- Explanation of Büchi's first complementation construction (or Sistla, Vardi, Wolper's, 1987)
- Proposition of optmisations for this construction
- Implementation of the construction
- Generation of random test automata in a similar way as Tsai, 2010
- Comparison of algorithm with and without optimisations
  - Identification of easy, medium, and hard complementation tasks, based on transition and acceptance densities
  - Filtering out of the trivial cases of universal automata
- Comparison of these results to the results in Tsai, 2010

Bachelor Thesis on the topic

Optimization and experimental analysis of the algebraic complementation construction for Büchi automata

Stefan Breuers
Student Id.: 280057

RWTH Aachen - October 27, 2010
Chair of Computer Science 7 - Logic and Theory of Discrete Systems
Prof. Dr. Dr.h.c. Wolfgang Thomas

Reviewers: Prof. Dr. Dr.h.c. Wolfgang Thomas, Dipl.-Inform. Christof Löding

Abstract

In this bachelor thesis we look at the well-discussed but still present and important problem of the complementation of Büchi automata. The discussion reaches back to the 60s and step-by-step the upper bound of states in a complement automaton shrank by many constructions. There exist three major approaches, via deterministic ω-automata, ranking functions and the algebraic approach, the latter being the one J.R. Büchi based his proof on, though getting a double-exponential state-blowup. Many optimizations have been given for the other two, but as the algebraic complementation leads to the worst asymptotic bound, they have been rather few here. So in this work, we redevelop this construction, presenting some optimizations and discuss them in an experimental analysis, leading to the result that in practice the algebraic approach is not that far behind the other ones as it actually leads to a comparable number of states.

Zusammenfassung

In dieser Bachelor-Thesis schauen wir auf das schon lange diskutierte aber immer noch präsente und wichtige Problem der Komplementierung von Büchi Automaten. Die Diskussion reicht bis in die 60er Jahre zurück, wobei die obere Grenze der Zustandszahl für einen Komplementautomaten durch viele Verfahren immer weiter verbessert wurde. Es gibt drei Hauptansätze, via deterministischen ω-Automaten, ranking fuctions und den algebraischen Ansatz, von denen zuletzt genannter derjenige ist, auf den J.R. Büchi seinen Beweis aufgebaut hat, jedoch mit einer zweifach-exponentiellen Zustandsexplosion. Für die anderen beiden wurden viele Optimierungen dargelegt, aber da die algebraische Komplementierung die schlechteste asymptotische Schranke aufweist, waren es hier eher wenige. In dieser Arbeit greifen wir also diese Konstruktion wieder auf, präsentieren einige Optimierungen und besprechen sie in einer experimentellen Analyse. Sie führt uns zum Ergebnis, dass in der Praxis der algebraische Ansatz nicht ganz so weit hinter den anderen zurückliegt, da er tatsächlich vergleichbare Zustandszahlen erzielt.

Erklärung

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 27.10.2010 Stefan Breuers

Contents

1 Introduction

When J.R. Büchi published his work of finite automata on infinite words in 1962 [Büc62] and showed the equivalence of this model and monadic second order logic (S1S) the discussion of the complementation problem of these Büchi automata begun. Büchi himself presented that Büchi automata (in the following denoted as BA) are closed under complement, but his proof via Ramsey´s Theorem leads to a double exponential state blow-up. So it was desirable to find optimizations or other strategies and throughout the centuries the boundaries for the state space of a complement automaton for a given BA has been decreased permanently.

The application area of the complementation problem is settled in formal verification. Checking if a (non-terminating) system fulfills a certain specification, formally means testing if there is any run in the system that does not hold to this specification. This problem, known as model-checking, leads to the test if the intersection of the system model and the complement of the specification model is empty. As the formulation of systems and specifications are more intuitively with BAs than logic formula, it is motivated to find an optimal complementation strategy for these automata. There also exist further applications, like verifying translation algorithms of linear temporal logic (LTL) without a reference algorithm [GKSV03]. In [TFTV10] it is pointed out that many works avoid complementation of BAs, when testing universality or also containment [DR09, FV09, FV10] but in special cases it is still mandatory [KV05].

A short side glance to finite automata on finite words already gives us a lower bound of $\Omega(2^n)$ induced by the powerset construction. To complement a given nondeterministic finite automaton one just uses this construction to get a corresponding deterministic automaton and switch the set of accepting and non-accepting states [RS59]. This is not sufficient in the case of BAs (or other so called ω-automata), not least because there are nondeterministic BAs for which no equivalent deterministic BA exists. The lower bounds for complementation of BA were even sharpened by Michel in 1988 to $\Omega((0.36n)^n)$ [Mic88] and by Yan in 2008 to $\Omega((0.76n)^n)$ [Yan08].

To further get on to the complementation of a BA you can differentiate between three major approaches that have been developed and optimized in the past 5 decades, namely over determinization [Saf88, MS95, Pit07, ATW06], rankings [Kla91, KV01, FKV06, KW08, VW07, Sch09, Tho99] and the algebraic approach [Büc62, SVW87]. Starting with the procedure last-mentioned, this is actually the one Büchi 1962 based his proof on and which produces the $2^{2^{\mathcal{O}(n)}}$ state space-explosion [Büc62]. Sistla, Vardi and Wolper developed the construction further in 1987 reaching a bound of $2^{\mathcal{O}(n^2)}$ [SVW87]. One year later Safra came up with an alternative approach using deterministic Rabin automata [Saf88]. Leading to an upper bound of $2^{\mathcal{O}(n \log n)}$ this construction matched with Michels lower bound of the same year [Mic88], providing BA complementation to be in $n^{\Theta(n)}$. It was later pointed out by Vardi [Var07] that the $\mathcal{O}$-notation still hides an exponential gap between the two boundaries. Independently, 1991 Klarlund provides progress measures to complement BAs [Kla91] which form the foundation of the third approach, the ranking functions. Kupferman and Vardi took this idea up in 2001 by using the so called level rankings to come down to a $\mathcal{O}((6n)^n)$ bound [KV01], further improved by Friedgut, Kupferman and Vardi in 2007 with "'tight level rankings"' and an $\mathcal{O}((0.97n)^n)$ bound [FKV06]. One year later, Kähler and Wilke provided a slightly different construction in 2008 [KW08] (based on preliminary version [VW07]), but also using rankings to get a $\mathcal{O}((3n)^n)$ blow-up. The latest step in this approach was performed by Schewe 2009, who presents an upper bound of $\mathcal{O}(n(0.76n)^n)$ [Sch09] with his optimization and as this meets the known lower bound of Yan [Yan08] he claims that "[this] concludes the quest for optimal Büchi complementation algorithms". Optimizations concerning the other approaches have yet been held back. 1995 Muller and Schupp continued with Safras approach over deterministic ω-automata [MS95] and 2007 Piterman advanced this using deterministic parity automata to get an upper bound of $\mathcal{O}(n^{2n})$ [Pit07]. As the algebraic construction has the worst asymptotic behaviour, optimization techniques have been more than rare here since 1988. Nevertheless, this is exactly where this thesis wants to step in. The latest work on the state of Büchi complementation [TFTV10] showed, the Piterman-construction trumps Schewe´s approach in an empirical study, although it has a worse asymptotic boundary. The algebraic approach "performs rather poorly" as it was not able to finish any given complementation task in time. In spite of or maybe exactly because of this fact, we want to analyze in this work what we can achieve through optimizations and heuristics for the algebraic approach and perform a first experimental analysis.

In order to make this possible, we first present the basic definitions in Section 2, mainly transition profiles which describe the equivalence classes of a congruence $\sim_{\mathfrak{A}}$ on the state set of a BA $\mathfrak{A}$. We will see that these equivalence classes are also

regular by building the simultaneous powerset automaton (SPA) that recognizes the corresponding languages. With Ramsey´s Theorem, which we introduce in Section 3, it is then pointed out that a word $w \in \Sigma^\omega$ is decomposable in two parts $w \in U \cdot V^\omega$, where U and V are equivalence classes of our congruence $\sim_{\mathfrak{A}}$. After finding adequate models for both components —powerset automaton PA for U, simultaneous powerset automata SPAs for V— we form pairs of states in PA and certain SPAs. Since PA covers the reachable states of all runs a word w can have on $\mathfrak{A}$ we are then able to examine if from these states we can have an accepting infinite run. To distinguish between accepting and non-accepting runs we define the mentioned pairs as safe and dangerous in order to build our complement automaton out of a subset from the safe pairs, as well as additional transitions to link both components and such for repeating the V-part properly. Several optimization techniques are presented in Section 4. As the SPAs are deterministic finite automata, it is possible to minimize them. This also holds for the PA, but here we have to ensure that the states are equivalent in consideration of the whole complement language. A further reduction of the state space can be reached by merging SPAs, if this does not lead to any new accepting runs, which can be tested by looking at the union of transition profiles. Transitions that are added to repeat the language of each SPA sometimes give rise to redundant states, too, and by checking the need of these we can spare them. Those four optimizations lead us to an optimized algorithm whose function and performance is analyzed in Section 5. After some case studies, we construct random automata and provide three levels of difficulty, according to their transition and acceptance density. Examining the different optimizations, we see that merging and especially minimizing SPAs result in great saving, whereas the other two just yield a little decrease of states. It turns out that on the one hand our optimized algorithm leads to satisfying results, but on the other hand still shows weaknesses regarding the running time for large inputs. Nevertheless, the state spaces are even catching up with the recent constructions, although further direct comparisons have to be made.

Acknowledgments

I would like to thank Jörg Olschewski for implementing and inducting me to the framework of the original complementation algorithm. Also, I give thanks to my supervisor Christof Löding for being weekly at hand with helpful advice and a remarkable patience in the course of this thesis.

2 Büchi automata

2.1 Automata models

As there exist many different notations in the area of words, languages and automata it is necessary to also name here the basic definitions that are used in this work.

Definition 2.1 *An* alphabet, *generally denoted as* Σ, *is a finite set of* symbols $a_1, \ldots, a_n$, *where* words $u, v, w, \ldots$ *are possibly infinite sequences of these symbols.*

Just like a finite word is denoted as $w \in \Sigma^*$, you can look at an inifinite word $w \in \Sigma^\omega$. $|w|$ refers to the length of a finite word and $w(i)$ to the i-th position of the sequence ($i \in \mathbb{N}$).

Definition 2.2 *A* language $L \subseteq \Sigma^*$ *is a possibly infinite set of finite words over an alphabet* Σ. *Languages* $L \subseteq \Sigma^\omega$ *of infinite words are also called* ω*-languages.*

Languages of finite words can be described by regular expressions (build over $+, \cdot, ^*$) and by finite automata, which are defined in the usual way.

Definition 2.3 *A* finite automaton *is a five tuple,* $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$, *consisting of*

- *finite state set* Q
- *input alphabet* Σ
- *initial state* $q_0 \in Q$
- *transition relation* $\Delta \subseteq Q \times \Sigma \times Q$

- *accepting state set* $F \subseteq Q$

This defines a nondeterministic finite automaton (NFA). If Δ *is a function* $Q \times \Sigma \to Q$*, we then denote it as* δ *and speak of a deterministic finite automaton (DFA).*

For a word $w \in \Sigma^*, w = w(0) \dots w(n)$ *we define a* run $\rho \in Q^*, \rho = \rho(0) \dots \rho(n)$ *of* $\mathfrak{A}$ *on* w *as a sequence of states that are visited when going by the corresponding transitions in* $\mathfrak{A}$ *reading the symbols of* w*. So* $\rho(0) = q_0, (\rho(i), w(i), \rho(i+1)) \in \Delta$ *(adjusted appropriately for a DFA). Of course an NFA can have several runs on the same word.*

A word $w \in \Sigma^*$ *is* accepted *by* $\mathfrak{A}$ *iff there exists a run* $\rho \in Q^*$ *of* $\mathfrak{A}$ *on* w *ending up in an accepting state, i.e.* $\rho(n) \in F$*. In the deterministic case the unique run has to fulfill this property. The language that is recognized by* $\mathfrak{A}$ *is* $L(\mathfrak{A}) = \{w \in \Sigma \mid \mathfrak{A}$ *accepts* $w\}$*.*

In order to describe ω-languages, we can make use of automata, too. There are many different models for recognizing several classes of ω-languages, all with different power. The one we concentrate on is the already mentioned automaton of J.R. Büchi [Büc62].

Definition 2.4 *A* nondeterministic Büchi automaton (NBA) $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ *is defined like a NFA offering an adequate accepting condition for infinite words.*

For a word $w \in \Sigma^\omega, w = w(0)w(1) \dots$ *we now have infinite* runs $\rho \in Q^\omega, \rho = \rho(0)\rho(1) \dots$ *of* $\mathfrak{A}$ *on* w*, so infinite sequences of states that depend on the symbols of* w *and the corresponding transitions in* Δ*.*

A word $w \in \Sigma^\omega$ *is accepted by* $\mathfrak{A}$ *iff there exists a run* $\rho \in Q^\omega$ *of* $\mathfrak{A}$ *on* w *where at least one state of* F *is visited infinitely often, i.e.* $\exists q \in F \exists^\omega i \rho(i) = q$*. The language that is recognized by* $\mathfrak{A}$ *is* $L(\mathfrak{A}) = \{w \in \Sigma^\omega \mid \mathfrak{A}$ *accepts* $w\}$*.*

This model recognizes all regular ω-languages, which also can be described with regular expressions (build over $+, \cdot, ^*, ^\omega$).

Definition 2.5 *An* ω*-language* $L \subseteq \Sigma^\omega$ *is called* regular*, if it can be written as* $\bigcup_{i=1}^{n} U_i \cdot V_i^\omega$*, with regular* $U_i, V_i \subseteq \Sigma^*$*.*

2.2 Equivalence relation $\sim_{\mathfrak{A}}$

Talking about words and their respective runs on an automaton, accepting or not, it is helpful to introduce an essential equivalence relation $\sim_{\mathfrak{A}}$ on the state set of an automaton.

