Empirical Performance Investigation of a Büchi Complementation Construction

Daniel Weibel

July 14, 2015

Abstract

This will be the abstract.

Acknowledgements

Contents

1 Background **3**
1.1 Büchi Automata and Other ω-Automata . . . 4
1.1.1 Büchi Automata . . . 4
1.1.2 Other ω-Automata . . . 5
1.2 Run Analysis in Non-Deterministic Büchi Automata . . . 6
1.3 Büchi Complementation Constructions . . . 6
1.3.1 Ramsey-Based Approach . . . 6
1.3.2 Determinization-Based Approach . . . 7
1.3.3 Rank-Based Approach . . . 8
1.3.4 Slice-Based Approach . . . 10
1.4 Empirical Performance Investigations . . . 11
1.5 Preliminaries . . . 11
1.5.1 Büchi Automata . . . 11
1.5.2 Other ω-Automata . . . 12
1.5.3 Complementation of Büchi Automata . . . 13
1.5.4 Complexity of Büchi Complementation . . . 14
1.6 Run Analysis . . . 14
1.6.1 From Run Trees to Split Trees . . . 15
1.6.2 Reduced Split Trees . . . 17
1.6.3 Run DAGs . . . 18
1.7 Run Analysis . . . 18
1.7.1 Run Trees . . . 18
1.7.2 Failure of the Subset-Construction for Büchi Automata . . . 19
1.7.3 Split Trees . . . 19
1.7.4 Reduced Split Trees . . . 20
1.7.5 Run DAGs . . . 20
1.8 Review of Büchi Complementation Constructions . . . 21
1.8.1 Ramsey-Based Approaches . . . 21
1.8.2 Determinisation-Based Approaches . . . 21
1.8.3 Rank-Based Approaches . . . 21
1.8.4 Slice-Based Approaches . . . 21
1.9 Empirical Performance Investigations . . . 21

2 The Fribourg Construction **22**
2.1 Basics . . . 23
2.2 The Construction . . . 23
2.2.1 Upper Part . . . 23
2.2.2 Lower Part . . . 23
2.3 First Stage: Constructing the Upper Part . . . 24
2.4 Second Stage: Adding the Lower Part . . . 25
2.4.1 Construction . . . 25
2.4.2 Meaning and Function of the Colours . . . 27
2.5 Intuition for Correctness . . . 27
2.6 Optimisations . . . 28
2.6.1 Removal of Non-Accepting States (R2C) . . . 28

2.6.2 Merging of Adjacent Sets (M1) 28
2.6.3 Reduction of 2-Coloured Sets (M2) 28

A Plugin Installation and Usage **29**

B Median Complement Sizes of the GOAL Test Set **30**

C Execution Times **33**

Chapter 1

Background

Contents

1.1 Büchi Automata and Other ω-Automata 4
1.1.1 Büchi Automata 4
1.1.2 Other ω-Automata 5
1.2 Run Analysis in Non-Deterministic Büchi Automata 6
1.3 Büchi Complementation Constructions 6
1.3.1 Ramsey-Based Approach 6
1.3.2 Determinization-Based Approach 7
1.3.3 Rank-Based Approach 8
1.3.4 Slice-Based Approach 10
1.4 Empirical Performance Investigations 11
1.5 Preliminaries 11
1.5.1 Büchi Automata 11
1.5.2 Other ω-Automata 12
1.5.3 Complementation of Büchi Automata 13
1.5.4 Complexity of Büchi Complementation 14
1.6 Run Analysis 14
1.6.1 From Run Trees to Split Trees 15
1.6.2 Reduced Split Trees 17
1.6.3 Run DAGs 18
1.7 Run Analysis 18
1.7.1 Run Trees 18
1.7.2 Failure of the Subset-Construction for Büchi Automata 19
1.7.3 Split Trees 19
1.7.4 Reduced Split Trees 20
1.7.5 Run DAGs 20
1.8 Review of Büchi Complementation Constructions 21
1.8.1 Ramsey-Based Approaches 21
1.8.2 Determinisation-Based Approaches 21
1.8.3 Rank-Based Approaches 21
1.8.4 Slice-Based Approaches 21
1.9 Empirical Performance Investigations 21

1.1 Büchi Automata and Other ω-Automata

Formal definitions for example in [39][40][53]

1.1.1 Büchi Automata

Büchi automata are a type of the so-called ω-automata ("omega"-automata). ω-automata are finite state automata that process infinite words. Thus, an ω-automaton never "stops" reading a word, because the word it is reading has an infinite number of symbols. But still, ω-automata can accept or reject the words they read by the means of special *acceptance conditions*. In fact, the only difference between classical finite state automata on finite words and ω-automata is the acceptance condition.

For the case of Büchi automata, this is the Büchi acceptance condition that we describe below.

Büchi Acceptance Condition

A Büchi automaton A is defined by the 5-tuple $A = (Q, \Sigma, q_0, \delta, F)$ with the following components:

- Q: a finite set of states
- Σ: a finite alphabet
- q_0: an initial state, $q_0 \in Q$
- δ: a transition function, $\delta : Q \times \Sigma \to 2^Q$
- F: a set of accepting states, $F \in 2^Q$

We denote by Σ^ω the set of all words of infinite length over the alphabet Σ. A Büchi automaton runs on the elements of Σ^ω. In the following, we define the acceptance behaviour of a Büchi automaton A on a word $\alpha \in \Sigma^\omega$.

