A Co-Büching Toolbox

Alexandre Gbaguidi Aïsse
(supervisor: Alexandre Duret-Lutz)

Technical Report n^o1705, May 2017
revision 94960218

Thanks to the HOA format there are less and less barriers between ω-automata with different acceptance conditions. Spot, a library for ω-automata manipulation, supports arbitrary acceptance condition and performs some conversions between them. We augment the set of supported acceptance conversions with new algorithms that convert (when possible) all common classes of non deterministic ω-automata to deterministic co-Büchi. Boker and Kupferman (2009, 2011) solve this conversion whith asymptotically tight constructions. We implement their methods with some optimizations and adjustments: we made it work with Streett-like acceptance, any acceptance in disjunctive normal form (including Rabin-like) and also transition-based acceptance. Using these conversions gives a new way to check whether an LTL formula is a persistence formula.

Grâce au format HOA, il y a de moins en moins de barrières entre des ω-automates avec différentes conditions d'acceptation. Spot, une bibliothèque manipulant les ω-automates supporte des conditions d'acceptation arbitraires et propose déjà quelques conversions entre elles. Nous rajoutons de nouveaux algorithmes de conversions capables de convertir (quand c'est possible) n'importe quelle condition d'acceptation classique vers du co-Büchi déterministe. Boker and Kupferman (2009, 2011) ont mis au point des méthodes asymptotiquement optimales réalisant ces conversions. Nous avons implémenté leurs méthodes avec quelques optimisations et adaptations: nous supportons les conditions d'acceptation dites "Streett-like", celles écrite sous forme normale disjonctive (y compris celles dites "Rabin-like") ainsi que les acceptations basées sur les transitions ("transition-based"). Ces algorithmes peuvent être utilisés pour vérifier si une formule LTL est de la famille "persistence".

Keywords
ω-automata, acceptance condition, conversion, co-Büchi, Streett-like, Rabin-like, Büchi, Parity.

Laboratoire de Recherche et Développement de l'EPITA
14-16, rue Voltaire – FR-94276 Le Kremlin-Bicêtre CEDEX – France
Tél. +33 1 53 14 59 22 – Fax. +33 1 53 14 59 13
aga@lrde.epita.fr – spot.lrde.epita.fr

Copying this document

Contents

Chapter 1

Introduction

When introduced, ω-automata where used to solve fundamental decision problems in mathematics and logics (Büchi, 1990). Today they are widely used in formal verification and system synthesis. They are a good way to represent systems which runs infinitely, express their behaviour using an acceptance condition and verify some properties on it. Usually, systems are deduced from a Linear Temporal Logic (LTL) (Pnueli, 1977) formula. It is a modal logic that extends propositional logic with temporal operators. ω-automata play a key role in LTL model checking as the automata-theoretic approach to verification of specifications reduces questions about systems to problem like emptiness check (Kurshan, 1994; Vardi, 2007). Over years automata have been the subject of significant research. People started to be interested in the complexity of problems and constructions involving automata.

Several formats have been used to represent ω-automata, some of them were restricted to a few specific acceptance conditions, others were made for a specific tool. Most of them were using common names given to classical acceptance conditions, to define them. The problem with this approach is that they can not support arbitrary acceptance conditions. The authors of some tools that manipulates ω-automata have made a concerted effort to deliver a unified format called HOA (Babiak et al., 2015). Their purpose is to break down the barriers between them. The HOA format introduces a way to describe automata and acceptance conditions as formula rather than names. Note that Emerson and Lei (1987) were the first to define a way to describe automata as formula with temporal operators. Spot (Duret-Lutz et al., 2016), a C++ library manipulating automata, supports the HOA format and works on automata with arbitrary acceptance conditions. We use acceptance conditions to classify ω-automata. Some classes have more expressivity than others, some classes are more easier to handle or have more achieved results. An important challenge for a better understanding of those classes is to provide algorithms which help to convert each of the different classes towards the others.

For most conversions, efficient methods have been developped. Efficient in the sense that the state blow-up involved in each conversion has been proved to be unavoidable. This is not the case of all conversions between classes, some of them still have a serious gap between the upper and the lower bound. This was the case of the long-standing problem of the conversion when possible of a nondeterministic automaton with a Büchi acceptance (NBA) to a nondeterministic automaton with a co-Büchi acceptance (NCA) or a deterministic automaton with a co-Büchi acceptance (DCA). "When possible" because co-Büchi automa, deterministic or not are less expressive than nondeterministic Büchi automata. This is probably the main reason why such a conversion is tenacious. However, they are as expressive as nondeterministic Büchi automata but we still do not known how to take advantage of the allowed nondeterminism.

Until 2009, the best known conversion of an NBA to an NCA was to go through an intermediate deterministic Streett (Safra, 1988) or parity (Piterman, 2007) automaton. Starting with an NBA of n states, we end up with a DCA with $2^{\mathcal{O}(n \log n)}$ states. Boker and Kupferman (2009) solve these problems with aymptotically tight constructions. They suggest one method to convert an NBA with n states to an NCA with $n2^n$ states and another one to convert an NBA with n states to a DCA with $2^{\mathcal{O}(n)}$ states using the well known breakpoint construction of Miyano and Hayashi (1984). Another problem is the conversion of the other common classes (Streett, Rabin, Parity) to NCA and DCA. The major known but not optimal approaches goes all under an intermediate automaton (for instance Streett or Parity) involving a state blow-up (Krishnan et al., 1994). Boker and Kupferman (2011) also solve this problem. They suggest a method to convert a Streett or parity automaton with n states to an NCA with $n2^n$ states and to a DCA with 3^n states. They also developped a method to convert a Rabin automaton with k clauses to an intermediate Streett automaton with nk states, then an NCA of $nk2^n$ states and a DCA of $k3^n$ states. In this report, we introduce our implementation of the methods of Boker and Kupferman (2009, 2011). We made some improvments and adjustments, the methods have been extended to any Streett-like acceptance and to any acceptance in DNF. Moreover we also supports transition-based and state-based representations. The existing conversions in Spot as well as my contributions are shown in Figure 1.1. This figure shows the different classical acceptance conditions with the way they are included in each other (represented by full edges). It also shows all conversions supported by Spot (with dotted edges and the name of the algorithm). When a conversion concerns a class included by another one (full edge), the name of the algorithm is simply added on the edge.

An automaton that express an LTL formula may tell some information about that formula. Manna and Pnueli (1990) define a temporal hierarchy that relates LTL properties to structural properties of automata. Currently Spot has some processing chains to test the membership of an LTL formula to each of the classes. The implemented methods give a new way to test if a formula is in persistence class. This comes from an important property of the constructions. When the language of the starting automaton is not expressible with a co-Büchi acceptance, the resulting automaton recognizes a superset of the same language. This is interesting since it is known that LTL formulas belonging to the persistence class can always be expressed with a co-Büchi automaton. Therefore, testing if a formula belongs to the persistence class is reduced to performing the conversion of Boker and Kupferman (2011) and testing the inclusion in the other way.

After introducing in chapter 2 some useful definitions for a better understanding of the report, we present the conversions to NCA with the adjustments that we made, the conversions to DCA and some benchmark results.

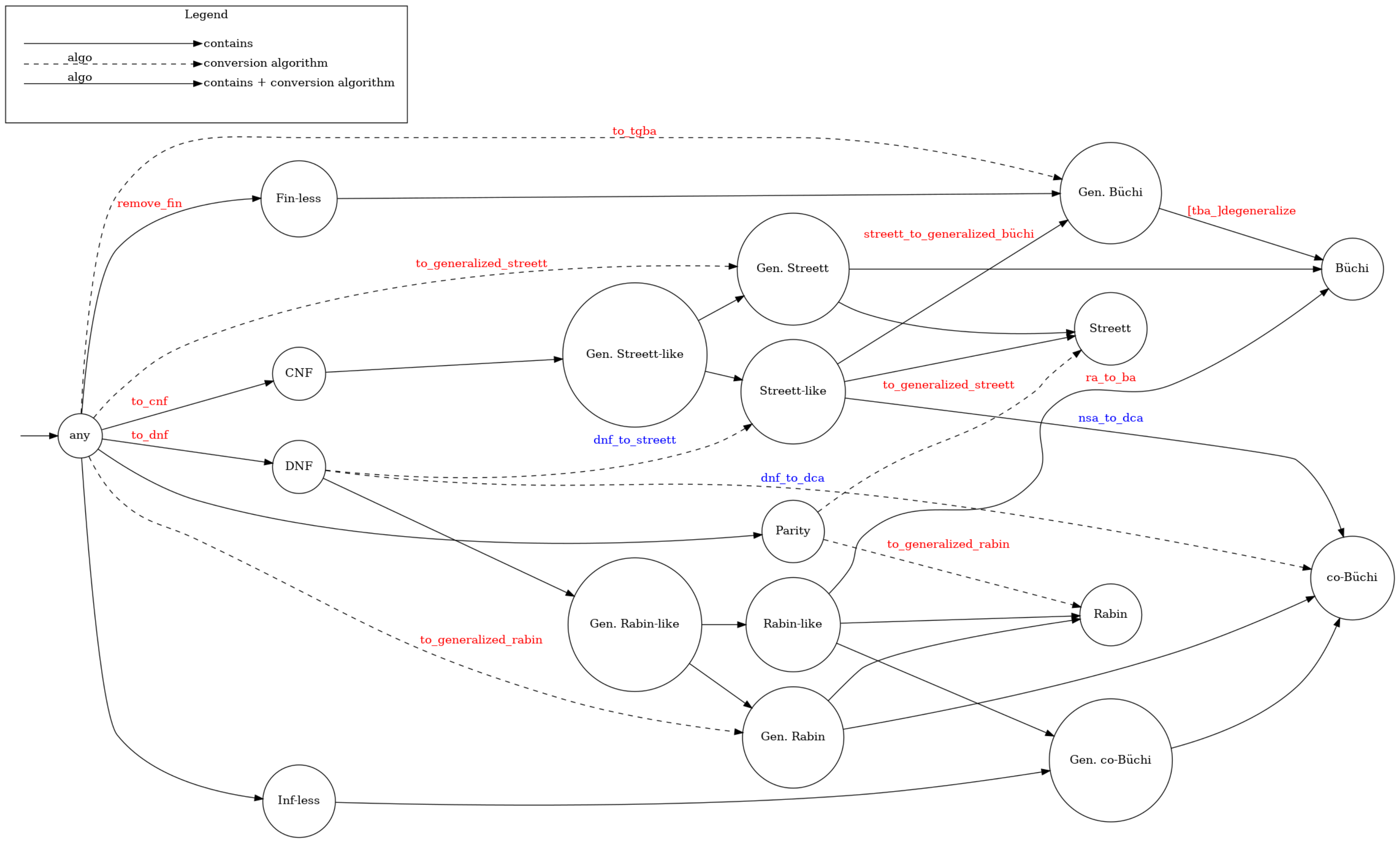


Figure 1.1 – Traditional acceptances and their inclusions. The conversions already supported by Spot (Duret-Lutz et al., 2016) are coloured in red. My contributions are shown in blue.

