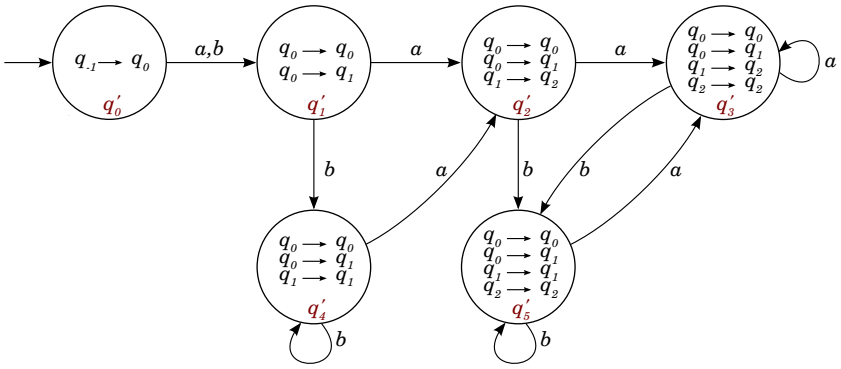
A Transition-sensitive Subset-construction for Büchi Automata

Joel Allred
University of Fribourg, Switzerland

September 3, 2010



Abstract

Is there a subset-construction-like complementation procedure for Büchi automata? Behind this interrogation hides the will to understand how the subset-construction operates on Büchi automata, and how to recover the information that is lost in the process. The main problem resides in the language gap that exists between non-deterministic and deterministic automata. Hence, when applied to a non-deterministic Büchi automaton, the subset-construction generally outputs an automaton accepting a superset of the language accepted by the original automaton. As a workaround, we created a new construction delivering a Muller automaton accepting the same language, and that works under some *partitioning condition*. The complementation of the automaton is then trivial. Whether the partitioning condition always holds or not, we do not know. But the construction is nevertheless interesting and triggers new possibilities for future research.

UNIVERSITAS FRIBURGENSIS

Contents

Nature uses only the longest threads to weave
her patterns, so each small piece of her fabric
reveals the organization of the entire tapestry.

Richard P. Feynman

1 Introduction

1.1 The Context

This paper is a Master's thesis written upon completion of the Master of Science in Computer Science at the University of Fribourg, Switzerland, in May 2010. It was supervised by Prof. Ulrich Ultes-Nitsche, which leads the *Foundations of Security and Dependability* research group, and is the head of the Department of Informatics.

1.2 The Subject

Automata Theory, which is now a branch of theoretical computer science, has been around way before there even were computers. It started in the 1930's with the well-known *Turing machines* and then went on through the 40's with the appearance of *finite automata* and the concept of grammar, introduced by Chomsky in the late 50's.

Being a center piece of how we understand and design more complex machines, such as today's computers, the subject received a good deal of attention for several decades. An automaton being such a simple concept, it is an everlasting marvel that the most advanced and powerful of today's computers can be reduced to simple machines made of states and transitions.

It is in the 60's that the concept of finite automata on infinite sequences was developed by Büchi, McNaughton and Rabin. This new construction proved to be a powerful tool for representing decision problems. It is now used in the specification and verification of concurrent programs

Considering the fascinating problems of decidability underlying the aspects of Büchi automata theory, it is surprising that research in this field seems to have been somewhat left aside in recent years judging by the age of the last important discoveries.

1.3 The Project

The project started with a simple question: *Is there a subset-construction-like complementation procedure for Büchi automata?* Behind this interrogation was the will to understand how the subset-construction operated on Büchi automata, and how to recover the information that was lost in the process.

The main problem resides in the language gap that exists between non-deterministic and deterministic automata. In fact, once the subset-construction has been applied to a non-deterministic Büchi automaton, there is no way of finding the true complement without considering the original automaton again.

The initial idea was to complete the determinised-and-complemented automaton with *something* coming from the original automaton, as shown below:

Let $\mathcal{A}$ be a non-deterministic Büchi automaton and $D(\mathcal{A})$ be the automaton obtained via the subset-construction:

$$\begin{aligned} & L_{\mathcal{A}} \subseteq L_{D(\mathcal{A})} \\ \Rightarrow & \overline{L_{\mathcal{A}}} \supseteq \overline{L_{D(\mathcal{A})}} \\ \Rightarrow & \overline{L_{\mathcal{A}}} = \overline{L_{D(\mathcal{A})}} \cup X \end{aligned}$$

The question was the following: *Is there a general method to find X?* No answer was found to this question, but future research will definitely be going in this direction.

Instead, we investigated several other complementation possibilities. Most notably, we created a new construction (in Section 5.3) which is a true complementation procedure that works under some *partitioning condition*. Whether this condition always holds or not, we do not know. But the construction is nevertheless interesting and triggers new possibilities for future research.

1.4 The Language

A great deal of attention was given throughout this paper to convey an acceptable English language. Some terms that have been used may however make any English-speaking reader squeal with horror. For example, the words *determinise*, or *determinisation* cannot be found in any dictionary. The terms that however *do* exist are *determine* and *determination*, but these are synonymous with *ascertain*, and what we really mean here is: *make deterministic*. These terms were nevertheless used in this paper to lighten the text and make it more comprehensible. Further mishandling include *complementable*, *recurringly*, *non-complete* and *loopable*.

2 Fundamentals

This section provides the necessary background theory to deal with complementation of automata on finite and infinite words. It starts by stating the definitions of deterministic and non-deterministic finite automata. The subset-construction, which is a central object in this thesis, is then defined as a link between the two types.

Two kinds of automata on ω-sequences (i.e. words of infinite length) are then defined. The most common is probably the Büchi automaton, which is very similar in construction to automata on finite words. The second kind is the Muller automaton, which has the remarkable property of having a deterministic structure, along with accepting the same class of language as the Büchi automaton, namely ω-regular languages.

From a notational point of view, for a set Σ of symbols, we generally consider Σ^* to be the set of all finite elements of Σ and Σ^ω the set of all sequences of Σ of infinite length, which we may also call *ω-sequences*.

2.1 Automata on Finite Words

In order to investigate the complex problem of complementing automata over infinite sequences (e.g. *Büchi automata*), we may first observe how complementation operates on automata over finite sequences. The latter problem is easily solved, and even trivial in the case of deterministic automata.

We shall therefore give the definitions of deterministic and non-deterministic automata over finite words, and then underline the important relation between the two, namely the *subset-construction*.

2.1.1 Deterministic Automata

The deterministic automaton is the most fundamental notion in automata theory. All other automata styles are very similar in structure. An automaton is to be seen as a machine that takes a string as input, and decides whether to accept it or not. An automaton can then be linked to a particular language (set of accepted strings).

A finite **deterministic automaton** $\mathcal{A}$ is represented by a 5-tuple

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F),$$

where Q is a set of states, Σ a finite set of symbols (or alphabet), δ the transition function, q_0 the initial state ($q_0 \in Q$) and F a set of final (or accepting) states ($F \subseteq Q$). Given an input letter $a \in \Sigma$, the transition function is of the form $\delta : Q \times \Sigma \to Q$.

To explain how an automaton decides whether to accept a sequence or not, we extend the given transition function to a more general function $\tilde{\delta} : Q \times \Sigma^* \to Q$, defined recursively by:

$$\begin{aligned} \tilde{\delta}(q, \epsilon) &:= q && \text{(where } \epsilon \text{ is the empty word)} \\ \tilde{\delta}(q, xa) &:= \delta(\tilde{\delta}(q, x), a) && (x \in \Sigma^*, a \in \Sigma) \end{aligned}$$

A **word** $x \in \Sigma^*$ is said to be **accepted** by a given automaton $\mathcal{A}$ if $\tilde{\delta}(q_0, x) \in F$. In other words, a sequence x is accepted by the automaton if, starting from the initial state, the run of states created by the reading of x ends on a final state. A **language** is said to be **accepted** by the automaton if all the words it contains are accepted by that automaton.

Automata can be efficiently represented graphically as shown below, where the non-final states are represented by circles, the final states by double circles, and the transitions by arrows. The initial state is pointed to by an arrow with no origin (Fig.1)

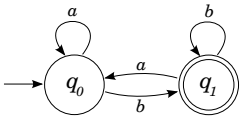


Figure 1: A deterministic automaton accepting all sequences of a's and b's ending with b

2.1.2 Non-deterministic Automata

A deterministic automaton produces a distinct sequence of states (or *run*) for each sequence. For example, in the above automaton, the string *baba* yields unambiguously the run $q_0q_1q_0q_1q_0$. Being straightforward and allowing all sorts of operations (as we will see later), this representation is however not very efficient (complexity-wise). In most cases, a language can be described far more efficiently through a *non-deterministic automaton*.

