Complementation of Büchi Automata via Non-Determinism Reduction

Master Thesis

Mathias Ølgaard

Universität Bern

August 2025

Abstract

This thesis introduces an alternative route to complementation of Büchi automata. By first reducing the automaton we want to complement to a non-determinism degree of two, we facilitate a simpler construction of its complement. The resulting method is derived from a general complementation algorithm, adapted to exploit the reduced non-determinism. We present the details of this construction, analyze its computational complexity, and compare its performance and conceptual simplicity to that of the conventional, direct complementation technique.

Ulrich Ultes-Nitsche, Foundations of Dependable Systems, Département d'informatique, Université de Fribourg, Supervisor

Dorian Guyot, Foundations of Dependable Systems, Département d'informatique, Université de Fribourg, Assistant

Content

1 Introduction

The background for the master project presented in this thesis is the process of complementing Büchi automata, and more specifically, the complementation algorithm described by Allred & Ultes-Nitsche in their paper: "*A Simple and Optimal Complementation Algorithm for Büchi Automata*"[1].

Together with Ultes-Nitsche's paper "*A power-set construction for reducing Büchi automata to non-determinism degree two*"[2], it sets the stage for the research question of the project:

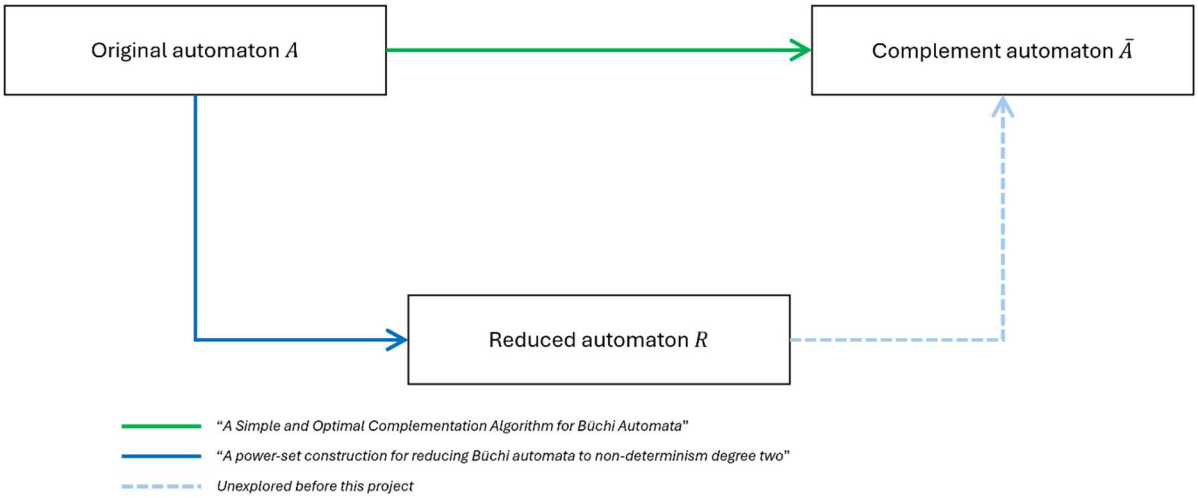

Figure 1: The two routes to complementation, respectively colored green and blue. The complexity of the solid lines is known from [1], [2]. The dashed line represents the area of research for this project.

How does the complementation process look like if one assumes that the input automaton is already reduced to non-determinism degree 2 as described in [2]? And is it somehow beneficial to reduce a Büchi automaton before performing the complementation?

Explained more visually, the goal of the project has been to explore the complexity of the dashed line in Figure 1, and to then compare the two routes to complementation, respectively the green and the blue route in Figure 1.

In parallel with researching this problem, I have developed a Python implementation for conceptualization and visualization of Büchi Automata and algorithms performed on them. The code was used several times during the project to identify example automata for different criteria.

2 Preliminaries

2.1 Büchi Automata

Let Σ be a finite set of symbols, called an alphabet, and let Σ^ω denote the set of all infinitely long strings over Σ. We call these infinite strings ω-words over Σ and define ω-languages as sets of ω-words. In other words, ω-languages over Σ are subsets of Σ^ω. For an ω-word x and an ω-language L, we have:

$$x \in \Sigma^\omega, \qquad L \subseteq \Sigma^\omega$$

A non-deterministic Büchi automaton (NBA) is a finite-state machine whose input are ω-words. It is defined by a set of states Q, an alphabet Σ, a transition function $\delta: Q \times \Sigma \to 2^Q$, an initial state $q_{in} \in Q$, and a set of accepting states $F \subseteq Q$:

$$A \coloneqq (Q, \Sigma, \delta, q_{in}, F)$$

A deterministic Büchi automaton (DBA) is a special case of an NBA, where the transition function δ is deterministic, i.e. $|\delta(q,a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. For DBAs, the transition function's codomain can be defined as Q rather than the power-set 2^Q, i.e. $\delta: Q \times \Sigma \mapsto Q$.

A run $r \coloneqq r_0 r_1 r_2 \ldots \in Q^\omega$ of a Büchi automaton A on an ω-word $x \coloneqq x_0 x_1 x_2 \ldots \in \Sigma^\omega$ is an infinitely long sequence of states, such that $r_0 = q_{in}$ and $r_{i+1} \in \delta(r_i, x_i)$ for all $i \geq 0$. A run of a Büchi automaton is accepting if and only if there are infinitely many integers $n \geq 0$, such that r_n is an accepting state, i.e.

$$r \; is \; accepting \Leftrightarrow \exists^\infty n \geq 0 \; (r_n \in F)$$

We say that a Büchi automaton A accepts an ω-word x if and only if there exists an accepting run of A on x. The ω-language accepted by A, $L_\omega(A)$, is the set of all ω-words that A accepts. For example, the Büchi automaton in Figure 2 accepts the following ω-language:

$$L_\omega(A) = (b(a|b)b^*a^+)^\omega$$

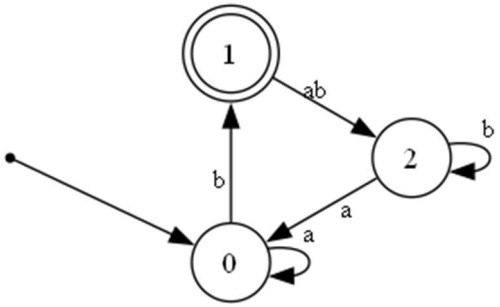

Figure 2: $Q = \{0,1,2\}, \Sigma = \{a,b\}$, $q_{in} = 0$, $F = \{1\}$

2.2 Reduction of non-determinism

A Büchi automaton's non-determinism degree is defined as the maximum number of different target states that can be reached from a single source state $q \in Q$ reading a single symbol $a \in \Sigma$:

$$nd(A) \coloneqq \max_{q \in Q, a \in \Sigma} |\delta(q,a)|$$

Karpiński showed in [3] that any Büchi automaton can be reduced to non-determinism degree 2. In other words, for any Büchi automaton A, a corresponding Büchi automaton R can be constructed such that $nd(R) \leq 2$ and $L_\omega(R) = L_\omega(A)$.