Chapter 2

Basic concepts of automata theory

This chapter[1] introduces the necessary concepts and definitions for a better understanding of the report.

Definition 2.1 (Transition-based semi-automaton) *A* transition-based semi-automaton *is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta \rangle$ where*

- *Σ is a finite alphabet,*
- *Q is a finite set of states,*
- *$q_0 \in Q$ is the initial state,*
- *$n \in \mathbb{N}$ is the number of acceptance marks and $[n] = \{0, 1, 2, \dots, n-1\}$ is the set of acceptance marks,*
- *$\Delta \subseteq Q \times \Sigma \times 2^{[n]} \times Q$ is a transition relation labeling each transition by a letter. We denote a transition by $\delta = (\delta^s, \delta^\ell, \delta^M, \delta^d) \in \Delta$ where δ^s is its source state, δ^ℓ its label, δ^M its set of acceptance marks and δ^d its destination state. When having a set of transitions $\Delta_x \subseteq \Delta$, by extension, $\Delta_x^M = \cup_{\delta \in \Delta_x} \delta^M$.*

A sequence of transitions $\rho = (\delta_0^s, \delta_0^\ell, \delta_0^M, \delta_0^d)(\delta_1^s, \delta_1^\ell, \delta_1^M, \delta_1^d) \ldots \in \delta^\omega$ is a run *of A if $\delta_0^s = q_0$ and for all $i \geq 0$ we have $\delta_i^d = \delta_{i+1}^s$. We say that ρ* recognizes *the word $w = \delta_0^\ell \delta_1^\ell \ldots \in \Sigma^\omega$. Also by extension, ρ^M is the set of acceptance marks visited infinitely often by ρ.*

In the previous definition, acceptance marks of the semi-automaton are set on transitions, this is why it is said to be *transition-based*. This is not the only way to set the acceptance marks, let us explain what a *state-based* semi-automaton is.

Definition 2.2 (State-based) *Formally, a semi-automaton is said to be state-based if the acceptance marks are set on states instead of transitions for a transition-based system. Actually it can be seen differently: to simulate a mark on a State, Spot (Duret-Lutz et al., 2016) marks all outgoing transitions of this state. Doing so allows us to interpret an automaton state-based acceptance as if it was an automaton with transition-based acceptance.*

[1] This chapter is inspired from a collaborative work of the students working on Spot (Duret-Lutz et al., 2016).

Definition 2.3 (Determinism) *An* ω-automaton *is deterministic iff for each pair of state and symbol* $(Q \times \Sigma)$ *there is at most one outgoing edge. That is:*

$$\forall\,(\delta_i,\ \delta_j) \in \Delta^2,\ (\delta_i^s = \delta_j^s \wedge \delta_i^\ell = \delta_j^\ell) \implies \delta_i^d = \delta_j^d$$

Until now, only semi-automaton has been defined. In order to explain what an ω-automaton is, the definition of *acceptance condition* is needed.

Definition 2.4 (Acceptance condition) *An* acceptance condition *(Babiak et al., 2015; Emerson and Lei, 1987)* ϕ *of an automaton* $\mathcal{A}$ *is a Boolean formula which respects the following grammar:*

$$\phi := t \mid f \mid \mathsf{Inf}(x) \mid \mathsf{Fin}(x) \mid \phi \wedge \phi \mid \phi \vee \phi \mid (\phi)$$

We take $\rho \vDash \phi$ *to mean the path* ρ *satisfies* ϕ *and* $\rho \nvDash \phi$ *to mean* ρ *does not satisfies* ϕ*. The satisfaction of an acceptance condition is interpreted over a run* ρ *by induction as follows:*

$$\begin{array}{lllll}
\rho & \vDash & t & & \\
\rho & \nvDash & f & & \\
\rho & \vDash & \mathsf{Inf}(x) & \iff & x \in \rho^M \\
\rho & \vDash & \mathsf{Fin}(x) & \iff & x \notin \rho^M \\
\rho & \vDash & \phi_1 \wedge \phi_2 & \iff & \rho \vDash \phi_1 \textit{ and } \rho \vDash \phi_2 \\
\rho & \vDash & \phi_1 \vee \phi_2 & \iff & \rho \vDash \phi_1 \textit{ or } \rho \vDash \phi_2
\end{array}$$

Definition 2.5 (Transition-based ω-automaton) *A* transition-based ω-automaton *is a transition-based semi-automaton with an acceptance condition. It is a tuple* $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \phi \rangle$ *where* $\Sigma, Q, q_0, n, \Delta$ *are defined exactly as for a semi-automaton and* ϕ *is the acceptance condition.*

A fundamental notion has not been introduced yet: the language of an automaton. Actually, the emptiness check problem is to answer whether an automaton has an empty language or not. Let us define the language of an automaton.

Definition 2.6 (Language) *For a run* $\rho = (\delta_0^s, \delta_0^\ell, \delta_0^M, \delta_0^d)(\delta_1^s, \delta_1^\ell, \delta_1^M, \delta_1^d) \ldots \in \delta^\omega$ *of the automaton* $\mathcal{A}$*, let* $\rho^\ell = \delta_0^\ell \delta_1^\ell \ldots \delta_\omega^\ell$ *be the word constructed by the run. The* language *of* $\mathcal{A}$*, denoted* $\mathscr{L}(\mathcal{A})$*, is the set of infinite words recognized by the accepting runs of* $\mathcal{A}$*.*

$$\mathscr{L}(\mathcal{A}) = \{\rho_i^\ell \mid \rho_i \vDash \phi\}$$

Definition 2.7 (CNF - DNF) *A conjunctive normal form (CNF) is a conjunction of clauses, where a clause is a disjunction. In other words it is an AND of ORs. On the contrary, a disjunctive normal form (DNF) is a disjunction of conjunctive clauses, an OR of ANDs.*

Acceptance conditions can be written in CNF or DNF. For instance, the following acceptance conditions are in CNF:

- Fin(⓿) $\wedge$ (Fin(❶) $\vee$ Inf(❷))
- (Inf(⓿) $\vee$ Inf(❶)) $\wedge$ (Fin(❷) $\vee$ Inf(❸) $\vee$ Fin(⓿)) $\wedge$ Inf(❶)

Over the history, some classical acceptance conditions have been studied. Let us define some of them.

Definition 2.8 (Büchi acceptance) *A transition-based automaton with a Büchi acceptance is written as* $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \mathsf{Inf}(⓿) \rangle$*. A run* ρ *of* $\mathcal{A}$ *is accepting iff* ρ *visits infinitely often* ⓿*. That is* $\rho^M = \{$⓿$\}$ *as card*$(n) = 1$.

The Figure 2.1 shows an automaton with a Büchi acceptance. The acceptance mark is highlighted by the bullet ⓪. Along this report, all acceptance marks will be shown with almost the same style (it may have a different number and a different color).

Definition 2.9 (co-Büchi acceptance) *A transition-based automaton with a co-Büchi acceptance is written as* $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \mathsf{Fin}(⓪) \rangle$*. A run* ρ *of* $\mathcal{A}$ *is accepting iff* ρ *visits finitely often* ⓪*. That is* $\rho^M = \emptyset$ *as card*$(n) = 1$.

Definition 2.10 (Streett acceptance) *A transition-based automaton with a Streett acceptance is written as* $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \phi \rangle$ *where*

$$\phi = \bigwedge_{i=1}^{k} \phi_i, \ \phi_i = \mathsf{Fin}(a_i) \vee \mathsf{Inf}(b_i)$$

Note that a Streett acceptance is in conjunctive normal form, $(a_i, b_i) \in [n]^2$ *and* $k \in \mathbb{N}$ *is the number of clause. A run* ρ *of* $\mathcal{A}$ *is accepting iff for each clause* ϕ_i*,* ρ *visits finitely often* a_i *or infinitely often* b_i*. The Figure 2.2 shows an automaton having a Streett acceptance with* $k = 2$.

In this report, *parity acceptances* (Xu, 2016) are not directly manipulated as any parity acceptance can be converted[2] to a Streett one keeping the structure of the associated automaton. Each time *Streett* is use, it also holds for *parity*. The Figure 2.3 shows an automaton with a parity acceptance condition.

Definition 2.11 (Rabin acceptance) *A transition-based automaton with a Rabin acceptance is written as* $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \phi \rangle$ *where*

$$\phi = \bigvee_{i=1}^{k} \mathsf{Fin}(a_i) \wedge \mathsf{Inf}(b_i)$$

Also note that a Rabin acceptance is in disjunctive normal form, $(a_i, b_i) \in [n]^2$ *and* $k \in \mathbb{N}$ *is the number of clause. A run* ρ *of* $\mathcal{A}$ *is accepting iff for at least one clause* ϕ_i*,* ρ *visits finitely often* a_i *and infinitely often* b_i*.*

Definition 2.12 (Streett-like acceptance) *A* Streett-like acceptance *is a conjunction of* Streett, Büchi *and* co-Büchi. *Formally,*

$$\phi = \bigwedge_{i=1}^{k} \phi_i \qquad \text{with } \phi_i = \begin{cases} \mathsf{Fin}(a), & a \in [n] \\ \mathsf{Inf}(b), & b \in [n] \\ \mathsf{Fin}(a) \vee \mathsf{Inf}(b), & (a,b) \in [n]^2 \end{cases}$$

Actually, a Streett-like acceptance includes all Streett acceptances and tolerates clauses with a missing Fin *or a missing* Inf*. Therefore, a Büchi acceptance is Streett-like and considered as is in this report. A co-Büchi acceptance is also Streett-like. The definition of an accepting run of such acceptance can be easily deduced from those of Streett, Büchi and co-Büchi.*

The Figure 2.4 shows an automaton with a Streett-like acceptance condition. Note that a Streett-like acceptance is in CNF.