Definition 2.6 $p \xrightarrow{w} q$ *iff* $\exists$ *run from* p *to* q *with* w *in* $\mathfrak{A}$.
$p \xrightarrow{w}\!\!\!\!\circ\!\!\rightarrow q$ *iff* $\exists$ *run from* p *to* q *with* w *in* $\mathfrak{A}$ *with a visit of a final state.*

Define equivalence relation $\sim_{\mathfrak{A}}$ *on* Q *with:*
$u \sim_{\mathfrak{A}} v \Leftrightarrow \forall p, q \in Q : p \xrightarrow{u} q \Leftrightarrow p \xrightarrow{v} q$ *and* $p \xrightarrow{u}\!\!\!\!\circ\!\!\rightarrow q \Leftrightarrow p \xrightarrow{v}\!\!\!\!\circ\!\!\rightarrow q$.

So $u \sim_{\mathfrak{A}} v$ means that starting from a certain state in $\mathfrak{A}$ you can reach exactly the same states by reading u and v. It shall be stated explicitly that $p \xrightarrow{w}\!\!\!\!\circ\!\!\rightarrow q$ automatically induces $p \xrightarrow{w} q$. We leave it to the reader to show that $\sim_{\mathfrak{A}}$ is an equivalence relation (reflexive, symmetric, transitive). We can also show that $\sim_{\mathfrak{A}}$ is congruent, of finite index and each equivalence class is regular. The equivalence classes of $\sim_{\mathfrak{A}}$ are described by the transition profiles, introduced below.

2.2.1 Transition Profiles

Definition 2.7 *Given an automaton* $\mathfrak{A}$ *and a word* u, *the* transition profile $\tau(u)$ *is a subset of* $Q \times \{0,1\} \times Q$, *containing*

- $(p, 0, q)$ *iff* $p \xrightarrow{u} q$, $p, q \in Q$
- $(p, 1, q)$ *iff* $p \xrightarrow{u}\!\!\!\!\circ\!\!\rightarrow q$, $p, q \in Q$

With $[\tau(u)] = \{v \in \Sigma^* \mid \tau(v) = \tau(u)\}$ *we call* u *a representative of* $\tau(u)$.

Two small special cases are formed by the empty transition profile $\tau(\varepsilon)$, having only self-loops with regard to final states, and the empty one which does not contain any element at all. In the following we use the arrow-notation, as it is more intuitive and convenient. Furthermore if the word u does not matter or is clear in a context we simply write τ instead of $\tau(u)$ and also omit the u over the arrows. As mentioned these transition profiles represent the equivalence classes of $\sim_{\mathfrak{A}}$, meaning that if two words u, v have the same transition profile, we reach

exactly the same states in $\mathfrak{A}$ reading u and v starting in any state. Because $\sim_{\mathfrak{A}}$ is of finite index, there are only finitely many transition profiles. Here we may not orientate ourselves by the infinite number of possible representatives (as Σ^* is infinite) but on the finite number of transition profile types, as there are $|Q|^2$ possible pairs, which leads us to $\mathcal{O}(2^{2|Q|^2}) = \mathcal{O}(4^{|Q|^2})$ equivalence classes.

Remark 2.8 $\tau(uv)$ can be computed from $\tau(u)$ and $\tau(v)$.

This is not hard to see, obtaining all the pairs $p \xrightarrow{uv} q$ by checking if there is a state r with $p \longrightarrow r$ in $\tau(u)$ and $r \longrightarrow$ in $\tau(v)$, respectively for $p \overset{uv}{\multimap} q$ where at least one pair in $\tau(u)$ and $\tau(v)$ has to go by final state. This also indicates that $\sim_{\mathfrak{A}}$ is a congruence.

A special group is formed by the idempotent transition profiles.

Definition 2.9 *A transition profile $\tau(u)$ is called* idempotent, *if $u \sim_{\mathfrak{A}} uu$, that is when $\tau(u) = \tau(uu)$. We call the set of all idempotent transition profiles ITP.*

As ε^ω is not defined, $\tau(\varepsilon)$ is only idempotent if another transition profile is equal to it and therefore also has a representative $u \neq \varepsilon$. The empty transition profile is always idempotent, since there are no elements in it, the concatenation of two representatives again leads to no elements. In the following two properties of idempotent transition profiles are given. Although being rather simple, they will help us to better understand our purpose of using them in our complement automaton.

Lemma 2.10 *In an idempotent transition profile τ with $p \longrightarrow q$ and $q \longrightarrow r$ in τ it holds that $p \longrightarrow r$ is also in τ.*

Proof. The left hand side leads to a run in the corresponding automaton $\mathfrak{A}$ of form $p \xrightarrow{[\tau]} q \xrightarrow{[\tau]} r$ and therefore $p \xrightarrow{[\tau][\tau]} r$. By the definition of an idempotent transition profile this means that there is also a run $p \xrightarrow{[\tau]} r$ in $\mathfrak{A}$, so $p \longrightarrow r$ is in τ. ■

Lemma 2.11 *In an idempotent transition profile τ with $p \longrightarrow q$ in τ it holds that there is r with $p \longrightarrow r$, $r \rightarrow r$ and $r \longrightarrow q$ in τ (of course $r = p$ or $r = q$ is possible).*

Proof. The proof is based on a simple pumping argument. With τ being idempotent, the run $p \xrightarrow{[\tau]} q$ on $\mathfrak{A}$ also provides the run $p \xrightarrow{[\tau]^n} q, \forall n \in \mathbb{N}$. This means

there is a run of form $p \xrightarrow{[\tau]} r_1 \xrightarrow{[\tau]} \ldots \xrightarrow{[\tau]} r_{n-1} \xrightarrow{[\tau]} q$. Now if $|Q| \leq n$ we have at least one r_i being repeated in the run and from Lemma 2.10 it follows directly that $p \longrightarrow r_i$, $r_i \rightarrow r_i$ and $r_i \longrightarrow q$ are in τ. ■

To put it very crudely (but convenient though), a idempotent transition profile represents several loops in a certain state set, starting and ending in the specified states. Lemma 2.11 shows that in an non-empty idempotent transition profile you always have at least one pair $q \longrightarrow q$, i.e. a self loop. The states with self loops define the states in which we can start and end by running in $\mathfrak{A}$ via the words of $[\tau]$ with regard of visiting final states. There also exist other runs of $\mathfrak{A}$ on $[\tau]$, but these runs must been compensated in exactly the mentioned self-looping states. Even the runs of $\mathfrak{A}$ that actually perform a loop but do not belong to an idempotent transition profile directly are also included in some idempotent transition profile, since the loops to the corresponding states are contained as subloops in an idempotent transition profile. So, just like the languages of all transition profiles form a partition of Σ^* (see [SVW87]), the set ITP covers all looping runs of $\mathfrak{A}$ starting from any state in Q. The empty transition profile, which was mentioned to be idempotent, too, forms a special case, as it describes all those words that lead to no state at all, even if we are allowed to start in any state. Repeating the words of all idempotent transition profiles over and over again which we will specify in Section 3, this makes it possible to look at all infinite runs $\mathfrak{A}$ can have on any word.

To give a small remark on complexity, it may be the case that all of the transition profiles of a given automaton $\mathfrak{A}$ are idempotent, also if it is non-universal.

2.2.2 Simultaneous Powerset Automaton

We already mentioned that all equivalence classes of $\sim_{\mathfrak{A}}$, the transition profiles, are regular. To clarify this we will now build a finite automaton out of the whole set of transition profiles. This procedure is based on the generalized subset construction used in [SVW87] to build a family of deterministic finite automata that catch the behavior of $\mathfrak{A}$, concerning reachability of states.

Definition 2.12 *Given the transition profiles of $\mathfrak{A}$ over an alphabet Σ one can construct the* simultaneous powerset automaton $SPA(\mathfrak{A})$ *by using them as states and add an a-labeled edge ($a \in \Sigma$) from $\tau(u)$ to $\tau(v)$ if $\tau(ua) = \tau(v)$. Use $\tau(\varepsilon)$ as initial state.*

This automaton model can be understood in the way that we look at all reachabilities of state sets starting from any state in $\mathfrak{A}$ regarding possible visits of a final state. In general the SPA can consist of at most 4^{n^2} states [SVW87].

Remark 2.13 The words of $\tau(u)$ are accepted by $SPA(\mathfrak{A})$ with $\tau(u)$ as only final state. We denote it as $SPA_{\tau(u)}(\mathfrak{A})$.

By this we are able to cover the word family of each transition profile by directly constructing a deterministic automaton out of the given input automaton.

3 Complementation Algorithm

Given a BA $\mathfrak{A}$, a word w is in $L(\mathfrak{A})$ if there is a run of w on $\mathfrak{A}$, visiting a state in F infinitely often. So, in order to check if a word is in the complement language we need to look if for all runs of w on $\mathfrak{A}$ we visit only finitely many states in F. This fact makes it clear that the complement automaton $\overline{\mathfrak{A}}$ we want to construct out of $\mathfrak{A}$, may not guess the run w takes on $\mathfrak{A}$ but has to check all the runs. Now we want to use the definitions and models of the last chapter to examine all these runs in order to build the general complementation algorithm. We then can proceed with optimization techniques in section 6, but first we want to go step-by-step to the quite intuitive construction of the complement automaton. Starting again with the representation of a regular ω-language, we move on to the different components that are needed to capture the corresponding parts of the complement language.

3.1 Complement Language

Given the input BA $\mathfrak{A}$, we know that $L(\mathfrak{A})$ is regular and by Definition 2.5 can be seen as $\bigcup_{i=1}^{n} U_i \cdot V_i^{\omega}$ with regular $U_i, V_i \subseteq \Sigma^*$. This also holds for the complement language, i.e. for every word $w \notin L(\mathfrak{A})$ it holds that $w = u \cdot v_i^{\omega}$. Actually, this is the core statement of Ramsey´s Theorem A [Ram29] on which the algebraic approach of BA complementation has been settled by Büchi [Büc62].

Lemma 3.1 *Given a congruence on the state set of a BA $\mathfrak{A}$, any word $w \in \Sigma^{\omega}$ can be represented by $w = uv_0v_1\ldots$, where u is part of an equivalence class U and every v_i of the same equivalence class V of the congruence. So for any $w \in \Sigma^{\omega}$ it holds that $w \in U \cdot V^{\omega}$, for some equivalence classes U and V of the congruence.*

For the proof we refer the reader to [Tho90].

As $\sim_{\mathfrak{A}}$ defines the desired congruence, all we need are suitable U- and V-components. Now, for the U-component we can simply use the powerset automaton of

$\mathfrak{A}$ (taken $\mathfrak{A}$ as NBA) instead of a specific set of equivalence classes like it is done in some previous works ([Roe08] et. al.). Of course the acceptance state set is reduced only to the sink state, if existent. We will denote this special powerset automaton as $PA_{\emptyset}(\mathfrak{A})$. This alternative approach has already been applied in the work [SVW87]. The idea behind this is that the different u any word $w \in \Sigma^{\omega}$ begins with, only define a finite prefix before moving on to the infinite V-component. So we can just memorize the set of states $P \subseteq Q$ we reach, reading this finite prefix, which is actually the purpose of the powerset automaton. Coming from a certain set P we can use specific transition profiles for the V-component. Specific means here, that all infinite runs starting from the state set P do not visit a final state infinitely often. Which transition profiles are needed here in conjunction with all the different state sets of the powerset automaton, will be discussed in the next section.

3.2 Dangerous and Safe Pairs

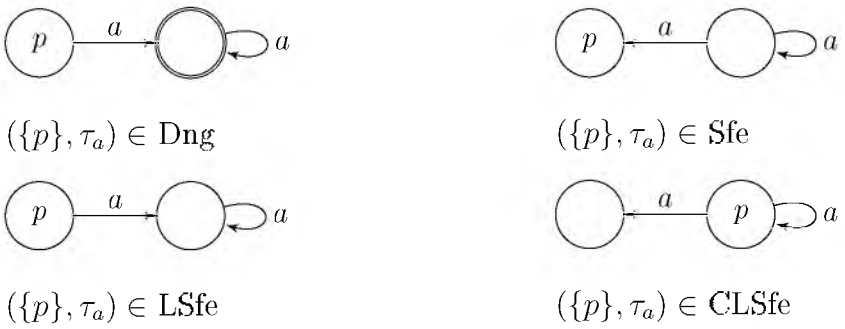
In Section 2 we have already described that only idempotent transition profiles (the set ITP) are needed to get all possible infinite runs on the state set of $\mathfrak{A}$ regarding the visit of final states. Both Büchi [Büc62] and Friedgut, Kupferman, Vardi [FKV06] defined a subset of ITP which is still sufficient. Orientating ourselves on these works we want to gather the set of transition profiles that is needed, by looking at all the pairs (P, τ).

Definition 3.2 *Given BA* $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F), P \subseteq Q$ $(P \neq \emptyset)$ *and* τ *in ITP, the pair* (P, τ) *is called...*

- ... dangerous *if from* P *we can reach an accepting loop in* τ*, i.e. there is a run of* $\mathfrak{A}$ *of form* $p \xrightarrow{[\tau]} q \xrightarrow{[\tau]} q, p \in P, q \in Q$.
- ... safe *if it is not dangerous.*
- ... looping safe *if it is safe and from* P *we can reach a non-accepting loop in* τ*, i.e. there is a run of* $\mathfrak{A}$ *of form* $p \xrightarrow{[\tau]} q \xrightarrow{[\tau]} q, p \in P, q \in Q$.
- ... completely looping safe *if it is looping safe with the non-accepting loop already being described by a state of* P*, i.e. there is a run of* $\mathfrak{A}$ *of form* $p \xrightarrow{[\tau]} p$, $p \in P$.

Define the sets Dng, Sfe, LSfe, CLSfe, and also $TP_{Dng}, TP_{Sfe}, TP_{LSfe}, TP_{CLSfe}$ containing all transition profiles that fulfill the corresponding property with at least one P.

