Complementing Büchi Automata

Guillaume Sadegh

Technical Report n^o0903, May 2009
revision 2073

Model checking is a field of formal verification which aims to automatically check the behavior of a system with the help of logic formulæ. SPOT is a model checking library which relies on one approach: the automata-theoretic approach.

In this technique, both system and formula are expressed with ω-automata, which are automata on infinite words. SPOT provides several algorithms to do operations with these automata.

However, an algorithm is missing: the complementation of Büchi automata. Because of its high complexity this algorithm is rarely used in practice, but it does not lack theoretical interests since SPOT uses a particular variant of ω-automata.

We present an implementation of this algorithm in SPOT.

Le model checking est un domaine de la vérification formelle, qui permet de vérifier le comportement d'un système à travers des formules logiques. SPOT est une bibliothèque de model checking qui repose sur une des techniques du domaine : l'approche automate.

Dans cette approche du model checking, le système et les formules sont représentés sous forme d'automates acceptant des mots de longueur infinie, et plus particulièrement des automates de Büchi. SPOT propose de nombreux algorithmes aux utilisateurs de la bibliothèque pour manipuler ce type d'automates, en vue d'applications au model checking.

Pourtant, un algorithme est manquant : celui de la complémentation d'automates de Büchi (qui produit un automate reconnaissant la négation du langage initialement reconnu). Cet algorithme est peu utilisé dans la pratique à cause de sa forte complexité, mais il ne manque pas d'intérêt du point de vue théorique.

Nous présenterons une implémentation d'un tel algorithme dans SPOT.

Keywords
Büchi automata, complementation, determinization, Safra construction, Streett to Büchi transformation.

Laboratoire de Recherche et Développement de l'Epita
14-16, rue Voltaire – F-94276 Le Kremlin-Bicêtre cedex – France
Tél. +33 1 53 14 59 47 – Fax. +33 1 53 14 59 22
`sadegh@lrde.epita.fr` – http://www.lrde.epita.fr/

Copying this document

Contents

Introduction

Model checking is a field of formal verification which aims to automatically check the behavior of a system with the help of logic formulæ. The goal of this verification is to confirm whether a system satisfies a set of properties.

SPOT (Duret-Lutz and Poitrenaud, 2004) is a model checking library that relies on one approach: the automata-theoretic approach. In this technique, both system and formula are expressed with ω-automata, which are automata on infinite words.

The automaton that represents the system to verify has for language $\mathscr{L}(A_M)$, the set of all possible executions of the system. Properties to verify are expressed with the negated formula automaton, whose language, $\mathscr{L}(A_{\neg\varphi})$ is the set of all executions that would invalidate the formula. Thus, the intersection of these two automata will produce an automaton whose language is the set of all possible executions of the system that would invalidate the formula. With this last automaton, a procedure called *emptiness check* checks whether the recognized language is empty. When this language is empty, the properties satisfy the system. However, if this language accepts a non-empty word, then there exists a run in the system that invalidates the logic formula.

SPOT provides a set of algorithms to work with a kind of ω-automata called Transition-based Generalized Büchi Automaton (TGBA). Most of the common operations on ω-automata used in model checking already exist in the library, but the complementation, which is a common procedure with automata on finite words is missing.

Complementing an automaton means constructing another automaton that would recognized the complemented language of the initial one. Even though this operation would be really useful in model checking since formulæ are negated to represent unexpected behaviors, in practice this operation is unrealistic. This is due to the lack of efficient algorithms, since even best ones have an exponential complexity. Therefore, model checking avoids complementation by different ways.

For example, to avoid to complement the automaton of a logic formula, algorithms that traduce formulæ into automata use the negation of the logic formula before the translation.

However, some systems to express properties like the expressions ω-regular cannot be negated easily. For those kind of systems, having a complementation on automata is useful.

Another purpose of implementing this algorithm is also to enrich our library with another operation. Since SPOT is a library that provides algorithm for its users, anyone could be interesting in this operation.

Moreover, since SPOT uses Transition-based Generalized Büchi automata as kind of ω-automata, it could also be an interesting theoretical subject to show how this kind of automata can improve the complexity of the algorithms that we will present later.

Outline: Chapter 1 will give some background on ω-automata, with definitions and an overview of the operations that will interest us.

In Chapter 2 we will discuss how to complement a Transition-based Generalized Büchi automaton by describing the two main algorithms we used. Chapter 3 will present how these algorithms are implemented in SPOT, and will discuss some benchmarks. Chapter 4 will present future works to accomplish on the complementation, and we will conclude in the last chapter.

Chapter 1

Automata on infinite words

While classical finite automata recognize words of finite length, ω-automata recognize words of infinite length but with a finite number of states.

1.1 Definitions

1.1.1 ω-automata

Figure 1.1 presents the graphical representation of ω-automata.

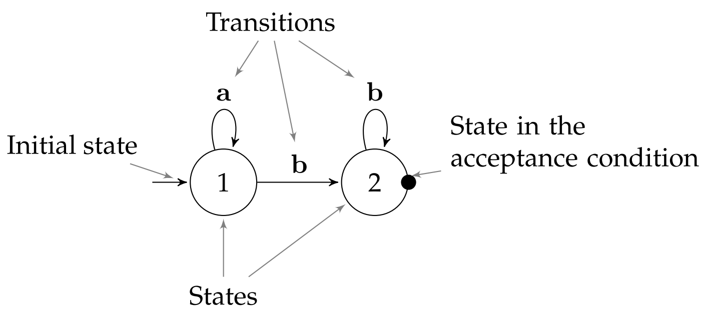


Figure 1.1: An ω-automaton

Definition 1.1.1. An ω-automaton is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ with:

- Q a finite set of states.
- Σ the alphabet.
- $\delta : Q \times \Sigma \rightarrow 2^Q$ the transition function.
- $q_0 \in Q$ the initial state.
- F the acceptance condition, which is a formula on states. The difference with automata on finite words lies in these acceptance conditions.

Definition 1.1.2. A sequence of states $\pi = \pi_0, \pi_1, \pi_2, \ldots \in Q^\omega$ is called a *run* over an infinite word $\sigma \in \Sigma^\omega$, if $\pi_0 = \sigma_0$ and for every i such as σ_i is the ith letter of σ, $\pi_{i+1} \in \delta(\pi_i, \sigma_i)$.
An accepting run is a run that satisfies the acceptance condition.

Example. A run of the automaton in Figure 1.1 could be $\pi =$①, ①, ②, … with the state ② meet infinitely often.

Definition 1.1.3. The *infinity set* of a run π is the set of states that occurs infinitely many time in π and is denoted $inf(\pi)$.

Example. With the previous example, $inf(\pi) = \{$②$\}$, since the state ② occurs infinitely often in the run.

Definition 1.1.4. The language of $\mathcal{A}$ is the set of all the inputs with an accepting run of $\mathcal{A}$, and is denoted $\mathscr{L}(\mathcal{A})$

Example. The language of the automaton in Figure 1.1 with an acceptance condition that requires to visit infinitely often ② to be accepting would be: a word with the letter **a** finitely often and then the letter **b** infinitely often ($\mathbf{a}^\star Bchi^\omega$).

Remark 1.1.5. The number of successors of a state $q \in Q$ for the label $\sigma \in \Sigma$ is denoted $|\delta(q, \sigma)|$.

The properties on ω-automata can be divided in three criteria used to denote the different kinds of automata.

- Their *mode of transition-function*, that defines the semantic of a run of an automaton.
- Their type of *acceptance condition*, that defines whether a run of the automaton is accepting.
- Their *labeling*, on states for most *model checkers* and algorithms, or on transitions for a few ones like SPOT.

These three criteria are detailed in the next sections.

1.1.2 Mode of transition-function

The mode of transition-function defines how a run of an automaton occurs. We usually distinguish: deterministic automata, non-deterministic automata and universal automata.

- An automaton is called *deterministic* when each transition function $\delta(q, \sigma)$ has exactly one successor. More formally, we can say: iff $\forall \sigma \in \Sigma_\mathcal{A}, \forall q \in Q_\mathcal{A}, |\delta(q, \sigma)| \leqslant 1$, then $\mathcal{A}$ is deterministic.

  Example. Figure 1.2a represents a deterministic automaton since each state has exactly one successor for each letter σ.

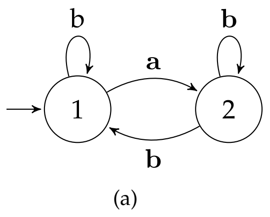
An automaton where transition function $\delta(q, \sigma)$ may have more than one successor is called:

- *non-deterministic*, when the semantic is to choose non-deterministically one state as successor when a transition function returns more than one successor.

  Example. Interpreted as a non-deterministic ω-automaton, a run of the word **ba** over the automaton in Figure 1.2b will reach states ①, ① OR ①, ②.

- *universal*, when the semantic is to visit all the successors when a transition function returns more than one successor.

  Example. Interpreted as a universal ω-automaton, a run of the word **ba** over the automaton in Figure 1.2b will reach states ①, ① AND ①, ②.



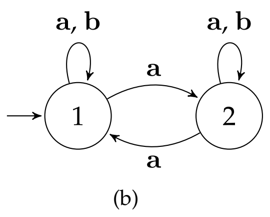


Figure 1.2: Deterministic, Non-deterministic and Universal automata.

1.1.3 Acceptances conditions

The acceptance condition is a formula on states that defines the semantic of an accepting run. We will present some acceptance conditions that will be useful for the complementation.

Büchi acceptance condition

Büchi (1962) was the first to introduce ω-automata with his acceptance condition. The Büchi acceptance condition is the most adapted to model checking since it supports all the operations presented in Section 1.2: Operations on ω-automata.

With his definition, the acceptance condition F is a set of states, and a run must visit infinitely often some states from F to be accepting.

More formally, a run π of a Büchi automaton with $F \subseteq Q$ as acceptance condition is accepting, iff $inf(\pi) \cap F \neq \emptyset$.

Example. Figure 1.3 presents a Büchi automaton with its states in the acceptance condition F marked with ●. A run of this automaton is accepting if it visits infinitely often states ② OR ③.

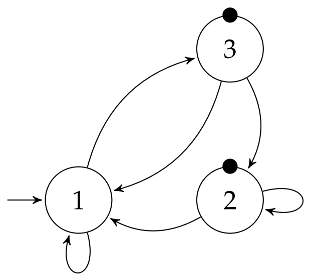


Figure 1.3: A Büchi automaton.

However, this acceptance condition has some drawbacks since deterministic Büchi automata are less expressive than non-deterministic ones (see Subsection 1.2.1 for the proof and Subsection 2.1.1 for an example). Moreover, with other acceptance conditions like Generalized Büchi acceptance condition, we can have automata that recognize the same language with a smaller number of states and transitions. Figure 1.7 presents several automata that recognize the same language but with different acceptance conditions.

There is a dual acceptance condition to Büchi automata which is called co-Büchi acceptance condition. A run π of a co-Büchi automaton with F as acceptance condition is accepting, iff $inf(\pi) \cap F = \emptyset$ with $F \subseteq Q$. This dual acceptance condition is interesting since the same deterministic automaton interpreted either with the Büchi acceptance condition or the co-Büchi acceptance condition will recognize two complementary languages. However, this property only works with deterministic automata.

Generalized Büchi acceptance condition

Generalized Büchi automata are a variant of Büchi automata that is more succint, since it allows to have automata that recognize the same language than Büchi automata but with a smaller number of states and transitions.

The Generalized Büchi acceptance condition has more than one set of acceptance conditions. A run is accepting if it passes through at least one state of each set infinitely often. Figure 1.4 illustrates this acceptance condition.

More formally, the definition is $\forall i \mid inf(\pi) \cap F_i \neq \emptyset$ with $F = \{F_1, F_2, \cdots, F_n\}$ and $F_i \subseteq Q$.

Example. Figure 1.4 presents a generalized Büchi automaton with an accepting run if a run visits infinitely often both acceptance conditions (states denoted with ● and ○).

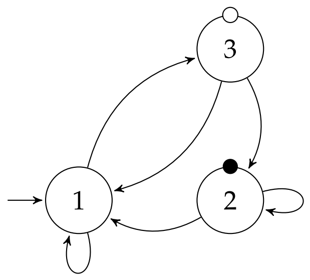


Figure 1.4: A generalized Büchi automaton.

Muller acceptance condition

Because non-deterministic Büchi automata (generalized or not) and deterministic Büchi automata are not equivalent, Muller (1963) proposed another acceptance condition for deterministic ω-automata. The acceptance condition he suggests is to specify explicitly all the "good" infinity sets. A run is accepting if the accepting set is included into the infinity set of visited states. The formal definition for this acceptance condition is: $inf(\pi) \in F$, with $F \subseteq 2^Q$, the set of set of accepting states.

Muller acceptance condition is a generalization of all acceptance qconditions listed here. However, to work with ω-automata that have the same expressiveness for deterministic automata and non-deterministic automata we prefer Rabin and Streett acceptance conditions, because in practice, storing all the acceptance conditions is not interesting.

Rabin acceptance condition

The Rabin (1969) condition consists of pairs (L, U) of subsets of Q. A run is accepting if there exists a pair $(L_i, U_i) \in F$ such that a run visits infinitely often L_i and only finitely often U_i.

More formally this definition is: $\exists i : inf(\pi) \cap L_i \neq \emptyset \wedge inf(\pi) \cap U_i = \emptyset$ with $F \subseteq 2^Q \times 2^Q$.

Example. Figure 1.5 presents a Rabin automaton with two pairs of acceptance conditions: $\{(●, ■), (○, □)\}$. A run of this automaton is accepting if it visits infinitely often ② and finitely often ③ (pair ●, ■) OR if it visits infinitely often ③ and finitely often ② (pair ○, □).

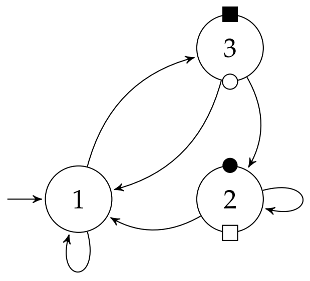


Figure 1.5: A Rabin automaton

Streett acceptance condition

Streett (1982) suggested a dual acceptance condition of Rabin's. So this condition consists in pairs (L_i, U_i) of subsets of Q. A run is accepting if there exists an i such that if it visits infinitely often L_i, then U_i is also infinitely often visited. More formally this definition is: $\forall i : inf(\pi) \cap L_i \neq \emptyset \Rightarrow inf(\pi) \cap U_i \neq \emptyset$ with $F \subseteq 2^Q \times 2^Q$, or, to explicit the duality, $\forall i : inf(\pi) \cap L_i = \emptyset \vee inf(\pi) \cap U_i \neq \emptyset$.

An interesting property is due to the duality of Streett and Rabin acceptance conditions that produce an equivalence between Deterministic Streett and the negation of Deterministic Rabin. We usually denote $\text{DS} \equiv \overline{\text{DR}}$.

Example. With Figure 1.5 that represents a Rabin automaton, its interpretation as a Streett automaton will have accepting runs if, when it visits infinitely often ②, then ③ is also visited infinitely often (pair ●, ■) OR when it visits infinitely often ③, then ② is also visited infinitely often (pair ○, □).

Switching between classes

With all these acceptance conditions we can have automata with the same expressiveness. Deterministic (Generalized) Büchi automata is an exception since they are less expressive.

Therefore there exists some procedures (Löding, 1998) to transform an automaton from one class to another. However, most of these transformations are exponential. Safra (1989) gives a figure to illustrate the complexity of these transformations. This illustration is presented in Figure 1.6.