[2] The idea is to put the acceptance condition in conjunctive normal form and then merge all the $\mathsf{Inf}(ⓐ) \vee \mathsf{Inf}(ⓑ) \vee \cdots$ that may occur in clauses into a single $\mathsf{Inf}(ⓧ)$.

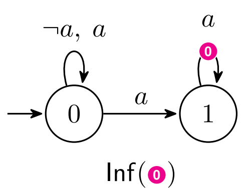


Figure 2.1 – An automaton with a Büchi acceptance

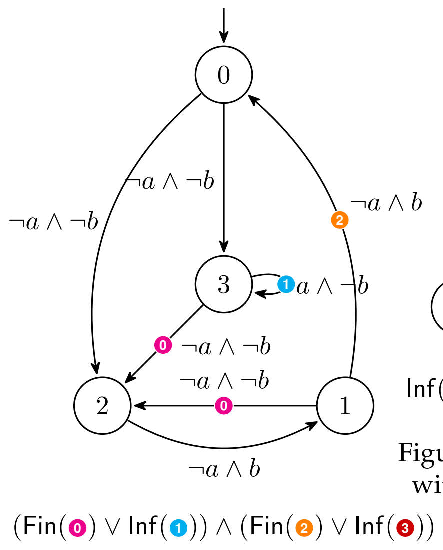


Figure 2.2 – An automaton with a Streett acceptance

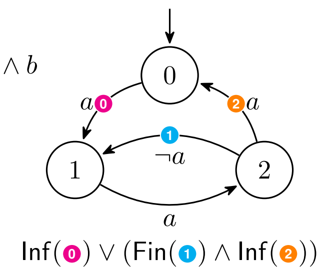


Figure 2.3 – An automaton with a Parity acceptance

Definition 2.13 (Rabin-like acceptance) *A* Rabin-like acceptance *is of the form*

$$\phi = \bigvee_{i=1}^{k} \phi_i \qquad \text{with } \phi_i = \begin{cases} \mathsf{Fin}(a), & a \in [n] \\ \mathsf{Inf}(b), & b \in [n] \\ \mathsf{Fin}(a) \wedge \mathsf{Inf}(b), & (a,b) \in [n]^2 \end{cases}$$

Compare to Streett-like, the only difference is that Rabin-like includes Rabin acceptances instead of Streett ones. A Rabin-like acceptance is in DNF.

Definition 2.14 (SCC) *A strongly connected component $C \subseteq Q$ of an automaton $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \phi \rangle$ is a non-empty set of states such that any ordered pair of states of C can be connected by a sequence of transitions. The set of transitions induced by C is denoted by $C_\delta = \{\delta = (\delta^s, \delta^\ell, \delta^M, \delta^d) \mid (s,d) \in Q^2\}$. Also let $C_\Delta = \Delta \cap \{C \times \Sigma \times 2^{[n]} \times C\}$ be the set containing all transitions in C and C_Δ^M be the set of marks appearing in C. An SCC is accepting if it contains an accepting cycle. A cycle c is a sequencing of C_Δ transitions and is accepting if there exists a run ρ following c such that $\rho \vDash \phi$. For $\delta \in \Delta$, we denote by $\mathrm{scc}(\delta) \subseteq Q$, the SCC which contains δ , that is $\delta \in \mathrm{scc}(\delta)_\Delta$.*

The Figure 2.4 shows an automaton with a Rabin acceptance. It has two accepting SCCs (area in green). From here to the end of this report, accepting SCC will be highlighted with a green color and rejecting one with a red color.

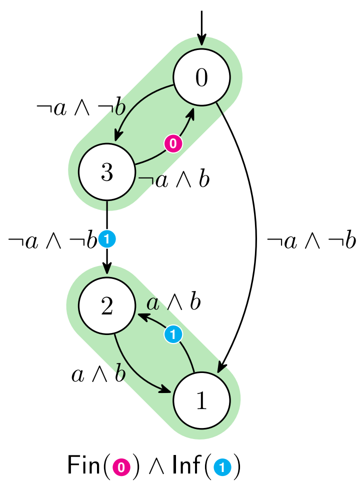


Figure 2.4 – An automaton with a Rabin acceptance and two accepting SCCs

Chapter 3

Conversion to nondeterministic co-Büchi automaton

This chapter explains how an automaton $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \phi \rangle$ where ϕ is either Streett-like or Rabin-like, can be converted to a co-Büchi automaton $\mathcal{B} = \langle \Sigma, Q_B, q_{0B}, n_B, \Delta_B, \phi_B \rangle$. Have in mind that with a co-Büchi acceptance $\mathsf{Fin}(⓿)$ we can only mark transitions that we want to exclude from accepting cycles. Therefore, our goal is to find transitions that can not be part of accepting cycles.

3.1 From Streett-like to co-Büchi

Boker and Kupferman (2009, section 3.1) introduced a first algorithm which converts a Büchi automaton to a co-Büchi automaton. Later, Boker and Kupferman (2011, section 3) generalized this method to work with an automaton with a Street acceptance condition.

This conversion is performed on top of the augmented subset construction described in Section 3.1.2. Let $\mathcal{A}' = \langle \Sigma, Q', q_0', n, \Delta' \rangle$ be the augmented subset consruction of $\mathcal{A}$. This is already the final structure of $\mathcal{B}$ but transitions that must be seen finitely often still need to be marked in order to have a co-Büchi acceptance. A transition that belongs to an accepting cycle of $\mathcal{A}'$ must be seen infinitely often. Such cycles are less evident to found with a Streett acceptance than a Büchi acceptance.

The implemented algorithm does not distinguish Büchi from Streett but for a deep understanding let us treat both cases separately in Section 3.1.3 and Section 3.1.4.

3.1.1 Subset construction

The subset construction, also known as powerset construction is a standard method for converting a nondeterministic finite automaton (NFA) into a deterministic automaton (DFA) which recognizes the same language. The subset construction of an automaton $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \phi \rangle$ is the semi automaton $\mathcal{A}_{\text{pow}} = \langle \Sigma, 2^Q, \{q_0\}, n, \Delta_{\text{pow}} \rangle$ where Δ_{pow} is constructed as follows: the transition relation of the resulting automaton maps a set of states $S \subseteq Q$ and a letter ℓ to the set of all states that can be reached from any state of S with the letter l in the starting automaton.

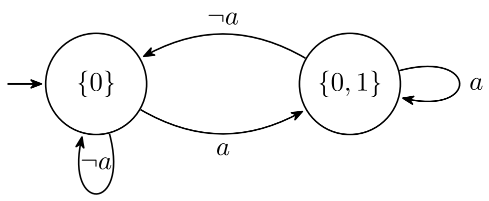


Figure 3.1 – Subset construction of the automaton of Figure 2.1

That is the set

$$\bigcup \left\{ \delta_i^d \mid \delta_i \in S \times \{\ell\} \times 2^{[n]} \times Q \text{ and } S \times \{\ell\} \times 2^{[n]} \times Q \subseteq \Delta \right\}$$

An example is shown in the Figure 3.1. While this method works fine for determinizing automata with finite run, it can not be used to determinize an ω-automaton. This is why all marks have been skipped. If the mark ⓿ was kept, it would have been put on the edge looping on $\{0, 1\}$. The problem is that by looping on it, we would accept runs that never left the state 0 of the starting automaton, which is wrong. The augmented subset construction enforces this.

3.1.2 Augmented subset construction

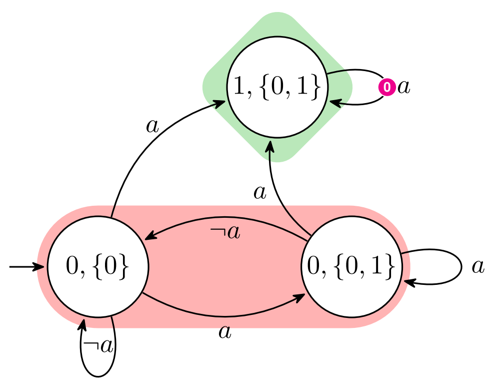
Let $\mathcal{A}_{\text{pow}} = \langle \Sigma, 2^Q, \{q_0\}, n, \Delta_{\text{pow}} \rangle$ be the subset construction of $\mathcal{A}$. Let us introduce a notation: If a transition $\delta = (\delta^s, \delta^\ell, \delta^M, \delta^d)$ belongs to a set of transition Δ, then $\Delta(\delta^s, \delta^\ell) = \delta^d$. It denotes the destination of a transition knowing its source and label. The augmented subset construction $\mathcal{A}'$ of $\mathcal{A}$ is the semi-automaton resulting from the product of $\mathcal{A}$ with $\mathcal{A}_{\text{pow}}$. Formally, $\mathcal{A}' = \langle \Sigma, Q', q'_0, n, \Delta' \rangle$ where:

- $Q' = Q \times 2^Q$ which means that all states of $\mathcal{A}'$ can be written as pairs $\langle q, E \rangle$ where $q \in Q$, $E \in 2^Q$ and $E \subseteq Q$,
- $q'_o = q_o \times \{q_o\}$,
- $\forall \langle q, E \rangle \in Q'$ and $\ell \in \Sigma$, $\Delta'(\langle q, E \rangle, \ell) = \Delta(q, \ell) \times \{\Delta_{\text{pow}}(E, \ell)\}$.

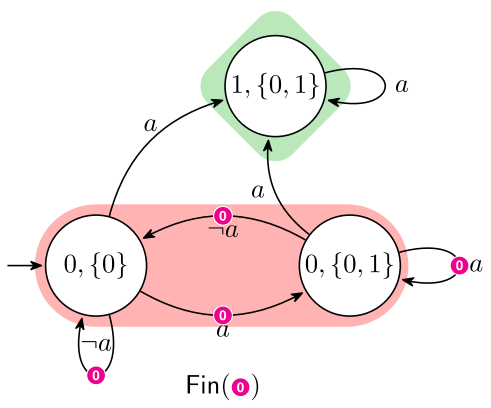
An example of this algorithm is shown in Figure 3.2a. Actually the augmented subset construction knows all state in which it could have been while recognizing the same word. In addition it keeps track of the followed path.