For the purpose of this thesis, such automata $R \coloneqq (Q_R, \Sigma, \delta_R, q_R, F_R)$ will be constructed as shown in [2]. In essence, R is constructed by an adjusted power-set construction, where the state base of R is $2^{Q \setminus F} \cup 2^F$ instead of 2^Q, which would be the result of the standard power-set construction. This is achieved by splitting the target states of a transition $\delta(q,a)$ into two sets, respectively containing the non-accepting and the accepting target states:

$$\delta_n(q,a) \coloneqq \delta(q,a) \cap Q \setminus F,$$

$$\delta_a(q,a) \coloneqq \delta(q,a) \cap F.$$

This ensures that any state in R contains only non-accepting states or only accepting states:

$$\forall q' \in Q_R: \; (q' \subseteq Q \setminus F) \vee (q' \subseteq F),$$

As every state $q' \in Q_R$ have one successor state representing $\delta_n(q',a)$ and another representing $\delta_a(q',a)$ for all symbols $a \in \Sigma$, the non-determinism degree of R is guaranteed to be at most 2:

$$nd(R) \leq 2$$

It is not necessarily equal to 2, as either δ_n or δ_a might output the empty set, which will add no new state to the automaton R. For example, if the original automaton A contains only accepting states, i.e. $F = Q$, then we have $nd(R) \leq 1$, because $Q \backslash F = \{\}$ and δ_n will output the empty set regardless of the input state and symbol. If A has no transitions, we have $nd(R) = 0$.

2.3 Complementation algorithm

2.3.1 Greedy runs

The complementation algorithm described in [1] is based on the idea of *greedy* runs. A run is said to be greedy if it always reaches the next accepting state as quickly as possible.

To define greediness formally, we use a function $f: Q \mapsto \{0,1\}$ that assigns 1's to accepting states and 0's to non-accepting states:

$$f(q) \coloneqq \begin{cases} 1, & q \in F \\ 0, & q \in Q \backslash F \end{cases}$$

Then, we define $f_k: Q^\omega \mapsto \{0,1\}^k$ for all $k \in \mathbb{N}$ such that

$$f_k(q_0 q_1 q_2 \dots) \coloneqq f(q_0) f(q_1) \dots f(q_{k-2}) f(q_{k-1})$$

Now, $f_k(r)$ can be interpreted as a binary integer with k digits, where $f_k(r) > f_k(r')$ implies that r reached the next accepting state earlier, i.e. with fewer steps than r'. Therefore, we define a run r on an ω-word x as greedy if and only if for all other runs r' on x, we have:

$$\forall k \in \mathbb{N}: \ \left(r_{k-1} = r'_{k-1} \Rightarrow f_k(r) > f_k(r')\right)$$

In other words, an infinite run is greedy if and only if all of its finite prefixes (of size k) are greedy.

The reason why greedy runs are important is that we can effectively ignore all other runs when checking if an ω-word is accepted by a BA. This is a consequence of Lemma 2.1 in [1]:

Lemma 2.1. *There exists an accepting run of A on x if and only if there exists a greedy accepting run of A on x.*

A corollary following from this lemma would be:

Corollary 2.1.1. *If there is no greedy accepting run of A on x, then A does not accept x. Therefore, the complement of A accepts x.*

2.3.2 Construction of $\bar{A}$

Motivated by the aforementioned corollary, Allred and Ultes-Nitsche's complementation algorithm constructs a Büchi automaton $\bar{A}$, systematically keeping track of all greedy runs of A. The idea is that $\bar{A}$ should accept ω-words, on which each greedy run of A visits accepting states only finitely many times. That is, if all runs of A on an ω-word x stops visiting accepting states – or becomes non-greedy – after finitely many steps, then $\bar{A}$ accepts x.

The construction of $\bar{A}$ consists of two deterministic "sub-automata"; the *upper automaton* $\hat{A}$ and the *lower automaton* $\check{A}$. The complement automaton $\bar{A}$ allows for a non-deterministic "jump" from $\hat{A}$ to $\check{A}$, which is mandatory for any accepting run of $\bar{A}$, since there are no accepting states in $\hat{A}$. Details on how the two sub-automata are constructed will be given in section **Feil! Fant ikke referansekilden.**, in parallel with the definition of an adjusted construction.

2.4 Mathematical concepts for counting

This section includes mathematical definitions needed for counting the number of states in automata output from the algorithms mentioned above in sections 2.2 and 2.3 above.2.3

2.4.1 Upper incomplete gamma function

The *upper incomplete gamma function* Γ is defined by:

$$\Gamma(n+1,x) \coloneqq \int_x^\infty t^n e^{-t}\,dt$$

A recursive definition can be derived by performing partial integration:

$$\int_x^\infty t^n e^{-t}\,dt$$

$$= [-t^n e^{-t}]_x^\infty - \int_x^\infty -nt^{n-1}e^{-t}\,dt$$

$$= \left[-\frac{t^n}{e^t}\right]_x^\infty + n\int_x^\infty t^{n-1}e^{-t}\,dt$$

Because $\lim_{t\to\infty} -\frac{t^n}{e^t} = 0$, we have:

$$\Gamma(n+1,x) = 0 - \left(-\frac{x^n}{e^x}\right) + n\int_x^\infty t^{n-1}e^{-t}\,dt = \frac{x^n}{e^x} + n\Gamma(n,x)$$

Recursive definition:

$$\boldsymbol{\Gamma(n+1,x) = \frac{x^n}{e^x} + n * \Gamma(n,x)}$$

The recursion naturally stops at $n = 0$, which gives the base case:

$$\Gamma(1,x) = \frac{x^0}{e^x} + 0 * \Gamma(0,x) = e^{-x}$$

From this recursion, we can derive the series representation of Γ:

$$\Gamma(n+1,x) = \frac{x^n}{e^x} + n\frac{x^{n-1}}{e^x} + n(n-1)\frac{x^{n-2}}{e^x} + \cdots + n!\,e^{-x} = \sum_{m=0}^{n}\frac{n!}{m!}\frac{x^m}{e^x} = \frac{n!}{e^x}\sum_{m=0}^{n}\frac{x^m}{m!}$$

Series definition:

$$\boldsymbol{\Gamma(n+1,x) = \frac{n!}{e^x}\sum_{m=0}^{n}\frac{x^m}{m!}}$$

2.4.2 Stirling numbers of the second kind

The Stirling numbers of the second kind $\left\{ {n \atop k} \right\}$ denotes the number of partitions of n elements into k non-empty subsets for all $n, k \in \mathbb{N}$. The Stirling numbers of the second kind can be defined recursively by ([4]):

$$\left\{ {n+1 \atop k} \right\} = \left\{ {n \atop k-1} \right\} + k \left\{ {n \atop k} \right\}$$

The intuition behind this recursive definition is that the $(n+1)$th element can either be put alone in its own subset or added into any of the subsets in a partition given by $\left\{ {n \atop k} \right\}$. If the $(n+1)$th element is put in its own subset, there are $\left\{ {n \atop k-1} \right\}$ ways to partition the rest. If not, the $(n+1)$th element can be put into any of the k subsets in every partition given by $\left\{ {n \atop k} \right\}$.

