Complementing Büchi Automata Via Alternating Automata

Guillaume Sadegh

Technical Report n^o0915, January 2010
revision 2165

Model checking is a field of formal verification that aims to automatically test the behavior of a system with the help of temporal logic formulae. SPOT is a model checking library which relies on one technique: the automata-theoretic approach. In this approach, both system and formula are expressed with Büchi automata, which are automata on infinite words. SPOT provides several algorithms to deal with these automata, with model checking as objective. An operation for automata was recently added in SPOT: the complementation. Research of algorithms for this operation is still relevant today, since there is still no algorithm reaching the theoretical optimal bound. SPOT aims to provide features to deal with automata used in model-checking, then adding some new complementation algorithms in SPOT is interesting. We will present two recent complementation algorithms implemented in SPOT.

Keywords
Büchi automata, complementation, ranks, weak alternating automata, generalized Büchi automata.

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 that aims to automatically check the behavior of a system with the help of logic formulae. 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. The last operation added in SPOT was an algorithm to complement TGBA, using Safra (1988)'s construction. It is presented in [Sadegh (2009)].

Complementing an automaton means constructing another automaton that would recognized the complemented language of the initial one. This operation would be really useful in model checking since formulae are negated to represent unexpected behaviors, however in practice this operation is unrealistic. This is due to the lack of efficient algorithms, since even best ones, like Safra's construction have an exponential complexity. Therefore, model checking avoids complementation by different ways, but sometimes there is no escape to this operation.

To complement an automaton, Safra's algorithm transforms the original automaton into a deterministic automaton. The high complexity of the algorithm is due to this operation, which is more complicated than with finite automata. State-of-the-art complementation algorithms avoid determinization and transform the original automaton into a *weak alternating* automaton, which is a special kind of ω-automata. This method is expected to produce complemented automata with a smaller size than with Safra's construction, and then to be more efficient on the whole process to do model checking.

We have implemented in SPOT a new algorithm complement Büchi automata with weak alternating automata, and an extension of this algorithm to support Generalized Büchi acceptance conditions.

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 with weak alternating automata, by describing the algorithms we used. Chapter 3 will present how these algorithms are implemented in SPOT. In Chapter 4 we will conclude and we will present future works to accomplish on these complementation algorithms.

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. They use acceptance conditions instead of final states.

1.1 Definitions

1.1.1 ω-automata

Figure 1.1 presents the graphical representation of ω-automata.

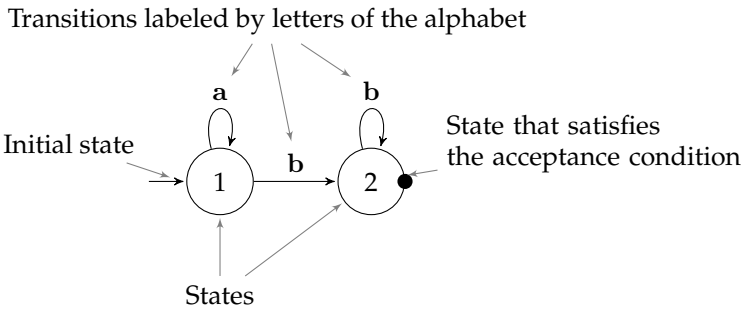


Figure 1.1: An ω-automaton

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
- $\delta : Q \times \Sigma \to 2^Q$ 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, is a word with the letter **a** repeated finitely and followed by an infinite number of **b** ($\mathbf{a}^\star\mathbf{b}^\omega$).

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

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 such as SPOT. However, we will focus in this report on state-labeled automata since all the algorithms that will be presented are designed for this kind of labeling.

Mode of transition-function and acceptance conditions 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, universal automata and alternating automata.

Definition 1.1.6. An automaton is called *deterministic* when each transition function $\delta(q, l)$ has exactly one successor. More formally, we can say: iff $\forall l \in \Sigma_\mathcal{A}$, $\forall q \in Q_\mathcal{A}, |\delta(q, l)| \leqslant 1$, then $\mathcal{A}$ is deterministic.
Example. Figure 1.2a represents a deterministic automaton since each state has exactly one successor for each letter l.

Definition 1.1.7. An automaton where transition function $\delta(q, l)$ may have more than one successor is called:

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

  Example. With the automaton in Figure 1.2b interpreted as a non-deterministic ω-automaton, the word **ba** is recognized if it has for run 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. With the automaton in Figure 1.2b interpreted as a universal ω-automaton, the word **ba** is recognized if it has for runs ①, ① AND ①, ②.

- *alternating*, when both non-deterministic and universal transitions can be used. To express these transition we use both a Boolean formula, denoted $\mathcal{B}^+(Q)$, with the logical quantifiers $\vee$ and $\wedge$.

  Example. Figure 1.2c represents an alternating automaton. In this automaton, successor states of ① are expressed with $\delta\left(①, \mathbf{a}\right) = \left(① \wedge ②\right) \vee ③$. This means that a run of the word **a** will reach states (① AND ②) OR ③.

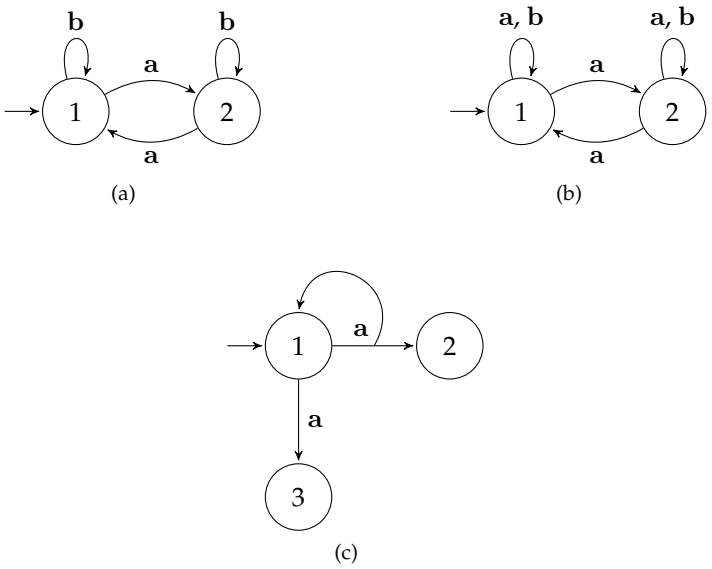


Figure 1.2: Deterministic (a), Non-deterministic and Universal (b), and Alternating (c) automata.

1.1.3 Acceptances conditions

The acceptance condition is a formula on states that defines the semantic of an accepting run. Spot relies on Büchi acceptation conditions, however to complement an automaton via an alternating automaton we also need to introduce co-Büchi acceptance conditions.

Büchi acceptance condition

Büchi (1962) introduced a kind of ω-automata that now bears his name. 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.

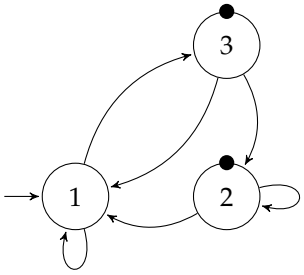


Figure 1.3: A (co-)Büchi automaton.

Generalized Büchi acceptance condition

Generalized Büchi automata are a variant of Büchi automata that is more succinct, 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 $\forall i \ \inf(\pi) \cap F_i \neq \emptyset$ 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 ○).

co-Büchi acceptance condition

Co-Büchi acceptance condition is the dual acceptance condition to Büchi acceptance condition.

The acceptance condition F is a set of states, and a run must visit finitely often all the states from F to be accepting.

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

Example. Figure 1.3 can represent a co-Büchi automaton with its states in the acceptance condition F marked with ●. A run of this automaton is accepting if it visits finitely often states ② AND ③.