A very rough example that illustrates the meaning of all pairs, referring to the original automaton is shown in Figure 3.1.



$(\{p\}, \tau_a) \in$ Dng $(\{p\}, \tau_a) \in$ Sfe

$(\{p\}, \tau_a) \in$ LSfe $(\{p\}, \tau_a) \in$ CLSfe

Figure 3.1: Dng, Sfe, LSfe and CLSfe referring to input automaton

It is trivial that dangerous and safe pairs form a partiton of all pairs. We also want to take down that if both Dng and Sfe are empty, this means that there is no infinite run possible at all. So as long as we look at accurate BAs, this may not be the case. Furthermore it is clear that CLSfe $\subseteq$ LSfe $\subseteq$ Sfe. But we can even show that $TP_{LSfe} = TP_{CLSfe}$.

Lemma 3.3 *(P_1, τ) in LSafe $\Rightarrow$ There is a P_2 with (P_2, τ) in CLSafe.*

Proof. (P_1, τ) is looping safe, so there is a run on $\mathfrak{A}$ of form $p \xrightarrow{[\tau]} q \xrightarrow{[\tau]} q, p \in P_1, q \in Q$. If $q \in P_1$ the assumption is right with $P_2 = P_1$. Otherwise the run implies, that in the powerset automaton of $\mathfrak{A}$ (taken as NBA) we have a run from the state with label P_1 to a state with label P_2 with $q \in P_2$ via $[\tau]$, because in $\mathfrak{A}$ we have a run $p \xrightarrow{[\tau]} q$. It also holds that there is a run $q \xrightarrow{[\tau]} q$, so (Q, τ) is in CLSfe. ■

The reason for splitting the looping safe and the completely looping safe pairs in spite of their equal TP sets is that only pairs of CLSfe are needed to connect the different components of the complement automaton in the end. We will elaborate this in the upcoming Section 3.3. Another reason is revealed in Section 4. There we will see that the completely safe pairs help us to reduce the state space of the complement automaton more efficiently.

Lemma 3.4 *If τ is idempotent and (P, τ) is safe but not looping safe it holds that τ is empty in the set P, i.e. there is no $p \longrightarrow q$ in τ with $p \in P, q \in Q$. This means that there is no run in the corresponding BA starting in some $p \in P$, reaching another state $q \in Q$ reading any word from $[\tau]$.*

Proof. τ is not looping safe, so there shall be no loop reachable in τ, i.e. there are no pairs $p \longrightarrow q$ and $q \longrightarrow q$ both in τ with $p \in P, q \in Q$. As a direct consequence of Lemma 2.11, $p \longrightarrow q$ in τ would mean that there is such a loop reachable, so there can be no $p \longrightarrow q$ in τ for any $p \in P, q \in Q$, so τ is empty in the set P. ■

By the definition of the pairs (P, τ) and the meaning of them referring to the runs they indicate on the BA they have been derived from, we can show some interesting trivial cases concerning the complement language.

Lemma 3.5 *Given BA $\mathfrak{A}$ and the sets Dng, Sfe, LSfe, the following holds:*

a) $Dng = \emptyset \Rightarrow \overline{L(\mathfrak{A})} = \Sigma^\omega$

b) $Sfe = \emptyset \Rightarrow \overline{L(\mathfrak{A})} = \emptyset$

c) $Sfe \neq \emptyset$ *and* $LSfe = \emptyset \Rightarrow \overline{L(\mathfrak{A})} = L(PA_\emptyset(\mathfrak{A}))$

Proof. a) Dng $= \emptyset$ means that there is no state set P reachable in $\mathfrak{A}$ where we have at least one state $p \in P$ from which we can perform an accepting run with the words of any idempotent transition profile. As all the idempotent transition profiles cover all possible infinite runs, there is no accepting infinite run reachable in $\mathfrak{A}$ and therefore $\overline{L(\mathfrak{A})} = \Sigma^\omega$

b) We proof the equivalent proposition $\overline{L(\mathfrak{A})} \neq \emptyset \Rightarrow$ Sfe $\neq \emptyset$. Assuming $\overline{L(\mathfrak{A})} \neq \emptyset$, we have a word $w \notin L(\mathfrak{A})$ which from some point on only visits non-final states. Decomposing w in the form that is provided by Lemma 3.1 ($w = uv_1v_2\ldots$), we see that by reading u we reach a certain state set P of the powerset automaton (as we have to consider all runs $\mathfrak{A}$ can have on w) and are able to continue with the words of a special equivalence class, namely a transition profile τ. But then (P, τ) would be in Sfe, so it holds that Sfe $\neq \emptyset$.

c) We know from Sfe $\neq \emptyset$ that there exists a non-accepting run of any form. LSfe $= \emptyset$ now means that all these non-accepting runs do not perform a

loop in any state set, but simply lead to no state of $\mathfrak{A}$ at all. This is a direct consequence of Lemma 3.4 as all corresponding pairs are safe but not looping safe. Now since all non-accepting runs of $\mathfrak{A}$ on any word lead to no state, which is captured by the sink state in the powerset automaton, it holds that $\overline{L(\mathfrak{A})} = L(PA_\emptyset(\mathfrak{A}))$, remembering that in this case the sink state is the only final state.

■

Here we assumed that we deal with a BA which has at least one infinite run. If this is not applicable, i.e. there is no infinite run at all, this means that Dng and Sfe are empty at the same time. Of course, then case a) is preferable to case b), as the language of the input automaton is empty. Since LSfe being non-empty results in CLSfe being non-empty (see Lemma 3.3) no further conclusion can be made at this point.

3.3 Constructing the complement automaton

Looking again at the regular expressions $\bigcup_{i=1}^{n} U_i \cdot V_i^\omega$ which represents the complement automaton in the end, we see that by now we have an automaton that covers the U-component ($PA_\emptyset(\mathfrak{A})$) and several automata for the V-component (SPAs). But yet there are still two steps missing in order to "put together" the complement automaton.

3.3.1 The ω-Operation

The V-component in which we loop at the end for a word $w \in U_i \cdot V_i^\omega$ shall consist of the word-family of one specific transition profile τ. We get these words by taking τ as the only final state in the SPA (SPA_τ), but yet we have not considered the ω. So in order to repeat the words of $[\tau]$ properly we need to build an automata for $[\tau]^\omega$.

Lemma 3.6 *Given a NFA* $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ *one can construct a BA* $\mathfrak{B} = (Q', \Sigma, q_0', \Delta', F')$ *with* $L(\mathfrak{B}) = L(\mathfrak{A})^\omega$ *by setting*

- $Q' = Q \cup q_0'$

- $q_0' = \begin{cases} q_0 & , q_0 \textit{ has no incoming transitions} \\ q_0' \textit{ as new initial state} & , \textit{else} \end{cases}$

- $\Delta' = \Delta \cup \{(q_0', a, p) \mid \exists a \in \Sigma, p \in Q : (q_0, a, p) \in \Delta\}$
  $\cup \{(q_0', a, q_0') \mid \exists a \in \Sigma, q_F \in F : (q_0, a, q_F) \in \Delta\}$
  $\cup \{(p_F, a, q_0') \mid \exists a \in \Sigma, p \in Q, q_F \in F : (p, a, q_F) \in \Delta\}$

- $F' = \{q_0'\}$.

Proof. "$\supseteq$": With $w \in L(\mathfrak{A})^\omega$ it is decomposable as $w = u_1 u_2 \ldots$ with $u_i \in L(\mathfrak{A})$ for all $i \geq 1$. That means that for each u_i there is a run on $\mathfrak{A}$ from q_0 to a final state. Now in $\mathfrak{B}$ we can jump from the predecessors of the final states in Q to q_0' again. So no matter if q_0' equals q_0 or is a new initial state we get a run from q_0' to q_0' in $\mathfrak{B}$ for each u_i. So with q_0' as (only) final state we get an accepting run of w on $\mathfrak{B}$, i.e. $w \in L(\mathfrak{B})$.

"$\subseteq$": With $w \in L(\mathfrak{B})$ it is decomposable as $w = u_1 u_2 \ldots$ with a visit of q_0' (as only final state) after each $u_i, i \geq 1$. So for each u_i there is a run on $\mathfrak{B}$ from q_0' to q_0'. Now if q_0' is a new initial state, we see that in $\mathfrak{A}$ we have a run from q_0 to a state in F for each u_i. Otherwise, if q_0' equals q_0, this means that q_0 has no incoming transitions. So it is ensured, that for each u_i the run on $\mathfrak{A}$ leads with its final step to a state in F and not to q_0 again. In both cases each u_i is accepted by $\mathfrak{A}$, i.e. $w \in L(\mathfrak{A}^\omega)$. ■

We can use this lemma to construct an automata for $[\tau]^\omega$ right out of the given SPA for $[\tau]$. The upcoming example shows, that the ω-operation is generally necessary.

3.3.2 Linking Transitions

The only thing that is left missing is the connection between the automaton $PA_\emptyset(\mathfrak{A})$ and the different SPAs. For those transitions we can simply add a-labeled edges ($a \in \Sigma$) from a state in the PA to an a-successor of the initial state in the SPA and mark the initial states in the SPAs as non-initial. The proof that this construction is leading to the desired language is quite intuitive but what we have to do is identifying the pairs (P, τ) for which such a linking transition is needed. Actually we have already done this by defining the completely looping safe pairs.

As already mentioned above this set CLSfe gives us the desired pairs, for which a linking transition is necessary, although you could think of using all safe pairs. In fact, this would do no harm to the complement language, however a closer look reveals that those transitions are redundant.

Remark 3.7 Linking transitions from a P-labeled states in $PA_\emptyset(\mathfrak{A})$ to a SPA_τ are only necessary for those (P, τ) that are in CLSfe.

Remembering Lemma 3.4 we can derive that from a P-labeled state in $PA_\emptyset$ reading words of $[\tau]$ leads us to the accepting sink state, if the pair (P, τ) is safe only. So no further examination is needed here, the complement automaton has already accepted the word reading its finite prefix u. Furthermore, Lemma 3.3 shows that with (P, τ) only being looping safe, we have a run of $PA_\emptyset$ on the words of $[\tau]$ from the P-labeled state to another Q-labeled state that is completely looping safe with τ. So from this point of view, the step from P to Q still belongs to the finite prefix and it is the completely looping safe pair (Q, τ), which covers the infinite run with the words of $[\tau]$. All in all, taking only the pairs of CLSfe in consideration for the linking transitions suffices in this context.

An illustration of how our final complement automaton is structured is given in Figure 3.2, containing $PA_\emptyset$ for the U-component, the linking transition to the SPAs for the V-component and the proper repetition of their languages, gained by the ω-operation.

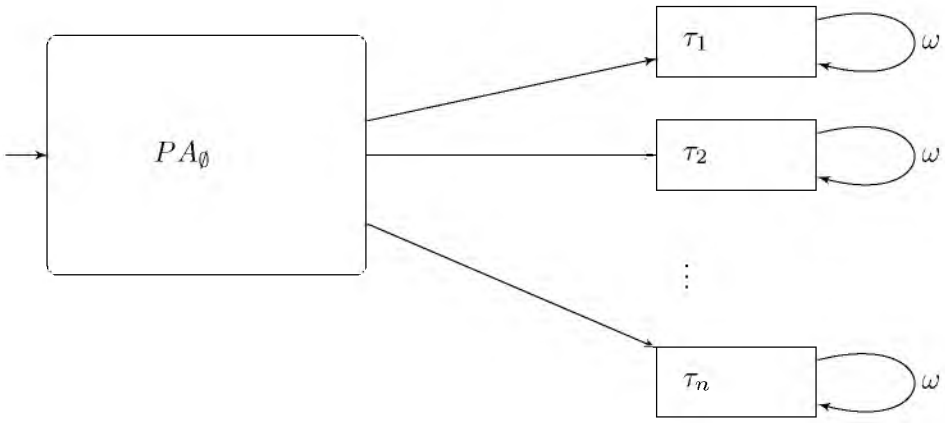


Figure 3.2: Illustration of the complement automaton

3.4 The Complementation Algorithm

We now have defined all the components and procedures that are needed to construct the complement automaton for a given NBA $\mathfrak{A}$. So, it is possible to declare an algorithm performing the different steps, namely

- Build powerset automaton $PA_\emptyset(\mathfrak{A})$, where $\mathfrak{A}$ is taken as NFA.
- Construct simultaneous powerset automaton SPA($\mathfrak{A}$).
- From SPA extract the set of all idempotent transition profiles ITP.
- Compute the sets Dng, Sfe, LSfe, CLSfe, containing all pairs of state sets in PA and idempotent transition profiles (P, τ) that fulfill the corresponding property.
- Catch the trivial cases as it is described in Lemma 3.5
- For every pair (P, τ) in CLSfe apply the ω-operation on SPA_τ and add the according linking transitions from P to SPA_τ

The method of building the powerset automaton of an automaton should be clear, so we move on to the simultaneous powerset automaton in Algorithm 1. There we start with the empty transition profile $\tau(\varepsilon)$ and with use of the Remark 2.8 –$\tau(uv)$ can be computed from $\tau(u)$ and $\tau(v)$– we simply construct all possible representatives, appending every symbol of Σ again and again to all gained transition profiles. By this we get the state set of the SPA and are also able to add all corresponding transitions. When there has no new transition profile occurred in an whole iteration, we are done and can return the SPA with $\tau(\varepsilon)$ as initial state and $F = \emptyset$.

Given the SPA, it is rather simple to extract all the idempotent transition profiles, as Algorithm 2 illustrates. For each transition profile $\tau(u)$ it checks if $\tau(u)$ equals $\tau(uu)$, according to the Definition 2.9. The empty transition profile $\tau(\varepsilon)$ is only added in ITP, if there is an incoming transition to it, what means that another nonempty transition is equal, so $[\tau(\varepsilon)] \neq \varepsilon$. In that case ε is just a common representative.

