{unc}_i(U'))$, and $l_{i+1}(U') = l_i(U)$. □

**Corollary 6.16.** *For every class $U$ on level $i$, it holds that $l_i(U) = g(gl(U))$.*

**Theorem 6.17.** *$T^{G'}$ has an accepting branch iff there is a label $m$ where $\{i \mid m$ dies in $l_i\}$ is finite, and $\{i \mid m$ succeeds in in $l_i\}$ is infinite.*

*Proof.* We prove a relation with Theorem 6.10. For the first direction, let $m$ be a label that is $gl$-successful infinitely often. We prove that $g(m)$ dies in only finitely many $l_i$, and succeeds in infinitely many $l_j$. Let $U$ on level $j$ be $\texttt{first}(m)$. First, as $m$ occurs on every level $k > j$, Lemma 6.15 implies $g(m)$ occurs on $k$, and thus $g(m)$ does not die in $l_k$. Second, let $k > j$ be a level on which $m$ is $gl$-successful. This implies there exist classes $U$ on level $k-1$ and $U'$ on level $k$, so that $gl(U) = gl(U') = m$ and $U'$ is not the non-$F$-child of $U$. Lemma 6.15 implies that $l_{k-1}(U) = l_k(U') = g(m)$, and thus that $g(m)$ succeeds in $l_k$. We thus conclude $g(m)$ succeeds in infinitely many $l_k$.

For the other direction, let $m'$ be a label that dies in $l_i$ for finitely many $i$, and succeeds in $l_i$ for infinitely many $i$. Since $m'$ dies only finitely often, there is some level after which $m'$ does not die. Let $j$ be the first level after which $m'$ ceases dying, on which $m'$ occurs. This implies $m'$ occurs on ever level $k > j$. Let $U$ on level $j$ be such that $l_j(U) = m'$. Since $m'$ does not occur on $j-1$, it must be that $\texttt{unc}_j(U) = \emptyset$. Thus $\texttt{labels}(U) = \emptyset$, and there is a label $m$ in $l$ so that $U = \texttt{first}(m)$, and $g(m) = m'$. By assumption, there are infinitely many $k > j$ so that $m'$ succeeds in $l_k$. On each of these $k$'s, there is a class $U$ on level $k-1$ and $U'$ on level $k$ so that $l_{k-1}(U) = m'$, $l_k(U') = m'$, and $U'$ is not the non-$F$-child of $U$. By Corollary 6.16, $m = gl(U)$ is $gl$-successful on level $k$, and $m$ is $gl$-successful infinitely often. □

## 6.3 The Determinization Construction

We now demonstrate a determinization construction for Büchi automata. Let $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, Q^{in}_{\mathcal{B}}, \rho_{\mathcal{B}}, F_{\mathcal{B}} \rangle$ be an NBW. For clarity, we call the states of our deterministic automaton *macrostates*. Each macrostate encodes one level of $T^{G'}$. Macrostates are four-tuples $\langle S, \preceq, l, \leqslant \rangle$, where $\langle S, \preceq \rangle$ is a preordered subset of $Q_{\mathcal{B}}$, $l \colon S \to \{0, \ldots, 2|Q_{\mathcal{B}}|\}$ is a labeling, and $\leqslant \subseteq \preceq$ is another preorder over $S$. For two states $q$ and $r$ in $Q_{\mathcal{B}}$, we say that $q \approx r$ if $q \preceq r$ and $r \preceq q$. We constrain the labeling $l$ so that it characterizes the equivalence classes of $S$ under $\preceq$. That is, $q \approx r$ iff $l(q) = l(r)$. Further, we constrain $\leqslant$ to be a partial order over the equivalence classes of $\preceq$. That is, if $q \approx r$, $s \approx t$, and $q \leqslant s$, then $r \leqslant t$.

**Transitions:** Recall that we define transitions with respect to preordered subsets as follows: $\rho_{S,\preceq}(q, \sigma) = \{r \in \rho_{\mathcal{B}}(q, \sigma) \mid \text{for every } q' \in S, \text{if } r \in \rho_{\mathcal{B}}(q', \sigma) \text{ then } q' \preceq q\}$. When a state has multiple incoming $\sigma$-transitions from $S$, the relation $\rho_{S,\preceq}$ keeps only the transition from states maximal under the $\preceq$ relation. For every $q' \in \rho_{\mathcal{B}}(S, \sigma)$, the set $\{q \in S \mid q' \in \rho_{S,\preceq}(q, \sigma)\}$ is an equivalence class under $\preceq$. Further, $\rho_{\mathcal{B}}(S, \sigma) = \rho_{S,\preceq}(S, \sigma) = \bigcup_{q \in S} \rho_{S,\preceq}(q, \sigma)$; thus, for every $q' \in \rho_{\mathcal{B}}(S, \sigma)$, there exist $q \in S$ such that $q' \in \rho_{S,\preceq}(q, \sigma)$.

For $\sigma \in \Sigma$, we define the $\sigma$-successor of $\langle S, \preceq, l, \leqslant \rangle$ to be $\langle S', \preceq', l', \leqslant' \rangle$, as follows. First, let $\langle S', \preceq' \rangle$ be the $\sigma$-successor of $\langle S, \preceq \rangle$, as defined in 5.2.2. Second, define the labeling $l'$ as follows. For $q \in S$, define the *$\leqslant$-nephews of* $q$ to be $\mathtt{neph}(q, \sigma) = \mathtt{min}_{\preceq'}(\{r' \mid \text{exists } r \in S \text{ such that } q \leqslant r \text{ and } r' \in \rho_{S,\preceq}(r, \sigma)\})$. Conversely, for $r' \in S'$ let the *$\leqslant$-uncles of* $r'$ be $\mathtt{unc}(r', \sigma) = \mathtt{min}_{\preceq}(\{q \mid r' \in \mathtt{neph}(q, \sigma)\})$. Intuitively, we want to give the label of a state in $S$ to its nephews, so a state in $S'$ inherits the label from its uncles. If a state in $S'$ does not have uncles, it should get a fresh label. Let

$\mathrm{FL}(l) = \{m \mid m \text{ not in the range of } l\}$ be the free labels in $l$, and let $\mathtt{mj}$ be the $\langle \preceq', < \rangle$-minjection from $\{r' \in S' \mid \mathtt{unc}(r', \sigma) = \emptyset\}$ to $\mathrm{FL}(l)$, where $<$ is the standard order on $\{0, \ldots, 2|Q_{\mathcal{B}}|\}$. For a $r' \in S'$, let

$$l'(r') = \begin{cases} l(q),\ q \in \mathtt{unc}(r', \sigma) & \text{if } \mathtt{unc}(r', \sigma) \neq \emptyset, \\ \mathtt{mj}(r') & \text{if } \mathtt{unc}(r', \sigma) = \emptyset. \end{cases}$$

Third, define the preorder $\lessdot'$ as follows. For states $q', r' \in S'$, define $q' \lessdot' r'$ iff $q' \approx' r'$ or there exists $q, r \in S$ so that $r' \in \rho_{S,\preceq}(r, \sigma)$, $q \in \mathtt{unc}(q')$, and $q \lessdot r$. Intuitively, the labeling $l'$ depends on recalling which states descend from the first equivalence class with a given label, and $\lessdot'$ keeps track of these descendants.

Lemma 6.18, proven in Section 6.3.1, demonstrates that $\sigma$-successors of macrostates are macrostates.

**Lemma 6.18.** *For a macrostate $\langle S, \preceq, l, \lessdot \rangle$ and $\sigma \in \Sigma$, the $\sigma$-successor $\langle S', \preceq', l', \lessdot' \rangle$ of $\langle S, \preceq, l, \lessdot \rangle$ is a macrostate.*

**Acceptance Condition:** Let $\mathbf{Q}$ be the set of macrostates. For $\sigma \in \Sigma$, label $m \in \{0, \ldots, 2|Q_{\mathcal{B}}|\}$, macrostate $\mathbf{q} = \langle S, \preceq, l, \lessdot \rangle \in \mathbf{Q}$, and its $\sigma$-successor $\mathbf{q}' = \langle S', \preceq', l', \lessdot' \rangle$, let $R = \{r \in S \mid l(r) = m\}$ and $R' = \{r' \in S' \mid l'(r') = m\}$. We say that $m$ *dies in* $\langle \mathbf{q}, \mathbf{q}' \rangle$ when $R \neq \emptyset$ but $R' = \emptyset$. We say that $m$ *succeeds in* $\langle \mathbf{q}, \mathbf{q}' \rangle$ when $R \neq \emptyset$, $R' \neq \emptyset$, and either $R' \subseteq F_{\mathcal{B}}$ or $\rho_{S,\preceq}(R, \sigma) \cap R' = \emptyset$. Intuitively, $m$ succeeds either when the branch it is following visits $F$-states, or the branch dies and it moves to another branch.

**Definition 6.19.** Define the DREW automaton $D^R(\mathcal{B})$ to be $\langle \Sigma, \mathbf{Q}, \mathbf{Q}^{in}, \rho_{\mathbf{Q}}, \alpha \rangle$, where:

- $\mathbf{Q}^{in} = \{\langle Q^{in}, \preceq_0, l_0, \lessdot_0 \rangle\}$, where:
  - $\preceq_0 = \lessdot_0 = Q^{in} \times Q^{in}$.

  - $l_0(q) = 0$ for all $q \in Q^{in}$.
- For $\mathbf{q} \in \mathbf{Q}$ and $\sigma \in \Sigma$, let $\rho_{\mathbf{Q}}(\mathbf{q}, \sigma) = \{\mathbf{q}'\}$, where $\mathbf{q}'$ is the $\sigma$-successor of $\mathbf{q}$.
- $\alpha = \langle G_0, B_0 \rangle, \ldots, \langle G_{2|Q_{\mathcal{B}}|}, B_{2|Q_{\mathcal{B}}|} \rangle$, where for a label $m \in \{0, \ldots, 2|Q_{\mathcal{B}}|\}$:
  - $G_m = \{\langle \mathbf{q}, \mathbf{q}' \rangle \mid m \text{ succeeds in } \langle \mathbf{q}, \mathbf{q}' \rangle\}$.
  - $B_m = \{\langle \mathbf{q}, \mathbf{q}' \rangle \mid m \text{ dies in } \langle \mathbf{q}, \mathbf{q}' \rangle\}$.

### 6.3.1 Correctness and Blow-up

We now prove the correctness of the construction and demonstrate its blowup is comparable with known determinization constructions. First, we demonstrate that the $\sigma$-successors of macrostates are macrostates.

**Lemma 6.18.** *For every macrostate $\langle S, \preceq, l, \lessdot \rangle$, its $\sigma$-successor $\langle S', \preceq', l', \lessdot' \rangle$ is also a macrostate.*

*Proof.* As $\langle S, \preceq, l, \lessdot \rangle$ is a macrostate, we have $\preceq$ is a linear preorder, $\lessdot \subseteq \preceq$, and for every $q, r, s, t \in S$: $q \approx r$ iff $l(q) = l(r)$; $q \approx r$ iff $q \lessdot r$ and $r \lessdot q$; and if $q \approx r$, $s \approx t$, and $q \lessdot s$, then $r \lessdot t$. We must prove this also holds for $\lessdot'$, $\preceq'$, and $l'$ over states in $S'$. Below, let $q', r', s', t'$ be states in $S'$, and $q, r, s, t \in S$ be such that $q' \in \rho_{S,\preceq}(q, \sigma)$, $r' \in \rho_{S,\preceq}(r, \sigma)$, $s' \in \rho_{S,\preceq}(s, \sigma)$, and $t' \in \rho_{S,\preceq}(t, \sigma)$.