3.1.3 Büchi to co-Büchi

The principle of the augmented subset construction is now known. As a reminder, we said that an automaton resulting from this construction has already the structure of the nondeterministic co-Büchi automaton. We are going to explain how to change the acceptance marks in case the input automata have a Büchi acceptance Inf(⓿). A transition must be seen finitely often if it does not belong to an accepting SCC. In our case, once an SCC has the mark ⓿, there will always be an accepting cycle c visiting ⓿ infinitely often and making this SCC accepting. Depending on the presence of ⓿ in an SCC, all its transitions must be seen either finitely or infinitely often.



(a) Augmented subset construction of the Büchi automaton of Figure 2.1. Note that for a better understanding, the acceptance mark has been kept where it belongs to be as well as rejecting or accepting SCCs.



(b) The corresponding Büchi automaton

Figure 3.2 – Construction of Section 3.1.3 applied on the automaton of Figure 2.1.

$\mathcal{B} = \langle \Sigma, Q', q_0', 1, \Delta_B, \mathsf{Fin}(\textcircled{0})\rangle$ is constructed from $\mathcal{A}' = \langle \Sigma, Q', q_0', n, \Delta'\rangle$ with

$$\Delta_B = \left\{ \delta \in Q' \times \Sigma \times \{\emptyset, 0\} \times Q' \mid \exists\, \delta_i \in \Delta' \mid \delta = (\delta_i^s, \delta_i^\ell, \delta^M, \delta_i^d) \text{ where } \delta^M = \begin{cases} \emptyset, & \text{if } \mathsf{scc}(\delta_i)_\Delta^M \neq \emptyset \\ \{0\}\,, & \text{else} \end{cases} \right\}$$

An example of this construction is shown in Figure 3.2.

3.1.4 Streett to co-Büchi

Finding accepting cycles in order to know which transition must be seen infinitely or finitely often with a Streett acceptance is less obvious than with Büchi acceptance. In the case of a Streett acceptance, even if an SCC is accepting all its transitions may not be accepting. For an acceptance condition having the Streett form

$$\phi = \bigwedge_{i=1}^{k} (\mathsf{Fin}(a_i) \vee \mathsf{Inf}(b_i)) \text{ where } (a_i, b_i) \in 2^{[n]}\,,$$

a cycle c is accepting if for all i, c avoids a_i or visits b_i. If for a precise i, a_i and b_i are in the same SCC, we are garanteed to always find a cycle visiting b_i infinitely often. The problem is if there is just a_i and not b_i, we do not know if there is still an accepting cycle avoiding a_i. The idea in this case is to temporaly remove all transitions having this mark to force their avoidance and then see if the SCC is still accepting.

$\mathcal{B} = \langle \Sigma, Q', q_0', 1, \Delta_B, \mathsf{Fin}(\textcircled{0})\rangle$ is constructed from $\mathcal{A}' = \langle \Sigma, Q', q_0', n, \Delta'\rangle$ as follows:

$$\forall\, \delta_i \in \Delta', \quad \delta_j = (\delta_i^s, \delta_i^\ell, \delta_j^M, \delta_i^d) \in \Delta_B \text{ where } \delta_j^M = \begin{cases} \emptyset, & \text{if } \delta_i \in f_{\text{check}}(\mathsf{scc}(\delta_i)) \\ \{0\}\,, & \text{else} \end{cases}$$

and

- f_{check} returns all the accepting transitions of an SCC,

$$\begin{aligned} f_{\text{check}}: \quad & 2^Q \to 2^\Delta \\ C \mapsto & \begin{cases} C_\Delta & \text{if } f_{\text{split\_on}}(C) = \emptyset \\ \displaystyle\bigcup_{C_i \in f_{\text{split}}(C, f_{\text{split\_on}}(C))} f_{\text{check}}(C_i) & \text{else} \end{cases} \end{aligned}$$

- $f_{\text{split\_on}}$ returns all the marks M on which we wish to split the SCC with regards to the acceptance condition ϕ

$$\begin{aligned} f_{\text{split\_on}}: \quad & 2^Q \to 2^{[n]} \\ & C \mapsto \left\{a_i \in C_\Delta^M \mid b_i \notin C_\Delta^M\right\} \end{aligned}$$

- f_{split} returns the possibly news SCCs after deleting some transitions having the specified marks

$$\begin{aligned} f_{\text{split}}: \quad & 2^Q \times 2^{[n]} \to 2^{2^Q} \\ & (C, M) \mapsto \{\text{scc}(\delta) \mid \delta \in f_{\text{rem}}(C, M)\} \end{aligned}$$

- f_{rem} either deletes transitions only having the specified marks or remove the specified marks from transitions having a superset of that marks

$$\begin{aligned} f_{\text{rem}}: \quad & 2^Q \times 2^{[n]} \to 2^\Delta \\ & (C, M) \mapsto \left\{(\delta^s, \delta^\ell, \delta^M \backslash M, \delta^d) \mid \delta \in C_\Delta \wedge \delta^M \backslash M \neq \emptyset\right\} \end{aligned}$$

3.1.5 Generalization

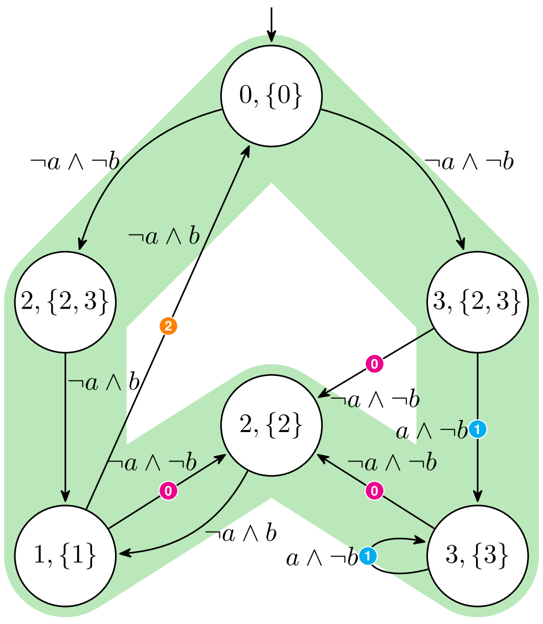
The implemented algorithm works not only for both Büchi and Streett acceptances but also for any Streett-like acceptance. As shown in Chapter 2 , any clause ϕ_i of a Streett-like acceptance ϕ, is

- either $\mathsf{Fin}(a)$, $a \in [n]$: In which case each time a appears in an SCC, we always want to temporaly remove all transitions having this mark.
- or $\mathsf{Inf}(b)$, $b \in [n]$ which does not rise any problem except that it must always appears in each accepting SCC candidate otherwise the SCC is rejecting, just as for Büchi acceptances.
- or $\mathsf{Fin}(a) \vee \mathsf{Fin}(b)$ $(a, b) \in [n]^2$ which is a formal Streett clause and is treated as such (Section 3.1.4).

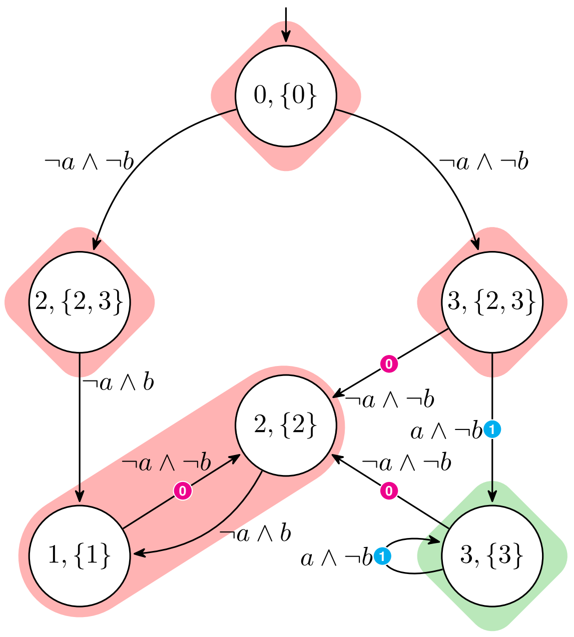
Furthermore, If the starting automaton was transition-based, we keep it as such by avoiding to set marks on transitions leaving the SCC being processed. This is a small adjustement of the original description of Boker and Kupferman (2011).

3.2 From any DNF acceptance to co-Büchi

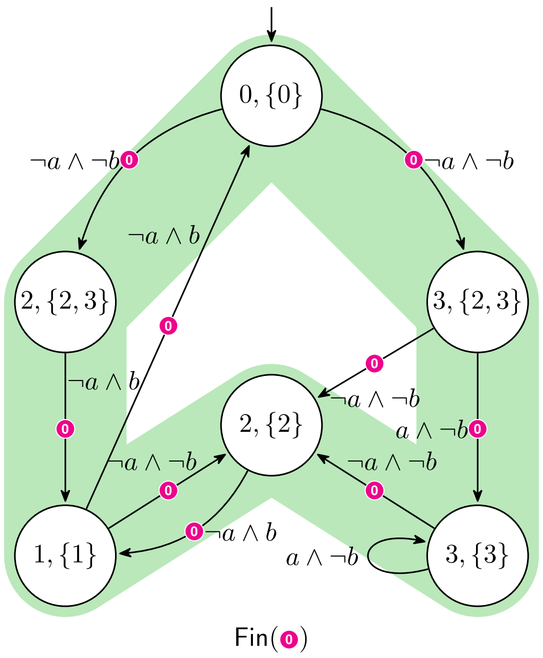
The Section 3.1 showed how to convert any Streett-like acceptance to co-Büchi. Here, we are going to deal with any DNF acceptance. Boker and Kupferman (2011) suggest a method to convert any Rabin acceptance into Streett and then use the algorithm introduced in the previous section. We extended that method to work with any DNF acceptance. Let us first explain their method.



(a) Augmented subset construction of the automaton of Figure 2.2. Note that for a better understanding, acceptance marks as well as rejecting or accepting SCCs have been kept.



(b) Step1: Deletion of all transitions having the acceptance mark 2. This splits the SCC into 5 SCCs. We now know that all transitions of the 4 rejecting SCCs must be seen finitely often.



(c) Updating acceptance marks of all transitions according to Step 1 and set the co-Büchi acceptance condition.

Figure 3.3 – Construction of Section 3.1.4 applied on the automaton of Figure 2.2.

