BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

NEXT GENERATION OF RANK-BASED ALGORITHMS FOR OMEGA AUTOMATA

NOVÁ GENERACE RANK-BASED ALGORITMŮ PRO OMEGA AUTOMATY

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **BARBORA ŠMAHLÍKOVÁ**
AUTOR PRÁCE

SUPERVISOR **Ing. ONDŘEJ LENGÁL, Ph.D.**
VEDOUCÍ PRÁCE

CONSULTANT **Ing. VOJTĚCH HAVLENA, Ph.D.**
KONZULTANT PRÁCE

BRNO 2022

Department of Intelligent Systems (DITS) Academic year 2021/2022

Bachelor's Thesis Specification

24442

Student:	**Šmahlíková Barbora**
Programme:	Information Technology
Title:	**Next Generation of Rank-Based Algorithms for Omega Automata**
Category:	Theoretical Computer Science

Assignment:

1. Study the theory of Büchi automata and other omega automata. Furthermore, study rank-based algorithms for complementation and testing universality and inclusion of such automata.
2. Propose optimisations of the algorithms from the previous point.
3. Implement the proposed optimisations.
4. Experimentally compare your implementation with other tools using a suitable benchmark set.

Recommended literature:

- Orna Kupferman, Moshe Y. Vardi: Weak alternating automata are not that weak. ACM Trans. Comput. Log. 2(3): 408-429 (2001)
- Ehud Friedgut, Orna Kupferman, Moshe Y. Vardi: Büchi Complementation Made Tighter. Int. J. Found. Comput. Sci. 17(4): 851-868 (2006)
- Sven Schewe: Büchi Complementation Made Tight. STACS 2009: 661-672
- Yu-Fang Chen, Vojtech Havlena, Ondrej Lengál: Simulations in Rank-Based Büchi Automata Complementation. APLAS 2019: 447-467
- Vojtech Havlena, Ondrej Lengál: Reducing (to) the Ranks: Efficient Rank-based Büchi Automata Complementation. CONCUR 2021

Requirements for the first semester:

- The first item of the assignment.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Lengál Ondřej, Ing., Ph.D.**
Consultant:	Havlena Vojtěch, Ing., Ph.D., UITS FIT VUT
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	November 1, 2021
Submission deadline:	May 11, 2022
Approval date:	November 3, 2021

Abstract

Büchi automata (BA) complementation is a crucial operation for termination analysis of programs, model checking, or decision procedures for various logics. Despite its prominence, practically efficient algorithms for BA complementation are still missing. This thesis deals with optimizations of Büchi automata complementation, focusing mainly on rank-based techniques. The original rank-based algorithm is asymptotically optimal, but it can still generate unnecessarily large state space. For a practical usage, it is therefore desirable to reduce the number of generated states in the complement as much as possible. We propose several techniques that can efficiently complement some special types of Büchi automata, occuring often in practice, based on their structure. Some of these techniques can also, to a certain degree, be extended to general Büchi automata. The developed techniques were implemented as an extension of the tool RANKER for Büchi automata complementation and evaluated on thousands of hard automata. Our optimizations significantly reduce the generated state space and RANKER produces in the majority of cases a smaller complement than other state-of-the-art tools.

Abstrakt

Komplementace Büchiho automatů je klíčovou operací pro terminační analýzu programů, model checking nebo rozhodovací procedury pro různé logiky. Tato práce se zabývá především optimalizacemi rank-based komplementace Büchiho automatů. Původní rank-based algoritmus je sice asymptoticky optimální, přesto může generovat nezbytně velký stavový prostor. Pro praktické použití je tedy žádoucí maximálně redukovat počet vygenerovaných stavů v komplementu. V této práci představíme několik technik pro efektivní komplementaci některých speciálních typů Büchiho automatů, často se vyskytujících v praxi, které jsou založené na jejich struktuře. Některé z navržených technik lze do určité míry rozšířit i pro obecné Büchiho automaty. Techniky představené v této práci byly implementovány jako rozšíření nástroje RANKER pro komplementaci Büchiho automatů. Tyto optimalizace výrazně redukují generovaný stavový prostor a RANKER ve většině případů produkuje menší komplement než ostatní existující nástroje pro komplementaci.

Keywords

Büchi Automata, Büchi Complementation, Rank-Based Complementation, Elevator Automata

Klíčová slova

Büchiho automaty, Komplementace Büchiho automatů, Rank-based komplementace, Elevator automaty

Reference

ŠMAHLÍKOVÁ, Barbora. *Next Generation of Rank-Based Algorithms for Omega Automata.* Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ondřej Lengál, Ph.D. Consultant Ing. Vojtěch Havlena, Ph.D.

Rozšířený abstrakt

Büchiho automaty jsou automaty nad nekonečnými slovy, které definují třídu ω-regulárních jazyků. Jejich komplementace má řadu využití, například v terminační analýze programů, model checkingu, rozhodovacích procedurách pro různé logiky (S1S, ETL, QPTL, ...), jazykové inkluzi nebo testu ekvivalence. Jedná se o velmi náročnou operaci. Původní komplementační algoritmus, navržený Büchim v roce 1962, produkoval pro n-stavový automat komplement s až 2^{2^n} stavy. Proto od té doby vznikla celá řada přístupů, jak Büchiho automaty komplementovat, se snahou snížit prostorovou složitost této operace.

V první části práce se věnuji zejména optimalizacím tzv. *rank-based* komplementace. Pro tento přístup existuje algoritmus, který asymptoticky dosahuje dolní hranice složitosti $(0.76n)^n$ [32]. Tento algoritmus ale stále produkuje stavy a přechody, které nejsou v komplementu nezbytně nutné, proto je žádoucí hledat optimalizace, které tyto stavy nebudou generovat, a tím budou redukovat generovaný stavový prostor. Rank-based komplementace spočívá mimo jiné v tom, že každému stavu v aktuálně dosažitelné množině stavů přiřazuje nějaké číslo (tzv. *rank*). Obecně je maximální rank každého stavu pro n-stavový automat roven $2n - 1$. Počet následníků nějakého makrostavu v komplementu je potom přibližně roven faktoriálu z tohoto maximálního ranku. Pokud má ale automat vhodnou strukturu, není takto vysoký rank potřeba a velké množství stavů se generuje zbytečně. Proto je v této práci představen algoritmus, který umí efektivně snížit maximální rank stavům v jednotlivých silně souvislých komponentách u tzv. *elevator* automatů. Jedná se o automaty, které se velmi často vyskytují v praxi a které mají takové vlastnosti, které umožňují efektivně snížit omezení na maximální rank stavům v každé silně souvislé komponentě. Tento algoritmus výrazně redukuje generovaný stavový prostor a lze do jisté míry rozšířit i pro obecné Büchiho automaty. Pomocí techniky inspirované data flow analýzou jsme poté schopni propagovat jistá omezení na ranky napříč automatem a tím ranky jednotlivých stavů ještě více omezovat.

Rank-based komplementaci můžeme použít pro jakýkoliv obecný Büchiho automat, ale jisté speciální typy automatů lze komplementaovat efektivněji pomocí specializovaných procedur. Další část práce proto obsahuje optimalizace komplementačních algoritmů pro inherently weak a semi-deterministické automaty.

Všechny optimalizace prezentované v této práci byly implementovány jako rozšíření nástroje Ranker [14] v C++, který komplementuje Büchiho automaty. Byla provedena řada experimentů na několika tisících těžkých automatech (používaných v praxi i náhodně vygenerovaných). Díky optimalizacím na omezení maximálního ranku dosahujeme často i exponenciálně lepších výsledků nejenom oproti původnímu rank-based algoritmu, ale i proti předchozí verzi nástroje bez těchto optimalizací. Rank-based komplementace se sama o sobě může v některých případech jevit jako neefektivní, nicméně při zapojení různých optimalizací může konkurovat ostatním komplementačním přístupům a dokonce může být v řadě případů i efektivnější. V praxi se ale velmi často vyskytují automaty speciálních typů, které lze na základě jejich struktury komplementovat efektivněji, různými specializovanými procedurami. Ranker proto podporuje několik optimalizovaných komplementačních algoritmů a na základě typu a vlastností vstupního automatu vybere tu pravděpodobně nejefektivnější, nebo v některých případech vyzkouší procedur více a vrátí nejmenší automat. Kromě sledování efektu jednotlivých optimalizací pomocí srovnávání výsledků s předchozí verzí nástroje bylo také provedeno pečlivé srovnání s ostatními dostupnými nástroji pro komplementaci Büchiho automatů, které používají různé algoritmy. V experimentech jsme se soustředili zejména na výsledný počet stavů komplementu. Ranker produkuje ve většině případů menší automaty než ostatní dostupné nástroje.

Next Generation of Rank-Based Algorithms for Omega Automata

Declaration

Acknowledgements

I would like to thank my supervisor Ondra Lengál and my consultant Vojta Havlena for their help, support, and guidance throughout my work on this thesis and other projects. I am grateful that they are always creating such a friendly atmosphere during our meetings that I feel comfortable presenting all of my ideas. Special thanks go to my loved ones, not only for always supporting me in my work, decisions and crazy ideas, but also for showing me what is truly important in life.

Contents

Chapter 1

Introduction

Omega automata (ω-automata, automata on infinite words) were introduced in 1960s as an auxiliary tool for a decision procedure of a fragment of a second-order arithmeric [8]. This thesis focuses on complementing Büchi automata (BA), special instance of ω-automata, which is a crucial operation for decision procedures of various logics, such as the monadic second-order logic S1S [8] or temporal logics ETL and QPTL [33], as well as for language inclusion and equivalence testing. Besides the theoretical point of view, Büchi automata complementation became important also in practice, for example in model checking of temporal properties [37] or termination analysis of programs [11, 17, 9].

The purpose of model checking is to automatically check whether a system meets its specification. Both the system and the specified (temporal) property can be represented by a Büchi automaton. The problem of system verification is then transformed into the problem of language inclusion of these automata. More precisely, a system meets its specification if the language of its Büchi automaton is a subset of the language of the automaton encoding the property. Language inclusion check is performed by complementing the property automaton and checking if its intersection with the system automaton is empty.

The idea behind termination analysis of programs [11, 17, 9] is to construct a difference of two Büchi automata — one representing the program and one representing a set of paths with already proved termination. These paths can be safely removed from the program automaton. The removal is done using automata difference, which is implemented as an intersection of the program automaton and the complement of the automaton with terminating paths.

Due to the high complexity of Büchi complementation, different approaches and further optimizations have been introduced since the original construction by Büchi with the state complexity 2^{2^n} was presented in 1962. Apart from reducing the upper bound of the size of the complemented automaton, there was also an effort to find the theoretical lower bound, finally refined by Yan to $(0.76n)^n$ [38]. In this thesis, we focus on the *rank-based* complementation, which was introduced by Kupferman and Vardi [20], improved with the help of Friedgut [12], and further optimized by Schewe [32], whose construction produces the complement with the size matching the lower bound modulo a $\mathcal{O}(n^2)$ polynomial factor.

Even though Schewe's construction is asymptotically optimal, it may still generate a lot of unnecessary states and transitions. Optimization heuristics are therefore critical for good performance in practice. In *rank-based* complementation, every state from a set of states reachable over the current input is assigned a number (called its *rank*). The main problem responsible for the generated state space blow-up is the amount of nondeterminism, caused by a lot of possibilities how to assign ranks to a set of states. The number of possibilities depends combinatorially on the maximum rank that can be assigned. It is therefore desirable to reduce the maximum rank as much as possible.

In this thesis, we first identify *elevator automata*, a subclass of Büchi automata with a specific structure, and present an algorithm that assigns a bound for maximum rank for states in each strongly connected component. This algorithm can be extended to general BAs containing containing elevator automaton as a substructure. We show that elevator automata can be complemented in $\mathcal{O}(16^n)$ space. Secondly, we propose a technique, inspired by data flow analysis, that can propagate the rank bounds throughout the automaton and restrict the ranks even more. We also carry over the proposed techniques to general BAs.

Although the optimizations of rank-based procedure work for all BAs, automata with a more specific structure can, however, be complemented more efficiently, using specialized constructions for complementation. We therefore present optimizations for complementing inherently weak and semi-deterministic Büchi automata.

Optimizations presented throughout this thesis are implemented on top of the tool Ranker [14], which uses several complementation approaches based on properties of the input Büchi automaton. We evaluated our approach on thousands of hard automata (occuring in practice as well as randomly generated). Even though the original rank-based complementation algorithm may be quite inefficient, our optimizations can significantly reduce the generated state space and in a lot of cases can produce even exponentially better results. We show that Ranker produces a smaller complement in the majority of cases compared to the other state-of-the-art tools.

Chapter 2

Automata Theory

In this chapter, we introduce some definitions for ω-automata that are necessary for the following chapters. We define ω-automata in general, and then we focus on Büchi automata whose complementation is the main subject of this thesis. We also introduce some special types of Büchi automata that are characterized by a specific structure and for which more efficient algorithms for complementation can be used in comparison to general Büchi automata.

2.1 Languages

An *alphabet* is a nonempty, finite set of symbols, usually denoted by Σ. A *word* $\alpha = \alpha_0\alpha_1 \dots \alpha_n$ over alphabet Σ is a sequence of symbols from Σ. An empty word has length 0 and is denoted by ϵ. The *concatenation* of two words $\alpha = \alpha_0 \dots \alpha_n$ and $\beta = \beta_0 \dots \beta_m$ is the word $\alpha\beta = \alpha_0 \dots \alpha_n\beta_0 \dots \beta_m$. For a word α, we define $\alpha^0 = \epsilon$ and $\alpha^{k+1} = \alpha^k\alpha$.

The set of all words over an alphabet Σ is denoted by Σ^*. A *language* over Σ is a set of words $\mathcal{L} \subseteq \Sigma^*$. The *concatenation* of two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ is the language $\mathcal{L}_1 \cdot \mathcal{L}_2 = \{\alpha\beta \in \Sigma^* \mid \alpha \in \mathcal{L}_1 \text{ and } \beta \in \mathcal{L}_2\}$. The *iteration* of a language $\mathcal{L} \subseteq \Sigma^*$ is the language $\mathcal{L}^* = \bigcup_{i\geq 0} \mathcal{L}^i$, where $\mathcal{L}^0 = \{\epsilon\}$ and $\mathcal{L}^{i+1} = \mathcal{L}^i \cdot \mathcal{L}$ for every $i \geq 0$.

A *regular expression* e over alphabet Σ is defined by the following grammar

$$e ::= \emptyset \mid \epsilon \mid a \mid e_1 + e_2 \mid e_1e_2 \mid e^*$$

where $a \in \Sigma$ and e_1, e_2 are regular expressions. The *language* $\mathcal{L}(e)$ is defined inductively as (i) $\mathcal{L}(\emptyset) = \emptyset$, (ii) $\mathcal{L}(\epsilon) = \{\epsilon\}$, (iii) $\mathcal{L}(a) = \{a\}$, (iv) $\mathcal{L}(e_1 + e_2) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$, (v) $\mathcal{L}(e_1e_2) = \mathcal{L}(e_1) \cdot \mathcal{L}(e_2)$, and (vi) $\mathcal{L}(e^*) = (\mathcal{L}(e))^*$. A language $\mathcal{L}$ is regular iff there is a regular expression e such that $\mathcal{L} = \mathcal{L}(e)$.

2.2 ω-Languages

The symbol ω is used to denote the set of non-negative integers $\{0, 1, 2, 3, \dots\}$. An *ω-word* α over alphabet Σ is represented as a function $\alpha \colon \omega \to \Sigma$ where the i-th symbol is denoted as α_i. We abuse notation and sometimes represent α as an infinite sequence $\alpha = \alpha_0\alpha_1 \dots$ The *concatenation* of a finite word $\alpha = \alpha_0 \dots \alpha_n$ and an ω-word $\beta = \beta_0\beta_1 \dots$ is the ω-word $\alpha\beta = \alpha_0 \dots \alpha_n\beta_0\beta_1 \dots$. If it is clear from the context, we use simply word instead of ω-word.

We use Σ^ω to denote the set of all infinite words over Σ. An *ω-language* over an alphabet Σ is a set of ω-words $\mathcal{L} \subseteq \Sigma^\omega$. The *complement* of an ω-language $\mathcal{L}$ is the

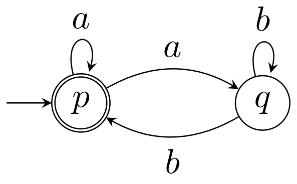


Figure 2.1: Büchi automaton $\mathcal{A}_{ex}$

ω-language $\Sigma^\omega \setminus \mathcal{L}$, often denoted by $\overline{\mathcal{L}}$. The *concatenation* of a language $\mathcal{L}_1$ and an ω-language $\mathcal{L}_2$ is the ω-language $\mathcal{L}_1 \cdot \mathcal{L}_2 = \{\alpha\beta \in \Sigma^\omega \mid \alpha \in \mathcal{L}_1 \text{ and } \beta \in \mathcal{L}_2\}$. The *$\omega$-iteration* of a language $\mathcal{L} \subseteq \Sigma^*$ is the ω-language $\mathcal{L}^\omega = \{\alpha_1\alpha_2 \ldots \mid \alpha_i \in \mathcal{L} \setminus \{\epsilon\} \text{ for every } i \geq 0\}$. (Note that the empty language $\emptyset$ can be defined as ϵ^ω.)

An *ω-regular expression* s over an alphabet Σ is defined by the following grammar

$$s ::= e^\omega \mid es \mid s_1 + s_2$$

where s_1, s_2 are ω-regular expressions and e is a regular expression. The *ω-language* $\mathcal{L}(s) \subseteq \Sigma^\omega$ of an ω-regular expression s is defined inductively as (i) $\mathcal{L}(e^\omega) = (\mathcal{L}(e))^\omega$, (ii) $\mathcal{L}(es) = \mathcal{L}(e) \cdot \mathcal{L}(s)$, and (iii) $\mathcal{L}(s_1 + s_2) = \mathcal{L}(s_1) \cup \mathcal{L}(s_2)$. A language $\mathcal{L}$ is ω-regular iff there is an ω-regular expression s such that $\mathcal{L} = \mathcal{L}(s)$.

2.3 Omega Automata

An *ω-automaton* is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, I, Acc)$, where Q is a finite set of states, Σ is an alphabet, δ is a transition function $\delta\colon Q \times \Sigma \to 2^Q$, $I \subseteq Q$ is a set of initial states, and Acc is an acceptance condition. Various types of ω-automata differ from each other in the definition of the acceptance condition Acc.

We sometimes treat δ as a set of transitions of the form $p \xrightarrow{a} q$, for instance, we use $p \xrightarrow{a} q \in \delta$ to denote that $q \in \delta(p, a)$. A *run* of $\mathcal{A}$ on a word α is an infinite sequence $\rho = q_0q_1q_2 \ldots$ such that $q_0 \in I$ and $q_{i+1} \in \delta(q_i, \alpha_i)$ for every $i \geq 0$. $\mathcal{A}$ is *complete* iff $|\delta(q, a)| \geq 1$ for every state $q \in Q$ and symbol $a \in \Sigma$.

$C \subseteq Q$ is a *strongly connected component* (SCC) of $\mathcal{A}$ if for any pair of states $q, q' \in C$ it holds that q is reachable from q' and q' is reachable from q. C is a *maximal strongly connected component* (MSCC) if it is not a proper subset of another SCC. The notation $\delta_{|S}$ for $S \subseteq Q$ is used to denote the restriction of the transition function $\delta \cap (S \times \Sigma \times S)$.

2.4 Büchi Automata

A (state-based) *Büchi automaton* (BA) is an ω-automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ where $F \subseteq Q$ is a set of accepting states.

Let $\inf(\rho)$ denote the set of states occuring infinitely often in the run ρ of $\mathcal{A}$ on a word α. The run ρ is called *accepting* iff $\inf(\rho) \cap F \neq \emptyset$. The word α is *accepted* by $\mathcal{A}$ if there exists an accepting run ρ of $\mathcal{A}$ on α. The set of all words accepted by $\mathcal{A}$ is called the *language* of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$.