The next Algorithm 3 computes the sets of transition profiles described in Definition 3.2, namely Dng, Sfe, LSfe, CLSfe. It requires the powerset automaton PA($\mathfrak{A}$)

Algorithm 1 Compute SPA($\mathfrak{A}$)

Require: NBA $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$
Ensure: Simultaneous powerset automaton SPA($\mathfrak{A}$)

```
1: Q' ← {τ(ε)}
2: Δ' ← ∅
3: allTP ← (τ(ε))
4: current ← 0
5: while current < |allTP| do
6:    currentTP ← allTP[current]
7:    for all a ∈ Σ do
8:       nextTP ← currentTP·a
9:       if nextTP ∉ allTP then
10:         Q' ← Q' ∪ nextTP
11:         allTP ← allTP ∪ nextTP
12:      end if
13:      Δ' ← Δ' ∪ {(currentTP, a, nextTP)}
14:   end for
15:   current ← current+1
16: end while
17: SPA ← (Q', Σ, {τ(ε)}, Δ', ∅)
18: return SPA
```

where

- allTP[x] is the element at the x-th position of allTP
- currentTP·a refers to the a-appended transition profile, as described in the Remark 2.8

Algorithm 2 Compute ITP

Require: SPA($\mathfrak{A}$)
Ensure: Set ITP containing all idempotent transition profiles

```
ITP ← ∅
if τ(ε) has an incoming transition then
    ITP ← ITP ∪ τ(ε)
end if
for all τ(u)-labeled states in SPA do
    if τ(u) = τ(uu) then
        ITP ← ITP ∪ τ(u)
    end if
end for
return ITP
```

and the set ITP in order to know which pairs of sets P and idempotent transition profiles τ have to be examined. Then it checks the conditions according to the definition, starting with a double depth first search (DFS) to look for accepting loops. If one is reachable from P in τ the pair (P, τ) is dangerous, otherwise safe. To check if it is looping safe, Lemma 3.4 provides that one just has to test τ for emptiness in P and if there is a loop in P itself, (P, τ) is known to be completely looping safe.

Finally, we can use these algorithms to provide the code of the complement construction, given in Algorithm 4. It works exactly the way we defined it in the beginning of this chapter, first computing $PA_\emptyset(\mathfrak{A})$ for the U-component and SPA($\mathfrak{A}$) for the V-component. After computing all idempotent transition profiles and corresponding pairs (P, τ) it first checks, if any trivial case can be derived from the sets Dng, Sfe, LSfe making use of Lemma 3.5. If it has not returned a complement automaton yet, the algorithm continues with the ω-operation on each SPA_τ and the linking transitions from every P to SPA_τ, where (P, τ) is in CLSfe. The complement automaton $\overline{\mathfrak{A}}$ is updated in every step and when we are done with putting together the components, $\overline{\mathfrak{A}}$ is returned, recognizing exactly the complement language of the input automaton.

Lemma 3.8 *Given NBA $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ the automaton $\overline{\mathfrak{A}}$ returned by Algorithm 4 is the complement of $\mathfrak{A}$, i.e. $\overline{L(\mathfrak{A})} = L(\overline{\mathfrak{A}})$.*

Proof. "$\subseteq$": Given $w \in \overline{L(\mathfrak{A})}$, so $w \notin L(\mathfrak{A})$, it holds that all runs of $\mathfrak{A}$ on w are not accepting, i.e. from some point on only non-final state are visited again and again. With the congruence $\sim_{\mathfrak{A}}$ and the decomposition of $w = uv_0v_1\ldots$ according

Algorithm 3 Compute Pairs

Require: PA($\mathfrak{A}$), set ITP
Ensure: Sets Dng, Sfe, LSfe, CLSfe with all corresponding TPs

```
1: Dng ← ∅
2: Sfe ← ∅
3: LSfe ← ∅
4: CLSfe ← ∅
5: for all P ∈ PA do
6:   for all τ ∈ ITP do
7:     if no accepting loop reachable from P in τ then
8:       Sfe ← Sfe ∪ (P, τ)
9:       if τ is not empty in P then
10:        LSfe ← LSfe ∪ (P, τ)
11:        if τ has loop in P then
12:          CLSfe ← CLSfe ∪ (P, τ)
13:        end if
14:      end if
15:    else
16:      Dng ← Dng ∪ (P, τ)
17:    end if
18:  end for
19: end for
20: return Dng, Sfe, LSfe, CLSfe
```

where

- PA($\mathfrak{A}$) is the powerset automaton of $\mathfrak{A}$
- checking if there is no accepting loop reachable from P in τ first computes all reachable states R from P in τ via DFS and checks for all $r \multimap q$ ($r \in R, q \in Q$) in τ if q reaches r again by DFS

Algorithm 4 Complement NBA $\mathfrak{A}$

Require: NBA $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$
Ensure: Complement automaton $\overline{\mathfrak{A}}$

1: PA $\leftarrow PA_{\emptyset}(\mathfrak{A})$
2: SPA $\leftarrow$ SPA($\mathfrak{A}$)
3: ITP $\leftarrow$ ITP(SPA)
4: Dng, Sfe, LSfe, CLSfe $\leftarrow$ ComputePairs(PA,ITP)
5: CatchTrivialCases
6: $\overline{\mathfrak{A}} \leftarrow$ PA
7: **for all** (P, τ) in CLSfe **do**
8: $\quad SPA_\tau \leftarrow \omega$-operation$(SPA_\tau)$
9: $\quad \overline{\mathfrak{A}} \leftarrow$ Link(P, SPA_τ)
10: **end for**
11: **return** $\overline{\mathfrak{A}}$

where

- SPA($\mathfrak{A}$) calls Algorithm 1, ITP(SPA) calls Algorithm 2 and ComputePairs(PA,ITP) calls Algorithm 3
- CatchTrivialCases returns the corresponding automaton if any case of Lemma 3.5 holds
- ω-operation(SPA_τ) applies the ω-operation described in Lemma 3.6 and Link(P, SPA_τ) adds the linking transitions from P to SPA_τ as illustrated in Section 3.3.2.

to Lemma 3.1 we can define the point from which no more final state is visited in $\mathfrak{A}$ after reading the prefix u. Looking at all runs, we are then in a state set P, captured by the powerset automaton PA, and continue reading the words of a certain transition profile τ, which is the desired equivalence class for the words v_i. Now, in $\overline{\mathfrak{A}}$ we get an accepting run w in two cases: For one thing, if u already lead us to no state in $\mathfrak{A}$ we are in the sink state of PA, which is accepting in $\overline{\mathfrak{A}}$ with each symbol. And for other thing, w is accepted if τ is completely looping safe with P, as this describes a run via a linking transition, described in Remark 3.7 to SPA_τ. The language of SPA_τ is then repeated properly by the ω-operation of Lemma 3.6, visiting the origin initial state of SPA_τ (as only final state) again and again. Together, this leads to $w \in L(\overline{\mathfrak{A}})$. The trivial cases described and proved in Lemma 3.5 are also captured.

"$\supseteq$": Given $w \in L(\overline{\mathfrak{A}})$ it is accepted in exactly two cases: If existent, we run in the sink state of $PA_\emptyset$, which means in $\mathfrak{A}$ we also run in the sink state, but with the difference that it is non-accepting, so $w \in \overline{L(\mathfrak{A})}$. The other case defines a run of $\overline{\mathfrak{A}}$ on w, first reading a finite prefix u, moving in $PA_\emptyset$ and then, jumps from a P-labeled state via a linking transition to a certain SPA_τ, where (P, τ) is completely looping safe. With (P, τ) being in CLSfe, by Definition 3.2 we do not reach any state in $\mathfrak{A}$ from P that does describe a run that is visiting a final state infinitely often. So as there is no accepting run of $\mathfrak{A}$ on w it holds that $w \notin L(\mathfrak{A})$, i.e. $w \in \overline{L(\mathfrak{A})}$. Again, the trivial cases proven in Lemma 3.5 are recognized, too. ■

As there are at most 2^n states in PA and we can have up to 4^{n^2} copies of SPA — that is when all transition profiles are idempotent and in TP_{CLSfe}— each with 4^{n^2} states, as those are equal to the number of transition profiles, this results in a total upper bound of $2^n + 4^{n^2} \cdot 4^{n^2} = 2^n + 16^{n^2}$ for the number of states in the complement automaton. This high number can be reduced by several optimizations, which we will present in Section 4.

3.5 Example

Let $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ be the NBA given in Figure 3.3. Before applying the algorithm, we want to analyze $\mathfrak{A}$ in its structure also concerning the complement of it. A regular expression, describing $L(\mathfrak{A})$ is given by $\Sigma^* ac^\omega + (c + a(a + c)^* b)^\omega$. Although this is a quite confusing language, the automaton reveals the main aspects of our complementation algorithm. To obtain a word that is not in $L(\mathfrak{A})$

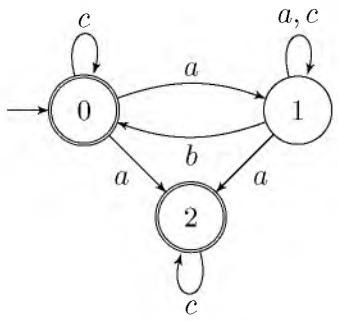


Figure 3.3: Input automaton $\mathfrak{A}$

we have to read one b in the state 0 or stay from some point on in the state 1, reading a combination of a and c. But as running from 0 to 1 is only possible by reading a and this also leads in the state 2, this combination may not "end" by infinitely manc c, because this would give an accepting run in 2. We will see that the Algorithm 4, we now want to apply, takes this into consideration.

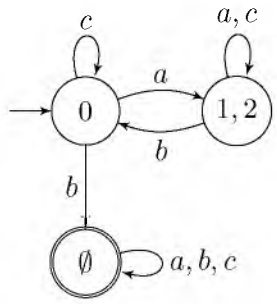
3.5.1 $\mathrm{PA}_\emptyset$ and SPA

First it is necessary to build $\mathrm{PA}_\emptyset(\mathfrak{A})$ and $\mathrm{SPA}(\mathfrak{A})$ out of the input automaton $\mathfrak{A}$ to capture all the runs a word can have on $\mathfrak{A}$. The construction of $\mathrm{PA}_\emptyset$ is done as the usual powerset construction, setting the sink state final, while building $\mathrm{SPA}(\mathfrak{A})$ is a straightforward application of Algorithm 1, appending the symbols of the alphabet one by one, comparing already gained transition profiles and setting the corresponding transitions. Both automata are given in Figure 3.4. For a comprehensive view we write τ_u instead of $\tau(u)$.

3.5.2 Extracting Pairs

In the next step we want to find the set ITP of all idempotent transition profiles by checking for each τ_u if $\tau_u = \tau_{uu}$, that is also when the unique run ρ of u (as SPA is a DFA) starting in τ_u ends in τ_u again. By this it is easy to check which transition profiles are idempotent. In the following we use a graph to note which pairs are in τ, where an arrow from p to q means that $p \longrightarrow q$ is in τ, and an F-labeled arrow indicates that the pair $p \multimap q$ is in τ. Figure 3.5 shows the elements of ITP.

$\mathrm{PA}_\emptyset(\mathfrak{A})$:



$\mathrm{SPA}(\mathfrak{A})$:

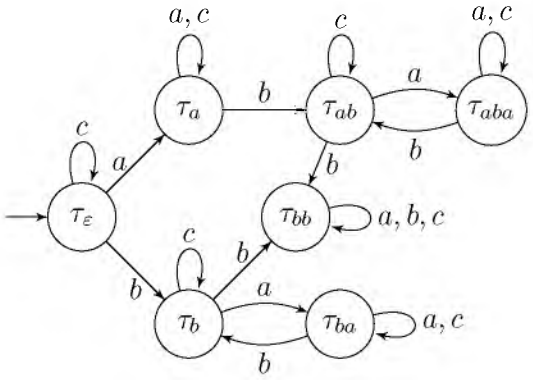


Figure 3.4: $\mathrm{PA}_\emptyset(\mathfrak{A})$ and $\mathrm{SPA}(\mathfrak{A})$

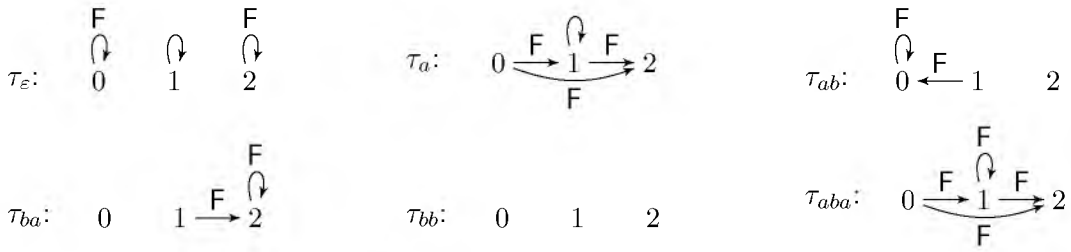


Figure 3.5: The set ITP

Now, we can declare the sets Dng, Sfe, LSfe, CLSfe (cf. 3.6) by checking the according property with all P-labeled states ($P \neq \emptyset$) in PA, namely $\{0\}$ and $\{1, 2\}$ directly in the graph of τ. As a reminder: (P, τ) is in Dng if it reaches an accepting loop, in Sfe if this is not the case, in LSfe if we also reach a non-accepting loop and finally in CLSfe if this non-accepting loop is already reachable in P.