In a **non-deterministic automaton**, the transition function is redefined: $\delta : Q \times \Sigma \to 2^Q$. That means that from a given state, we can have one transition, several transitions or none. So a given word may yield several different runs of the automaton. The acceptance criterion is therefore updated: We say that a word $x \in \Sigma^*$ is accepted by $\mathcal{A}$ if, starting from the initial state, there exists at least one path triggered by the word x that leads to a final state. A graphical representation of a non-deterministic automaton is given on Fig.2.

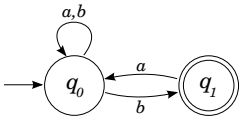


Figure 2: A non-deterministic automaton accepting all sequences of a's and b's ending with b

2.1.3 The Subset-Construction

The two automata depicted above incidentally represent the same language. Surprisingly enough, all non-deterministic automata can be reduced to deterministic automata, via the well-known *subset-construction*, which will be presented below. Given that deterministic automata are, quite clearly, a subset of non-deterministic automata, and that we yet have a method reducing the latter to the former, we can state an *equivalence* between deterministic and non-deterministic automata.

Let $\mathcal{N} = (Q_\mathcal{N}, \Sigma, \delta_\mathcal{N}, q_0, F_\mathcal{N})$ be a non-deterministic automaton. We construct a deterministic automaton $\mathcal{D} = (Q_\mathcal{D}, \Sigma, \delta_\mathcal{D}, q_0, F_\mathcal{D})$ in the following way:

$$Q_\mathcal{D} = 2^{Q_\mathcal{N}}$$
$$F_\mathcal{D} = \{S \in 2^{Q_\mathcal{N}} | S \cap F_\mathcal{N} \neq \emptyset\}$$
$$\forall S \subseteq Q_\mathcal{N}, \forall a \in \Sigma, \quad \delta_\mathcal{D}(S, a) = \bigcup_{p \in S} \delta_\mathcal{N}(p, a)$$

Practically speaking, we first create the initial state of the deterministic automaton. We then follow each transition. If several transitions with the same symbol reach different states, those states are put together into a new state. From a joint state, we keep following the transitions of all the states included in the joint state. Each state of the deterministic automaton that contains a final state of the original automaton is set final.

The proof of the equivalence of the languages of the non-deterministic and the *determinised* versions can be found in [1].

Let us consider the non-deterministic automaton of Fig.2 and apply the subset-construction. We get (Fig.3):

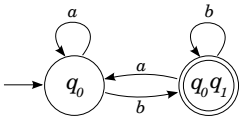


Figure 3: Deterministic version of the automaton of Fig.2

This deterministic automaton is equivalent to the one of Fig.1, which accepts the same language as the one of Fig.2. So we see that the subset-construction conserves the language.

2.2 Automata on Infinite Words

Having been investigated since the 1940's, most aspects of automata on finite words are fairly straightforward. Our interest will therefore move to automata over sequences of infinite length and, in particular, to *Büchi Automata*. Even if the definitions of these various kinds of automata look alike, there is a fundamental difference in the acceptance condition: clearly we cannot look at were an input sequence *stops* to determine if it is accepted. The acceptance condition will therefore contain the idea of *recurring visits* of a state or a set of states.

There several kinds of automata that accept words of infinite length, two of which we discuss here. *Büchi automata* accept words based on the recurring visit of independent states, whereas a *Muller automaton* checks if a certain subset of states is visited infinitely often, and rejects the word if the subset of *recurringly* visited states does not match its acceptance condition. The two types of automata are defined below.

2.2.1 Büchi Automata

Being the main subject of this paper, we start by defining Büchi automata. You will notice that the transition function δ that appears in the definitions above has now been replaced by a transition *relation* δ, being a better way of representing non-deterministic automata. This is a choice of the author of this definition (from [2]).

Definition 1. *A **Büchi automaton** over the alphabet Σ is of the form $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ with finite state set Q, initial state $q_0 \in Q$, transition relation $\delta \subseteq Q \times \Sigma \times Q$, and a set $F \subseteq Q$ of final states. A **run** of $\mathcal{A}$ on an ω-word $\alpha = \alpha(0)\alpha(1)\ldots$ from A^ω is a sequence $\sigma = \sigma(0)\sigma(1)\ldots$ such that $\sigma(0) = q_0$ and $(\sigma(i), \alpha(i), \sigma(i+1)) \in \delta$ for $i \geq 0$; the run is called **successful** if $\omega(\sigma)^1 \cap F \neq \emptyset$,*

[1] $\omega(\sigma) := \{s \in S | \exists^\omega n, \sigma(n) = s\}$

i.e. some state of F occurs infinitely often in it. $\mathcal{A}$ accepts α if there is a successful run of $\mathcal{A}$ on α. Let

$$L(\mathcal{A}) = \{\alpha \in \Sigma^\omega | \mathcal{A} \text{ accepts } \alpha\}$$

be the ω-language recognized by $\mathcal{A}$. If $L = L(\mathcal{A})$ for some Büchi automaton $\mathcal{A}$, L is said to be ***Büchi-recognizable****.*

When graphically represented, a Büchi automaton can not be distinguished from an automaton of the finite word kind. Here's an often cited deterministic Büchi automaton (Fig. 4):

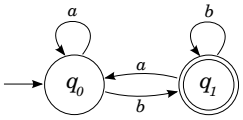


Figure 4: Deterministic Büchi automaton accepting $(a^* \cdot b)^\omega$.

The definition encompasses both deterministic and non-deterministic automata. Interestingly, these two types accept different classes of language. As an illustration, the following non-deterministic automaton (Fig.5), that accepts the language of all ω-sequences of a's and b's containing finitely many a's, cannot be *determinised*. There exists no deterministic Büchi automaton representing such a language.

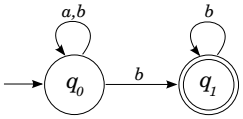


Figure 5: Non-deterministic Büchi automaton accepting $(a + b)^* \cdot b^\omega$.

2.2.2 Muller Automata

Similarly to a Büchi automaton, a Muller automaton accepts words of infinite length. Its acceptance condition is however different because, instead of loosely checking if a given state is visited an infinite number of times, we must check if the entire set of recurringly visited states is an element of the final set $\mathcal{F} \subseteq 2^Q$.

Definition 2. *A* ***Muller automaton*** *over the alphabet Σ is of the form $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ with finite state set Q, initial state $q_0 \in Q$, transition function $\delta : Q \times \Sigma \to Q$, and a collection $\mathcal{F} \subseteq 2^Q$ of final state sets. A run σ of $\mathcal{A}$ is* ***successful*** *if $\omega(\sigma) \in \mathcal{F}$, i.e. the states that $\mathcal{A}$ assumes infinitely often in σ form a set from $\mathcal{F}$. $\mathcal{A}$ accepts an ω-word α if the unique run σ of $\mathcal{A}$ on α is successful. An ω-language $L \subseteq \Sigma^\omega$ is called* ***Muller recognizable*** *if it consists of all ω-words accepted by some Muller automaton over Σ.*

Corollary 1. *As stated in [2], if $\mathcal{A}$ recognizes L, then $\mathcal{A}' := (Q, \Sigma, \delta, q_0, 2^Q \setminus \mathcal{F})$ recognizes $\Sigma^\omega \setminus L$.*

Because the acceptance condition involves sets of states instead of independent states, there is no immediate way to depict the acceptance conditions of a Muller automaton. What we will do, is draw the automaton, as usual, but with no final states, and specify the acceptance sets on a side note (Fig.6).

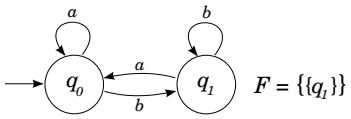


Figure 6: Muller automaton accepting $(a^* \cdot b)^\omega$.

Note that the definition implies that Muller automata are always deterministic. It is however possible to define a non-deterministic Muller automaton, but the relevance of such a form is limited in our case.

It is also important to mention that (non-deterministic) Büchi and (deterministic) Muller automata are equivalent in recognition power (cf. [4]. From a Muller automaton, we can always define a deterministic Muller automaton accepting the same language; the way to do that is indicated in [2]. The converse is also true.)

3 Complementation

The complementation of automata has many applications in the area of formal verification. Notably, having the complement of an automaton facilitates the verification of the containment property. For example, if we want to check that the language $L_{\mathcal{A}'}$ of an automaton $\mathcal{A}'$ is contained in the language $L_{\mathcal{A}}$ of an automaton $\mathcal{A}$, we can simply check that the intersection of $L_{\mathcal{A}'}$ and $L_{\mathcal{A}^c}$ (where A^c is an automaton complement to $\mathcal{A}$) is empty. However, the complementation of a language as it stands is not straightforward. A common practice is to translate the language into an automaton, complement it, and translate it back to a regular expression. But complementing automata can be tricky.