3.2.1 Intermediate Streett automaton

We are going to present a conversion of a Rabin automaton $\mathcal{A} = \langle \Sigma, Q, q_0, n, \Delta, \phi \rangle$ where ϕ has k clauses to an automaton $\mathcal{C}$ with a Streett acceptance. Let us first have an intuition of what this conversion involves. The conversion does not change the structure of the automaton if $k = 1$. Otherwise it copies k times the input automaton and after some processings, returns the union of those copies. The algorithm changes the acceptance condition and adds some acceptance marks. Each clause ϕ_i of ϕ has the form $\mathsf{Fin}(a_i) \wedge \mathsf{Inf}(b_i)$, $(a_i, b_i) \in [n]^2$.

Let us consider that ϕ has only one clause $\mathsf{Fin}(a) \wedge \mathsf{Inf}(b)$. In order to convert it to a Streett acceptance which is a conjunction of disjunctions, the idea is to add a wrong disjunction to $\mathsf{Fin}(a)$ and another one to $\mathsf{Inf}(b)$:

- To $\mathsf{Fin}(a)$ we associate an Inf operator applied on a new acceptance mark $x = n$. To always make $\mathsf{Inf}(x)$ evaluated as false, x must never appears in the automaton. In this way, it will never be seen infinitely often.
- To $\mathsf{Inf}(b)$ we associate a Fin operator applied on a new acceptance mark $y = n + 1$. To always make $\mathsf{Fin}(y)$ evaluated as false, y must appears on every transitions of the automaton. It will always be seen infinitely often.

More formally, $\mathcal{C} = \langle \Sigma, Q, q_0, n + 2, \Delta_\mathcal{C}, \phi_\mathcal{C} \rangle$ where:

$$\phi_\mathcal{C} = (\mathsf{Fin}(a) \vee \mathsf{Inf}(n+1)) \wedge (\mathsf{Fin}(n) \vee \mathsf{Inf}(b))$$
$$\Delta_\mathcal{C} = \left\{ (\delta^s, \delta^\ell, \delta^M \cup \{n+1\}, \delta^d) \mid (\delta^s, \delta^\ell, \delta^M, \delta^d) \in \Delta \right\}$$

An example of the conversion of a Rabin acceptance with one clause is shown in Figure 3.4.

Now, when there is more than one clause in ϕ, the starting automaton is copied as many times as there are clauses. Each copy is associated to one clause so will only contain the marks of that clause. On this basis, each copy is modified just as before, that means just as it is a single Rabin automaton with just one clause in its acceptance condition. The resulting automaton is the union of all those copies.

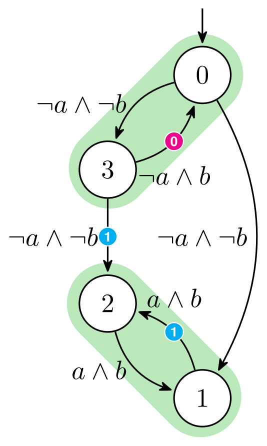
Let k be the number of clause, $\mathcal{C}_i$ the copy of $\mathcal{A}$ associated to the clause ϕ_i and let $\Delta_{\mathcal{C}_i} \subseteq \Delta_\mathcal{C}$ be the set of transitions that belongs to $\mathcal{C}_i$ such that $\delta_{\mathcal{C}_i} \in \Delta_{\mathcal{C}_i}$. More formally, $\mathcal{C} = \langle \Sigma, k \times Q + 1, q', n + k + 1, \Delta_\mathcal{C}, \phi_\mathcal{C} \rangle$ where:

$$\phi_\mathcal{C} = \bigvee_{i=1}^{k} (\mathsf{Fin}(a_i) \vee \mathsf{Inf}(n+k)) \wedge (\mathsf{Fin}(n+i-1) \vee \mathsf{Inf}(b_i))$$
$$\Delta_\mathcal{C} = \left\{ (\delta^s_{\mathcal{C}_i}, \delta^\ell_{\mathcal{C}_i}, \delta^M_{\mathcal{C}_i} \cup \{n+i-1\}, \delta^d_{\mathcal{C}_i}) \mid (\delta^s_{\mathcal{C}_i}, \delta^\ell_{\mathcal{C}_i}, \delta^M_{\mathcal{C}_i}, \delta^d_{\mathcal{C}_i}) \in \Delta_{\mathcal{C}_i} \right\} \cup \left\{ (q', \delta^\ell_{\mathcal{C}_i}, \delta^M_{\mathcal{C}_i}, \delta^d_{\mathcal{C}_i}) \mid (q_{0_{\mathcal{C}_i}}, \delta^\ell_{\mathcal{C}_i}, \delta^M_{\mathcal{C}_i}, \delta^d_{\mathcal{C}_i}) \in \Delta_{\mathcal{C}_i} \right\}$$

Note that because of the union, there is a constraint on all marks $n + i - 1$. They must be used once per copy. Conversly, $n + k$ can be used multiple times as it never appears in the resulting automaton. An example of the conversion of a Rabin acceptance with two clauses is shown in Figure 3.5.

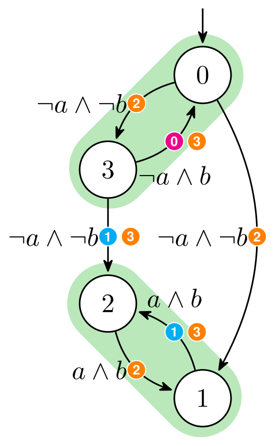
3.2.2 Optimizations

In the Figure 3.5d, it is notable that we end up with two rejecting SCCs that are not essential to preserve the recognized language. What we would like is to connect the initial state 8 to the



Fin(⓪) ∧ Inf(①)

(a) The Rabin automaton of Figure 2.4.



(Fin(⓪) ∨ Inf(❸)) ∧ (Fin(②) ∨ Inf(①))

(b) Add a false disjunction to Inf(①) and Fin(⓪).

Figure 3.4 – Conversion of a Rabin acceptance with one clause to Streett

SCC $\{4, 7\}$ and connect that SCC to $\{1, 2\}$. In this way, we will only have useful SCCs. Instead of making blindness copies of the whole automaton (as many as there are clauses in the acceptance condition), we will work SCC by SCC following this:

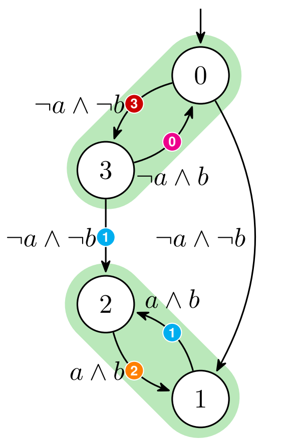
- If the SCC was accepting in the original automaton, it will be copied as many times as there are clauses for which it is accepting.
- If the SCC was rejecting in the original automaton, only one copy of this SCC will be made to remain consistent with the original language.

The optimized algorithm applied on the same automaton of the Figure 3.5d is shown in Figure 3.6.

Also, we found that as long as the acceptance condition is in DNF, the construction still works regardless of the possible combination of Fin and Inf operators. The principle will be the same, add to each operator in the acceptance condition, a disjunction with the corresponding wrong operator.

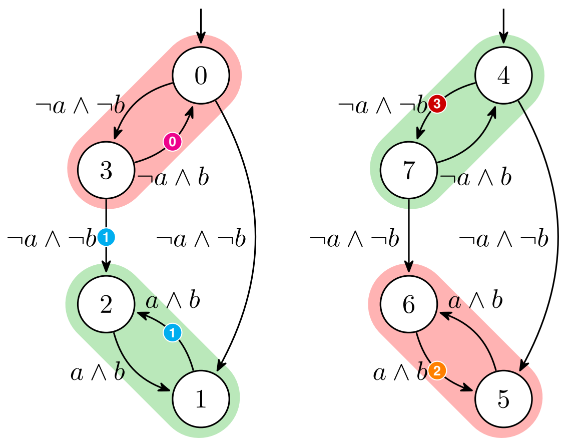
Since the previous algorithm that converts to co-Büchi works with Streett-like acceptance conditions, we do not need to convert DNF acceptance conditions into formal Streett. Therefore, all disjunctions Inf(ⓧ) added are dispensable.

Finally, if the original initial state is not reacheable from itself, it will never be accessible in the resulting automaton. Therefore, we managed to check this at the beginning and avoid creating it in such case. Moreover, if it has only one copy (like in Figure 3.6), we keep the original itinial state.



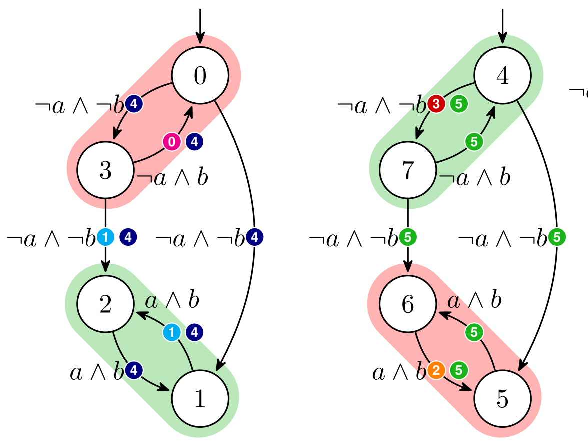
(Fin(⓪) ∧ Inf(①))
∨(Fin(②) ∧ Inf(③))

(a) An automaton with a Rabin acceptance having two clauses.



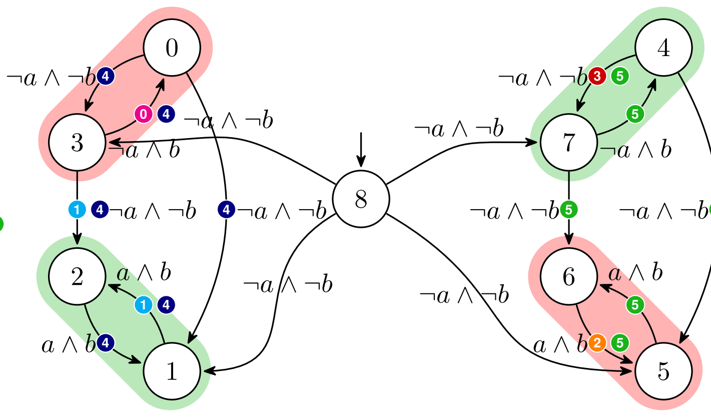
(Fin(⓪) ∧ Inf(①))
∨(Fin(②) ∧ Inf(③))

(b) Step1: There are two copies of the starting automaton as there are two clauses in the acceptance condition. Each copy is associated to one clause and has only its acceptance marks.