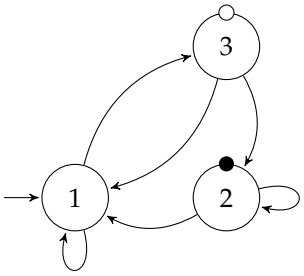


Figure 1.4: A generalized (co-)Büchi automaton.

Generalized co-Büchi acceptance condition

In the same manner as Büchi acceptance conditions, co-Büchi acceptance condition can have generalized conditions.

The Generalized co-Büchi acceptance condition has more than one set of acceptance conditions. A run is accepting if it passes through all the state of at least one set finitely often.

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

Example. Figure 1.4 represents a generalized co-Büchi automaton. A run is accepting if it visits finitely often one of the two acceptance conditions (states denoted with ● and ○).

Weak alternating automata

Muller et al. (1986) introduced weak alternating automata (WAA). In a WAA, the acceptance set is a set of states, and there exists a partition of Q (all the states of the automaton) into disjoint sets, Q_i, such that for each set Q_i, either $Q_i \subseteq F$, in that case Q_i is an *accepting* set, or $Q_i \cap F = \emptyset$, in that case Q_i is a *rejecting* set. There also exists a partial order between sets, such that for every $q \in Q_i$ and $q' \in Q_j$, for which $q' \in \delta(q, l)$ for some $l \in \Sigma$, we have $Q_j < Q_i$.

Weak alternating automata can be viewed as an automata with both a Büchi acceptance condition F and a co-Büchi acceptance condition $Q \setminus F$ (Kupferman and Vardi, 1997).

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 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), 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 alternating automata and universal co-Büchi automata. Universal co-Büchi automata are dual on the acceptance condition and the transition mode to non-deterministic Büchi automata. An universal co-Büchi automata with n states can be translated into an alternating automata with n^2 states. The exponential blow-up in this complementation is due to the transformation from alternating automata into Büchi automata, which has an $2^{O(n)}$ complexity. However, they decrease the upper bound up to $(6n)^n$. Kupferman and Vardi (2005) have presented some techniques to enhance this approach by the reduction the size of the alternating automata before its transformation into a Büchi automata. Theirs techniques rely on simulation on the automata. (Gurumurthy et al., 2003) has presented some techniques to support Generalized Büchi acceptances conditions with this construction.

Chapter 2

Complementing Büchi automata

In this section, we will present algorithms that we use to complement Büchi automata. We can resume the use of the different algorithms that we will present in Figure 2.1.

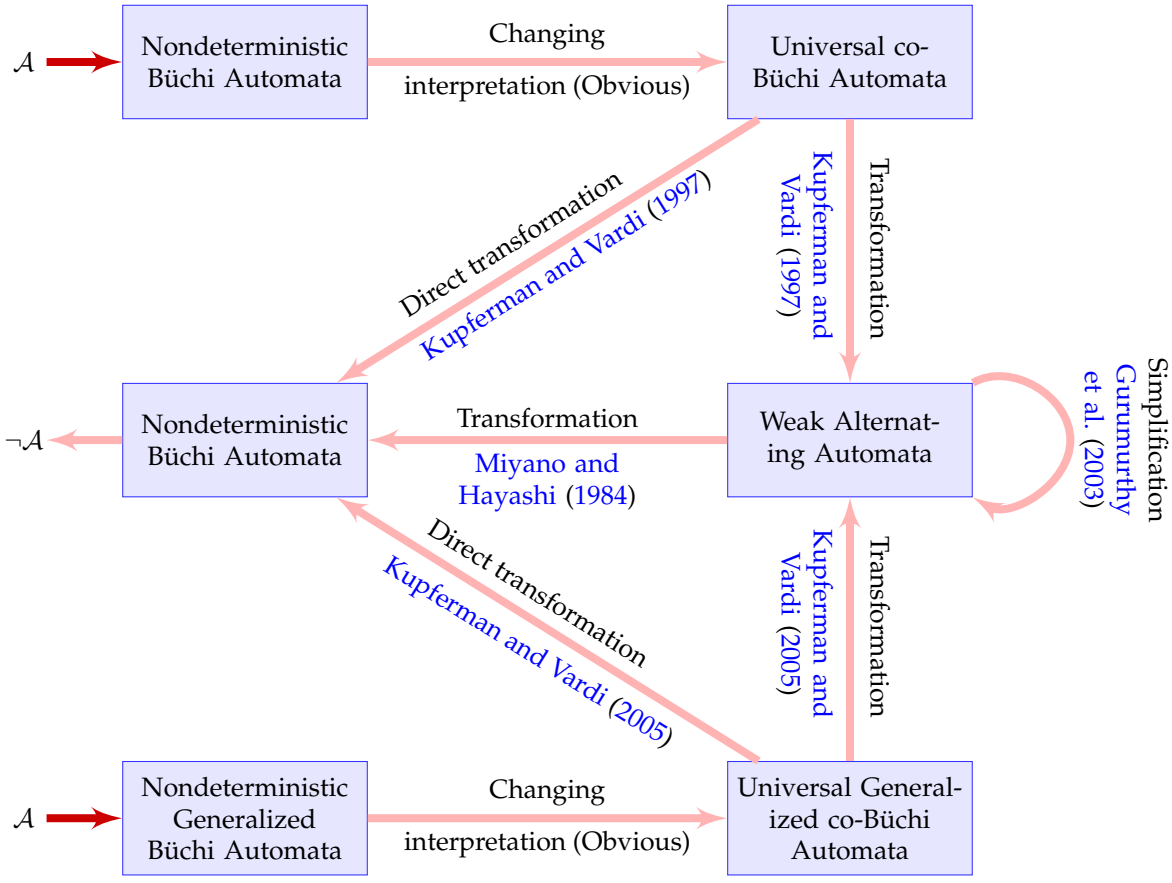


Figure 2.1: Steps of the complementation.

2.1 Ranks and complementation

The complementation introduced by Kupferman and Vardi (1997) relies on ranks, that we will present in this section.

Let $\mathcal{A} = (\Sigma, Q, q_{in}, \delta, F)$ an universal co-Büchi automaton and $n = |Q|$. The run of $\mathcal{A}$ on an infinite word can be represented by a directed acyclic graph (DAG) $G_r = (V, E)$, with

- $V \subseteq Q \times \mathbb{N}$.
- $E \subseteq \bigcup_{l \geq 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ is such that $E(\langle q, l\rangle, (q', l+1))$ iff there exists a transition that reaches q' from q. Formally the condition is $q' \in \delta(q, \sigma_e)$ with $\sigma_e \in \Sigma$.

Figure 2.2 represents a co-Büchi automaton and Figure 2.3 its DAG.

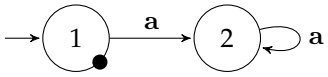


Figure 2.2: A co-Büchi automaton.

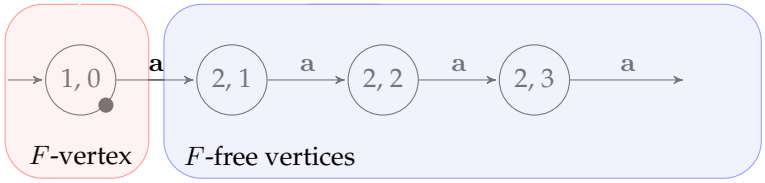


Figure 2.3: DAG of co-Büchi automaton.

Definition 2.1.1. A vertex $\langle q, l\rangle$ is a F-vertex iff $q \in F$. It is easy to see that a run π is accepting iff all paths in G_r have only finitely many F-vertices. Indeed, by definition in 1.1.3 a run in a co-Büchi automaton is accepting if $\inf(\pi) \cap F = \emptyset$

The DAG in Figure 2.3 has only one F-vertex: $(1, 0)$

Definition 2.1.2. Consider a sub-DAG $G \subseteq G_r$. We say that a vertex (q, l) is F-free in G iff all the vertices in G that are reachable from $\langle q, l\rangle$ are not F-vertices.