3.1 Complementing Automata on Finite Words

As we will see, the complementation of automata on finite sequences is straightforward. For the deterministic case, it is even trivial. Since the acceptance of a word relies on the last visited state, a simple switch of the states between *final* and *non-final* does the trick.

3.1.1 Complementing Deterministic Automata

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a deterministic automaton accepting language $L_{\mathcal{A}}$. Then, the automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q \setminus F)$ accepts the language complement to $L_{\mathcal{A}}$. Informally, this can be proved in the following way:

If a word x of Σ^* is accepted by $\mathcal{A}$, then the unique run of $\mathcal{A}$ on x ends on an accepting state, meaning it will end on a non-accepting state of $\mathcal{A}'$ and will therefore not be accepted by $\mathcal{A}'$. Conversely, if x is not accepted by $\mathcal{A}$, then its unique run of $\mathcal{A}$ ends on a non-final state, meaning that the unique run of $\mathcal{A}'$ ends on a final state, rendering that sequence accepting on $\mathcal{A}'$.

3.1.2 Complementing Non-Deterministic Automata

There is no easy method of directly complementing a non-deterministic automaton, because non-determinism implies some uncertainty about which state is being visited during the reading of a sequence.

To illustrate this, let's show an example of a non-deterministic automaton, where it is clear that switching the states from accepting to non-accepting does not have the desired effect on the language (in this case: no effect!):

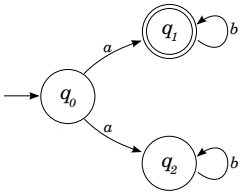


Figure 7: A non-deterministic automaton

To complement a non-deterministic automaton, the solution is to *determinise* it, and then complement it. As we have seen, for any non-deterministic automaton, there exists a deterministic automaton that can be built using the subset-construction.

3.2 Complementing Deterministic Büchi Automata

Complementing automata over infinite sequences appears to be more complex than for the finite case. Nevertheless, deterministic cases are easily solvable. In the case of deterministic Büchi automata, the following algorithm (slightly corrected from [5]) delivers a complement automaton:

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a deterministic Büchi automaton. We construct the complement automaton $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ such that, $\forall q \in Q, q' \notin Q, f \notin Q$:

$$
\begin{aligned}
Q' &= Q \cup \{q' | q \in Q \setminus F\} \cup \{f\} \\
\delta' &= \begin{cases}
\delta & \cup \\
\{(q, \epsilon, q') | q \in Q \setminus F\} & \cup \\
\{(q', a, p') | q, p \in Q \setminus F \wedge (q, a, p) \in \delta\} & \cup \\
\{(q, a, f) | q \in Q \wedge a \in \Sigma \wedge \nexists p \in Q : (q, a, p) \in \delta)\} & \cup \\
\{(q', a, f) | q \in Q \setminus F \wedge a \in \Sigma \wedge \nexists p \in Q : (q, a, p) \in \delta\} & \cup \\
\{(f, a, f) | a \in \Sigma\}
\end{cases} \\
F' &= \{q' | q \in Q \setminus F\} \cup \{f\}.
\end{aligned}
$$

In words, the obtained automaton $\mathcal{A}'$ contains accepting sets of states that are non-deterministically reachable from any non-accepting state of $\mathcal{A}$. All other states are non-accepting. So $\mathcal{A}'$ accepts exactly the complement language of $\mathcal{A}$.

3.3 Complementing (deterministic) Muller Automata

As seen in section 2.2.2, Muller automata are trivially *complementable*, thus yielding a seemingly trivial way of complementing Büchi automata. It suffices to translate our non-deterministic Büchi automaton into a deterministic Muller automaton (via *Safra's determinisation construction*), complement it, and translate it back into a Büchi automaton. However, these translations are far from being efficient, and are also difficult to apprehend, being somewhat unintuitive. For details, refer to [2].

3.4 The Relevance of Completeness

Completeness is a property that automata in general can have.

Definition 3. *An automaton (of any kind) $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is said to be* ***complete*** *if, for each state $q \in Q$, and for each symbol $a \in \Sigma$, the transition $\delta(q, a)$ leads to at least one state of Q.*

Note that deterministic automata (or deterministic Büchi automata), as they are defined in this thesis, are always complete. Non-deterministic automata (or Büchi) can be non-complete, but can always be completed. It suffices to create a new non-accepting state that is self-looping for all symbols (i.e. we create a *dead-end*) and for each missing transition, we insert a transition to that newly create state.

In the above algorithm, we see that the fourth and fifth lines of the transition relation are only applicable to a non-complete Büchi automaton. Hence, systematically using complete automata (which can always be obtained easily) simplifies the complementation process.

Generally speaking about complementation, it appears unsafe to consider non-complete automata, because some of the information needed to build the complement automaton may be hidden

in the *uncompleted* part of the automaton. The argument is the following: to build an automaton $\mathcal{A}'$ complement to $\mathcal{A}$, we may look for the words that do not achieve *infinite visits* of final states in $\mathcal{A}$, but also words that do not achieve *anything* in $\mathcal{A}$, being in the uncompleted part of $\mathcal{A}$. Failing to represent all possible runs in an automaton may cause a loss of the information needed to build the complement.

4 The Problem with Complementation

As we have seen, complementing a deterministic Büchi automaton is not a problem. The methods we can use are straightforward and easy from a computational complexity point of view. The underlying factor is the uniqueness of the run for any given ω-string, which allows to unambiguously define the currently visited state at any point of the string.

However, for non-deterministic cases, there exist methods (e.g. Safra's Determinisation Construction), but those are complicated and not very intuitive.

The problem with non-determinism is that, to construct a complement automaton, we need to check that each possible run of the original automaton is non-accepting, that means checking infinitely many different paths.

We know we cannot make a non-deterministic Büchi automaton deterministic. We will nevertheless observe what happens when applying the subset-construction (for automata on finite words) to a Büchi automaton:

Let $\mathcal{A}$ be a non-deterministic Büchi automaton as depicted in Fig.8:

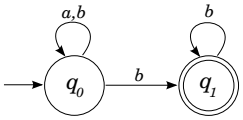


Figure 8: Non-deterministic Büchi automaton accepting $(a + b) \cdot b^{\omega}$.

Let $\mathcal{A}'$ (Fig.9) be the automaton obtained by applying the subset-construction to $\mathcal{A}$:

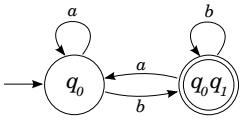


Figure 9: Deterministic version of Fig.9, accepting $(a^* \cdot b)^{\omega}$.

We see that accepting and non-accepting states may be put together, and be set as *accepting*, following the subset-construction algorithm. In the general case (as in this particular one), the method enlarges the originally accepted language. Avoiding the joining of accepting and non-accepting states renders a modified subset-construction (described in [3]) that reduces the non-determinism to two (instead of one) but preserves the language.

As suggested by some attempts described in section 6, non-determinism degree two does not seem to be enough to make complementation easy. What we would need is a true deterministic automaton, but we know that in general, non-deterministic Büchi automata cannot be translated into deterministic ones. Nevertheless, if a straightforward complementation procedure exists, it will arguably resemble the subset-construction determinisation process.

Originally, in this paper, we were seeking such a modified subset-construction that would lead us to the complement. Although we did not find this, what we have (described in the next section) is a new construction that delivers a Muller automaton under some conditions. So we definitely believe to be a step closer to such an ideal construction. This *half-way through* construction is entirely formalised and proved in the following section.

5 A Transition-based Complementation Procedure

5.1 Introduction

The method described below is the most promising attempt to deliver a complementation procedure that is easy to understand, and can be done by hand without too much trouble (except, perhaps, that of complexity). It introduces a new kind of subset-construction that does not base its labelling of states on visited states of the original automaton (as in the common subset-construction), but rather on activated transitions. Each state holds in its *memory* (i.e. label), the last transitions that were used. The great advantage of such a construction is that it holds as much information as the original automaton[2] (leaving out the final states) and is furthermore deterministic.

At this time, unfortunately, there is no generalised method to set the acceptance conditions. The construction below allows us to create a Muller automaton, but only under a (partitioning[3]) condition. It is nevertheless our belief that this condition is always verified and future research will be conducted in order to prove that.

The construction does not immediately deliver a complement automaton, but instead creates a Muller automaton accepting the same language as the original. This Muller automaton can then be complemented easily.

5.2 Preliminaries

The elaboration of the construction requires the preliminary definition of a few concepts, mainly because we extensively use pairs and sets and sequences of them to describe the transitions in the automata.