An ω-language is a set of infinite words. According to Büchi's characterization theorem, languages that can be recognized by Büchi automata are ω-regular. Such languages can be defined by ω-regular expressions of Section 2.2.

Figure 2.1 shows an example of Büchi automaton $\mathcal{A}_{ex} = (Q, \Sigma, \delta, I, F)$ with $Q = \{p, q\}$, $\Sigma = \{a, b\}$, $I = \{p\}$, $F = \{p\}$, and $\delta = \{p \xrightarrow{a} p, p \xrightarrow{a} q, q \xrightarrow{b} q, q \xrightarrow{b} p\}$. The language of $\mathcal{A}_{ex}$ can be described using the ω-regular expression $(ab^*)^\omega$. Intuitively, it is the language of words with infinitely many occurences of the symbol a.

A *transition-based Büchi automaton* (TBA) is an ω-automaton $\mathcal{A}_\delta = (Q, \Sigma, \delta, I, \delta_F)$ where $\delta_F \subseteq \delta$ is a set of accepting transitions. Let $\inf_\delta(\rho)$ denote the set of transition occuring infinitely often in the run ρ of $\mathcal{A}_\delta$ on a word α. The run ρ is called *accepting* iff $\inf(\rho) \cap \delta_F \neq \emptyset$. The word α is *accepted* by $\mathcal{A}$ if there exists an accepting run ρ of $\mathcal{A}_\delta$ on α.

2.5 Special Types of Büchi Automata

In this section, we introduce various types of Büchi automata, characterized by a special structure, which can be complemented more efficiently than general Büchi automata.

A Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is

- *deterministic* if $|I| \leq 1$ and $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$,
- *semi-deterministic* if the automaton $(Q, \Sigma, \delta, \{q_F\}, F)$ is deterministic for each $q_F \in F$ (intuitively, the automaton behaves deterministically after traversing the first accepting state),
- *weak* if for every MSCC C of $\mathcal{A}$ it holds that either $C \cap F = \emptyset$ or $C \cap F = C$,
- *inherently weak* if for every MSCC C of $\mathcal{A}$ it holds that (i) $C \cap F = \emptyset$, or (ii) every cycle in C contains at least one accepting state $q_F \in F$, and
- *unambiguous* if there is at most one accepting run of $\mathcal{A}$ on any given word.

2.6 Simulations

Direct simulation on a Büchi automaton $\mathcal{A}$ is the relation $\preceq_{di} \subseteq Q \times Q$ defined as the largest relation s.t. $p \preceq_{di} q$ implies (i) $p \in F \Rightarrow q \in F$ and (ii) $p \xrightarrow{a} p' \in \delta \Rightarrow \exists q' \in Q \colon q \xrightarrow{a} q' \in \delta \wedge p' \preceq_{di} q'$ for each $a \in \Sigma$.

Fair simulation on a Büchi automaton $\mathcal{A}$ is the relation $\preceq_f \subseteq Q \times Q$ where $p \preceq_f q$ iff (i) for all runs ρ_p starting in p there is a run ρ_q starting in q over the same word, and (ii) if ρ_p is accepting, then ρ_q is accepting.

Chapter 3

Complementing Büchi Automata

The first complementation algorithm for Büchi automata was introduced by Büchi [8] in 1962. The construction showed that Büchi automata are closed under complementation. However, Büchi's approach leads to a doubly exponential blow-up. Various approaches and their further optimizations have been therefore presented since, with the aim of reducing the generated state space of BA complementation. In particular, we can distinguish several complementation approaches, briefly described below.

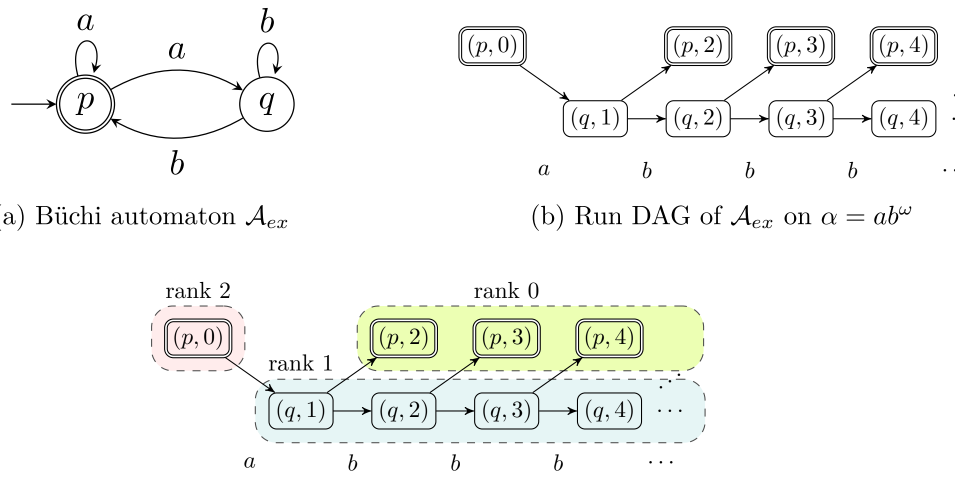
The complementation approach introduced by Büchi in [8] was *Ramsey-based* complementation with $2^{2^{\mathcal{O}(n)}}$ states in the complemented BA. The correctness of this method relies on a combinatorial result by Ramsey [29] to obtain a periodic decomposition of the possible behaviors of a BA on an infinite word. This construction was later improved by Sistla et al in [33] to produce BAs with $2^{\mathcal{O}(n^2)}$ states. The complexity was further reduced by Breuers et al in [7] to $2^{\mathcal{O}(n \log n)}$.

Determinization-based complementation was introduced by Safra [31], producing a complement with $2^{\mathcal{O}(n \log n)}$ states, and further improved by Piterman in [28] and Redziejowski in [30]. The principle of the determinization-based approach is to convert a (nondeterministic) Büchi automaton to an equivalent deterministic automaton with a different acceptance condition (e.g. Rabin automaton) that can be easily complemented. The result is then converted back into a BA.

Slice-based complementation was proposed by Kähler and Wilke in [18] with the complexity $2^{\mathcal{O}(n \log n)}$. The slice-based approach uses a reduced abstraction on a run tree to track the acceptance condition.

In this thesis, we focus on *rank-based* complementation, which was first introduced by Kupferman and Vardi [20] with the space complexity $2^{\mathcal{O}(n \log n)}$, then improved by Kupferman, Vardi, and Friedgut [12] to $\mathcal{O}((0.96n)^n)$ and made asymptotically optimal by Schewe in [32]. The space complexity of Schewe's construction matches the theoretical lower bound $\mathcal{O}((0.76n)^n)$ given by Yan [38] modulo a quadratic factor $\mathcal{O}(n^2)$. Optimizations of this construction were presented in [16].

In this chapter, we describe the principle of rank-based complementation algorithms. We start with the definition of run DAGs and explain how the ranking procedure works. Then we present three rank-based algorithms—we start with the original construction by Kupferman and Vardi [20], then we introduce the complementation with tight rankings by Friedgut, Kupferman, and Vardi [12], and finally, we describe Schewe's asymptotically optimal construction [32], which is the basis for our optimizations presented further in this thesis. Apart from rank-based complementation, we also present specialized complementa-



(a) Büchi automaton $\mathcal{A}_{ex}$

(b) Run DAG of $\mathcal{A}_{ex}$ on $\alpha = ab^\omega$

(c) Run DAG of $\mathcal{A}_{ex}$ on $\alpha = ab^\omega$ with assigned ranks

Figure 3.1: Example of a run DAG for BA $\mathcal{A}_{ex}$

tion constructions for inherently weak and semi-deterministic Büchi automata, which are usually more efficient than rank-based algorithms.

3.1 Run DAGs

In order to determine whether a given word should be accepted by the complement of a Büchi automaton, we have to examine all possible runs of the automaton on the given word. If none of these runs is accepting, the complement automaton should accept the word. Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be a BA and let α be a word. The set of all possible runs of $\mathcal{A}$ on α can be represented as a directed acyclic graph $\mathcal{G}_\alpha = (V, E)$, called the run DAG, with vertices V and edges E, where

- $V \subseteq Q \times \omega$ s.t. $(q, i) \in V$ iff there exists a run ρ of $\mathcal{A}$ on α with $\rho_i = q$, and
- $E \subseteq V \times V$ s.t. $((q, i), (q', i')) \in E$ iff $i' = i + 1$ and $q \xrightarrow{\alpha_i} q' \in \delta$.

A vertex $(q, i) \in V$ is called *accepting* if $q \in F$. A path in a run DAG is *accepting* if it visits infinitely often an accepting vertex. $\mathcal{A}$ accepts α iff there exists an accepting path in $\mathcal{G}_\alpha$. Hence, the complement should accept α only if there is no accepting path in $\mathcal{G}_\alpha$.

Consider the automaton from Figure 3.1a and a word $\alpha = ab^\omega$. A corresponding run DAG is shown in Figure 3.1b. Since there is no accepting run on α, the word should be accepted by the complement.

To determine if a given word should be accepted by the complement or not, we start by assigning a *rank* to each vertex in the corresponding run DAG. A vertex $v \in \mathcal{G}_\alpha$ is called *finite* if there are only finite number of vertices reachable from v, and *endangered* if there is no accepting vertex reachable from v.

The ranking procedure is performed as follows: let $\mathcal{G}_\alpha^0 = \mathcal{G}_\alpha$ and $j = 0$. The following steps are repeated until $j > 2|Q|$ or a fixpoint is reached.

1. Assign rank j to all finite vertices in $\mathcal{G}_\alpha^j$ and set $\mathcal{G}_\alpha^{j+1}$ to $\mathcal{G}_\alpha^j$ minus the vertices with rank j.

2. Assign rank $j+1$ to all endangered vertices in $\mathcal{G}_\alpha^{j+1}$ and set $\mathcal{G}_\alpha^{j+2}$ to $\mathcal{G}_\alpha^{j+1}$ minus the vertices with rank $j+1$.
3. Increase j by 2.

Vertices with no assigned rank have rank ω. It can be shown that if $\alpha \notin \mathcal{L}(\mathcal{A})$, the maximum assigned rank is at most $2|Q|$.

Theorem 3.1 ([20, Corollary 3.3])**.** If $\alpha \notin \mathcal{L}(\mathcal{A})$, then $\mathcal{G}_\alpha^{2|Q|+1}$ is empty.

Figure 3.1c shows how ranks are assigned to vertices from the run DAG in Figure 3.1b.

3.2 Basic Rank-Based Complementation

The ranking procedure of a run DAG described in Section 3.1 is used in the rank-based complementation algorithms in a way that the complemented automaton tracks all runs of the original automaton on the given word and all possible ranks of each of the runs. Every state of the complemented automaton is a macrostate containing, among other information, the set of all currently reachable states of the original automaton with rank assigned to each of the states. In this section, we present the rank-based algorithm originally proposed by Kupferman and Vardi [20].

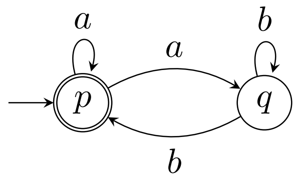
Let us first introduce some necessary definitions and notions. For a given Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, a *level ranking* is a function $f\colon Q \to \{0, 1, \dots, 2|Q|\}$ such that $\{f(q_F) \mid q_F \in F\} \subseteq \{0, 2, \dots, 2|Q|\}$, i.e., f maps all accepting states of $\mathcal{A}$ to even ranks. We use $\mathcal{R}$ to denote the set of all level rankings of $\mathcal{A}$ and $odd(f)$ to denote the set of states assigned an odd rank in a level ranking f. For a ranking f, the *rank* of f is defined as $rank(f) = \max\{f(q) \mid q \in Q\}$. The condition $f \leq f'$ holds iff for every state $q \in Q$ we have $f(q) \leq f'(q)$ and $f < f'$ iff $f \leq f'$ and there is a state $p \in Q$ with $f(p) < f'(p)$.

The procedure proposed by Kupferman and Vardi [20], denoted by KV, constructs the BA $\mathrm{KV}(\mathcal{A}) = (Q', \Sigma, \delta', I', F')$ whose components are defined as follows:

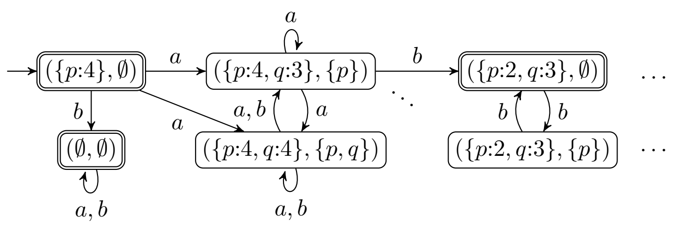
- $Q' = \{(S, O, f) \in 2^Q \times 2^Q \times \mathcal{R} \mid O \subseteq S\}$,
- $I' = \{I\} \times \{\emptyset\} \times \mathcal{R}$,
- $(S', O', f') \in \delta'((S, O, f), a)$ iff
  - $S' = \delta(S, a)$,
  - for every $q \in S$ and $q' \in \delta(q, a)$ it holds that $f'(q') \leq f(q)$, and
  - $O' = \begin{cases} \delta(S, a) \setminus odd(f') & \text{if } O = \emptyset \\ \delta(O, a) \setminus odd(f') & \text{otherwise, and} \end{cases}$
- $F' = 2^Q \times \{\emptyset\} \times \mathcal{R}$.

Theorem 3.2 ([20])**.** Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\mathrm{KV}(\mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

The macrostates of $\mathrm{KV}(\mathcal{A})$ consist of three components: S, O, and f. The S-component tracks all runs of $\mathcal{A}$, i.e., it contains all states reachable over the current input. The O-component tracks all runs whose rank has been even since the last cut-point (a point where $O = \emptyset$). The f component is a level ranking assigning rank to every state in S. A run of $\mathrm{KV}(\mathcal{A})$ is accepting iff it empties the O-component infinitely often, i.e., there is no run



(a) Büchi automaton $\mathcal{A}_{ex}$



$(\{p{:}4\}, \emptyset)$
$(\{p{:}4, q{:}3\}, \{p\})$
$(\{p{:}2, q{:}3\}, \emptyset)$
$(\emptyset, \emptyset)$
$(\{p{:}4, q{:}4\}, \{p, q\})$
$(\{p{:}2, q{:}3\}, \{p\})$
a
b
a, b

(b) KV($\mathcal{A}_{ex}$)

Figure 3.2: An example of a BA $\mathcal{A}_{ex}$ and its complement constructed using KV

where states have only even rank from some point, and, therefore, there is no accepting run of the original automaton on the input word.

Figure 3.2b shows a complement of a Büchi automaton $\mathcal{A}_{ex}$ given in Figure 3.2a. Only a part of the automaton is shown due to a quite large state space generated by this procedure (13 states). In the worst case, KV constructs a BA with approximately $(6n)^n$ states [20]. Note that for a more compact representation of a macrostate we often merge components S and f and in the first component we assign a rank only to the states present in the S-component. For example, we represent a macrostate $(\{p, q\}, \{p\}, \{p \mapsto 4, q \mapsto 3, r \mapsto 0\})$ as $(\{p{:}4, q{:}3\}, \{p\})$.

3.3 Complementation with Tight Rankings

The construction described in Section 3.2 was further improved by Friedgut, Kupferman and Vardi [12]. They observed that a special condition eventually holds for the ranks of the run DAG of a rejected word. The constructed automaton is composed of two parts: the *waiting* part, which tracks all runs of the original automaton (macrostates store all states reachable over the current input), and the *tight* part, which is similar to the KV construction, except that all level rankings are restricted to the so-called *tight* rankings.

Given a set of states $S \subseteq Q$, a (level) ranking $f\colon Q \to \{0, 1, \dots, 2|Q|\}$ is called *S-tight* if it has an odd rank r, $\{f(s) \mid s \in S\} \supseteq \{1, 3, \dots, r\}$, and $\{f(q) \mid q \notin S\} = \{0\}$. A ranking is *tight* if it is Q-tight.

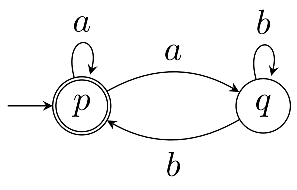
We use $\mathcal{T}$ to denote the set of all tight level rankings. Friedgut, Kupferman and Vardi observed that for every run DAG $\mathcal{G}_\alpha$ with a finite rank r, it holds that (i) r is odd and (ii) there exists a level $l \geq 0$ such that for all levels $l' \geq l$ and all odd ranks $o \in \{1, 3, \dots, r\}$, there is a vertex $(q, l') \in \mathcal{G}_\alpha$ with $rank((q, l')) = o$.

For $\ell \in \omega$, we define the ℓ-th *level* of $\mathcal{G}_\alpha$ as $level_\alpha(\ell) = \{q \mid (q, \ell) \in \mathcal{G}_\alpha\}$. Furthermore, we use f_ℓ^α to denote the ranking of level ℓ of $\mathcal{G}_\alpha$. Formally,

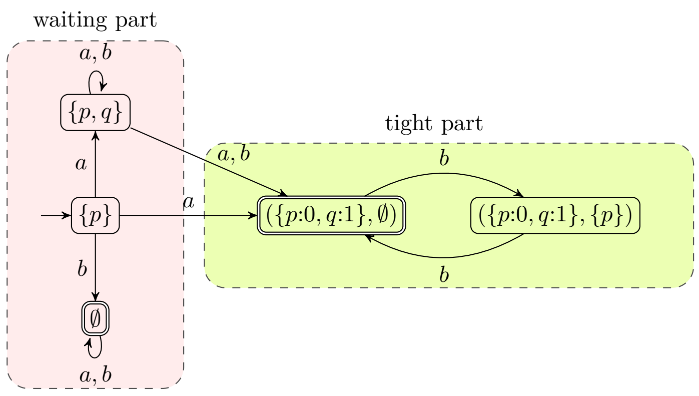
$$f_\ell^\alpha(q) = \begin{cases} rank_\alpha((q, \ell)) & \text{if } q \in level_\alpha(\ell), \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

We say that a level ℓ is *tight in $\mathcal{G}_\alpha$* if for all $k \geq \ell$ it holds that (i) f_k^α is tight, and (ii) $rank(f_k^\alpha) = rank(f_\ell^\alpha)$. Let $\rho = S_0 S_1 \dots S_{\ell-1}(S_\ell, O_\ell, f_\ell, i_\ell) \dots$ be a run on a word α in FKV($\mathcal{A}$). We say that ρ is a *super-tight run* [16] if $f_k = f_k^\alpha$ for each $k \geq \ell$. Finally, we say that a mapping $\mu\colon 2^Q \to \mathcal{R}$ is a *tight rank upper bound (TRUB) wrt α* iff

$$\exists \ell \in \omega\colon level_\alpha(\ell) \text{ is tight} \wedge (\forall k \geq \ell\colon \mu(level_\alpha(k)) \geq f_k^\alpha). \tag{3.2}$$



(a) Büchi automaton $\mathcal{A}_{ex}$



(b) FKV($\mathcal{A}_{ex}$)

Figure 3.3: An example of a BA $\mathcal{A}_{ex}$ and its complement constructed using FKV

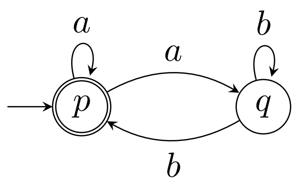
Intuitively, a TRUB is a ranking that gives an estimate on the necessary ranks of states in a super-tight run.