Summary on acceptance conditions

All this acceptances conditions can be summarized in Table 1.1.

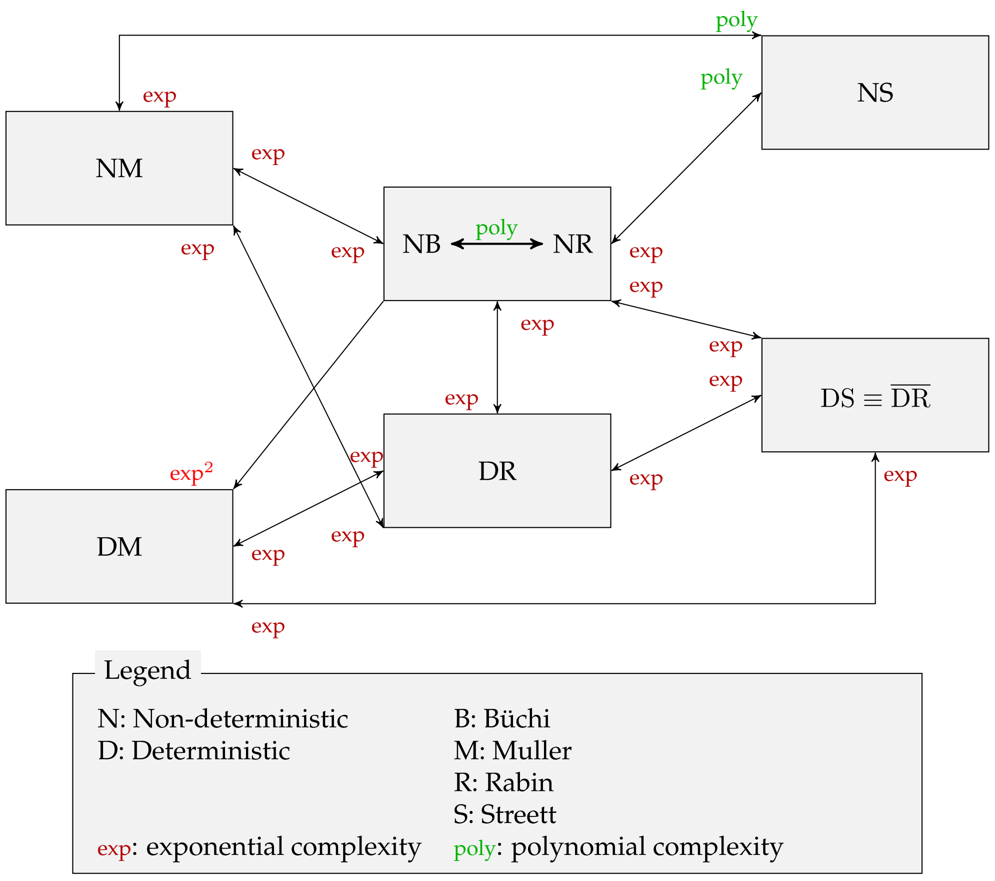


Figure 1.6: The complexity of transformation between the different classes of ω-automata.

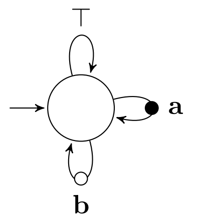
Name	**Domain**	**Semantics**
Büchi	$F \subseteq Q$	$inf(\pi) \cap F \neq \emptyset$
Co-Büchi	$F \subseteq Q$	$inf(\pi) \cap F = \emptyset$
Generalized Büchi	$F \subseteq 2^Q$	$\forall i : inf(\pi) \cap F_i \neq \emptyset$
Muller	$F \subseteq 2^Q$	$inf(\pi) \in F$
Rabin	$F \subseteq 2^Q \times 2^Q$	$\exists i : inf(\pi) \cap L_i \neq \emptyset \wedge inf(\pi) \cap U_i = \emptyset$
Streett	$F \subseteq 2^Q \times 2^Q$	$\forall i : inf(\pi) \cap L_i = \emptyset \vee inf(\pi) \cap U_i \neq \emptyset$

Table 1.1: Common acceptance conditions on ω-automata

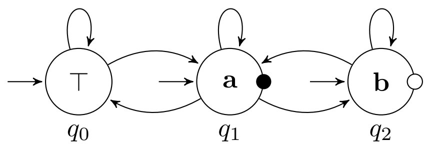
In order to complement Büchi automata, we will switch to Rabin acceptance condition, then to Streett acceptance condition and finally return on Büchi acceptance condition. This process will be presented in Chapter 2.

1.1.4 Labeling

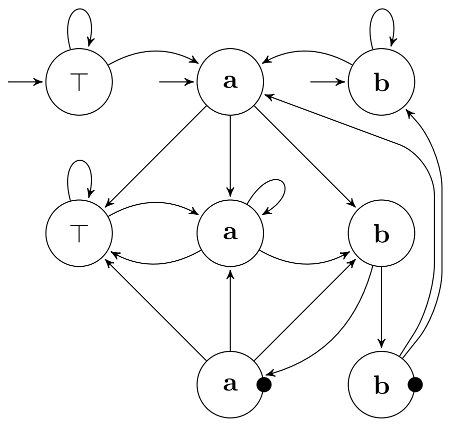
In the previous example, acceptance conditions were labeled on states. However, labeling on transitions can produce automata with a smaller number of states and transitions. Figure 1.7 presents three automata that recognize the same language. With labeling on transitions the automaton has one state and three transitions but three states and seven transitions with labeling on states.



(a) A Transition-based Generalized Büchi Automaton



(b) The equivalent Generalized Büchi Automaton



(c) The equivalent Büchi Automaton

Figure 1.7: Transition-based Generalized Büchi automata, Generalized Büchi and Büchi automata.

The model checking library SPOT is designed to work with non-deterministic Transition-based Generalized Büchi Automata (TGBA), and implements several algorithms that work for this kind of ω-automata. However, since the algorithms we have implemented are working on states, we will use states labeled automata up to the final automaton that will be labeled on transitions.

Switching between state-based and transition-based Büchi automata is presented in (Duret-Lutz, 2007, section 3.3.5). From a state-based Büchi automaton with n states the transformation algorithm results a transition-based Büchi automaton with also n states. However, for a transition-based Büchi automaton with n states, the resulting state-based Büchi automaton will have at most $2n$ states.

1.2 Operations on ω-automata

Some operation on ω-automata are common, such as

- The *product* or intersection of two ω-automata $\mathcal{A}_1$ and $\mathcal{A}_2$, that produces an automaton that recognizes $\mathscr{L}(\mathcal{A}_1) \cap \mathscr{L}(\mathcal{A}_2)$. The operation is useful in model checking to produce an automaton from the system automaton and the formula automaton that will recognize runs of the system that invalidate the formula.

  For two automata with n_1 and n_2 states, the product automaton will have $O(n_1 \times n_2)$ states.

- The *sum*, or union of two ω-automata $\mathcal{A}_1$ and $\mathcal{A}_2$, that produces an automaton that recognizes $\mathscr{L}(\mathcal{A}_1) \cup \mathscr{L}(\mathcal{A}_2)$. This operation is quite easy to implement for non-deterministic automata since it only requires to add an initial state to join initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$.

  For two automata with n_1 and n_2 states, the sum automaton will have $O(n_1 + n_2)$ states.

- The *emptiness check*, that is common for Büchi automata and checks whether an ω-automaton recognizes a non-empty word.

1.2.1 Determinization

The determinization operation consists in the transformation of a non-deterministic ω-automaton into a deterministic one. However, any non-deterministic Büchi automaton cannot be determinized into an automaton with Büchi acceptance conditions.

Proof. Given a non-deterministic Büchi automaton $\mathcal{A}$ that recognizes an ω-word $(\mathbf{a} + \mathbf{b})^{\star} b^{\omega}$. This notation means that "$\mathbf{a}$" must be recognized only finitely often but "$\mathbf{b}$" infinitely often. This automaton is represented by Figure 1.8.

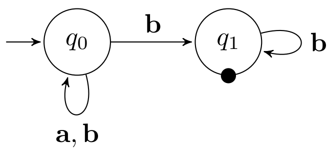


Figure 1.8: A Büchi automaton to determinize.

Now suppose that we have a deterministic Büchi automaton $\mathcal{B} = (\{\mathbf{a}, \mathbf{b}\}, Q, \delta, \{q_0\}, F)$ which recognizes the same language as $\mathcal{A}$.
Then:

$u_0 = \mathbf{b}^{\omega}$, so $u_0 \in \mathscr{L}(\mathcal{A})$ and there exists a finite prefix v_0 of u_0 that brings $\mathcal{B}$ to F.

$u_1 = v_o\mathbf{ab}^{\omega}$, so $u_1 \in \mathscr{L}(\mathcal{A})$ and there exists a finite prefix $v_0\mathbf{a}v_1$ of u_1 that brings $\mathcal{B}$ to F.

$\vdots$

$u_n = v_{n-1}\mathbf{ab}^\omega$, so $u_n \in \mathscr{L}(\mathcal{A})$ and there exists a finite prefix $v_0\mathbf{a}v_1\mathbf{a}\cdots\mathbf{a}v_n$ of u_n that brings $\mathcal{B}$ to F.

Since the number of states Q is finite, there exists i and j, $0 \leqslant i < j$ such as words $v_0\mathbf{a}v_1\mathbf{a}\cdots\mathbf{a}v_i$ and $v_0\mathbf{a}v_1\mathbf{a}\cdots\mathbf{a}v_j$ bring to the same state. Then $m = v_0\mathbf{a}v_1\mathbf{a}\cdots\mathbf{a}v_i(\mathbf{a}\cdots v_j)^\omega$ is accepted by $\mathcal{B}$. But since m has an infinity of $\mathbf{a}$, it cannot be recognized by $\mathcal{A}$.

□

This theorem implies that to determinize a Büchi automaton, the Büchi class cannot be used.

The first construction for determinizing is due to Muller (1963) which appears to be faulty. McNaughton (1966) gave a theorem to prove that any non-deterministic Büchi automaton can be converted into a Deterministic Muller automaton. To prove his theorem, McNaughton proposed a construction with a large blow-up in $2^{2^{O(n)}}$, doubly exponential in the size of the original Büchi automaton.

Safra's construction (Safra, 1988, 1989) produces a Muller or Rabin automaton from a non-deterministic Büchi automaton. For a Büchi automaton with n states, it creates a deterministic automaton with $2^{O(n\log n)}$ states, which is optimal for Rabin automata.

1.2.2 Complementation

A state-of-the-art on the complementation of Büchi automata can be found in Vardi (2007).

The complementation for deterministic ω-automata is an easy operation since it only requires to switch to the dual acceptance condition (Büchi/Co-Büchi, Rabin/Streett), but for non-deterministic ones this operation is more complicated.

Büchi (1962) introduced a complementation construction that transforms a Büchi automaton with n states into a Büchi automaton with $2^{2^{O(n)}}$ states. Sistla et al. (1985) suggested an improved version of this construction with $2^{O(n^2)}$ states. Finally, Safra (1988) proposed a construction by determinizing the automaton that produces an Büchi automaton with $2^{O(n\log n)}$ states. From the theoretical point of view, this construction matches the lower bound described by Michel (1988) and is therefore optimal.

However, constants that are hidden by the $O()$ notation can be improved since Safra's upper bound is n^{2n} while Michel's is $n!$. Kupferman and Vardi (1997) used complementation based on universal Co-Büchi automata, that are dual on the acceptance condition and the transition mode to non-deterministic Büchi automata. They decrease the upper bound up to $(6n)^n$. The current best procedure (Yan, 2008) also use these universal Co-Büchi automata, and has for upper bound $(0.97n)^n$.

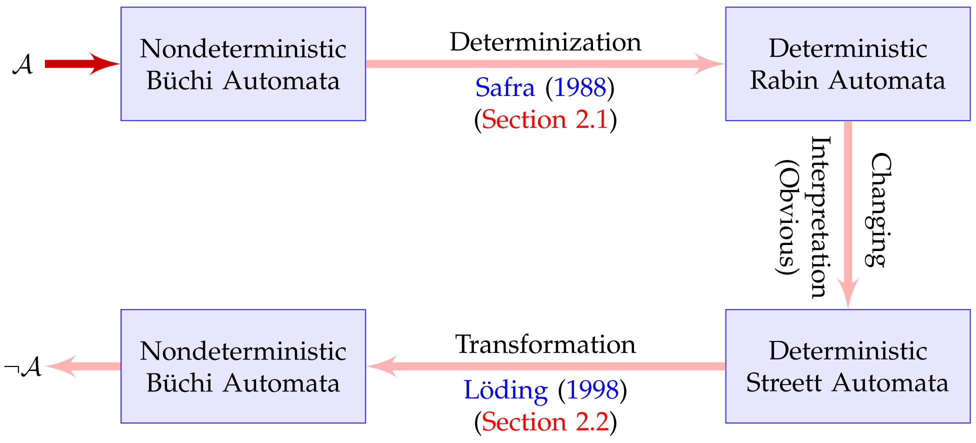
Chapter 2

Complementing Büchi automata

We will present the Safra (1988) complementation.

The first step for the complementation is to determinize the initial automaton. Since a non-deterministic Büchi automaton cannot be determinized into a Büchi automaton, we need to switch to another class.

Safra's construction (Safra, 1988) converts a non-deterministic Büchi into a deterministic Rabin automaton. In the previous chapter, we have presented that $\overline{\mathrm{DS}} \equiv \mathrm{DR}$. Therefore, interpreting the produced Rabin automaton as a Streett automaton will recognized the negation of the initial language. Then, we need to transform this Streett automaton into a non-deterministic Büchi automaton, using a transformation presented in Löding (1998). This workflow is illustrated in Figure 2.1.



$\mathcal{A}$
Nondeterministic Büchi Automata
Determinization
Safra (1988)
(Section 2.1)
Deterministic Rabin Automata
Changing Interpretation (Obvious)
Deterministic Streett Automata
Transformation
Löding (1998)
(Section 2.2)
Nondeterministic Büchi Automata
$\neg\mathcal{A}$

Figure 2.1: Steps of the complementation.

2.1 Determinization of Büchi automata

2.1.1 The classical powerset construction

For finite automaton, the classical operation for determinization is the powerset construction (Rabin and Scott, 1959). Given an automaton $\mathcal{A}$ with Q states, this construction uses subsets of Q as states of the deterministic automaton.

An example to prove that this construction does not work with Büchi automata is to consider the automaton $\mathcal{A} = (\{q_0, q_1\}, \{\mathbf{a}, \mathbf{b}\}, \delta, \bullet)$ of Figure 1.8, that recognizes a language with only finitely "a" and infinitely "b". The powerset construction is presented in Figure 2.2. We notice that the created automaton (automaton 2.2e) does not recognize the same language as the original since $\mathbf{a}$ can be reached infinitely often. This result is obvious since it was already demonstrated in Subsection 1.2.1 that this automaton cannot be determinized with Büchi acceptance conditions.

2.1.2 Safra's construction

The weakness of the powerset construction is that it constructs too many accepting runs. Safra's idea is to modify the classical powerset construction to track accepting runs of the original Büchi automaton.

Safra's construction use multiple powerset constructions in parallel in a tree structure called Safra trees. The states of the deterministic automaton are not simply sets of states but some Safra trees.

A Safra tree consists of nodes with

- A name to refer to them and keep track of their existence over multiples trees. This name must be unique in the tree and must be in the set $\{1, \dots, 2n\}$ with n the number of states of the original automaton.
- A label, which is a set of states of the original automata.
- A mark, which is a Boolean that will be introduced later.