The first definition will be used to signify the set of states that occur in a given word.

Definition 4. *For a sequence* $\sigma = \sigma(0)\sigma(1)\ldots \in Q^\omega$ *of states,*
$\mathbf{States}(\sigma) := \{s \in Q | \exists i, \sigma(i) = s\}$.

As mentioned above, we use pairs to represent transitions. We also need a way of extracting the left and right terms of the pair, respectively the origin and the destination of a transition. The $Left$ and $Right$ operators also apply to a collection of pairs of states.

Definition 5. *For a pair of states* $t = (q_1, q_2)$, $\mathbf{Left(t)} := q_1$ *and* $\mathbf{Right(t)} := q_2$.

Definition 6. *For a collection* $q \in 2^{Q \times Q}$ *of pairs of states,* $\mathbf{Left(q)} := \{a | \exists b \in Q : (a, b) \in q\}$ *and* $\mathbf{Right(q)} := \{b | \exists a \in Q : (a, b) \in q\}$.

At some point in the construction, we will be working on the original automaton, as well as the constructed one in parallel. We will then need a notion of *combined state* of two automata, which is basically a pair containing a state of one automaton, and one of the other.

Definition 7. *A* ***combined state*** *of two automata* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *and* $\mathcal{A}' = (Q', \Sigma', \delta', q_0', F')$ *is a pair* (q, q') *s.t.* $q \in Q$ *and* $q' \in Q'$.

[2] This statement comes from a quick observation that the constructed automaton can always be translated back to the original automaton, because all the states and transitions are represented (multiple times in general) in the construction.

[3] c.f. Theorem 1 of Section 5.3

We introduce the concept of *continuity*, which is important for the construction below. It insures that what happens in the constructed automaton $\mathcal{A}'$ can also happen in the original automaton $\mathcal{A}$, and that we are not creating new paths in $\mathcal{A}'$ that do not correspond to existing paths in $\mathcal{A}$. It is a way of verifying that the language is *not* augmented which generally happens when applying the subset-construction. We also extend the definition of continuity to sequences and sets of states.

Definition 8. *A sequence $t = t(0)t(1)\ldots \in (Q \times Q)^\omega$ of pairs of states is said to be **continuous** if $\forall i \in \mathbb{N}$, $Right(t(i)) = Left(t(i+1))$.*

Definition 9. *For a set of states Q of an automaton $\mathcal{A}$ and a set of states $Q' \subseteq 2^{Q\times Q}$ of an automaton $\mathcal{A}'$, a sequence $\sigma = \sigma(0)\sigma(1)\ldots \in Q'^\omega$ of states of $\mathcal{A}'$ is said to be **continuous** if $\forall i \in \mathbb{N}$, $Right(\underbrace{\sigma(i)}_{\in 2^{Q\times Q}}) = Left(\underbrace{\sigma(i+1)}_{\in 2^{Q\times Q}})$.*

Definition 10. *For a set of states Q of an automaton $\mathcal{A}$, a set of states $Q' \subseteq 2^{Q\times Q}$ of a Muller automaton $\mathcal{A}'$, is said to be **continuous** if $\forall q' \in Q'$, $\forall a \in \Sigma$, $Right(q') = Left(\delta(q', a))$.*

5.3 The Construction

This section describes the main object of this paper. From a non-deterministic complete Büchi automaton, we construct a Muller automaton accepting the same language. The construction is not yet proved to be universal, hence it is theoretically possible that the construction gives no Muller automaton at all in certain (not yet encountered) cases, where the partitioning condition would not be verified.

The construction is fairly similar to the subset-construction, but compensates the loss of information by keeping the last visited transition in the state labels of the new automaton.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ be a non-deterministic complete Büchi automaton and let $q_{-1} \notin Q$ be a new (dummy) state. Let Q', q'_0, δ', A'_{acc} and A'_{nacc} be defined as follows:

$$\begin{aligned} q'_0 &:= (q_{-1}, q_0) \\ Q' &= 2^{Q\times Q} \cup q'_0 \\ \delta' &: Q' \times \Sigma \to Q' \\ \delta'(q', a) &:= \{(q_1, q_2) | q_1 \in Right(q'), \delta(q_1, a) \ni q_2\} \end{aligned}$$

$$
\begin{aligned}
A'_{acc} := \quad & \{Q_f \in 2^{Q'} \mid \\
& \exists q' \in Q_f s.t. \\
& \quad \forall \ \omega\text{-sequences } \sigma' = \sigma'(0)\sigma'(1)\ldots \in Q_f^{\omega}, \text{ s.t.} \\
& \qquad \sigma'(0) = q' \\
& \qquad \wedge \exists a \in \Sigma : \delta'(\sigma'(i), a) = \sigma'(i+1), \forall i \in \mathbb{N} \\
& \qquad \wedge \mathit{States}(\sigma') = Q_f \\
& \qquad \wedge \omega(\sigma') = Q_f, \\
& \quad \exists \ \omega\text{-sequence } t = t(0)t(1)\ldots \text{ with } t(i) \in \sigma'(i), \forall i \in \mathbb{N} \text{ s.t.} \\
& \qquad t \text{ is continuous} \\
& \qquad \wedge \mathit{Right}(\omega(t)) \cap F \neq \emptyset\} \\
A'_{nacc} := \quad & \{Q_f \in 2^{Q'} \mid \\
& \exists q' \in Q_f s.t. \\
& \quad \forall \ \omega\text{-sequences } \sigma' = \sigma'(0)\sigma'(1)\ldots \in Q_f^{\omega}, \text{ s.t.} \\
& \qquad \sigma'(0) = q' \\
& \qquad \wedge \exists a \in \Sigma : \delta'(\sigma'(i), a) = \sigma'(i+1), \forall i \in \mathbb{N} \\
& \qquad \wedge \mathit{States}(\sigma') = Q_f \\
& \qquad \wedge \omega(\sigma') = Q_f, \\
& \quad \nexists \ \omega\text{-sequence } t = t(0)t(1)\ldots \text{ with } t(i) \in \sigma'(i), \forall i \in \mathbb{N} \text{ s.t.} \\
& \qquad t \text{ is continuous} \\
& \qquad \wedge \mathit{Right}(\omega(t)) \cap F \neq \emptyset\}
\end{aligned}
$$

Theorem 1.
Hyp:
Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \mathcal{F})$ be a non-deterministic Büchi automaton. Let Q', δ', q_0, A'_{acc} and A'_{nacc} be constructed as above.

Aff:
If $\{A'_{acc}, A'_{nacc}\}$ is a partition of $2^{Q'}$ (partitioning condition), then $\mathcal{A}' = (Q', \Sigma, \delta', q'_0, A'_{acc})$ is a Muller automaton s.t. $L(\mathcal{A}) = L(\mathcal{A}')$.

Proof:

$L(\mathcal{A}) \subseteq L(\mathcal{A}')$**:**
Let x be an ω-string in $L(\mathcal{A})$. Let σ be an accepting run of $\mathcal{A}$ on x. Let $t = t(0)t(1)\ldots \in (Q \times Q)^{\omega}$ be the sequence of transitions of σ, i.e. $t(i) = (\sigma(i), \sigma(i+1)), \forall i \in \mathbb{N}$. By hypothesis, we know that $\omega(\sigma) \cap F \neq \emptyset$, hence $\mathit{Right}(\omega(t)) \cap F \neq \emptyset$. Let $\sigma' = \sigma'(0)\sigma'(1)\ldots$ be the run of $\mathcal{A}'$ on x. By construction, it follows that $t(i) \in \sigma'(i), \forall i \in \mathbb{N}$. Let $Q_f \in 2^{Q'}$ be the set of states of $\mathcal{A}'$ s.t. $Q_f = \omega(\sigma')$. Let $n \in \mathbb{N}$ be an integer and $\sigma'_t = \sigma'_t(0)\sigma'_t(1)\ldots \in Q_f^{\omega}$ be a substring of σ' s.t. $\mathit{States}(\sigma'_t) = Q_f$ and $\sigma'_t(i) = \sigma'(n+i)$, $\forall i \in \mathbb{N}$ [4]. Let $t_t = t_t(0)t_t(1)\ldots \in (Q_f \times Q_f)^{\omega}$ be a substring of t s.t. $t_t(i) = t(n+i)$, $\forall i \in \mathbb{N}$. Since we know that $\forall i \in \mathbb{N}$, $t(i) \in \sigma'(i)$, we have that $t_t(i) \in \sigma'_t(i)$.