Given the recursive relation above, the following conditions are sufficient to derive all other Stirling numbers of the second kind:

$$\left\{ {n \atop n} \right\} = 1, \qquad n \geq 0$$

$$\left\{ {n \atop 0} \right\} = \left\{ {0 \atop n} \right\} = 0, \qquad n > 0$$

To illustrate a simple example, consider the number $\left\{ {3 \atop 2} \right\}$. Let $\{x, y, z\}$ be the set of $n = 3$ elements. Then, all possible partitions into $k = 2$ non-empty subsets are given by:

$$\{\{x\}, \{y, z\}\}, \qquad \{\{y\}, \{x, z\}\}, \qquad \{\{z\}, \{x, y\}\}$$

Since there are three possible partitions, $\left\{ {3 \atop 2} \right\} = 3$. We can also derive $\left\{ {3 \atop 2} \right\}$ from the recursive relation:

$$\left\{ {3 \atop 2} \right\} = \left\{ {2 \atop 1} \right\} + 2 \left\{ {2 \atop 2} \right\}$$

$$= \left(\left\{ {1 \atop 0} \right\} + \left\{ {1 \atop 1} \right\} \right) + 2 \left(\left\{ {1 \atop 1} \right\} + 2 \left\{ {1 \atop 2} \right\} \right)$$

$$= (0+1) + 2 \left(1 + 2 \left(\left\{ {0 \atop 1} \right\} + 2 \left\{ {0 \atop 2} \right\} \right) \right)$$

$$= 1 + 2 \left(1 + 2 \left(0 + 2(0) \right) \right) = 3$$

2.4.3 Ordered Bell numbers

The ordered Bell numbers $\alpha(n)$ represents the number of ordered partitions – also known as *weak orderings* – on a set of n elements into non-empty subsets. They are defined by the following formula ([5]):

$$\alpha(n) := \sum_{k=0}^{n} k! * \left\{ {n \atop k} \right\}$$

where $\left\{ {n \atop k} \right\}$ are the Stirling-numbers of the second kind. The ordered Bell numbers grow incredibly fast. Starting from $n = 0$, the first 10 ordered Bell-numbers are:

1, 1, 3, 13, 75, 541, 4 683, 47 293, 545 835, 7 087 261

3 Complementation via non-determinism reduction

3.1 Property π

Let R be an automaton constructed as shown in [2]. Then, R will have the property that all non-determined transitions will lead to exactly one non-accepting state and exactly one accepting state. We call this property π, and define it for any Büchi automaton $A = (Q, \Sigma, \delta, q_{in}, F)$:

$$\pi(A) \Leftrightarrow \forall q \in Q, a \in \Sigma : \ (|\delta_n(q,a)| \leq 1) \wedge (|\delta_a(q,a)| \leq 1),$$

where δ_n and δ_a denotes all transitions to non-accepting or accepting states, respectively:

$$\delta_n(q,a) = \delta(q,a) \cap \{Q \backslash F\},$$

$$\delta_a(q,a) = \delta(q,a) \cap F.$$

In other words, an automaton A has property π if and only if for all states $q \in Q$, reading a symbol $a \in \Sigma$ gives at most one accepting successor state and at most one non-accepting successor state. It follows from the definition of π that A must also have a non-determinism degree of at most 2:

$$\pi(A) \Rightarrow nd(A) \leq 2$$

As mentioned above, any reduced automaton R constructed by the non-determinism reduction algorithm will satisfy $\pi(R)$ by definition.

3.2 Complementation assuming $\pi(A)$

Let $A \coloneqq (Q, \Sigma, \delta, q_{in}, F)$ be the automaton that we want to complement. If we assume that A satisfies $\pi(A)$, we can simplify the original complementation algorithm described in [1]. The simplified complementation can then be applied directly to the reduced automaton R, since it satisfies $\pi(R)$. This section will expand on such a simplified algorithm, and how it differs from the original algorithm.

For the remainder of this section, OCA will be used when referring to the original complementation algorithm as described in [1]. Meanwhile, SCA will refer to the simplified version that assumes property π.

3.2.1 Upper (Non-accepting) Part

In OCA, the upper automaton $\hat{A} \coloneqq \left(\hat{Q}, \Sigma, \hat{\delta}, (\{q_{in}\}), \emptyset\right)$ is built with the following state base:

$$\hat{Q} \coloneqq \bigcup_{m=1}^{|Q|} \{(S_1, \ldots, S_m) \in \ (2^Q \backslash \emptyset)^m \mid \left(\forall j \neq k : S_j \cap S_k = \emptyset\right)\}$$

In the transition function, each set S_i will be considered individually. Starting from the rightmost component in the tuple, we calculate $\sigma(S_i, a)$:

$$\sigma\ (S_i, a) \coloneqq \ \delta\ (S_i, a) \setminus \bigcup_{j=i+1}^{m} \delta\ (S_j, a),$$

which is split into sets of non-accepting and accepting states, respectively:

$$\sigma_n\ (S_i, a) \coloneqq \ \sigma\ (S_i, a) \cap \ (Q \setminus F\ ),$$

$$\sigma_a\ (S_i, a) \coloneqq \ \sigma\ (S_i, a) \cap \ F\ .$$

Finally, if we let $\hat{p} \coloneqq (S_1, \dots, S_m)$ be a state in $\hat{Q}$, then $\hat{\delta}(\hat{p}, a) = \hat{q}$, where $\hat{q}$ is obtained from removing all empty sets in the following tuple:

$$(\sigma_n\,(S_1, a), \sigma_a\,(S_1, a), \dots, \sigma_n\,(S_m, a), \sigma_a\,(S_m, a))$$

Now, in SCA, we can simplify the state base from $(2^Q)^m$ to $(Q)^m$. This follows from the fact that if S_i is a singleton set representing exactly one A-state, and A satisfies $\pi(A)$, then $\sigma_n(S_i, a)$ and $\sigma_a(S_i, a)$ are also singletons (or empty sets). With the same initial state as in OCA, which contains only singletons, we know that all other states also will contain only singletons. Since all sets S_i are singleton sets, we can simply disregard the notion of sets and call them $p_i \in Q$. Note that although this defines SCA independently from OCA, it could also be interpreted as a special case of the more general algorithm OCA, where all sets S_i contain exactly one element.