3.5.3 Complement Automaton

As we have seven dangerous and one completely looping safe pair, namely $(\{1, 2\}, \tau_a)$, none of the trivial cases described in Lemma 3.5 holds. So we now put together the complement automaton, first applying the ω-operation to SPA_{τ_a} in order to repeat the languages of this automaton in the right way. It can be seen that τ_ε has an incoming transition, so the ω-operation yields a new initial state q_0'. Here, we also see that omitting the ω-operation results in a false complement. With τ_a as

Dng	Sfe	LSfe	CLSfe
$(\{0\}, \tau_\varepsilon)$	$(\{0\}, \tau_a)$	$(\{0\}, \tau_a)$	$(\{1,2\}, \tau_a)$
$(\{0\}, \tau_{ab})$	$(\{0\}, \tau_{ba})$	$(\{1,2\}, \tau_a)$	
$(\{0\}, \tau_{aba})$	$(\{0\}, \tau_{bb})$		
$(\{1,2\}, \tau_\varepsilon)$	$(\{1,2\}, \tau_a)$		
$(\{1,2\}, \tau_{ab})$	$(\{1,2\}, \tau_{bb})$		
$(\{1,2\}, \tau_{ba})$			
$(\{1,2\}, \tau_{aba})$			

Figure 3.6: The sets Dng, Sfe, LSfe, CLSfe

final state we would also accept those words ending with ac^ω, but these are obviously accepted by $\mathfrak{A}$, too, since state 2 offers an accepting run. The last step adds the linking transitions from $PA_\emptyset$ to SPA_{τ_a}. By now, we see that SPA_{τ_a} covers the run described above of staying in 1 and reading the particular combination of a, c. Running into the accepting sink refers to reading b in the state 0. The final complement automaton $\overline{\mathfrak{A}}$ is given in Figure 3.7.

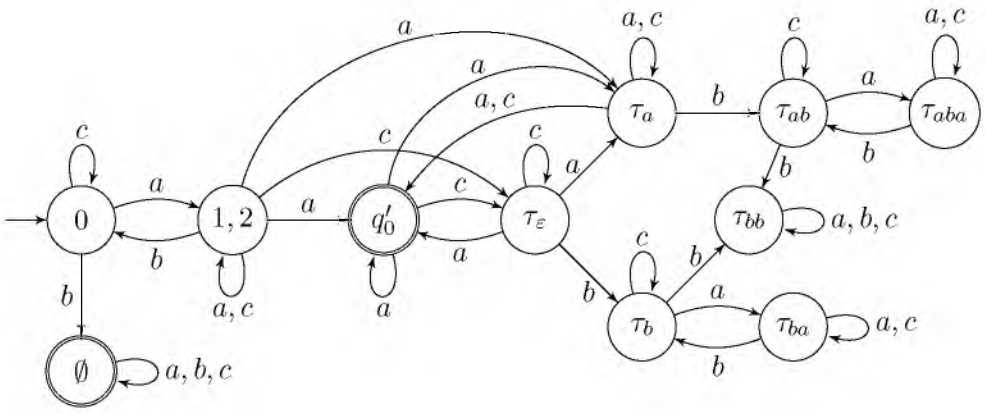


Figure 3.7: Complement automaton $\overline{\mathfrak{A}}$

4 Optimization

Yet, we see in Figure 3.7 that there are many superfluous states in the complement automaton, as for example the unproductive branches in each SPA. Furthermore, the 4^{n^2} copies of SPA results in an explosion of the state space, too, and we will show that there even exist mergeable states in the PA for the U-component. So, in this chapter we want to present several techniques and heuristics in order to give an optimized version of Algorithm 4, where its function is shown in an example at the end of this section.

4.1 Optimization Techniques

4.1.1 Minimizing SPAs

As the SPA is a deterministic finite automaton one basic optimization is clear to see. Before applying the ω-operation we can minimize this SPA with the common minimization algorithms on finite automata. In this work we use the constructions of Valmari and Lehtinen for minimizing DFAs [VL08] and Matz and Potthoff in order to reduce a given NFA [MP95]. With $\mathfrak{A}'$ as minimized form of $\mathfrak{A}$ it is trivial that $L(\mathfrak{A})^\omega = L(\mathfrak{A}')^\omega$. Various tests showed that minimizing the SPA with the DFA-minimization-algorithm [VL08] or a NFA-reduction [MP95] resulted in no difference concerning to the number of states of the reduced automaton. A closer analysis of this phenomenon is left as future work.

As can be seen in the example given at the end of Section 3, minimizing the SPA already gives us a saving of many superfluous states (cf. Figure 4.1). It shall be stated explicitly that the construction of Valmari and Lehtinen [VL08] results in a DFA with partial transition functions, so also leaving out the sink state.

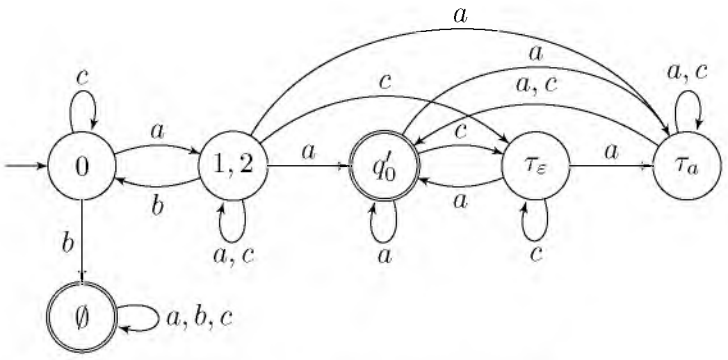


Figure 4.1: $\overline{\mathfrak{A}}$ with minimized SPA

4.1.2 Merging Transition Profiles

The next optimization makes use of the fact that there are transition profiles in which no new accepting loop occurs if you combine them, i.e. without gaining an accepting run in the input automaton. Combine, here means reading from some point on (in the V-component) only words of the one and also the other transition profile in an arbitrary order.

Definition 4.1 *We call the union of* $\tau(u_1), \ldots, \tau(u_n)$ merge, *written* $\tau(u_0/\ldots/u_n)$, *or simply* $\tau_{0/\ldots/n}$ *for* $\tau_1, \ldots, \tau_n$

The words of $\tau_{0/\ldots/n}$ *are accepted by* $SPA(\mathfrak{A})$ *with* $\tau_1, \ldots, \tau_n$ *as set of final states, written as* $SPA_{\tau_{0/\ldots/n}}(\mathfrak{A})$

Definition 4.2 *Given two transition profiles* τ_1, τ_2 *and* $\mathcal{P} = \{P_1, \ldots, P_n\}$ *being all state sets either* τ_1 *or* τ_2 *is completely looping safe with, the two transition profiles are* mergeable *if there is no accepting loop reachable in* $\tau_{1/2}$ *from any* $P_i \in \mathcal{P}$.

Of course, merging matching transition profiles gives rise to new pairs in the sets Dng, Sfe, LSfe, CLSfe. As the upcoming Section 4.1.3 needs to know the safe pairs, we want to note briefly that the merge of two transition profiles is only safe with a state set when both of them are safe with it. But we infer from Remark 3.7 that we are mostly interested in completely looping safe pairs in order to get the needed linking transitions.

Lemma 4.3 $(P, \tau_{1/2})$ *is completely looping safe if* τ_1 *and* τ_2 *are mergeable and* (P, τ_1) *or* (P, τ_2) *is in CLSfe.*

Proof. From Definition 4.1 it holds that $\tau_{1/2}$ contains all loops of both τ_1 and τ_2, so it is guaranteed that $\tau_{1/2}$ has the desired loop in P. Furthermore Definition 4.2 directly shows that there are no accepting loops reachable, so $(P, \tau_{1/2})$ is completely looping safe. ■

Referring to the runs in our input automaton, this also ensures the correctness of the construction concerning the complement language.

If one visualizes what this optimization aims for, two facts are desirable. We want to merge as many SPAs as possible to lessen the number of copies and taking a closer look at this we are interested in the state space each copy has and get a best possible number, here, too. However, finding an optimal solution here is difficult, as the problems are equal to the set-covering-problem, being NP-complete.

Definition 4.4 *The* set-covering-problem *takes a set M, n subsets $N_1, \ldots, N_n$ of M and a $k \leq n$ to look if there is a union of k or less subsets in $N_1, \ldots, N_n$ that equals M.*

Formulated as optimization problem one searches for a covering with k being as small as possible, or if each N_i has a cost value $c(N_i) \in \mathbb{N}$, one wants a covering, such that $\sum_{i=1}^{k} c(N_i)$ is minimal.

In our case, M is the set TP_{CLSfe} and $N_1, \ldots, N_n$ are all possible subsets, i.e. all elements taken from the powerset of TP_{CLSfe}. If we now want to merge as many SPAs as possible, this refers to the optimization problem of finding a minimal k, as every subset represents one copy of SPA. Considering the number of states, we have to optimize the cost values, which correspond to the states of the according minimized SPA, where a set of transition profiles has not competing costs if the elements are not pairwise mergeable, e.g. the incremented number of n times the state size in the original SPA. So, showing that the problem of merging transition profiles is in NP is easy. Proving the assumption of the problem being NP-hard, too, and therefore NP-complete — e.g. by reduction of the set- covering-problem — remains for future work.

In Algorithm 5 we use is a greedy heuristic. Starting with some transition profile, we iterate over TP_{CLSfe} and merge the current transition profile, if possible. When all possible transition profiles have been considered, we continue with the ones we have not merged yet, until every transitions profile has been checked. The set of completely looping safe pairs CLSfe is also updated with every new merge. Then we return the partition $MTP_0, \ldots, MTP_n$ of TP_{CLSfe}, each containing a set

Algorithm 5 Find mergeable TP

Require: TP_{CLSfe}

Ensure: Partition $MTP_0, \ldots, MTP_n$ of TP_{CLSfe} all with pairwise mergeable TP

```
1: i ← 0
2: notChecked ← TP_CLSfe
3: for all τ1 in TP_CLSfe do
4:   if τ1 in notChecked then
5:     notChecked ← notChecked − τ1
6:     MTP_i ← MTP_i ∪ τ1
7:     for all τ2 in TP_CLSfe do
8:       if τ2 in notChecked then
9:         if τ2 mergeable with τ1 then
10:          notChecked ← notChecked − τ2
11:          τ1 ← merge(τ1, τ2)
12:          MTP_i ← MTP_i ∪ τ2
13:        end if
14:      end if
15:    end for
16:    i ← i + 1
17:  end if
18: end for
19: return MTP_0, ..., MTP_n
```

where

- τ_2 mergeable with τ_1 is tested corresponding to Definition 4.2, by merging the two transition profile and searching for an accepting loop, reachable (just like in Algorithm 3)
- merge(τ_1, τ_2) simply merges the two transition profiles according to Definition 4.1 also updating CLSfe as it is described in the Remark 4.3.

of pairwise mergeable transition profiles. This yields a quite good result on the constructed cases and also leads to much less states in random generated automata (see Section 5). But as there exist several heuristics for the set-covering-problem ([Pra92]) there is still some room to sharpen this optimization.

4.1.3 Reducing Powerset Automaton

The idea behind the next optimization is to merge equivalent states in the powerset automaton $PA_\emptyset$ for the U-component. Merging, here refers to the usual definition that provides an NFA-homomorphism on the states, leading to a quotient automaton. It is well known that the powerset automaton for an NFA is not minimal, especially when one changes the set of final states only to the sink state, like we do. But we also have to take in consideration that this merging regards the V-component, namely the SPAs reached by the states.

Definition 4.5 *Two states of $PA_\emptyset$, labeled with P_1 and P_2 are* ω-equivalent *if they are equivalent in $PA_\emptyset$ itself and for each $\tau \in TP_{CLSfe}$ it holds that (P_1, τ) in Sfe $\Leftrightarrow$ (P_2, τ) in Sfe.*

So, additionally to the common equivalence, we check if two states have the same meaning for the whole complement automaton, i.e. being safe with the same SPAs in the V-component.

Lemma 4.6 *Merging ω-equivalent states in the complement automaton $\overline{\mathfrak{A}}$ preserves the language $L(\overline{\mathfrak{A}})$.*

Proof. We give the idea of this proof by first stating that the language of a quotient automaton is a superset of the language which is recognized by the original automaton. So, we only have to internalize that we get no new accepting runs by merging two ω-equivalent states in $PA_\emptyset$. Considering the positive sink in $PA_\emptyset$ this is quite clear from P_1 and P_2 being equivalent in $PA_\emptyset$ itself. Now, we have to look which linking transitions can be taken from the states, as these lead to the accepting runs in the V-component, i.e. the SPAs. As a linking transition is only added from those states, that are completely looping safe with the transition profile τ that corresponds to the SPA, it holds that one pair, say (P_1, τ) (w.l.o.g.) is in CLSfe, so especially also in Sfe. Now by the definition we can derive that (P_2, τ) is in Sfe, too. But this means, referring to Definition 3.2 that starting from any state in P_2 in the original automaton, reading the words that belong to τ, no

accepting loop can be reached. So by merging ω-equivalent states we either lose nor get new accepting runs of $\overline{\mathfrak{A}}$, which means $L(\overline{\mathfrak{A}})$ is preserved. ■

4.1.4 ω-Examination

As seen in Section 3.3.1 the ω-operation on a SPA may cause an additional initial state, but sometimes this not essential for repeating the language properly (cf. Figure 4.2).

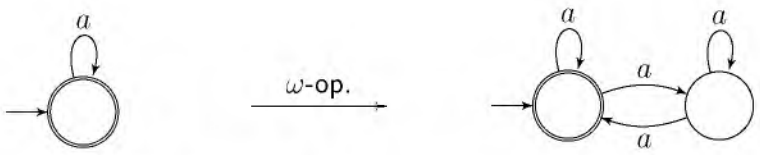


Figure 4.2: No need for ω-operaion, typical example

So it could be useful to check if the ω-operation is really necessary in order to spare these possible additional states. In this work, we just provide a very elementary test for this, only capturing the simplest cases.

Remark 4.7 For a given SPA_τ the ω-operation is not necessary if the outgoing transitions of the initial state are equal to the outgoing transitions of all final states.

This condition guarantees that after running into a final state we definitely go by a transition that introduces a new word of the language, as we could have taken this transition from the initial state, too.