To demonstrate that $\preceq'$ is a linear preorder, we show it is reflexive, relates every two elements, and is transitive. That $\preceq'$ is reflexive follows from the definition. To show that $\preceq'$ relates every two elements, note that as $\preceq$ is a linear preorder, either $q \prec r$, $r \prec q$, or $q \approx r$. By the definition of $\preceq'$, either $q' \prec' r'$, $q' \approx' r'$, or $r' \prec' q$. To show that $\preceq'$ is transitive, assume $q' \preceq' r' \preceq' s'$. By definition of $\preceq'$ we then have $q \preceq r$ and $r \preceq s$. Since $\preceq$ is transitive, we have $q \preceq s$. In order for $q' \not\preceq' s'$, it would need to be that $q \approx s$, $q' \in F_{\mathcal{B}}$, and $s' \notin F_{\mathcal{B}}$. If $q \approx s$, then $q \approx r$ and $r \approx s$. Thus if $r' \in F_{\mathcal{B}}$, we would have $s' \preceq r'$, a contradiction. If $r' \notin F_{\mathcal{B}}$, we would have $r' \preceq q'$, a contradiction. Thus it cannot be the case that $q \approx s$, $q' \in F_{\mathcal{B}}$, and $s' \notin F_{\mathcal{B}}$, and either $q' \approx' s'$, or $q' \prec' s'$, and $\preceq'$ is transitive and a linear preorder.

Next, we prove the labeling must give unique labels to the equivalence classes of $S'$ under $\preceq$': that $q' \approx' r'$ iff $l'(q') = l'(r')$. By the above properties, if $q \approx r$, then $\texttt{neph}(q, \sigma) = \texttt{neph}(r, \sigma)$. Further, $\texttt{neph}(q, \sigma)$ is an equivalence class under $\preceq'$ or is empty. In one direction, let $q' \approx' r'$ and let $m = l'(q')$. That $q' \approx r'$ implies for every $q \in \texttt{unc}(q', \sigma)$ that $q' \approx' r'$, and thus $\texttt{unc}(q', \sigma) = \texttt{unc}(r', \sigma)$. If $\texttt{unc}(q', \sigma) = \emptyset$, then $\texttt{unc}(r', \sigma) = \emptyset$. As a minjection maps equivalent elements to the same value, we have $l(q') = \texttt{mj}(q') = \texttt{mj}(r') = l(r')$. Alternately, if $\texttt{unc}(q', \sigma) \neq \emptyset$ then $m = l(q)$ for $q \in \texttt{unc}(q', \sigma)$, and $q \in \texttt{unc}(r', \sigma)$, and $l(r') = m$.

Finally, we must demonstrate two things about $\lessdot'$: that $\lessdot' \subseteq \preceq'$, and that $\lessdot'$ is a partial order over the equivalence classes of $\preceq'$. Assume $q' \lessdot' r'$. If $q' \approx' r'$, then $q' \preceq' r'$. Otherwise there exists a $q_2 \in \texttt{unc}(q')$ so that $l(q_2) = l(q')$ and $q_2 \lessdot r$. This implies both $r', q' \in \texttt{neph}(q_2, \sigma)$. Since $l(q') = l(q_2)$, it must be that $q_2 \in \texttt{unc}(q', \sigma)$ and thus $q' \in \texttt{neph}(q_2, \sigma)$. Thus $q' \preceq' r'$, and $\lessdot' \subseteq \preceq'$. This implies $q' \approx' r'$ iff $q' \lessdot' r'$ and $r' \lessdot' q'$ It remains to show if $q' \approx' r'$, $s' \approx t'$, and $q' \lessdot' s'$, then $r' \lessdot' t'$. If $q' \approx' s'$, then $r' \approx' t'$ and $r' \lessdot' t'$. Otherwise there exists a $q_2$ so that $l(q_2) = l(q')$ and $q_2 \lessdot s$. Since $r' \approx' q'$, it holds that $l(r') = l(q_2)$. Since $s' \approx' t'$, it holds that $s \approx t$ and $q_2 \lessdot t$. Thus $r' \lessdot' t'$, and we have satisfied all requirements for $\langle S', \preceq', l', \lessdot' \rangle$ to be a macrostate. □

We now link the macrostates of $D^R(\mathcal{B})$ to the various relations over classes of $T^{G'}$. We can already use Lemma 5.10 to assert that, in a macrostate, the set of states $S$ and the linear preorder $\preceq$ correspond to the nodes on a level $i$ of $G'$ and the preorder $\preceq_i$. Further, the edges in $G'$ correspond to transitions in $\rho_{S,\preceq}$.

**Lemma 6.20.** *Let $G$ be the run* DAG *of $\mathcal{B}$ on $w$ and let $\mathbf{q}_i = \langle S, \preceq, l, \lessdot \rangle$ be the $i$-th macrostate in the run of $D^R(\mathcal{B})$ on $w$:*

*(1)* $S = \{q \mid \langle q, i\rangle \in G\}$

*(2) For* $q, r \in S$*, it holds that* $q \preceq r$ *iff* $\langle q, i\rangle \preceq_i \langle r, i\rangle$.

*(3) For* $q \in S$ *and* $q' \in Q_{\mathcal{B}}$*, it holds that* $q' \in \rho_{S,\preceq}(q, \sigma_i)$ *iff* $\langle\langle q, i\rangle, \langle q', i+1\rangle\rangle \in E'$.

*Proof.* These follow from, and actually simply restate, Lemma 5.10.(1), (2), and (3). □

Lemma (1) shows that the partial order $\lessdot_i$ over classes corresponds to the preorder $\lessdot$ in macrostates of $D^R(\mathcal{B})$, and the labeling $l$ over classes corresponds to the labeling $l_i$ in macrostates of $D^R(\mathcal{B})$.

**Lemma 6.21.** *Let* $G$ *be the run* DAG *of* $\mathcal{B}$ *on* $w$ *and* $\mathbf{q}_i = \langle S, \preceq, l, \lessdot\rangle$ *be the ith macrostate in the run of* $D^R(\mathcal{B})$ *on* $w$.

*(1) For* $q, r \in S$ *it holds that* $q \lessdot r$ *iff* $[\langle q, i\rangle] \lessdot_i [\langle r, i\rangle]$

*(2) For* $q \in S$*, it holds that* $l(q) = l_i([\langle q, i\rangle])$.

*Proof.* We prove this by induction over $i$. As a base case, for $i = 0$, we have $S = Q^{in}_{\mathcal{B}}$, $\lessdot = Q^{in}_{\mathcal{B}} \times Q^{in}_{\mathcal{B}}$, and $l(q) = 0$ for every $q \in S$. By definition, the 0th level of $G'$ is $\{\langle q, 0\rangle \mid q \in Q^{\in}_{\mathcal{B}}\}$. As $Q^{in}_{\mathcal{B}} \cap F_{\mathcal{B}} = \emptyset$, for every $u, v$ on level 0 of $G'$ $h_u = 0 = h_v$ and $u \preceq_0 v$. Since there is only one equivalence class $U$, we have $U \lessdot_0 U$, and $l(U) = 0$.

Inductively, assume this holds for $\mathbf{q}_i = \langle S, \preceq, l, \lessdot\rangle$, and let $\mathbf{q}_{i+1} = \langle S', \preceq', l', \lessdot'\rangle$ be the $\sigma$-successor of $\mathbf{q}_i$. Note by Lemma 6.18 that $l'$ gives unique labels to the equivalence classes of $\preceq'$, and $\lessdot'$ is a partial order over the equivalence classes of $\preceq'$.

**Proof of (2)** For $q' \in S'$, we prove $l'(q') = l_{i+1}([\langle q', i+1\rangle])$ as follows. First, by definition for $q \in S$ and $r' \in S'$, $r' \in \texttt{neph}(q, \sigma)$ if there exists $r \in S$ so that $q \lessdot r$ and $r' \in \rho_{S,\preceq}(r, \sigma_i)$. By Lemma 6.20(3), the inductive hypothesis, and the definition of $T^{G'}$, this holds if there is a $V$ so that $[\langle r', i+1\rangle]$ is a child of $V$ and $[\langle q, i\rangle] \lessdot_i V$: the definition of $[\langle r', i+1\rangle] \in \texttt{neph}_i([\langle q, i\rangle])$. Thus $\texttt{neph}_i([\langle q, i\rangle]) = \{[\langle r', i+1\rangle] \mid r' \in \texttt{neph}(q, \sigma)\}$. By

Lemma (2) this implies for every $r' \in S'$, $\mathtt{unc}_i([\langle r', i{+}1\rangle]) = \{[\langle q, i\rangle] \mid q \in \mathtt{unc}(r', \sigma)\}$. We have two cases. If $\mathtt{unc}(q', \sigma) \neq \emptyset$, then $\mathtt{unc}_i([\langle q', i+1\rangle]) \neq \emptyset$, and $l'(q') = l(q)$ for $q \in \mathtt{unc}(q', \sigma)$. By the inductive hypothesis and Lemma 6.20(2) this implies $[\langle q, i\rangle] = \mathtt{min}_{\preceq_i}(\mathtt{unc}_i([\langle q', i+1\rangle]))$ and $l_{i+1}([\langle q', i+1\rangle]) = l_i([\langle q, i\rangle]) = l(q)$. Alternately, if $\mathtt{unc}(q', \sigma) = \emptyset$, then $\mathtt{unc}_i([\langle q', i+1\rangle]) = \emptyset$. The inductive hypothesis implies that $\mathrm{FL}(l_i)$, the set of unused labels in $l_i$, is identical to $\mathrm{FL}(l)$, the set of unused labels in $l$. Thus the $\langle \preceq_{i+1}, <\rangle$-minjection from the classes on level $i+1$ of $T^{G'}$ to $\mathrm{FL}(l_i)$ corresponds to the $\langle \preceq', <\rangle$-minjection from $S'$ to $\mathrm{FL}(l)$, and $\mathtt{mj}(q') = \mathtt{mj}_{i+1}([\langle q, i{+}1\rangle])$.

**Proof of (1)** There are two cases in which $q' \leqq' r'$. First, if $q' \approx' r'$, then by Lemma 6.20(2) $[\langle q', i+1\rangle] = [\langle r', i+1\rangle]$ and, as $\leqq_{i+1}$ is reflexive, $[\langle q', i+1\rangle] \leqq_{i+1} [\langle r', i+1\rangle]$. Otherwise $q' \not\approx' r'$ and $q' \leqq' r'$ iff there exists $r, q_2 \in S$ so that $q_2 \in \mathtt{unc}(q')$, $r' \in \rho_{S,\preceq}(r, \sigma_i)$, and $q_2 \leqq r$. By Lemma 6.20.(2) and (3) this entails $q' \leqq' r'$ iff there exists $U$ and $V$ so that $V$ is the parent of $[\langle r', i+1\rangle]$, $l_i(U) = l_{i+1}([\langle q', i+1\rangle])$, and $U \leqq_i V$. By Lemmas 6.13 and 6.15, this is precisely the condition under which $[\langle q', i+1\rangle] \leqq_{i+1} [\langle r', i+1\rangle]$. $\square$

Lemma 6.22 demonstrate the correlation between $l_i$ and the labelings in macrostates of $D^R(\mathcal{B})$, and show that success of a label in $l_i$ corresponds to success of that label in the run of $D^R(\mathcal{B})$.

**Lemma 6.22.** *Let $G$ be the run* DAG *of $\mathcal{B}$ on $w$ and let $\mathbf{q}_i$ and $\mathbf{q}_{i+1}$ be two consecutive macrostates in the run of $D^R(\mathcal{B})$ on $w$. For every label $m$, we have that $m$ dies in $l_{i+1}$ iff $m$ dies in $\langle \mathbf{q}_i, \mathbf{q}_j\rangle$, and $m$ succeeds in $l_{i+1}$ iff $m$ succeeds in $\langle \mathbf{q}_i, \mathbf{q}_j\rangle$.*

*Proof.* Let $\mathbf{q}_i = \langle S, \preceq, l, \leqq\rangle$ and $\mathbf{q}_{i+1} = \langle S', \preceq', l', \leqq'\rangle$. Recall that, with respect to $i$, $R = \{r \in S \mid l(r) = m\}$ and $R' = \{r' \in S' \mid l'(r') = m\}$. By Lemma 6.18, we have that $R$ is an equivalence class under $\preceq$ and $R'$ is an equivalence class under $\preceq'$. By definition, $m$ dies in $l_i$ when $m$ is in the range of $l_i$, but not in the range of $l_{i+1}$. By Lemma 6.21(2) this is true iff $R \neq \emptyset$, but $R' = \emptyset$: the definition of $m$ dies in $\langle \mathbf{q}_i, \mathbf{q}_{i+1} \rangle$.