Let $\hat{A}_\pi \coloneqq \left(\hat{Q}_\pi, \Sigma, \hat{\delta}_\pi, (q_{in}), \emptyset\right)$ be the upper automaton in SCA. Then $\hat{A}_\pi$ has the simplified set of states:

$$\hat{Q}_\pi \coloneqq \bigcup_{m=1}^{|Q|} \{(p_1, \dots, p_m) \in (Q)^m \mid (\forall j \neq k\colon\ p_j \neq p_k)\}$$

To define the transition function of $\hat{A}_\pi$, we do the same steps as in OCA:

$$\sigma\,(p_i, a) = \delta\,(p_i, a) \setminus \bigcup_{j=i+1}^{m} \delta\,(p_j, a),$$

which is split into a non-accepting and an accepting state, respectively. To account for the cases where $\sigma(p_i, a) < 2$, we redefine σ_n and σ_a to always output a single state $p \in Q$ or the "empty state" $\perp$:

$$\sigma_n\,(p_i, a) \coloneqq \begin{cases} \sigma(p_i, a) \cap (Q \setminus F), & |\sigma(p_i, a) \cap (Q \setminus F)| = 1 \\ \perp, & |\sigma(p_i, a) \cap (Q \setminus F)| = 0 \end{cases}$$

$$\sigma_a\,(p_i, a) \coloneqq \begin{cases} \sigma(p_i, a) \cap F, & |\sigma(p_i, a) \cap F| = 1 \\ \perp, & |\sigma(p_i, a) \cap F| = 0 \end{cases}$$

Finally, if we let $\hat{p}_\pi \coloneqq (p_1, \dots, p_m)$ be a state in $\hat{Q}_\pi$, then $\hat{\delta}_\pi(\hat{p}_\pi, a) = \hat{q}_\pi$, where $\hat{q}_\pi = (q_1, \dots, q_{m'})$ is obtained from removing all empty states $\perp$ in the following tuple:

$$(\sigma_n\,(p_1, a), \sigma_a\,(p_1, a), \dots, \sigma_n\,(p_m, a), \sigma_a\,(p_m, a))$$

3.2.2 Lower (Accepting) Part

Let $\check{A} \coloneqq (\check{Q}, \Sigma, \check{\delta}, \check{q}_{in}, \check{F})$ be the lower part of $\bar{A}$ in OCA, and $\check{A}_\pi \coloneqq (\check{Q}_\pi, \Sigma, \check{\delta}_\pi, \check{q}_{in}, \check{F}_\pi)$ the corresponding lower part in SCA. Since runs of $\bar{A}$ enter $\check{A}$ by "jumping" to any state $\check{q} \in \check{Q}$ from $\hat{A}$, the initial state of the lower part is irrelevant. Hence, there is no need to define a different initial state in SCA.

In the lower part, a coloring of state components is introduced to define the appropriate acceptance condition for the complemented automaton. Let μ be a run of $\bar{A}$, and let j be the point where μ makes the jump to the lower part, i.e. $i \geq j \Leftrightarrow \mu_i \in \check{A}$. Then, the three colors 0, 1 and 2 are used to keep track of what happens in the greedy runs of A after j steps. For all $i \geq j$, we have:

<u>color $c = 0$</u>: If a tuple component $(p_k, 0)$ is 0-colored in μ_i, then each greedy run r of A such that $r_i = p_k$ has not yet visited an accepting A-state "since" j.

<u>color $c = 2$</u>: If a tuple component $(p_k, 2)$ is 2-colored in μ_i, then the greedy run r of A such that $r_i = p_k$ has visited an accepting A-state since j.

color $c = 1$: If a tuple component $(p_k, 1)$ is 1-colored in μ_i, then the greedy run r of A such that $r_i = p_k$ has visited an accepting A-state since j, but there also exist 2-colored components that have not yet disappeared. We say that 1-colored components are on hold, meaning that for the moment, we wait until the 2-colored components disappear.

This formulation differs a bit from OCA, where the tuple components $(S_k, 0)$ holds sets of states $S_k \in 2^Q$ instead of single states $p_k \in Q$. Nonetheless, the principle is the same.

In OCA, the state base of $\check{A}$ is defined as:

$$\check{Q} \coloneqq \bigcup_{m=1}^{|Q|} \{((S_1, c_1), \dots, (S_m, c_m)) \in (2^Q \setminus \{\emptyset\} \times [0,2])^m \mid \forall j \neq k \colon S_j \cap S_k = \emptyset\}.$$

Similarly to what we did in the upper part, this is simplified in SCA to:

$$\check{Q}_\pi \coloneqq \bigcup_{m=1}^{|Q|} \{\big((p_1, c_1), \dots, (p_m, c_m)\big) \in (Q \times [0,2])^m \mid \forall j \neq k \colon p_j \neq p_k\}$$

Now, to define the transition function $\check{\delta}_\pi \colon \check{Q}_\pi \times \Sigma \mapsto \check{Q}_\pi$, we expand the definition of $\hat{\delta}_\pi \colon \hat{Q}_\pi \times \Sigma \mapsto \hat{Q}_\pi$ to include the colorings c_i. Let $\check{p}_\pi \coloneqq \big((p_1, c_1), \dots (p_m, c_m)\big) \in \check{Q}_\pi$ and let $d \colon \check{Q}_\pi \mapsto \hat{Q}_\pi$ be a function that "de-colors" all components in a state's tuple:

$$d(\check{p}_\pi) = (p_1, \dots, p_m)$$

Then, for $a \in \Sigma$, we have $\check{\delta}_\pi(\check{p}_\pi, a) = \check{q}_\pi$ if and only if $\hat{\delta}_\pi(d(\check{p}_\pi), a) = d(\check{q}_\pi)$ and all the color transitions $c \to c'$ from components (p, c) in $\check{p}_\pi$ to (q, c') in $\check{q}_\pi$, where $q \in \sigma(p, a)$ follow the following rules:

- $0 \to 1$: if $q \in F$ and $\check{p}_\pi$ is not a breakpoint,
- $0 \to 2$: if $q \in F$ and $\check{p}_\pi$ is a breakpoint,
- $1 \to 2$: if $\check{p}_\pi$ is a breakpoint,
- $c \to c$: otherwise,

where a "breakpoint" is a state of $\check{Q}_\pi$ that does not contain any 2-colored components. Finally, like in OCA, we define the set of accepting states $\check{F}_\pi$ to be all the breakpoint states in $\check{Q}_\pi$, i.e. states without any 2-colored components:

$$\check{F}_\pi \coloneqq \bigcup \{\big((p_1, c_1), \dots, (p_m, c_m)\big) \in \check{Q}_\pi \mid \forall 1 \leq i \leq m \colon\ c_i \neq 2\}.$$

3.2.3 Merging the two sub-automata

The full complement automaton $\bar{A}_\pi \coloneqq \big(\bar{Q}_\pi, \Sigma, \bar{\delta}_\pi, (q_{in}), \bar{F}_\pi\big)$ is built by joining $\hat{A}_\pi$ and $\check{A}_\pi$. This is done exactly like in OCA:

- $\bar{Q}_\pi \coloneqq \hat{Q}_\pi \cup \check{Q}_\pi$
- $\bar{q}_\pi \coloneqq (q_{in})$
- $\bar{F}_\pi \coloneqq \check{F}_\pi$
- The transitions $\bar{\delta}_\pi \colon \bar{Q}_\pi \times \Sigma \to 2^{\bar{Q}_\pi}$ permit all transitions in $\hat{\delta}_\pi$ and $\check{\delta}_\pi$, plus the "jump transitions" from $\hat{A}$ to $\check{A}$ for all $\hat{p}_\pi \coloneqq (p_1, \dots, p_m) \in \hat{Q}_\pi$ and $a \in \Sigma$:

$$\check{\delta}\Big(\big((p_1, 0), \dots, (p_m, 0)\big), a\Big) \in \bar{\delta}_\pi(\hat{p}_\pi, a)$$

This yields an automaton that accepts the complement of the ω-language accepted by A, assuming that $\pi(A)$ is satisfied:

$$\pi(A) \Rightarrow L_\omega(\bar{A}_\pi) = \overline{L_\omega(A)}$$

3.3 Complexity

This section will mainly discuss the complexity of SCA. A small comparison to OCA will follow in section 3.3.3. For an in depth report on the complexity of OCA, please refer to sections 6.1-6.3 in [6].