All the vertices from the DAG in Figure 2.3 with $q = 2$ are F-free. This is obvious since from the state 2, we cannot reach the state 1 that holds the acceptance condition.

Definition 2.1.3. Consider a sub-DAG $G \subseteq G_r$. We say that a vertex $\langle q, l\rangle$ is finite in G iff only finitely many vertices in G are reachable from $\langle q, l\rangle$.

The DAG in Figure 2.3 has no finite vertices. However, if we remove all the F-free vertices of this graph, the vertex (1.0) would be finite.

Given an accepting run π, we define an infinite sequence $G_0 \supseteq G_1 \supseteq G_2 \supseteq \ldots$ of DAGs define as follow:

- $G_0 = G_r$
- $G_{2i+1} = G_{2i} \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is finite in } G_{2i}\}$

- $G_{2i+2} = G_{2i+1} \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is } F\text{-free in } G_{2i+1}\}$

The interesting property with DAG is that if π is accepting, then G_{2n} will always be empty, with n the number of states in the initial automaton. In an accepting run, infinite path are F-free, and the accepting condition is on finite states. There is at most n finite states, then each deletion has to be called at most n times to produce an empty set.

Definition 2.1.4. Each vertex $\langle q, l\rangle$ has an unique index $i \geq 1$ such that $\langle q, l\rangle$ is either finite in G_{2i} or F-free in G_{2i+1}. Given a vertex $\langle q, l\rangle$, we define the *rank* of $\langle q, l\rangle$ denoted rank($\langle q, l\rangle$), as follows:

$$\text{rank}(\langle q, l\rangle) = \begin{cases} 2i & \text{If } \langle q, l\rangle \text{ is finite in } G_{2i} \\ 2i+1 & \text{If } \langle q, l\rangle \text{ is } F\text{-free in } G_{2i+1} \end{cases}$$

For $k \in \mathbb{N}$, let $[k]$ denote set $\{0, 1, \ldots, k\}$, and let $[k]^{\text{odd}}$ denote the set of odd members of $[k]$. The rank of every vertex in G_r is in $[2n]$.

If a state has an odd rank, then it was deleted because it was considered as F-free. If the state was F-free, there was not condition to visit finitely often this state, so a run can visit infinitely often this state.

Example

We can construct DAGs of the co-Büchi automaton in Figure 2.2 to compute the rank of each vertex. This construction is presented in Figure 2.4.

2.2 From universal co-Büchi to weak alternating automata

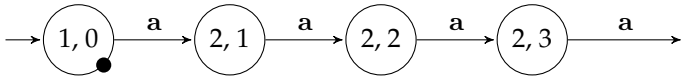
The transformation from Büchi or co-Büchi to weak alternating automata is a quadratic construction that relies on the ranks introduced in the previous section.

Let $\mathcal{A} = (\Sigma, Q, q_{in}, \delta, F)$ be an alternating co-Büchi automaton and $n = |Q|$. There exists a weak alternating Büchi automaton $\mathcal{A}' = (\Sigma, Q', q'_{in}, \delta', F')$ such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{A}')$. This automaton is constructed as follows:

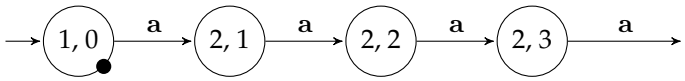
- $Q' = Q \times [2n]$. A state $\langle q, l\rangle \in Q'$ indicates that the current level of the DAG contains the state q and the guessed level ranking for the current level is l.
- $q'_{in} = \langle q_{in}, 2n\rangle$, with $2n$ the upper bound of the rank of $\langle q_{in}, 0\rangle$.
- We define δ' by means a function $release : \mathcal{B}^+(Q) \times [k] \to \mathcal{B}^+(Q')$. Given a formula $\theta \in \mathcal{B}^+(Q)$ and a rank $i \in [k]$, the formula $release(\theta, i)$ is obtained from θ by replacing an atom q by the disjunction $\bigvee_{i' \leq i} (q, i')$. For example, $release(q_1 \wedge q_2, 2) = (\langle q_1, 2\rangle \vee \langle q_1, 1\rangle \vee \langle q_1, 0\rangle) \wedge (\langle q_2, 2\rangle \vee \langle q_2, 1\rangle \vee \langle q_2, 0\rangle)$. We define δ' for a state $\langle q, i\rangle \in Q'$ and $\sigma_e \in \Sigma$ as follows:

$$\delta'(\langle q, i\rangle, \sigma_e) = \begin{cases} release(\delta(q, \sigma_e), i) & \text{If } q \notin F \text{ or } i \text{ is even} \\ \text{false} & \text{If } q \in F \text{ and } i \text{ is odd} \end{cases}$$

That is, if the current guessed rank is i then, by employing *release*, the run can move to its successors at any rank that is smaller than i. However, if $q \in F$ and the current guessed rank is odd, then by the definition of ranks the current guessed rank is wrong, and the run is rejecting.



(a) $G_0 = G_r$

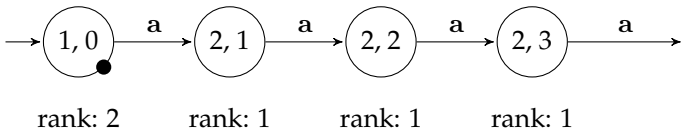


(b) $G_1 = G_0 \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is finite in } G_0\}$

(c) $G_2 = G_1 \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is } F\text{-free in } G_1\}$

(d) $G_3 = G_2 \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is finite in } G_2\}$

(e) $G_4 = \emptyset$. So the run-DAG was accepting.



(f) Define the rank of each vertex.

Figure 2.4: DAG and ranks of co-Büchi automaton.

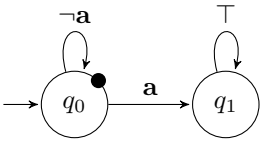


Figure 2.5: An automaton to complement

- $F' = Q \times [k]^{odd}$, since infinitely many guessed ranks along a path should be odd.

This algorithm is illustrated in Figure 2.6 that shows the algorithm on the automaton illustrated in Figure 2.5. Since this automaton has only one letter in its alphabet (a), the resulting automaton doesn't have alternation.

Once the translation from Universal Co-Büchi Automata to Weak Alternating Automata is done, complementation is reduced to remove the alternation from the Alternating Büchi Automata, since Weak Alternating is a special case of Büchi Alternating.

2.3 Complementing non-determinitic Büchi automata

From the transformation into weak alternating automata that we have presented, its easy to complement a Büchi automata. Figure 2.7 presents current steps of the complementation.

2.3.1 Complementation via alternating automata

From the illustration 2.7 we can notice that we need an algorithm to convert weak alternating automata into Büchi automata to have a complementation working. Miyano and Hayashi (1984) have introduced an algorithm to do this transformation.