To put it formally, for any SPA_τ it holds that $\lim(L(SPA_\tau)) \supseteq L(SPA_\tau)^\omega$, where $\lim(L) = \{w \in \Sigma^\omega \mid w(0)\dots w(i) \in L \text{ for infinitely many } i \geq 0\}$, containing all words that have an infinite number of prefixes in $L \subset \Sigma^*$. The ω-operation would not be necessary if $\lim(L(SPA_\tau)) \subseteq L(SPA_\tau)^\omega$. An effective implementation for this test is left as future work, as well as a closer analysis of examining the need of the ω-operation.

4.2 Optimized Complementation Algorithm

The optimized Algorithm 6 is structured like the origin Algorithm 4, but inherits the optimizations listed above. It also starts with constructing the powerset automatan $P_\emptyset$ and the simultaneous powerset automaton SPA out of the givem input automaton and continues with extracting the set ITP of all idempotent transition profiles. The next step is still similar to the original version, as it computes the sets Dng, Sfe, LSfe, CLSfe and catches the trivial cases discussed in Lemma 3.5. Now, instead of building all the SPA out of the set TP_{CLSfe} directly, the algorithm searches for mergeable transition profiles as it is described in Algorithm 5. We gain a partition of TP_{CLSfe}, namely $MTP_0, \ldots, MTP_n$, all containing pairwise mergeable transition profiles and update the pairs in CLSfe refering to Lemma 4.3, too. Now it forms the corresponding SPA for each MTP_i, minimizes it and add the linking transitions according to all pairs in CLSfe. Finally we again apply a minimization on the whole complement, pruning of dead states and reducing the state space in PA, by merging ω-equivalent states. Since we have discussed the correctness of each optimization and gave a proof of the original Algorithm 4, we will spare an additional verification of $\overline{L(\mathfrak{A})} = L(\overline{\mathfrak{A}})$ for the above procedure.

4.3 Example

Let $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ be the NBA given in Figure 4.3. It is in a way related

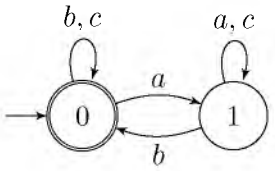


Figure 4.3: Input automaton $\mathfrak{A}$

to the example given above, but by the few differences it allows us to show the function of several optimizations nicely. Concerning to its language, $L(\mathfrak{A}) = (b + c + a(a + c)^* b)^\omega$, which means that after each occurrence of a there is some b. The only way to not accept a word, i.e. getting a word of the complement language, is to definitely have an occurrence of such an a, after which no more b is read. Concerning to the structure of $\mathfrak{A}$ this means to stay from some point on in the state 1, which is possible by two self-loops.

Algorithm 6 Complement NBA $\mathfrak{A}$ (optimized)

Require: NBA $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$
Ensure: Complement automaton $\overline{\mathfrak{A}}$

```
PA ← PA_∅(𝔄)
SPA ← SPA(𝔄)
ITP ← ITP(SPA)
Dng, Sfe, LSfe, CLSfe ← ComputePairs(PA,ITP)
CatchTrivialCases
𝔄̄ ← PA
MTP_0, ..., MTP_n ← FindMergables(TP_CLSfe)
for all (P, MTP_i) in CLSfe do
    SPA_MTPi ← DFA-minimization(SPA_MTPi)
    if ω-operation is necessary then
        SPA_MTPi ← ω-operation(SPA_MTPi)
    end if
    𝔄̄ ← Link(P, SPA_MTPi)
end for
𝔄̄ ← Reduce(𝔄)
return 𝔄̄
```

where

- FindMergables(TP_{CLSfe}) calls Algorithm 5
- DFA-minimization(SPA_{MTP_i}) applies a common construction for minimizing DFAs ([VL08]) on SPA_{MTP_i}
- Testing if the ω-operation is necessary is done as decribed in the Remark 4.7
- With MTP_i we actually describe the transition profile $\tau_{1/.../n}$, where $MTP_i = \{\tau_1, \ldots, \tau_n\}$, so (P, MTP_i) refers to the new pairs in CLSfe, as well as SPA_{MTP_i} refers to the corresponding SPA of the merged transition profile
- Reduce($\mathfrak{A}$) reduces the PA according to the description in Section 4.1.3 and also prunes of dead states that were obtained in the PA or a SPA by leaving out the ω-operation. If the output is only $PA_\emptyset$, which is possible when catching the trivial cases, we reduce this one, too.

4.3.1 $PA_\emptyset$ and SPA

Because of $\mathfrak{A}$ being deterministic, the powerset automaton reveals no change. So in $PA_\emptyset$ only the accepting state set turns to the empty set. The meaning behind this is that there is no run of $\mathfrak{A}$ on any infinite word that jumps out of the language in the finite prefix, referring to the decomposition $w = uv_0v_1\ldots$ — a word starting with any u can still fulfill the property of $L(\mathfrak{A})$ described above. $PA_\emptyset$ and SPA are given in Figure 4.4.

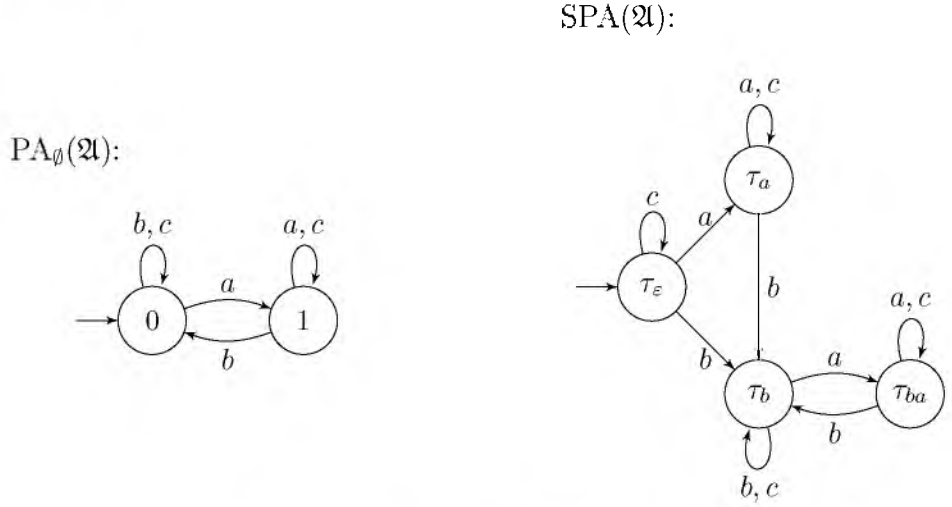
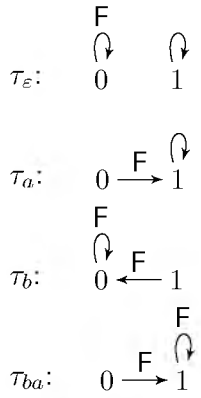


Figure 4.4: $\mathrm{PA}_\emptyset(\mathfrak{A})$ and $\mathrm{SPA}(\mathfrak{A})$

4.3.2 Extracting Pairs

Again, we first search for those transition profiles that are idempotent and then determine the sets Dng, Sfe, LSfe, CLSfe (cf. Figure 4.5), proceeding just like the way we have done this in the example of Section 3.

It can be seen that the two self-loops of state 1 are described by two completely looping safe pairs, namely $(\{1\}, \tau_\varepsilon)$ and $(\{1\}, \tau_a)$. Since there are also elements in the set Dng, we cannot finish with any trivial case.



Dng	Sfe	LSfe	CLSfe
$(\{0\}, \tau_\varepsilon)$	$(\{0\}, \tau_a)$	$(\{0\}, \tau_a)$	$(\{1\}, \tau_\varepsilon)$
$(\{0\}, \tau_b)$	$(\{1\}, \tau_\varepsilon)$	$(\{1\}, \tau_\varepsilon)$	$(\{1\}, \tau_a)$
$(\{0\}, \tau_{ba})$	$(\{1\}, \tau_a)$	$(\{1\}, \tau_a)$	
$(\{1\}, \tau_b)$			
$(\{1\}, \tau_{ba})$			

Figure 4.5: ITP and the sets Dng, Sfe, LSfe, CLSfe

4.3.3 Merging Transition Profiles

Now we are heading for the first optimization, finding mergeable transition profiles. So, in order to find those we apply the greedy Algorithm 5, which means that we are looking at all transition profiles in $TP_{CLSfe} = \{\tau_\varepsilon, \tau_a\}$, as these are the ones we need, to form a partition $MTP_1, \ldots, MTP_n$ of it. Starting with τ_ε, we merge it with τ_a to get $\tau_{\varepsilon/a}$ (cf. Figure 4.6).

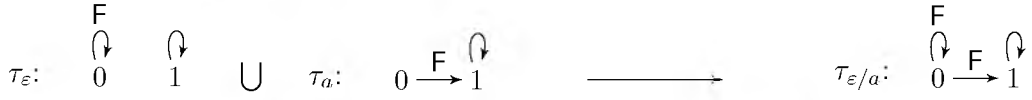


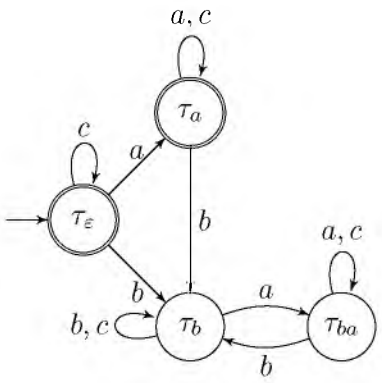
Figure 4.6: The merged transition profile $\tau_{\varepsilon/a}$

We continue with examining if there is no accepting loop in $(\{1\}, \tau_{\varepsilon/a})$, since $\{1\}$ is the only state set either τ_ε or τ_a is completely looping safe with. As we just reach a non-accepting self loop in 1, this property holds and therefore τ_ε and τ_a are mergeable. So, the desired partition only consists of $MTP_1 = \{\tau_\varepsilon, \tau_a\}$. Concerning to the structure of the input automaton $\mathfrak{A}$ we have now combined the two looping runs that causes $\mathfrak{A}$ to stay in the non-accepting state 1, which have been before regarded separately by these two idempotent transition profiles.

4.3.4 Complement Automaton

In order to construct the final complement automaton, we first want to minimize the only SPA, the one of $\tau_{\varepsilon/a}$ (cf. Figure 4.7), and then test if the ω-operation is absolutely needed to repeat the words properly.

SPA($\mathfrak{A}$):



Minimized SPA($\mathfrak{A}$):

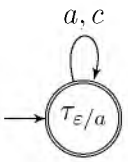


Figure 4.7: SPA($\mathfrak{A}$) and its minimization

Remark 4.7 provided the simple condition that there is no need for the ω-operation if every outgoing transition of the initial state is also an outgoing transition of each final state, leading to the same states. As we only have one state here, which is initial and accepting, this property is fulfilled, so it is not necessary to apply the ω-operation. Adding the linking transition then leads us to the final complement automaton $\overline{\mathfrak{A}}$ given in Figure 4.8. This example results in a quite small complement

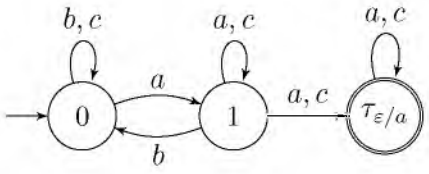


Figure 4.8: Complement automaton $\overline{\mathfrak{A}}$

automaton, being only one state apart from the minimal complement (we leave the declaration of this to the reader). Though, one optimization of the above, has not been applicable here, namely merging ω-equivalent states to reduce the number of states in $PA_\emptyset$. We will provide a corresponding example in the case studies of the upcoming section.

5 Experimental Analysis

Up next in this last chapter we are interested in how good our complementation algorithm works. First we are looking at some constructed cases, revealing the function of the algorithm in some interesting as well as in problematic cases. Then we move on to a performance test, where the algorithm is given random input automata with different parameters. Overall we compare our results to the recent constructions ([Pit07, Sch09, VW07]) all implemented in the GOAL tool [TCT+08], naming them deterministic ([Pit07]), rank-based ([Sch09]) and slice-based ([VW07]) construction, according to their corresponding approach. As these were last examined in the work [TFTV10], we try to guarantee the same boundary conditions, but we will discuss this in more detail in the corresponding section.

5.1 Case Studies

Case 1

We first want to look at a very simple automaton over $\Sigma = \{a, b\}$, recognizing all the words that start with b. To make it a bit more difficult, we choose a more complex automaton of this language as input for the algorithm (cf Figure 5.1). With the corresponding set $ITP = \{\tau_a, \tau_b, \tau_{ab}, \tau_{ba}, \tau_{bab}\}$, as well as 13 pairs in Dng, and 2 in Sfe, but none in LSfe or CLSfe, we see that we gain the trivial case c), described in Lemma 3.5: Because of LSfe being empty and Sfe being non-empty, the result is the automaton $PA_\emptyset$, which is reduced before put out (cf. Figure 5.2). Although the input automaton is a bit complex, we get a result that is similar to the minimal complement, in this case, even outranking the complement of the deterministic construction (8 states) and the ones of the rank-based and slice-based (4 states). This even holds for all languages of form $w\Sigma^\omega$, $w \in \Sigma^+$, as checking the correctness of a finite prefix is done only in the U-component, which is described by $PA_\emptyset$.

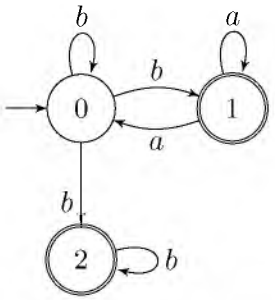
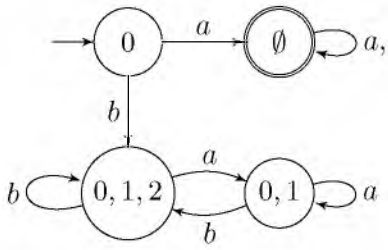


Figure 5.1: Input automaton $\mathfrak{A}_1$

$\mathrm{PA}_\emptyset(\mathfrak{A}_1)$:



$\overline{\mathfrak{A}_1}$:

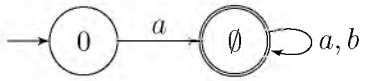


Figure 5.2: $\mathrm{PA}_\emptyset(\mathfrak{A}_1)$ and its minimization, which is $\overline{\mathfrak{A}_1}$

Concerning to the structure of $PA_\emptyset$ we see that it always has only one positive sink for the accepting state set. These languages are known to be E-recognizable, accepting a word iff a run leads to state of F is reached. It is known that a language is E-recognizable iff the complement of this language is A-recognizable, accepting a word iff a run only stays in states of F. So the question if our algorithm results in $PA_\emptyset$ for every A-recognizable language, not depending on the structure of the corresponding automaton, may be interesting to find out and will be left as future work.