Let $A \coloneqq (Q, \Sigma, \delta, q_{in}, F)$ be the original input automaton and let $\gamma(n, \lambda)$ be the maximum number of states in $\bar{Q}_\pi$, given $n \coloneqq |Q|$ and the number of colors λ used when coloring tuple components in the lower part. Then, $\gamma(n, \lambda)$ is simply calculated by summing the maximum number of states in the two sub-automata:

$$\gamma(n, \lambda) \coloneqq u(n) + l(n, \lambda),$$

where $u(n)$ and $l(n, \lambda)$ are the maximum number of states in the upper part and lower part, respectively, i.e. the smallest possible numbers such that $|\hat{Q}_\pi| \leq u(n)$ and $|\check{Q}_\pi| \leq l(n, \lambda)$.

3.3.1 Upper part $\hat{A}_\pi$

To count the number of states in $\hat{Q}_\pi$, we need to calculate how many tuples can possibly be made from states in Q. Let m be the number of states included in a tuple. Then, there are $m!$ ways to order such a tuple. Furthermore, there are $\binom{n}{m}$ ways to choose m states from Q. So, the number of possible tuples of size m is:

$$\binom{n}{m} m!$$

Now, since m can be any number between 1 and n, we take the sum over all possible tuple sizes. The result gives us the maximum number of states in the upper part:

$$\begin{aligned} u(n) &\coloneqq \sum_{m=1}^{n} \binom{n}{m} m! \\ &= \sum_{m=1}^{n} \frac{n!}{(n-m)!} \\ &= \sum_{m=0}^{n} \frac{n!}{(n-m)!} - 1 \\ &= n! \sum_{m=0}^{n} \frac{1}{(n-m)!} - 1 \\ &= \boldsymbol{n!} \sum_{\boldsymbol{m=0}}^{\boldsymbol{n}} \frac{\mathbf{1}}{\boldsymbol{m!}} - \mathbf{1} \end{aligned}$$

Now, recall the series representation of the *upper incomplete gamma function* presented in section 2.4.1:

$$\Gamma(n+1, x) = \frac{n!}{e^x} \sum_{m=0}^{n} \frac{x^m}{m!}$$

This yields an expression for $u(n)$ in terms of Γ:

$$u(n) = n! \sum_{m=0}^{n} \frac{1}{m!} - 1$$

$$= e \frac{n!}{e} \sum_{m=0}^{n} \frac{1}{m!} - 1$$

$$= \boldsymbol{e * \Gamma(n+1, 1) - 1}$$

3.3.2 Lower part $\check{A}_\pi$

The coloring of tuple components in the lower part adds an extra layer of complexity, compared to that of the upper part. Let λ be the number of colors used in our algorithm (in SCA, $\lambda = 3$). Then, every tuple can be colored in at most λ^m ways. So, the total number of colored ordered tuples built with states from Q is:

$$l(n, \lambda) \coloneqq \sum_{m=1}^{n} \frac{n!\, \lambda^m}{(n-m)!}$$

$$= \sum_{m=0}^{n} \frac{n!\, \lambda^m}{(n-m)!} - 1$$

$$= n! \sum_{m=0}^{n} \frac{\lambda^m}{(n-m)!} - 1$$

$$= n! \sum_{m=0}^{n} \frac{\lambda^{n-m}}{m!} - 1$$

$$= \boldsymbol{n!\, \lambda^n \sum_{m=0}^{n} \frac{1}{m!\, \lambda^m} - 1}$$

This can also be expressed in terms of the upper incomplete gamma function:

$$l(n, \lambda) = n!\, \lambda^n \sum_{m=0}^{n} \frac{1}{m!\, \lambda^m} - 1$$

$$= \lambda^n\, n! \sum_{m=0}^{n} \frac{\left(\frac{1}{\lambda}\right)^m}{m!} - 1$$

$$= e^{\left(\frac{1}{\lambda}\right)} \lambda^n \frac{n!}{e^{\left(\frac{1}{\lambda}\right)}} \sum_{m=0}^{n} \frac{\left(\frac{1}{\lambda}\right)^m}{m!} - 1$$

$$= \boldsymbol{\sqrt[\lambda]{e}\, \lambda^n\, \Gamma\left(n+1, \frac{1}{\lambda}\right) - 1}$$

3.3.3 Comparison with OCA

As states in OCA contain components that are sets of A-states $S_i \in 2^Q$ instead of singleton A-states $p_i \in Q$, the number of possible tuples is a lot higher than in SCA. Let $P \subseteq Q$ be the set of A-states included

in an OCA-tuple, and let $t \coloneqq |P|$. Then, like before, there are $\binom{n}{t}$ ways to pick t states from Q. However, instead of simply counting the number of orderings over t elements, we now count the number of *ordered partitions* over t elements.

Therefore, we use the *ordered Bell numbers* to express how many ordered partitions there exists for a given subset $P \subseteq Q$. With t elements, there are $\alpha(t)$ possible partitions. However, since P can be *any* subset of Q, we should sum for all possible subsets. The number of possible subsets of Q is:

$$|2^Q \backslash \emptyset| = \sum_{t=1}^{n} \binom{n}{t}$$

For each of these subsets, there are $\alpha(t)$ possible partitions, which gives us the total number of possible states in $\hat{Q}$:

$$u_{OCA}(n) \coloneqq \sum_{t=1}^{n} \binom{n}{t} \alpha(t)$$

As the ordered Bell numbers are larger than the factorials by an exponential factor ([5]), we have that $u_{OCA}(n) \gg u(n)$, which underlines that SCA is indeed a "simpler" algorithm than OCA.

3.3.4 Possibility for improvement

Even though the calculations above correctly identify an upper bound on the state growth of SCA, there are improvements that could decrease it further.