This transformation is defined as follow:

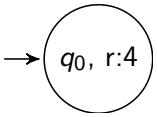
Let $\mathcal{A} = (\Sigma, Q, q_{in}, \delta, F)$ be an alternating Büchi automaton. There is a non-deterministic Büchi automaton $\mathcal{A}' = (\Sigma, 2^Q \times 2^Q, \langle\{q_{in}\}, \emptyset\rangle, \delta', 2^Q \times \{\emptyset\})$, such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{A}')$, where δ' is defined, for all $\langle S, O\rangle \in 2^Q \times 2^Q$ and $\sigma_e \in \Sigma$, with:

- if $O \neq \emptyset$, then
  $\delta'(\langle S, O\rangle, \sigma_e) = \{\langle S', O' \setminus F\rangle \mid S' \text{ satisfies } \bigwedge_{q \in S} \delta(q, \sigma_e), O' \subseteq S' \text{ and } O' \text{ satisfies } \bigwedge_{q \in O} \delta(q, \sigma_e)\}$.
- if $O = \emptyset$, then
  $\delta'(\langle S, O\rangle, \sigma_e) = \{\langle S', S' \setminus F\rangle \mid S' \text{ satisfies } \bigwedge_{q \in S} \delta(q, \sigma_e).\}$

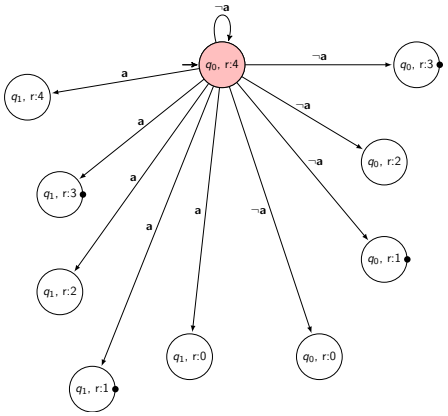
2.3.2 Complementation without alternating automata

We can avoid the construction of the weak alternating automata with a composition of the three constructions. This approach is presented by Kupferman and Vardi (1997).

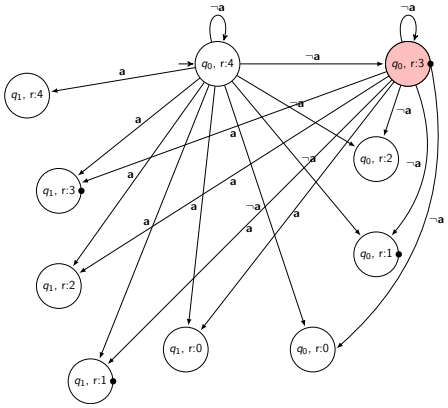
Given a non-determinitic Büchi automaton $\mathcal{A} = (\Sigma, Q, q_{in}, \delta, F)$, we define a non-deterministic Büchi automaton $\mathcal{A}'$ such that $\mathscr{L}(\mathcal{A}') = \Sigma^\omega \setminus \mathscr{L}(\mathcal{A})$. Before we give the construction method for $\mathcal{A}'$, we need to introduce some notations. A *level ranking* for $\mathcal{A}$ is a function $g : Q \to \mathbb{N}$ such that $g(q)$ is odd if $q \notin F$. Let $\mathcal{R}$ be the set of all level rankings. For two level rankings g and g' in $\mathcal{R}$, a set $S \subseteq Q$, and a letter $\sigma_e \in \Sigma$, we say $g' covers \langle g, S, \sigma_e\rangle$ if for all $q \in S$ and $q' \in Q$, if



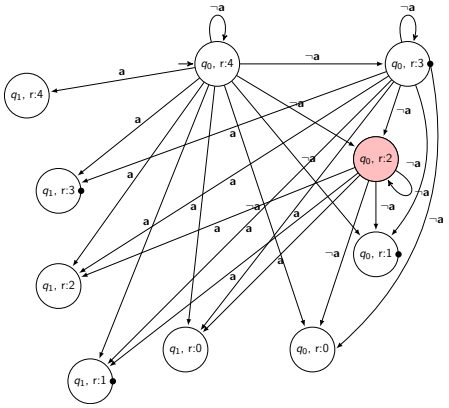
(a) The initial state, with a rank of 4 (the initial automaton has 2 states)



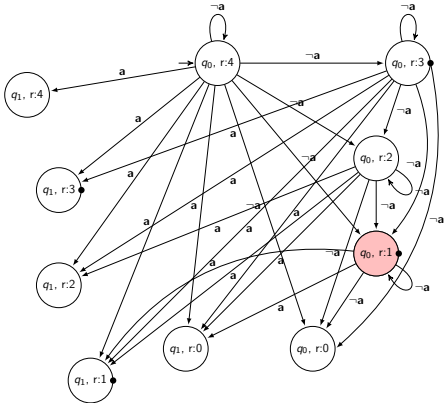
(b) The state takes as successor all its successors in the original automaton with all the ranks ≤ 4. All states with an odd rank are accepting.



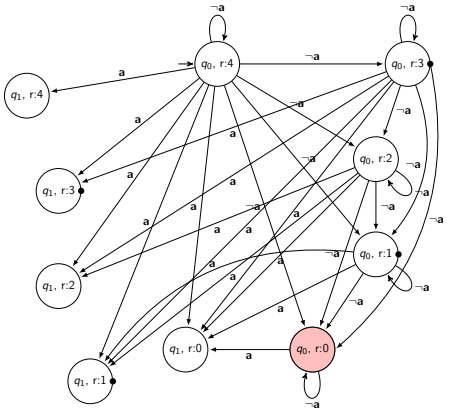
(c) The state takes as successor all its successors in the original automaton with all the ranks ≤ 3



(d) The state takes as successor all its successors in the original automaton with all the ranks ≤ 2

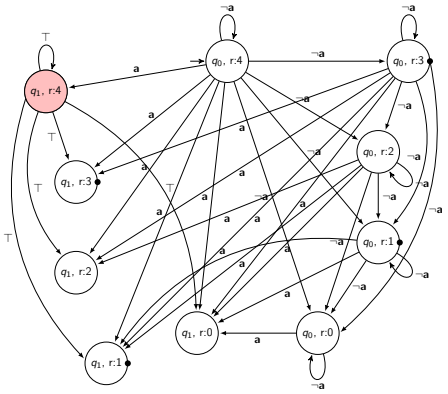


(e) The state takes as successor all its successors in the original automaton with all the ranks ≤ 1

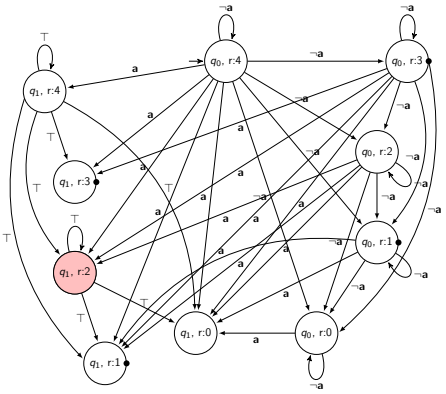


(f) The state takes as successor all its successors in the original automaton with all the ranks equal to 0

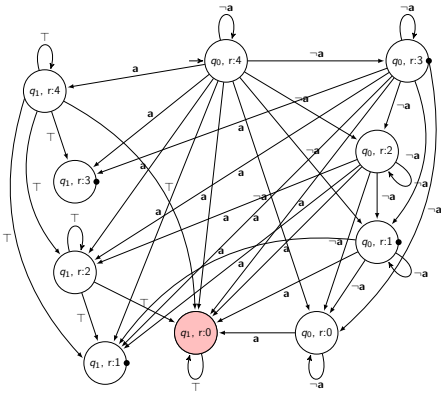
Figure 2.6: A weak alternating automaton produced from the automaton in Figure 2.5. Since this automaton has only one letter in its alphabet, no conjunction of states appears.



(g) The state takes as successor all its successors in the original automaton with all the ranks ≤ 4



(h) The state takes as successor all its successors in the original automaton with all the ranks ≤ 2



(i) The final loop make the automaton constructed.

Figure 2.6: A weak alternating automaton produced from the automaton in Figure 2.5. Since this automaton has only one letter in its alphabet, no conjunction of states appears.