Case 2

With the automaton given in Figure 5.3 we want to provide a small example, in which the state space of $PA_\emptyset$ is reduced by merging ω-equivalent states. After building $PA_\emptyset(\mathfrak{A}_2)$, $\mathrm{SPA}(\mathfrak{A}_2)$, ITP, Dng, Sfe, LSfe, CLSfe and testing the need for ω-operations, we get the intermediate result at the left-hand side of Figure 5.4. By the linking transitions, we directly see that both states in the powerset automaton are completely looping safe with the two transition profiles τ_a and τ_b. As they are equivalent by the common definition, too, the states are ω-equivalent and therefore can be merged. The result can be seen on the right-hand side of Figure 5.4.

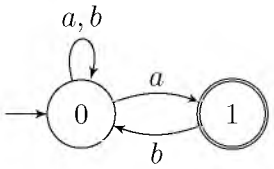
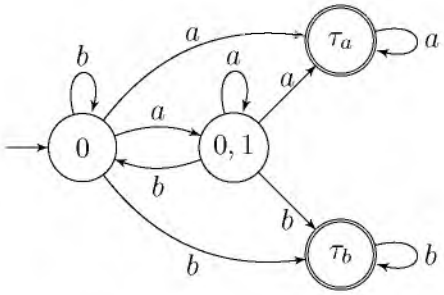


Figure 5.3: Input automaton $\mathfrak{A}_2$

$\overline{\mathfrak{A}_2}$ before merge:



$\overline{\mathfrak{A}_2}$ after merge:

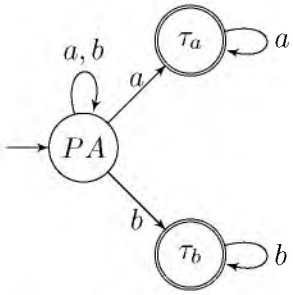


Figure 5.4: $\overline{\mathfrak{A}_2}$ before and after merge of ω-equivalent states

The idea behind this, referring to the structure of the language, is that we have to read from some point on only a or only b in order to get a word of the complement, which means staying from some point on in the state 0. Since we have a self-loop in this state, it is guaranteed that we are never leaving it. So, in this context it does not matter if we have already read a, which lead us to the state set $\{0, 1\}$ regarding all possible runs, or are in the state 0 when we reach the point from which we only read a or b. The number of states of the other constructions are 7 (deterministic) and 6 (rank- and slice-based), so again we perform best here.

Case 3

The next case reveals two interesting points. First, we show that the number of states of the result automaton depends on the partition of TP_{CLSfe} we choose, when we unite mergeable transition profiles. Second, this case turns out to be a bit problematic for our algorithm, as the complement automata of the other constructions, especially the slice-based, are resulting in a lower number of states. The input automaton is shown in Figure 5.5. The language is given by $L(\mathfrak{A}_3) = ((b + c + ab)^* ac)^\omega$, i.e. a word is in $L(\mathfrak{A}_3)$ iff it contains infinitely often the infix ac, but never the infix aa. For the complement language $\overline{L(\mathfrak{A}_3)}$ this means that it contains a word iff it contains the infix ac only finitely often or has the infix aa. We leave the direct construction of an intuitive complement automaton to

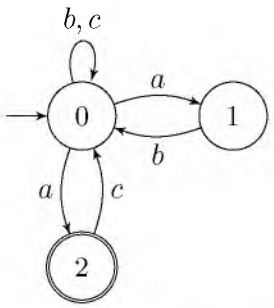
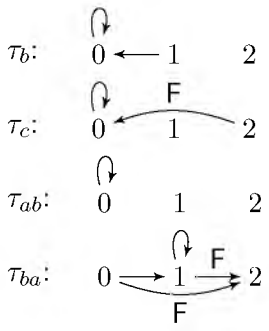


Figure 5.5: Input automaton $\mathfrak{A}_3$

the reader, but want to note explicitly that although the complement language is quite intuitive, our algorithm has its problems with it. Looking at the structure of $\mathfrak{A}_3$ it is remarkable that there are four possibilities to loop in a state without accepting, namely via two self-loops in state 0, starting in 0 reading ab and starting in 1 reading ba over and over again. This leads to four completely looping safe pairs, with $TP_{CLSfe} = \{\tau_b, \tau_c, \tau_{ab}, \tau_{ba}\}$, all describing one of this runs from the corresponding state set. In Figure 5.6 we provide the graphs of TP_{CLSfe}, describing the elements of each and give only the interesting pairs of Dng, Sfe, LSfe, CLSfe. By this, we can define which transition profiles are mergeable, by looking for



Dng	Sfe	LSfe	CLSfe
⋮	$(\{0\}, \tau_{ba})$	$(\{0\}, \tau_{ba})$	$(\{0\}, \tau_{ab})$
	$(\{0\}, \tau_{ab})$	$(\{0\}, \tau_{ab})$	$(\{0\}, \tau_b)$
	$(\{0\}, \tau_b)$	$(\{0\}, \tau_b)$	$(\{0\}, \tau_c)$
	$(\{0\}, \tau_c)$	$(\{0\}, \tau_c)$	$(\{1\}, \tau_{ba})$
	$(\{1\}, \tau_{ba})$	$(\{1\}, \tau_{ba})$	
	$(\{1\}, \tau_{ab})$	$(\{1\}, \tau_b)$	
	$(\{1\}, \tau_b)$	$(\{1\}, \tau_c)$	
	$(\{1\}, \tau_c)$		
	⋮		

Figure 5.6: TP_{CLSfe} and the pairs of them in Dng, Sfe, LSfe, CLSfe

accepting loops. It turns out that only τ_{ba} and τ_c are not mergeable, as their union leads to an accepting loop between 0 and 2. This is also clear when looking at the input automaton, as starting in 0 reading the sequence $bacbac\ldots$ yields an accepting run of $\mathfrak{A}_3$, visiting state 2 infinitely often. Now, there exist many

possible partitions of TP_{CLSfe}. To itemize all of them would be tedious, so we only want to look at the both, a greedy algorithm finds, depending on the order in which the transition profiles are considered. The first one is $\{\{\tau_b, \tau_c, \tau_{ab}\}, \{\tau_{ba}\}\}$, leading to a complement automaton with 7 states. In fact, this is the partition our Algorithm 6 returns. Though, the second partition, we want to look at is given by $\{\{\tau_b, \tau_{ba}, \tau_{ab}\}, \{\tau_c\}\}$ and results in a final complement automaton with 11 states. As we are not that interested in how the complement automaton exactly looks like we omit them at this point. We only want to take down, that the number of states in the output automaton depends on the chosen partition, even when one performs a greedy search. As already mentioned, the other constructions mainly perform better. The deterministic approach leads to 9, the rank-based to 6 and the slice-based to 5 states.

5.2 Performance

Finally, we now want to examine our complementation algorithm by performing an experimental analysis. In order to do so, we implemented all the described procedures and their optimizations, as well as a generator for random automata. We orientate ourselves by the work [TFTV10], providing parameters for the number of states (n), a certain transition density (t) and an acceptance density (f). Just like in [TFTV10] we want to fix the alphabet size to two. A random automaton, then consists of n states (one being initial), up to $\lceil tn \rceil$ random transitions for each letter and up to $\lceil fn \rceil$ random accepting states. Of course, this can also result in rather useless automata that e.g. have no infinite run at all. Although we recognize such cases, putting out a universal or empty automaton, respectively, these examples bias the average number of all complement automata. For this reason we always provide the average number of states both with and without the trivial outputs taken into account.

Build an own test set similarly to the one in Tsai, 2010

In a first step our aim is to determine suitable densities t and f that stand for easy and hard complementation tasks. Keeping one variable on a constant, intuitively chosen value we vary the other one and look how this changes the average number of states of the complement automata. As it is also interesting to see how good our presented optimizations work, we here want to compare the results of our origin Algorithm 4 directly to the ones of the optimized Algorithm 6. In Figure 5.7 we provide the results of our first analysis, where "avg(Alg.)" stands for the overall average of number of states in the output automaton by using the stated algorithm, "trivial" stands for the number of trivial cases, in which the output automaton has

non-trivial cases

tasks	n	t	f	avg(Alg.4)	avg(Alg.6)	trivial	n-t(Alg.4)	n-t(Alg.6)
1,000	5	1.5	0.2	1,571.57	14.27	328	2,391.52	20.31
1,000	5	1.5	0.4	328.40	7.92	274	454.47	10.48
1,000	5	1.5	0.5	105.76	5.89	303	149.17	8.12
1,000	5	1.5	0.9	32.37	4.20	346	49.79	5.81
1,000	5	1.0	0.5	70.70	5.04	400	114.70	7.86
1,000	5	1.5	0.5	170.99	6.45	300	241.78	8.82
1,000	5	2.0	0.5	32.14	3.47	384	51.89	4.99
1,000	5	2.5	0.5	9.31	2.09	569	20.19	3.54

Difficulaties: similar picture than with our experiments in GOAL

Figure 5.7: Comparision of original/optimized algorithm and effect of densities

only one state, and "n-t(Alg.)" then stands for the average number of all outputs that are not trivial. Three simple observations can be made. First of all, the unmodified Algorithm 4 really leads to poor results, whereas the optimized version, performs very much better. We discuss this in more detail later (cf. Figure 5.8). Talking about the densities we can deduce that with rising f (which means more and more states become accepting), the difficulty level of the complementation task and so number of states shrink. This is clear from looking at the extreme cases. With nearly all states being accepting, there are hardly completely looping safe pairs (describing a stay in a non-accepting state set) and therefore less SPAs are needed. If only one certain state is accepting we have a few particular accepting runs, but many different possibilities to stay in a set of non-accepting states, which results in more needed SPAs. The transition density t also provides different levels of difficulty. A low density leads to fewer possible runs, as well as a high density results in a great coverage of the states and therefore to a rough separation of all possible runs. The complex cases lie in between, as they yield evenly distributed runs. It is clear that the number of trivial cases increases when we come closer to the extreme cases (high/low f/t). We define three levels of difficulty for a complementation task: easy ($t = 2.0, f = 0.8$), medium ($t = 1.0, f = 0.5$), hard ($t = 1.5, f = 0.3$).

Matches very well with the results of Christian, even though it's a different algorithm

At this point, we want to admit that there has been a large variance in the output sizes of the unmodified algorithm, caused by sometimes occurring extreme outliers (18,000-30,000 states). Figure 5.7 still leads to interpretable values but when we now want to talk about the different optimizations, it is necessary that here all algorithms run on the same tasks, what we have guaranteed when getting the results of Figure 5.8. Now, listing the optimizations separately with fixed medium level of difficulty (cf. Figure 5.8), we can derive many interesting points. In the first column of Figure 5.8 the used optimization is given, where again

medium test automata

Optimization	tasks	n	t	f	avg(states)	trivial	n-t(states)
none	1,000	5	1.0	0.5	62.85	429	109.32
reduce SPA	1,000	5	1.0	0.5	8.09	429	13.42
reduce PA+SPA	1,000	5	1.0	0.5	7.30	429	12.04
merge SPAs	1,000	5	1.0	0.5	31.39	429	54.22
test ω-op.	1,000	5	1.0	0.5	62.85	429	109.32
merge+red. SPAs	1,000	5	1.0	0.5	6.06	429	9.87
m.+r. SPAs+ω-op.	1,000	5	1.0	0.5	6.06	429	9.87
all	1,000	5	1.0	0.5	5.29	429	8.51

Figure 5.8: Effect of different optimizations with same tasks

"avg(states)" stands for the average number of output states and "n-t(states)" for the corresponding number without taking trivial outputs into account. We do not provide results for all possible combinations of optimizations, but concentrate on the most interesting ones. First of all, we dissociate each optimization from the unmodified algorithm. Minimizing the SPAs yields to the largest savings, which is clear, hence there are so many useless branches in the automaton SPA and also so many copies of them. Talking about the copies we have to go into the effect of merging those. It performs not as well as minimizing, but still gives us a rough 50 percent cut-off. The explanation is that we always have all possible states in the SPA (up to 4^{n^2}) but less often all possible copies (also up to 4^{n^2}), as some define accepting looping runs of the input automaton. Furthermore after merging SPAs, the state space is still left untouched. We look at a combination of these both optimizations in a second step, giving us the result that the merge of SPAs is leading to smaller sizes, anyway. Now back to the two remaining optimizations. Since we connected the reduction of PA and the SPAs in our actual implementation, we have to look at the effect of those together. But in spite of that, we see that the merging of ω-equivalent still yields to a slightly better result. The test for the ω-operation in fact leads to no reduction of the state space. We have checked that in several runs but only got very, if any, insignificantly savings. The reason is that we just provided a really small test, only catching the simplest cases. But as these cases, e.g. the SPA only consists of one state (cf. Figure 4.2, 4.7) could be assumed to happen more often when we first merge and minimize the SPAs we give the ω-operation a second chance by combining it with these optimizations. But as can be seen, this also fails, so those simple cases must be appearing very rarel. So, the ω-examination just prevents defective appearances of small, constructed examples.