[4] the $_t$ suffix stands for *tail*

Let $q' \in Q_f$ be a state s.t. $q' = \sigma'_t(0)$. The following statement is hence verified:

$\exists q' \in Q_f s.t.$
$\exists$ *ω-sequence* $\sigma'_t = \sigma'_t(0)\sigma'(1)_t \ldots \in Q'^{\omega}_f$, *s.t.*
$\sigma'(0)_t = q'$
$\wedge \exists a \in \Sigma : \delta'(\sigma'_t(i), a) = \sigma'_t(i+1), \forall i \in \mathbb{N}$ *(clear, because σ'_t is a substring of σ')*
$\wedge\, States(\sigma'_t) = Q_f$
$\wedge\, \omega(\sigma'_t) = Q_f$, *and*
$\exists$ *ω-sequence* $t_t = t_t(0)t_t(1)\ldots$ *with* $t_t(i) \in \sigma'_t(i), \forall i \in \mathbb{N}$ *s.t.*
t_t *is continuous (clear, because t_t is a substring of t)*
$\wedge\, Right(\omega(t_t)) \cap F \neq \emptyset$

It follows that $Q_f \notin A'_{nacc}$. The partitioning property: $2^{Q'} = \{A'_{acc}, A'_{nacc}\}$, implies that $Q_f \in A'_{acc}$, and hence $x \in L(\mathcal{A}')$.

$L(\mathcal{A}) \supseteq L(\mathcal{A}')$:
Let x be an ω-string in $L(\mathcal{A}')$. Let $\sigma' \in Q'^{\omega}$ be the run of $\mathcal{A}'$ on x. So we have: $\omega(\sigma') \in A'_{acc}$. Let $q' \in Q_f$ be a state as defined in A'_{acc} of the construction. Let $\sigma'_t \in Q'^{\omega}_f$ be a substring of σ' and $n \in \mathbb{N}$ s.t. $\sigma'_t(0) = q_0$ and $\sigma'_t(i) = \sigma'(n+i), \forall i \in \mathbb{N}$. By hypothesis, we have:

$\exists \omega$*-sequence* $t = t(0)t(1)\ldots$ *with* $t(i) \in \sigma'(i), \forall i \in \mathbb{N}$, *s.t.*
t *is continuous*
$\wedge\, Right(\omega(t)) \cap F \neq \emptyset$

This sequence t of transitions is an infinite substring of the sequence of transitions triggered by an existing run of $\mathcal{A}$ on the ω-string x. Given the fact that this run recurringly visits an accepting state of F, we have that $x \in L(A)$. □

The following Corollary is a direct property of Muller automata, which allows complementation to be done by a simple complementation operation over the set of accepting subsets.

Corollary 2. *$\mathcal{A}'_c = (Q', q'_0, \delta', A'_{nacc})$ is a Muller automaton complement to $\mathcal{A}$.*
Proof:
From the partitioning condition, we derive that $A'_{nacc} = 2^{Q'} \setminus A'_{acc}$. Thus, the complement automaton is constructed as in Corollary 1. □

As it is written down, the construction appears to have a terrifying complexity. The construction presented here is probably not usable in practice, but brings us a step further in the understanding of Büchi automata, and possibly closer to a more efficient complementation procedure.

The couple of examples below illustrate how the method works in practice. The first step, which consists in writing down the states and transitions, is fairly straightforward. The second step, which determines what the the set F' of accepting subsets should be, requires more work and observation.

5.4 Example 1

We start with one of the simplest examples of non-deterministic complete Büchi automata. As we have seen in section 4, the normal subset-construction gives us a deterministic automaton accepting a language that is larger than that of the original automaton. In fact, this automaton cannot be represented by a deterministic Büchi automaton. Its complement however, can be represented deterministically.

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a non-deterministic Büchi automaton accepting language $L_{\mathcal{A}} = (a + b)^* \cdot b^\omega$ (i.e. all words having a finite number of a's), as shown in Fig.10.

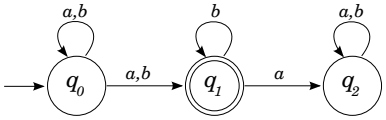


Figure 10: Example 1: a non-deterministic Büchi automaton $\mathcal{A}$

Construction 5.3 gives us the following states and transitions (Fig.11):

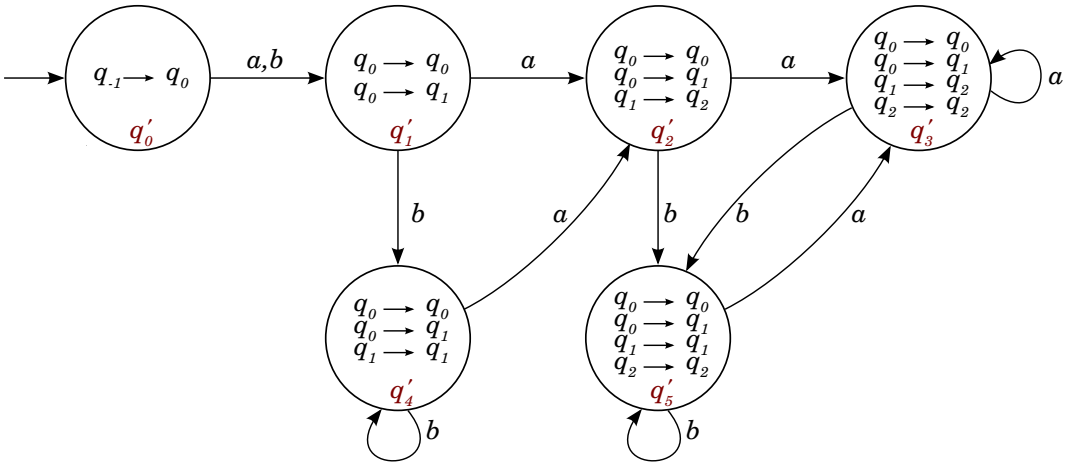


Figure 11: Example 1: the states and transitions constructed from $\mathcal{A}$

Each state holds the set of the last transitions that were used to get there. To avoid a cumbersome notation, the bottom (red) labels constitute a relabelling of the states.

At this point, we do not know if this is indeed a Muller automaton or not. In order to decide this, we have to check the partition condition, which implies checking all relevant sets of states for the existence of an accepting cycle of $\mathcal{A}$ and hence checking their membership to A'_{acc} or A'_{nacc} (for readability, we strip the q character from the label, i.e. q'_4 becomes $4'$). These subsets are the strongly connected subsets.

$$\text{strongly connected subsets: } SCS = \big\{\{3'\}, \{4'\}, \{5'\}, \{3', 5'\}\big\}$$

All the other subsets will obviously belong to $\overline{A'_{acc}} \cap A'_{nacc}$, because there is no way of visiting these states indefinitely. As a matter of fact, it makes no difference whether these *not strongly connected* subsets are included into F' or not, so from now on we will simply omit them.

Let's describe the method to define A'_{acc} and A'_{nacc}:
For each subset $S \in SCS$, we build trees in the following way:

S = {3'}:
$\forall q' \in S$, $\forall q \in Right(q') \cap F$, we build a tree of combined states of $\mathcal{A}$ and $\mathcal{A}'$, starting with (q, q') (Fig.12):

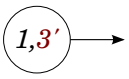


Figure 12: Example 1: first node of the tree for $\{3'\}$

We then recursively build the tree as follows:
Let (q, q') be a given leaf of the tree.
If $\exists$ a parent (q_p, q'_p) of (q, q') s.t. $(q, q') = (q_p, q'_p)$, then stop. Otherwise:
$\forall a \in \Sigma$,
$\forall q'_1 \in S$ s.t. $\delta'(q', a) = q'_1$,
$\forall t \in q'_1$ s.t. $Left(t) = q$,
$\forall q_1 \in Right(t)$, create node (q_1, q'_1) and set it as a child of (q, q').

In our case, we start with node $(1, 3')$. We see in Fig.11 that the unique transition that allows us to remain in S is the a-transition on state $3'$ itself. In that state, the only transition t s.t. $Left(t) = 1$ is $(1, 2)$. So the unique node that is created is $(2, 3')$. From that newly created leaf, the same argument creats a new leaf $(2, 3')$. We then stop, because the leaf has a clone parent, and continuing would only create copies of existing portions of the tree. (Fig.13)

Figure 13: Example 1: the tree for $\{3'\}$

The algorithm only creates one tree in this case. We may now check if $\{3'\}$ belongs to A'_{acc} and/or A'_{nacc}: Since the algorithm stopped without displaying any revisiting of $(1, 3')$, we can deduce that the final state 1 of $\mathcal{A}$ can not be visited in a recurrent manner. Therefore, $\{3'\} \notin A'_{acc}$ and $\{3'\} \in A'_{nacc}$.