For example, as explained in section 3.7 in [1], if A is complete, all tuples with the color 2 in the rightmost component can be ignored, as they can never be succeeded by accepting states in $\bar{A}$. In this case, instead of λ^m possible colorings of a tuple, we get $\lambda^{m-1}(\lambda - 1)$ possible colorings, since the last component can only have $(\lambda - 1)$ different colors. This gives us an updated upper bound $l'(n, \lambda)$ for $|\check{Q}_\pi|$:

$$l'(n,\lambda) \coloneqq \sum_{m=1}^{n} \binom{n}{m} m! * \lambda^{m-1}(\lambda - 1)$$

$$= (\lambda - 1) \sum_{m=1}^{n} \frac{n!\,\lambda^{m-1}}{(n-m)!}$$

$$= \frac{(\lambda - 1)}{\lambda} \sum_{m=1}^{n} \frac{n!\,\lambda^{m}}{(n-m)!}$$

$$= \frac{(\lambda - 1)}{\lambda} * l(n,\lambda)$$

For $\lambda = 3$, this reduces the upper bound to $\frac{2}{3}$ of its original size.

3.4 Hypothesis $U(A) \cong U(R)$

Let U be a function that maps an automaton A to the upper part of its complement $\bar{A}$, as constructed in OCA. Next, let $A \coloneqq (Q, \Sigma, \delta, q_{in}, F)$ be a Büchi automaton and let $R \coloneqq (Q_R, \Sigma, \delta_R, q_R, F_R)$ be the corresponding automaton that is reduced to non-determinism degree 2. Now, a hypothesis that arose from performing both complementation (OCA) and non-determinism reduction by hand on several

example automata, was the possible isomorphism $\boldsymbol{U(A) \cong U(R)}$. It seemed feasible, as both algorithms include grouping of non-accepting and accepting states.

If this conjecture were to be proven, it would mean that the complementation algorithm would give the exact same output for A and R respectively, since the upper part implies the lower part in the construction of the complement. In other words, complementation via non-determinism reduction would lead to the same state growth as direct complementation.

3.4.1 Motivation

To illustrate the motivation behind the hypothesis, we will look at an example where the identity holds true. Let $E \coloneqq (Q, \Sigma, \delta, q_{in}, F)$ be the example Büchi automaton where $Q = \{0,1,2,3,4,5,6\}$, $\Sigma = \{a, b\}$, $q_{in} = 0$, $F = \{2\}$ and the transition function δ holds all transitions shown in Figure 3.

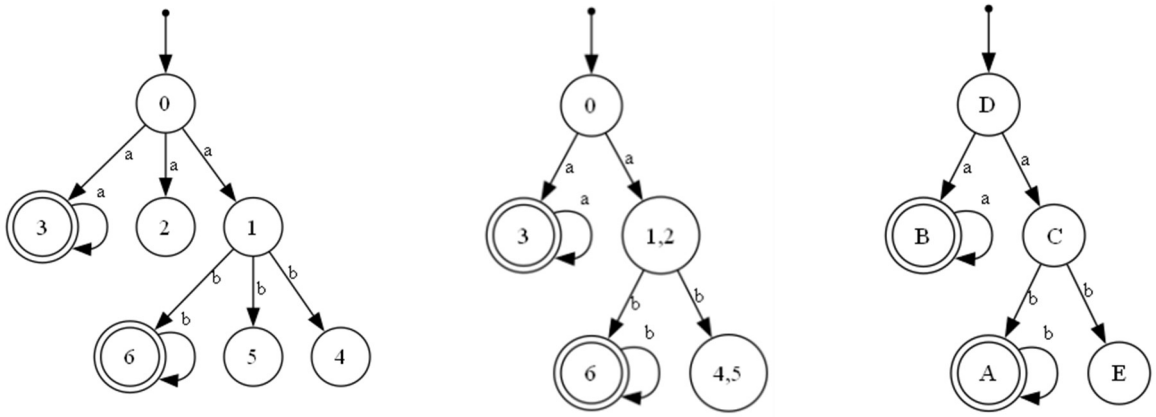

Figure 3 - From left to right: E, RE and RE with renamed states.

Next, let RE be the result of reducing E to non-determinism degree 2, also shown in Figure 3. To run the Python script developed for the upper part construction, we rename the states of RE such that each state is represented by a single symbol. Now, performing OCA on both E and RE (with the renamed states) yields the two automata illustrated in Figure 4. One can clearly see that the result is two isomorphic automata. If we substitute back the old names of the RE-states, the two automata are even completely identical.

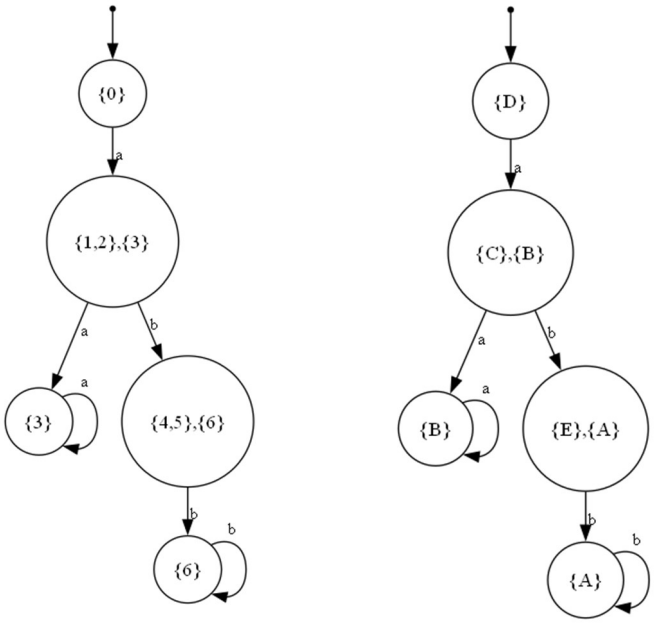

Figure 4 - The upper part construction from E (left) and RE (right).

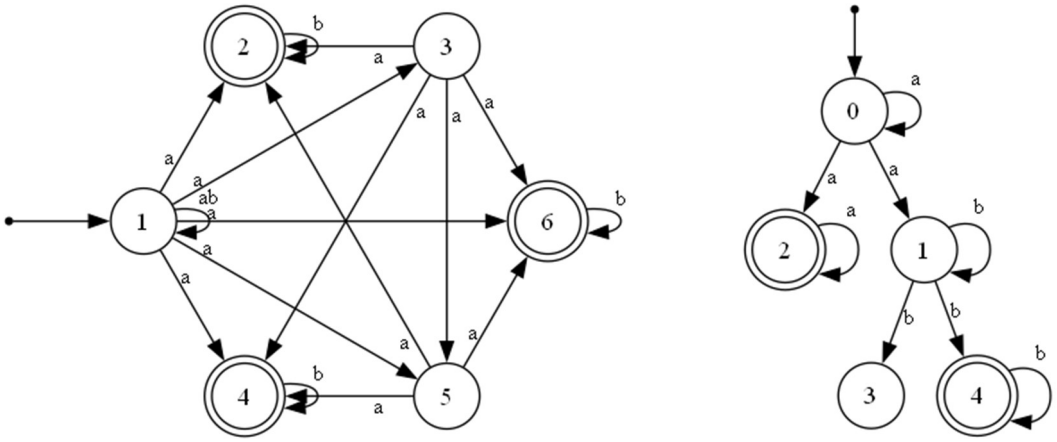

Figure 5 - Examples of automata, for which the identity $U(A) \cong U(R)$ holds.