Now that we have noticed all that, we continue the analysis only with the opti-

mized Algorithm 6. Here we want pick up on the outliers again, just giving the remark that there has been no such large discrepancy in the output sizes, as the optimizations have performed very well in reducing them. So without compunction, we can now test this algorithm with all three levels of difficulty and three different sizes of the input automata. The result can be seen in Figure 5.9.

level	tasks	n	t	f	avg(Alg.6)	trivial	n-t(Alg.6)
easy	10,000	5	2.0	0.8	3.03	4141	4.46
medium	10,000	5	1.0	0.5	5.77	4076	9.05
hard	10,000	5	1.5	0.3	9.68	2998	13.39
easy	1,000	10	2.0	0.8	4.81	336	6.73
medium	1,000	10	1.0	0.5	18.26	487	34.19
hard	1,000	10	1.5	0.3	53.35	311	75.79
easy	100	15	2.0	0.8	4.84	37	7.10
medium	100	15	1.0	0.5	23.20	23	37.50
hard	100	15	1.5	0.3	74.10	11	137.56

Figure 5.9: Testing three level of difficulties with rising input size

The main purpose of this test is to give a overview of the performance our algorithm has. It is not surprising that with higher level of difficulty the resulting number of states grows. We already discussed that with the first test and Figure 5.7. But it can be seen that all three input sizes are performing well with easy complementation tasks. With higher level the trivial cases decrease, also resulting in more states of the output. Choosing a medium level of difficulty, the number of states in the output already goes up with rising input size. For hard complementation tasks it is even worse. The more states we have, the more runs are possible, and especially more state sets can be reached, increasing size of both PA and SPA. The expansion rate from one level to another also grows when the input size gets bigger. This is also the reason for taking less tasks with rising task sizes. With more states in the input automaton, the running time becomes extremely high, even leading to timeouts and memory-errors, especially in hard inputs. This is mainly because of the construction of the SPA, which is the step in the algorithm that costs much effort, as it has to concern all reachable state sets, starting from any state in the input automaton. Surely, this is still one main disadvantage of the algorithm, already pointed out in [TFTV10]. The question if there are any further optimizations possible here, too, may remain as future work. Here, we only concentrate on finished examples, although it would be interesting as well (also comparing to the other constructions) how many timeout- or memory-errors we have, like it is done in the work [TFTV10].

As already mentioned, the "state of Büchi complementation" was last examined in the same-titled work [TFTV10], comparing the recent constructions ([Pit07, Sch09, VW07]). We now want to compare some of our results to these ones, as far as possible, as we do not know the exact boundary conditions [TFTV10] used in the analysis, mainly the values of the transition density t and the acceptance density f. Furthermore, only those complementation tasks are considered in [TFTV10] that have been effective with each construction, i.e. have been solved by all constructions. Comparing our results to those we have to take all this into consideration. The resulting output size given in the column "avg(states)" in Figure 5.10 has

Construction	tasks	n	avg(states)[n-t]
easy(Alg.6)	10,000	15	4.84 [7.10]
medium(Alg.6)	1,000	15	23.20 [37.50]
hard(Alg.6)	100	15	74.10 [137.56]
deterministic	10,977	15	58.72
deterministic+opt	10,977	15	37.47
rank-based	5,697	15	33.96
rank-based+opt	5,697	15	28.41
slice-based	4,514	15	70.67
slice-based+opt	4,514	15	36.11

Figure 5.10: Direct comparison of all constructions

taken trivial cases into account. The corresponding numbers to only non-trivial outputs are given in [...] for our algorithm. The work [TFTV10] also provided several optimizations for the other three approaches. We are referring to these with "+opt" in Figure 5.10. We want to note that we simply copied the values from [TFTV10], that compare each original construction with its optimization, as this provides an isolated view on each construction. But it shall be stated that the deterministic construction by Piterman [Pit07] trumps the others ([Sch09, VW07]) in a direct comparison. Such an examination has not been possible for us and shall be left for future work. As pointed out by the remarks given above the comparison is only of little avail. Although, it allows us a first rough classification of our optimized algorithm compared to the recent constructions. Looking at the tasks of a medium difficulty level, as this is an appropriate average, the output size of our construction is even more than just drawing level with the other constructions. If we gather the sizes that does not allow for the trivial cases, we are still meeting the other ones. All in all it can be pointed out that our optimized algorithm can now at least keep up with the number of states in the resulting complement automaton.

6 Conclusion

In this bachelor thesis we revisited the issue-area of Büchi complementation, which reaches back to the 1960s. After given an introduction of the three major approaches and their development throughout the centuries, we presented the basic definitions that are needed in order to construct a complementation algorithm in Section 2. This construction is based on the algebraic approach, Büchi used to prove the closure under complement of his automaton model [Büc62] and which has the worst asymptotic behavior compared against the other approaches. It makes use of an equivalence relation $\sim_{\mathfrak{A}}$ on the state set of a given automaton $\mathfrak{A}$, to capture all possible runs of $\mathfrak{A}$. We found that $\sim_{\mathfrak{A}}$ is a congruence with finitely many equivalence classes, called transition profiles (τ), each of them defining a regular language. A special subset of these is formed by the idempotent transition profiles (ITP), which covers all looping runs of $\mathfrak{A}$. The simultaneous powerset automaton (SPA), whose definition was provided at the end of Section 2 takes all transition profiles as the set of states. It can be used to recognize the language of a certain τ, by marking it as only accepting state.

With these definitions we were able to make use of Ramsey Theorem in Section 3, which points out that with a given congruence on the state set of an automaton, any word $w \in \Sigma^{omega}$ can be split up in two components $w \in U \cdot V^{\omega}$, where $U, V \in \Sigma^*$ are languages of a certain equivalence classes of the congruence. As $\sim_{\mathfrak{A}}$ defines such a congruence, it was possible for us to decompose any infinite word in the way $w = uv_0v_1\ldots$, with U and V being languages of certain transition profiles. But as the U-component just stands for a finite prefix, we chose the powerset automaton (PA) in order to recognize this part, where the accepting state set is changed to the sink-state, if existent. For the V-component we aimed for using various SPAs, namely the ones of idempotent transition profiles as these were the ones that define looping runs on $\mathfrak{A}$. But in fact, only those elements of ITP are needed that describe a non-accepting run. We found those by forming pairs of reachable state sets in PA and idempotent transition profiles, denoting them as safe or dangerous, corresponding to the runs they define. By the presented ω-operation we guaranteed that the words of each SPA are repeated properly, and with some linking transitions that connect the models for the U- and the V-

component we were able to present a first complementation algorithm, together with a first example.

As this algorithm lead to many superfluous states, we examined possible optimizations and heuristics in Section 4. First, we noted that the SPAs for the V-component are deterministic automata [07]) therefore can be minimized before applying the ω-operation, with the common procedures like the DFA-minimization provided in [VL08]. After minimizing the models of the V-component we looked for SPAs that can be merged, without changing the complement language. That is when in the union of the corresponding transition profiles, no new accepting runs arise, meaning that the input automaton does not gain new accepting runs by reading the words of both transition profiles in an arbitrary order. We determined the partition of all needed transition profiles, containing sets of pairwise mergeable ones, by a greedy algorithm. A further optimization took the PA for the U-component in consideration. We defined an ω-equivalence concerning to the states of PA, which is applicable to two states, if they are equivalent from the common point of view and have the same meaning for the whole complement automaton, i.e. being safe with the same SPAs. Merging those ω-equivalent states also resulted in a reduction of the state space of the complement automaton. The last optimization given in Section 4 was a small embryonic test for the need of the ω-operation, as sometimes its application leads to new superfluous states. All those four optimizations lead us to an optimized version of our already constructed complementation algorithm, whose function was illustrated in an example, too.

In the last Section 5 we performed an experimental analysis, starting with a discussion of two interesting and one problematic case studies. The latter one yielded different output sizes, depending on the order in which we merge suitable transition([Pit07s and here our algorithm mainly was inferior to other constructions ([Pit07, Sch09, VW07]). The other two cases were solved, resulting in the intuitive minimal complement. We moved on with testing our implementation with a number of randomly generated automata. By determining values for the both parameters, namely transition density t and acceptance density f, looking at their effect on the output size, we defined three different levels of difficulty of a given complementation task. We compared our unmodified algorithm with the optimized version leading to the unsurprising fact that latter one performed much better. In the next step the effect of rising input size was investigated, giving quite good results for easy tasks, but showed that with a higher level of difficulty and especially a high number of states in the given task, not only the output size but also the running time highly increases. Compared to the already mentioned constructions [Pit07, Sch09, VW07], we noted that our algorithm was able to yield comparable

results.

Although, from performing quite good in the experimental analysis, the full potential of our construction has not been tapped yet. Throughout this work, several problems have been suggested as future work. What should be done first of all, is to examine the potential of our complementation algorithm in a direct comparison to the recent implementations, because it was not possible to find out, which parameters have been chosen in the work [TFTV10]. It also analyzes the effect of a preminimization [SB00] which we have not considered yet in our case. Furthermore, minimizing our output with [SB00] or other minimization strategies for Büchi automata, to investigate the size difference of a further reduced automaton, would be nice to know. Concerning to the algorithm itself, it also would be interesting to look for further optimizations in building the SPA, as this is the step that leads to the high running time. As many branches are cut off, anyway in this automaton one could search for a test that recognizes these branches, before considering the whole set of transition profiles. Maximizing the accepting state set before using our algorithm could result in a reduction in the state space, too, as more accepting states lead to more dangerous pairs and so, to less needed SPAs. Talking about those, we also have to mention that a complete complexity classification of the SPA-merging-problem is left open. The two phenomena, if every A-recognizable language comes to a small output only consisting of the modified powerset automaton (cf. Section 3), as well as that there is no difference in the size of a SPA, either with DFA-minimization or NFA-reduction applied (cf. Section 4), remain as future work.

So, this work offers some further steps but laid the foundation for several optimization techniques in the algebraic approach of complementing Büchi automata.

Bibliography

[ATW06] Althoff, Thomas, and Wallmeier. Observations on Determinization of Büchi Automata. *TCS: Theoretical Computer Science*, 363, 2006.

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

[DR09] Laurent Doyen and Jean-François Raskin. Antichains for the Automata-Based Approach to Model-Checking. *Logical Methods in Computer Science*, 5(1), 2009.

[FKV06] Friedgut, Kupferman, and Vardi. Büchi Complementation Made Tighter. *IJFCS: International Journal of Foundations of Computer Science*, 17, 2006.

[FV09] Seth Fogarty and Moshe Y. Vardi. Büchi Complementation and Size-Change Termination. In *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2009.

[FV10] Seth Fogarty and Moshe Y. Vardi. Efficient Büchi Universality Checking. In *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2010.

[GKSV03] Sankar Gurumurthy, Orna Kupferman, Fabio Somenzi, and Moshe Y. Vardi. On Complementing Nondeterministic Büchi Automata. In *CHARME*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer, 2003.

[Kla91] Nils Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *SFCS '91: Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 358–367. IEEE Computer Society, 1991.

[KV01] Kupferman and Vardi. Weak Alternating Automata Are Not that Weak. *ACMTCL: ACM Transactions on Computational Logic*, 2, 2001.

[KV05] Kupferman and Vardi. Safraless Decision Procedures. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2005.

[KW08] Detlef Kähler and Thomas Wilke. Complementation, Disambiguation, and Determinization of Büchi Automata Unified. In *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 724–735. Springer, 2008.

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

[MP95] O. Matz and A. Potthoff. Computing small nondeterministic automata. In *Proc. Workshop on Tools and Alg's for the Cons'n and Analysis of Sys's.*, BRICS Notes, 1995.

[MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1–2):69–107, 1995.

[Pit07] Nir Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. *CoRR*, abs/0705.2205, 2007.

[Pra92] Ira Pramanick. Application of a Parallel Heuristic Framework to the Set Covering Problem. In *ICPP (3)*, pages 185–189, 1992.

[Ram29] F. P. Ramsey. On a problem of formal logic. *London Math. Society*, 30:264–286, 1929.

[Roe08] Andreas Roell. *Complementation of Büchi Automata Algorithms and Implementations.* Diploma thesis, RWTH Aachen, 2008.

[RS59] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3:114–125, 1959.

[Saf88] Shmuel Safra. On the Complexity of omega-Automata. In *FOCS*, pages 319–327. IEEE, 1988.

[SB00] Somenzi and Bloem. Efficient Büchi Automata from LTL Formulae. In *CAV: International Conference on Computer Aided Verification*, 2000.

[Sch09] Sven Schewe. Büchi complementation made tight. *CoRR*, abs/0902.2152, 2009.

[SVW87] Sistla, Vardi, and Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *TCS: Theoretical Computer Science*, 49, 1987.

[TCT+08] Yih-Kuen Tsay, Yu-Fang Chen, Ming-Hsien Tsai, Wen-Chin Chan, and Chi-Jian Luo. GOAL Extended: Towards a Research Tool for Omega Automata and Temporal Logic. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 346–350. Springer, 2008.

[TFTV10] Ming-Hsien Tsai, Seth Fogarty, Yih-Kuen Tsay, and Moshe Y. Vardi. State of Büchi Complementation (Full Version). In *CIAA*, 2010.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, volume B, Formal models and semantics, pages 133–191. Elsevier, 1990.

[Tho99] Wolfgang Thomas. Complementation of Büchi Automata revisited. In *Jewels are forever – Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122. Springer, 1999.

[Var07] Moshe Y. Vardi. The Büchi Complementation Saga. In *STACS*, volume 4393 of *Lecture Notes in Computer Science*, pages 12–22. Springer, 2007.

[VL08] Antti Valmari and Petri Lehtinen. Efficient Minimization of DFAs with Partial Transition Functions. *CoRR*, abs/0802.2826:645–656, 2008.

[VW07] Moshe Y. Vardi and Thomas Wilke. Automata: From Logics to Algorithms. In *Logic and Automata: History and Perspective*, volume 2 of *Texts in Logic and Games*, pages 629–736, 2007.

[Yan08] Qiqi Yan. Lower Bounds for Complementation of omega-Automata Via the Full Automata Technique. *CoRR*, pages 1–20, 2008.