Note that the labels on the edges are not used. They will be omitted from now on. The other subsets are evaluated in a similar way:

S = {4'}:
We see here, that a recurring visit of state 1 of $\mathcal{A}$ is unavoidable when visiting state $4'$ of $\mathcal{A}'$ in a recurrent way (Fig.14).

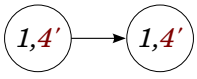


Figure 14: Example 1: the tree for $\{4'\}$

So $\{4'\} \in A'_{acc} \cap \overline{A'_{nacc}}$.

S = {5'}:
Same as for $\{4'\}$: $\{5'\} \in A'_{acc} \cap \overline{A'_{nacc}}$ (Fig.15).

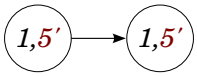


Figure 15: Example 1: the tree for $\{5'\}$

S = {3', 5'}:
This subset requires the building of two trees: one for the final state 1 on state $3'$, and one for the final state 1 on state $5'$ (Fig.16).

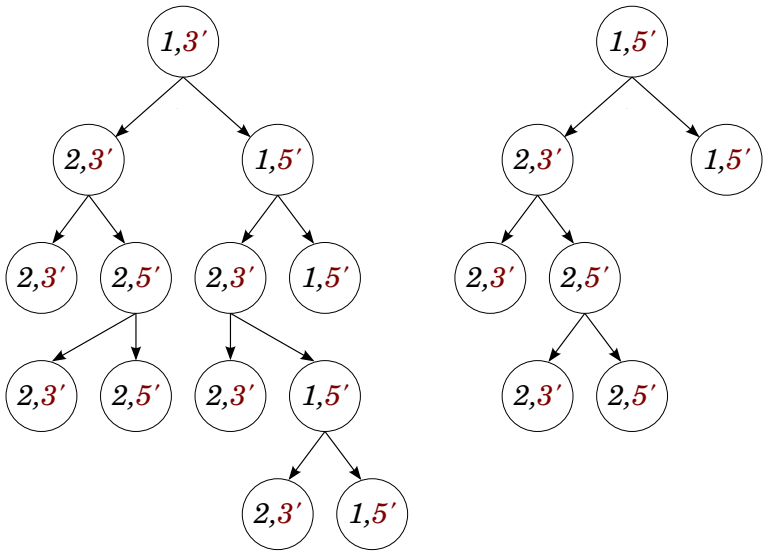


Figure 16: Example 1: the two trees for $\{3', 5'\}$

In both these trees, a leaf is equal to the root iff some states of S were left out, so $\{3', 5'\} \in \overline{A'_{acc}} \cap A'_{nacc}$.

After observing all the relevant subsets, we may safely conclude that $A'_{acc} \cap A'_{nacc} = \emptyset$ and $\mathcal{A}'_{acc} \cup A'_{nacc} = 2^{Q'}$, namely that $\{A'_{acc}, A'_{nacc}\}$ is a partition of $2^{Q'}$. Thus, the automaton depicted in Fig.11 with set of final subsets $F' = \big\{\{4'\}, \{5'\}\big\}$ is a Muller automaton accepting the same language as the Büchi automaton of Fig.10.

The complement can be found easily be complementing the set of accepting subsets. So the complement automaton is $\mathcal{A}'' = (Q', \Sigma, \delta', q'_0, F'')$ with

$$F'' = \big\{\{3'\}, \{3', 5'\}\big\} \cup \{\text{some unloopable subsets}\}.$$

and is depicted on Fig.17.

Some Muller automata are easily translatable into Büchi automata, as it is the case here. Considering the automaton of Fig.17 as Büchi and setting q_5 as a final state, we get an automaton accepting the same language (Fig.18):

This automaton is equivalent (although more complicated) to the automaton of Fig.19, which we know is a true complement to $\mathcal{A}$, because it accepts all words with an infinite number of a's, namely the language $(a \cdot b^*)^\omega$.

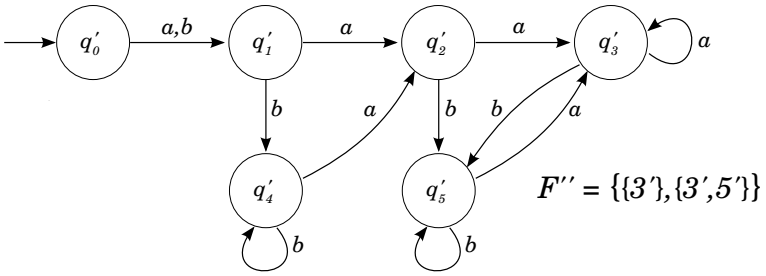


Figure 17: Example 1: The complement Muller automaton to $\mathcal{A}$

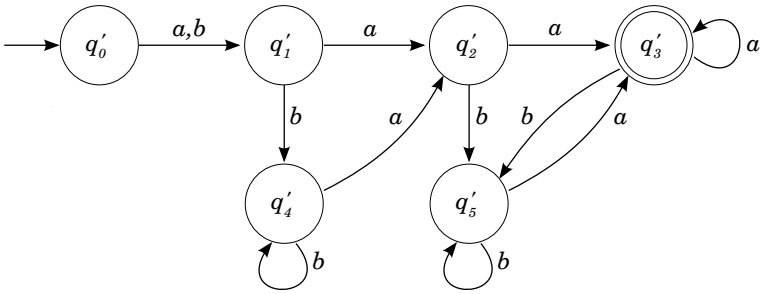


Figure 18: Example 1: The complement Büchi automaton to $\mathcal{A}$

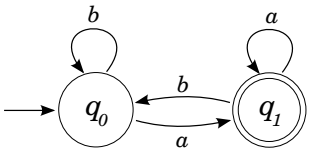


Figure 19: Example 1: a known complement to $\mathcal{A}$

5.5 Example 2

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a deterministic Büchi automaton accepting language $L_{\mathcal{A}} = (a \cdot b^*)^\omega$, as shown in Fig.20.

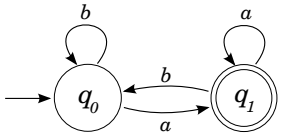


Figure 20: Example 2: a deterministic Büchi automaton $\mathcal{A}$

Construction 5.3 gives us the following states and transitions (Fig.21):

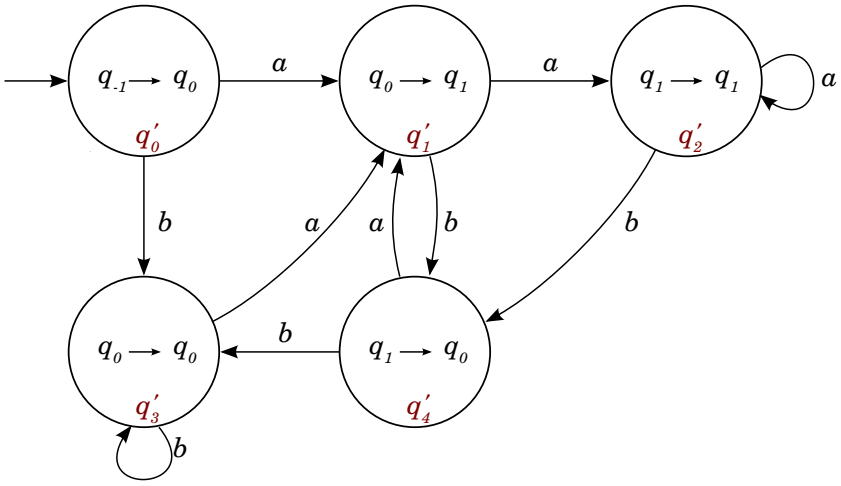


Figure 21: Example 2: the states and transitions constructed from $\mathcal{A}$

We now study the relevant subsets of Q', which are:

$$SCS = \big\{\{2'\}, \{3'\}, \{1', 4'\}, \{1', 3', 4'\}, \{1', 2', 4'\}.\{1', 2', 3', 4'\}\big\}.$$

For each subset, we draw the trees and decide if the subset is in $\mathcal{A}'_{acc}$ or $\mathcal{A}'_{nacc}$.

{2'}:
The tree depicted on (Fig.22) gives us that $\{2'\} \in A'_{acc} \cap \overline{A'_{nacc}}$.

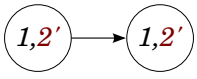


Figure 22: Example 2: the tree for $\{2'\}$

{3'}:
There is no final state of $\mathcal{A}$ in $Right(3')$, so no tree can be built. Thus, no final loop of $\mathcal{A}$ can be concluded (or even started) when looping within $\{3'\}$. So $\{3'\} \in \overline{A'_{acc}} \cap A'_{nacc}$.

{1', 4'}:
The tree depicted on (Fig.23) gives us that $\{1', 4'\} \in A'_{acc} \cap \overline{A'_{nacc}}$.