((Fin(⓪) ∨ Inf(⑥)) ∧ (Fin(④) ∨ Inf(①)))
∨((Fin(②) ∨ Inf(⑥)) ∧ (Fin(⑤) ∨ Inf(③)))

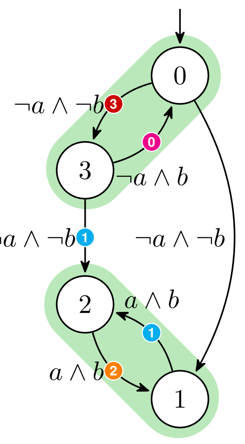
(c) Step2: Add a disjunction to Fin(⓪), Fin(②), Inf(①) and Inf(③).



((Fin(⓪) ∨ Inf(⑥)) ∧ (Fin(④) ∨ Inf(①)))
∧((Fin(②) ∨ Inf(⑥)) ∧ (Fin(⑤) ∨ Inf(③)))

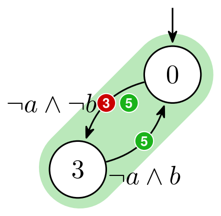
(d) Step3: Union of the copies. The conversion is done.

Figure 3.5 – Conversion of a Rabin acceptance with two clauses to Streett



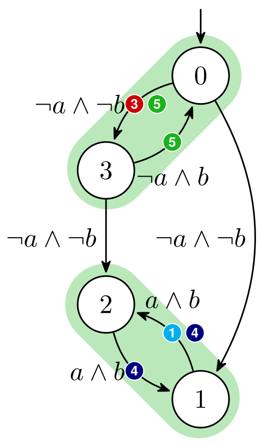
$(\mathsf{Fin}(0) \wedge \mathsf{Inf}(1))$
$\vee(\mathsf{Fin}(2) \wedge \mathsf{Inf}(3))$

(a) An automaton with a Rabin acceptance having two clauses.



$(\mathsf{Fin}(0) \wedge \mathsf{Inf}(1))$
$\vee((\mathsf{Fin}(2) \vee \mathsf{Inf}(6)) \wedge (\mathsf{Fin}(5) \vee \mathsf{Inf}(3)))$

(b) Step 2: Work on the first SCC. Only the second clause makes it accepting, so it is copied one time by keeping only the second clause's marks. Then, acceptance marks are updated as before as well as the second clause.



$((\mathsf{Fin}(0) \vee \mathsf{Inf}(6)) \wedge (\mathsf{Fin}(4) \vee \mathsf{Inf}(1)))$
$\wedge((\mathsf{Fin}(2) \vee \mathsf{Inf}(6)) \wedge (\mathsf{Fin}(5) \vee \mathsf{Inf}(3)))$

(c) Step 3: Work on the second SCC. Only the first clause makes it accepting, so it is copied one time by keeping only the first clause's marks. Then, acceptance marks are updated as before as well as the acceptance condition. The conversion is done.

Figure 3.6 – Conversion of a Rabin acceptance with two clauses

Chapter 4

Conversion to deterministic co-Büchi automaton

In Chapter 3 we explained how to convert any automaton with a DNF or Streett-like acceptance to a non deterministic automaton with a co-Büchi acceptance. We previously said that this first conversion is necessary for the second one which construct the deterministic co-Büchi automaton. This chapter shows what is the link between those two conversions before explaining the second one.

4.1 Link with the nondeterministic conversion

Let $\mathcal{B}$ be a nondeterministic co-Büchi automaton resulting from the previous conversion applied on an automaton $\mathcal{A}$. It is well known that a variant of the subset construction known as the breakpoint construction can be used in order to convert a nondeterministic co-Büchi automaton with n states to a deterministic one with state space $\mathcal{O}(3^n)$ (Miyano and Hayashi, 1984). The key observation of Boker and Kupferman (2009) is that the special structure of $\mathcal{B}$ which results from the augmented subset construction makes $\mathcal{B}$ transparent to additional applications of the subset construction. It means that applying the subset construction again and again will not change its structure.

If $\mathcal{A}$ has a Streett-like acceptance, the determinization of $\mathcal{B}$ is straightforward following the standard breakpoint construction (Miyano and Hayashi, 1984). However, when starting with a DNF acceptance, because of the k copies of the automaton, each of the k parts needs to be determinized separately and connected in a round-robin fashion. Let us first understand the breakpoint construction in case of Streett-like.

4.2 The breakpoint construction with a starting Streett-like acceptance

Let $\mathcal{D} = \langle \Sigma, Q', q'_0, 1, \Delta', \mathsf{Fin}(⓿)\rangle$ be the breakpoint construction of $\mathcal{B} = \langle \Sigma, Q, q_0, 1, \Delta, \mathsf{Fin}(⓿)\rangle$.

The intuition is that the runs of $\mathcal{D}$ follows the runs of $\mathcal{B}$. To accept a run, $\mathcal{D}$ keeps track of runs that do not leave the accepting cycles of $\mathcal{B}$. Keeping track of such runs can be done by maintaining the subset of states that belong to these runs.

Let $\alpha_{\mathcal{B}}$ be the set of accepting states of $\mathcal{B}$, that is the set of states that must be seen infinitely often. For instance, if $\mathcal{B}$ is the automaton of Figure 3.2b, $\alpha_{\mathcal{B}} = \{1, \{0, 1\}\}$ and if $\mathcal{B}$ is the automaton of Figure 3.3c, $\alpha_{\mathcal{B}} = \{3, \{3\}\}$. Boker and Kupferman (2009) define a function

$$\begin{aligned} f: \quad & 2^Q \to 2^Q \\ & E \mapsto \{q \mid \langle q, E\rangle \in \alpha_{\mathcal{B}}\} \end{aligned}$$

When a subset component of $\mathcal{D}$ is in state E, it should continue and check the membership in $\alpha_{\mathcal{B}}$ only for states in $f(E)$.

We define a function d that takes a set of states and a letter and returns the set of all possile destinations.

$$\begin{aligned} d: \quad & 2^Q \times \Sigma \to 2^Q \\ & (E, l) \mapsto \left\{\delta^d \in \Delta \mid \delta^s \in E, \delta^\ell = l, \delta^M \in [n]\right\} \end{aligned}$$

$\mathcal{D}$ is constructed as follows:

- $Q' = \{\langle S, O\rangle \mid S \subseteq Q \wedge O \subseteq S \cap f(S)\}$,
- $q'_0 = \langle q_0, \emptyset\rangle$
- for all $\langle S, O\rangle \in Q'$ and $\ell \in \Sigma$, the transition relation is defined as follows
  - If $O \neq \emptyset, U = \{\langle d(S, \ell),\ \ d(O, \ell)\ \cap\ f(d(O, \ell))\rangle\} \subseteq Q'$ and $\{\delta \in \{\langle S, O\rangle\} \times \{\ell\} \times \{\emptyset\} \times U\} \subseteq \Delta'$
  - If $O = \emptyset, U = \{\langle d(S, \ell),\ \ d(S, \ell)\ \cap\ f(d(S, \ell))\rangle\} \subseteq Q'$ and $\{\delta \in \{\langle S, O\rangle\} \times \{\ell\} \times \{0\} \times U\} \subseteq \Delta'$

Thus, the accepting runs of $\mathcal{D}$ must visit the states in $2^Q \times \{\emptyset\}$ only finitely often, which implies that their equivalent runs in $\mathcal{B}$ visits infinitely often some states in $\alpha_{\mathcal{B}}$.

An example of this algorithm is shown in Figure 4.1.

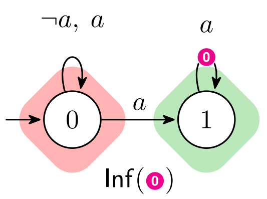
4.3 The breakpoint construction with a DNF acceptance

As explained in Section 3.2, in case of a starting automaton $\mathcal{A}$ with a DNF acceptance having k clauses, it is firstly converted to an intermediate Streett-like automaton $\mathcal{B}$ which is the union of at most k copies $\mathcal{C}_1$, $\mathcal{C}_2 \cdots \mathcal{C}_k$ of $\mathcal{A}$. To avoid the blow-up of the determinization of $\mathcal{B}$, we determinize each of the $\mathcal{C}_i$ separately and connect the resulting $\mathcal{D}_i$ in a round-robin fashion. Instead of having one set $\alpha_{\mathcal{B}}$ which contains all accepting states, we will have k sets $\alpha_{\mathcal{C}_i}$ and the fonction f is extended to f_i.

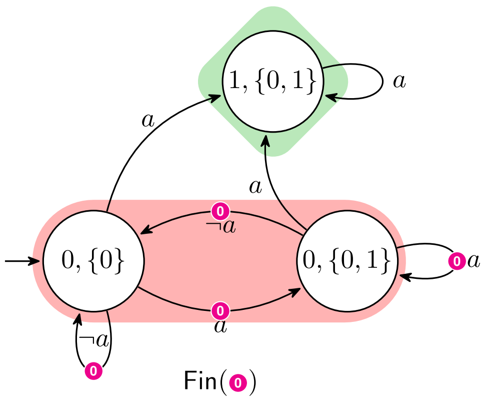
$$\begin{aligned} f_i: \quad & 2^Q \to 2^Q \\ & E \mapsto \{q \mid \langle q, E\rangle \in \alpha_{\mathcal{C}_i}\} \end{aligned}$$

Let $\mathcal{D} = \langle \Sigma, Q', q'_0, 1, \Delta', \mathsf{Fin}(\textcircled{0})\rangle$ with a DNF acceptance, be the breakpoint construction of $\mathcal{B} = \langle \Sigma, Q, q_0, 1, \Delta, \mathsf{Fin}(\textcircled{0})\rangle$. $\mathcal{D}$ is constructed as follows:

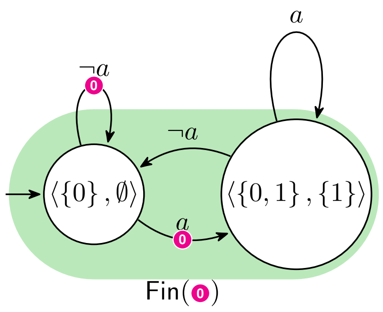
- $Q' = \{\langle S, O, i\rangle \mid S \subseteq Q \wedge O \subseteq S \cap f_i(S) \wedge i \in \{1, 2, \ldots k\}\}$,



(a) The starting automaton $\mathcal{A}$. This is the automaton of Figure 2.1.