A Safra tree is illustrated in Figure 2.3.

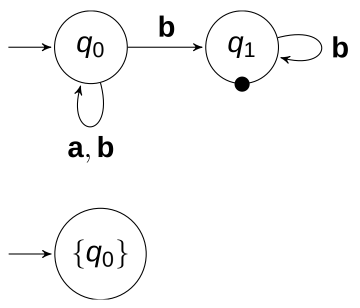
Safra trees have an ordered relation since when a node is inserted into the tree, it is considered as the youngest.

Moreover, some conditions must be satisfied.

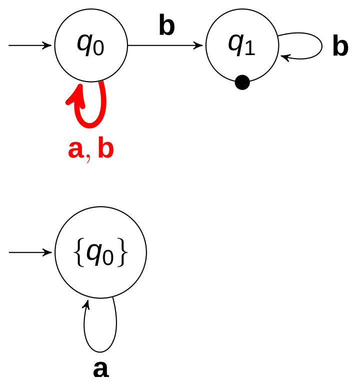
1. The label of a node is a strict superset of the labels of its children.
2. Siblings nodes have labels that are disjoint.

As consequences of these conditions, a tree has at most n nodes, with n the number of states of the automaton to determinize. We can also deduce that the height of a Safra tree is at most n (due to the first condition), and that the number of children of a node is a most n (due to the second condition).

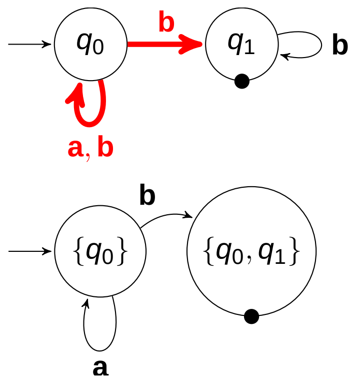
The basic idea is to add a new child for every node that contains an accepting state in its label. The label of this child will be the set of accepting labels in its parent. This idea is illustrated in Figure 2.4.



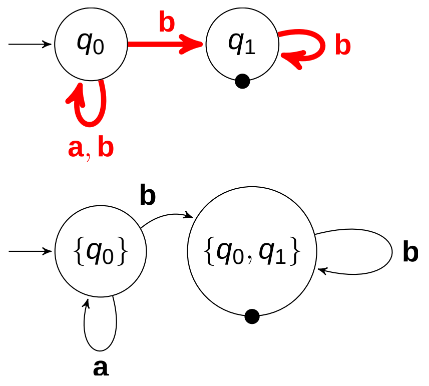
(a) The beginning of the construction.



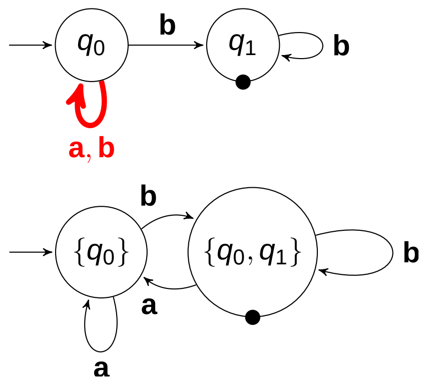
(b) A **a** in q_0 reaches the state q_0.



(c) A **b** in q_0 reaches states q_0 and q_1.



(d) A **b** in $\{q_0, q_1\}$ reaches states q_0 and q_1.



(e) A **a** in $\{q_0, q_1\}$ reaches the state q_0. This is the final automaton

Figure 2.2: The classical powerset construction: step-by-step.

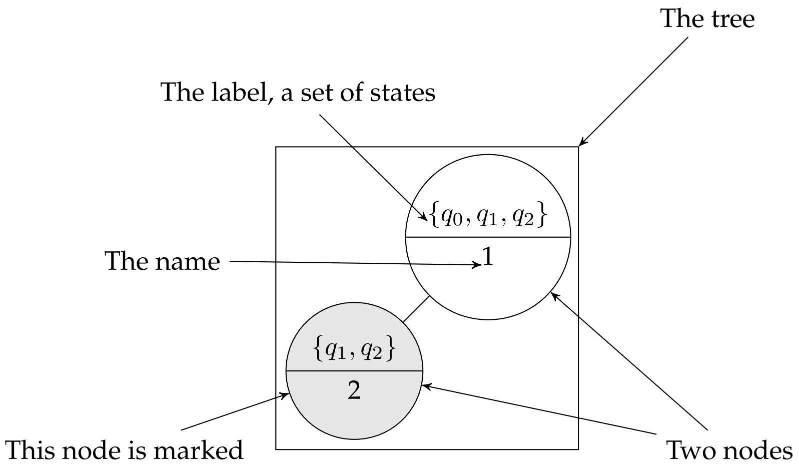


Figure 2.3: A Safra tree.

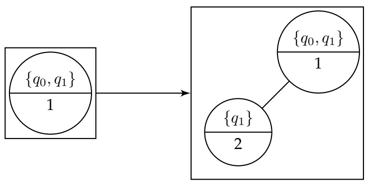


Figure 2.4: Create a new child: q_1 is an accepting state in the original automaton.

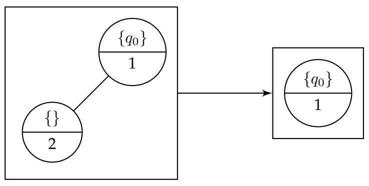


Figure 2.5: Remove empty

When a node is empty, it means that the run tracked by this node is finite, so it can be removed since missing nodes will denote finite runs. This is illustrated in Figure 2.5.

When a node has exactly the same label as its children, then they all track the same run. This means that the run tracked by this node is infinite. Children can be removed, and the parent is marked with a flag to record this deletion. This is illustrated in Figure 2.6.

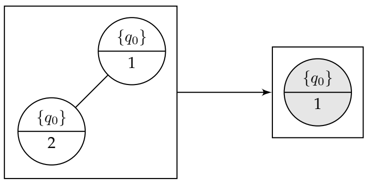


Figure 2.6: Vertical merge

Construction of the Safra trees

From the Büchi automaton $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ we create the deterministic Rabin automaton $\mathcal{R} = (Q', \Sigma, \delta', q_0', \{(L_1, U_1), \dots, (L_{2n}, U_{2n})\})$.

- The initial state q_0' is a Safra tree with only one node not marked, with q_0 as label and 1 as name.
- The transition function $\delta'(T, a)$ for a Safra tree T and $a \in \Sigma$ is computed as follows:

  (1) Remove marks All the marks in the Safra tree T are removed.

  (2) Branch accepting For every node in the Safra tree T, if a label has some accepting states then a new child of this node is inserted as youngest. This node has a unique name in the tree, no mark and the set of accepting nodes of its parent for label.

  (3) Powerset For every node n of T, its label is replaced with $\cup_{q \in \text{label of } n} \delta(q, a)$.

  (4) Horizontal merge For every two sibling nodes of T that share the same state $q \in Q$ in their label, the q in the label of the youngest node is removed, and from all its children.

  (5) Remove empty Every empty node is removed.

  (6) Vertical merge For every node whose label is equal to the union of the labels of its children, its children are removed and the node is marked.

- The set of states Q' is all the Safra trees reachable from the initial Safra tree.

The Rabin acceptance condition is the set of pairs $F = \{(L_1, U_1), \cdots, (L_{2n}, U_{2n})\}$ where

- L_i consists of all Safra trees with the node named i marked. (infinite run)
- U_i consists of all Safra trees without the node named i. (finite run)

2.1.3 Example

To show how this determinization works, we can give an example. Figure 2.1.3 is a part of the determinization of the automaton in Figure 2.7, that shows each step of the procedure. The full determinization of this automaton is in example in the appendix A.1.

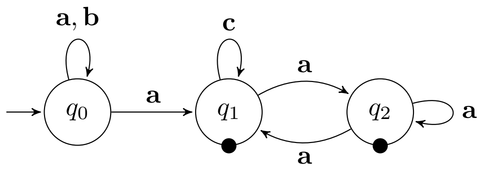


Figure 2.7: The Büchi automaton to determinize

2.2 Deterministic Streett to nondeterministic Büchi automata

Once the original Büchi automaton is determinized as a Rabin automaton, interpreting this Rabin automaton as a Streett one will recognize the negation of the original language.

Then, the next step is to transform the deterministic Streett automaton into a non-deterministic Büchi (Figure 2.8). We will present an algorithm that comes from Löding (1998).

The main idea is to create an automaton with three pools of states. The first one represents finite runs, the second pool is visited when the run does not need to visit some precise states to be accepting, and the last one is a staging area for runs that must visit some states to be accepting. The first set does not track the acceptance conditions since it represents finite runs, but the two others pools keep track of the conditions.

2.2.1 Construction of the transformation

Let $\mathcal{S} = (Q, \Sigma, \delta, q_0, \{(L_1, U_1), \dots, (L_r, U_r)\})$ be a deterministic Streett automaton to transform into an equivalent non-deterministic Büchi automaton $\mathcal{B} = (Q', \Sigma, \delta', q_0, F)$

The pairs of acceptance conditions of the Streett automaton are tracked into two sets: I and J. Every time $I \subseteq J$, for every visited L_i the corresponding U_i is also visited. Then we reset $I = J = \emptyset$, which represents our second pool of states that must be infinitely visited to be accepting.

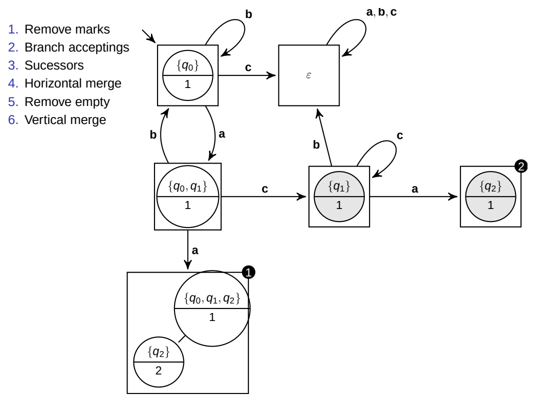
The automaton $\mathcal{B}$ is defined as follows:

- $Q' = Q \cup \left(Q \times 2^{\{1,\dots,r\}} \times 2^{\{1,\dots,r\}}\right)$
- For $a \in \Sigma, q \in Q, I \subseteq \{1, \dots, r\}, J \subseteq \{1, \dots, r\}, I' = I \cup \{i \mid q \in L_i\}$ and $J' = J \cup \{j \mid q \in U_j\}$

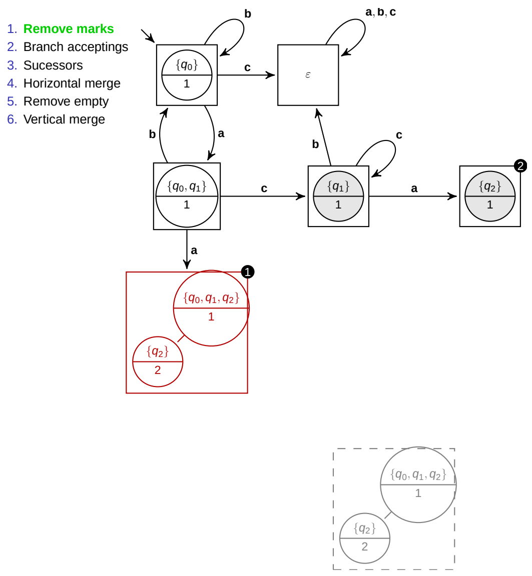
$$\begin{aligned}
\delta'(q, a) &= \delta(q, a) \cup \{(p, \emptyset, \emptyset) \mid p \in \delta(q, a)\} \\
\delta'((q, I, J), a) &= \begin{cases} \{(p, I', J') \mid p \in \delta(q, a)\} & \text{if } I' \not\subseteq J' \\ \{(p, \emptyset, \emptyset) \mid p \in \delta(q, a)\} & \text{if } I' \subseteq J' \end{cases}
\end{aligned}$$

- All states with I and J empty are accepting.

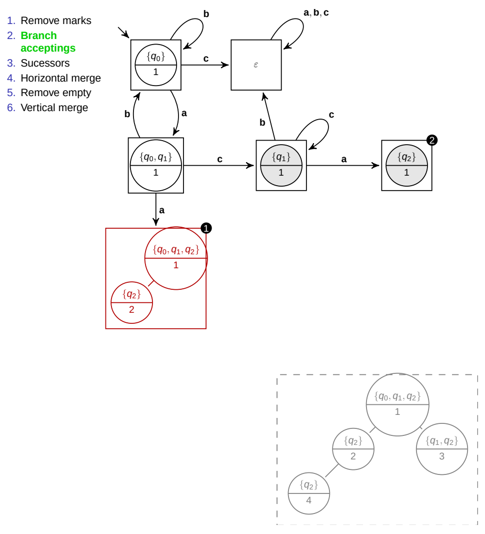
For a Street automaton with n states and r acceptance conditions, this construction creates a Büchi automaton with $n \cdot (4^n - 3^n + 2)$ states.



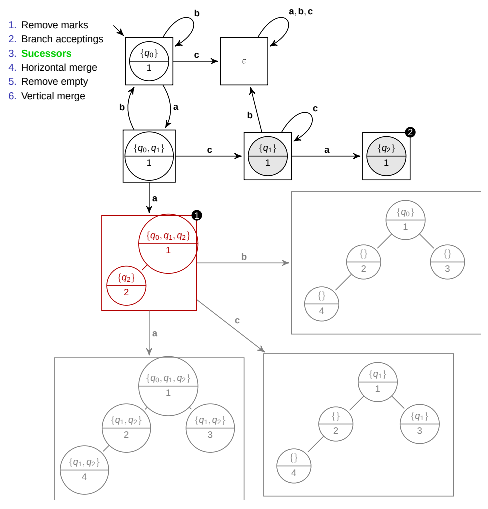
(a) The successor function is called on the tree denoted 1



(b) The original tree had no marked nodes.

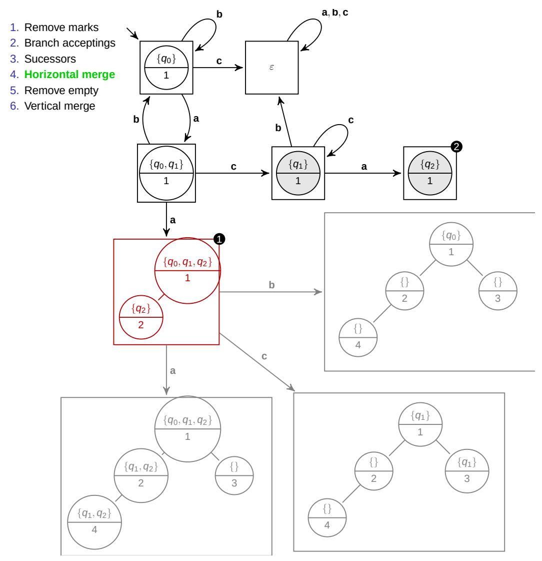


(c) q_1 and q_2 are accepting states. Therefore, each original node has a new child.