Similarly, $m$ succeeds in $l_{i+1}$ if there are classes $U$ on level $i-1$ and $U'$ on level $i$ so that $l_i(U) = l_{i+1}(U') = m$, and $U'$ is not the non-$F$-child of $U$. By Lemmas 6.20(1) and 6.21(2), a $U$ and $U'$ exist so that $l_i(U) = l_{i+1}(U') = m$, iff $U = \{\langle r, i \rangle \mid r \in R\}$ and $U' = \{\langle r', i+1 \rangle \mid r' \in R'\}$. This entails that $m$ succeeds in $l_{i+1}$ iff $R \neq \emptyset$ and $R' \neq \emptyset$, and either $U'$ is an $F$-class, or $U'$ is not a child of $U$. If $U'$ is an $F$-class then $R' \subseteq F$. If $U'$ is not a child of $U$, then by the definition of $T^{G'}$ and Lemmas (3) there is no $r \in R$, $r' \in R'$ where $r' \in \rho_{S,\preceq}(r, \sigma_i)$. This entails $\{\rho_{S,\preceq}(r, \sigma) \mid r \in R\} \cap R' = \emptyset$. We conclude that $m$ succeeds in $l_{i+1}$ iff $m$ succeeds in $\langle \mathbf{q}_i, \mathbf{q}_{i+1} \rangle$. $\square$

The correctness of $D^R(\mathcal{B})$ can now be proven with existing machinery, but we must first bound the number of preorders $\lessdot$ to bound the size of the automaton.

**Lemma 6.23.** *For a level $i$, the preorder $\lessdot_i$ is a tree order over the classes on level $i$ of $T^{G'}$.*

*Proof.* Let $\mathcal{U}$ be the set of classes on level $i$ of $T^{G'}$. By definition, $\lessdot_i$ is a tree order if for every $V \in \mathcal{U}$, the $\{U \mid U \leq V\}$ is totally ordered by $\lessdot_i$. Consider two classes $U \lessdot_i V$ and $W \lessdot_i V$. By definition, $V$ is a descendant of both $\texttt{first}(gl(U))$ and $\texttt{first}(gl(W))$. Since $T^{G'}$ is a tree, one of $\texttt{first}(gl(U))$ or $\texttt{first}(gl(W))$ is a descendant of the other. Without loss of generality, assume $\texttt{first}(gl(U))$ is a descendant of $\texttt{first}(gl(W))$. In this case, $W$ is a descendant of $\texttt{first}(gl(U))$, so $U \lessdot_i W$. $\square$

**Theorem 6.24.** *For a Büchi automaton $\mathcal{B}$ with $n$ states, $L(D^R(\mathcal{B})) = L(\mathcal{B})$ and $D^R(\mathcal{B})$ has $n^{O(n)}$ states.*

*Proof.* That $L(D^R(\mathcal{B})) = L(\mathcal{B})$ follows from Theorem 6.17 and Lemma 6.22. To bound the number of macrostates $\langle S, \preceq, l, \leqslant \rangle$, we observe that the number of subsets $S$ and linear orders $\preceq$ is $n^{O(n)}$ [Var80]. The number of labelings is likewise $n^{O(n)}$. By Lemma 6.23 and Lemma 6.21(1), $\leqslant$ is a tree-order over the equivalence classes of $S$ under $\preceq$. By Cayley's formula, the number of tree orders is bounded by $n^{n-2}$. Thus the number of macrostates is bounded by $n^{O(n)}$. □

### 6.3.2 Compact Determinization Constructions

We here present two variants of Definition 6.19, both of which only use a variation of macrostates where labels are restricted to $\{0, \ldots, |Q_{\mathcal{B}}| - 1\}$. The first still uses a Rabin condition, the second a parity condition. Define the sight of *tight macrostates* to be four-tuples $\langle S, \preceq, l, \leqslant \rangle$, where $S$, $\preceq$, and $\leqslant$ are defined as for normal macrostates, and where $l : S \to \{0, \ldots, |Q_{\mathcal{B}}| - 1\}$ is a tighter labeling. Let $\mathbf{Q}^t$ be the set of tight macrostates.

**Tight Rabin Variant:** Given a tight macrostate $\mathbf{q} \in \mathbf{Q}^t$ and $\sigma \in \Sigma$, define the *Rabin $\sigma$-successor of* $\mathbf{q}$ to be $\mathbf{q}' = \langle S', \preceq', l', \leqslant' \rangle$ where $S'$, $\preceq'$, and $\leqslant'$ are defined as in Section 6.3, and $l'$ is defined as follows:

(1) For $q \in S$, let $\texttt{neph}(q, \sigma) = \texttt{min}_{\preceq'}(\{r' \mid \text{exists } r \in S,\ q \leqslant r,\ r' \in \rho_{S,\preceq}(r, \sigma)\})$, as in Section 6.3.

(2) For $r' \in S'$, let $\texttt{unc}(r', \sigma) = \texttt{min}_{\preceq}(\{q \mid r' \in \texttt{neph}(q, \sigma)\})$, as in Section 6.3.

(3) $\mathrm{FL}(l) = \{m \mid m \text{ not in the range of } l\} \cup \{l(q) \mid \text{ for every } r' \in S', q \notin \texttt{unc}(r', \sigma)\}$.

(4) $\texttt{mj}$ is the $\langle \preceq', < \rangle$-minjection from $\{r' \in S' \mid \texttt{unc}(r', \sigma = \emptyset\}$ to $\mathrm{FL}(l)$.

(5) For $r' \in S'$, let $l'(r') = \begin{cases} l(q),\ q \in \texttt{unc}(r', \sigma) & \text{if } \texttt{unc}(r', \sigma) \neq \emptyset, \\ \texttt{mj}(r') & \text{if } \texttt{unc}(r', \sigma) = \emptyset. \end{cases}$

For $\sigma \in \Sigma$ and label $m \in \{0, \ldots, |Q_{\mathcal{B}}| - 1\}$, given a tight macrostate $\mathbf{q} = \langle S, \preceq, l, \leqslant \rangle \in$

$\mathbf{Q}^t$ and its Rabin $\sigma$-successor $\mathbf{q}' = \langle S', \preceq', l', \lessdot' \rangle$ let $R = \langle r \in S \mid l(r) = m \rangle$ and $R' = \langle r' \in S' \mid l'(r') = m \rangle$. Say that $m$ ***Rabin-dies in*** $\langle \mathbf{q}, \mathbf{q}' \rangle$ when $R \neq \emptyset$ and $m \in \mathrm{FL}(l)$. Say that $m$ ***Rabin-succeeds in*** $\langle \mathbf{q}, \mathbf{q}' \rangle$ when it does not die in $\langle \mathbf{q}, \mathbf{q}' \rangle$, $R \neq \emptyset$, $R' \neq \emptyset$, and either $R' \subseteq F_{\mathcal{B}}$ or $\rho_{S,\preceq}(R, \sigma) \cap R' = \emptyset$.

**Definition 6.25.** Define the DREW automaton $D^T(\mathcal{B})$ to be $\langle \Sigma, \mathbf{Q}^t, \mathbf{Q}^{in}, \rho_{\mathbf{Q}}, \alpha \rangle$ where:

- $\mathbf{Q}^{in}$ is as defined in Definition 6.19
- For $\mathbf{q} \in \mathbf{Q}^t$ and $\sigma \in \Sigma$, $\rho_{\mathbf{Q}}(\mathbf{q}, \sigma) = \{\mathbf{q}'\}$ where $\mathbf{q}'$ is the Rabin $\sigma$-successor of $\mathbf{q}$.
- $\alpha = \langle G_0, B_0 \rangle, \ldots, \langle G_{|Q_{\mathcal{B}}|-1}, B_{2|Q_{\mathcal{B}}|-1} \rangle$ where for a label $m \in \{0, \ldots, 2|Q_{\mathcal{B}}|\}$:
  - $G_m = \{\langle \mathbf{q}, \mathbf{q}' \rangle \mid m$ Rabin-succeeds in $\langle \mathbf{q}, \mathbf{q}' \rangle\}$.
  - $B_m = \{\langle \mathbf{q}, \mathbf{q}' \rangle \mid m$ Rabin-dies in $\langle \mathbf{q}, \mathbf{q}' \rangle\}$

**Theorem 6.26.** *For a Büchi automaton $\mathcal{B}$, $L(D^T(\mathcal{B})) = L(\mathcal{B}))$.*

*Proof.* For every word $w$, we show that the run $\mathbf{q}_0, \mathbf{q}_1, \ldots$ of $D^R(\mathcal{B})$ on $w$ is accepting iff the run $\mathbf{q}^p_0, \mathbf{q}^p_1, \ldots$ of $D^P(\mathcal{B})$ on $w$ is accepting. For convenience, let $\mathbf{q}_i = \langle S_i, \preceq_i, l_i, \lessdot_i \rangle$. We first note that for every $i$, it holds that $\mathbf{q}^p_i == \langle S_i, \preceq_i, l^p_i, \lessdot_i \rangle$: that that is to say $\mathbf{q}_i$ and $\mathbf{q}^p_i$ match on $S_i$, $\preceq_i$, and $\lessdot_i$. For $S$ and $\preceq$, this is easy to see: the definitions of $S'$ and $\preceq$' are identical in $\sigma$-successors and Rabin-$\sigma$-successors. For $\lessdot$, this follows from the fact that $\lessdot'$ is defined solely with respect to $\rho_{S,\preceq}$ and $\mathtt{unc}$, which do not change from $\sigma$-successors to Rabin-$\sigma$-successors. We pause to note that, for every $i$ and $q \in S_i$, $q' \in S_{i+1}$, we have that $l_{i+1}(q') = l_i(q)$, iff $q \in \mathtt{unc}(q', \sigma_i)$, which holds iff both $l^p_{i+1}(q') = l^p_i(q)$ and $l^p_i(q) \not\in\in \mathrm{FL}(l^p_i)$.

In one direction, assume there is a label $m$ that dies in finitely many $\langle \mathbf{q}_i, \mathbf{q}_{i+1} \rangle$, and succeeds in infinitely many $\langle \mathbf{q}_i, \mathbf{q}_{i+1} \rangle$. We pause to note that, for every $i$ and $q \in S_i$, $q' \in S_{i+1}$, we have that $l_{i+1}(q') = l_i(q)$, iff $q \in \mathtt{unc}(q', \sigma_i)$, which holds iff both $l^p_{i+1}(q') = l^p_i(q)$ and $l^p_i(q) \not\in\in \mathrm{FL}(l^p_i)$. Let $j$ be the first index so that $m$ occurs in $\mathbf{q}_j$, but for every $k > j$, $m$ does not die in $\langle \mathbf{q}_k, \mathbf{q}_{k+1} \rangle$. Let $q \in S_j$ be such that $l_j(q) =$

$m$, and let $m' = l^p_j(q)$. For $k > j$, define $R_k = \{r \in S_k \mid l_k(r) = m\}$, and $R^p_k = \{r \in S_k \mid l^p_k(r) = m'\}$. Since $m$ does not die in $\langle \mathbf{q}_k, \mathbf{q}_{k+1}\rangle$, $R_k$ and $R_{k+1}$ are both non-empty, and by our above observations $m'$ does not Rabin-die in $\langle \mathbf{q}^p_k, \mathbf{q}^p_{k+1}\rangle$. Further, $R^p_k = R_k$, and $R^p_{k+1} = R_{k+1}$. This implies that if $m$ succeeds in $\langle \mathbf{q}_k, \mathbf{q}_{k+1}\rangle$, then $m'$ Rabin-succeeds $\langle \mathbf{q}^p_k, \mathbf{q}^p_{k+1}\rangle$. Thus $m'$ Rabin-dies in finitely many $\langle \mathbf{q}^p_i, \mathbf{q}^p_{i+1}\rangle$, and Rabin-succeeds in infinitely many $\langle \mathbf{q}^p_i, \mathbf{q}^p_{i+1}\rangle$, and $D^T(\mathcal{B})$ accepts $w$.