(b) Its conversion to a nondeterministic co-Büchi automaton $\mathcal{B}$. This is the resulting automaton of Figure 3.2. Here is the information retained: $\alpha_{\mathcal{B}} = \{1, \{0, 1\}\}$.


a
$\neg a$
$\neg a$
$\langle\{0\}, \emptyset\rangle$
$\langle\{0, 1\}, \{1\}\rangle$
a
Fin(0)

(c) The deterministic co-Büchi automaton

Figure 4.1 – Conversion of a Büchi automaton to a deterministic co-Büchi automaton performing a breakpoint construction.

- $q'_0 = \langle q_0 \text{ of } \mathcal{C}_1, \emptyset, 1\rangle$
- for all $\langle S, O, i\rangle \in Q'$ and $\ell \in \Sigma$, the transition relation is defined as follows
  - If $O \neq \emptyset$,

$$\begin{aligned} U = \{\langle d(S,\ell),\ \ d(O,\ell)\ \cap\ f_i(d(O,\ell)),\ \ i \bmod k+1\rangle\} \subseteq Q' \\ \text{and }\ \{\delta \in \{\langle S,O,i\rangle\} \times \{\ell\} \times \{\emptyset\} \times U\} \subseteq \Delta' \end{aligned}$$

  - If $O = \emptyset$,

$$\begin{aligned} U = \{\langle d(S,\ell),\ \ d(S,\ell)\ \cap\ f_i(d(S,\ell)),\ \ i \bmod k+1\rangle\} \subseteq Q' \\ \text{and }\ \{\delta \in \{\langle S,O,i\rangle\} \times \{\ell\} \times \{0\} \times U\} \subseteq \Delta' \end{aligned}$$

Chapter 5

Some benchmark results

In this chapter, we present some applications of the algorithms presented above and some benchmark results.

All benchmarks have been performed on a computer having this configuration:

- CPU: Intel Core i7-6700HQ (Quad-Core 2.6 GHz / 3.5 GHz Turbo - Cache 6 Mo)
- RAM: 2 $\times$ G.Skill DDR4 2133 MHz 16 Go. 32 Go in total.

5.1 Deciding whether an LTL formula is a persistence formula

A hierarchy of temporal properties was defined by Manna and Pnueli (1990). This hierarchy relates omega-regular languages to structural properties of the automata that can recognize them. This hierarchy is shown in Figure 5.1. The following sections explains and compare two ways to test if a formula has the persistence property.

5.1.1 Current processing chain to test persistence property in Spot

Since recurrence and persistence are dual classes, instead of testing if φ has a persistence property, Spot tests if $\neg\varphi$ has the recurrence property. The recurrence subclass contains all properties that can be recognized by a deterministic Büchi automaton.

The processing chain of Spot to test persistence property is summarized in Figure 5.2. Firstly, $\neg\varphi$ is translated into a Büchi automaton which may not be deterministic, then it is determinized. A deterministic automaton with parity acceptance is obtained. This DPA is converted to a Rabin automaton. Finally, it is converted to a B üchi automaton (Krishnan et al., 1994). Note that this algorithm necessarily produces a DBA if it exists. If a DBA is produced, then $\neg\phi$ has the recurrence property, otherwise it has not.

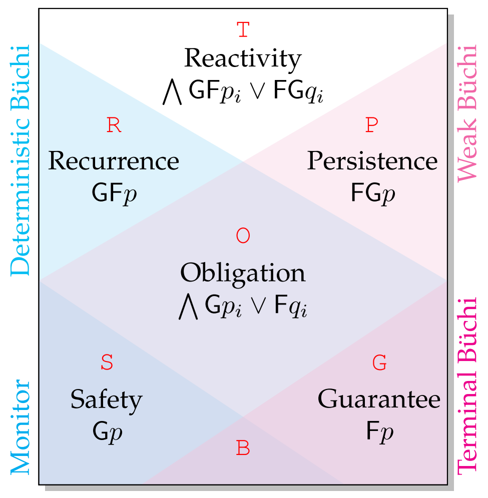


Figure 5.1 – Hierarchy of temporal properties as defined by Manna and Pnueli (1990). This picture comes from Spot's (Duret-Lutz et al., 2016) website `https://spot.lrde.epita.fr/hierarchy.html`. Each class includes everything below it. For instance, the recurrence class includes the obligation class which also includes the safety and guarantee classes. The reactivity class represents all possible omega-regular languages, i.e., all languages that can be recognized by a non-deterministic Büchi automaton. The recurrence subclass contains all properties that can be recognized by a deterministic Büchi automaton. The dual class, persistence properties, are those that can be reresented by a weak Büchi automaton (i.e., in each SCC either all states are accepting, or all states are rejecting).

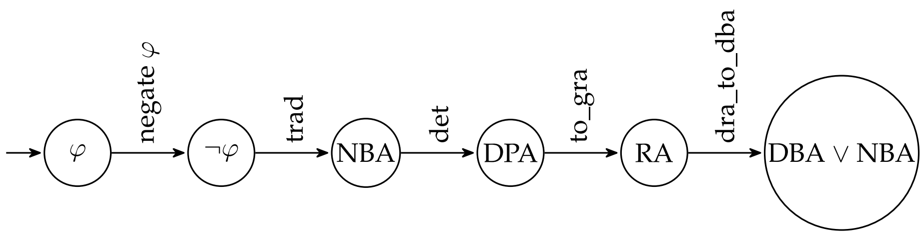


Figure 5.2 – Current processing chain for persistence membership testing.

5.1.2 New processing chain using the co-Büching Toolbox

All the contributions made gave Spot a new way for persistence membership testing. It is summarized in Figure 5.3.

As explained in the introduction, not all LTL formula can be expressed with a co-Büchi acceptance. Only the formula having the persistence property can do so. It should be recalled that the constructions of Boker and Kupferman (2011) guarantee that the resulting automaton recognizes at least the same language as the starting automaton. That means, the language of the starting automaton is always included in the language of the resulting automaton. Testing if a formula ϕ can be expressed with a co-Büchi automaton, that means testing if ϕ has the persistence property is reduced to testing the inclusion in the other direction.

To do so, an emptiness check on the product of the nondeterministic co-Büchi automaton

NCA_φ and an automaton $NBA_{\neg\varphi}$ (recognizing the dual language of φ) is performed. There is no need to determinize the NCA obtained because nondeterministic and deterministic co-Büchi automata have the same expressivity.

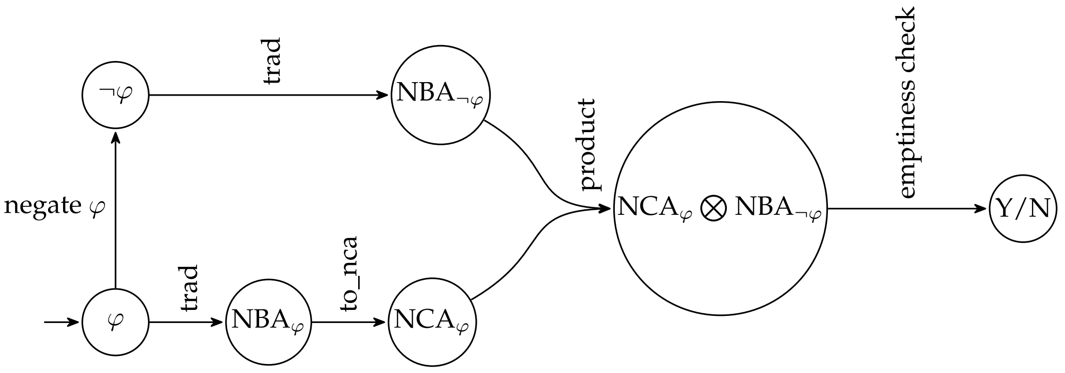


Figure 5.3 – New processing chain for persistence membership testing.

5.1.3 Benchmark results

1,000,000 LTL formulas have been tested. Each of them have either the recurrence property or the persistence property.

Firstly, we asked both processing chain to find recurrence properties and measured the running time of each method. It is presented in Figure 5.4. There is a group of condensed points at the left and the bottom of the Figure. This is due to a syntactic checking performed by both methods. Apart from that, there is 50,457 cases where the old method is faster than the new one and 268,939 cases where the new method excel. It seems that the new method is more comfortable when deciding recurrence property.

Secondly, we asked both processing chain to find persistence properties. The results are shown in Figure 5.5. The new method has 544,894 cases where it is faster and 62,989 cases where it is slower. The new method seems to also be a good method for testing persistence property.

Also note that overall formulas tested, the new method respond under 0.4 seconds regardless of the complexity of the formula. Moreover, the total walltime to proccess all 1.000.000 formulas is

- 214.9 seconds for the old processing chain to find LTL formula in persistence class
- 198.5 seconds for the new processing chain to find LTL formula in persistence class
- 944.2 seconds for the old processing chain to find LTL formula not in persistence class
- 468.5 seconds for the new processing chain to find LTL formula not in persistence class

In total (persistence or recurrence), the new method is 81.3% faster than the old one.

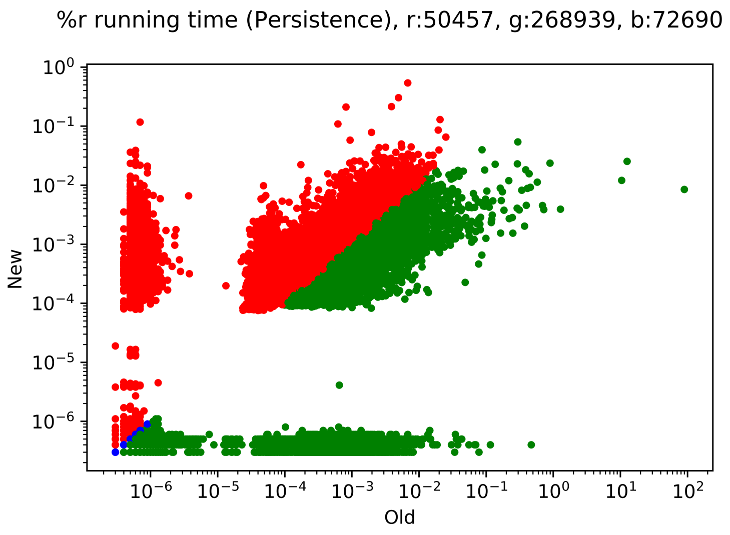


Figure 5.4 – Benchmark result comparing the CPU-time of the old processing chain and the new one. The objective is to find which LTL formulas belong to the persistence class over 1,000,000 formulas.