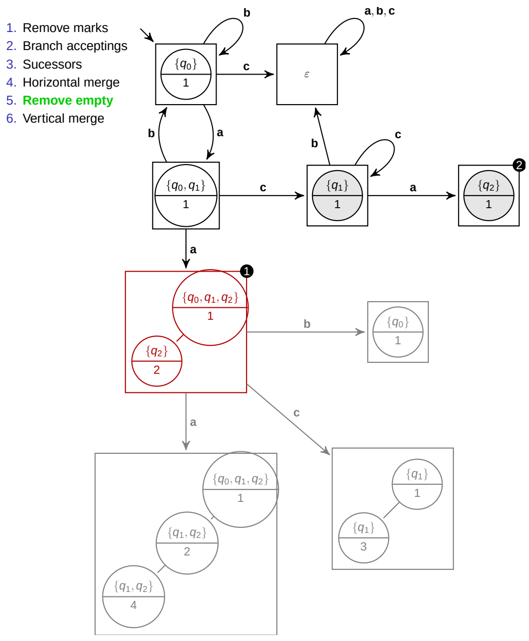


(d) We can apply the transition function on the tree created on the previous figure with the entire alphabet.

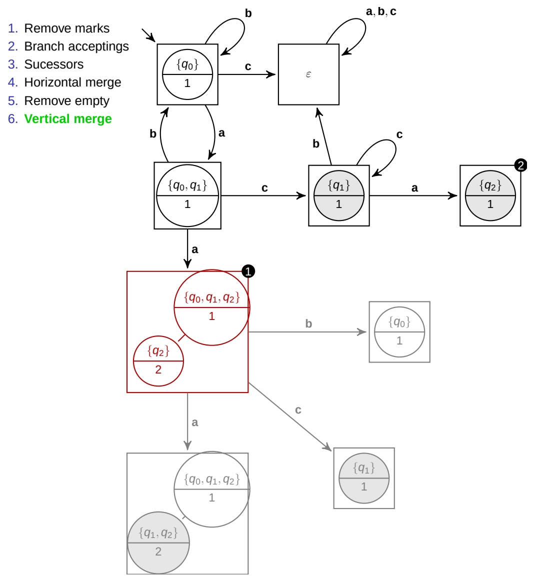
Figure 2.8: Some steps of the determinization of the automaton of Figure 2.7



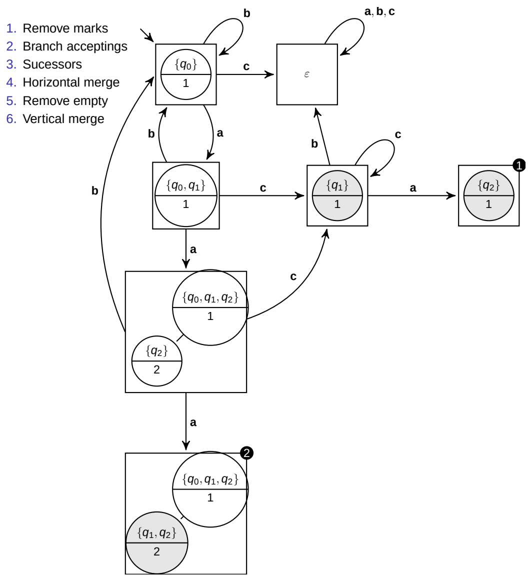
(e) Horizontal merge: two sibling nodes have q_1, q_2 for label.



(f) Empty nodes are removed.



(g) Vertical merge: the parent and children have the same set of nodes as label. Children a removed and the parent is marked.



(h) Some successors already exist.

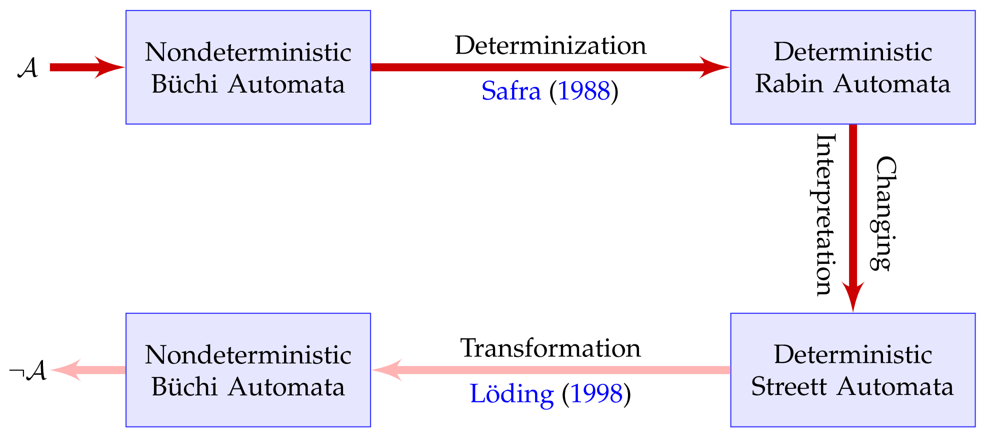

$\mathcal{A}$
Nondeterministic Büchi Automata
Determinization
Safra (1988)
Deterministic Rabin Automata
Interpretation
Changing
Deterministic Streett Automata
Transformation
Löding (1998)
Nondeterministic Büchi Automata
$\neg\mathcal{A}$

Figure 2.8: Reminder: steps of the complementation.

2.2.2 Example

To illustrate this algorithm, Figure 2.10 presents the non-deterministic Büchi automaton created from the deterministic Streett automaton of Figure 2.9. Section A.2 details more precisely the steps of this transformation.

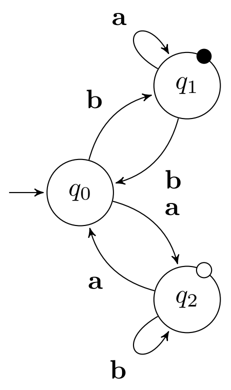
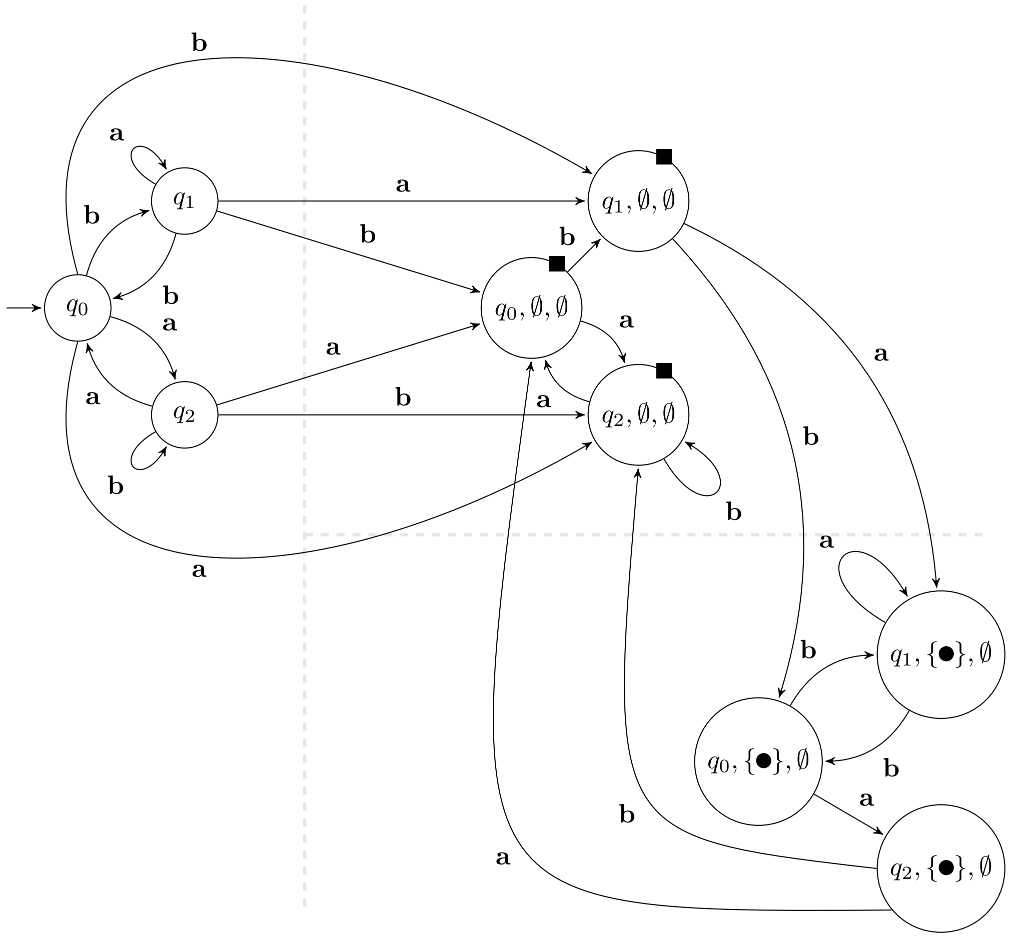


Figure 2.9: A Streett automaton $\mathcal{S}$ with one pair of acceptance condition denoted by (●, ○). If a run visits infinitely often q_1, it must also visit infinitely often q_2 to be accepting.



q_0
q_1
q_2
$q_1, \emptyset, \emptyset$
$q_0, \emptyset, \emptyset$
$q_2, \emptyset, \emptyset$
$q_1, \{\bullet\}, \emptyset$
$q_0, \{\bullet\}, \emptyset$
$q_2, \{\bullet\}, \emptyset$
a
b

Figure 2.10: The final Büchi automaton with its accepting states denoted by ■.

Chapter 3

Implementation in SPOT

Because SPOT is a library that works with Transition-based Generalized Büchi automata, and the algorithms presented in the previous section are for Büchi automaton, a first step of degeneralization is required. This operation is presented in (Duret-Lutz, 2007, sections 3.3.3 and 3.3.5) and illustrated in Figures 1.7b and 1.7c.

3.1 Details on the implementation

The complementation is implemented through a proxy class called `tgba_complement`, that inherits from the virtual class `tgba` which is common to all the implementations of Transition-based Generalized Büchi automata.

All the implementation is done in a single file: `src/tgba/tgbacomplement.cc`, and the header file (`src/tgba/tgbacomplement.hh`) exports the only public class of the implementation: `tgba_complement`.

The file `tgbacomplement.cc` contains several internal classes for the complementation. Those classes are divided in two parts: for Safra's construction and for the Streett-to-Büchi transformation.

3.1.1 Implementation of Safra's construction

Safra's construction is implemented in three classes:

`safra_tree` represents nodes of the tree. Root nodes implicitly represent whole trees. All the nodes transformations on trees to compute successors are implemented in this class. The root node is also responsible to return the Rabin's acceptance conditions that contain the tree. To represent those acceptance conditions we use two `boost::dynamic_bitset` for the sets L and U. If the tree is in one acceptance condition L_i, then the ith bit of the L-bitset will be set to `true`.

The `boost::dynamic_bitset` has the advantage to easily support bit-to-bit operations like union or intersection, which are useful during the Streett to Büchi transformation. A `std::vector<bool>` does not support these operations, and a `std::bitset` has a static size, that we do not know at compilation-time since it depends on the number of pairs of acceptance conditions of the automaton.

`safra_tree_automaton` represents an automaton with `safra_tree` as state. This structure has methods to return the initial state and to browse the automaton.

`safra_determinization` has the main loop of the algorithm, but also auxiliary methods. The main loop constructs a `safra_tree_automaton` by duplicating `safra_tree`s and calling successors operation on those trees. The auxiliary methods compute for each tree which subset of the alphabet must be used to compute successors. This subset is the conjunction of all the atomic properties that are reachable in the original automaton from states in the labels of the tree.

3.1.2 Implementation of the Streett-to-Büchi transformation

The Streett-to-Büchi transformation is directly implemented into the class `tgba_complement`. At the construction of an instance of this class, `safra_tree_automaton` is computed and the Rabin automaton is saved as attribute of the class. Then, when the initial state or the successors of a state need to be returned, the transformation is computed on-the-fly. Doing this transformation on-the-fly is a kind of optimization that avoid to create the whole automaton when it is unnecessary. During the operations like the synchronized product or the emptiness check, we do not always need to browse all the states of the automaton the find whether the system satisfies properties. A common example is during the emptiness check when algorithm find a non-empty accepting word. The algorithm stops browsing the states, so it was useless to compute whose states.

3.1.3 Testing the implementation

To test whether our implementation works, a simple way is to use algorithms that already exist in SPOT.

From a logic formula φ, we produce the automata $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$ with the help of an algorithm that traduces a formula into an automaton (Couvreur, 1999). Then we compute $\neg\mathcal{A}_\varphi$ and $\neg\mathcal{A}_{\neg\varphi}$ with our complementation. These two automata should complementary. To make sure they are, we compute the synchronized product between those two automata. The synchronized product produces an automaton that recognizes $\mathscr{L}(\neg\mathcal{A}_\varphi) \cap \mathscr{L}(\neg\mathcal{A}_{\neg\varphi})$. If this automaton does not recognize the empty language, then the complementation has issues.

This kind of tests is already done by LBTT (Tauriainen, 2000; Tauriainen and Heljanko, 2000), a test suite for logics to automata translators, that is already used in SPOT.

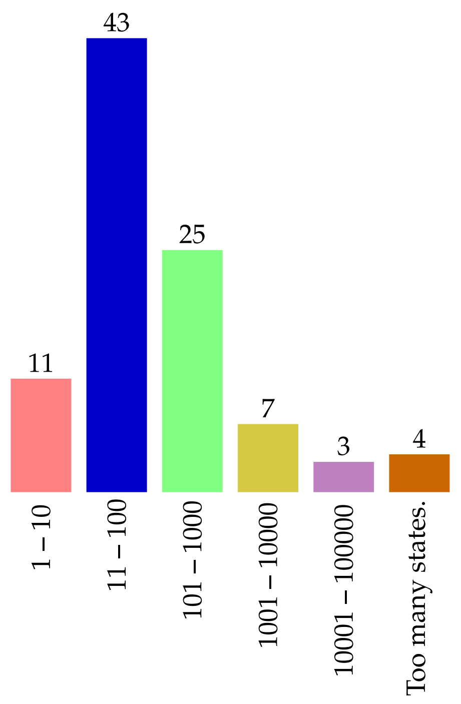
3.2 Benchmark

To benchmark our complementation, we will use our complementation algorithm on automata generated from temporal-logic formulæ. The set of logic formulæ in Table A.1 (p.45) comes from the SPOT's test suite, that gathers formulæ that comes from the literature (Dwyer et al., 1998; Etessami and Holzmann, 2000; Somenzi and Bloem, 2000). We compare states and transitions of four automata: $\mathcal{A}_\varphi$, $\mathcal{B}_{\mathcal{A}_\varphi}$, $\neg\mathcal{B}_{\mathcal{A}_\varphi}$ and $\mathcal{A}_{\neg\varphi}$.

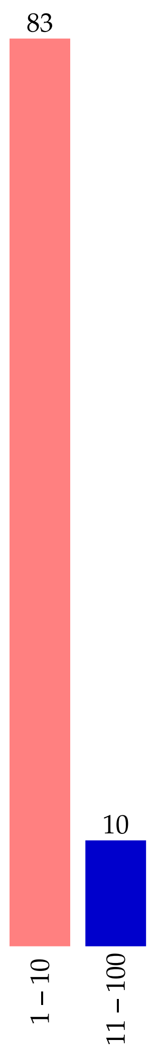
Like for testing, the two automata $\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$ are produced by the translation algorithm of Couvreur (1999) that traduces formulæ into Transition-based Generalized Büchi automata. $\mathcal{B}_{\mathcal{A}_\varphi}$ is a degeneralization of $\mathcal{A}_\varphi$ into traditional Büchi automata, and $\neg\mathcal{B}_{\mathcal{A}_\varphi}$ is the complementation of this automaton that uses our implementation. The two automata $\neg\mathcal{B}_{\mathcal{A}_\varphi}$ and $\mathcal{A}_{\neg\varphi}$ recognize the same language.