The procedure that makes use of tight rankings, denoted by FKV, constructs the BA FKV($\mathcal{A}$) $= (Q', \Sigma, \delta', I', F')$ whose components are defined as follows:

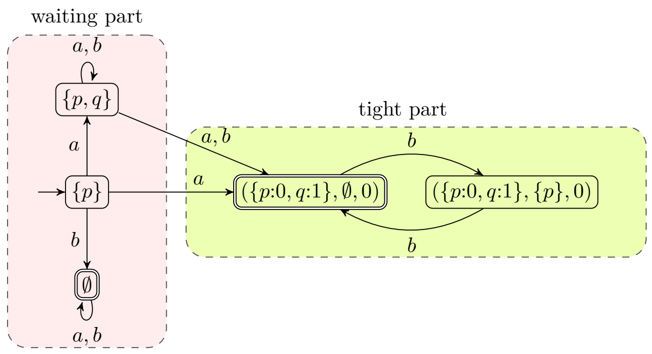
- $Q' = Q_1 \cup Q_2$ where
  - $Q_1 = 2^Q$ and
  - $Q_2 = \{(S, O, f) \in 2^Q \times 2^Q \times \mathcal{T} \mid f \text{ is } S\text{-tight}, O \subseteq S\}$,
- $I' = \{I\}$,
- $\delta' = \delta_1 \cup \delta_2 \cup \delta_3$ where
  - $\delta_1 \colon Q_1 \times \Sigma \to 2^{Q_1}$ such that $\delta_1(S, a) = \{\delta(S, a)\}$,
  - $\delta_2 \colon Q_1 \times \Sigma \to 2^{Q_2}$ such that $\delta_2(S, a) = \{(S', \emptyset, f) \in Q_2 \mid S' = \delta(S, a) \text{ and } f \text{ is } S\text{-tight}\}$,
  - $\delta_3 \colon Q_2 \times \Sigma \to 2^{Q_2}$ such that $(S', O', f') \in \delta_3((S, O, f), a)$ iff
    - $S' = \delta(S, a)$,
    - for every $q \in S$ and $q' \in \delta(q, a)$ it holds that $f'(q') \leq f(q)$,
    - $rank(f) = rank(f')$, and
    - $O' = \begin{cases} \delta(S, a) \setminus odd(f') & \text{if } O = \emptyset, \\ \delta(O, a) \setminus odd(f') & \text{otherwise, and} \end{cases}$
- $F' = \{\emptyset\} \cup ((2^Q \times \{\emptyset\} \times \mathcal{T}) \cap Q_2)$.

Theorem 3.3 ([12])**.** Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\text{FKV}(\mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

The *waiting* part is composed of the states in Q_1 and the states in Q_2 create the *tight* part. Intuitively, an accepting run on the complemented automaton stays in the waiting part until it holds that all successive level rankings are tight. Then it can move to the tight part where the word is accepted. See Figure 3.3b for FKV($\mathcal{A}_{ex}$) for the BA $\mathcal{A}_{ex}$ from Figure 3.3a. Note that FKV($\mathcal{A}_{ex}$) with 5 states is significantly smaller than KV($\mathcal{A}_{ex}$) with 13 states. In the worst case, FKV constructs a BA with $\mathcal{O}((0.96n)^n)$ states [12].



(a) Büchi automaton $\mathcal{A}_{ex}$



(b) SCHEWE($\mathcal{A}_{ex}$)

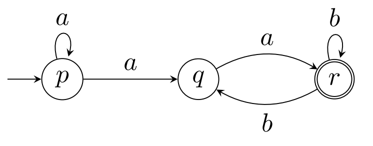
Figure 3.4: An example of a BA $\mathcal{A}_{ex}$ and its complement constructed using SCHEWE

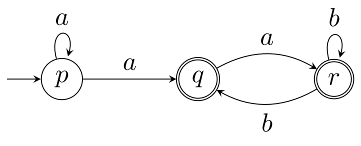
3.4 Optimal Rank-Based Complementation

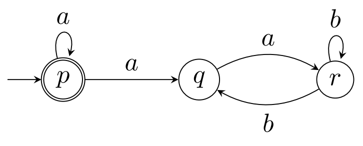
The improved construction by Friedgut, Kupferman, and Vardi [12], described in Section 3.3, was finally made asymptotically optimal by Schewe [32], using a more efficient cut-point construction. Instead of checking that no trace has an even rank since the last cut-point ($O = \emptyset$), this procedure, denoted by SCHEWE, cycles through all possible even ranks and checks that there is eventually no trace with this rank since the last cut-point. This leads to a significant reduction of the generated state space and the construction then matches the lower bound of $(0.76n)^n$ established by Yan [38] modulo a $\mathcal{O}(n^2)$ polynomial factor.

The procedure SCHEWE constructs the BA SCHEWE($\mathcal{A}$) $= (Q', \Sigma, \delta', I', F')$ whose components are defined as follows:

- $Q' = Q_1 \cup Q_2$ where
  - $Q_1 = 2^Q$ and
  - $Q_2 = \{(S, O, f, i) \in 2^Q \times 2^Q \times \mathcal{T} \times \{0, 2, \ldots, 2|Q| - 2\} \mid f$ is S-tight and $O \subseteq S \cap f^{-1}(i)\}$,
- $I' = \{I\}$,
- $\delta' = \delta_1 \cup \delta_2 \cup \delta_3$ where
  - $\delta_1 \colon Q_1 \times \Sigma \to 2^{Q_1}$ such that $\delta_1(S, a) = \{\delta(S, a)\}$,
  - $\delta_2 \colon Q_1 \times \Sigma \to 2^{Q_2}$ such that $\delta_2(S, a) = \{(S', \emptyset, f', 0) \mid S' = \delta(S, a), f$ is S-tight$\}$, and
  - $\delta_3 \colon Q_2 \times \Sigma \to 2^{Q_2}$ such that $(S', O', f', i') \in \delta_3((S, O, f, i), a)$ iff
    - $S' = \delta(S, a)$,
    - for every $q \in S$ and $q' \in \delta(q, a)$ it holds that $f'(q') \leq f(q)$,
    - $rank(f) = rank(f')$,
    - and
      - $i' = (i + 2) \bmod (rank(f') + 1)$ and $O' = f'^{-1}(i')$ if $O = \emptyset$ or
      - $i' = i$ and $O' = \delta(O, a) \cap f'^{-1}(i)$ if $O \neq \emptyset$, and
- $F' = \{\emptyset\} \cup ((2^Q \times \{\emptyset\} \times \mathcal{T} \times \omega) \cap Q_2)$.


(a) Inherently weak BA $\mathcal{A}$


(b) Equivalent weak BA $\mathcal{W}$


(c) Equivalent co-BA $\mathcal{C}$

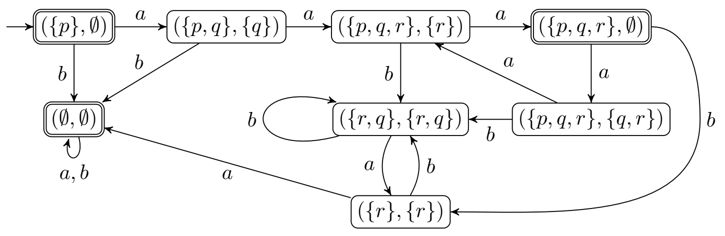

(d) MiHay($\mathcal{C}$)

Figure 3.5: An example of an inherently weak BA $\mathcal{A}$, an equivalent weak BA $\mathcal{W}$, an equivalent co-BA $\mathcal{C}$, and the complement MiHay($\mathcal{C}$)

Theorem 3.4 ([32, Corollary 3.3])**.** Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\textsc{Schewe}(\mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

The macrostates (S, O, f, i) in Schewe($\mathcal{A}$) are composed of four components. The S-component tracks all runs of $\mathcal{A}$ over the input word in the same way as the algorithms described in previous sections. The f-component assigns a rank to every state in S. The O-component tracks all runs having an even rank i since the last cut-point. After another cut-point is reached, the i component is increased by 2 modulo the maximal even rank. An accepting run therefore goes through all possible even ranks and checks that there is no infinite path having this particular even rank in the corresponding run DAG.

See Figure 3.4b for Schewe($\mathcal{A}_{ex}$) for the BA $\mathcal{A}_{ex}$ from Figure 3.4a. Note that in this particular example, Schewe($\mathcal{A}_{ex}$) has exactly the same structure as FKV($\mathcal{A}$) except that the i-component was added to macrostates in the tight part. This is because the original automaton has only 2 states and one of them was accepting. Hence, the rank is at most 1 and the i-component can be only 0. For automata with more states and a more involved structure, a significant decrease in generated state space can, however, be observed.

3.5 Inherently Weak Büchi Automata Complementation

Inherently weak Büchi automata can be easily transformed into weak BAs (without adding new states) by making all states in accepting SCCs accepting. In order to accept an input word in the complement, there must not exist a run with infinitely many accepting states. Since every run stays forever in some SCC and the automaton is weak, it contains either infinitely many accepting or infinitely many nonaccepting states. It cannot contain infinitely many accepting and nonaccepting states at the same time. It is therefore sufficient to check if every run contains infinitely many nonaccepting states. The idea behind the Miyano-Hayashi cut-point construction [26] is to periodically sample all runs and check if they contain a nonaccepting state. After all of them visit an accepting state, a cut-point is reached and new runs are sampled. The complement then accepts a word if there is a run where a cut-point is reached infinitely many times on this word.

Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be an inherently weak BA. We first construct an equivalent weak BA $\mathcal{W} = (Q, \Sigma, \delta, I, F_W)$, where F_W contains all states from MSCCs containing at least one accepting state of $\mathcal{A}$. We then convert $\mathcal{W}$ to an equivalent co-Büchi automaton $\mathcal{C} = (Q, \Sigma, \delta, I, F_C = Q \setminus F_W)$. A *co-Büchi automaton* (co-BA) $\mathcal{C} = (Q, \Sigma, \delta, I, F_C)$ accepts an input word α if there exists a run ρ such that for every state $q \in Q$ occuring infinitely often in ρ it holds that $q \in F_C$. The procedure, denoted by MiHay, constructs the (deterministic) BA $\text{MiHay}(\mathcal{C}) = (Q', \Sigma, \delta', I', F')$ whose components are defined as follows:

- $Q' = 2^Q \times 2^Q$,
- $I' = \{(I, I \setminus F_C)\}$,
- $\delta'((S, B), a) = (S', B')$ where
  - $S' = \delta(S, a)$,
  - and
    - $B' = S' \setminus F_C$ if $B = \emptyset$ or
    - $B' = (\delta(B, a) \cap S') \setminus F_C$ if $B \neq \emptyset$, and
- $F' = 2^Q \times \{\emptyset\}$.

Theorem 3.5 ([26])**.** Let $\mathcal{C} = (Q, \Sigma, \delta, I, F_C)$ be a co-BA. Then $\mathcal{L}(\text{MiHay}(\mathcal{C})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{C})$.

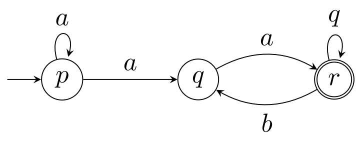
Figure 3.5 shows an example of an inherently weak Büchi automaton, equivalent weak and co-Büchi automata and the complement constructed using MiHay.

3.6 Semi-deterministic Büchi Automata Complementation

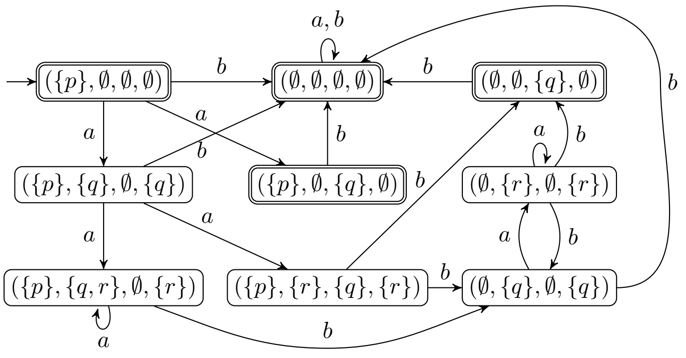
Semi-deterministic Büchi automata have a specific structure allowing to use a more efficient complementation construction. If a rank-based complementation is used, the maximum rank can be bounded by 3. More precisely, ranks of the states in the nondeterministic part can be bounded by 3, and states in the deterministic part by 2. Even though bounding the maximum rank can significantly reduce the generated state space in the *tight* part, the complemented automaton can have a lot of states because of the presence of the *waiting* part. Semi-deterministic BAs can be complemented using the NCSB construction [5], which does not consider the *waiting* part and keeps only rough information about the ranks in comparison to rank-based algorithms.

Let $\mathcal{A} = (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_t \cup \delta_2, I, F)$ be a semi-deterministic Büchi automaton such that Q_1 is a set of states in the nondeterministic part, Q_2 is a set of states in the deterministic part, $\delta_1 \colon Q_1 \times \Sigma \to 2^{Q_1}$, $\delta_t \colon Q_1 \times \Sigma \to 2^{Q_2}$, and $\delta_2 \colon Q_2 \times \Sigma \to 2^{Q_2}$. The procedure, denoted by NCSB, constructs the BA $\text{NCSB}(\mathcal{A}) = (Q', \Sigma, \delta', I', F')$ whose components are defined as follows:

- $Q' = \{(N, C, S, B) \in 2^{Q_1} \times 2^{Q_2} \times 2^{Q_2 \setminus F} \times 2^{Q_2} \mid B \subseteq C\}$,
- $I' = \{(Q_1 \cap I, Q_2 \cap I, \emptyset, Q_2 \cap I)\}$,
- $\delta' \colon Q' \times \Sigma \to 2^{Q'}$ such that $(N', C', S', B') \in \delta'((N, C, S, B), a)$ iff
  - $N' = \delta_1(N, a)$,
  - $C' \cup S' = \delta_t(N, a) \cup \delta_2(C \cup S, a)$,
  - $C' \cap S' = \emptyset$,



(a) Semi-deterministic BA $\mathcal{A}$



(b) NCSB($\mathcal{A}$)

Figure 3.6: An example of a semi-deterministic BA $\mathcal{A}$ and its complement NCSB($\mathcal{A}$)

- $S' \supseteq \delta_2(S, a)$,
- $C' \supseteq \delta_2(C \setminus F, a)$, and
- $B' = C'$ if $B = \emptyset$, otherwise $B' = \delta_2(B, a) \cap C'$, and

- $F' = \{(N, C, S, B) \in Q' \mid B = \emptyset\}$.

Theorem 3.6 ([5]). Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be a semi-deterministic BA. Then $\mathcal{L}(\text{NCSB}(\mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

An example of a semi-deterministic Büchi automaton with its complement is shown in Figure 3.6. Macrostates of NCSB($\mathcal{A}$) consist of four components: (N, C, S, B). The N-component tracks runs in the nondeterministic part of the automaton. The C-component represents the runs that have entered the deterministic part and are not *safe* (they did not visit an accepting state for the last time), whereas the S-component represents the *safe* runs. The last component is a breakpoint that is used to check that no run stays forever in component C.

Chapter 4

Next Generation of Rank-Based Algorithms for Büchi Automata

Even though Schewe's rank-based complementation construction described in Section 3.4 asymptotically matches the lower bound of $(0.76n)^n$ [32], it still produces a complemented automaton with potentially unnecessary states or transitions and, due to the high space complexity, further optimizations of this algorithm are crucial for practical applications. In this chapter, we first introduce *elevator automata* [15], a large class of Büchi automata with a specific structure, occuring often in practice. We analyze run DAGs of these automata and present an algorithm for reducing the bound on maximum rank of states in each strongly connected component. We also extend this algorithm for general Büchi automata. Moreover, we show that elevator automata can be complemented in $\mathcal{O}(16^n)$ space. Secondly, we present a technique based on data flow analysis that can be used to propagate rank restrictions throughout the automaton and thus reduce the ranks even more. In the next part of the thesis, we focus on optimizations for specialized complementation constructions for inherently weak and semi-deterministic automata. Thanks to their properties, we are able to use more efficient procedures for their complementation than the rank-based construction.

4.1 Elevator Büchi Automata

In this section, we introduce *elevator automata*, a class of Büchi automata with a specific structure. We analyze run DAGs for all types of strongly connected components of elevator automata and present an algorithm assigning bounds on maximum rank for states in each component. Finally, we show that elevator automata can be complemented in $\mathcal{O}(16^n)$ space.

Let C be an MSCC of a given Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ and $\mathcal{A}_{|\mathcal{C}} = (C, \Sigma, \delta_{|C}, I \cap C, F \cap C)$. We say that C is *deterministic* iff the BA $\mathcal{A}_{|\mathcal{C}}$ is deterministic, *non-accepting* iff $C \cap F = \emptyset$, *inherently weak accepting* iff every cycle in the transition diagram of $\mathcal{A}_{|C}$ contains an accepting state $q_F \in F$, and *trivial* iff $|C| = 1$ and $\delta_{|C} = \emptyset$.

A Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is an *elevator* (Büchi) automaton if for every MSCC C of $\mathcal{A}$ it holds that C is (i) deterministic (D), (ii) inherently weak accepting (IWA), or (iii) non-accepting (N). An example of an elevator automaton with assigned type to each strongly connected component is shown in Figure 4.1.

The number of successors of a given macrostate in rank-based complementation is given by the number of possible tight rankings, which rises combinatorially with the macrostate's

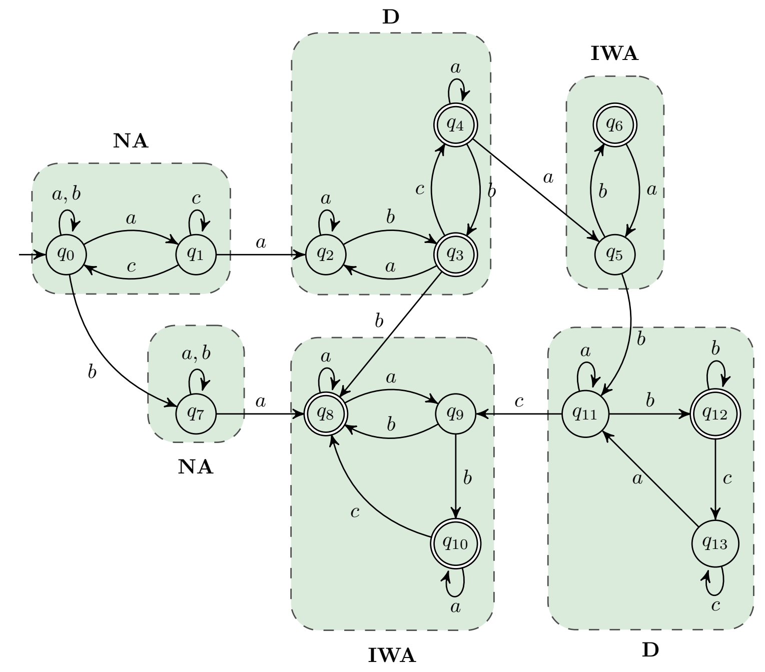
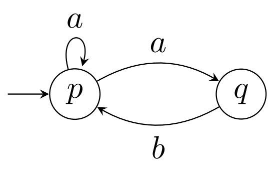


Figure 4.1: An example of an elevator automaton

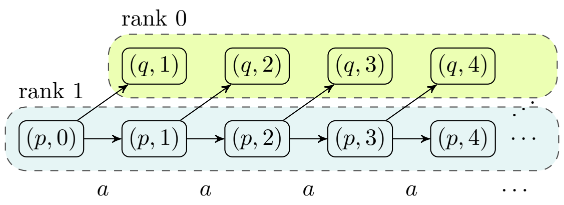
maximum rank. More precisely, for a given set of states, the number of possible tight rankings corresponds to the Stirling number of the second kind of the maximum rank [12]. For general BAs, the bound on maximum rank for all states of the automaton is $2|Q| - 1$. However, this bound is often unnecessarily high and many redundant states can be generated. Thanks to the specific structure of elevator automata, we can reduce the bound on maximum rank for states in every MSCC. Before we formally describe the rank restriction for elevator automata, let us give an intuition behind the maximum rank reduction by analyzing run DAGs of a BA containing only one MSCC of one of the three types that can be present in an elevator automaton.