In the other direction if $D^T(\mathcal{B})$ accepts $w$, this implies is a label $m$ that Rabin-dies in finitely many $\langle \mathbf{q}^p_i, \mathbf{q}^p_{i+1}\rangle$, and Rabin-succeeds in infinitely many $\langle \mathbf{q}^p_i, \mathbf{q}^p_{i+1}\rangle$. Let $j$ be the first index so that $m$ occurs in $\mathbf{q}^p_j$, but for every $k > j$, $m$ does not Rabin-die in $\langle \mathbf{q}^p_k, \mathbf{q}^p_{k+1}\rangle$. Let $q \in S_j$ be such that $l^p_j(q) = m$, and let $m' = l_j(q)$. For $k > j$, define $R^p_k = \{r \in S_k \mid l^p_k(r) = m\}$, and $R_k = \{r \in S_k \mid l_k(r) = m'\}$, Since $m$ does not Rabin-die in $\langle \mathbf{q}^p_k, \mathbf{q}^p_{k+1}\rangle$, $m \notin F_{l^p_k}$ and $R_k$, $R_{k+1}$ are both non-empty. By our above observations $m'$ does not die in $\langle \mathbf{q}_k, \mathbf{q}_{k+1}\rangle$. Further, $R^p_k = R_k$, and $R^p_{k+1} = R_{k+1}$. This implies that if $m$ Rabin-succeeds in $\langle \mathbf{q}^p_k, \mathbf{q}^p_{k+1}\rangle$, then $m'$ succeeds $\langle \mathbf{q}_k, \mathbf{q}_{k+1}\rangle$. Thus $m'$ dies in finitely many $\langle \mathbf{q}_i, \mathbf{q}_{i+1}\rangle$, and succeeds in infinitely many $\langle \mathbf{q}_i, \mathbf{q}_{i+1}\rangle$, and $D^R(\mathcal{B})$ accepts $w$. $\square$

**Parity Variant** The parity variation simply shifts labels down, instead of giving arbitrary free labels to new nodes. This means labels in the automaton are no longer consistent with with the labels $l_i$ over $T^{G'}$. To simplify this, we use an intermediate labeling that keeps labels consistent between two levels, but can use the labels $\{|Q_{\mathcal{B}}|, \ldots 2|Q_{\mathcal{B}}|\}$. Given a tight macrostate $\mathbf{q} \in \mathbf{Q}^t$ and $\sigma \in \Sigma$, define the *parity $\sigma$-successor of* $\mathbf{q}$ to be $\mathbf{q}' = \langle S', \preceq', l', \lessdot' \rangle$ where $S'$, $\preceq'$, and $\lessdot'$ are defined as in Section 6.3, and $l'$ is defined as follows:

(1) For $q \in S$, let $\texttt{neph}(q, \sigma) = \texttt{min}_{\preceq'}(\{r' \mid \text{exists } r \in S,\ q \lessdot r,\ r' \in \rho_{S,\preceq}(r, \sigma)\})$

(2) For $r' \in S'$, let $\texttt{unc}(r', \sigma) = \texttt{min}_{\preceq}(\{q \mid r' \in \texttt{neph}(q, \sigma)\})$

(3) $\mathtt{mj}$ is the $\langle \preceq', < \rangle$-minjection from $\{r' \in S' \mid \mathtt{unc}(r', \sigma = \emptyset\}$ to $\{|Q_{\mathcal{B}}|, \ldots, 2|Q_{\mathcal{B}}|\}$

(4) For $r' \in S'$, define the intermediate labeling

$$l^{int}(r') = \begin{cases} l(q),\ q \in \mathtt{unc}(r', \sigma) & \text{if } \mathtt{unc}(r', \sigma) \neq \emptyset, \\ \mathtt{mj}(r') & \text{if } \mathtt{unc}(r', \sigma) = \emptyset. \end{cases}$$

(5) For $r' \in S'$, define the final labeling $l'(r') = |\{l^{int}(q') \mid l^{int}(q') < l^{int}(r')\}|$

For $\sigma \in \Sigma$ and label $m \in \{0, \ldots, |Q_{\mathcal{B}}| - 1\}$, given a tight macrostate $\mathbf{q} = \langle S, \preceq, l, \leqslant \rangle$ and its parity $\sigma$-successor $\mathbf{q}' = \langle S', \preceq', l', \leqslant' \rangle$ let $l^{int}$ be the intermediate labeling defined above. Let $R = \langle r \in S \mid l(r) = m \rangle$ and $R' = \langle r' \in S' \mid l^{int}(r') = m \rangle$. Note that $R'$ is defined with respect to the intermediate labeling. Say that a label $m$ *parity-dies in* $\langle \mathbf{q}, \mathbf{q}' \rangle$ if $m \in R$, but $m \notin R'$. Say that $m$ *parity-succeeds in* $\langle \mathbf{q}, \mathbf{q}' \rangle$ when $R \neq \emptyset$, $R' \neq \emptyset$, and either $R' \subseteq F_{\mathcal{B}}$ or $\rho_{S,\preceq}(R, \sigma) \cap R' = \emptyset$. Define the priority function $\gamma : \mathbf{Q}^t \times \mathbf{Q}^t \to \{1, \ldots, 2|Q_{\mathcal{B}}|\}$ so that $\gamma(\langle \mathbf{q}, \mathbf{q}' \rangle)$ is:

$$\mathtt{min}(\{2m + 2 \mid m \text{ parity-succeeds in } \langle \mathbf{q}, \mathbf{q}' \rangle\} \cup \{2m + 1 \mid m \text{ parity-dies in } \langle \mathbf{q}, \mathbf{q}' \rangle\})$$

**Definition 6.27.** Define the DPEW automaton $D^P(\mathcal{B})$ to be $\langle \Sigma, \mathbf{Q}^t, \mathbf{Q}^{in}, \rho_{\mathbf{Q}}, \gamma \rangle$ where:

- $\mathbf{Q}^{in}$ is as defined Definition 6.19
- For $\mathbf{q} \in \mathbf{Q}$ and $\sigma \in \Sigma$, $\rho_{\mathbf{Q}}(\mathbf{q}, \sigma) = \{\mathbf{q}'\}$ where $\mathbf{q}'$ is the parity $\sigma$-successor of $\mathbf{q}$.

**Theorem 6.28.** *For a Büchi automaton $\mathcal{B}$, $L(D^P(\mathcal{B})) = L(\mathcal{B}))$.*

*Proof.* As above, for every word $w$, we show that the run $\mathbf{q}_0, \mathbf{q}_1, \ldots$ of $D^R(\mathcal{B})$ on $w$ is accepting iff the run $\mathbf{q}_0^p, \mathbf{q}_1^p, \ldots$ of $D^P(\mathcal{B})$ on $w$ is accepting. For convenience, let $\mathbf{q}_i = \langle S_i, \preceq_i, l_i, \leqslant_i \rangle$. Again, it holds that for every $i$ $\mathbf{q}_i^p == \langle S_i, \preceq_i, l_i^p, \leqslant_i \rangle$: $\mathbf{q}_i$ and $\mathbf{q}_i^p$ match on $S_i$, $\preceq_i$, and $\leqslant_i$. For every $i$, let $l_i^{int}$ be the intermediate labeling defined above. However, it is no longer that case that the labels of a branch will be consistent from $\mathbf{q}_i^p$ to $\mathbf{q}_{i+1}^p$. Instead, we must look for consistency in the intermediate labeling. For every $i$ and

$q \in S_i$, $q' \in S_{i+1}$, we have that $l_{i+1}(q') = l_i(q)$ iff $l^{int}_{i+1}(q') = l^p_i(q)$. If $l^p_{i+1}(q') \neq l^{int}_{i+1}(q')$, this implies there was a label $n < l^p_i(q)$ that occurs in the range of $l^p_i$, but not in the range of $l^{int}_{i+1}$.

In one direction, assume there is a label $m$ that dies in finitely many $\langle \mathbf{q}_i, \mathbf{q}_{i+1}\rangle$, and succeeds in infinitely many $\langle \mathbf{q}_i, \mathbf{q}_{i+1}\rangle$. Let $j$ be the first index so that $m$ occurs in $\mathbf{q}_j$, but for every $k > j$, $m$ does not die in $\langle \mathbf{q}_k, \mathbf{q}_{k+1}\rangle$. For every $j' > j$, let $q'_j \in \S_{j'}$ be such that $l_{j'}(q_{j'}) = m$. Note that the values of $l^p_{j'}(q_{j'})$ can only decrease: new labels are only introduced above $|Q_\mathcal{B}|$, and $l^p_j(q_j) < |Q_\mathcal{B}|$. Thus at some point the labels of $q_{j'}$ cease decreasing, and reach a stable point. Let $k$ be this point, and let $m' = l^p_k(q_k)$. For a level $k' > k$, define $R_{k'} = \{r \in S_{k'} \mid l_{k'}(r) = m\}$, and $R^p_{k'} = \{r \in S_{k'} \mid l^{int}_{k'}(r) = m'\}$. Since the labels of $q_{k'}$ have stopped decreasing, we have that $R^p_{k'} = R_{k'}$. For every $k' > k$, it holds that $m'$ does not parity-die in $\langle \mathbf{q}^p_{k'}, \mathbf{q}^p_{k'+1}\rangle$. Further, every label $n < m'$ must occur on every level $k' > k$: otherwise $l^p_{k'}(q_{k'})$ would not equal $l^{int}_{k'}(q_{k'})$. Thus for every $k' > k$, there is no label $n < m'$ that parity-dies in $\langle \mathbf{q}^p_{k'}, \mathbf{q}^p_{k'+1}\rangle$. Therefore $\gamma(\mathbf{q}^p_{k'}, \mathbf{q}^p_{k'+1}) \geq 2m' + 1$. Now consider a level $k' > k$ where $m$ succeeds in $\langle \mathbf{q}_{k'}, \mathbf{q}_{k'+1}\rangle$. By the note above, $R^p_{k'} = R_{k'}$, and $m'$ parity-succeeds in $\langle \mathbf{q}^p_{k'}, \mathbf{q}^p_{k'+1}\rangle$, and $\gamma(\mathbf{q}^p_{k'}, \mathbf{q}^p_{k'+1}) = 2m' + 2$. We have thus shown that the smallest priority occurring infinitely often in $2m' + 2$, and thus $w$ is accepted by $D^P(\mathcal{B})$.