The same result can be shown with many other automata. Figure 5 illustrates two more examples, for which the identity $U(A) \cong U(R)$ holds.

3.4.2 Refutation

After several efforts to prove the identity, it turned out that it cannot be done. On the contrary, the identity was proven to be invalid. The key difference between $U(A)$ and $U(R)$ lies in how the transition function of the upper part is defined. As explained in section 3.2.1, target states in $U(A)$ are built from components $\sigma_n(S_i, a)$ and $\sigma_a(S_i, a)$ which again are subsets of:

$$\sigma\,(S_i, a) \coloneqq\ \delta\,(S_i, a) \setminus \bigcup_{j=i+1}^{m} \delta\,(S_j, a).$$

Let $S_{U(A)} \in 2^{Q \setminus F} \cup 2^F$ denote such components in $U(A)$-states and let $S_{U(R)} \in 2^{Q_R \setminus F_R} \cup 2^{F_R}$ represent components in $U(R)$-states. Then, since R satisfy property π, each component $S_{U(R)}$ in the states of $U(R)$ will be singleton sets, where each singleton set corresponds to a single R-state (or a set of A-states). This is because for every non-determined transition in δ_R, the two possible target states will be one non-accepting and one accepting R-state. When constructing the upper part, these two will be split into two different sets, both necessarily being singleton sets. Therefore, the state tuples in $U(R)$ have components $S_{U(R)} \in Q_R = 2^{Q \setminus F} \cup 2^F$; similar to the components $S_{U(A)} \in 2^{Q \setminus F} \cup 2^F$.

However, while σ guarantees that any component in $U(A)$-states can include an A-state $q \in Q$ only once, the corresponding $\sigma_{U(R)}$ does not promise the same. We define $\sigma_{U(R)}$ formally as:

$$\sigma_{U(R)}\,(S_i, a) \coloneqq\ \delta_R\,(S_i, a) \setminus \bigcup_{j=i+1}^{m} \delta_R\,(S_j, a)$$

Now, because the transitions δ_R maps to $2^{Q_R} \subset 2^{2^Q}$ (δ only maps to 2^Q), the exclusion in the definition above only makes sure that the sets $\sigma_{U(R)}\,(S_i, a)$ are pairwise disjoint over Q_R, not over Q. That is, components $S_{U(R)} \in Q_R$ cannot be equal, but they *can* overlap, i.e. contain the same A-state.

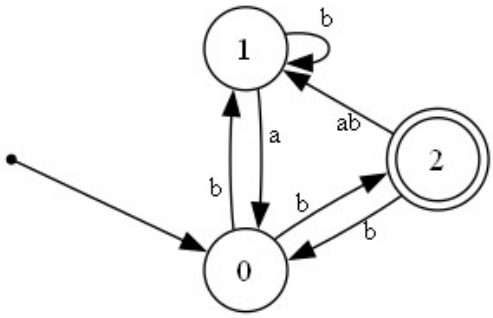

Figure 7 - Counter example (CE) proving that U(R) *is not equal to* U(A).

To make this clear, we will look at an example where $U(A) \neq U(R)$. Let $CE \coloneqq (Q, \Sigma, \delta, q_{in}, F)$ be the counter example automaton such that $Q = \{0,1,2\}$, $\Sigma = \{a, b\}$, $q_{in} = 0$, $F = \{2\}$ and the transition function δ holds all transitions shown in Figure 7.

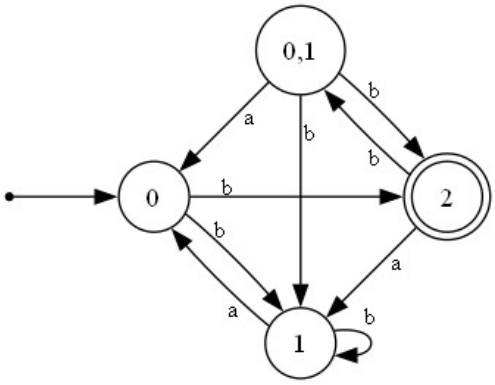

Figure 6 - The reduced version of CE.

Then, the reduction of CE leads to RCE as shown in Figure 6. Now, performing OCA on both CE and RCE yields the two automata illustrated in Figure 8. As $U(A)$ and $U(R)$ do not even have the same number of states, it is clear that they cannot be isomorphic.

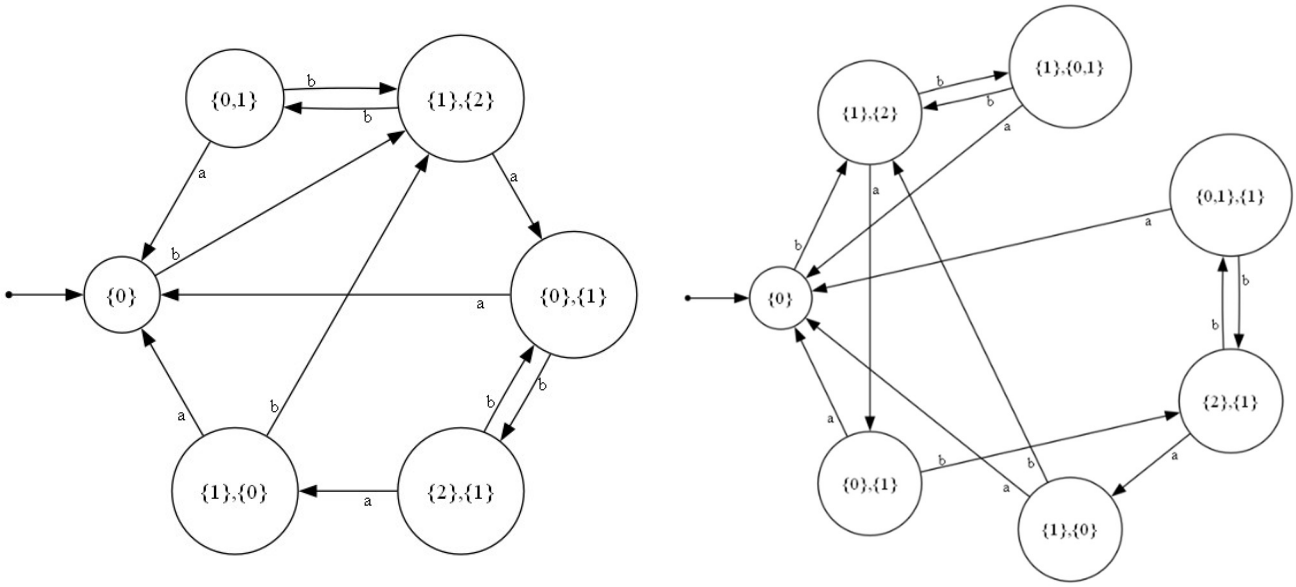

Figure 8 - The upper part constructions: $U(A)$ to the left and $U(R)$ to the right

With a closer look, one can observe that the transition from $(\{1\}, \{2\})$ with the symbol b leads to different target states in the two automata. This is a perfect example of how the two constructions behave differently. To understand what is happening, keep in mind that the components in state tuples in $U(R)$

technically are singleton sets $S_{U(R)} \in 2^{2^Q}$. In other words, the state $(\{1\},\{2\})$ really looks like $(\{\{1\}\},\{\{2\}\})$ in $U(R)$.