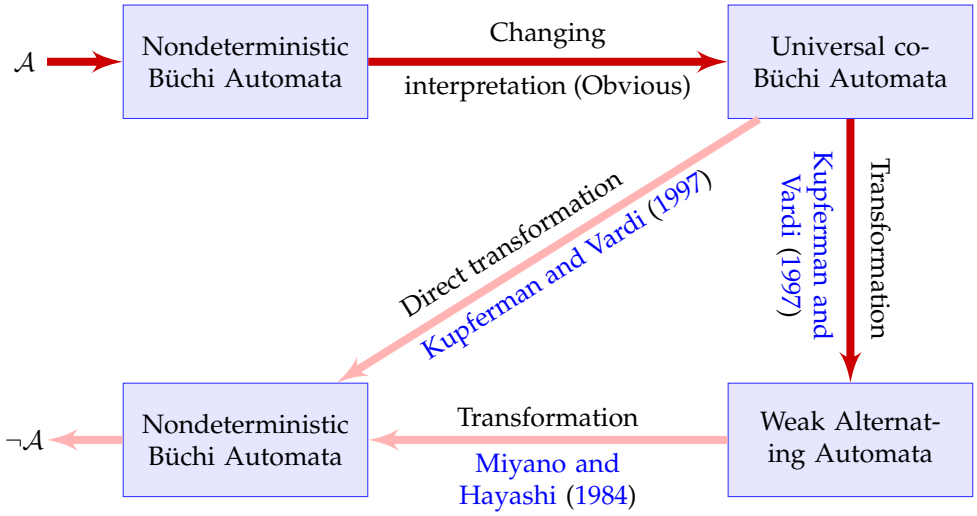


Figure 2.7: Steps of the complementation.

$q' \in \delta(q, \sigma_e)$, then $g'(q') \leq g(q)$. Finally, we define $odd(g)$ that contains states to which g gives an odd rank.

Now, we can construct $\mathcal{A}' = (\Sigma, Q', q'_{in}, \delta', F')$ where:

- $Q' = 2^Q \times 2^Q \times \mathcal{R}$. A state $\langle S, O, g\rangle \in Q'$ indicates that the current level of the DAG contains the state S and the guessed level ranking for the current level is g. The set $O \subseteq S$ contains states along paths that have not visited a vertex with an odd rank since the last time O has been empty.
- $q'_{in} = \langle\{q_{in}, \emptyset, g_{in}\}\rangle$, where $g_{in}(q) = 2n$ for all $q \in Q$.
- δ' is defined, for all $\langle S, O, g\rangle \in Q$ and $\sigma_e \in \Sigma$, as follows.
  - If $O \neq \emptyset$, then $\delta'(\langle S, O, g\rangle, \sigma_e) = \{\langle \delta(S, \sigma_e), \delta(O, \sigma_e) \setminus odd(g'), g'\rangle : g' \text{ covers } \langle g, S, \sigma_e\rangle\}$.
  - If $O = \emptyset$, then $\delta'(\langle S, O, g\rangle, \sigma_e) = \{\langle \delta(S, \sigma_e), \delta(S, \sigma_e) \setminus odd(g'), g'\rangle : g' \text{ covers } \langle g, S, \sigma_e\rangle\}$.
- $F' = 2^Q \times \{\emptyset\} \times \mathcal{R}$.

2.4 Simplifications of alternating automata

From a non-determinitic Büchi automaton with n states, we get an universal co-Büchi automaton with n states, then a weak alternating automaton with $O(n^2)$ states, and finally a non-deterministic Büchi automaton with $2^{O(n^2)}$. However, Michel (1988) and Safra (1988) have shown that an optimal complementation for a non-deterministic Büchi automaton results in an automaton with $2^{O(n \log n)}$ states.

Gurumurthy et al. (2003) present some simplification techniques on weak alternating automata to reduce the size of the automata from n^2 to $n \log n$. The most interesting technique relies on simulation on the automata. For more details we refer to Gurumurthy et al. (2003).

2.5 Complementing non-determinitic generalized Büchi automata

Since SPOT relies on generalized Büchi acceptance conditions, having a complementation algorithm that works directly on those conditions is interesting. Kupferman and Vardi (2005) have presented an extension of their method with ranks that works for generalized Büchi automata. We will present this method in this section.

2.5.1 Ranks updated for generalized acceptance conditions

We introduce ranks for generalized co-Büchi acceptance conditions in the same way that we did for co-Büchi acceptance conditions.

Let $\mathcal{A} = (\Sigma, Q, q_{in}, \delta, F)$ an universal generalized co-Büchi automaton and $n = |Q|$. The run of $\mathcal{A}$ on an infinite word can be represented by a directed acyclic graph (DAG) $G_r = (V, E)$, with

- $V \subseteq Q \times \mathbb{N}$.
- $E \subseteq \bigcup_{l \geq 0} (Q \times \{l\}) \times (Q \times \{l+1\})$ is such that $E(\langle q, l\rangle, (q', l+1))$ iff there exists a transition that reaches q' from q. Formally the condition is $q' \in \delta(q, \sigma_e)$ with $\sigma_e \in \Sigma$.

Figure 2.8 represents a Generalized co-Büchi automaton with $F = \{\bullet, \circ\}$ and Figure 2.9 its DAG.

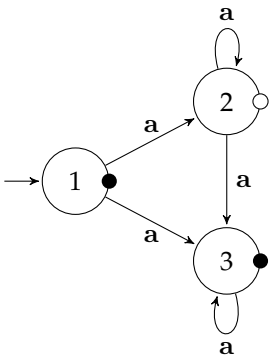


Figure 2.8: A generalized co-Büchi automaton.

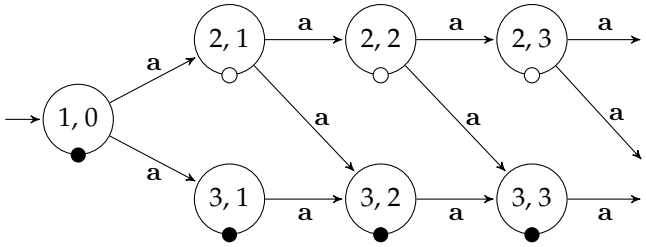


Figure 2.9: A DAG with $\mathbf{a}^\omega$ for the automaton in Figure 2.8.

We consider automata with $F = \{F_1, \ldots, F_n\}$ as acceptance condition.

Definition 2.5.1. A vertex $\langle q, l\rangle$ is a F_j-vertex iff $q \in F_j$.

Example: the DAG in Figure 2.3 has an infinite number of ●-vertices (from the state 3) and ○-vertices (from the state 2).

Definition 2.5.2. Consider a DAG $G \subseteq G_r$. We say that a vertex (q, l) is F_j-free in G iff all the vertices in G that are reachable from $\langle q, l\rangle$ are not F_j-vertices.

Example: the vertices that refers to the state 3 in the DAG in Figure 2.3 are ○-free since no ○ acceptance condition can be reach from those vertices.

Definition 2.5.3. Consider a DAG $G \subseteq G_r$. We say that a vertex (q, l) is finite in G iff only finitely many vertices in G are reachable from $\langle q, l\rangle$.

Given an accepting run π and k the number of accepting conditions, we define an infinite sequence $G_0 \supseteq G_1^1 \supseteq G_1^2 \supseteq \cdots G_1^k \supseteq G_1^{k+1} \supseteq G_3^1 \supseteq \cdots G_3^{k+1} \ldots$ of DAGs. To simplify notations, we sometimes refer to G_{2i+1}^{k+1} as G_{2i+2}. Thus, $G_2 = G_1^{k+1}$.

- $G_0 = G_r$
- $G_{2i+1}^1 = G_{2i} \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is finite in } G_{2i}\}$
- $G_{2i+1}^{j+1} = G_{2i+1}^j \setminus \{\langle q, l\rangle \mid \langle q, l\rangle \text{ is } F_j\text{-free in } G_{2i+1}^j\}$, for $1 \leq j \leq k$.

As for co-Büchi acceptance condition, if π is accepting, then G_{2n} will be empty.