4.1.1 Non-accepting Components

Let $\mathcal{A}$ be a BA with only one non-accepting MSCC and $\alpha \notin \mathcal{L}(\mathcal{A})$ be an input word. In $\mathcal{G}^1_\alpha$, constructed by the ranking procedure from Section 3.1, all finite vertices are removed. Since $\mathcal{A}$ contains no accepting state, all vertices in $\mathcal{G}^1_\alpha$ are endangered, and therefore the maximum rank can be bounded by 1. See Figure 4.2a for an example of a BA with one non-accepting

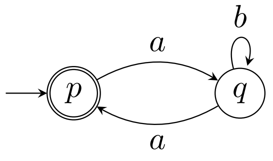


(a) Non-accepting BA

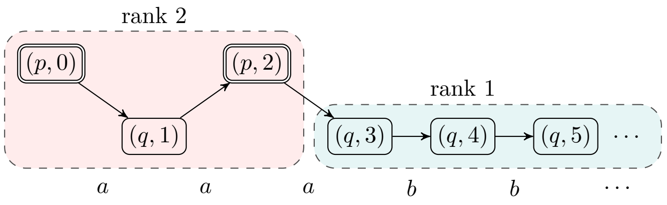


(b) Run DAG

Figure 4.2: The run DAG of a non-accepting MSCC over word a^ω



(a) Deterministic BA



(b) Run DAG

Figure 4.3: The run DAG of a deterministic MSCC over word $aaab^\omega$

MSCC. The corresponding run DAG $\mathcal{G}_\alpha$ with assigned ranks for a word $\alpha = a^\omega$ is shown in Figure 4.2b.

4.1.2 Deterministic Components

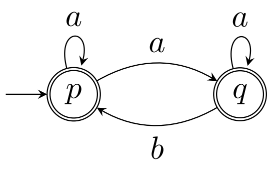
Let $\mathcal{A}$ be a deterministic BA with only one MSCC and $\alpha \notin \mathcal{L}(\mathcal{A})$ be an input word. Since the automaton is deterministic, there is at most one run of $\mathcal{A}$ on α. The corresponding run DAG $\mathcal{G}_\alpha$ therefore contains at most one vertex in each level, and because there are only finitely many accepting states in the run of $\mathcal{A}$ on α, there is a level l such that for all levels $l' \geq l$ it holds that all vertices in level l' are endangered. All vertices in level $l' \geq l$ have rank 1. Due to the determinism, $\mathcal{G}_\alpha^2$ is always finite. All vertices in levels smaller than l have therefore rank 2 and a greater rank is not needed. (Note that the maximal rank restriction also holds if the automaton has more than one initial state.) Consider the deterministic BA with one MSCC in Figure 4.3a and the word $\alpha = aaab^\omega$. The corresponding run DAG with assigned ranks is shown in Figure 4.3b.

4.1.3 Inherently Weak Accepting Components

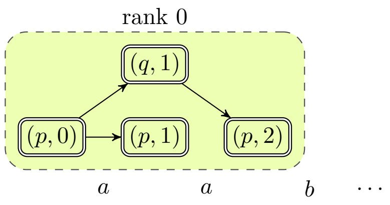
Let $\mathcal{A}$ be a BA with only one inherently weak accepting MSCC and $\alpha \notin \mathcal{L}(\mathcal{A})$ be an input word. Since every cycle of $\mathcal{A}$ contains an accepting state and $\alpha \notin \mathcal{L}(\mathcal{A})$, all possible runs of $\mathcal{A}$ on α must be finite. The whole run DAG $\mathcal{G}_\alpha$ is therefore finite and the maximal rank is 0. See Figure 4.4a for an example of a BA with only one inherently weak accepting MSCC. Figure 4.4b shows the corresponding run DAG $\mathcal{G}_\alpha$ with assigned ranks for $\alpha = ab^\omega$.

4.1.4 Rank Restriction for Elevator Automata

In this section, we present an algorithm that assigns each MSCC a label of the form TYPE:*rank* with the type of MSCC and the bound on the maximum rank of its states.



(a) Inherently weak accepting BA



(b) Run DAG

Figure 4.4: The run DAG of an inherently weak accepting MSCC over word aab^ω

The assignment is performed from terminal MSCCs (i.e., MSCCs that cannot reach to any other MSCC) towards MSCCs with initial states. More precisely, a label can be assigned to MSCC C only if (i) C is terminal or (ii) a label was already assigned to all MSCCs reachable from C. Note that there can be more options how to assign a type for some MSCCs. The algorithm assigns the type that is most suitable in terms of keeping the rank bound as low as possible in a greedy way (i.e., based on local information). This can be different for every MSCC, depending on the labels of its successors. For the following algorithm, we assume that an elevator automaton contains no useless states (there is therefore no terminal non-accepting MSCC).

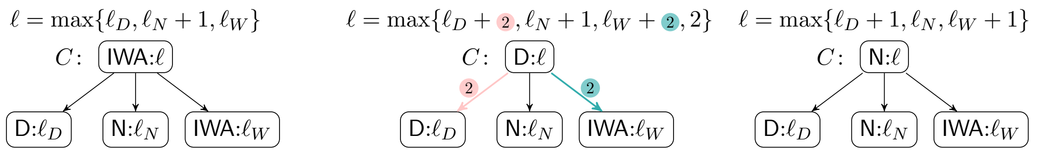
For a terminal MSCC C, we assign the following label:

1. IWA:0 if C is inherently weak accepting,
2. D:2 otherwise (i.e., if C is deterministic accepting).

Note that the previous two options are complete because the automaton contains no useless states. For non-terminal MSCCs, we use the corresponding rules from Figure 4.5. Children nodes denote already processed successive MSCCs. In particular, a child node of the form k:ℓ_k denotes an aggregate of all siblings of the type k with ℓ_k being the maximum rank of these siblings. For a non-terminal MSCC C, the rules for assigning a label are the following:

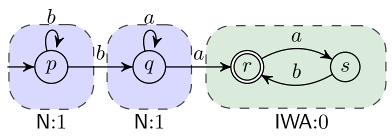
1. If C is trivial, we try both rules from Figure 4.5a and Figure 4.5c and use the one with the smaller rank.
2. Else if C is IWA, we use the rule in Figure 4.5a.
3. Else if C is deterministic accepting, we use the rule in Figure 4.5b.
4. Else if C is deterministic and non-accepting, we use one of the rules in Figure 4.5b and Figure 4.5c that gives us a smaller rank.
5. Else if C is nondeterministic and non-accepting, we use the rule in Figure 4.5c.

The maximum rank of each MSCC is then assigned to all its states and macrostates with higher ranks are not generated. We denote the procedure as ElevBound. Formally, the result of the algorithm is a mapping $\chi : Q \to \omega$ that gives a bound on the maximum rank to each state of the automaton. This mapping can be plugged in, e.g., Schewe to prune the generated state space. Figure 4.6a shows an elevator automaton $\mathcal{A}_{el}$ with assigned label for

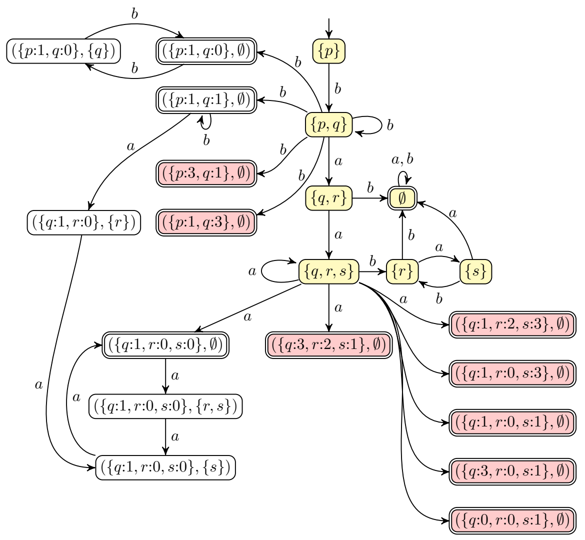

$\ell = \max\{\ell_D, \ell_N + 1, \ell_W\}$
C: IWA:ℓ
D:ℓ_D
N:ℓ_N
IWA:ℓ_W
$\ell = \max\{\ell_D + ②, \ell_N + 1, \ell_W + ②, 2\}$
C: D:ℓ
②
②
D:ℓ_D
N:ℓ_N
IWA:ℓ_W
$\ell = \max\{\ell_D + 1, \ell_N, \ell_W + 1\}$
C: N:ℓ
D:ℓ_D
N:ℓ_N
IWA:ℓ_W

(a) C is IWA (b) C is D (c) C is N

Figure 4.5: Rules for assigning types and rank bounds to MSCCs. The symbols ② and ② are interpeted as 0 if all the corresponding edges from the components having rank ℓ_D and ℓ_W, respectively, are deterministic; otherwise they are interpreted as 2. Transitions between two components C_1 and C_2 are deterministic if the BA $(C, \delta|_C, \emptyset, \emptyset)$ is deterministic for $C = \delta(C_1, \Sigma) \cap (C_1 \cup C_2)$.



(a) Elevator automaton $\mathcal{A}_{el}$



$(\{p{:}1, q{:}0\}, \{q\})$
$(\{p{:}1, q{:}0\}, \emptyset)$
$\{p\}$
$(\{p{:}1, q{:}1\}, \emptyset)$
$\{p, q\}$
$(\{p{:}3, q{:}1\}, \emptyset)$
$(\{p{:}1, q{:}3\}, \emptyset)$
$\{q, r\}$
$\emptyset$
$(\{q{:}1, r{:}0\}, \{r\})$
$\{q, r, s\}$
$\{r\}$
$\{s\}$
$(\{q{:}1, r{:}2, s{:}3\}, \emptyset)$
$(\{q{:}1, r{:}0, s{:}0\}, \emptyset)$
$(\{q{:}3, r{:}2, s{:}1\}, \emptyset)$
$(\{q{:}1, r{:}0, s{:}3\}, \emptyset)$
$(\{q{:}1, r{:}0, s{:}0\}, \{r, s\})$
$(\{q{:}1, r{:}0, s{:}1\}, \emptyset)$
$(\{q{:}1, r{:}0, s{:}0\}, \{s\})$
$(\{q{:}3, r{:}0, s{:}1\}, \emptyset)$
$(\{q{:}0, r{:}0, s{:}1\}, \emptyset)$

(b) Complement of $\mathcal{A}_{el}$. Our procedure will not generate the red states (and their successors, which are not shown in the figure).

Figure 4.6: An example of an elevator BA with its complement

each MSCC. The complement of $\mathcal{A}_{el}$ is in Figure 4.6b. Red macrostates are not generated because the value assigned to some state is higher than the rank bound on the maximum rank – for example a macrostate $(\{q{:}0, r{:}0, s{:}1\}, \emptyset)$ was not generated, because state s is assigned the rank 1, which is higher than the rank bound 0 given by our algorithm.

Lemma 4.1. Let $\mathcal{A}$ be an elevator automaton. ElevBound($\mathcal{A}$) is a TRUB.

Proof. Consider an elevator automaton $\mathcal{A}$. Let $\mathcal{G}_\alpha$ be a run DAG over some word $\alpha \notin \mathcal{L}(\mathcal{A})$ and C be an MSCC of $\mathcal{A}$. We proceed by induction on the structure of $\mathcal{A}$. We start with the base case.

Claim 1: *Let C be a terminal* IWA *component in $\mathcal{G}_\alpha$. Then, all vertices of $\mathcal{G}_\alpha$ labelled by C will have the rank 0.*

Proof: All cycles in inherently weak accepting components are accepting. Since $\alpha \notin \mathcal{L}(\mathcal{A})$, there is no run staying in C forever. All vertices labelled by a state from C are therefore finite in $\mathcal{G}_\alpha$ and are, therefore, assigned rank 0. ∎

Claim 2: *Let C be a terminal* D *component in $\mathcal{G}_\alpha$. Then, all vertices of $\mathcal{G}_\alpha$ labelled by C will have the rank at most 2.*

Proof: We prove that $\mathcal{G}_\alpha^2$ contains only finite vertices labelled by C. If it is not true, either $\alpha \in \mathcal{L}(\mathcal{A})$ or C is not terminal and deterministic. ∎

Now we prove the main lemma by induction on given rules. We prove that if a state q was assigned rank k, then $\mathcal{G}_\alpha^{k+1}$ does not contain a node labelled by q.

- Base case: If a terminal component C is IWA, from Claim 1 we get that all states from C will have the rank 0. If a terminal component is D, then from Claim 2 we have that all states from C will have the rank bounded by 2.

- Inductive case: Assume that for all states q from already processed components, if q were assigned rank m, $\mathcal{G}_\alpha^{m+1}$ does not contain node labelled by q. Note that inside each rule we can investigate cases D, N, and IWA separately since the adjacent components do not affect each other.

Figure 4.5a Observe that after $\ell = \max\{\ell_D, \ell_N + 1, \ell_W\}$ steps of the ranking procedure, in the worst case, all vertices labelled by C in $\mathcal{G}_\alpha^\ell$ are finite (otherwise it is a contradiction with induction hypothesis). Therefore, in $\mathcal{G}_\alpha^{\ell+1}$ there are no vertices labelled by C. The ranks of vertices labelled by C is hence $\max\{\ell_D, \ell_N + 1, \ell_W\}$.

Figure 4.5b We prove that in $\mathcal{G}_\alpha^\ell$ all vertices labelled by C are finite (ℓ is from the rule). From the induction hypothesis, after $\ell - 1$ steps (in the worst case) all vertices labelled by adjacent D, IWA components are finite in $\mathcal{G}_\alpha^\ell$ (provided that the transitions are deterministic). Vertices labelled by adjacent N components are not present in $\mathcal{G}_\alpha^\ell$. Therefore, if there is some vertex v labelled by C in $\mathcal{G}_\alpha^\ell$ that is not finite, the only possibility is that for each $v' \in \mathit{reach}_{\mathcal{G}_\alpha^\ell}(v)$ we reach in $\mathcal{G}_\alpha^\ell$ from v' some vertex labelled by a state from the D, IWA components. However, it is a contradiction with the transition determinism.

Figure 4.5c We prove that in $\mathcal{G}_\alpha^\ell$ all vertices labelled by C are endangered. This follows from the fact that after $\ell - 1$ steps no vertex labelled by the adjacent D, IWA components is present in $\mathcal{G}_\alpha^\ell$ (induction hypothesis). Therefore, all vertices labelled by C in $\mathcal{G}_\alpha^\ell$ are endangered. □

4.1.5 Refined Ranks for Non-Elevator Automata

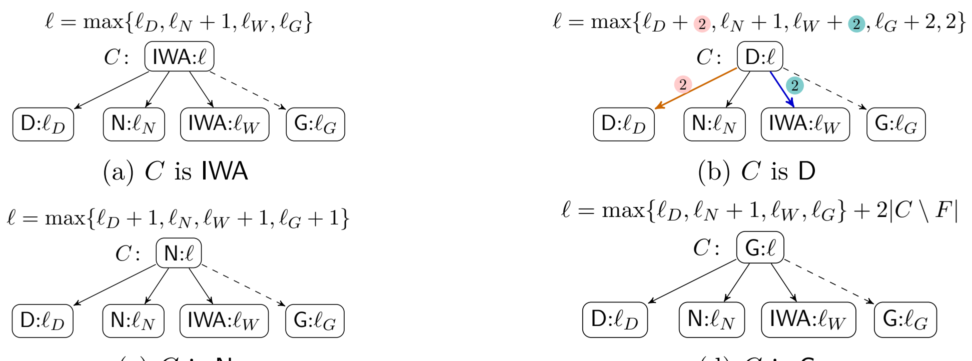
The algorithm from Section 4.1.4 computing bound on the maximum rank for states in each MSCC of an elevator automaton can be extended to general BAs. Non-elevator automata contain at least one nondeterministic accepting component. We refer to these MSCCs as *general* components and denote them as G. For deterministic, nonaccepting, or inherently weak MSCCs, we are able to set a rank bound independently of the number of states they contain, thanks to the structure of run DAGs for every possible word. However, for general MSCCs, the rank bound depends on the number of states, more precisely on the number of nonaccepting states in the component. This follows the original argument that maximum rank for each state in a Büchi automaton with n states is bounded by $2|Q|$. Since a maximum rank of a tight ranking depends only on the number of nonaccepting states, we can bound maximum rank in a general component C to $2|C \setminus F|$.

For a terminal MSCC C, we extend the algorithm assigning a label to each MSCC as follows (in the given order):

1. IWA:0 if C is inherently weak accepting,
2. D:2 if C is deterministic accepting, and
3. G:$2|C \setminus F|$ otherwise.

For non-terminal MSCCs, we use the rules from Figure 4.7. The structure of these rules is the same as for elevator automata in Section 4.1.4. For a non-terminal MSCC C, the rules for assigning a label are the following (in the given order):

1. If C is trivial, we try both rules from Figure 4.7a and Figure 4.7c and use the one with the smaller rank.
2. If C is IWA, we use the rule in Figure 4.7a.


$\ell = \max\{\ell_D, \ell_N + 1, \ell_W, \ell_G\}$
C: IWA:ℓ
D:ℓ_D
N:ℓ_N
IWA:ℓ_W
G:ℓ_G
$\ell = \max\{\ell_D + 2, \ell_N + 1, \ell_W + 2, \ell_G + 2, 2\}$
C: D:ℓ
2
2
D:ℓ_D
N:ℓ_N
IWA:ℓ_W
G:ℓ_G
$\ell = \max\{\ell_D + 1, \ell_N, \ell_W + 1, \ell_G + 1\}$
C: N:ℓ
D:ℓ_D
N:ℓ_N
IWA:ℓ_W
G:ℓ_G
$\ell = \max\{\ell_D, \ell_N + 1, \ell_W, \ell_G\} + 2|C \setminus F|$
C: G:ℓ
D:ℓ_D
N:ℓ_N
IWA:ℓ_W
G:ℓ_G

(a) C is IWA (b) C is D (c) C is N (d) C is G

Figure 4.7: Rules assigning types and rank bounds for non-elevator automata.

3. If C is deterministic accepting, we use the rule in Figure 4.7b.
4. If C is deterministic and non-accepting, we use one of the rules in Figure 4.7b and Figure 4.7c that gives us a smaller rank.
5. If C is nondeterministic and non-accepting, we use the rule in Figure 4.7c.
6. Otherwise, we use the rule in Figure 4.7d.

The maximum rank of each MSCC is assigned to all its states and macrostates with higher ranks are not generated. We denote the procedure as NonElevBound. Formally, the result of the algorithm is a mapping $\chi : Q \to \omega$ that gives a bound on the maximum rank to each state of the automaton.

Lemma 4.2. Let $\mathcal{A}$ be a Büchi automaton. NonElevBound($\mathcal{A}$) is a TRUB.

Proof. Consider some BA $\mathcal{A}$. Let $\mathcal{G}_\alpha$ be a run DAG over some word $\alpha \notin \mathcal{L}(\mathcal{A})$. In this proof, we use the claims and notation introduced in the proof of Lemma 4.1.

Claim 3: *Let C be a terminal* G *component in $\mathcal{G}_\alpha^{2k+1}$. Then, all vertices in $\mathcal{G}_\alpha$ labelled by C will have the rank at most $2k + 2|C \setminus Q_F|$.*

Proof: Since $2k + 1 > 0$, $\mathcal{G}_\alpha^{2k+1}$ does not contain any finite vertices. Since C is a terminal component, there is some $i \in \omega$ s.t. $\forall j > i : |level_{\mathcal{G}_\alpha^{2k+1}}(j) \cap C| < |level_{\mathcal{G}_\alpha^{2k+2}}(j) \cap C|$ (if we remove an endangered vertex, we decrease the width of the run DAG from some level at least by 1). Moreover, since endangered vertices do not contain accepting states, the previous observation can be refined to $|(level_{\mathcal{G}_\alpha^{2k+1}}(j) \cap C) \setminus Q_F| < |(level_{\mathcal{G}_\alpha^{2k+2}}(j) \cap C) \setminus Q_F|$. If we apply the reasoning multiple times, we get that in $\mathcal{G}_\alpha^{2k+2|C \setminus Q_F|}$, there remain only finite vertices labelled by a state from C, therefore the rank is at most $2k + 2|C \setminus Q_F|$. ■

Now we prove the main lemma. First, observe that after application of any rule, we have that D, IWA, and G components have an even rank and N components have an odd rank. We prove the lemma by induction on certain rules. In particular, we prove that if a state q was assigned rank k, $\mathcal{G}_\alpha^{k+1}$ does not contain any node labelled by q.