Figure 23: Example 2: the tree for $\{1', 4'\}$

{1', 3', 4'}:
The tree depicted on (Fig.24) gives us that $\{1', 3', 4'\} \in A'_{acc} \cap \overline{A'_{nacc}}$.

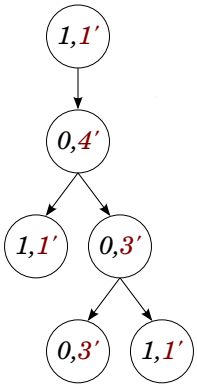


Figure 24: Example 2: the tree for $\{1', 3', 4'\}$

{1', 2', 4'}:
The trees depicted on (Fig.25) gives us that $\{1', 2', 4'\} \in A'_{acc} \cap \overline{A'_{nacc}}$.

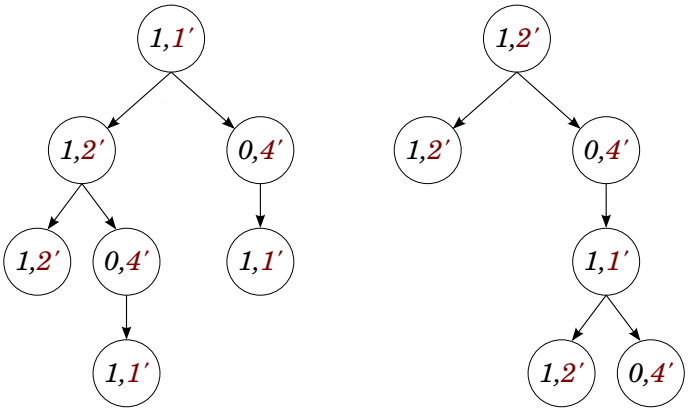


Figure 25: Example 2: the tree for $\{1', 2', 4'\}$

{1', 2', 3', 4'}:
The trees depicted on (Fig.26) give us that $\{1', 2', 3', 4'\} \in A'_{acc} \cap \overline{A'_{nacc}}$.

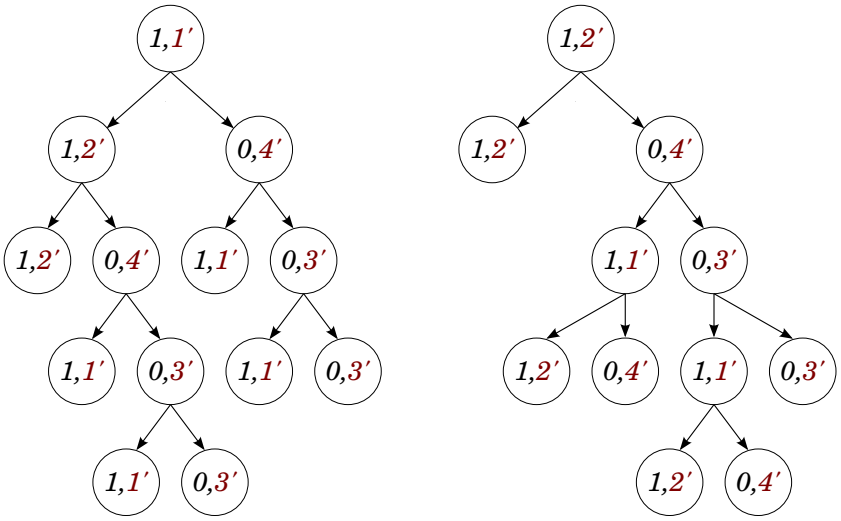


Figure 26: Example 2: the tree for $\{1', 2', 3', 4'\}$

So, $A'_{acc} = \big\{\{2'\}, \{1', 4'\}, \{1', 3', 4'\}, \{1', 2', 4'\}.\{1', 2', 3', 4'\}\big\}$ and $A'_{nacc} = \big\{\{3'\}\big\} \cup \{\text{non-looping subsets}\}$. Finally, $\{A'_{acc}, A'_{nacc}\}$ is a partition of 2^Q and the automaton $\mathcal{A}' = (Q', \Sigma, \delta, q_0, \mathcal{A}_{acc})$, as depicted in Fig.21 is an automaton complement to $\mathcal{A}$ of Fig.20.

6 Other Ideas

The construction described above is the result of numerous observations on Büchi automata, and a number of unsuccessful or unachieved attempts. Some of them are sketched below. Even if these attempts were unfruitful, they were all part of an essential process of better understanding the structure and issues of Büchi automata and what their complement should look like.

6.1 Extending Complementation of Deterministic Büchi Automata

One of the early approaches was based on the observation that Büchi automata that are determinised via the subset construction and then complemented using the common complementation algorithm, give results that, in several cases, are not too far off from the actual complement automaton.

In theory, the subset construction generally outputs an automaton $D(\mathcal{A})$ accepting a superset of the language of the input automaton $\mathcal{A}$. So we have $L_{\mathcal{A}} \subseteq L_{D(\mathcal{A})}$[5]. The complementation algorithm of section 3.2 applied to the deterministic automaton $D(\mathcal{A})$ outputs a non-deterministic automaton $C(D(\mathcal{A}))$, which is a true complement to $D(\mathcal{A})$. So basically, we get the following inequalities:

$$\begin{aligned} & L_{\mathcal{A}} \subseteq L_{D(\mathcal{A})} \\ \Leftrightarrow \quad & \overline{L_{\mathcal{A}}} \supseteq \overline{L_{D(\mathcal{A})}} \\ \Leftrightarrow \quad & \overline{L_{\mathcal{A}}} \supseteq L_{C(D(\mathcal{A}))} \end{aligned}$$

The underlying question is: *What should we add to $C(D(\mathcal{A}))$ for it to become a complement of $\mathcal{A}$?* Formally, if $L_{C(D(\mathcal{A}))} \cup X = \overline{L_{\mathcal{A}}}$, what does X look like, and how is it related to the automaton $C(D(\mathcal{A}))$?

Let's look at an example. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be an non-complete non-deterministic Büchi automaton accepting language $(a + b)^* \cdot b^\omega$ as in Fig.27.

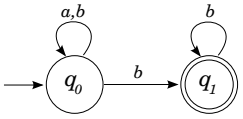


Figure 27: Example of non-deterministic Büchi automaton $\mathcal{A}$

The subset construction gives us a deterministic automaton $D(\mathcal{A})$ accepting language $(a^* \cdot b)^\omega$ (Fig.28).

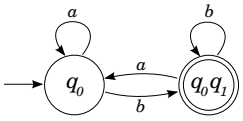


Figure 28: Deterministic version of the automaton of Fig.27

If we now apply the complementation procedure of 3.2 to this deterministic complete Büchi automaton, we get the non-deterministic Büchi automaton of Fig.29:

But it would suffice to add a b-transition from q_0' to q_0q_1 (as in Fig.30) in order to make it an actual complement automaton to $\mathcal{A}$.

[5] $L_{\mathcal{A}}$ is the language accepted by an automaton $\mathcal{A}$.

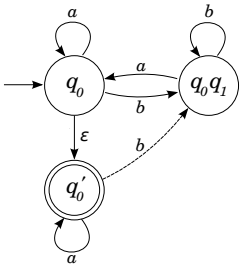


Figure 29: Complement of the automaton of Fig.28

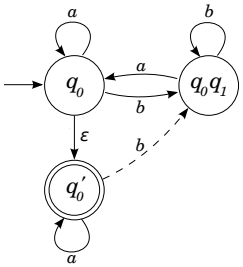


Figure 30: Complement of the automaton of Fig.27

Unfortunately, no generalisation of this method could be found. It is also clear, from more complicated examples, that it is not sufficient to add extra transitions to find the complement automaton; new states may also need to be added.

6.2 2-DET Complementation

Büchi automata with non-determinism degree two[6] (2-DET) possess unique properties. In particular, any Büchi automaton can be reduced to a 2-DET automaton (see [3]). Some investigations have been done to find out if 2-DET automata were easier to complement than others. If that is the case, then we could systematically reduce any Büchi automaton to 2-DET and then complement it. Even so no evidence of that was found, it can still remain a good idea to apply such a reduction, be it solely to reduce the complexity of the complementation procedure.

6.3 Undeterminisation Method

We know that the subset-construction, when applied to a non-deterministic Büchi automaton, returns a deterministic Büchi automaton accepting a superset of the original language. But since the original and the determinised automata can be very different, it is hard to see what exactly has been added. The idea of *undeterminisation* is to revert the determinised automaton $D(\mathcal{A})$ to a non-deterministic one $U(D(\mathcal{A}))$, resembling the original, such that $D(U(D(\mathcal{A}))) = D(\mathcal{A})$. That way, it is easy to see what exactly was added to the automaton during the determinisation process, possibly giving us some information about what to add to the complemented automaton of section 6.1. Here's an example illustrating how it works:

[6]The non-determinism degree is defined by the maximum number of states that can be reached from a single state, with a unique symbol.