In the other direction if $D^P(\mathcal{B})$ accepts $w$, this implies is a label $m$ and level $j$ so that for every $k > j$, it holds $\gamma(\mathbf{q}^p_k, \mathbf{q}^p_{k+1}) \geq 2m + 2$, and for infinitely many $k > j$ it holds $\gamma(\mathbf{q}^p_k, \mathbf{q}^p_{k+1}) = 2m + 2$. As noted above, this implies for every $k > j$ and $n \leq m$, $n$ does not parity-die in $\langle \mathbf{q}^p_k, \mathbf{q}^p_{k+1}\rangle$, and for infinitely many $k > j$, $m$ parity-succeeds in $\langle \mathbf{q}^p_k, \mathbf{q}^p_{k+1}\rangle$. Thus we conclude that for every $k > j$ and $q \in S_k$, $l^p_k(q) = m$ iff $l^{int}_k(q) = m$. Let $q \in S_j$ be such that $l^p_j(q) = m$, and let $m' = l_j(q)$. For every $k > j$, let $R^p_k = \{r \in S_k \mid l^p_k(r) = m\}$, and let $R_k = \{r \in S_k \mid l_k(r) = m'\}$. Again, we have that

$R^p_k = R_k$, thus for every $k > j$, $m'$ does not die in $\langle \mathbf{q}_k, \mathbf{q}_{k+1} \rangle$, and for infinitely many $k > j$ we have $m'$ succeeds in $\langle \mathbf{q}_k, \mathbf{q}_{k+1} \rangle$. Thus $w$ is accepted by and $D^R(\mathcal{B})$. $\square$

# Chapter 7

# Conclusion and Future Work

Reactive systems are increasingly pervasive, and their correctness is of increasing importance. Formal verification for reactive systems is a vigorous field of research in computer science. Central to this is the creation of specifications that describe the properties we wish to verify. This work addresses the algorithmic limitations to using nondeterministic Büchi automata as specifications. In Chapter 3, we demonstrated the necessity of specializing complementation constructions into containment checking algorithms. Direct complementation is not viable. However, optimizations for the constructions do significantly reduce both the size of the complemented automaton and the time required to compute it. For the rank-based approach, the naive approach, which lacked tight rankings or Schewe's improved cut-point, is orders of magnitude larger on automata of size 6. Ramsey-based complementation does not have the same history of optimization. Only very recently did work begin on improving Ramsey-based complementation, instead of the containment checking procedure [BLO12]. Further investigations into the scalability of complementation constructions may provide surprising results.

In Chapter 4, we showed that neither the rank nor Ramsey-approach dominates on all problem configurations. Given the cheap availability of processing power, the best way to solve a problem would thus be to run both algorithms simultaneously. Preferable to running the algorithms in parallel would be to employ a *portfolio approach*. A portfolio approach attempts to predict which algorithm would perform better on a given problem [LBNA$^+$03]. To do this, we would have to examine the space of containment problems and discover significant attributes of problems. Transition and acceptance density are not the only observable attributes of an automaton, or even necessarily the most important ones. While they are significant for randomly generated problems, there is no reason to expect that transition and acceptance density are good indicators of difficulty for real-world problems. In the case of SAT solvers, over ninety pertinent attributes were found [NLBD$^+$04]. Machine-learning techniques were used to identify which features suggest which approach to SAT solving. The challenge that now faces us is discovering a similar body of features with which to characterize Büchi automata, and to create a corpus of automata to characterize. In addition to transition and acceptance density, attributes could include the density of initial states, the number of strongly connected components in the automata, and the density of strongly connecting components containing an accepting state.

Both Chapter 3 and 4 provide strong evidence that both worst-case complexity and examinations of the size of complemented automata are poor predictors of an algorithms performance, and no substitute for empirical analysis of derived algorithms. The massive gap in complexity between the rank and Ramsey-based approach did not manifest in any experiment. Further, complementation and universality testing exhibiting different behaviors over the terrain. While Ramsey-based complementation found transition densities of 1 to be the most difficult, Ramsey-based universality testing found transition densities of 1.5 to be harder, and had no problem with transition densities of 1. Similarly, rank-based complementation was more sensitive to acceptance density, and less sensitive to transition density, than rank-based universality checking. When dealing with problems where heuristics and optimizations are necessary for scalability, worst-case analysis simply does not provide a good metric on which to compare algorithms. Experimental analysis must be performed. To further the validity of the experimental analysis, we must develop better models on which to perform experiments. Random examples provide only so much insight, especially when we start considering advanced heuristics based on simulations relations in the input automata [ACC$^+$11]. These simulations relations reflect structure in the input automata, and it is known in the case of SAT problems that far more structure can be exploited in real-world than random problems [NLBD$^+$04].

The profile-based complementation construction of Chapter 5 provides exciting theoretical results and restricted nondeterminism. The ability to unify these two seemingly disparate approaches suggests it may be possible to use odd rankings for determinization, automata with other accepting conditions, and automata on infinite trees. Further, [BLO12] suggest that the Ramsey-based approach might also be unified with the rank and slice-based approaches. Doing so might yield further insights into what makes the Ramsey-based approach so surprisingly competitive. Profile-based complementation must also be extended into more concrete algorithms. As we demonstrated on the Ramsey and rank based approaches, explicit complementation does not scale as well as heuristically-improved containment-checking algorithms. Thus the reduced determinism of the profile-based approach can only be exploited by developing containment checking algorithms based on the profile-based complementation constructions. Heuristics then need to be develop for these containment checking algorithms, including perhaps a subsumption relation extending the work of Doyen and Raskin [DR07]. Further, our empirical analysis demonstrate that theoretical results, such as worst-case complexity, are not always reflected in real-world performance. Thus to truly evaluate the real-world effects of restricting nondeterminism, we must empirically test the algorithms we derive from the profile-based complementation construction.

Despite decades of improvement on Safra's original construction, every existing

Büchi determinization construction employs trees to encode macrostates. Chapter 6 describes a *relational* approach to determinization, using labeled preorders to encode macrostates. There is a connection between our macrostates and Safra trees. A step in the transition between Safra trees is to remove states if they appear in more than one node. This is analogous to the pruned transition relation $\rho_{S,\preceq}$. We also identified what additional information, namely the relation $\leq$, is encoded in Safra trees. This provides us with a mathematically crisp determinization construction that can be seen as an augmentation of the subset construction for automata on finite words. However, this approach is still nascent, and the bounds on the resulting deterministic automata are still loose. In the future, a more thorough analysis, perhaps leveraging recent work on Safra-tree based determinization [Sch09b], could likely improve the upper bound on the size of our construction. Although the current use of profiles for determinization does not admit the same relation with odd rankings that we employed for complementation, further work may yet permit the use of odd rankings for determinization. More practically, despite their combinatorial complexity, complementation based on Safra trees has proven very for automata with a small number of states [TFVT11]. It remains to be seen if the approach taken here will provide benefits in practice.

One limiting factor for research in this area is the paucity of real-world examples. Most existing systems use logical, instead of Büchi, specifications. These specifications are negated and then translated into Büchi automata, avoiding the necessity of complementation. Even systems that allow automata as specifications do not allow nondeterministic Büchi automata, requiring either pre-complemented automata or deterministic automata. Thus the generation of Büchi specifications remains an outstanding problem, both for practical use and to provide a better experimental base. Such specifications could be written by hand or automatically extracted. In Size-Change

Termination, proofs of termination are derived from programs in the form of Büchi automata [LJBA01]. This technique has been extended from first-order functional programs to imperative programs [ZGSV11] and even the untyped lambda calculus [JB04]. One could also imagine using Büchi specifications to prove variants of a system equivalent, for instance sequential and concurrent interpretations of the same program, or a model and it's refinement. However, until we have tools that can handle Büchi specifications, we are stuck in a chicken-and-egg paradox, and Büchi specifications will not be generated.

# Bibliography

[ACC$^+$11] P. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi automata inclusion testing. In Joost-Pieter Katoen and Barbara Knig, editors, *CONCUR 2011 Concurrency Theory*, volume 6901 of *Lecture Notes in Computer Science*, pages 187–202. Springer Berlin / Heidelberg, 2011.

[Ake78] Sheldon B. Akers. Binary decision diagrams. *IEEE Trans. Computers*, 27(6):509–516, 1978.

[ATW05] C.S. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of Büchi automata. In *Proc. 10th Int. Conf. on the Implementation and Application of Automata*, volume 3845 of *Lecture Notes in Computer Science*, pages 262–272. Springer, 2005.

[ATW06] C.S. Althoff, W. Thomas, and N. Wallmaier. Observations on determinization of Büchi automata. *Theor. Comput. Sci.*, 363(2):224–233, 2006.

[BAL07] A.M. Ben-Amram and C.S. Lee. Program termination analysis in polynomial time. *ACM Trans. Program. Lang. Syst.*, 29:5:1–5:37, January 2007.

[BCM$^+$92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: $10^{20}$ states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BLO12] S. Breuers, C. Löding, and J. Olschewski. Improved Ramsey-based Büchi complementation. In *Proc. 15th Int. Conf. on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science. Springer, 2012.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.

[Cho74] Y. Choueka. Theories of automata on $\omega$-tapes: A simplified approach. *Journal of Computer and Systems Science*, 8:117–141, 1974.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42:857–907, 1995.

[DR07] L. Doyen and J.-F. Raskin. Improved algorithms for the automata-based approach to model-checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2007.

[DR09] L. Doyen and J.-F. Raskin. Antichains for the automata-based approach to model-checking. *Logical Methods in Computer Science*, 5(1), 2009.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 328–337, 1988.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional $\mu$-calculus. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 267–278, 1986.

[FKV06] E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. *Int. J. Found. Comput. Sci.*, 17(4):851–868, 2006.

[FKVW10] S. Fogarty, O. Kupferman, M.Y. Vardi, and T. Wilke. Unifying Büchi complementation constructions. 2010. Submitted.

[FV09] S. Fogarty and M.Y. Vardi. Büchi complementation and size-change termination. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2009.

[FV10] S. Fogarty and M.Y. Vardi. Efficient Büchi universality checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2010.

[GKSV03] S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *Proc. 12th Conf. on Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer, 2003.

[Har03] J. Harrison. Formal verification at intel. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on*, pages 45 – 54, june 2003.

[HHK96] R.H. Hardin, Z. Har'el, and R.P. Kurshan. COSPAN. In *Proc. 8th Int. Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 423–427. Springer, 1996.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[JB04] N.D Jones and Nina Bohr. Termination analysis of the untyped $\lambda$-calculus. In *Rewriting Techniques and Applications. Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2004.

[Kar90] R.M. Karp. The transitive closure of a random digraph. *Random Struct. Algorithms*, 1(1):73–94, 1990.

[KC09] H. Karmarkar and S. Chakraborty. On minimal odd rankings for Büochi complementation. In *7th Int. Symp. on Automated Technology for Verification and Analysis*, volume 5799 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2009.

[KC11] H. Karmarkar and S. Chakraborty. Determinization of $\omega$-automata unified. *CoRR*, 2011.

[Kla90] N. Klarlund. *Progress Measures and finite arguments for infinite computations*. PhD thesis, Cornell University, 1990.

[Kur87] R.P. Kurshan. Complementing deterministic Büchi automata in polynomial time. *Journal of Computer and Systems Science*, 35:59–71, 1987.

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

[KV01] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(2):408–429, 2001.

[KV05] O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

[KVW01] O. Kupferman, M.Y. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164:322–344, 2001.

[KW08] D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *Proc. 35th Int. Colloq. on Automata, Languages, and Programming*, volume 525 of *Lecture Notes in Computer Science*, pages 724–735. Springer, 2008.

[Lan69] L.H. Landweber. Decision problems for $\omega$–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

[Lan11] M. Lange. Size-change termination and satisfiability for linear-time temporal logics. In *Proceedings of the 8th international conference on Frontiers of combining systems*, FroCoS'11, pages 28–39, Berlin, Heidelberg, 2011. Springer-Verlag.

[LBNA$^+$03] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, , and Y. Shoham. A portfolio approach to algorithm selection. In *Proc. 18th Int. Joint Conf. on Artificial Intelligence*, pages 1542–1543. Morgan Kaufmann, 2003.

[LJBA01] C.S. Lee, N.D. Jones, and A.M. Ben-Amram. The size-change principle for program termination. In *Proc. 28th ACM Symp. on Principles of Programming Languages*, pages 81–92, 2001.

[LW09] W. Liu and J. Wang. A tighter analysis of Piterman's Büchi determinization. *Inf. Process. Lett.*, 109(16):941–945, 2009.

[McN66] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on $\omega$-words. *Theoretical Computer Science*, 32:321–330, 1984.

[Mic88] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

[MS95] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[NLBD$^+$04] E. Nudelman, K. Leyton-Brown, A. Devkar, Y. Shoham, and H. Hoos. Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proc. 10th Int. Conf. on Principles and Practice of Constraint Programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 438–452. Springer, 2004.