- Base case: If a terminal component C is IWA, from Claim 1 we obtain all states from C will have the rank 0. If a terminal component is D, then from Claim 2 we have that all states from C will have the rank bounded by 2. If a terminal component is G, from Claim 3 we have that all states from C will have the rank bounded by $2|C \setminus Q_F|$.

- Inductive case: Assume that for all states q from already processed components, if q was assigned by rank m, $\mathcal{G}_\alpha^{m+1}$ does not contain any node labelled by q.

Figure 4.7a Observe that after $\ell = \max\{\ell_D, \ell_N{+}1, \ell_W, \ell_G\}{-}1$ steps of the ranking procedure, in the worst case, all vertices labelled by C in $\mathcal{G}_\alpha^\ell$ are finite (otherwise it is a contradiction with the induction hypothesis). Therefore, in $\mathcal{G}_\alpha^{\ell+1}$ there are no vertices labelled by C. The ranks of vertices labelled by C is hence $\max\{\ell_D, \ell_N + 1, \ell_W, \ell_G\}$.

Figure 4.7b We prove that in $\mathcal{G}_\alpha^\ell$ all vertices labelled by C are finite (ℓ is from the rule). From the induction hypothesis, after $\ell - 1$ steps (in the worst case) all vertices labelled by adjacent D and IWA components are finite in $\mathcal{G}_\alpha^\ell$ (provided that the transitions are deterministic). Vertices labelled by adjacent N components are not present in $\mathcal{G}_\alpha^\ell$. Vertices labelled by adjacent G components are finite in $\mathcal{G}_\alpha^\ell$. Therefore, if there is some vertex v labelled by C in $\mathcal{G}_\alpha^\ell$ which is not finite, the only possibility is that for each $v' \in \mathit{reach}_{\mathcal{G}_\alpha^\ell}(v)$ we reach in $\mathcal{G}_\alpha^\ell$ from v' some vertex labelled by a state from the D, IWA components. This is however a contradiction with the transition determinism.

Figure 4.7c We prove that in $\mathcal{G}_\alpha^\ell$ all vertices labelled by C are endangered. This follows from the fact that after $\ell - 1$ steps no vertex labelled by the adjacent $\mathsf{D}, \mathsf{IWA}, \mathsf{G}$ components is present in $\mathcal{G}_\alpha^\ell$ (induction hypothesis). Therefore, all vertices labelled by C in $\mathcal{G}_\alpha^\ell$ are endangered.

Figure 4.7d From the induction hypothesis we have that in $\mathcal{G}_\alpha^\ell$ where $\ell = \max\{\ell_D, \ell_N + 1, \ell_W, \ell_G\}$ all vertices labelled by adjacent $\mathsf{D}, \mathsf{IW}, \mathsf{G}$ components are finite. Vertices labelled by adjacent N components are not present in $\mathcal{G}_\alpha^\ell$. Therefore, C is terminal in $\mathcal{G}_\alpha^{\ell+1}$. From Claim 3 we have that the rank of C is bounded by $\ell + 2|C \setminus Q_F|$. □

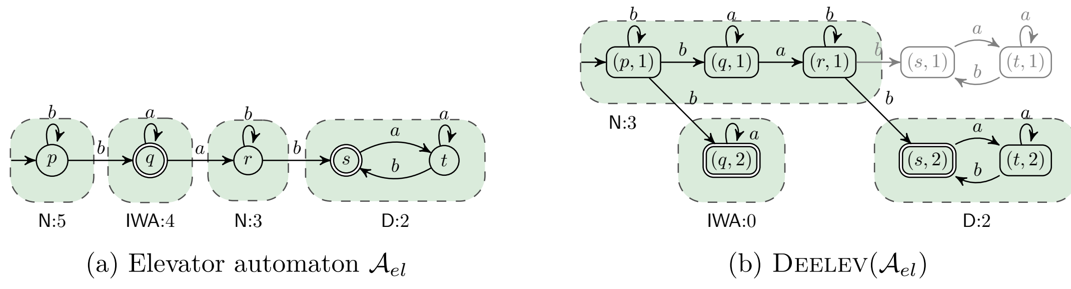
4.1.6 Efficient Complementation of Elevator Automata

The algorithm for assigning rank bounds to MSCCs of an elevator automaton, presented in Section 4.1.4, can in practice have a huge impact on the generated state space. However, we cannot bound the maximum rank by a constant, because it depends, among other, on the number of MSCCs. In this section, we show that it is possible to bound the rank by a constant if we construct an equivalent automaton with at most double the size of the input elevator automaton.

The increment of maximum rank for two successive MSCCs depends mainly on the alternation of accepting components and some nondeterminism. We can change the structure of an input elevator automaton such that for every possible run we start in a nonaccepting MSCC and then take a transition to deterministic or inherently weak MSCC at most once.

Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be an elevator automaton. The *deelevated* automaton $\textsc{Deelev}(\mathcal{A}) = (Q', \Sigma, \delta', I', F')$ is given as follows:

- $Q' = Q \times \{1, 2\}$,
- $I' = I \times \{1\}$,
- $\delta' = \delta_1 \cup \delta_2$ such that
  - $\delta_1((q, 1), a) = \delta(q, a) \times \{1, 2\}$,



N:5
IWA:4
N:3
D:2
(a) Elevator automaton $\mathcal{A}_{el}$
N:3
IWA:0
D:2
(b) Deelev($\mathcal{A}_{el}$)

Figure 4.8: An example of an elevator automaton $\mathcal{A}_{el}$ and a deelevated automaton Deelev($\mathcal{A}_{el}$). The bound on the maximum rank is decreased from 5 to 3.

  - $\delta_2((q, 2), a) = \delta(q, a) \times \{2\}$, and

- $F' = F \times \{2\}$.

Intuitively, we copy each MSCC with an accepting state and all transitions going to this MSCC, and we remove accepting conditions from the original MSCC. It is easy to see from the construction that the number of states of Deelev($\mathcal{A}$) is bounded by $2|Q|$. Any possible run on Deelev($\mathcal{A}$) starts in a nonaccepting MSCC, and it either stays in some nonaccepting MSCC or it moves to a deterministic accepting or inherently weak accepting MSCC where it stays forever. The bound on the maximum rank for Deelev($\mathcal{A}$) is therefore always 3, which gives us the upper bound $\mathcal{O}(16^n)$ for complementing elevator automata. This is based on the number of possible tight rankings for an automaton with sufficiently many states n and rank bound 3. An example of a deelevated automaton is given in Figure 4.8.

Lemma 4.3. Let $\mathcal{A}$ be a BA. Then, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\text{Deelev}(\mathcal{A}))$.

Proof. Let $\alpha \in \mathcal{L}(\mathcal{A})$ be a word. There is an accepting run $\rho = q_0 q_1 \ldots$ of $\mathcal{A}$ on α. For an accepting run ρ, there is an MSCC C and some $i \in \omega$ such that $\rho_k \in C$ for all $k \geq i$. There is an accepting run $\rho' = (q_0, 1) \ldots (q_{i-1}, 1)(q_i, 2)(q_{i+1}, 2) \ldots$ on Deelev($\mathcal{A}$).

Let $\alpha \in$ Deelev($\mathcal{A}$) be a word. There is an accepting run $\rho = (q_0, 1) \ldots (q_{i-1}, 1)(q_i, 2)$ $(q_{i+1}, 2) \ldots$ on Deelev($\mathcal{A}$). States q_i, q_{i+1}, … are in the same MSCC. There is therefore an accepting run $\rho' = q_0 q_1 \ldots$ of $\mathcal{A}$ on α. □

4.2 Data Flow Analysis

In this section, we propose a way to get bounds for maximum rank based on the structure of the automaton using *data flow analysis* [27]. In particular, rank bounds can be decreased based on the ranks and rankings of the local neighbourhood of the macrostates. For an input BA $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, the analysis is performed on the BA $\mathcal{K}_{\mathcal{A}} = (2^Q, \Sigma, \delta', \emptyset, \emptyset)$ where $\delta' = \{R \xrightarrow{a} S \mid S = \delta(R, a)\}$. The structure of $\mathcal{K}_{\mathcal{A}}$ is similar to the structure of the waiting part of Schewe($\mathcal{A}$). We are, however, not interested in the language of $\mathcal{K}_{\mathcal{A}}$, but only in its structure. We get the bounds for states in a macrostate based on the bounds of the states of its predecessors.

For a function $f : X \to Y$ and a set $S \subseteq X$, we define $f(S) = \{f(x) \mid x \in S\}$. In the following, we use $f \triangleleft \{x \mapsto y\}$ to denote the function $(f \setminus \{x \mapsto f(x)\}) \cup \{x \mapsto y\}$ for $x \in X$ and $y \in Y$. For $i \in \omega$ we use $\llbracket i \rrbracket$ to denote the largest even number smaller or equal to i.

4.2.1 Outer Macrostate Analysis

Our first analysis, called *outer macrostate analysis*, is based on the sizes of macrostates and is used for estimating their ranks. Since the rank of a run in SCHEWE($\mathcal{A}$) does not decrease once it enters a tight part, we can set a rank bound for each cycle of $\mathcal{K}_\mathcal{A}$ to $2m-1$ where m is the smallest number of nonaccepting states of macrostates in this cycle. The maximum rank of the macrostate is then given by the maximum rank of all the cycles going through this macrostate. The rank of each cycle can also be estimated by our elevator analysis from Section 4.1.

Since the number of cycles in $\mathcal{K}_\mathcal{A}$ can be double-exponential to the size of $\mathcal{A}$, we use data flow analysis instead of enumerating all cycles. The function $\mu : 2^Q \to \omega$ gives a maximum rank to each macrostate. For a macrostate S and its predecessors $R_1, \dots, R_i$, we use the update function $up_{out} : (2^Q \to \omega) \times (2^Q)^{i+1} \to \omega$, which is defined as follows: $up_{out}(\mu, S, R_1, \dots, R_i) = \min\{\mu(S), \max\{\mu(R_1), \dots, \mu(R_i)\}\}$. The new bound on the maximum rank of S is set to the smaller of the previous bound $\mu(S)$ and the largest of the bounds of all predecessors of S. The new value is propagated forward by the data flow analysis until the fixpoint is reached.

Lemma 4.4. If μ is a TRUB, then $\mu \lhd \{S \mapsto up_{out}(\mu, S, R_1, \dots, R_m)\}$ is a TRUB.

Proof. Let $\alpha \notin \mathcal{L}(\mathcal{A})$ and $\mathcal{G}_\alpha$ be the run DAG of $\mathcal{A}$ over α. Further, let us use $\mu' = \mu \lhd \{S \mapsto up_{out}(\mu, S, R_1, \dots, R_m)\}$.

1. There are finitely many $i \in \omega$ such that $level_\alpha(i) = S$. Let k be the last level of $\mathcal{G}_\alpha$ where S occurs (or 0 if S does not occur on any level of $\mathcal{G}_\alpha$). Then we can set the ℓ in the definition of a TRUB in (3.2) to be the least $\ell > k$ such that ℓ is a tight level. Then the condition holds trivially.

2. There are infinitely many $i \in \omega$ such that $level_\alpha(i) = S$. Then, since μ is a TRUB, let ℓ be the ℓ in (3.2) for which μ satisfies (3.2). We need to show that for every $k > \ell$ such that $level_\alpha(k) = S$, it holds that $\mu'(S) \geq f^\alpha_k$. Let $\mathcal{P} \subseteq \{R_1, \dots, R_m\}$ be the set of predecessors of all occurrences of S on $\mathcal{G}_\alpha$ below ℓ, i.e., for all $k > \ell$ such that $level_\alpha(k) = S$, we have $level_\alpha(k-1) \in \mathcal{P}$. Since the ranks of levels in a run DAG are lower for levels that are higher, it is sufficient to consider only the first such a k. Let R be the predecessor of S at k, i.e., $R = level_\alpha(k-1)$. Since we do not know which particular $R_j \in \mathcal{P}$ it is, we need to consider all $R_j \in \mathcal{P}$. Since $k-1$ is already a tight position, we have that $\mu'' = \mu \lhd \{S \mapsto \max\{\mu(R_1), \dots, \mu(R_m)\}\}$ is a TRUB for the same $\ell'' = \ell$ in (3.2). Further, μ is also a TRUB, therefore, $\mu' = \mu \lhd \{S \mapsto \min\{\mu''(S), \mu(S)\}\}$ for ℓ. □

Corollary 4.5. When started with a TRUB μ_0, the outer macrostate analysis terminates and returns a TRUB μ^*_{out}.

Proof. Let μ be a TRUB and $\mu' = \mu \lhd \{S \mapsto up_{out}(\mu, S, R_1, \dots, R_m)\}$. From Lemma 4.4 we have that μ' is a TRUB as well, which means that starting from μ_0 using up_{out} we get TRUBs only. Moreover, $\mu(P) \geq \mu'(P)$ and $\mu'(P) \geq 0$ for each $P \in 2^Q$. The fixpoint evaluation hence eventually stabilizes. □

4.2.2 Inner Macrostate Analysis

Inner macrostate analysis is used for estimating rankings within macrostates. In a super-tight run, the rank of a state $q \in S$ is bounded by the rank of the predecessors of q. The function $\mu : 2^Q \to \mathcal{R}$, where $\mathcal{R}$ denotes the set of all rankings, gives a ranking for each macrostate.

Let $f, f' \in \mathcal{R}$ be rankings. We use $f \sqcup f'$ to denote the ranking $\{q \mapsto \max\{f(q), f'(q)\} \mid q \in Q\}$, and $f \sqcap f'$ to denote the ranking $\{q \mapsto \min\{f(q), f'(q)\} \mid q \in Q\}$. Moreover, we define $\mathit{max\text{-}succ\text{-}rank}^a_S(f) = \max_{\leq}\{f' \in \mathcal{R} \mid f'(q') \leq f(q)$ for each $q \in S$ and $q' \in \delta(q, a)\}$ and a function $\mathit{dec}\colon \mathcal{R} \to \mathcal{R}$ such that $\mathit{dec}(\theta)$ is the ranking θ' for which

$$\theta'(q) = \begin{cases} \theta(q) - 1 & \text{if } \theta(q) = \mathit{rank}(\theta) \text{ and } q \notin F, \\ \llbracket \theta(q) - 1 \rrbracket & \text{if } \theta(q) = \mathit{rank}(\theta) \text{ and } q \in F, \\ \theta(q) & \text{otherwise.} \end{cases} \tag{4.1}$$

Intuitively, $\mathit{max\text{-}succ\text{-}rank}^a_S(f)$ is the maximum ranking that can be reached from macrostate S with ranking f over a and $\mathit{dec}(\theta)$ decreases the maximum ranks in a ranking θ by one (or by two for even maximum ranks and accepting states).

For a macrostate S and its predecessors $R_1, \ldots, R_i$, we use the update function $\mathit{up}_{\mathit{in}}\colon (2^Q \to \mathcal{R}) \times (2^Q)^{i+1} \to \mathcal{R}$, which is defined by the following algorithm:

```
1 $\mathit{up}_{\mathit{in}}(\mu, S, R_1, \ldots, R_m)$:
2     foreach $1 \leq i \leq m$ and $a \in \Sigma$ do
3         if $\delta(R_i, a) = S$ then
4             $g^a_i \leftarrow \mathit{max\text{-}succ\text{-}rank}^a_{R_i}(\mu(R_i))$
5     $\theta \leftarrow \mu(S) \sqcap \bigsqcup\{g^a_i \mid g^a_i$ is defined$\}$;
6     if $\mathit{rank}(\theta)$ is even then $\theta \leftarrow \mathit{dec}(\theta)$;
7     return $\theta$;
```

The update function updates $\mu(q)$ for every $q \in S$ to hold the maximum rank compatible with the rank of its predecessors.

Lemma 4.6. If μ is a TRUB, then $\mu \lhd \{S \mapsto \mathit{up}_{\mathit{in}}(\mu, S, R_1, \ldots, R_m)\}$ is a TRUB.

Proof. Let $\alpha \notin \mathcal{L}(\mathcal{A})$ and $\mathcal{G}_\alpha$ be the run DAG of $\mathcal{A}$ over α. Further, let us use $\mu' = \mu \lhd \{S \mapsto \mathit{up}_{\mathit{in}}(\mu, S, R_1, \ldots, R_m)\}$. First, we prove the following claim:

Claim 4: *Let μ_1, μ_2 be two TRUBs wrt α. Then μ', defined as $\mu'(S) := \mu_1(S) \sqcap \mu_2(S)$ is a TRUB wrt α.*

Proof: The proof follows from the definition (with choosing $\ell' = \max\{\ell_1, \ell_2\}$) where ℓ_1 is from the definition of a TRUB for μ_1 and ℓ_2 is for μ_2. ■

We need to consider the following two cases:

1. There are finitely many $i \in \omega$ such that $\mathit{level}_\alpha(i) = S$. Let k be the last level of $\mathcal{G}_\alpha$ where S occurs (or 0 if S does not occur on any level of $\mathcal{G}_\alpha$). Then we can set the ℓ in the definition of a TRUB in (3.2) to be the least $\ell > k$ such that ℓ is a tight level. Then the condition holds trivially.

2. There are infinitely many $i \in \omega$ such that $\mathit{level}_\alpha(i) = S$. Then, since μ is a TRUB, let ℓ be the ℓ in (3.2) for which μ satisfies (3.2). We need to show that for every $k > \ell$ such that $\mathit{level}_\alpha(k) = S$, it holds that $\mu'(S) \geq f^\alpha_k$. Let $\mathcal{P} \subseteq \{R_1, \ldots, R_m\}$ be the set of predecessors of all occurrences of S on $\mathcal{G}_\alpha$ below ℓ, i.e., for all $k > \ell$ such that $\mathit{level}_\alpha(k) = S$, we have $\mathit{level}_\alpha(k-1) \in \mathcal{P}$. Since the ranks of levels in a run DAG are lower for levels that are higher, it is sufficient to consider only the first

such a k. Let R be the predecessor of S at k, i.e., $R = \mathit{level}_\alpha(k-1)$. Since we do not know which particular $R_j \in \mathcal{P}$ it is, we need to consider all $R_j \in \mathcal{P}$. Let $M = \{\mathit{max\text{-}succ\text{-}rank}^a_{R_j}(\mu(R_j)) \mid R_j \in \mathcal{P}, a \in \Sigma\}$. Then, since μ is a TRUB, $\bigsqcup M$ will also be a TRUB. Moreover, from Claim 4, $\theta = \mu(S) \sqcap \bigsqcup M$ will also be a TRUB, and so $\theta \geq f^\alpha_k$. Then, if the rank of θ is even, we can decrease it to the nearest odd rank, since tight rankings are, by definition, of an odd rank. □

Corollary 4.7. When started with a TRUB μ_0, the inner macrostate analysis terminates and returns a TRUB μ^*_{in}.

Proof. Let μ be a TRUB and $\mu' = \mu \lhd \{S \mapsto \mathit{up}_{in}(\mu, S, R_1, \ldots, R_m)\}$. From Lemma 4.6 we have that μ' is a TRUB as well, which means that starting from μ_0 using up_{in} we get TRUBs only. Moreover, $\mu(P) \geq \mu'(P)$ and $\mu'(P) \geq \{q \mapsto 0 \mid q \in Q\}$ for each $P \in 2^Q$. The fixpoint evaluation hence eventually stabilizes. □

4.3 Optimization of Inherently Weak BA Complementation

In this section, we introduce new optimizations of inherently weak Büchi automata complementation. Our optimizations are based on the Miyano-Hayashi construction [26] described in Section 3.5. Our two optimizations are inspired by optimizations of the determinization algorithm for automata over finite words [13] and by macrostates saturation in rank-based complementation of Büchi automata [9]. In both optimization, simulation relations are used in order to construct a smaller automaton. We either try to make the macrostates of the complement as small as possible (*pruning*) or as big as possible (*saturating*). This construction can help reducing the generated state space, because more states obtained from the original Miyano-Hayashi construction [26] can be mapped to one pruned or saturated macrostate.

Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be an inherently weak BA. We first construct an equivalent BA $\mathcal{W} = (Q, \Sigma, \delta, I, F_W)$, where F_W contains all states from inherently weak accepting MSCCs of $\mathcal{A}$. We then convert $\mathcal{W}$ to an equivalent co-Büchi automaton $\mathcal{C} = (Q, \Sigma, \delta, I, F_C = Q \setminus F_W)$. We use $\preceq^{\mathcal{W}}_{di}$ to denote a *direct simulation* on $\mathcal{W}$ and $\preceq^{\mathcal{C}}_{f}$ to denote a *fair simulation* on $\mathcal{C}$. A fair simulation $\preceq^{\mathcal{C}}_{f}$ can be approximated by a direct simulation $\preceq^{\mathcal{W}}_{di}$.

Let $\sqsubseteq$ be a relation on the states of $\mathcal{C}$ defined as follows: $p \sqsubseteq q$ iff (i) $p \preceq^{\mathcal{C}}_{f}$, (ii) q is reachable from p in $\mathcal{C}$, and (iii) either p is not reachable from q in $\mathcal{C}$ or $p = q$.

We define two adjustment functions $\mathit{pr}, \mathit{sat}\colon 2^Q \to 2^Q$ for each $S \subseteq Q$ as follows:

- *pruning*: $\mathit{pr}(S) = S'$ where $S' \subseteq S$ is the lexicographically smallest set (given a fixed ordering on Q) such that $\forall q \in S \exists q' \in S'\colon q \sqsubseteq q'$ and
- *saturating*: $\mathit{sat}(S) = \lfloor S \rfloor_{\preceq^{\mathcal{C}}_{f}}$, where $\lfloor S \rfloor_{\preceq^{\mathcal{C}}_{f}} = \{p \in Q \mid \exists q \in Q\colon p \preceq^{\mathcal{C}}_{f} q\}$.

For a given co-BA $\mathcal{C}$ and an adjustment function $\theta : 2^Q \to 2^Q$, the construction $\textsc{Mihay}_\theta$ produces a BA $\textsc{Mihay}_\theta(\mathcal{C}) = (Q', \Sigma, \delta', I', F')$, whose components are defined as follows:

- $Q' = 2^Q \times 2^Q$,
- $I' = \{(\theta(I), \theta(I) \setminus F_C)\}$,
- $\delta'((S, B), a) = (S', B')$ where
  - $S' = \theta(\delta(S, a))$,

- and
  - $B' = S' \setminus F_C$ if $B = \emptyset$ or
  - $B' = (\delta(B, a) \cap S') \setminus F_C$ if $B \neq \emptyset$, and
- $F' = 2^Q \times \{\emptyset\}$.

In the following we fix a co-BA $\mathcal{C} = (Q, \Sigma, \delta, I, F_C)$. We use $p \rightsquigarrow q$ to denote that q is reachable from p. Let $\alpha \in \Sigma^\omega$ be a word. Let Π, Π' be sets of traces over α. We say that Π and Π' are *acc-equivalent*, denoted as $\Pi \sim \Pi'$ if $\exists \pi \in \Pi : \pi$ is accepting in $\mathcal{C}$ iff $\exists \pi' \in \Pi' : \pi'$ is accepting in $\mathcal{C}$. Let $\rho = S_1 S_2 \ldots$ be a sequence of sets of states and α be a word. We define Π_ρ to be a set of traces over α matching the sets of states. Formally, $\Pi_\rho = \{\pi \mid \pi \text{ over } \alpha, \pi_i \in S_i \text{ for each } i\}$. For a trace $\pi = \pi_0 \pi_1 \ldots$ we use $\pi_{i:\,\omega}$ to denote a trace $\pi_{i:\omega} = \pi_i \pi_{i+1} \ldots$. Moreover, for a set of traces Π_ρ, we define $\Pi_{\rho_{i:\omega}}$ as $\Pi_{\rho_{i:\omega}} = \{\pi_{j:\omega} \mid \pi \in \Pi_\rho \text{ and } j \geq i\ \}$. We also define $\Pi^{\cup}_\rho = \bigcup_{i \in \omega} \Pi_{\rho_{i:\omega}}$. Further, for a set of states B we use ρ^B_α to denote the sequence $S_1 S_2 \ldots$ s.t. $S_1 = B$, $S_{i+1} = \delta(S_i, \alpha_i)$ for each $i \in \omega$. We use ρ_α to denote ρ^I_α. Moreover, for a given mapping $\theta : 2^Q \to 2^Q$ and a sequence of sets of states ρ we define $\theta(\rho) = \theta(\rho_1)\theta(\rho_2) \ldots$. A trace π is *eventually fair-simulated* by π' if there is some $i \in \omega$ s.t. $\pi_{i:\omega} \preceq^{\mathcal{C}}_f \pi'_{i:\omega}$.

Lemma 4.8. Let α be a word, $\Pi_{\rho_\alpha} \sim \Pi_{\rho'_\alpha}$, and $\Pi_{\rho_\alpha} \subseteq \Pi_{\rho'_\alpha}$. Then, $\Pi_{\rho_\alpha} \sim \Pi^{\cup}_{\rho'_\alpha}$.

Proof. Assume that $\Pi_{\rho_\alpha} \sim \Pi_{\rho'_\alpha}$ and $\Pi_{\rho_\alpha} \subseteq \Pi_{\rho'_\alpha}$. Since $\Pi_{\rho'_\alpha} \subseteq \Pi^{\cup}_{\rho'_\alpha}$, it means that if there is an accepting trace in Π_{ρ_α}, there is the same accepting trace in $\Pi^{\cup}_{\rho'_\alpha}$. If there is no accepting trace in Π_{ρ_α}, it means that all traces contain infinitely many accepting states. Hence, every infinite suffix is also an accepting trace and, therefore, $\Pi^{\cup}_{\rho'_\alpha}$ contains all traces that are not accepting in $\mathcal{C}$ (i.e., with infinitely many accepting states). $\square$

Lemma 4.9. Let α be a word. Then, $\Pi_{\rho_\alpha} \sim \Pi_{pr(\rho_\alpha)}$.

Proof. Since $\Pi_{pr(\rho_\alpha)} \subseteq \Pi_{\rho_\alpha}$, it suffices to show that if there is an accepting trace $\pi \in \Pi_{\rho_\alpha}$, there is also an accepting trace $\pi' \in \Pi_{pr(\rho_\alpha)}$. We show that there is $\pi' \in \Pi_{pr(\rho_\alpha)}$ s.t. π is eventually fair-simulated by π'. If $\pi' = \pi$, we are done. Now, assume that $\pi' \neq \pi$ and that there is a maximum set of traces $P = \{\pi^1, \pi^2, \ldots\} \subseteq \Pi_{\rho_\alpha}$ with indices $\ell_1 < \ell_2 < \ldots$ s.t. $p_i = \pi^i_{\ell_i} \sqsubseteq \pi^{i+1}_{\ell_i} = p'_i$ for each i, and moreover $\pi_1 = \pi$. We show that P is finite by showing that $p'_i \neq p'_j$ for each $i \neq j$. Assume that $p'_j = p'_i$ for some $i < j$. But then we have $p'_i \rightsquigarrow p_j \sqsubseteq p'_j = p'_i$ meaning that $p'_i \rightsquigarrow p'_j$ (from the definition of $\sqsubseteq$). From the definition of $\sqsubseteq$ we also have that p_j is not reachable from $p'_j = p'_i$, which is a contradiction. Since the set $P = \{\pi_1, \ldots, \pi_n\}$ is maximal and finite, we have $\pi_n \in \Pi_{pr(\rho_\alpha)}$. Moreover, $\pi' = \pi_n$ eventually fair-simulates π, which concludes the proof. $\square$

Lemma 4.10. Let α be a word. Then, $\Pi_{\rho_\alpha} \sim \Pi^{\cup}_{sat(\rho_\alpha)}$.

Proof. First observe that $\Pi_{\rho_\alpha} \subseteq \Pi^{\cup}_{sat(\rho_\alpha)}$. Therefore, it suffices to show that if there is an accepting trace $\pi \in \Pi^{\cup}_{sat(\rho_\alpha)}$, there is also an accepting trace $\pi' \in \Pi_{\rho_\alpha}$. We fix $\rho = \rho_\alpha$. Consider some accepting trace $\pi \in \Pi^{\cup}_{sat(\rho_\alpha)}$. If $\pi \in \Pi_{\rho_\alpha}$, we are done. If not, there is some position ℓ s.t. $\pi \in \Pi_{\rho_{\ell:\omega}}$ and $\pi_1 \preceq_f q$ where $q \in \rho_\ell$. Therefore, there is some trace $\pi' \in \rho$ s.t. $\pi'_\ell = q$. Moreover, π is accepting, hence there is a trace π'' leading from q, which is accepting as well. Hence, $\pi'_{1:\ell}.\pi'' \in \rho$ and moreover this trace is accepting. $\square$

Lemma 4.11. Let θ be an adjustment function. If $\Pi_{\rho_\alpha} \sim \Pi^{\cup}_{\theta(\rho_\alpha)}$ for each $\alpha \in \Sigma^\omega$ then $\mathcal{L}(\textsc{MiHay}_\theta(\mathcal{C})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{C})$.

Proof. Consider a word $\alpha \in \mathcal{L}(\mathcal{C})$. Hence, there is an accepting trace $\pi \in \Pi_{\rho_\alpha}$ and also an accepting trace $\pi' \in \Pi_{\theta(\rho_\alpha)_{k:\omega}}$ for some $k \in \omega$. Since π' emerges eventually in the B set, α is not accepted by $\mathcal{L}(\textsc{MiHay}_\theta(\mathcal{C}))$.

Conversely, assume that $\alpha \notin \mathcal{L}(\mathcal{C})$. Then, all traces in Π_{ρ_α} as well as in $\Pi^{\cup}_{\theta(\rho_\alpha)}$ contain infinitely many accepting states. Hence, we flush B-set infinitely many times yielding $\alpha \in \mathcal{L}(\textsc{MiHay}_\theta(\mathcal{C}))$. □

Lemma 4.12. For a co-BA $\mathcal{C}$, $\mathcal{L}(\textsc{MiHay}_{sat}(\mathcal{C})) = \mathcal{L}(\textsc{MiHay}_{pr}(\mathcal{C})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{C})$.

Proof. We get the proof for Lemma 4.12 directly from the fact that $\Pi_{pr(\rho_\alpha)} \subseteq \Pi_{\rho_\alpha}$ for any word α, and from Lemmas 4.8, 4.9, and 4.11. The correctness of the construction for $\textsc{MiHay}_{sat}(\mathcal{C})$ is given by Lemmas 4.10 and 4.11. □

4.4 Optimization of Semi-Deterministic BA Complementation

A problem with the NCSB algorithm for complementing semi-deterministic Büchi automata described in Section 3.6 is a high degree of nondeterminism. In this section, we propose an optimization of the original NCSB construction, inspired by the MaxRank construction in rank-based complementation from [16], which we denote as NCSB-MaxRank.

Let $\mathcal{A} = (Q_1 \uplus Q_2, \Sigma, \delta = \delta_1 \uplus \delta_2 \uplus \delta_t, I, F)$ be a semi-deterministic BA where Q_2 is the set of states reachable from some accepting state and Q_1 is the rest, $\delta_1 = \delta_{|Q_1}$, $\delta_2 = \delta_{Q_2}$, and δ_t is the transition function between Q_1 and Q_2. The NCSB-MaxRank construction produces a BA NCSB-MaxRank$(\mathcal{A}) = (Q', \Sigma, \delta', I', F')$ whose components are the following:

- $Q' = \{(N, C, S, B) \in 2^{Q_1} \times 2^{Q_2} \times 2^{Q_2 \setminus F} \times 2^{Q_2} \mid B \subseteq C\}$,
- $I' = \{(Q_1 \cap I, Q_2 \cap I, \emptyset, Q_2 \cap I)\}$,
- $\delta' = \gamma_1 \cup \gamma_2$ where
  - $\gamma_1((N, C, S, B), a) = \{(N', C', S', B')\}$ where
    - $N' = \delta_1(N, a)$,
    - $S' = \delta_2(S, a)$,
    - $C' = (\delta_t(N, a) \cup \delta_2(C, a)) \setminus S'$, and
    - $B' = C'$ if $B = \emptyset$, otherwise $B' = \delta_2(B, a) \cap C'$,
  - If $B' \cap F \neq \emptyset$, then $\gamma_2((N, C, S, B), a) = \emptyset$. Otherwise, we set $\gamma_2((N, C, S, B), a) = \{N', C'', S'', B''\}$ with
    - $B'' = \emptyset$,
    - $S'' = S' \cup B'$, and
    - $C'' = C' \setminus S''$.
- $F' = \{(N, C, S, B) \in Q' \mid B = \emptyset\}$.

NCSB-MaxRank reduces the degree of nondeterminism by providing at most two choices for each macrostate. The first choice is to keep all states in B and the second choice is to move all states from B to S if B contains no accepting state. The construction is incomparable to the original NCSB algorithm [5] due to the condition $C' \subseteq \delta_2(C \setminus F, a)$, which does not generally hold in NCSB-MaxRank.

Lemma 4.13. Let $B \subseteq Q_2$ be a set of deterministic states and let α be a word. If $\alpha \notin \mathcal{L}(\mathcal{A})$, then $\exists k : \forall \ell \geq k : (\rho_\alpha^B)_\ell \cap F = \emptyset$.

Proof. Assume that $\alpha \notin \mathcal{L}(\mathcal{A})$. Since B is a set of states of the deterministic part, we have $|\Pi_{\rho_\alpha^B}| \leq |B|$. Moreover, for each trace $\pi \in \Pi_{\rho_\alpha^B}$ there is some k_π s.t. $\pi_\ell \notin F$ for each $\ell \geq k_\pi$. Taking $k = \max\{k_\pi \mid \pi \in \Pi_{\rho_\alpha^B}\}$, we fulfill the condition of the lemma. $\square$

Lemma 4.14. Let $\mathcal{A}$ be an SDBA. Then $\mathcal{L}(\text{NCSB-MaxRank}(\mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

Proof. First, we prove that if $\alpha \in \mathcal{L}(\mathcal{A})$, then $\alpha \notin \mathcal{L}(\text{NCSB-MaxRank}(\mathcal{A}))$. In that case, there is an accepting run ρ on α in $\mathcal{A}$. Moreover, $\rho_\ell \in Q_2$ for some $\ell \in \omega$ and for all $k \geq l$. Therefore, for every run $R = (N_1, C_1, S_1, B_1)\dots$ on α in NCSB-MaxRank$(\mathcal{A})$, we have that either $\rho_\ell \in S_\ell$ or $\rho_\ell \in C_\ell$. Now assume the first case, $\rho_\ell \in S_\ell$. At some point, we reach an accepting state in ρ ($\rho_k \in Q_F$, $k \geq \ell$). $\rho_k \in S_k$ therefore means that R is a finite trace of at most $k-1$ elements. Now we turn to the second case, $\rho_\ell \in C_\ell$. In that case, either $\rho_l \in C_l$ and $\rho_l \in B_l$ for each $l \geq l_0 \geq \ell$, or we apply γ_2 and move ρ to S, i.e., $\rho_m \in S_m$ for some $m \geq \ell$. In the first case, B is not empty anymore, hence R is not accepting. In the latter, we get the case similar to the first examined run $\rho_\ell \in S_ell$. Hence, $\alpha \notin \mathcal{L}(\text{NCSB-MaxRank}(\mathcal{A}))$.

Now, we prove that if $\alpha \notin \mathcal{L}(\mathcal{A})$, then $\alpha \in \mathcal{L}(\text{NCSB-MaxRank}(\mathcal{A}))$. We construct an accepting run R on α in $\mathcal{A}$. Let R_0 be a macrostate $R_0 = (N_1, C_1, S_1, B_1) = (Q_1 \cap I, Q_2 \cap I, \emptyset, Q_2 \cap I)$. From Lemma 4.13 we have that there is a k_1 s.t. $\forall \ell \geq k_1 : (\rho_\alpha^{B_1})_\ell \cap F = \emptyset$. We set $R_{i+1} = \gamma_1(R_i)$ for $1 \leq i < k$. Further, we set $R_{k+1} = \gamma_2(R_k)$. Then, we use Lemma 4.13 (on $\alpha_{k_1:\omega}$) to obtain a position k_2 giving us the point where γ_2 is applied. Such a constructed run R is infinite, because Lemma 4.13 ensures that we cannot reach an accepting state from S on α. It remains to show that R is accepting. From the construction, we have that γ_2 was used infinitely many times (and each successor of γ_2 is an accepting state). The run therefore contains infinitely many accepting states. $\square$

Chapter 5

Implementation

The optimizations presented in this thesis are implemented in the tool Ranker [14] in C++. We added these optimizations on top of the techniques from [16]. Ranker uses optimized rank-based complementation for general BAs and also optimized special constructions for complementing inherently weak and semi-deterministic automata.

5.1 Architecture

Ranker [14] is a publicly available command line tool for complementing Büchi automata, written in C++. It accepts input Büchi automata in the HOA [3] or the BA [1] format. Both state-based and transition-based input Büchi automata are supported. The architecture is shown in Figure 5.1. After preprocessing, the input automaton is complemented using a complementation procedure chosen based on the structural properties of the automaton, and then postprocessed.

5.1.1 Preprocessing and Postprocessing

Ranker supports various options for preprocessing, including reduction of the input automaton, deelevation, saturation of accepting states, or feature extraction.

In order to reduce the input automaton before complementation, Ranker uses quotienting based on *direct simulation* [25] (`--preprocess=red`). Inherently weak and semi-deterministic automata are also transformed into equivalent transition-based BAs (TBA), since this may reduce the number of states. We do not transform other BAs into TBAs. Even though it could reduce the number of states, our optimizations of rank-based complementation procedure are more effective on state-based automata. For elevator automata, we can use some of the *deelevation* strategies. Deelevation decreases the rank bounds for rank-based complementation at the cost of at most doubling the number of states. Ranker supports three different deelevation strategies: (i) `--preprocess=copyall` where every component is deelevated (as described in Section 4.1.6), (ii) `--preprocess=copyiwa` where only inherently weak accepting components are deelevated, and (iii) `--preprocess=copyheur` which combines two previous methods: if the input BA is inherently weak and the the rank bound estimation is at least 5, then all MSCCs with an accepting state/transition are deelevated; if on all paths from all initial states, the first non-trivial MSCC is non-accepting, then the initial part of the BA (up to the first non-trivial MSCC) is determinized and the sizes of macrostates in the rank-based complementation are therefore reduced.

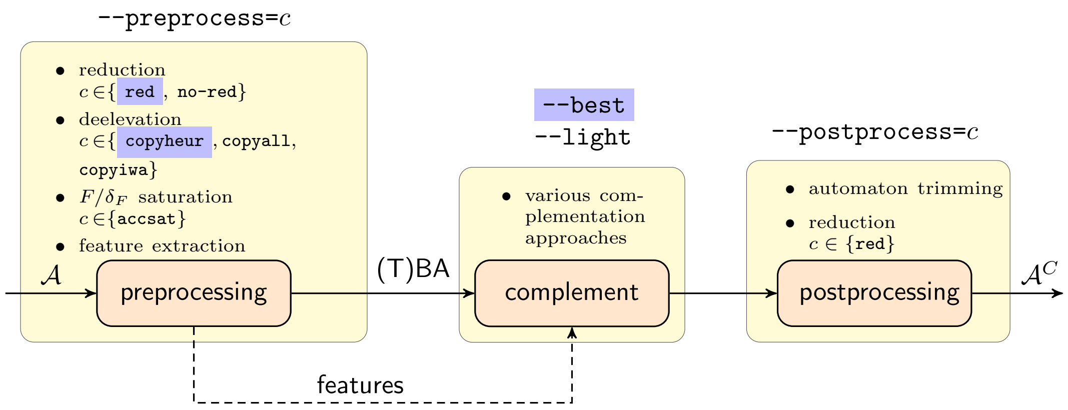


Figure 5.1: Overview of the architecture of Ranker with the most important command-line options. Default settings are highlighted in blue. F and δ_F denote accepting states and transitions, respectively.