5.2 Translating a recurrence formula to a deterministic Büchi automaton

Another application of the co-Büching toolbox is the construction of deterministic Büchi automaton thanks to the duality between Büchi and co-Büchi. Since a recurrence formula can be represented by a deterministic Büchi automaton (Manna and Pnueli, 1990), 649.924 LTL formula belonging to the recurrence class have been tested on two processing chains. The current one is the same explained in Section 5.1.1.

The new one is the following:

$$\begin{tikzcd} {} \arrow[r] & \varphi \arrow[r, "\text{negate } \varphi"] & \neg\varphi \arrow[r, "\text{trad}"] & \text{NBA} \arrow[r, "\text{to\_nca}"] & \text{NCA} \arrow[r, "\text{to\_dca}"] & \text{DCA} \arrow[r, "\text{dualize}"] & \text{DBA} \end{tikzcd}$$

Figure 5.6 – Processing chain to build deterministic Büchi automaton using the co-Büching toolbox

The results are shown in Figure 5.7 and Figure 5.8. Regarding Figure 5.7, we can see that in 469,500 cases, both methods build automata with the same number of state. But, in 110,600 cases, the new method builds smaller automaton against 69824 cases where the old method is better. In Figure 5.8, we might be surprised that the new method is generally smaller than the old method compare to results of Figure 5.5. There is 525,593 cases where the old method is faster and 124,331 cases where the new method is faster. This is because this time, as we want

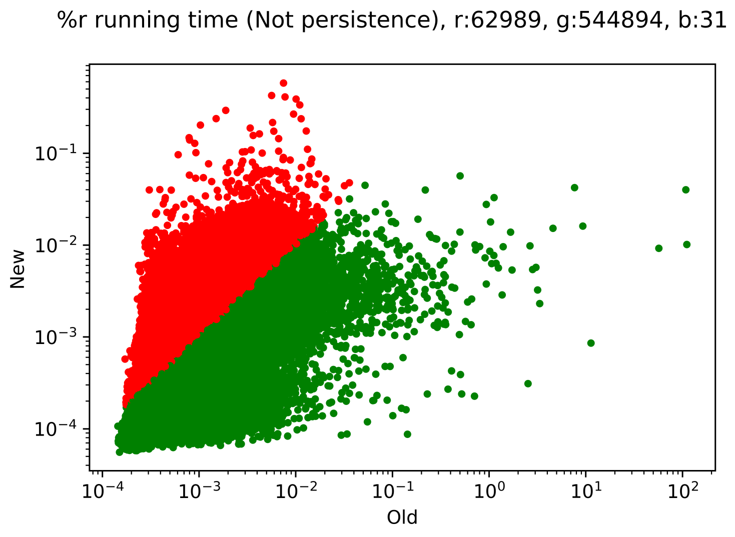


Figure 5.5 – Benchmark result comparing the CPU-time of the old processing chain and the new one. The objective is to find which LTL formulas belong to the recurrence class over 1,000,000 formulas.

to build deterministic Büchi automata, we need to determinize the co-Büchi automaton. This is probably why the ned method is slower here. It takes more time to build smaller automaton. Also, when translating ϕ to a BA, the old method might end with DBA directly. Such cases may come more often than one might think and give the old method a serious advantage.

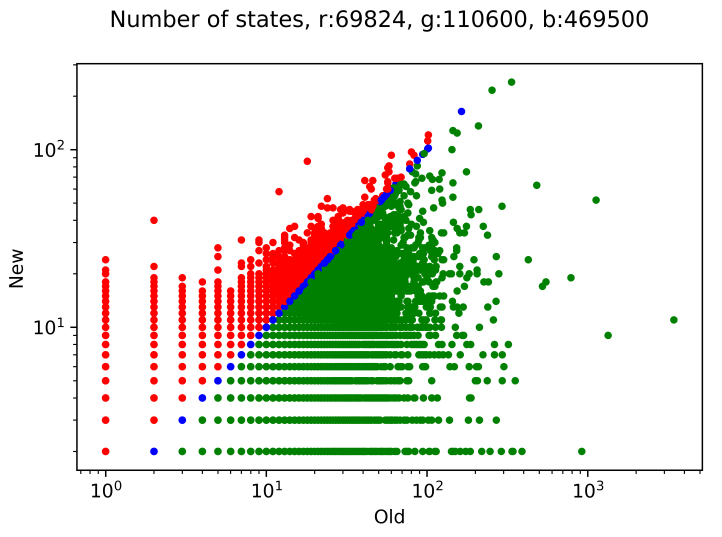


Figure 5.7 – Benchmark result comparing the old processing chain and the new on on the number of states that each resulting deterministic Büchi automaton has. 649.924 formula in recurrence class have been tested.

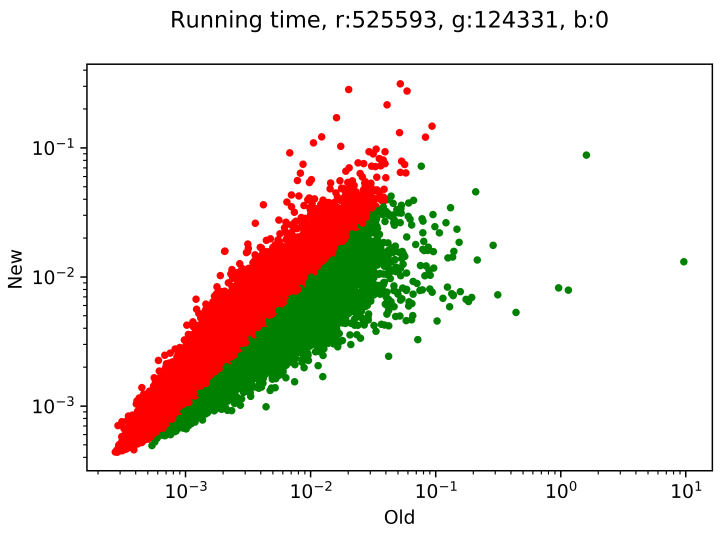


Figure 5.8 – Benchmark result comparing the old processing chain and the new on on the running time of the construction of deterministic Büchi automata from 649.924 formula in the recurrence class.

Chapter 6

Conclusion

This report shows a set of tools that helps to convert any DNF or Streett-like acceptance condition to both nondterministic and deterministic co-Büchi automaton. The implemented algorithms are an improved version of the algorithms described by Boker and Kupferman (2009, 2011) which solve the long-standing problem of the state blow-up that such a conversion involves. These conversions are some extensions of the augmented subset construction and the breakpoint construction Miyano and Hayashi (1984).

An application of such methods is deciding the membership of an LTL formula to the persistence class. Boker and Kupferman (2011) show that when the starting language is not recognizable by a co-Büchi acceptance, the language of the resulting automaton will at least contain the same language. Since it is known that all formula in the persistence class are expressible by a co-Büchi acceptance, testing their membership to persistence class can be easily done by testing in the inclusion in the other way. This means testing if the language of the resulting automaton is included in the language of the starting automaton. Comparing this processing chain to the existing one of Spot has shown that on average the new way is faster.

Overall, the new method is definitely the best. It is 81,3% faster for checking either persistence or recurrence class membership. We believe it can be improved further by looking at the cases where the old method is still better. Also, concerning the number of states of deterministic Büchi automata produced from LTL formula in recurrence class, we should look at the respective worst case of each processing chain trying to figure out some remarks that can be played upstream.

Chapter 7

Bibliography

Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J., Müller, D., Parker, D., and Strejček, J. (2015). The Hanoi Omega-Automata format. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV'15)*, volume 9206 of *Lecture Notes in Computer Science*, pages 479–486. Springer. (pages 4 and 8)

Boker, U. and Kupferman, O. (2009). Co-ing büchi made tight and useful. In *Proceedings of the Twenty-Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 2009)*, pages 245–254. IEEE Computer Society Press. (pages 1, 5, 12, 21, 22, and 31)

Boker, U. and Kupferman, O. (2011). Co-büching them all. In *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 184–198. (pages 1, 5, 12, 15, 26, and 31)

Büchi, J. R. (1990). On a decision method in restricted second order arithmetic. In Mac Lane, S. and Siefkes, D., editors, *The Collected Works of J. Richard Büchi*, pages 425–435, New York, NY. Springer New York. (page 4)

Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., and Xu, L. (2016). Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer. (pages 4, 6, 7, and 26)

Emerson, E. A. and Lei, C.-L. (1987). Modalities for model checking: branching time logic strikes back. *Science of Computer Programming*, 8(3):275 – 306. (pages 4 and 8)

Krishnan, S. C., Puri, A., and Brayton, R. K. (1994). Deterministic ω-automata vis-a-vis deterministic büchi automata. In Du, D.-Z. and Zhang, X.-S., editors, *Algorithms and Computation, 5th International Symposium, ISAAC '94, Beijing, P. R. China, August 25-27, 1994, Proceedings*, pages 378–386, Berlin, Heidelberg. Springer Berlin Heidelberg. (pages 5 and 25)

Kurshan, R. P. (1994). *Computer-aided Verification of Coordinating Processes: The Automata-theoretic Approach*. Princeton University Press, Princeton, NJ, USA. (page 4)

Manna, Z. and Pnueli, A. (1990). A hierarchy of temporal properties (invited paper, 1989). In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC '90, pages 377–410, New York, NY, USA. ACM. (pages 5, 25, 26, and 28)

Miyano, S. and Hayashi, T. (1984). Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330. (pages 5, 21, and 31)

Piterman, N. (2007). From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3):5. (page 5)

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA. IEEE Computer Society. (page 4)

Safra, S. (1988). On the complexity of omega-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, pages 319–327, Washington, DC, USA. IEEE Computer Society. (page 5)

Vardi, M. Y. (2007). *Automata-Theoretic Model Checking Revisited*, pages 137–150. Springer Berlin Heidelberg, Berlin, Heidelberg. (page 4)

Xu, L. (2016). Product of parity automata. Technical Report 1607, EPITA Research and Development Laboratory (LRDE). (page 9)