Safra's complementation is expected to produce $2^{O(n \log n)}$ states for an initial automaton $\mathcal{A}_\varphi$ with n states. This high complexity makes the complementation unusable with more than 10 input states, however, we can notice that just 11% of the formulæ from the set produce initial automata with more than 10 states, and when its occurs, complemented automata have an acceptable size. However, in some case the complementation did not finish after many weeks of computation. Figure 3.1 shows on the same set of formulæ how inefficient is our complementation comparing to the original method, but the complementation can be used on more than three quarters of the test forumulæ. We consider an automaton can be used up to 1000 states since operations such as the synchronized product and the emptiness check are applied on the automaton.

With these benchmarks we notice that our complementation is not efficient on formulæ, but may be used when it does not exists alternative techniques.



(a) Number of states of $\neg\mathcal{A}_\varphi$ on the test suite (our complementation).



(b) Number of states of $\mathcal{A}_{\neg\varphi}$ on the test suite.

Figure 3.1: Benchmarks with the same set of formulæ on $\neg\mathcal{A}_\varphi$ and $\mathcal{A}_{\neg\varphi}$

Chapter 4

Perspectives

We have presented and implemented the complementation of Büchi automata using Safra's construction. A major drawback of this implementation inside SPOT is that this algorithm works with State-based non-generalized Büchi automata whereas SPOT works with Transition-based Generalized Büchi automata.

4.1 Working with Transition-based Generalized Büchi automata

To work with Transition-based Generalized Büchi automata instead of traditional Büchi automata we need to modify Safra's construction and to have a Streett automata to Transition-based Generalized Büchi automata procedure.

But since Safra's construction is not the best complementation procedure, and some construction were published to deal with generalized acceptance constructions (**?**), we will focus on this construction.

The approach of this construction is quite different from Safra's construction, because it avoids to determinize the Büchi automaton, and prefers using another kind of intermediate automaton: universal generalized Co-Büchi automata, that are dual to non-deterministic generalized Büchi automata.

4.1.1 Streett to Generalized Büchi automata

The transformation of Streett acceptance conditions presented in Subsection 2.2.1 works for traditional Büchi automata. We can modify this algorithm to produce a Generalized Büchi automata. The idea of this modification is to stop tracking the set U of acceptance conditions in states, but to use generalized conditions on states to mark when a pair has its two sets L_i and U_i matching the Streett acceptance condition.

Let $\mathcal{S} = (Q, \Sigma, \delta, q_0, \{(L_1, U_1), \ldots, (L_r, U_r)\})$ be a Streett automaton to transform into an equivalent Generalized Büchi automata $\mathcal{B} = (Q', \Sigma, \delta', q_0, \{B_1, \ldots, B_r\})$. This construction is defined as follows:

- $Q' = Q \cup \left(Q \times 2^{\{1,\ldots,r\}}\right)$
- For $a \in \Sigma, q \in Q, I \subseteq \{1, \ldots, r\}$ and $I' = I \cup \{i \mid q \in L_i\} \setminus \{j \mid q \in U_j\})$

$$\begin{aligned}\delta'(q, a) &= \delta(q, a) \cup \{(p, \emptyset) \mid p \in \delta(q, a)\}\\ \delta'((q, I), a) &= \{(p, I') \mid p \in \delta(q, a)\}\end{aligned}$$

- The generalized Büchi acceptance conditions $\{B_1, \dots, B_r\}$ is defined by: $\forall q \in Q, \forall a \in \Sigma, \forall j \in \{1, \dots, r\} : B_j = \bigcup\limits_{x \in 2^{\{1,\cdots,r\} \setminus j}} \delta((q, x), a)$.

4.2 Safra optimizations

Our implementation of Safra's construction is the original presented in Safra (1988). However, there exists some optimization methods (Klein and Baier, 2006) to decrease the complexity (even if the complexity with the $O()$ notation stays the same). We could implement these optimizations inside SPOT.

Conclusion

This report has presented a technique to complement Büchi automata using Safra's construction for the determinization, and a Streett-to-Büchi transformation. This technique has been implemented in our model checking library SPOT, to enrich its collection of algorithms.

Due to the high complexity of the algorithms presented, in practice this complementation will have limited application. One application is to check whether alternative methods like computing the negation of a formula produce equivalent automata. Moreover, if we express properties in a language that does not support the negation, this complementation would be also useful.

All the algorithms are presented for state-based Büchi automata, since they are designed for this kind of automata. However SPOT uses Transition-based Generalized Büchi automata, then a transformation from Transition-based Generalized Büchi Automaton to Büchi automata is required in our current implementation. We want to directly work with Transition-based Generalized Büchi automaton to make benefit of their compact representation. Thus, our future work will be to implement a new complementation procedure, that supports generalized acceptance conditions.

Appendix A

Example

A.1 Safra construction

To illustrate the Safra's construction, we will determinize the Büchi automaton of Figure A.1.

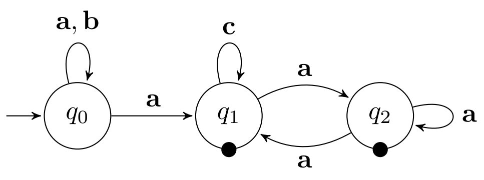
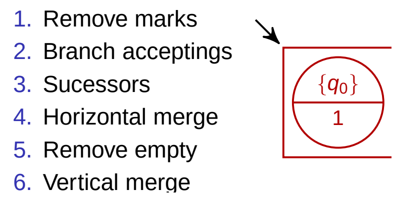
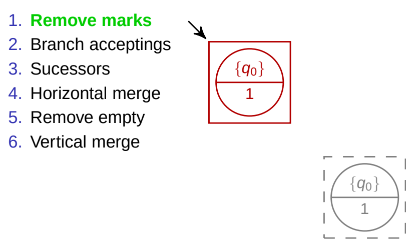


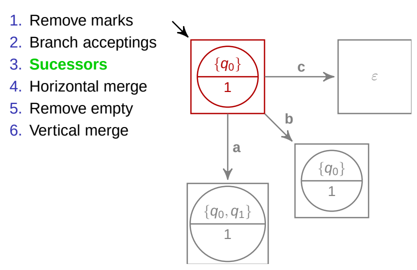
Figure A.1: The Büchi automaton to determinize



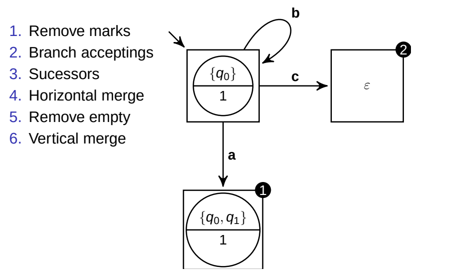
(a) The construction starts with an initial state.



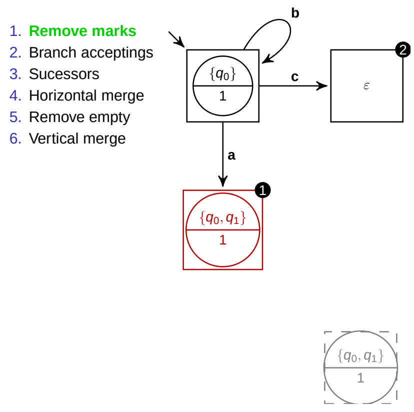
(b) The initial state was not marked.



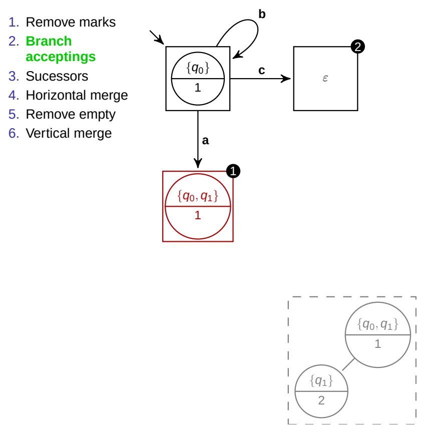
(c) Successors are computed as with the powerset construction.



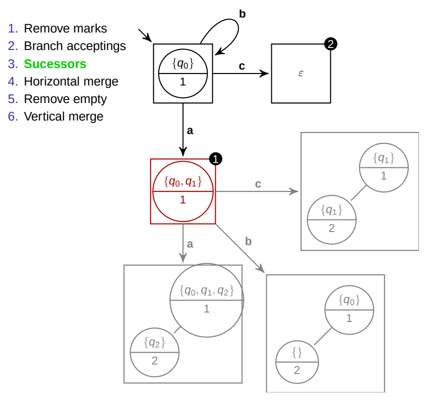
(d) The construction must continue on the successors.



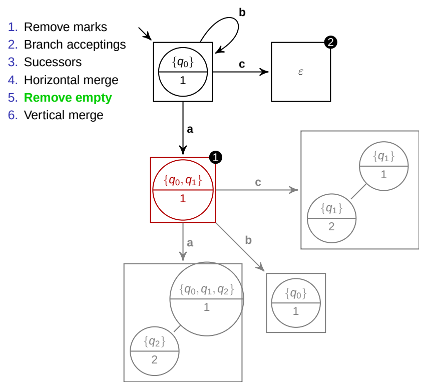
(e) The node 1 was not marked.



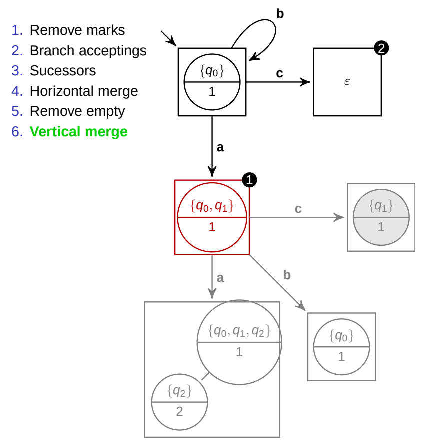
(f) q_1 is accepting, therefore a new child with q_1 as label is inserted.



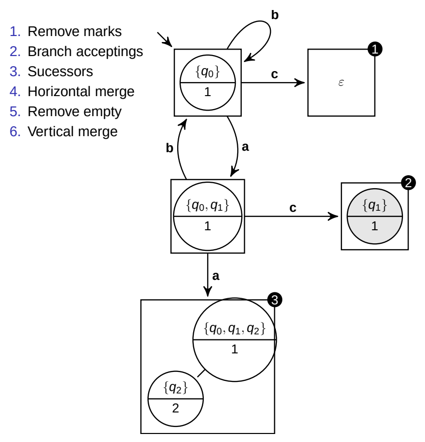
(g) The powerset construction with the whole alphabet is applied on the tree previously created.



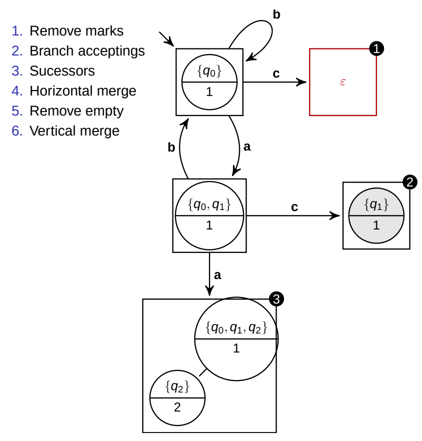
(h) Empty nodes are removed.



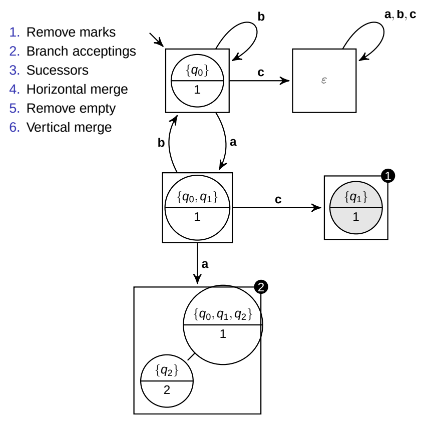
(i) A tree has q_1 in its root and its child. The root is marked and the child removed.



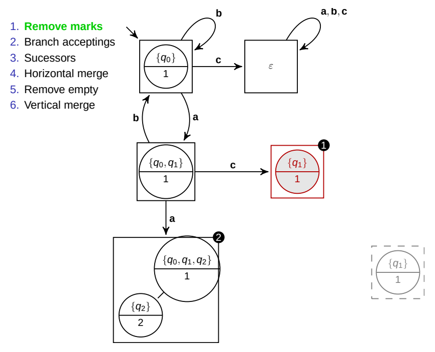
(j) The construction continue with a BFS.



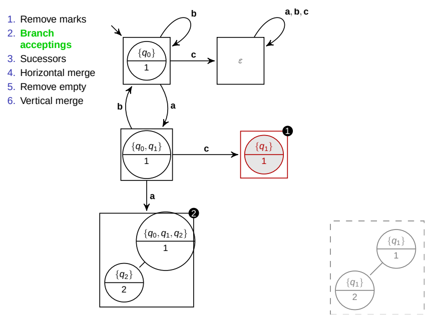
(k) The node ε is computed. . .



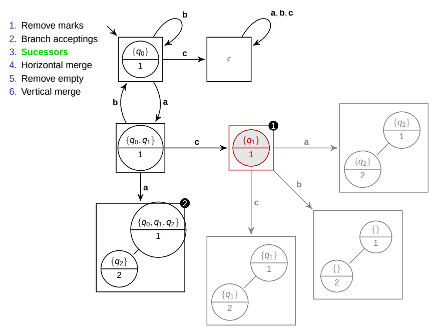
(l) . . . and has a true self loop.



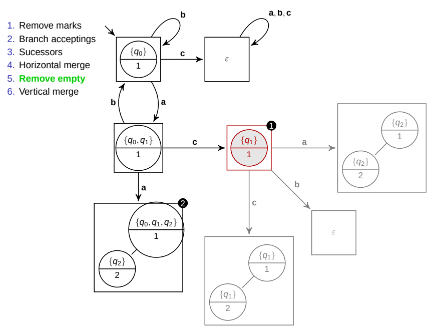
(m) The node 1 is unmarked.



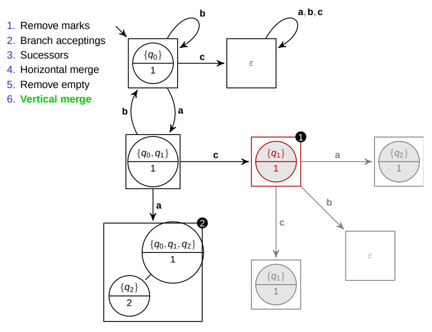
(n) q_1 is an accepting state, therefore a child is inserted.



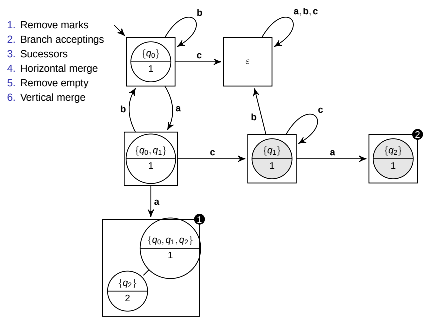
(o) Successors are computed with the powerset construction on label's nodes.



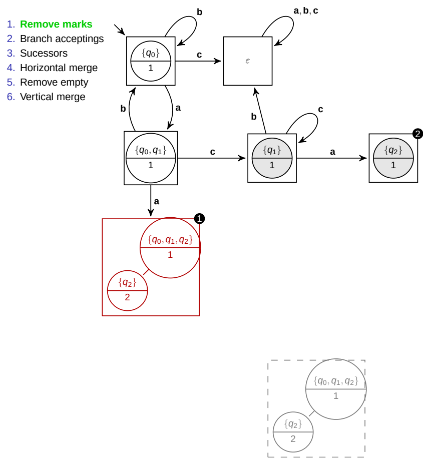
(p) Empty node are removed.