Definition 2.5.4. Each vertex $\langle q, l\rangle$ has an unique index $i \geq 1$ such that $\langle q, l\rangle$ is either finite in G_{2i} or F_j-free in G_{2i+1}^j. Given a vertex $\langle q, l\rangle$, we define the *rank* of $\langle q, l\rangle$ denoted $rank : V \to R$, as follows:

$$\text{rank}(\langle q, l\rangle) = \begin{cases} 2i & \text{If } \langle q, l\rangle \text{ is finite in } G_{2i} \\ \langle 2i+1, j\rangle & \text{If } \langle q, l\rangle \text{ is } F_j\text{-free in } G_{2i+1}^j \end{cases}$$

Example

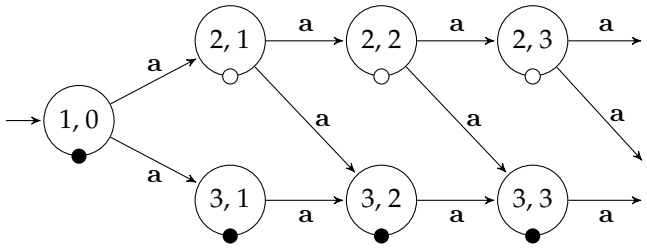
We can construct DAGs of the generalized co-Büchi automaton in Figure 2.8 to compute the rank of each vertex. This construction is presented in Figure 2.10.

2.5.2 The complementation

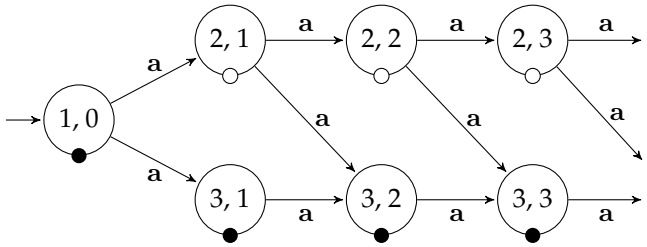
The complementation construction presented in Subsection 2.3.2 can be adapted to generalized acceptance conditions with some adjustments on ranks.

Given a non-determinitic generalized Büchi automaton $\mathcal{A} = (\Sigma, Q, q_{in}, \delta, F)$, we define a non-deterministic Büchi automaton $\mathcal{A}'$ such that $\mathscr{L}(\mathcal{A}') = \Sigma^\omega \setminus \mathscr{L}(\mathcal{A})$. Before we give the construction method for $\mathcal{A}'$, we need redefine some notations. A *level ranking* for $\mathcal{A}$ is a function $g : Q \to R$ such that $g(q)$ is odd with an index j if $q \notin F_j$. Let $\mathcal{R}$ be the set of all level rankings. The definition for covers is the same than in Subsection 2.3.2: for two level rankings g and g' in $\mathcal{R}$, a set $S \subseteq Q$, and a letter $\sigma_e \in \Sigma$, we say $g' covers \langle g, S, \sigma_e\rangle$ if for all $q \in S$ and $q' \in Q$, if $q' \in \delta(q, \sigma_e)$, then $g'(q') \leq g(q)$. Finally, we define $odd(g)$ that contains states to which g gives an odd rank.

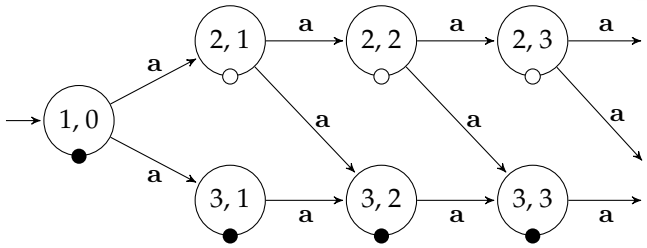
Now, we can construct $\mathcal{A}' = (\Sigma, Q', q'_{in}, \delta', F')$ where:



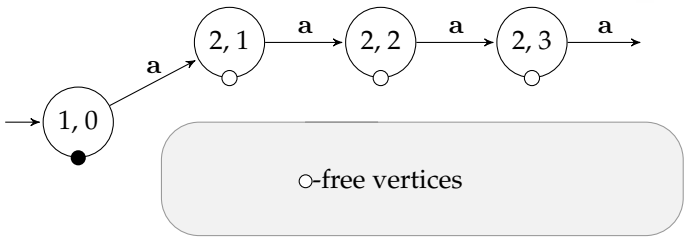
(a) $G_0 = G_r$



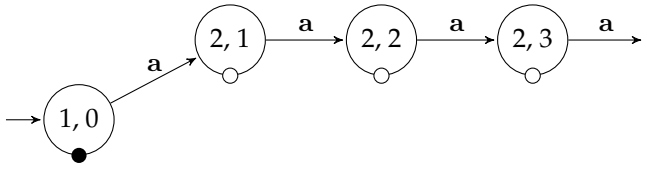
(b) $G_1^1 = G_0 \setminus \{angleq, langle \mid angleq, langle \text{ is finite in } G_0\}$



(c) $G_1^2 = G_1^1 \setminus \{angleq, langle \mid angleq, langle \text{ is } \bullet\text{-free in } G_1^1\}$



(d) $G_2 = G_1^2 \setminus \{angleq, langle \mid angleq, langle \text{ is } \circ\text{-free in } G_1^2\}$

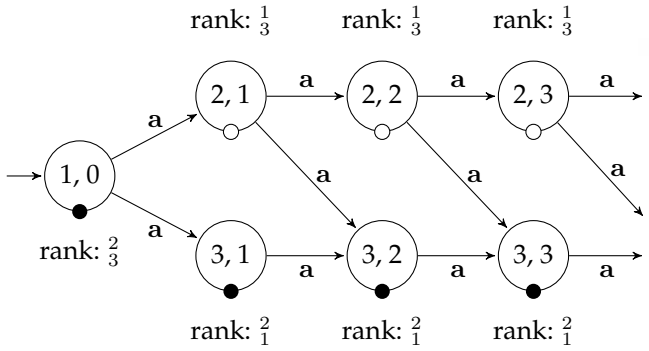


(e) $G_3^1 = G_2 \setminus \{angleq, langle \mid angleq, langle \text{ is finite in } G_2\}$

Figure 2.10: DAG ...

(f) $G_3^2 = G_3^1 \setminus \{angleq, langle \mid angleq, langle$ is ● -free in $G_3^1\}$

(g) $G_4 = G_3^2 \setminus \{angleq, langle \mid angleq, langle$ is ○-free in $G_3^2\}$



(h) We can set ranks

Figure 2.10: . . . and ranks of generalized co-Büchi automaton.

- $Q' = 2^Q \times 2^Q \times \mathcal{R}$. A state $\langle S, O, g\rangle \in Q'$ indicates that the current level of the DAG contains the state S and the guessed level ranking for the current level is g. The set $O \subseteq S$ contains states along paths that have not visited a vertex with an odd rank since the last time O has been empty.
- $q'_{in} = \langle\{q_{in}, \emptyset, g_{in}\}\rangle$, where $g_{in}(q) = 2n$ for all $q \in Q$.
- δ' is defined, for all $\langle S, O, g\rangle \in Q$ and $\sigma_e \in \Sigma$, as follows.
  - If $O \neq \emptyset$, then $\delta'(\langle S, O, g\rangle, \sigma_e) = \{\langle \delta(S, \sigma), \delta(O, \sigma_e) \setminus odd(g'), g'\rangle : g' \text{ covers } \langle g, S, \sigma_e\rangle\}$.
  - If $O = \emptyset$, then $\delta'(\langle S, O, g\rangle, \sigma_e) = \{\langle \delta(S, \sigma_e), \delta(S, \sigma_e) \setminus odd(g'), g'\rangle : g' \text{ covers } \langle g, S, \sigma_e\rangle\}$.
- $F' = 2^Q \times \{\emptyset\} \times \mathcal{R}$.

We illustrate this algorithm in Figure 2.11 that presents the beginning of the complementation of the automaton in Figure 2.8.