Using `--preprocess=accsat`, Ranker can *saturate* accepting states or transitions in the input BA. The reason for this is that a higher number of accepting states reduces the maximum rank in the rank-based complementation. On the other hand, this technique is not always beneficial for other optimizations, since it may, for example, break the structure for elevator rank estimation.

During preprocessing, Ranker also extracts information about the input BA that helps the complementation procedure: for example the type of the BA, therank bound for individual states, etc.

The preprocessed automaton is then complemented and the result is postprocessed. Ranker removes useless states of the complement and optionally reduces the result using *direct simulation* [25] (`--postprocess=red`).

5.1.2 Complementation

The complementation procedure is chosen based on the type of the input Büchi automaton. We have a different procedure for each of the following types: inherently weak, semi-deterministic, and other BAs (ordered by decreasing priority). See Figure 5.2 for an overview of complementation approaches used in Ranker.

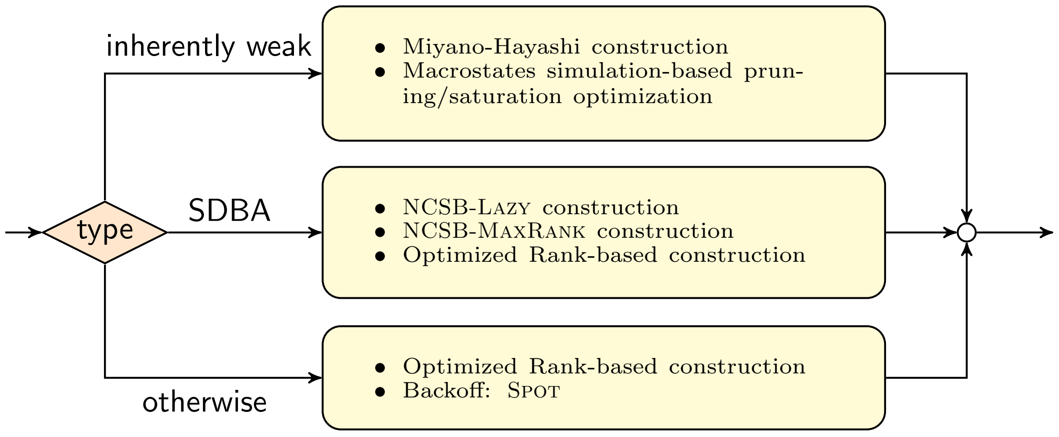

inherently weak
Miyano-Hayashi construction
Macrostates simulation-based pruning/saturation optimization
type
SDBA
NCSB-Lazy construction
NCSB-MaxRank construction
Optimized Rank-based construction
otherwise
Optimized Rank-based construction
Backoff: Spot

Figure 5.2: Overview of complementation approaches used in Ranker.

For *inherently weak* BAs, both the Miyano-Hayashi construction [26] and its optimization for macrostates saturation (described in Section 4.3) are used. By default (`--best`), Ranker constructs the complement using both approaches and then outputs the smaller result. For the option `--light`, only the optimized construction is used.

For *semi-deterministic* BAs, by default (`--best`) both NCSB-MaxRank (described in Section 4.4) and optimized rank-based construction with advanced rank estimation [16, 15] is used and the smaller result is picked. For the option `--light`, only NCSB-MaxRank is used. We can also turn on the `NCSB-Lazy` construction from [9] by using `--ncsb-lazy`, but this algorithm usually gives worse results.

For other BAs, we use the optimized rank-based complementation construction from [16, 15] with Spot as the backoff [16]. Ranker can determine if the input automaton has a structure bad for the rank-based procedure and use another approach.

Chapter 6

Experimental Evaluation

In this chapter, we compare Ranker with other state-of-the-art tools for Büchi automata complementation and show that it can produce a strictly smaller complement than other state-of-the-art tools in the majority of cases. Moreover, we show that even if the original rank-based complementation is not very efficient, with all our optimizations it becomes competitive to other BA complementation approaches.

6.1 Tools and Evaluation Environment

In our experiments, we compared Ranker with other state-of-the-art tools, namely, Goal [36] (implementing Piterman [28], Safra [31], and Fribourg [2]), Spot 2.9.3 [10] (implementing Redziejowski's algorithm [30]), Seminator 2 [4], LTL2dstar 0.5.4 [19], and Roll [21]. All tools were set to the mode where they output a state-based BA.

We tested the correctness of Ranker using Spot's `autcross` on all BAs in our benchmark. The experimental evaluation was performed on a 64-bit GNU/Linux Debian workstation with an Intel(R) Xeon(R) CPU E5-2620 running at 2.40 GHz with 32 GiB of RAM and using a timeout of 5 minutes.

6.2 Structure of Experiments

In this chapter, we present results of two sets of experiments. The first set was performed after the optimized rank-based construction for elevator automata and data flow analysis (described in Section 4.1) were implemented on top of the previous version of Ranker from [16]. The results of these optimizations were published in [15]. The second set was performed on the version of Ranker with optimizations from Sections 4.3 and 4.4 implemented on top of the version from [15]. In our experiments, we focus mainly on the number of states of the complement. Axes in all scatter plots are logarithmic.

The first experiment from each set shows the effectiveness of our heuristics for reducing the generated state space by comparing the sizes of complemented BAs with other rank-based algorithms without postprocessing. These results are useful for applications where postprocessing is not needed, for example language inclusion or equivalence checking. We compare Ranker with Schewe (the version Reduced Average Outdegree from [32], implemented in Goal under `-m rank -tr -ro`), and also with its previous version to see the impact of our new optimizations.

The second experiment from each set compares Ranker with other state-of-the-art tools. It compares sizes of output BAs, therefore, each automaton was postprocessed with `autfilt` (simplification level `--high`). The statistics for each set of experiments are shown in a table. For the second experiment, scatter plots compare Ranker with Spot and Roll, which currently give the best results among other state-of-the-art tools.

6.3 Elevator Automata and Data Flow Analysis

In this section, we show the effect of our optimized rank-based construction for elevator automata and data-flow analysis. We implemented these optimizations on top of the previous version of Ranker from [16].

6.3.1 Datasets

We used two datasets for our experiments: (i) `random` with 11,000 BAs over a two letter alphabet used in [35], which were randomly generated via the Tabakov-Vardi approach [34], starting from 15 states and with various different parameters, and (ii) `LTL` containing 1,721 BAs over larger alphabets (up to 128 symbols) used in [4], obtained from LTL formulae from literature (221) or randomly generated (1500). The automata were preprocessed using Rabit [24] and Spot's `autfilt` (using the `--high` simplification level), transformed to state-based acceptance BAs (if they were not already), and converted to the HOA format [3]. From this set, we removed automata that were (i) semi-deterministic, (ii) inherently weak, (iii) unambiguous, or (iv) having an empty language, since for these automata types there exist more efficient complementation procedures than for unrestricted BAs [5, 4, 6, 23]. In the end, we were left with 2 592 (`random`) and 414 (`LTL`) *hard* automata. We use `all` to denote their union (3 006 BAs). Of these hard automata, 458 were elevator automata.

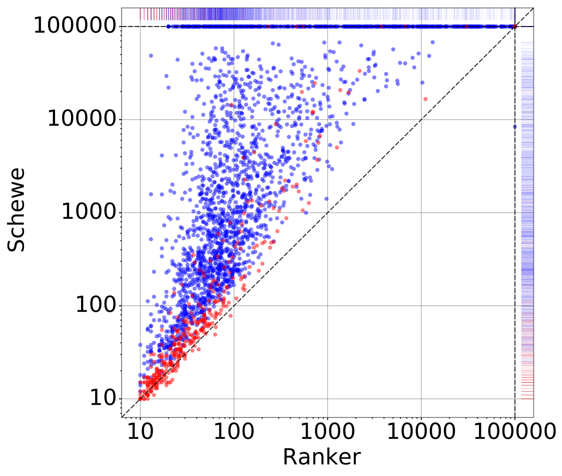
6.3.2 Comparison with Rank-Based Algorithms

Our first experiment shows the effectiveness of our optimizations by comparing the sizes of complemented Büchi automata without postprocessing. Figure 6.1 compares the number of states of the automata generated by Ranker with the automata generated by Schewe [32] and the previous version of Ranker from [16], denoted as Ranker$_{\text{Old}}$. We can see that the improvement was in many cases exponential when compared not only with Schewe, but also with the previous optimizations in Ranker$_{\text{Old}}$.

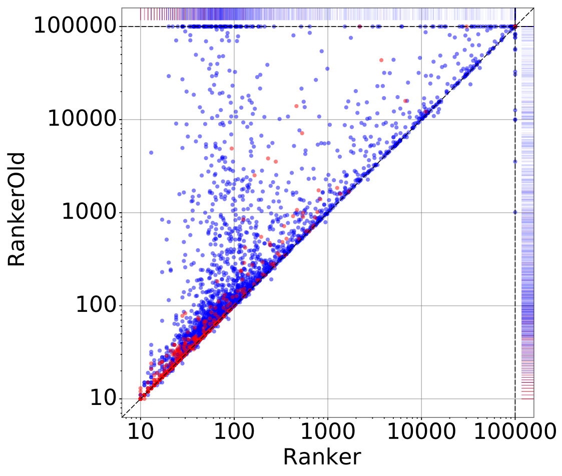
The upper part of Table 6.3 gives summary statistics. The number of timeouts decreased by 23% and the median decreased by 44% w.r.t. Ranker$_{\text{Old}}$.

6.3.3 Comparison with Other Tools

Our second experiment compares the number of states of the complement generated by Ranker with other state-of-the-art tools with postprocessing. Scatter plots in Figure 6.2 show a comparison of Ranker with Spot and Roll. Figure 6.2a shows that Ranker produces a smaller BA than Spot in the majority of cases, especially on BAs from `random`. Roll uses a learning-based approach, which is completely different from any other tool. This approach can output a much smaller automaton in some cases, but it is a more heavyweight technique and the number of timeouts is therefore much higher compared to other tools.



(a) RANKER vs SCHEWE



(b) RANKER vs RANKER$_{\text{OLD}}$

Figure 6.1: Comparison of the state space generated by our optimizations and other rank-based procedures (horizontal and vertical dashed lines represent timeouts). Blue data points are from random and red data points are from LTL. Axes are logarithmic.

Summary statistics are in the lower part of Table 6.3. RANKER has the second lowest mean (after ROLL) and the third lowest median (after ROLL and SEMINATOR 2). Columns **wins** and **losses** show the number of cases where RANKER outputs a strictly smaller or strictly bigger automaton, respectively. Observe that in comparison with all other tools, RANKER gives more wins than losses.

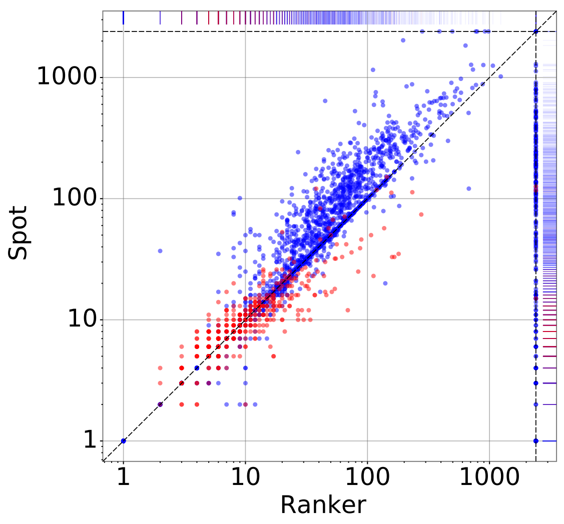
The number of timeouts of RANKER is still higher than of some other tools, especially SPOT, PITERMAN, and FRIBOURG.

6.4 Inherently Weak and Semi-Deterministic BAs

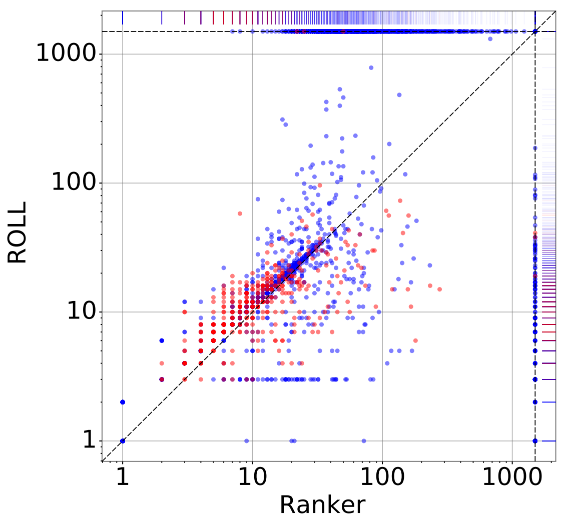
In this section, we present the results of our second set of experiments, with optimizations of the complementation of inherently weak and semi-deterministic Büchi automata, described in Sections 4.3 and 4.4. For this set of experiments, we denote the version of RANKER from [15], described in Section 6.3, as RANKER$_{\text{OLD}}$.

Table 6.1: Statistics for our experiments. The upper part compares various optimizations of the rank-based procedure (no postprocessing). The lower part compares RANKER to other approaches (with postprocessing). The left-hand side compares sizes of complement BAs and the right-hand side runtimes of the tools. The **wins** and **losses** columns give the number of times when RANKER was strictly better and worse. The values are given for the three datasets as "`all` (`random` : `LTL`)". Approaches in GOAL are labelled with ⚽.

method	mean	median	wins	losses	mean runtime [s]	median runtime [s]	timeouts
RANKER	3812 (4452 : 207)	79 (93 : 26)			7.83 (8.99 : 1.30)	0.51 (0.84 : 0.04)	279 (276 : 3)
RANKER$_{\text{OLD}}$	7398 (8688 : 358)	141 (197 : 29)	2190 (2011 : 179)	111 (107 : 4)	9.37 (10.73 : 1.99)	0.61 (1.04 : 0.04)	365 (360 : 5)
SCHEWE ⚽	4550 (5495 : 665)	439 (774 : 35)	2640 (2315 : 325)	55 (1 : 54)	21.05 (24.28 : 7.80)	6.57 (7.39 : 5.21)	937 (928 : 9)
RANKER	47 (52 : 18)	22 (27 : 10)			7.83 (8.99 : 1.30)	0.51 (0.84 : 0.04)	279 (276 : 3)
PITERMAN ⚽	73 (82 : 22)	28 (34 : 14)	1435 (1124 : 311)	416 (360 : 56)	7.29 (7.39 : 6.65)	5.99 (6.04 : 5.62)	14 (12 : 2)
SAFRA ⚽	83 (91 : 30)	29 (35 : 17)	1562 (1211 : 351)	387 (350 : 37)	14.11 (15.05 : 8.37)	6.71 (6.92 : 5.79)	172 (158 : 14)
SPOT	75 (85 : 15)	24 (32 : 10)	1087 (936 : 151)	683 (501 : 182)	0.86 (0.99 : 0.06)	0.02 (0.02 : 0.02)	13 (13 : 0)
FRIBOURG ⚽	91 (104 : 13)	23 (31 : 9)	1120 (1055 : 65)	601 (376 : 225)	17.79 (19.53 : 7.22)	9.25 (10.15 : 5.48)	81 (80 : 1)
LTL2DSTAR	73 (82 : 21)	28 (34 : 13)	1465 (1195 : 270)	465 (383 : 82)	3.31 (3.84 : 0.11)	0.04 (0.05 : 0.02)	136 (130 : 6)
SEMINATOR 2	79 (91 : 15)	21 (29 : 10)	1266 (1131 : 135)	571 (367 : 204)	9.51 (11.25 : 0.08)	0.22 (0.39 : 0.02)	363 (362 : 1)
ROLL	18 (19 : 14)	10 (9 : 11)	2116 (1858 : 258)	569 (443 : 126)	31.23 (37.85 : 7.28)	8.19 (12.23 : 2.74)	1109 (1106 : 3)



(a) RANKER vs SPOT



(b) RANKER vs ROLL

Figure 6.2: Comparison of the complement size obtained by RANKER and other state-of-the-art tools (horizontal and vertical dashed lines represent timeouts). Axes are logarithmic.

6.4.1 Datasets

For this set of experiments, we used automata from theh following three datasets: (i) random, (ii) LTL, and (iii) Automizer. The first two datasets are the same as in the first set of experiments from Section 6.3.1. Automizer contains 906 BAs over larger alphabets (up to 2^{35} symbols) used in [9], obtained from the ULTIMATE AUTOMIZER tool. We did not use the last benchmark in our previous experiments, because we focused mainly on hard automata and most of the automata form Automizer are semi-deterministic.

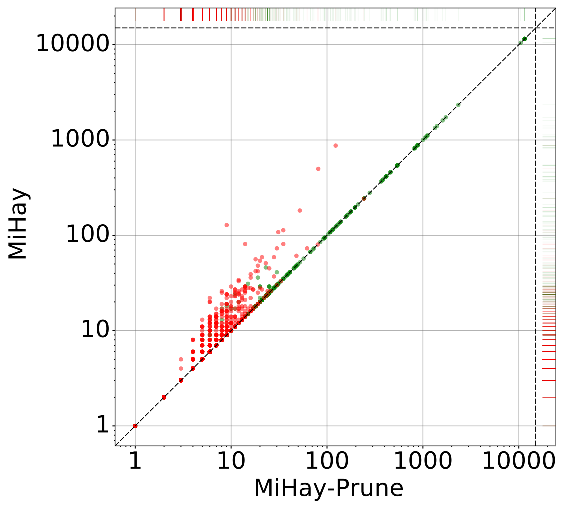
In contrast to the first set of experiments, where we removed some special types of automata from the dataset, in this experiment we removed only trivial one-state BAs, because RANKER contains a more efficient implementation for complementing these automata, especially inherently weak and semi-deterministic BAs. The final dataset contains 7,155 BAs (denoted as all) with 4,533 random, 1,716 LTL, and 906 Automizer automata.

6.4.2 Effect of the New Optimizations

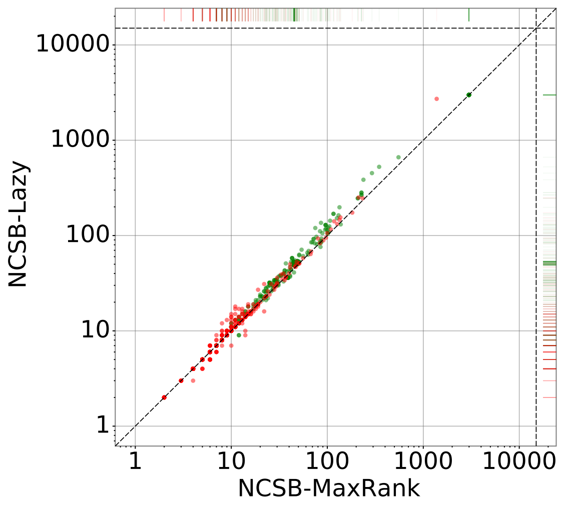
In the first experiment, we measured the effect of our optimizations for inherently weak and semi-deterministic Büchi automata without postprocessing. The evaluation was performed on LTL and Automizer benchmarks. We use both to denote their union. Most of the automata from these benchmarks are either inherently weak or semi-deterministic.

Table 6.2: Effects of our optimizations for IW and SDBA automata. Sizes of output BAs are given as "both (LTL : Automizer)".

method	mean				median			
MIHAY$_{pr}$	43.4	(7.3	:	140.7)	7	(5	:	21)
MIHAY	46.1	(10.9	:	141.3)	7	(6	:	23)
NCSB-MAXRANK	30	(20.3	:	38.3)	12	(8	:	28)
NCSB-LAZY	35.7	(25.1	:	44.8)	13	(9	:	32)



(a) MiHay$_{pr}$ vs MiHay



(b) NCSB-MaxRank vs NCSB-Lazy

Figure 6.3: Evaluation of the effect of our optimizations for IW and SDBA automata.