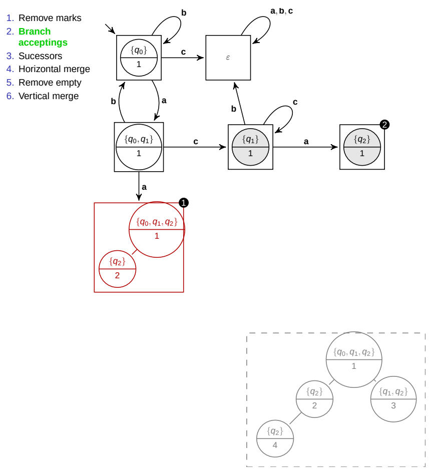
(q) Vertical nodes are merged.



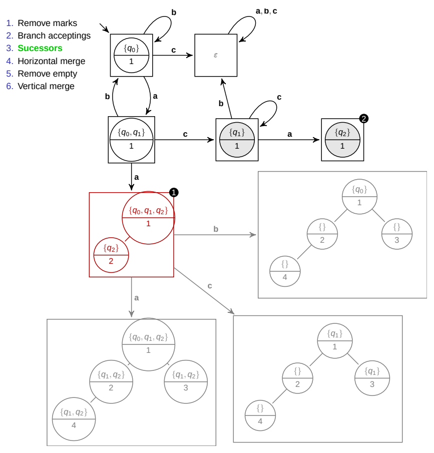
(r) The BFS continue.



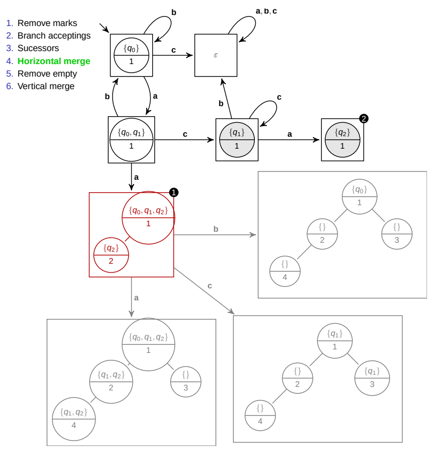
(s) The tree 1 has any marked node.



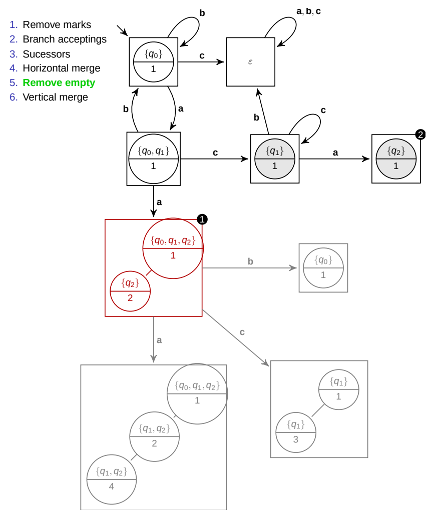
(t) Each node has a new child.



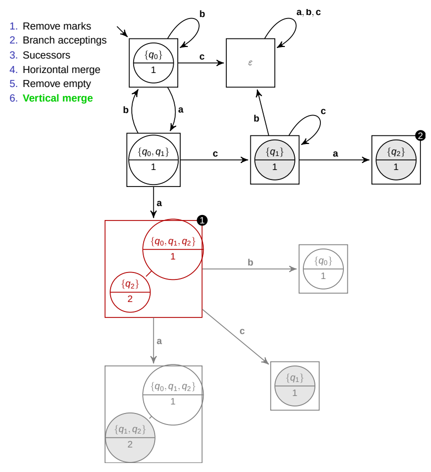
(u) The powerset construction is applied.



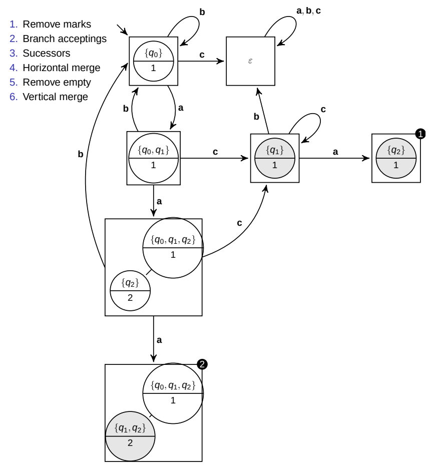
(v) Two sibling nodes shared the same label. States of the youngest node are removed.



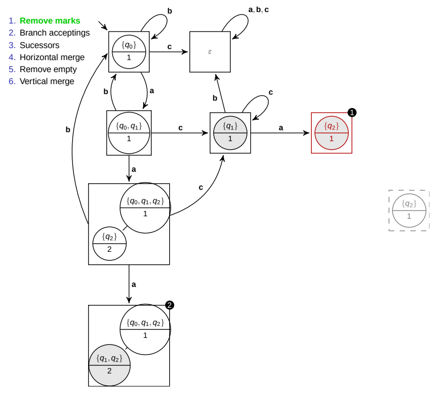
(a) Empty nodes are removed.



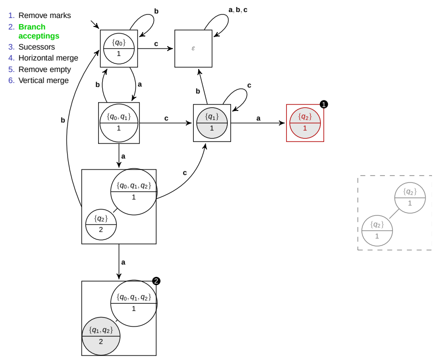
(b) Vertical merge is done.



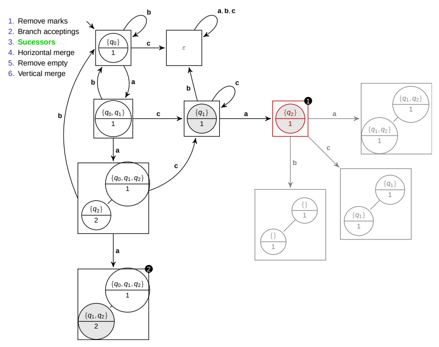
(c) The BFS continue.



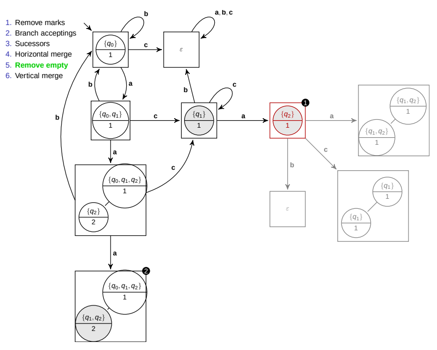
(d) The node is unmarked.



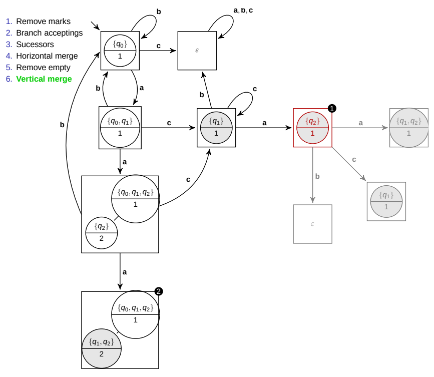
(e) q_2 is an accepting state.



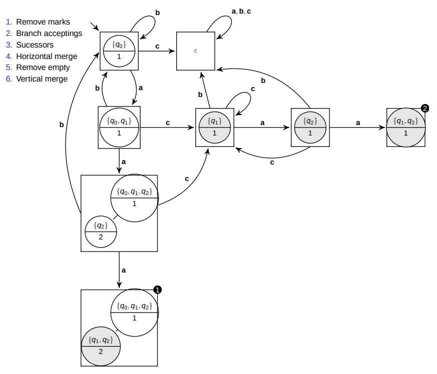
(f) The powerset construction.



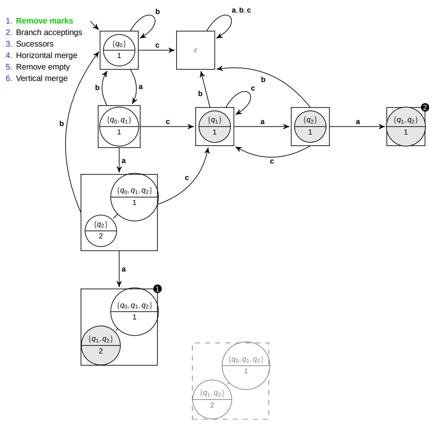
(g) Empty nodes are removed.



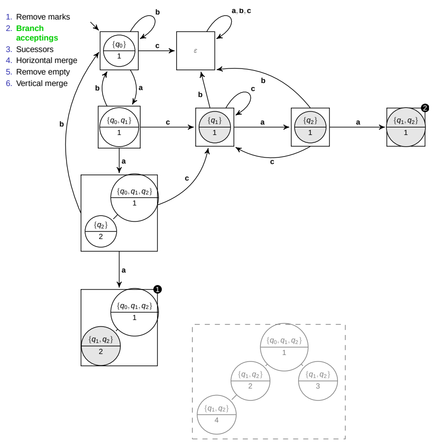
(h) Vertical merge is done.



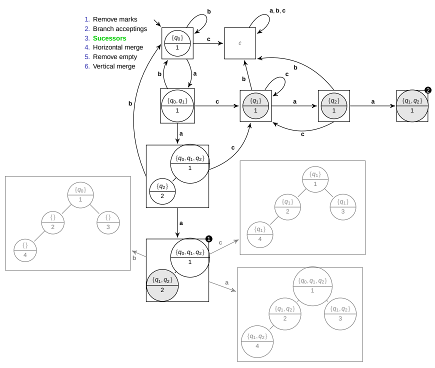
(i) The BFS continue.



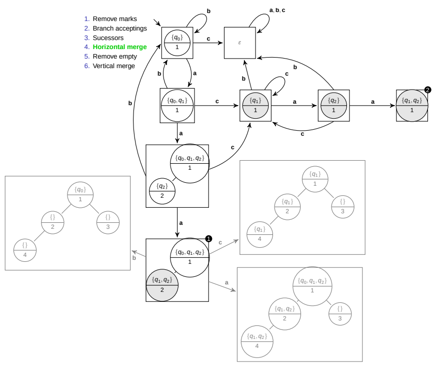
(j) Nodes are unmarked.



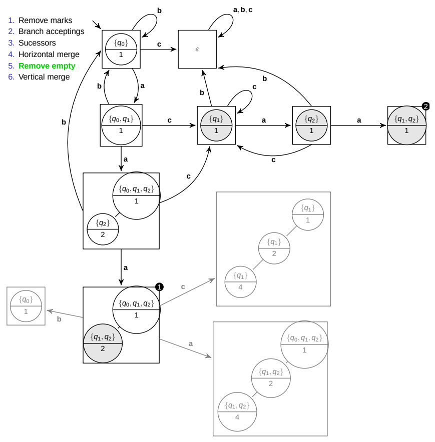
(k) Each node has a new child.



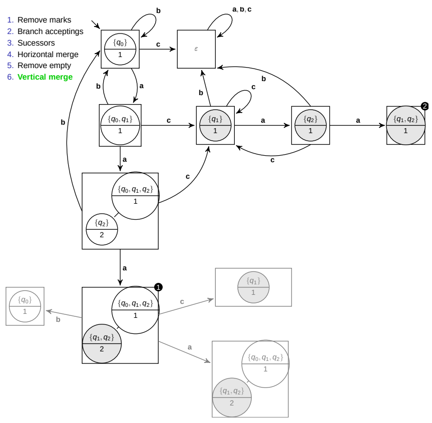
(l) Successors are computed.



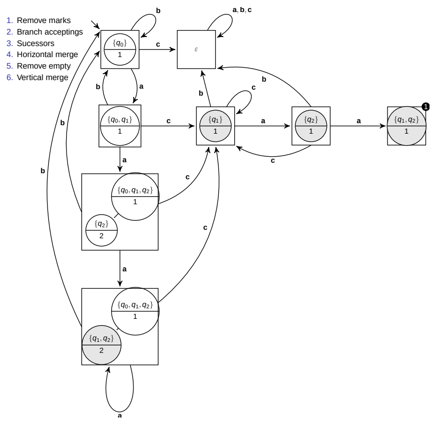
(m) Horizontal merge.



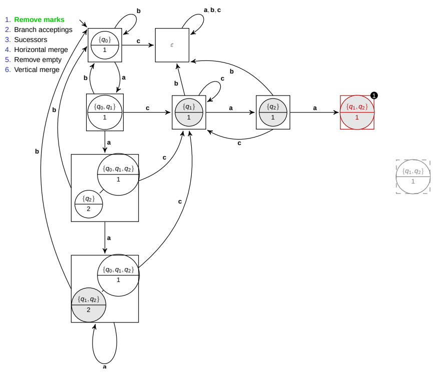
(n) Empty nodes are removed.



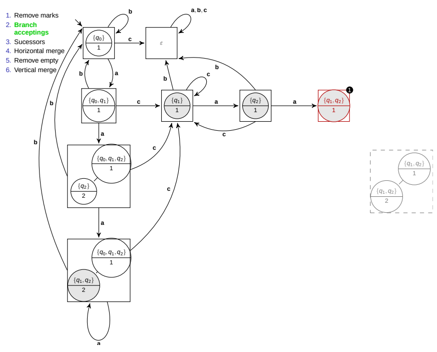
(o) Vertical merge.



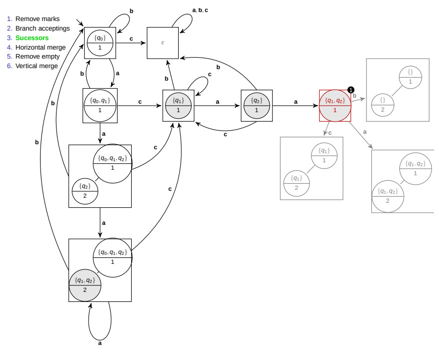
(p) The BFS continue with only one tree in the queue..

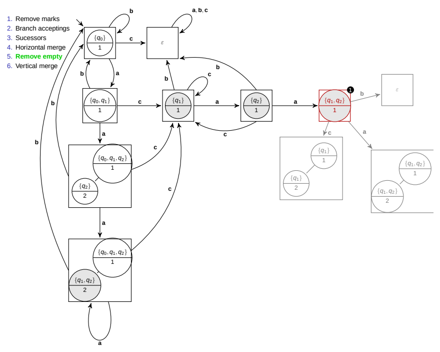


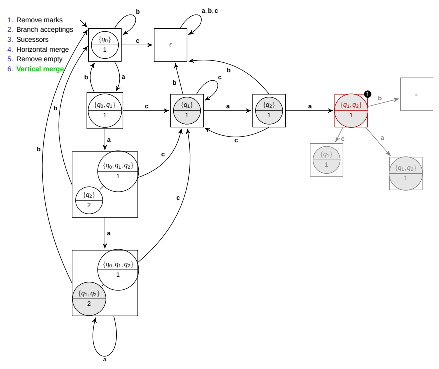
(q) The node is unmarked.