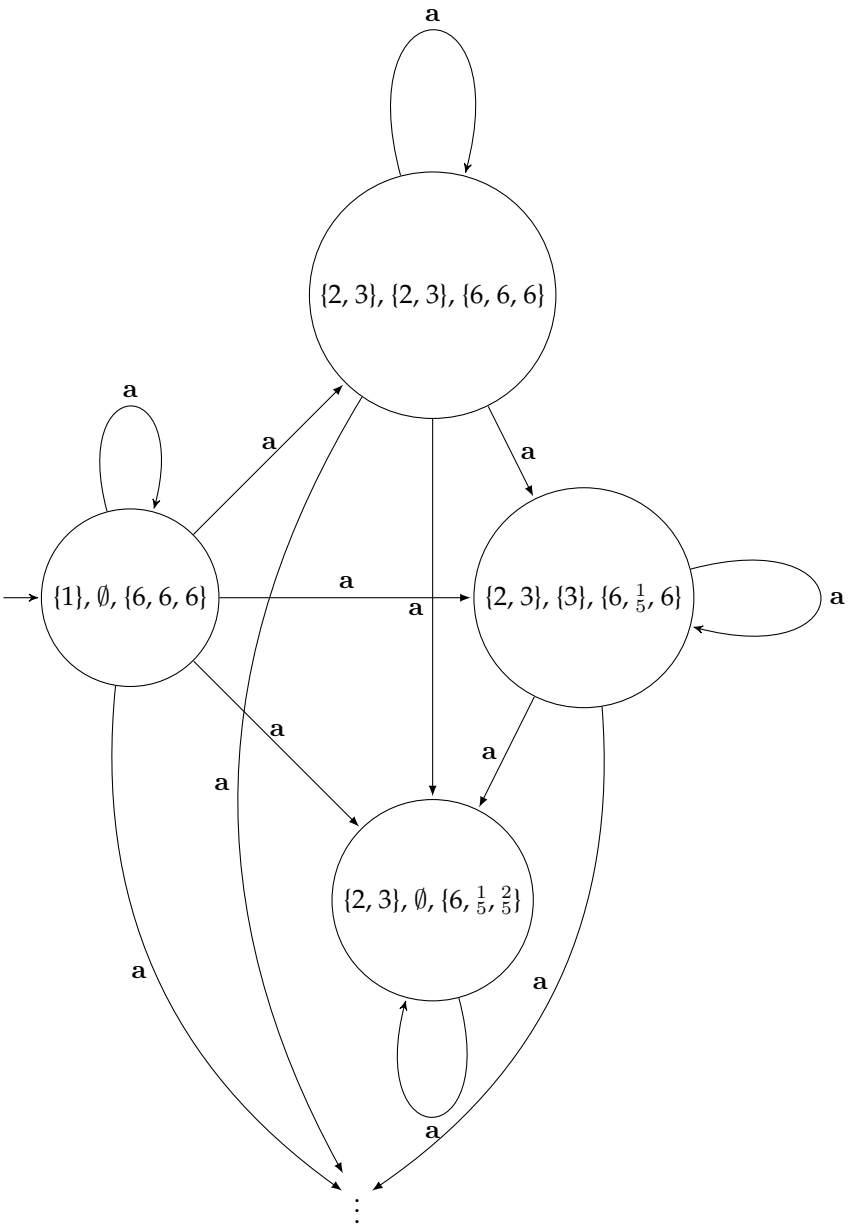


Figure 2.11: The beginning of the complementation of the automaton in Figure 2.8.

Chapter 3

Implementation in SPOT

3.1 Complementation on Büchi acceptance conditions

3.1.1 Alternating Automata in SPOT

To implement the complementation algorithm of Büchi acceptance conditions through weak alternating automata, we have added in SPOT a new interface to represent alternating automata. We have implemented an interface for a kind of automata that we called `saba`, for State-based Alternating Büchi Automata, which is the most common kind of alternating automata.

This interface is inspired by the interface of `tgba` (Transition-based Generalized Büchi Automata), which is the kind of automata used in the entire library. Figure 3.1 represents the class hierarchy of this new interface.

3.1.2 The algorithm

We have implemented the algorithm presented in Section 2.2 in a class that implements the `saba` interface. The algorithm follows the construction given previously:

- A state is a pair of an original state with a rank
- Transitions are computed *on-the-fly* in a class that implements saba_succ_iter.
- Acceptance conditions are store in states.

3.2 Complementation on Generalized Büchi acceptance conditions

We have also implemented the algorithm to complement Generalized Büchi Automata presented in Section 2.5. This algorithm was implemented without constructing explicitly a weak alternating automaton. This construction is presented in Subsection 2.5.2. The algorithm implement an interface of `tgba` in a class called `tgba_complement`, and can be used with all the tools provided by SPOT. The construction of the automaton is done *on-the-fly* since the successors of a state are computed only on request. Moreover, since this algorithm works for automata with labels on states, and SPOT relies on automata with labels on transitions, we have added an algorithm to transform automata with labels on transitions in automata with labels

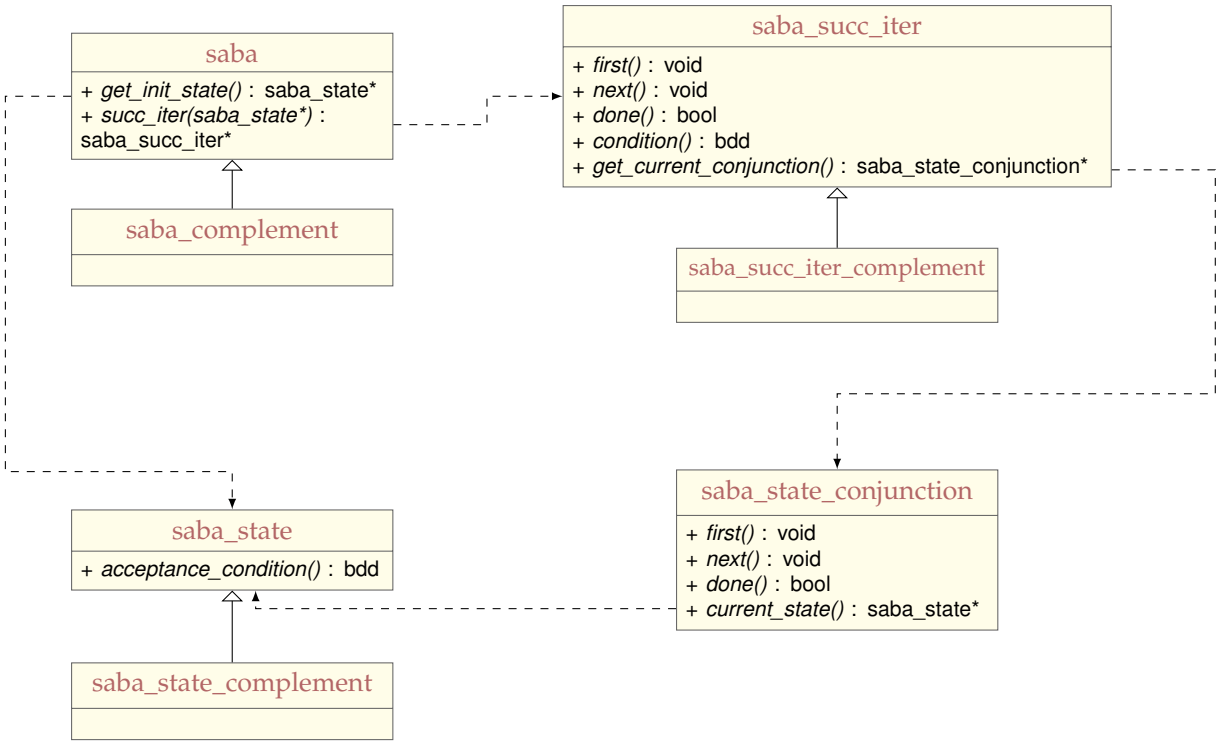
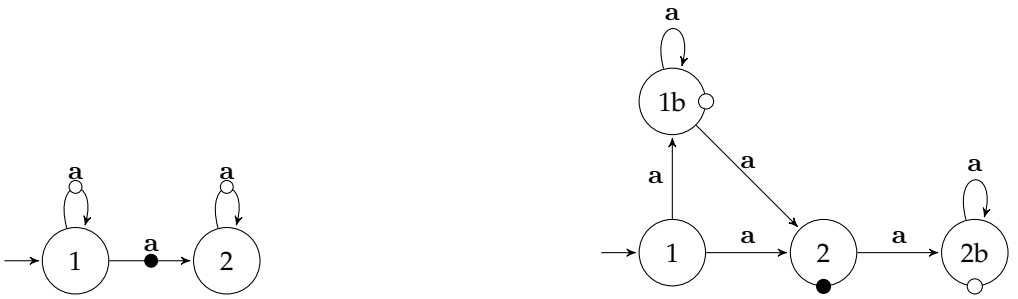