Let us consider the non-deterministic Büchi automaton $\mathcal{A}$ of Section 6.1 (Fig.27), and consider its determinised version $D(\mathcal{A})$ (Fig.28). The undeterminisation $U(D(\mathcal{A}))$ of $D(\mathcal{A})$ operates as follows:

Write down the states as in the original automaton namely, set states q_0 and q_1. Then reproduce each transition of $D(\mathcal{A})$ into $U(D(\mathcal{A}))$ (e.g. if an a-transition exists from q_0 to q_0q_1 in $D(\mathcal{A})$, then set a-transitions in $U(D(\mathcal{A}))$ between q_0 and q_0, and between q_0 and q_1). The result is given in Fig.31.

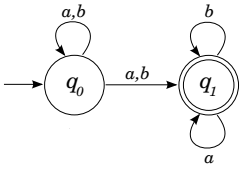


Figure 31: A non-deterministic automaton

It is now very clear what has been added to $\mathcal{A}$ during the determinisation process (namely the looping a-transition on q_1). The idea is to then use this information to *complete* the complemented automaton $C(D(\mathcal{A}))$ into $X(C(D(\mathcal{A})))$ s.t. $L_{X(C(D(\mathcal{A}))} = \overline{L_{\mathcal{A}}}$. At this time however, no general relation was found between X and the added transitions of $U(D(\mathcal{A}))$, but this appears to be a valuable tool for future research, to understand the mechanics of the subset-construction.

6.4 Memory of Visited States

The construction of section 5.3 originated from an idea of Prof. Ultes-Nitsche, which consisted in a subset-construction which labelled the states according to the previously visited states (instead of the transitions). It allows us to make a difference between the states that have visited an accepting state, and those which have not. Because the construction records a separate history of visited states for each state of the original automaton within a single combined state, it also solves the problem of loss of information that occurs when combining states.

Let's look at an example. Let $\mathcal{A}$ be a non-deterministic Büchi automaton as depicted in Fig.32

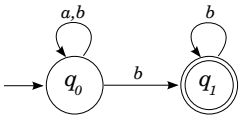


Figure 32: Example of non-deterministic Büchi automaton $\mathcal{A}$

The construction operates just as the subset construction, but in addition to the label of the state, we add, for each state of $\mathcal{A}$ within the label, the set of states of $\mathcal{A}$ that were visited. We can then set as *accepting* any state that does not contain a final state in its visited states sets (Fig.33).

In this case, the construction gives us a correct complement automaton. However, other constructions, such as in Fig.35, constructed from the automaton of Fig.34, illustrate problems when setting the final states. In this case, clearly, the automaton is not *accepting enough*, and the issue is not immediately solvable, because the bottom-right state should be accepting and non-accepting at the same time, for the constructed automaton to be a real complement to Fig.34.

Actually, the problem is much more basic: the construction will only allow deterministic Büchi automata to be constructed, and we know that we should obtain non-deterministic automata in general.

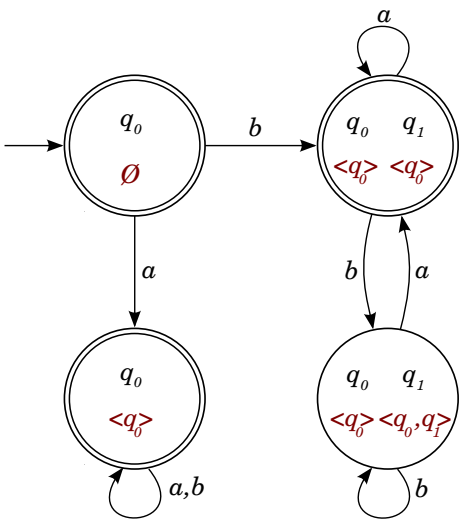


Figure 33: A subset-construction recording the visited states

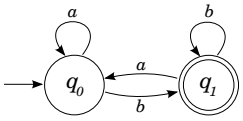


Figure 34: A bad case for the construction

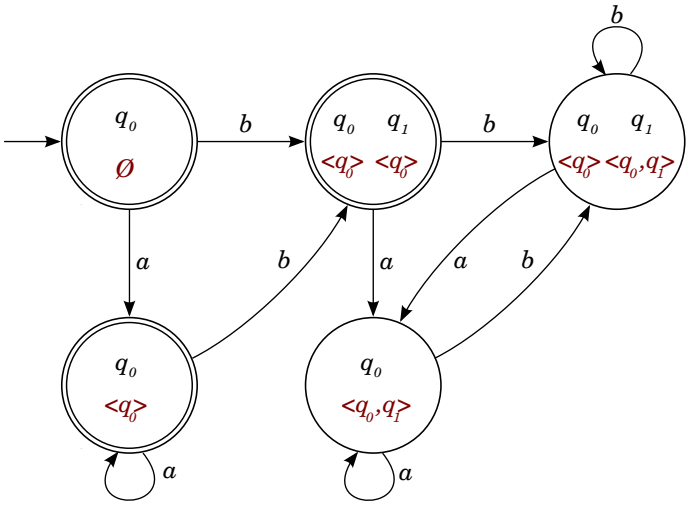


Figure 35: A bad case for the construction

Furthermore, the construction may sometimes be ambiguous, because it is not always clear, in a non-deterministic automaton, if a state has been visited or not. The example of Fig.36 illustrates this issue, where upon building state q_3 of the constructed automaton, it is unclear if we have to include states q_1, q_2, both, or none, in the set of visited states, because two independent paths responding to the same string may lead to that state.

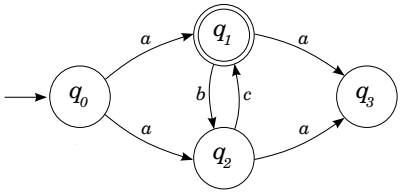


Figure 36: Another bad case for the construction

These problems having been mentioned, this construction was a cornerstone to the construction from section 5.3. The idea of keeping a memory of the visited states (or transitions) within the state labels proves to be very useful. The downside of this way of building automata is, of course, a huge theoretical complexity.

7 Conclusion

7.1 The Research

This paper is the result of a very deep and intensive research on Büchi automata. The initial goal of the research was a little blurry, but it was about finding new ways of complementing Büchi automata, ways that made use of the apparent relevance of the subset-construction. It became very clear that any complementation method for non-deterministic automata would be tightly linked to determinisation.

The biggest problem, when dealing with infinity (as in Büchi automata), and with uncertainty (as with non-determinism), is the difficulty of obtaining a valuable mental representation of what language is accepted by the automaton, what a deterministic version could look like, and what the complement should be. In contrast to the straightforwardness of determinisation and complementation of automata on finite words, the corresponding concepts are hard to grasp when dealing with automata on infinite sequences.

After some unsuccessful attempts, the research led to a complementation procedure that works, under some conditions. It would have been nice to find a universal and efficient method, but this has just to be left to future work. This construction has nevertheless many interesting aspects, being an entirely new way of dealing with the matter.

7.2 Future Work

Future research might consider such a construction again, but the idea would be to bypass the step where we have a Muller automaton, and directly deduce the complement. It appears however unlikely that we can find a complement without having a clear deterministic representation of the original automaton at some point, and the Muller automaton does the trick here.

Again, the main thread of this project was to have a better comprehension of automata on infinite words, and in that way, it was very interesting, fruitful and rewarding.

7.3 Acknowledgements

The author would like to thank Prof. Ulrich Ultes-Nitsche for his unconditional support, his availability and the many interesting discussions that allowed the completion of this project.

References

[1] J. E. Hopcroft, R. Motwani and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Pearson Education, Boston MA, 2007 (third edition)

[2] W. Thomas, *Automata on Infinite Objects*, In: Handbook of Theoretical Computer Science, Elsevier Science Publishers B.V., 1990

[3] U. Ultes-Nitsche, *A power-set construction for reducing Büchi automata to non-determinism degree two*, Information Processing Letters 101 (2007) 107-111

[4] R. McNaughton, *Testing and generating infinite sequences by a finite automaton*, Inform. and Control 9 (1966) 521-530

[5] F. Nießner, U. Nitsche, P. Ochsenschläger, *Deterministic ω-Regular Liveness Properties*, Proceedings of the 3rd International Conference, Developments in Language Theory (1997) 237-247

[6] J. Allred, *ω-Language preserving operations on Büchi Automata*, Bachelor Thesis, University of Fribourg, Switzerland (2008)