In the construction of $U(A)$, we have (starting from the rightmost component $\{2\}$):

$$\sigma\ (\{2\},b) = \ \delta\ (2,b) = \{0,1\}$$

$$\sigma\ (\{1\},b) = \ \delta\ (1,b) \setminus \delta\ (2,b) = \{1\} \setminus \{0,1\} = \emptyset$$

Meanwhile, in the construction of $U(R)$, we have (starting from the rightmost component $\{\{2\}\}$):

$$\sigma_{U(R)}\ (\{\{2\}\},b) = \ \delta_R\ (\{2\},b) = \{\{0,1\}\}$$

$$\sigma_{U(R)}(\{\{1\}\},b) = \ \delta_R\ (\{1\},b) \setminus \delta_R\ (\{2\},b) = \{\{1\}\} \setminus \{\{0,1\}\} = \{\{1\}\}$$

As you can see, the target set $\{1\}$ is excluded from $\sigma(\{1\},b)$, but not from $\sigma_{U(R)}(\{\{1\}\},b)$, which ultimately leads to different target states. In general, every time $\sigma(S_i,a)$ excludes a state $q \in Q$ because it is contained in $\delta\ (S_j,a)$ for some $j > i$, the same exclusion does not happen in $\sigma_{U(R)}\ (S_i,a)$, except if $\delta_R\ (S_i,a) = \delta_R\ (S_j,a)$.

4 Python implementation of BA

As part of the research project, a Python implementation of Büchi Automata was made to support quicker evaluation of new hypotheses.

The code consists of 3 core scripts: `ba.py`, `ba_generator.py` and `ba_saver.py`. The former defines the class `BuchiAutomaton` which holds all the data of a BA, as well as class-specific methods. The BA-generator can generate random Büchi Automata, with parameters controlling their size and non-determinism degree. Finally, ba_saver.py provides a file management system, taking care of saving and loading BA-files.

The class `BuchiAutomaton` holds field variables for an automaton's states, alphabet, transition function, initial state, and the set of accepting states. Class-specific methods include simple operations, such as adding a transition, as well as more complex algorithms, like the reduction algorithm described in [2] and the upper part construction from the complementation algorithm in [1]. There is also a method to check for isomorphism with another automaton, using a `DiGraphMatcher` from the library NetworkX ([7]). The `visualize()`-method imports functionality from another library, Graphviz ([8]), and is used to render all figures of Büchi automata in this thesis.

To test the hypothesis posed in section 3.4, a script called `equality_test.py` was developed to visualize the results of both paths to the upper part construction, given a Büchi automata A; the direct path without reduction and the indirect path via non-determinism reduction. To identify counter examples, the script was built to generate random automata and compare the output automata $U(A)$ and $U(R)$.

A full documentation of the Python implementation can be found in [9].

5 Conclusion

The problem of complementing Büchi automata continues to be a complex one, regardless of the method one tries to solve it with. This thesis presents an alternative route to complementation, starting with a reduction to non-determinism degree 2, which facilitates a simpler construction of the complement.

From the rejection of the conjecture $U(A) \cong U(R)$, we can conclude that complementation of Büchi automata via reduction of non-determinism does not have the same state growth as the direct complementation. In section 3.3, we show that assuming property π does greatly reduce the complexity of the complementation construction. However, this simplification may not be enough to compensate for the extra complexity of performing the reduction first.

Although the path via non-determinism reduction may be a computationally more expensive one, one could argue that it constitutes a better algorithm, in the sense that it is easier to comprehend, and thereby also more straightforward to perform or implement. Removing the notion of sets S_i as tuple components arguably leads to a less intricate structure in the complementation construction. As the non-determinism reduction is not much harder to grasp than a basic power-set construction, it might be worth the detour to gain a simpler construction in the next step.

6 Acknowledgements

I would like to thank Prof. Dr. Ulrich Ultes-Nitsche for his incredible support and patience, always willing to rephrase himself and break things down until I understood his input. I am especially thankful for Ultes-Nitsche's ability to communicate the subject of automata theory in an engaging and interactive manner, which sparked my interest for the topic when I took his class *Automata on Infinite Structures* back in 2023. Without it, I would probably not have started this master project in the first place.

I would also like to thank the ever-present Dorian Guyot for his guidance, always quick to reply to my emails full of silly questions. Dorian has been a great moral support, especially in times when I did not feel like I made enough progress. He has consistently reminded me that theoretical research is an art of patience and never giving up.

Finally, I want to thank my fiancé Kristin, without whom I would have struggled to maintain the sanity and discipline it takes to endure a full master's degree. She has been my rock in stressful moments, my motivator in lazy moments, and my best friend in happy moments.

7 Bibliography

[1] J. D. Allred and U. Ultes-Nitsche, “A Simple and Optimal Complementation Algorithm for Büchi Automata,” in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, Oxford United Kingdom: ACM, July 2018, pp. 46–55. doi: 10.1145/3209108.3209138.

[2] U. Ultes-Nitsche, “A power-set construction for reducing Büchi automata to non-determinism degree two,” *Inf. Process. Lett.*, vol. 101, no. 3, pp. 107–111, Feb. 2007, doi: 10.1016/j.ipl.2006.08.014.

[3] M. Karpiński, “Almost Deterministic ω-Automata with Existential Output Condition,” *Proc. Am. Math. Soc.*, vol. 53, no. 2, pp. 449–452, 1975, doi: 10.2307/2040034.

[4] “Stirling numbers of the second kind,” *Wikipedia*. Apr. 20, 2025. Accessed: Aug. 03, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stirling_numbers_of_the_second_kind&oldid=1286569093#Recurrence_relation

[5] “Ordered Bell number - Wikipedia.” Accessed: Aug. 03, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Ordered_Bell_number#Summation

[6] J. D. Allred, “A Direct Complementation Method for Büchi Automata,” PhD Thesis, University of Fribourg, 2017.

[7] “DiGraphMatcher.__init__ — NetworkX 3.5 documentation.” Accessed: Aug. 03, 2025. [Online]. Available: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.isomorphism.DiGraphMatcher.__init__.html

[8] “Graphviz — graphviz 0.21 documentation.” Accessed: Aug. 03, 2025. [Online]. Available: https://graphviz.readthedocs.io/en/stable/

[9] “Ma10as00/BuchiAutomata: Tool for conceptualizing and visualizing Büchi automata.” Accessed: Aug. 03, 2025. [Online]. Available: https://github.com/Ma10as00/BuchiAutomata