Figure 3.1: Class diagram for the family of `saba` classes

on states. This transformation can produce an automaton with $|F| \times |Q|$ states, with $|F|$ the number of acceptance conditions and $|Q|$ the number of states in the initial automaton. This transformation is illustrated in Figure 3.2. Transforming an automaton with labels on states into an automaton with labels on transitions is straightforward. Each transition has to take the acceptance conditions of its source. This transformation is illustrated in Figure 3.3.



(a) The original automaton (transitions-labeled)

(b) The transformed automaton (states-labeled)

Figure 3.2: An automaton with labels on transitions into an automaton with labels on states.

(a) The original automaton (states-labeled)

(b) The transformed automaton (transitions-labeled)

Figure 3.3: An automaton with labels on states into an automaton with labels on transitions.

3.3 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 be 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.

These tests have been done on the algorithm that complements Generalized Büchi acceptance conditions. Since the algorithm produce a `tgba`, we can use our tools to test its implementation. The algorithm that deals with Büchi acceptance conditions via alternating automata has not

Original	Ranks			Safra			$\neg\varphi$		
st	**st**	**tr**	**acc**	**st**	**tr**	**acc**	**st**	**tr**	**acc**
1	4.3	10.7	1.0	6.0	13.7	1.0	3.0	5.7	1.3
2	19.6	236.9	1.0	11.3	44.2	1.2	2.6	4.9	0.9
3	1595.8	387510.6	1.0	19.4	99.7	1.4	2.7	4.8	1.2
4	6486.3	3757235.0	1.0	33.2	273.3	2.2	3.3	6.5	1.4
5	9060.5	4689062.5	1.0	141.0	730.0	3.0	2.5	6.0	1.0
6	12107.7	81361931.3	1.0	27.2	195.8	2.0	3.0	5.7	1.7
7	x	x	x	157.0	1325.5	5.0	3.0	7.0	1.0

Table 3.1: Some benchmarks with this new implementation

been tested with these tools since the algorithm to transform a weak alternating automaton into a `tgba` is still missing. Generating a display of the alternating automaton was the only way to test whether the algorithm seems to work.

3.4 Benchmarks

It will be interesting to show some benchmarks between the complementation routines implemented in SPOT (Safra's construction, Büchi automata through weak automata, generalized Büchi automata directly into Büchi automata). However, since currently we have no implementation for the translation from weak alternating automaton to Büchi automaton, we can compare Safra's complementation and the direct complementation presented in the report.

The direct complementation uses generalized Büchi acceptance conditions as input, and produces an automaton with Büchi acceptance conditions. Safra's complementation takes an automaton with generalized Büchi acceptance conditions as input, and has a first step to transform this automaton with Büchi acceptance conditions. It produces an automaton with generalized Büchi acceptance conditions. It was presented in Sadegh (2009).

Methodology For each formula φ from a set of logic formulae, we produce a generalized Büchi automaton that represents this formula. Then we use our algorithm that relies on ranks, our previous algorithm that relies on Safra's complementation, and we also produce the automaton whose language recognizes $\neg\varphi$.

For each original automaton with n states, we present the average number of states, transitions and acceptance conditions in the three resulting automata.

Result Table 3.1 presents the results. We can notice that the algorithm that relies on rank has a really bad behavior, compared to Safra's complementation. However, both algorithms are supposed to have the same complexity in term of $O()$, but in practice we realize that Safra's complementation produces automata with a smaller number of states and transitions. We can notice that from an original automaton with 5 states, the rank based complementation doesn't always success to produce a complemented automaton, due to memory overflow.

Chapter 4

Conclusion and perspectives

4.1 Conclusion

SPOT provides an algorithm to complement Büchi automata called Safra's construction. This algorithm is supposed to be theoretically optimal, but some algorithm with the same complexity were presented.

We have started to implement these algorithms that rely on alternating automata and ranks. Implementing those algorithms has require to introduce a new hierarchy of automata in SPOT to deal with alternation. In the same time, we have implemented several versions of the original algorithm: for Büchi acceptance conditions and generalized Büchi acceptance conditions, with the construction of the alternating automaton or without its construction.

Even with some bricks missing to provide some benchmarks with all the version of this complementation, we have notice with our available implementations that they produce automata less interesting than Safra's complementation. But implementing these algorithm was still interesting. We can now add some refinements to these approaches to try to decrease the size of the resulting automata. Moreover, since SPOT is a library that aims to provide algorithms that can be useful in model-checking, providing several implementation of a complementation algorithm is interesting, since an user may choose to use this complementation for a specific problem.

4.2 Perspectives

Not all the algorithms presented in Chapter 2 have been implemented in SPOT. Implementing missing algorithms to have a complementation as least as efficient as Safra's complementation is the future work that we have to do on SPOT.

4.2.1 Implementing the translation from weak alternating automata to non-deterministic Büchi automata

We have to implement the translation from weak alternating automata to non-deterministic Büchi automata presented by Miyano and Hayashi (1984) to have a complementation procedure via alternating automata working. Once this algorithm will be working, we will have a complementation that produces $2^{O(n \log n)}$ states for an initial Büchi automaton with n states.

Then, we need to add techniques to reduce the size of the intermediate weak alternating automaton.

4.2.2 Simulations in SPOT

To reduce the size of the alternating automaton a technique is to perform some simulation on the automaton. SPOT doesn't provide any routines that works to do this operation. Adding simulations in SPOT is one of the future operation we would like to provide.

4.2.3 Merging everything

Once we will have some reduction techniques to reduce the size of the alternating automaton, we will have to merge all the algorithms that we have implemented. Then we will use generalized Büchi automata with n states and k acceptance conditions as input, we will convert these automata in weak alternating automata and will reduce these automata. It will produce non-deterministic Büchi automata with $2^{O(k \cdot n \log k \cdot n)}$ states, which is supposed to be optimal. However, as it was presented in Section 3.4 the $O()$ notation may hide the fact that another algorithm with the same complexity can be better.

Chapter 5

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. 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.

Gurumurthy, S., Kupferman, O., Somenzi, F., and Vardi, M. Y. (2003). On complementing nondeterminisitic Büchi automata. In *Proceedings of the 12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'03)*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer-Verlag.

Kupferman, O. and Vardi, M. (2005). From complementation to certification. *Theoretical Computer Science*, 345(1):83–100.

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. (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.

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

Miyano, S. and Hayashi, T. (1984). Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330.

Muller, D. E., Saoudi, A., and Shupp, P. E. (1986). Alternating automata, the weak monadic theory of the tree and its complexiy. In Kott, L., editor, *Proceedings 13th of the International Colloquium on Automata, Languages and Programming (ICALP'86)*, volume 226 of *Lecture Notes in Computer Science*, pages 233–244. Springer.

Sadegh, G. (2009). Complementing Büchi automata. Technical report, EPITA Research and Development Laboratory (LRDE).

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.

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.

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.