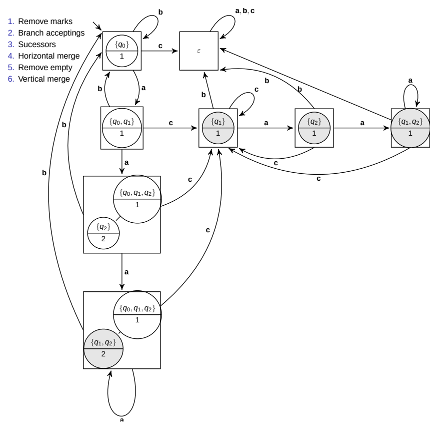


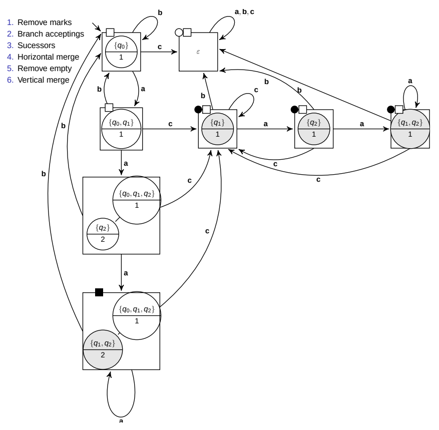
(r) A new child is inserted.


(s) Successors are computed.


(t) Empty nodes are removed.

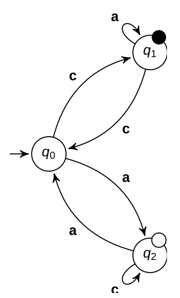

(u) Vertical merge.


(v) The automaton has all its states and transitions.

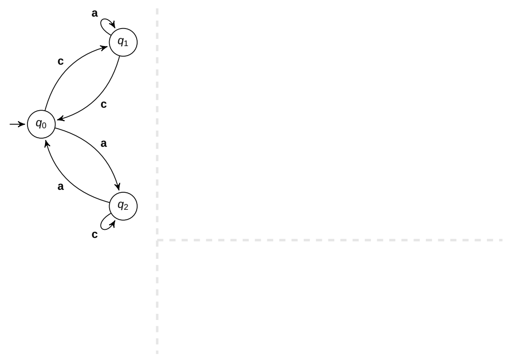

(w) Rabin's acceptance conditions are added.

A.2 Transformation of a deterministic Streett automaton into a non-deterministic Büchi

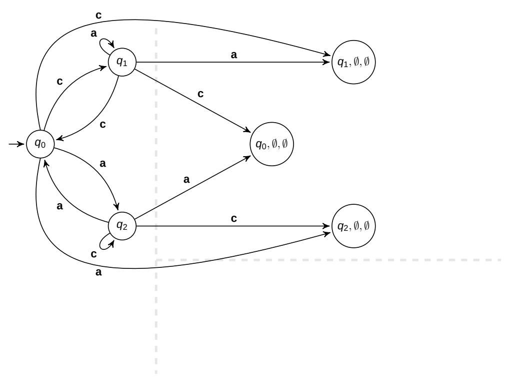
Figure A.-4 gives more details on the example of transformation of a deterministic Streett automaton into a non-deterministic Büchi presented at Subsection 2.2.2.



(a) The Streett automaton

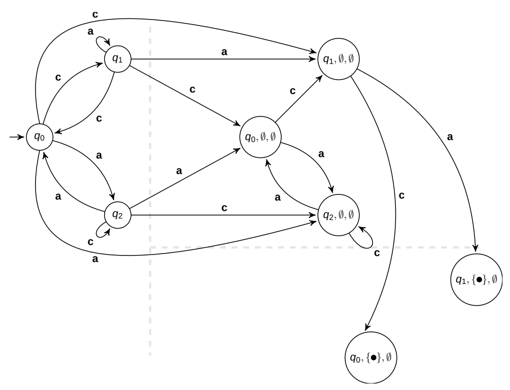


(b) Acceptance conditions are removed.



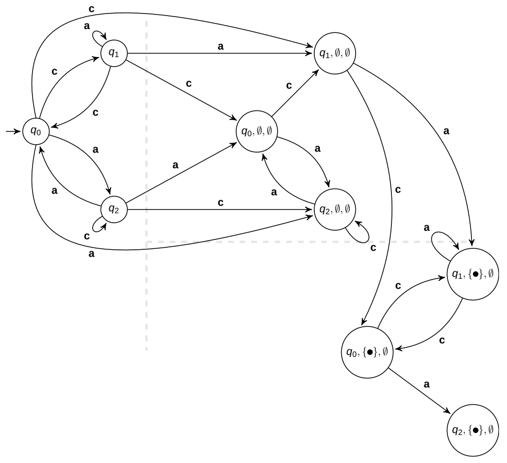
q_0
q_1
q_2
$q_1, \emptyset, \emptyset$
$q_0, \emptyset, \emptyset$
$q_2, \emptyset, \emptyset$

(c) Each state has its successors duplicated with sets I and J empty

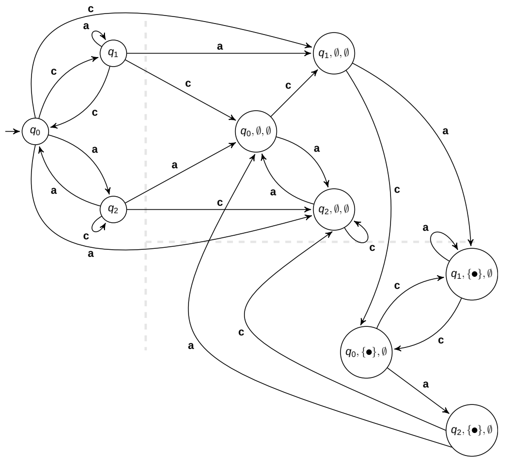


q_0
q_1
q_2
$q_1, \emptyset, \emptyset$
$q_0, \emptyset, \emptyset$
$q_2, \emptyset, \emptyset$
$q_1, \{\bullet\}, \emptyset$
$q_0, \{\bullet\}, \emptyset$

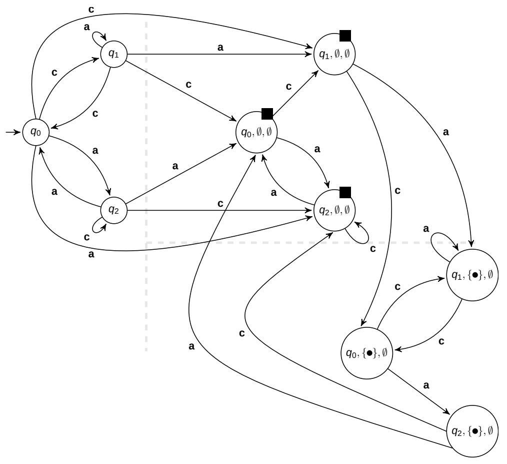
(d) The transition function is applied on the states with I and J empty. Because $q_1 \in L_1$, successors of $(q_1, \emptyset, \emptyset)$ have the acceptance condition in their set I


q_0
q_1
q_2
$q_0, \emptyset, \emptyset$
$q_1, \emptyset, \emptyset$
$q_2, \emptyset, \emptyset$
$q_1, \{\bullet\}, \emptyset$
$q_0, \{\bullet\}, \emptyset$
$q_2, \{\bullet\}, \emptyset$
a
c

(e) The transition function is applied on the states with I that contains the acceptance condition.


q_0
q_1
q_2
$q_0, \emptyset, \emptyset$
$q_1, \emptyset, \emptyset$
$q_2, \emptyset, \emptyset$
$q_1, \{\bullet\}, \emptyset$
$q_0, \{\bullet\}, \emptyset$
$q_2, \{\bullet\}, \emptyset$
a
c

(f) Because $q_2 \in U_1$ and $q_1 \in L_i$ has been visited during the run, successors reach to the accepting pool with I and J empty.


q_0
q_1
q_2
$q_0, \emptyset, \emptyset$
$q_1, \emptyset, \emptyset$
$q_2, \emptyset, \emptyset$
$q_1, \{\bullet\}, \emptyset$
$q_0, \{\bullet\}, \emptyset$
$q_2, \{\bullet\}, \emptyset$
a
c

(g) All states with I and J empty are in the Büchi acceptance condition set.

Figure A.-4: Steps of the transformation of a deterministic Streett automaton into a non-deterministic Büchi automaton

A.3 Benchmarks

Table A.1 presents the results of our benchmarks. Those results are discussed in Section 3.2.

Formulæ	**Original $\mathcal{A}_\varphi$**		**Büchi $\mathcal{B}_{A_\varphi}$**		**Complementation $\neg\mathcal{B}_{A_\varphi}$**		**Original $\mathcal{A}_{\neg\varphi}$**	
	St	Tr	St	Tr	St	Tr	St	Tr
$((\top \mathrm{U}(\top \mathrm{U} p_0)) \wedge (\bot V \neg p_0)) \wedge ((\top \mathrm{U} p_0) \wedge (\bot V(\bot V \neg p_0)))$	1	1	1	1	4	9	5	11
$\mathrm{G}\neg p_0$	1	1	1	1	7	16	2	3
$\mathrm{G}p_0$	1	1	1	1	7	16	2	3
$\bot V((\top \mathrm{U} p_0) \wedge (\top \mathrm{U} p_1))$	1	4	3	12	26	140	3	5
$((((\bot V(\top \mathrm{U} p_0)) \wedge (\bot V(\top \mathrm{U} p_1))) \wedge (\bot V(\top \mathrm{U} p_2))) \wedge (\bot V(\top \mathrm{U} p_3))) \wedge (\bot V(\top \mathrm{U} p_4))$	1	32	6	192	50	2176	11	25
$\bot V(p_1 \vee (\mathrm{X}p_0 \wedge \mathrm{X}\neg p_0))$	2	2	2	2	10	20	4	6
$((p_0 \mathrm{U}(p_1 \mathrm{U} p_2)) \vee (p_1 \mathrm{U}(p_2 \mathrm{U} p_0))) \vee (p_2 \mathrm{U}(p_0 \mathrm{U} p_1))$	2	2	2	2	6	23	2	2
$p_0 \mathrm{U} p_1$	2	3	2	3	9	22	2	3
$p_0 V(p_0 \vee p_1)$	2	3	2	3	10	32	2	3
$(p_0 \mathrm{U} p_0) \vee (p_1 \mathrm{U} p_0)$	2	3	2	3	9	22	2	3
$p_0 \mathrm{U}(p_1 \wedge (\bot V p_2))$	2	3	2	3	27	150	4	11
$\top \mathrm{U}(p_0 \wedge \mathrm{X}(\bot V p_1))$	2	3	2	3	11	52	2	5
$\mathrm{G}(p_2 \Rightarrow \mathrm{G}\neg p_0)$	2	3	2	3	13	52	3	6
$\mathrm{F}p_0$	2	3	2	3	7	13	1	1
$\mathrm{G}(p_2 \Rightarrow \mathrm{G}p_0)$	2	3	2	3	13	52	3	6
$\bot V(\neg p_0 \vee (\top \mathrm{U} p_1))$	2	4	3	6	13	56	2	3
$\bot V(\neg p_0 \vee (p_1 \mathrm{U} p_2))$	2	4	3	6	22	180	3	6
$\mathrm{G}((p_2 \wedge \neg p_1) \Rightarrow (\neg p_1 \mathrm{U}(p_0 \wedge \neg p_1)))$	2	4	3	6	16	116	3	5
$\mathrm{G}(p_0 \Rightarrow \mathrm{F}p_3)$	2	4	3	6	13	56	2	3
$(\bot V(\top \mathrm{U} p_0)) \wedge (\top \mathrm{U}(\bot V \neg p_1))$	2	6	4	12	21	116	4	9
$p_0 \mathrm{U}(p_1 \wedge \mathrm{X}(p_2 \mathrm{U} p_3))$	3	5	4	7	80	694	4	11
$\top \mathrm{U}(p_0 \wedge \mathrm{X}(p_1 \mathrm{U} p_2))$	3	5	4	7	28	238	2	4
$\neg p_1 \mathrm{U}((p_0 \wedge \neg p_1) \vee \mathrm{G}\neg p_1)$	3	5	3	5	28	87	2	3
$\neg p_0 \mathrm{U}(p_3 \vee \mathrm{G}\neg p_0)$	3	5	3	5	29	88	2	3
$p_0 \mathrm{U}(p_1 \mathrm{U} p_2)$	3	6	5	11	35	227	3	6
$\neg p_0 V(\neg p_1 V \neg p_2)$	3	6	3	6	19	152	3	6
$\mathrm{G}(((p_2 \wedge \neg p_1) \wedge \mathrm{F}p_1) \Rightarrow (\neg p_0 \mathrm{U} p_1))$	3	6	4	9	45	289	4	8
$\mathrm{G}(((p_2 \wedge \neg p_1) \wedge \mathrm{F}p_1) \Rightarrow (p_0 \mathrm{U} p_1))$	3	6	4	9	65	449	4	8
$\mathrm{G}(((p_2 \wedge \neg p_1) \wedge \mathrm{F}p_1) \Rightarrow (\neg p_0 \mathrm{U}(p_3 \vee p_1)))$	3	6	4	9	70	1047	4	8
$\mathrm{G}((p_2 \wedge \neg p_1) \Rightarrow (\neg p_0 \mathrm{U}(p_1 \vee \mathrm{G}\neg p_0)))$	3	7	4	10	93	566	3	6
$\mathrm{G}((p_2 \wedge \neg p_1) \Rightarrow (\neg p_1 \mathrm{U}((p_0 \wedge \neg p_1) \vee \mathrm{G}\neg p_1)))$	3	7	4	10	112	727	3	5
$\mathrm{G}((p_2 \wedge \neg p_1) \Rightarrow (p_0 \mathrm{U}(p_1 \vee \mathrm{G}p_0)))$	3	7	4	10	134	815	3	6
$\mathrm{G}((p_2 \wedge \neg p_1) \Rightarrow (\neg p_0 \mathrm{U}((p_3 \vee p_1) \vee \mathrm{G}\neg p_0)))$	3	7	4	10	134	1487	3	6
$\mathrm{G}(p_2 \Rightarrow \mathrm{G}(p_0 \Rightarrow \mathrm{F}p_3))$	3	7	4	10	35	264	3	6
$\mathrm{G}((((p_2 \wedge \neg p_1) \wedge \mathrm{F}p_1) \Rightarrow (p_0 \Rightarrow (\neg p_1 \mathrm{U}(p_3 \wedge \neg p_1)))) \mathrm{U} p_1)$	3	7	6	17	32	344	4	8
$\mathrm{G}(p_2 \Rightarrow \mathrm{G}(p_0 \Rightarrow (p_3 \wedge \mathrm{XF}p_4)))$	3	8	5	14	154	2006	4	9
$\mathrm{G}(p_2 \Rightarrow \mathrm{G}(p_0 \Rightarrow ((p_3 \wedge \neg p_5) \wedge \mathrm{X}(\neg p_5 \mathrm{U} p_4))))$	3	8	5	14	180	4406	4	10
$(\bot V p_0) \mathrm{U} p_1$	4	6	4	6	18	71	3	6
$\top \mathrm{U}(p_0 \wedge \mathrm{X}(p_1 \wedge \mathrm{X}(\top \mathrm{U} p_2)))$	4	6	5	8	1866	11358	3	6
$(\top \mathrm{U} p_0) \wedge (\top \mathrm{U} \neg p_0)$	4	7	4	7	10	23	3	4
$\mathrm{F}p_1 \Rightarrow (\neg p_0 \mathrm{U} p_1)$	4	7	4	7	25	79	3	5
$\mathrm{F}p_1 \Rightarrow (p_0 \mathrm{U} p_1)$	4	7	4	7	25	79	3	5
$\mathrm{F}p_1 \Rightarrow (\neg p_0 \mathrm{U}(p_3 \vee p_1))$	4	7	4	7	37	155	3	5
$(\mathrm{X}p_0 \mathrm{U} p_1) \vee \mathrm{X}(\neg p_0 V(\neg p_0 \vee \neg p_1))$	4	8	4	8	16	54	3	4
$(\bot V(p_1 \vee \mathrm{X}(\bot V p_0))) \wedge (\bot V(p_2 \vee \mathrm{X}(\bot V \neg p_0)))$	4	8	4	8	16	80	6	13
$(\mathrm{F}p_1 \Rightarrow (p_0 \Rightarrow (\neg p_1 \mathrm{U}(p_3 \wedge \neg p_1)))) \mathrm{U} p_1$	4	8	6	14	14	78	3	5
$(\top \mathrm{U}(\bot V \neg p_0)) \vee (\bot V(\top \mathrm{U} p_1))$	4	9	5	11	405	1864	2	6
$\bot V(\neg p_0 \vee (p_1 \mathrm{U}((\bot V p_2) \vee (\bot V p_3))))$	4	9	5	13	882	6099	6	26
$(\top \mathrm{U} p_0) \mathrm{U}(\bot V p_1)$	4	10	7	19	89	496	3	7
$\mathrm{G}(p_0 \Rightarrow \mathrm{F}(p_3 \wedge \mathrm{XF}p_4))$	4	14	8	28	610	5140	3	6