[Pit06] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

[RS59] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[RvH93] J.M. Rushby and F. von Henke. Formal verification of algorithms for critical systems. *Software Engineering, IEEE Transactions on*, 19(1):13 –23, jan 1993.

[Saf88] S. Safra. On the complexity of $\omega$-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.

[Saf92] S. Safra. Exponential determinization for $\omega$-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symp. on Theory of Computing*, 1992.

[Sch09a] S. Schewe. Büchi complementation made tight. In *Proc. 26th Symp. on Theoretical Aspects of Computer Science*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[Sch09b] S. Schewe. Tighter bounds for the determinisation of büchi automata. In *Proc. 12th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2009.

[ST03] R. Sebastiani and S. Tonetta. “more deterministic” vs. “smaller” büchi automata for efficient LTL model checking. In *Proc. 12th Conf. on Cor-*

*rect Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2003.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[TFVT11] M.-H. Tsai, S. Fogarty, M.Y. Vardi, and Y.-K. Tsay. State of Büchi complementation. In *Proceedings of the 15th International Conference on Implementation and Application of Automata*, CIAA'10, pages 261–271, Berlin, Heidelberg, 2011. Springer-Verlag.

[THB95] S. Tasiran, R. Hojati, and R.K. Brayton. Language containment using non-deterministic omega-automata. In *Proc. 8th Conf. on Correct Hardware Design and Verification Methods*, volume 987 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 1995.

[TV05] D. Tabakov and M.Y. Vardi. Experimental evaluation of classical automata constructions. In *Proc. 12th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning*, volume 3835 of *Lecture Notes in Computer Science*, pages 396–411. Springer, 2005.

[TV07] D. Tabakov and M.Y. Vardi. Model checking Büechi specifications. In *1st International Conference on Language and Automata Theory and Application Principles of Programming Languages*, 2007.

[Var80] M.Y. Vardi. Expected properties of set partitions. Research report, The Weizmann Institute of Science, 1980.

[Var85] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th IEEE Symp. on Foundations of Computer Science*, pages 327–338, 1985.

[Var07a] M. Y. Vardi. The büchi complementation saga. In *Proc. 24th Sympo. on Theoretical Aspects of Computer Science*, volume 4393 of *Lecture Notes in Computer Science*, pages 12–22. Springer, 2007.

[Var07b] M.Y. Vardi. Automata-theoretic model checking revisited. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 137–150. Springer, 2007.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

[WDHR06] M.D. Wulf, L. Doyen, T. A. Henzinger, and J.-F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proc 18th Int. Conf. on Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 17–30, 2006.

[Yan06] Q. Yan. Lower bounds for complementation of $\omega$-automata via the full automata technique. In *Proc. 33rd Int. Colloq. on Automata, Languages, and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2006.

[ZGSV11] F. Zuleger, S. Gulwani, M. Sinn, and H. Veith. Bound analysis of imperative programs with the size-change abstraction. In *Proceedings of the*

*18th international conference on Static analysis*, SAS'11, pages 280–297, Berlin, Heidelberg, 2011. Springer-Verlag.

# Appendix A

# Symbolic Subsumption

For the explicit approach, subsumption has proven to be very effective. It remains an open question as to whether subsumption would also benefit the symbolic approach. Recall that automata-theoretic algorithms, such as Algorithms 4.2 and 4.4, often construct and manipulate sets of states. In the explicit approach, these sets are stored as a collection of the individual states. With subsumption, only the minimal elements under the subsumption relation are stored. Since these sets can be exponentially sized, in the 90's attention turned towards alternatives to the explicit approach to representing sets. When humans represent sets, we avoid explicitly listing all the elements. Instead, we use symbolic representations like set comprehensions, which use a characteristic function to define a set. In [BCM$^+$92], Burch et al. defined a similar method of using a characteristic function to symbolically represent a set. The function is represented using ordered binary decision diagrams (BDDs) [Ake78]. While in the worst case BDDs are as large as explicit representations, in many cases they are exponentially smaller. The size of a BDD is not necessarily related to the number of elements in the represented set: intuitively, it is related to the number of bits that must be checked to determine if an element is in the set. Thus it is not at all obvious that removing subsumed elements from a set results in a BDD with smaller size. None the less, in many cases the size of a BDD *is* related to the number of elements in the represented set, and it is worth investigating subsumption's effect on symbolic algorithms.

Symbolic algorithms have never found much traction in the realm of $\omega$-automata.

However, for automata over finite words, symbolic algorithms have proven very effective [TV05]. In these experiments, we check the universality of an automaton over finite words. This chapter investigates the effectiveness of subsumption on universality-checking algorithms for classical automata over finite words. We discover that, in initial experiments, subsumption did not improve the running time of symbolic approaches, and that in fact improvements in BDD-based packages have rendered them faster than explicit approaches.

## A.1 Symbolic Subsumption for NFAs

A *nondeterministic finite automaton* (NFA) is a tuple $\mathcal{B} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$, where $\Sigma$ is a finite nonempty alphabet, $Q$ a finite nonempty set of states, $Q^{in} \subseteq Q$ a set of initial states, $F \subseteq Q$ a set of accepting states, and $\rho : Q \times \Sigma \to 2^Q$ a nondeterministic transition function. In a *deterministic finite automaton* (DFA), the transition function is deterministic: $\rho : Q \times \Sigma \to Q$.

Much like automata on infinite words, a *run* of an NFA $\mathcal{B}$ on a word $w \in \Sigma^\star$ is a sequence of states $q_0 ... q_n \in Q^*$ such that $q_0 \in Q^{in}$ and, for every $i,\ 0 \leq i \leq (n-1)$, we have $q_{i+1} \in \rho(q_i, w_i)$. A run is *accepting* iff $q_n \in F$. A word $w \in \Sigma^\star$ is accepted by $\mathcal{B}$ if there is an accepting run of $\mathcal{B}$ on $w$. The words accepted by $\mathcal{B}$ form the *language* of $\mathcal{B}$, denoted by $L(\mathcal{B})$. We generalize the notion of a run to a path: we say there is a path from a sate $q$ to a state $r$ if there is a sequence of states $q_0 ... q_n \in Q^*$ such that $q_0 = q$, $q_n = r$, and for every $i,\ 0 \leq i \leq (n-1)$ there is a $w_i \in \Sigma$ so that $q_{i+1} \in \rho(q_i, w_i)$. Say that a state $r$ in an automaton is *reachable* where there is a state $q \in Q^{in}$ and a path from $q$ to $r$. We know that the language of an automaton on finite words is non-empty iff there is a reachable accepting state: there are states $q \in Q^{in},\ r \in F$ such that there is a path from $q$ to $r$.

To determine the universality of an NFA, we must first complement it. We use the Rabin-Scott subset construction [RS59]. Given an NFA $\mathcal{B} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$, define the subset complement of $\mathcal{B}$ to be the DFA $\bar{\mathcal{B}} = \langle \Sigma, Q_s, Q_s^{in}, \rho_s, F_s \rangle$ where $Q_s = 2^Q$ is the set of subsets of $Q$, $Q_s^{in}$ is the singleton set $\{Q^{in}\}$, $\rho_s(R, \sigma) = \bigcup_{r \in R} \rho(r, \sigma)$, and $F = \{R \in Q_s \mid F \cap R = \emptyset\}$.

**Lemma A.1.** [RS59] $L(\bar{\mathcal{B}}) = \overline{L(\mathcal{B})}$

Given an automaton $\mathcal{B}$, define $Q_s^r$ to be the set of all reachable states in $\bar{\mathcal{B}}$.

**Lemma A.2.** *$\mathcal{B}$ is universal iff $Q_s^r$ does* not *include an element $R$ such that $F \cap R = \emptyset$.*

*Proof.* If such a set exists, then there is a path from the start state of $\bar{\mathcal{B}}$ to this set. Thus $L(\bar{\mathcal{B}})$ is non-empty, and $L(\mathcal{B})$ is non-universal. If no such set exists, then there is no reachable accepting state in $\bar{\mathcal{B}}$, $L(\bar{\mathcal{B}})$ is empty, and $L(\mathcal{B})$ is universal. □

For the subset construction, the subset relation is a subsumption relation [WDHR06]. Thus we now discuss minimal elements in a set, under the subset relation. Given a set of sets $S$, say that an element $s \in S$ is *minimal in* $S$ if there does not exist an $s' \in S$ such that $s' \subsetneq s$. We now us define $Q_s^{\subseteq} = \{R \mid R \in Q_s^r, R \text{ is minimal in } Q_s^r\}$ to be the set of all minimal elements of $Q_s^r$. Say that a set $S$ is *subset-minimal* if every $s \in S$ is minimal in $S$.

**Lemma A.3.** *$\mathcal{B}$ is universal iff $Q_s^{\subseteq}$ does* not *include an element $R$ such that $F \cap R = \emptyset$.*

*Proof.* In one direction, assume $\mathcal{B}$ is not universal. Lemma A.2 implies there exists an $R \in Q_s^r$ such that $F \cap R = \emptyset$. This set can only fail to occur in $Q_s^{\subseteq}$ if there is some $R' \in Q_s^r$, $R' \subsetneq R$. However, if $F \cap R = \emptyset$, then surely $F \cap R' = \emptyset$, and in fact every subset of $R$ is disjoint from $F$. Since $\subsetneq$ is transitive, some such subset of $R$ is minimal in $Q_s^r$, and occurs in $Q_s^{\subseteq}$.

In the other direction, assume such an $R$ such that $F \cap R = \emptyset$ exists. Since $Q_s^{\subset} \subseteq Q_s^r$, Lemma A.2 implies $\mathcal{B}$ is not universal. □

We now present an algorithm that checks universality by computing the set $Q_s^{\subset}$ and checking for an element that is disjoint from $F$. For convenience, given a set $S \subseteq Q_s^r$ of states in the subset construction, define $\texttt{Post}(S) = \{R' \mid R \in S,\ R' \in Q_s^r,\ \sigma \in \Sigma,\ R' = \rho_s(R, \sigma)\}$ and $\texttt{Min}(S) = \{R \mid R \text{ is minimal in } S\}$

**Algorithm A.1:** `SubsumptiveUniversality`($\mathcal{B}$)

```
   Data: A NFA $\mathcal{B} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$
   Result: Whether $L(\mathcal{B}) = \Sigma^\star$.
1  Frontier $\Leftarrow \{Q^{in}\}$
2  Reachable $\Leftarrow \{Q^{in}\}$
3  while Frontier $\neq \emptyset$ do
4      AlreadyReachable $\Leftarrow$ Reachable
5      Image $\Leftarrow$ Post (Frontier)
6      Reachable $\Leftarrow$ Reachable $\cup$ Image
7      Reachable $\Leftarrow$ Min (Reachable)
8      Frontier $\Leftarrow$ Reachable $\setminus$ AlreadyReachable
9      if There exists $R \in$ Reachable such that $F \cap R = \emptyset$ then
10         return Not Universal
11 return Universal
```

**Lemma A.4.** *At every point,* Frontier $=$ `Min` *(*Frontier*).*

*Proof.* When initially assigned, Frontier is a singleton. On line 8, Frontier is assigned a subset of Reachable, which by line 7 contains only elements minimal in Reachable. A subset of a subset-minimal set is also subset-minimal. □

**Lemma A.5.** *If an element $R$ is ever added to* Reachable*, when Algorithm A.1 terminates there is an element $R' \in$* Reachable*, $R' \subseteq R$.*

*Proof.* Note that $\mathsf{Reachable}$ is changed only on lines 6 and 7. Line 6 only adds elements to $\mathsf{Reachable}$. By the definition of `Min`, an element $R$ can only be removed from $\mathsf{Reachable}$ on line 7 if another element $R'$, $R' \subsetneq R$, is added to $\mathsf{Reachable}$. $\square$

**Lemma A.6.** *When Algorithm A.1 terminates,* $\mathsf{Reachable} = Q_s^{\subset}$

*Proof.* Normally this would be proven by demonstrating that every $R \in \mathsf{Reachable}$ is also in $Q_S^{\subset}$, and the converse. Instead, we relax this claim and show that for every element $R \in \mathsf{Reachable}$ (resp. $Q_s^{\subset}$), there is an element $R' \in Q_s^{\subset}$ (resp. $\mathsf{Reachable}$) where $R' \subseteq R$. After this is proven in both directions, we demonstrate that for subset-minimal sets, this claim is sufficient to prove equality.