We first compared the number of states generated by MiHay and by the optimization MiHay$_{pr}$ from Section 4.3 on inherently weak BAs (1,308 BAs – 948 from LTL and 360 from `Automizer`). The scatter plot is shown in Figure 6.3a and summary statistics are in the upper part of Table 6.2. The optimization clearly reduces the number of states of the complement, especially for automata from LTL, and it decreases both the mean and the median.

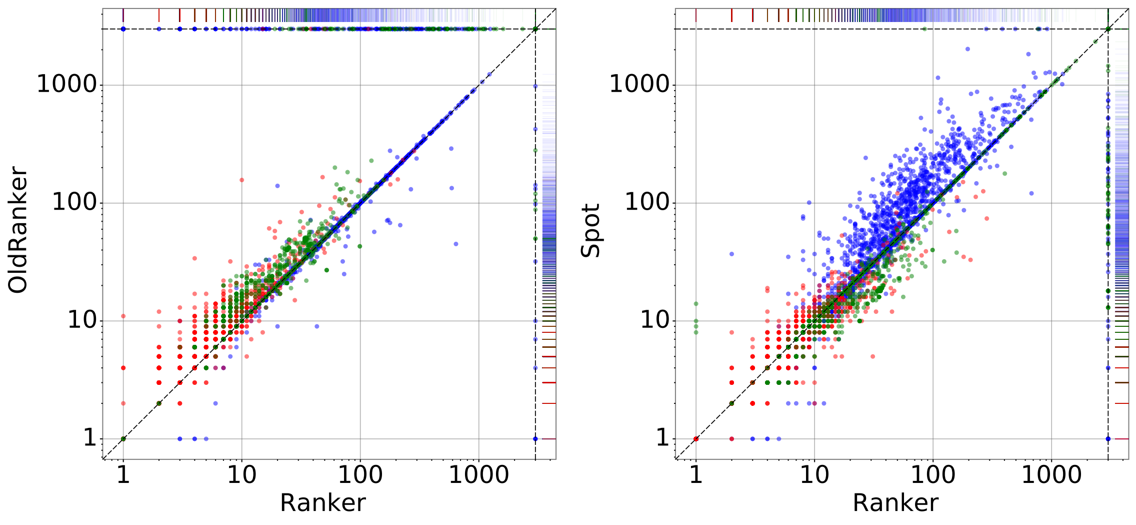
We also compared the number of states generated by NCSB-Lazy [9] and NCSB-MaxRank from Section 4.4 on semi-deterministic BAs that are not inherently weak (735 BAs – 328 from LTL and 407 from `Automizer`). The scatter plot is in Figure 6.3b and summary statistics are in the lower part of Table 6.2. Our optimization works better especially for big automata on the output. Both the mean and the median are lower for NCSB-MaxRank.

6.4.3 Comparison with Other Tools

In the second experiment, we compared Ranker with other state-of-the-art tools for Büchi automata complementation. We focused on the number of states of the output automata after postprocessing. Comparison of the number of states of automata generated by Ranker,

Table 6.3: Statistics for our experiments. The table compares the sizes of complement BAs obtained by Ranker and other approaches (after postprocessing). The **wins** and **losses** columns give the number of times when Ranker was strictly better and worse. The values are given for the three datasets as "`all` (`random` : LTL : `Automizer`)". Approaches in Goal are labelled with ⚽.

method	mean	median	wins	losses	timeouts
Ranker	38 (44 : 9 : 67)	11 (18 : 5 : 22)			158 (53 : 0 : 105)
Ranker$_{Old}$	30 (38 : 10 : 32)	12 (18 : 6 : 22)	1554 (356 : 650 : 548)	264 (142 : 69 : 53)	458 (259 : 7 : 192)
Piterman ⚽	43 (56 : 12 : 38)	14 (19 : 8 : 24)	2881 (1279 : 966 : 636)	392 (263 : 68 : 61)	309 (12 : 4 : 293)
Safra ⚽	49 (60 : 17 : 56)	15 (18 : 10 : 24)	3109 (1348 : 1117 : 644)	274 (229 : 31 : 14)	599 (160 : 30 : 409)
Spot	46 (57 : 8 : 66)	11 (18 : 5 : 18)	1347 (935 : 339 : 73)	1057 (327 : 343 : 387)	73 (13 : 0 : 60)
Fribourg ⚽	49 (68 : 8 : 27)	11 (18 : 6 : 19)	2223 (1177 : 503 : 543)	586 (245 : 207 : 134)	399 (93 : 2 : 304)
LTL2dstar	44 (56 : 12 : 47)	14 (19 : 7 : 24)	2794 (1297 : 924 : 573)	448 (283 : 88 : 77)	288 (130 : 13 : 145)
Seminator 2	46 (58 : 8 : 64)	11 (17 : 5 : 21)	1626 (1297 : 291 : 38)	1113 (286 : 398 : 429)	419 (368 : 1 : 50)
Roll	18 (15 : 11 : 54)	9 (8 : 8 : 28)	6050 (3824 : 1551 : 675)	620 (369 : 125 : 126)	1893 (1595 : 8 : 290)



(a) Ranker vs Ranker$_{\text{Old}}$ (b) Ranker vs Spot

Figure 6.4: Comparison of the complement size obtained by Ranker, Ranker$_{\text{Old}}$, and Spot (horizontal and vertical dashed lines represent timeouts).

Ranker$_{\text{Old}}$, and Spot are given in Figure 6.4. Summarizing statistics are in Table 6.3. The backoff strategy was applied in 278 cases (264 for `random`, 1 for `LTL`, and 13 for `Automizer`).

The number of timeouts was reduced by 65% w.r.t. Ranker$_{\text{Old}}$ – this is the reason of the higher mean. Ranker also has the third smallest mean and median, after Roll and Ranker$_{\text{Old}}$, but they have a much higher number of timeouts. From the columns **wins** and **losses** we can see that Ranker gives a strictly smaller automaton in the majority of cases compared to all other tools.

Regarding runtimes, we can see from Table 6.4 that Ranker is comparable to Seminator 2. Spot still remains the fastest tool for BA complementation.

Table 6.4: Run times of the tools given as "`all` (`random` : `LTL` : `Automizer`)"

method	**mean**								**median**							
Ranker	3.72	(4.34	:	0.45	:	7.30)			0.05	(0.10	:	0.04	:	0.08)		
Ranker$_{\text{Old}}$	4.62	(5.33	:	0.72	:	9.69)			0.07	(0.19	:	0.03	:	0.15)		
Piterman ⚽	8.06	(6.07	:	5.95	:	28.38)			5.12	(4.96	:	5.08	:	8.68)		
Safra ⚽	11.58	(10.41	:	6.51	:	38.65)			5.41	(5.32	:	5.26	:	9.02)		
Spot	0.64	(0.57	:	0.02	:	2.28)			0.02	(0.02	:	0.01	:	0.02)		
Fribourg ⚽	13.13	(14.14	:	6.06	:	23.88)			5.69	(6.82	:	4.92	:	6.57)		
LTL2dstar	2.1	(2.25	:	0.34	:	5.15)			0.02	(0.02	:	0.01	:	0.05)		
Seminator 2	4.16	(6.33	:	0.03	:	1.88)			0.03	(0.08	:	0.01	:	0.03)		
Roll	23.65	(29.82	:	3.88	:	49.02)			3.34	(6.19	:	1.71	:	17.14)		

Chapter 7

Conclusion

In this thesis, we presented several optimizations for efficient complementation of Büchi automata. Firstly, we focused on rank-based complementation. We identified the main source of a state space blow-up, which is often an unnecessarily high bound on maximum rank for each state, and observed that if an automaton has a specific structure, we can reduce the rank bound for states in each strongly connected component of the automaton. We identified a subclass of Büchi automata, called elevator automata, whose structure enables to reduce the rank bound. We presented an algorithm assigning rank bound for states in each strongly connected component of the automaton. This algorithm can also to a certain degree be extended to general Büchi automata with no specific structure. Then we presented a technique based on data flow analysis that enables propagation of rank restrictions throughout the automaton. Moreover, we showed that elevator automata can be complemented in $\mathcal{O}(16^n)$ space. The definition of elevator automata has already been used by other members of the research community [22].

In the second part of the thesis, we focused on optimizations of specialized complementation constructions for inherently weak and semi-deterministic Büchi automata. Due to special properties of these types of BAs, we can use more efficient algorithms than the rank-based construction. We presented an optimization removing states from a macrostate of the complement of an inherently weak automaton based on direct simulation. The optimization of semi-deterministic BA complementation was inspired by an optimization of rank-based construction.

All techniques presented in this thesis were implemented as an extension of the tool Ranker for complementation of Büchi automata. We performed a thorough experimental evaluation on thousand of hard Büchi automata occuring in practice, as well as randomly generated automata. Our optimizations significantly reduced the generated state space compared to the previous version of Ranker. The algorithm for efficient complementation of elevator automata caused an exponential improvement in a lot of cases. We also compared the results with other state-of-the-art tools for Büchi automata complementation. Ranker produces a smaller automaton than any other tool in the majority of cases. Even though the original rank-based construction may be quite inefficient, with the optimizations presented in this paper, together with some previous optimizations implemented in Ranker, we get a tool that is competitive with other state-of-the-art tools and that can in the majority of cases even produce smaller automata than any other tool.

The ideas presented in this thesis are a part of my research on which we worked together with my supervisor and my consultant. My own contribution is especially the extension of the definition of elevator automata (originally without inherently weak accepting components), some optimizations, the implementation of the algorithm assigning rank bounds to states of each strongly connected component, and the implementation of complementation algorithms for inherently weak and semi-deterministic Büchi automata, as well as their optimizations. The first part of the thesis, in particular the algorithms for elevator automata and data flow analysis, is a part of a paper published at TACAS'22 [15], and the second part, including the implementation of RANKER, is a part of a tool paper at the time of writing conditionally accepted at CAV'22.

7.1 Future Work

We plan to extend the rank-based complementation algorithm and some of the optimizations to (transition-based) *Emerson-Lei automata* (TELA) – ω-automata with a richer acceptance condition than Büchi automata. Due to the richer acceptance condition, TELAs enable a more compact representation than BAs.

The next subject of our future work is a decomposition-based Büchi automata complementation. The idea is that we keep information about each strongly connected component of the automaton separately and using a round-robin strategy, we inspect the runs in only one component at the same time. This strategy should reduce the degree of nondeterminism and thus reduce the number of generated states of the complement.

Bibliography

[1] Abdulla, P. A., Chen, Y., Clemente, L., Holík, L., Hong, C.-D. et al. Simulation Subsumption in Ramsey-based Büchi Automata Universality and Inclusion Testing. In: Springer. *Proc. of CAV'10.* 2010, p. 132–147.

[2] Allred, J. D. and Ultes Nitsche, U. A Simple and Optimal Complementation Algorithm for Büchi Automata. In: *Proceedings of the Thirty third Annual IEEE Symposium on Logic in Computer Science (LICS 2018).* IEEE Computer Society Press, July 2018, p. 46–55.

[3] Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J. et al. The Hanoi Omega-Automata Format. In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I.* Springer, 2015, vol. 9206, p. 479–486. Lecture Notes in Computer Science. DOI: 10.1007/978-3-319-21690-4_31. Available at: `https://doi.org/10.1007/978-3-319-21690-4_31`.

[4] Blahoudek, F., Duret Lutz, A. and Strejček, J. Seminator 2 Can Complement Generalized Büchi Automata via Improved Semi-Determinization. In: *Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV'20).* Springer, July 2020, vol. 12225, p. 15–27. Lecture Notes in Computer Science.

[5] Blahoudek, F., Heizmann, M., Schewe, S., Strejček, J. and Tsai, M. Complementing Semi-deterministic Büchi Automata. In: *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings.* Springer, 2016, vol. 9636, p. 770–787. Lecture Notes in Computer Science. DOI: 10.1007/978-3-662-49674-9_49. Available at: `https://doi.org/10.1007/978-3-662-49674-9_49`.

[6] Boigelot, B., Jodogne, S. and Wolper, P. On the Use of Weak Automata for Deciding Linear Arithmetic with Integer and Real Variables. In: *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings.* Springer, 2001, vol. 2083, p. 611–625. Lecture Notes in Computer Science. DOI: 10.1007/3-540-45744-5_50. Available at: `https://doi.org/10.1007/3-540-45744-5_50`.

[7] Breuers, S., Löding, C. and Olschewski, J. Improved Ramsey-Based Büchi Complementation. In: *Proc. of FOSSACS'12.* Springer, 2012, p. 150–164.

[8] Büchi, J. R. On a Decision Method in Restricted Second Order Arithmetic. In: *Proc. of International Congress on Logic, Method, and Philosophy of Science 1960.* Stanford Univ. Press, Stanford, 1962.

[9] Chen, Y., Heizmann, M., Lengál, O., Li, Y., Tsai, M. et al. Advanced automata-based algorithms for program termination checking. In: *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018.* ACM, 2018, p. 135–150. DOI: 10.1145/3192366.3192405. Available at: `https://doi.org/10.1145/3192366.3192405`.

[10] Duret Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É. et al. Spot 2.0 — A Framework for LTL and ω-Automata Manipulation. In: *Automated Technology for Verification and Analysis.* Cham: Springer International Publishing, 2016, p. 122–129. ISBN 978-3-319-46520-3.

[11] Fogarty, S. and Vardi, M. Y. Büchi complementation and size-change termination. In: Springer. *Proc. of TACAS'09.* 2009, p. 16–30.

[12] Friedgut, E., Kupferman, O. and Vardi, M. Büchi Complementation Made Tighter. *International Journal of Foundations of Computer Science.* 2006, vol. 17, p. 851–868.

[13] Glabbeek, R. and Ploeger, B. Five Determinisation Algorithms. In: *Proc. of CIAA'08.* Springer, 2008, p. 161–170. ISBN 978-3-540-70843-8.

[14] Havlena, V., Lengál, O. and Šmahlíková, B. *Ranker.* 2021. `https://github.com/vhavlena/ranker`.

[15] Havlena, V., Lengál, O. and Šmahlíková, B. Sky Is Not the Limit: Tighter Rank Bounds for Elevator Automata in Buchi Automata Complementation. In: *Proceedings of TACAS'22.* Springer Verlag, 2022. ISSN 0302-9743.

[16] Havlena, V. and Lengál, O. Reducing (To) the Ranks: Efficient Rank-Based Büchi Automata Complementation. In: *32nd International Conference on Concurrency Theory (CONCUR 2021).* Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, vol. 203, p. 2:1–2:19. Leibniz International Proceedings in Informatics (LIPIcs). DOI: 10.4230/LIPIcs.CONCUR.2021.2. ISBN 978-3-95977-203-7. ISSN: 1868-8969. Available at: `https://drops.dagstuhl.de/opus/volltexte/2021/14379`.

[17] Heizmann, M., Hoenicke, J. and Podelski, A. Termination Analysis by Learning Terminating Programs. In: Springer. *Proc. of CAV'14.* 2014, p. 797–813.

[18] Kähler, D. and Wilke, T. Complementation, Disambiguation, and Determinization of Büchi Automata Unified. In: Springer. *Proc. of ICALP'08.* 2008, p. 724–735.

[19] Klein, J. and Baier, C. On-the-Fly Stuttering in the Construction of Deterministic *omega* -Automata. In: *Implementation and Application of Automata, 12th International Conference, CIAA 2007, Prague, Czech Republic, July 16-18, 2007, Revised Selected Papers.* Springer, 2007, vol. 4783, p. 51–61. Lecture Notes in Computer Science. DOI: 10.1007/978-3-540-76336-9_7. Available at: `https://doi.org/10.1007/978-3-540-76336-9_7`.

[20] Kupferman, O. and Vardi, M. Y. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.* 2001, vol. 2, no. 3, p. 408–429. DOI: 10.1145/377978.377993. Available at: `https://doi.org/10.1145/377978.377993`.

[21] Li, Y., Sun, X., Turrini, A., Chen, Y. and Xu, J. ROLL 1.0: ω-Regular Language Learning Library. In: *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I.* Springer, 2019, vol. 11427, p. 365–371. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-17462-0_23. Available at: `https://doi.org/10.1007/978-3-030-17462-0_23`.

[22] Li, Y., Turrini, A., Feng, W., Vardi, M. and Zhang, L. Divide-and-Conquer Determinization of Büchi Automata based on SCC Decomposition. In: *Proc. of CAV'22.* 2022. To appear.

[23] Li, Y., Vardi, M. Y. and Zhang, L. On the Power of Unambiguity in Büchi Complementation. In: Proceedings 11th International Symposium on *Games, Automata, Logics, and Formal Verification,* Brussels, Belgium, September 21-22, 2020. Open Publishing Association, 2020, vol. 326, p. 182–198. Electronic Proceedings in Theoretical Computer Science. DOI: 10.4204/EPTCS.326.12.

[24] Mayr, R. and Clemente, L. Advanced automata minimization. In: *Proc. of POPL'13.* 2013, p. 63–74.

[25] Mayr, R. and Clemente, L. Efficient Reduction of Nondeterministic Automata with Application to Language Inclusion Testing. *Logical Methods in Computer Science.* Episciences. org. 2019, vol. 15.

[26] Miyano, S. and Hayashi, T. Alternating finite automata on ω-words. *Theoretical Computer Science.* 1984, vol. 32, no. 3, p. 321–330. DOI: https://doi.org/10.1016/0304-3975(84)90049-5. ISSN 0304-3975. Available at: `https://www.sciencedirect.com/science/article/pii/0304397584900495`.

[27] Nielson, F., Nielson, H. R. and Hankin, C. *Principles of program analysis.* Springer, 1999. ISBN 978-3-540-65410-0. Available at: `https://doi.org/10.1007/978-3-662-03811-6`.

[28] Piterman, N. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. In: IEEE. *Proc. of LICS'06.* 2006, p. 255–264.

[29] Ramsey, F. P. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society.* 1930, no. 1, p. 264–286. DOI: 10.1112/plms/s2-30.1.264. ISSN 0024-6115. Available at: `https://doi.org/10.1112/plms/s2-30.1.264`.

[30] Redziejowski, R. R. An Improved Construction of Deterministic Omega-automaton Using Derivatives. *Fundam. Informaticae.* 2012, vol. 119, 3-4, p. 393–406. DOI: 10.3233/FI-2012-744. Available at: `https://doi.org/10.3233/FI-2012-744`.

[31] Safra, S. On the Complexity of ω-automata. In: IEEE. *Proc. of FOCS'88.* 1988, p. 319–327.

[32] Schewe, S. Büchi Complementation Made Tight. In: *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings.* Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009, vol. 3, p. 661–672. LIPIcs. DOI: 10.4230/LIPIcs.STACS.2009.1854. Available at: `https://doi.org/10.4230/LIPIcs.STACS.2009.1854`.

[33] Sistla, A. P., Vardi, M. Y. and Wolper, P. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science.* Elsevier. 1987, vol. 49, 2-3, p. 217–237.

[34] Tabakov, D. and Vardi, M. Y. Experimental Evaluation of Classical Automata Constructions. In: *Proc. of LPAR'05.* Springer, 2005, p. 396–411. ISBN 978-3-540-31650-3.

[35] Tsai, M., Fogarty, S., Vardi, M. Y. and Tsay, Y. State of Büchi Complementation. In: *Implementation and Application of Automata.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, p. 261–271. ISBN 978-3-642-18098-9.

[36] Tsai, M., Tsay, Y. and Hwang, Y. GOAL for Games, Omega-Automata, and Logics. In: *Computer Aided Verification.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, p. 883–889. ISBN 978-3-642-39799-8.

[37] Vardi, M. Y. and Wolper, P. An Automata-Theoretic Approach to Automatic Program Verification. In: IEEE. *Proceedings of the First Symposium on Logic in Computer Science.* 1986, p. 322–331.

[38] Yan, Q. Lower Bounds for Complementation of ω-Automata Via the Full Automata Technique. In: *Automata, Languages and Programming.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, p. 589–600. ISBN 978-3-540-35908-1.