$G(p_0 \Rightarrow F((p_3 \land \neg p_5) \land X(\neg p_5 U p_4)))$	4	14	8	28	594	9204	3	7
$((\bot V(\top U p_0)) \land (\top U(\bot V \neg p_1))) \land ((\bot V(\top U p_1)) \land (\top U(\bot V \neg p_0)))$	4	16	6	20	10	60	7	18
$(X p_0 U X p_1) \lor X(\neg p_0 V \neg p_1)$	5	9	5	9	26	110	3	4
$\top U(p_0 \land X(\top U(p_1 \land X(\top U(p_2 \land X(\top U p_3))))))$	5	9	8	15	124	2170	4	7
$F p_0 \Rightarrow (\neg p_0 U((p_3 \land \neg p_0) \land X(\neg p_0 U p_4)))$	5	9	6	11	148	1091	3	6
$(\top U(\bot V p_0)) \lor (\top U(\bot V p_1))$	5	10	5	10	235	1096	1	4
$G \neg p_2 \lor (\neg p_2 U((p_2 \land F p_0) \Rightarrow (\neg p_0 U((p_3 \land \neg p_0) \land X(\neg p_0 U p_4)))))$	5	10	6	12	167	1121	4	9
$F(p_3 \land XF p_4) \Rightarrow (\neg p_3 U p_0)$	5	10	5	10	46	198	4	8
$G \neg p_2 \lor F(p_2 \land F p_0)$	5	11	6	13	51	238	2	3
$F p_1 \Rightarrow (\neg p_0 U(p_1 \lor ((p_3 \land \neg p_0) \land X(\neg p_0 U p_4))))$	5	11	6	13	242	2477	4	8
$G((p_2 \land \neg p_1) \Rightarrow ((p_0 \Rightarrow (\neg p_1 U(p_3 \land \neg p_1)))U(p_1 \lor G(p_0 \Rightarrow (\neg p_1 U(p_3 \land \neg p_1))))))$	5	18	8	29	9398	102836	4	8
$F p_1 \Rightarrow (\neg((p_3 \land \neg p_1) \land X(\neg p_1 U(p_4 \land \neg p_1)))U(p_1 \lor p_0))$	6	14	6	14	92	629	4	7
$(\bot V(\neg p_0 \lor (\top U p_1))) \land ((X p_0 U p_1) \lor X(\neg p_0 V(\neg p_0 \lor \neg p_1)))$	6	15	6	15	112	562	5	9
$G \neg p_2 \lor F(p_2 \land (\neg p_0 U(p_3 \lor G \neg p_0)))$	6	15	7	18	656	4437	3	7
$G((p_2 \land F p_1) \Rightarrow (\neg((p_3 \land \neg p_1) \land X(\neg p_1 U(p_4 \land \neg p_1)))U(p_1 \lor p_0)))$	6	16	7	20	3742	102403	5	10
$\neg p_0 U((p_0 U((\neg p_0 U((p_0 U(G \neg p_0 \lor G p_0)) \lor G \neg p_0)) \lor G \neg p_0)) \lor G \neg p_0)$	6	20	10	38	123	303	6	11
$G((p_2 \land F p_1) \Rightarrow (\neg p_0 U(p_1 \lor ((p_3 \land \neg p_0) \land X(\neg p_0 U p_4)))))$	6	22	10	41	10723	312622	5	12
$G(p_2 \Rightarrow (F p_0 \Rightarrow (\neg p_0 U(p_1 \lor ((p_3 \land \neg p_0) \land X(\neg p_0 U p_4))))))$	6	22	10	41	18664	529780	4	10
$G(((p_2 \land F p_1) \Rightarrow (p_0 \Rightarrow (\neg p_1 U((p_3 \land \neg p_1) \land X(\neg p_1 U p_4)))))U p_1)$	6	22	9	34	96	1696	5	13
$G(((p_2 \land F p_1) \Rightarrow (p_0 \Rightarrow (\neg p_1 U(((p_3 \land \neg p_1) \land \neg p_5) \land X((\neg p_1 \land \neg p_5)U p_4)))))U p_1)$	6	22	9	34	96	3392	5	14
$G((p_3 \land XF p_4) \Rightarrow XF(p_4 \land F p_0))$	6	27	11	53	424	3924	3	5
$p_0 U(p_1 \land X(p_2 \land (\top U(p_3 \land X(\top U(p_4 \land X(\top U(p_5 \land X(\top U p_6)))))))))$	7	13	10	19	33302	571049	12	37
$G \neg p_2 \lor (\neg p_2 U(p_2 \land (F(p_3 \land XF p_4) \Rightarrow (\neg p_3 U p_0))))$	7	18	8	20	69	387	5	12
$(\bot V(\neg p_0 \lor (\top U p_1))) \land ((X p_0 U X p_1) \lor X(\neg p_0 V \neg p_1))$	7	19	8	20	105	520	5	9
$(F p_1 \Rightarrow (p_0 \Rightarrow (\neg p_1 U((p_3 \land \neg p_1) \land X(\neg p_1 U p_4)))))U p_1$	7	23	9	30	34	482	4	9
$(F p_1 \Rightarrow (p_0 \Rightarrow (\neg p_1 U(((p_3 \land \neg p_1) \land \neg p_5) \land X((\neg p_1 \land \neg p_5)U p_4)))))U p_1$	7	23	9	30	34	954	4	10
$(((\bot V(p_1 \lor (\bot V(\top U p_0)))) \land (\bot V(p_2 \lor (\bot V(\top U \neg p_0))))) \lor (\bot V p_1)) \lor (\bot V p_2)$	7	28	11	40	298	1228	17	76
$G((p_2 \land F p_1) \Rightarrow ((\neg p_0 \land \neg p_1)U(p_1 \lor ((p_0 \land \neg p_1)U(p_1 \lor ((\neg p_0 \land \neg p_1)U(p_1 \lor ((p_0 \land \neg p_1)U(p_1 \lor (\neg p_0 U p_1))))))))))$	7	28	12	49	5765	37471	8	16
$G(p_2 \Rightarrow G((p_3 \land XF p_4) \Rightarrow X(\neg p_4 U(p_4 \land F p_0))))$	7	31	12	58	1025	12448	4	8
$F p_1 \Rightarrow ((\neg p_0 \land \neg p_1)U(p_1 \lor ((p_0 \land \neg p_1)U(p_1 \lor ((\neg p_0 \land \neg p_1)U(p_1 \lor ((p_0 \land \neg p_1)U(p_1 \lor (\neg p_0 U p_1)))))))))$	8	29	12	43	350	1375	7	13
$G(p_2 \Rightarrow ((\neg p_0 \land \neg p_1)U(p_1 \lor ((p_0 \land \neg p_1)U(p_1 \lor ((\neg p_0 \land \neg p_1)U(p_1 \lor ((p_0 \land \neg p_1)U((p_1 \lor (\neg p_0 U(p_1 \lor G \neg p_0))) \lor G p_0)))))))))$	8	39	13	68	8314	53840	7	14
$G(((p_2 \land F p_1) \Rightarrow ((p_3 \land X(\neg p_1 U p_4)) \Rightarrow X(\neg p_1 U(p_4 \land F p_0))))U p_1)$	8	47	21	133	982	18008	5	11
$F p_2 \Rightarrow (\neg p_2 U(p_2 \land (\neg p_0 U((p_0 U((\neg p_0 U((p_0 U(G \neg p_0 \lor G p_0)) \lor G \neg p_0)) \lor G \neg p_0)) \lor G \neg p_0))))$	9	36	13	54	159	407	7	14
$(F p_1 \Rightarrow ((p_3 \land X(\neg p_1 U p_4)) \Rightarrow X(\neg p_1 U(p_4 \land F p_0))))U p_1$	10	50	10	50	90	1476	4	8
$(((\bot V(p_1 \lor (\top U(\bot V p_0)))) \land (\bot V(p_2 \lor (\top U(\bot V \neg p_0))))) \lor (\bot V p_1)) \lor (\bot V p_2)$	11	39	13	43	441	1756	13	70
$(((\top U(\neg p_1 \land (\bot V(\top U \neg p_0)))) \lor (\top U(\neg p_2 \land (\bot V(\top U p_0))))) \land (\top U p_1)) \land (\top U p_2)$	17	94	19	98	485	4816	11	39
$(((\top U(\neg p_1 \land (\top U(\bot V \neg p_0)))) \lor (\top U(\neg p_2 \land (\top U(\bot V p_0))))) \land (\top U p_1)) \land (\top U p_2)$	25	112	33	148	1843	20008	7	28
$G(p_2 \Rightarrow (\neg((p_3 \land \neg p_1) \land X(\neg p_1 U(p_4 \land \neg p_1)))U((p_1 \lor p_0) \lor G \neg(p_3 \land XF p_4))))$	8	29	9	34	unfinished		8	27
$G((p_2 \Rightarrow ((p_3 \land X(\neg p_1 U p_4)) \Rightarrow X(\neg p_1 U(p_4 \land F p_0))))U(p_1 \lor G((p_3 \land X(\neg p_1 U p_4)) \Rightarrow X(\neg p_1 U(p_4 \land F p_0)))))$	12	97	26	242	unfinished		9	31
$G((p_2 \Rightarrow (p_0 \Rightarrow (\neg p_1 U((p_3 \land \neg p_1) \land X(\neg p_1 U p_4)))))U(p_1 \lor G(p_0 \Rightarrow (p_3 \land XF p_4))))$	11	62	19	100	unfinished		11	38
$G((p_2 \Rightarrow (p_0 \Rightarrow (\neg p_1 U(((p_3 \land \neg p_1) \land \neg p_5) \land X((\neg p_1 \land \neg p_5)U p_4)))))U(p_1 \lor G(p_0 \Rightarrow ((p_3 \land \neg p_5) \land X(\neg p_5 U p_4)))))$	11	62	19	100	unfinished		12	56

Table A.1: Benchmarks between $\mathcal{B}_{\mathcal{A}_\phi}$, $\neg\mathcal{B}_{\mathcal{A}_\phi}$ and $\mathcal{A}_{\neg\phi}$ with ϕ a Linear-time Temporal-Logic formula, on the number of states and transition of the automata. Formulæ are sorted according to the initial number of states and transitions.

References

Büchi, J. R. (1962). On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science, Berkley, 1960*, pages 1–11. Standford University Press. Republished in Lane and Siefkes (1990).

Couvreur, J.-M. (1999). On-the-fly verification of temporal logic. In Wing, J. M., Woodcock, J., and Davies, J., editors, *Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM'99)*, volume 1708 of *Lecture Notes in Computer Science*, pages 253–271, Toulouse, France. Springer-Verlag.

Duret-Lutz, A. (2007). *Contributions à l'approche automate pour la vérification de propriétés de systèmes concurents*. PhD thesis, Université Pierre et Marie Curie, Paris, France.

Duret-Lutz, A. and Poitrenaud, D. (2004). Spot: an extensible model checking library using transition-based generalized Büchi automata. In *Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04)*, pages 76–83, Volendam, The Netherlands. IEEE Computer Society Press.

Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. (1998). Property specification patterns for finite-state verification. In Ardis, M., editor, *Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP'98)*, pages 7–15, New York. ACM Press.

Etessami, K. and Holzmann, G. J. (2000). Optimizing Büchi automata. In Palamidessi, C., editor, *Proceedings of the 11th International Conference on Concurrency Theory (Concur'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 153–167, Pennsylvania, USA. Springer-Verlag.

Klein, J. and Baier, C. (2006). Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363:182–195.

Kupferman, O. and Vardi, M. Y. (1997). Weak alternating automata are not that weak. In *Proceedings of the 5st Israeli Symposium on Theory of Computing and Systems (ISTC'97)*, pages 147–158. IEEE Computer Society Press.

Lane, S. M. and Siefkes, D., editors (1990). *The Collected Works of J. Richard Büchi*. Springer-Verlag.

Löding, C. (1998). Methods for the transformation of ω-automata: Complexity and connection to second order logic. Master's thesis, University of Kiel.

Löding, C. (1999). Optimal bounds for transformations of ω-automata. In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 97–109.

McNaughton, R. (1966). Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530.

Michel, M. (1988). Complementation is more difficult with automata on infinite words. CNET, Paris, manuscrit cited by Löding (1999).

Muller, D. E. (1963). Infinite sequences and finite machines. In *SWCT '63: Proceedings of the 1963 Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 3–16.

Rabin, M. O. (1969). Decidability of second order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35.

Rabin, M. O. and Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125.

Safra, S. (1988). On the complexity of ω-automata. In *SFCS '88: Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 319–327, Washington, DC, USA. IEEE Computer Society.

Safra, S. (1989). *Complexity of Automata on Infinite Objects*. PhD thesis, The Weizmann Institute of Science, Rehovot, Israel.

Sistla, A. P., Vardi, M. Y., and Wolper, P. (1985). The complementation problem for Büchi automata with applications to temporal logic (extended abstract). In *Proceedings of the 12th Colloquium on Automata, Languages and Programming*, pages 465–474.

Somenzi, F. and Bloem, R. (2000). Efficient Büchi automata for LTL formulæ. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 247–263, Chicago, Illinois, USA. Springer-Verlag.

Streett, R. S. (1982). Propositional dynamic logic of looping and converse is elementary decidable. *Information and Control*, 54:121–141.

Tauriainen, H. (2000). Automated testing of Büchi automata translators for Linear Temporal Logic. Research Report A66, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland. Reprint of Master's thesis.

Tauriainen, H. and Heljanko, K. (2000). Testing SPIN's LTL formula conversion into Büchi automata with randomly generated input. In Havelund, K., Penix, J., and Visser, W., editors, *Proceedings of the 7th International SPIN Workshop on Model Checking of Software (SPIN'2000)*, volume 1885 of *Lecture Notes in Computer Science*, pages 54–72, Stanford University, California, USA. Springer-Verlag.

Vardi, M. Y. (2007). The Büchi complementation saga. In *Proceedings of the 17th Symposium on Theoretical Aspects of Computer Science (STACS'07)*, Aachen, Germany. Invited paper.

Yan, Q. (2008). Lower bounds for complementation of omega-automata via the full automata technique. *LMCS-4*, 1:5.