In one direction, assume a state $R \in \mathsf{Reachable}$. $R$ is reachable, as it was added to $\mathsf{Reachable}$ by some sequence of `Post` operations, and so $R \in Q_s^r$. Either there exists an $R' \in Q_s^{\subset}$ such that $R' \subsetneq R$, or there does not. If such an $R'$ exists, we have satisfied our claim. If such an $R'$ does not exist, then $R$ must be minimal in $Q_s^{\subset}$ and, by the definition of $Q_s^{\subset}$, it holds that $R \in Q_s^{\subset}$.

In the other direction, assume a state $R \in Q_s^{\subset}$. The fact that $R \in Q_s^{\subset}$ implies $R$ can be reached by some path $Q_0...Q_n \in Q^*$ where $Q_0 = \{Q^{in}\}$, $Q_n = R$ and, for every $i,\ 0 \leq i \leq n-1$ there exists a $\sigma_i$ such that $Q_{i+1} = \rho_s(Q_i, \sigma_i)$. We demonstrate by induction that for every $i,\ 0 \leq i \leq n$ there must exist a $Q_i' \in \mathsf{Reachable}$, $Q_i' \subseteq Q_i$. As $Q_n = R$, this implies there exists an $R' \in \mathsf{Reachable}$, $R' \subseteq R$.

As a base case, $Q_0 = \{Q^{in}\}$. At the beginning of the algorithm, the only element in $\mathsf{Reachable}$ is $\{Q^{in}\}$. By Lemma A.5, at the end of the algorithm there is a $Q_0' \in \mathsf{Reachable}$, $Q_0' \subseteq Q_0$. As the inductive step, we assume there exists a $Q_i' \in \mathsf{Reachable}$, $Q_i' \subseteq Q_i$. By the definition of $\rho_s$, $Q_{i+1} = \bigcup_{r \in Q_i} \rho(r, \sigma_i)$. Define $Q_{next}$ to be $\rho_s(Q_i', \sigma_i) = \bigcup_{r \in Q_i'} \rho(r, \sigma_i)$. Note that $Q_{next} \subseteq Q_{i+1}$. In the iteration of the loop where $Q_i'$ was first added to $\mathsf{Reachable}$, line 8 includes $Q_i'$ in $\mathsf{Frontier}$. Thus in the following iteration, line 6 adds $Q_{next}$ to $\mathsf{Reachable}$. By Lemma A.5, at the end of the algorithm there exists a $Q'_{next} \in \mathsf{Reachable}$, $Q'_{next} \subseteq Q_{next} \subseteq Q_{i+1}$. Take $Q'_{i+1}$ to be this $Q'_{next}$.

We have now established that for every $R \in \mathsf{Reachable}$, there is an $R' \in Q_s^{\subseteq}$ such that $R' \subseteq R$, and the converse. We also note that both $\mathsf{Reachable}$ and $Q_s^{\subseteq}$ are subset-minimal. We can now prove equality. For every $R \in \mathsf{Reachable}$ (resp. $Q_s^{\subseteq}$), there is a $R' \in Q_s^{\subseteq}$ (resp. $\mathsf{Reachable}$) and an $R'' \in \mathsf{Reachable}$ (resp. $Q_s^{\subseteq}$), such that $R'' \subseteq R' \subseteq R$. However, as $\mathsf{Reachable}$ (resp. $Q_s^{\subseteq}$) contains only minimal elements, $R'' \subseteq R$ implies $R'' = R$, and thus $R' = R$. Therefore every element in $\mathsf{Reachable}$ (resp. $Q_s^{\subseteq}$) is also in $Q_s^{\subseteq}$ (resp. $\mathsf{Reachable}$), and the two sets are equal. $\square$

**Theorem A.7.** *$\mathcal{B}$ is universal iff* `SubsumptiveUniversality` *($\mathcal{B}$) returns Universal.*

*Proof.* Immediate from Lemma A.3, Lemma A.6, and lines 9 through 11 of Algorithm A.1. $\square$

## A.2 Implementation and Experimentation

To test the effectiveness of Algorithm A.1, we implemented the algorithm symbolically in NuSMV and ran some experiments. This required modifying the NuSMV algorithms for checking safety properties to use the `Min` function. Upon reading in the specification, we have NuSMV define and store a strict superset relation over states symbolically. The strict superset relation is be defined as Superset$(R, R') = (\bigwedge_{q\in Q} q \in R \rightarrow q \in R') \wedge (\bigvee_{q\in Q} q \notin R \wedge q \in R')$. If we then have a set $S$ of states of the subset construction, the image of $S$ under the Superset relation are all states in $S$ that are strict supersets of another state in $S$. Call this image of $S$ under the Superset relation $S_{img}$. The set

$\texttt{Min}(S)$ is then $S \setminus S_{img}$. We compared the default NuSMV on-the-fly invariant checking algorithm, our symbolic implementation of Algorithm A.1, and the semi-symbolic approach of Wulf et al. [WDHR06]. We note that the semi-symbolic approach, which does use subsumption, was built on NuSMV 2.3, while our implementations were built on NuSMV 2.5.

The automata are derived from the random model of Chapter 3. However, instead of viewing the resulting five-tuples as Büchi automata, we view them as automata over finite words. This model was used in [TV05] and [WDHR06]. We initially performed terrain experiments over automata of of size 200. Note that both emptiness checking and complementation is easier for NFA than for Büchi automata, and thus we can scale to larger universality problems. Figure A.1 displays the behavior of our three algorithms. Since the timeout is 3600 seconds, the picture is flattened for Algorithm A.1. The overall hardest configuration seems to be at transition density 2 and acceptance density 0.9. However, with more moderate acceptance densities, all approaches also have difficulty with transition density 2.25. While we cannot draw conclusions about scalability from terrain experiments, these experiments are at the least not promising for subsumption.

To compare the scalability of these three approaches, we chose two configurations. The most difficult point was with transition density 2.0 and acceptance density 0.9. Figure A.2(a) displays the running time for the default NuSMV algorithm and Algorithm A.1 at the first configuration. Unfortunately, data for the semi-symbolic approach is not available at this configuration. Because a huge number of problems timed out for Algorithm A.1, we measure the timeout percentage. At this configuration, it is clear that subsumption is a dramatically poor choice of optimization. Figure A.2(b) shows the scalability of all three approaches at an easier configuration with transition density 1.75 and acceptance density 0.3. Here we see that, again, subsumption does not improve

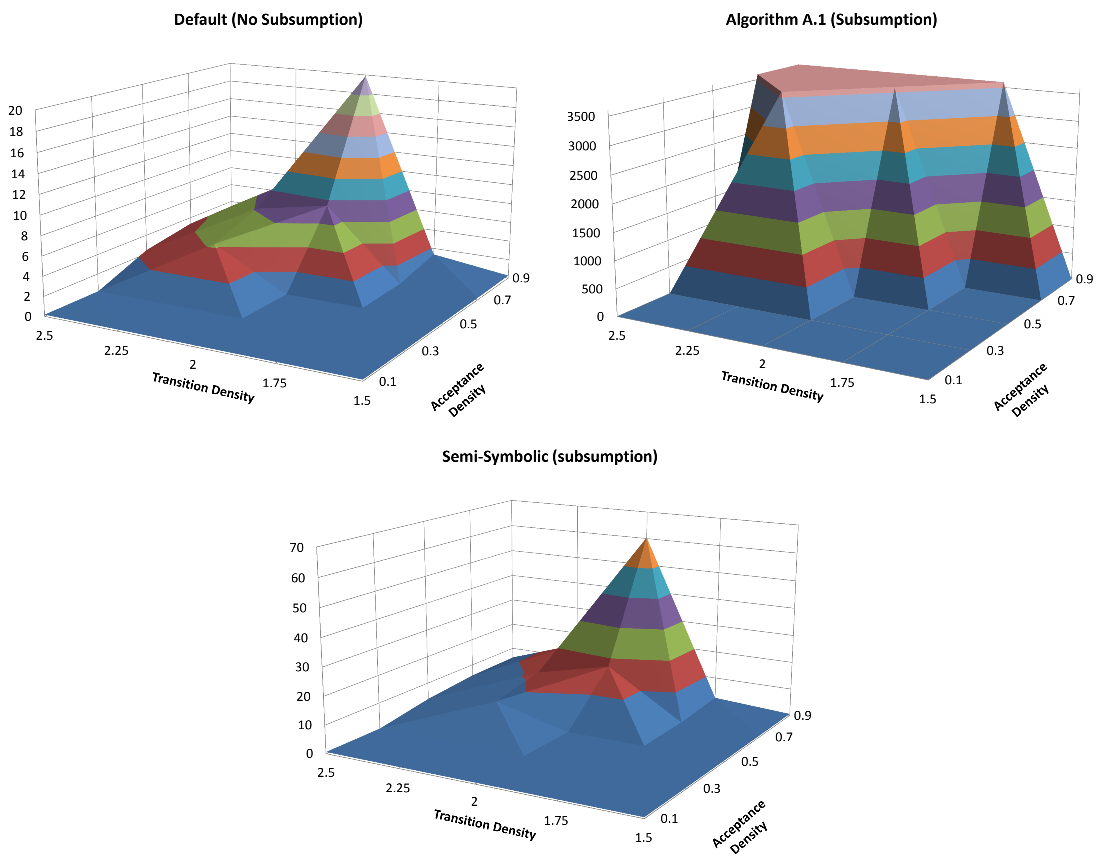
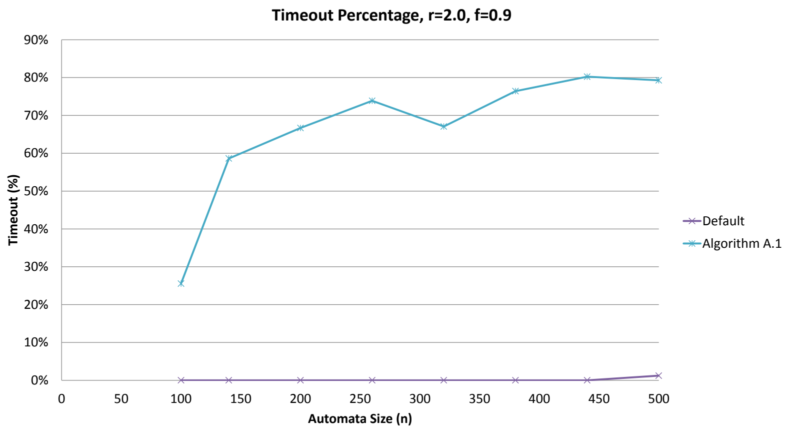
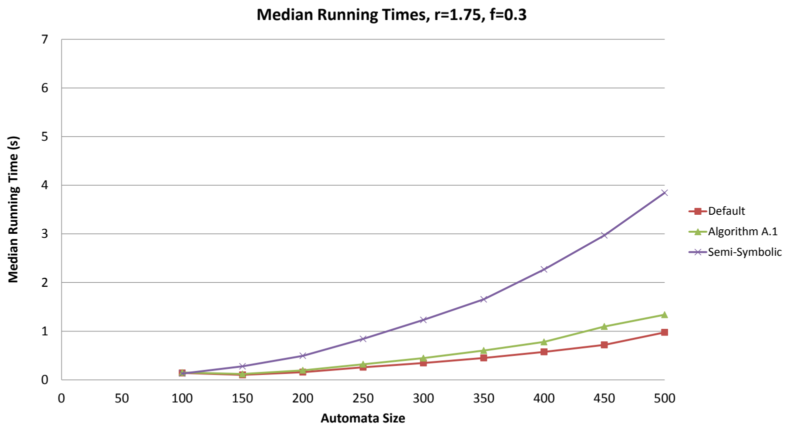


**Figure A.1 :** Differences in behavior between universality checking algorithms for non-deterministic finite automata, over a terrain of random automata with 200 states, measured as the median running time. Times are cut off at 3600 seconds.

the symbolic approach. Perhaps more surprisingly, the semi-symbolic approach scales much worse than either symbolic approach. This may be due to updates in the NuSMV code base between versions 2.3 and 2.5. However, conclusions gained from this data must be tempered by the fact that this configuration is not very difficult: median running times never topped eight seconds. Paired with the other scalability results, we can still conclude that subsumption does not improve scalability for symbolic algorithms for NFA universality.



**(a)** At the most difficult configuration, subsumption dramatically reduces the number of problems that can be solved in an hour



**(b)** At an easier configuration, subsumption seems to slightly increase the median running time, but conclusions are tempered by low running times.

**Figure A.2 :** Subsumption does not improve symbolic NFA universality testing.

# Appendix B

# Experimental Documentation

A key element of science is repeatability. Repeatability poses a particular challenge in computer science: our experiments rely on underlying architectures and frameworks that are constantly changing. This appendix documents all of the software tools employed. Copies of all software can be found at `http://www.cs.rice.edu/CS/Verification/`under the 'Theses' link.

## B.1 Framework for Experiments

As mentioned above, All experiments were performed on the Shared University Grid at Rice (SUG@R), a cluster of Sunfire x4150 nodes, each with two 2.83GHz Intel Xeon quad-core processors and 16GB of RAM. As of 6/13/2012, SUG@R nodes are running Red Hat Enterprise Linux Server release 5.6. Each experiment partitions the set of experimental cases, and hands each partition to a compute node in the cluster. The node then runs each tool on each case in turn, devoting the full resources of the compute node to the task for 3500 seconds. Timeouts were handled using the custom-written and very simple `timedrun` tool, with the invocation `timedrun -q -csv -t 3500`. The bash builtin `time` is used to calculate the elapsed wall time of each run. We use wall time to equalize memory-intensive and memory-sparing approaches, so that tools that thrash on disk or suffer many page faults are appropriately penalized. Since each node is dedicated to the job currently running on it, we are not worries about other tasks consuming CPU time that our experiment could be using. Of course, on a modern operating systems there are dozens of background processes running at all times, but we expect their impact to be vanishingly small. In addition to median running time and timeout status, we collect the output of each tool, to ensure consistency between results, as well as the CPU and memory usage profiles derived from the `top` command at 30-second intervals. These profiles give us some insight into the memory or CPU-bound nature of each tool and problem.

To generate automata in the Tabakov-Vardi model, which can be viewed either as Büchi automata (as in the bulk of this work) or as NFA (as in Appendix A), we use the tool `dk`, written by Deian Tabakov. To generate automata, edit the `create.sh` file in the `extra` directory. This file defines the starting, ending, and step size in the parameters size $n$, transition density $r$, and acceptance density $f$. When generating Büchi automata, the ”initTrans” option is selected, which ensures that the initial state has outgoing transitions on all characters. This avoids trivial cases of non-universality. For NFA, we also select the ”initAcpt” option, which ensures the initial state is accepting. This ensures the empty word is accepted, avoiding a trivial case of non-universality for NFA. The resulting automata are in the `.aut` file format, defined below.

## B.2 Complementation of Automata

To complement automata, we used the 2012-04-03 version of the *GOAL* tool. Documentation, downloads, and a description of the `gff` file format can be found at `http://goal.im.ntu.edu.tw`. We used the flags `-tr -cp -ro -macc` for rank-based complementation, and `-macc` for Ramsey-based complementation.

## B.3 Büchi Universality and Containment Checking

There are two tools we used to check the universality and containment in Chapter 4. The first is RANK, our modified version of the **mh** tool developed by Doyen and Raskin [DR09]. The **mh** tool can be found at `http://www.lsv.ens-cachan.fr/~doyen/antichains/antichains.html`. The RANK tool supports the one-way catch-them-young modification to the Emerson-Lei algorithms, but this modification was not used in our experiments, as it proved ineffective. Unfortunately, the RANK tool does not generate counterexamples. Input, in the `.aut` format, can either be given as a file name or piped through stdin. Finally, the maximum rank can be set artificially. If the rank is set too low, RANK may incorrectly report containment or universality.

The second tool, RAMSEY, is based on the **sct/scp** program– an optimized C implementation of the SCT algorithm from Ben-Amram and Lee [BAL07]. We have rewritten the RAMSEY tool to solve arbitrary Büchi universality problems by implementing Algorithm 4.4. By default, RAMSEY does output counterexamples. However, these counterexamples are very difficult to interpret. The RAMSEY tool accepts input in the `.svw` format.

## B.4 NFA Universality

In Appendix A, we modified NuSMV 2.5.0 to implement symbolic subsumption. The modified code is available at `http://www.cs.rice.edu/CS/Verification/`. To activate the symbolic subsumption, invoke NuSMV with the `-sub` flag. The automaton is fully complemented and symbolically represented in the subset construction before being given to NuSMV. For details of the encoding, see the section on file formats below. Symbolic subsumption may only be used with two kinds of specifications. The first are CTL specifications of the form AG: safety properties. It is necessary to use the `-AG` flag for these properties, or they will be checked using the quadratic CTL algorithm. With the `-AG` flag, NuSMV instead computes the reachable states and checks to ensure they satisfy the safety property. Note, however, that by computing the reachable states NuSMV is not operating on the fly, and thus is not fully implementing Algorithm A.1. To search on the fly, we must use the INVARSPEC specification format provided in NuSMV to enforce an invariant. This invokes a separate NuSMV function that searches for a counterexample as the set of reachable states grows. Subsumption has been implemented in both functions.

## B.5 File Formats

This section describes the file formats used for each tool, and presents the trivial automaton over an alphabet of two characters, consisting of one initial, accepting, state that transitions to itself on each character. For formats that only support containment, we look at the containment of this automaton in itself.

To translate between these formats, we use the `sctp` tool, which is written in Haskell. While this tool does perform Ramsey-based containment checking, the implementation is not written with an eye towards efficiency. However, it is very flexible in file formats, and has become a general-purpose conversion and sanity checking tool.

### B.5.1 aut

The `.aut` file format is used by `dk` and `rank`. To translate automata to the `.aut` format with `sctp`, use the `-oa` flag. This format uses newlines and spaces for delimiters and # to begin comment lines. Empty lines and comments are ignored, and do not matter for the count below. The first line is a sentinel: 2 for for a single automaton, suitable for universality checking; 3 for two automata representing a containment problem.

The second line delineates the number of letters in the alphabet: for the Tabakov-Vardi model, this is always 2. With $k$ characters, the characters are always $\{0 \ldots k-1\}$. The third line denotes the number of states in the first (and perhaps only) automaton. As with characters, with $n$ states the states are always $\{0 \ldots n-1\}$. The fourth line is a list of initial states, separated by spaces. After the fourth line begins a list of transitions. Each transition is of the form `state character state`, with the transition being from the first state to the second on the character. The list of transitions is ended with the single line containing `-`. The final line describing the first automaton is the list of accepting states, which is separated by spaces like the list of initial states. If the sentinel was 3, then a second automaton is described, identical to lines three and below from the first automaton. The two automata share the same alphabet size. An example of this format can be found in Figure B.1.

```
#Arc List Universality
2
#Language size
2
#Automaton size
1
#Initial states:
0
#Transitions
0 0 0
0 1 0
-
#Accepting states:
0
```

**Figure B.1 :** Example Automaton in the `aut` format

### B.5.2 svw

The `.svw` format directly represents supergraphs, hiding which characters are responsible for which supergraphs. To translate automata to the `.svw` format with `sctp`, use the `-oasn` flag. Note that the number of supergraphs for a containment problem $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is equal to the number of transitions in $\mathcal{A}$. However, the number of unique graphs is equal to the size of the alphabet. Since this format does not easily allow for memory-sharing between supergraphs, it is unsuitable for containment problems with large left-hand automata. The format does not allow comments. The svw format has three sections: the states of $\mathcal{A}$, the states of $\mathcal{B}$, and the supergraphs. Each section is enclosed in curly brackets and separated by a newline. Other than this requirement, newlines are ignored. The states of $\mathcal{A}$ are given as a space-separated list, in which each state is given as a space-separated triple enclosed in parens. The first word is the state name, the second is a bit indicating the state is initial, and the third is a bit indicating if the state is accepting. The states of $\mathcal{B}$ are likewise given as a space-separated list, where each state is a space-separated tuple enclosed in parens. The first word is the state name, and the second is a bit indicating if the state is initial. Since tests on supergraphs do not rely on checking if states in Bare accepting, this information is not encoded. Finally, the last section describes supergraphs. Each supergraph is contained in parens, and comprises of three parts: the source state from $\mathcal{A}$, the second state from $\mathcal{A}$, and the graph over states in $\mathcal{B}$. The graph is surrounded by curly braces, and consists of a list of space-separated arcs. Each arc is a space-separated triple enclosed in parents. The first word in the triple is the source state of $\mathcal{B}$. The second word the label of the arc, either 0 or 1. The third bit is the destination state of $\mathcal{B}$. Figure B.2 displays the encoding of our sample automaton, which results in a file with two supergraphs.

### B.5.3 gff

The gff file format, used in the `GOAL` tool The format uses XML to verbosely describe an automaton. We will not describe the format in detail here, simply provide an example

```
{
(astate 1 1)
}
{
(bstate 1)
}
{
(astate astate {(bstate 1 bstate) })
(astate astate {(bstate 1 bstate) })
}
```

**Figure B.2 :** Example Automaton in the `svw` format

in Figure B.3. A detailed description is available at `http://goal.im.ntu.edu.tw`. To translate automata to the `.gff` format with `sctp`, use the `-oagff` flag.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<structure label-on="transition" type="fa">
        <alphabet type="classical">
                <symbol> 0</symbol>
                <symbol> 1</symbol>
        </alphabet>
        <stateSet>
                <state sid="0"></state>
        </stateSet>
        <transitionSet>
                <transition tid="0">
                        <from>0</from>
                        <to>0</to>
                        <read>0</read>
                </transition>
                <transition tid="1">
                        <from>0</from>
                        <to>0</to>
                        <read>1</read>
                </transition>
        </transitionSet>
        <initialStateSet>
                <stateID>0</stateID>
        </initialStateSet>
        <acc type="buchi">
                <stateID>0</stateID>
        </acc>
</structure>
```

**Figure B.3 :** Example Automaton in the `gff` format

### B.5.4 NuSMV

`sctp` supports three different encodings of NFA as NuSMV files. In all cases the NuSMV file encodes the subset construction. The three formats differ in how the acceptance condition is encoded. Thus all three formats begin the same, as shown in

Figure B.4.

```
MODULE main
VAR
  state: array 0..0 of boolean;

IVAR
  input: boolean;

ASSIGN
  init(state[0]) := 1;

  next(state[0]) := (
  ( state[0] & (! input) )|
  ( state[0] & (input) )  );
```

**Figure B.4 :** The automaton encoding, without acceptance condition, in the NuSMV format.

The simplest encoding, using an AG spec, uses the `-osmv` flag in `sctp`. This encodes the acceptance condition as a CTL specification.

```
SPEC
  AG (state[0]);
```

In order to perform on-the-fly universality checking, we must encode the automaton with the `-ofsmv` flag in `sctp`. This encodes the acceptance condition as an invariant specification.

```
INVARSPEC
  (state[0]);
```

Finally, the semi-symbolic approach of [WDHR06] required the acceptance condition to be given as an invariant. Note that this does not line up with the normal NuSMV meaning of an invariant, and only works with the semi-symbolic approach of Wulf et al. To encode automata in this way, use the `-oismv` flag in `sctp`.

```
INVAR
  (state[0]);
```