- A *run* of Büchi automaton A on a word $\alpha \in \Sigma^\omega$ is a sequence of states $q_0 q_1 q_2 \dots$ such that q_0 is A's initial state and $\forall i \geq 0 : q_{i+1} \in \delta(q_i, \alpha_i)$
- $\inf(\rho) \in 2^Q$ is the set of states that occur infinitely often in a run ρ
- A run ρ is accepting if an only if $\inf(\rho) \cap F \neq \varnothing$
- A Büchi automaton A accepts a word $\alpha \in \Sigma^\omega$ if and only if there is an accepting run of A on α

Expressivity

Büchi automata are expressively equivalent to the ω-regular languages. This means that every language recognised by a Büchi automaton is an ω-regular language, and that for every ω-regular language there exists a Büchi automaton that recognises it. This property has been proved by Büchi himself in his initial publication in 1962 [4].

However, this equivalence with the ω-regular languages does only hold for *non-deterministic* Büchi automata. Deterministic Büchi automata are less expressive than non-deterministic Büchi automata. In particular, the class of languages represented by deterministic Büchi automata is a strict subset of the class of languages represented by non-deterministic Büchi automata. This property has also been proved by Büchi [4].

This means that there exist languages that can be recognised by a non-deterministic Büchi automaton, but not by a deterministic one. A typical example is the language $(0+1)^*1^\omega$. This is the language of all words consisting of 0 and 1 with a finite number of 0 and an infinite number of 1. It is proved in various publications that this language can be recognised by a non-deterministic Büchi automaton, but not by a deterministic Büchi automaton [49][32].

The most important consequence of this fact is that Büchi automata can, in general, not be determinised. This means that it is not possible to turn *every* non-deterministic Büchi automaton into a deterministic one. This contrasts with the case of the classical finite state automata on finite words, where *every* non-deterministic automata (NFA) can be turned into a deterministic automaton (DFA), by the means of, for example, the subset constrution introduced by Rabin and Scott in 1959 [29].

It has been stated that the fact that Büchi automata can in general not be determinised is the main reason that Büchi complementation is such a hard problem [25]. We will see why this is the case below.

Complementation

Büchi automata are closed under complementation. This means that the complement of every Büchi automaton (non-deterministic or deterministic) is in turn a Büchi automaton. This result has been proved by Büchi in his introducing paper from 1962 [4].

The difficulty of the concrete complementation problem does however strongly depend on whether the Büchi automaton is deterministic or non-deterministic. For deterministic Büchi automata, the complementation is "easy" and regarded as a "solved problem". There is a well-known construction introduced by Kurshan in 1987 that complements a deterministic Büchi automaton in polynomial time [17]. The resulting complement is a non-deterministic Büchi automaton and has a size that is at most the double of the input automaton.

For non-deterministic Büchi automata, on the other hand, the complementation problem is much more difficult. The main reason is, as mentioned, the fact that non-deterministic Büchi automata cannot be determinised. If they could be determinised, then a non-deterministic Büchi automata could be complemented by first determinising it, and then complementing the deterministic automaton with the Kurshan construction. If the determinisation construction would also be efficient (that is, having polynomial complexity), then we would have an efficient complementation procedure for non-deterministic Büchi automata. In this case, "Büchi complementation" would probably be no active research topic but rather a solved problem.

However, non-deterministic Büchi automata cannot be determinised, and hence this straightforward complementation approch is not possible. Consequently, different ways for complementing non-deterministic Büchi automata have to be found, and these ways turn out to be very complex. It is this complexity that makes Büchi complementation an active research topic as, regarding the concrete usages of Büchi complementation in, for example, model checking, it is of great importance to find more and more efficient ways to complement non-deterministic Büchi automata.

1.1.2 Other ω-Automata

After the introduction of Büchi automata in 1962, several other types of ω-automata have been proposed. The most notable ones are Muller automata (Muller, 1963 [22]), Rabin automata (Rabin, 1969 [30]), Streett automata (Streett, 1982 [38]), and parity automata (Mostowski, 1985 [21]).

These automata differ from Büchi automata only in their acceptance condition, that is, the condition that a run ρ is accepted. Table 1.1 gives a formal definition of the acceptance conditions of these types of ω-automata.

Type	Definitions	Acceptance condition
Muller	$F \subseteq 2^Q$	$\inf(\rho) \in F$
Rabin	$\{(E_1, F_1), \ldots, (E_r, F_r)\}$, $E_i, F_i \subseteq Q$	$\exists i : \inf(\rho) \cap E_i = \varnothing \wedge \inf(\rho) \cap F_i \neq \varnothing$
Streett	$\{(E_1, F_1), \ldots, (E_r, F_r)\}$, $E_i, F_i \subseteq Q$	$\forall i : \inf(\rho) \cap E_i \neq \varnothing \vee \inf(\rho) \cap F_i = \varnothing$
Parity	$c : Q \to \{1, \ldots, k\}$, $k \in \mathbb{N}$	$\min\{c(q) \mid q \in \inf(\rho)\} \bmod 2 = 0$
Büchi	$F \subseteq Q$	$\inf(\rho) \cap F \neq \varnothing$

Table 1.1: Acceptance conditions of Muller, Rabin, Streett, parity, and Büchi automata.

For the Muller acceptance condition, the set of infinitely occuring states of a run ($\inf(\rho)$) must match one of several predefined set of states. The Muller acceptance condition is the most general one, and all the other acceptance conditions in Table 1.1 can be expressed by the Muller condition [18].

The Rabin and Streett acceptance conditions are the negations of each other. This means that a run satisfies the Rabin acceptance condition, if and only if it does not satisfy the Streett acceptance condition. They both use a list of pairs of state sets. A run is accepted if there is a pair for which the first element contains an infinitely occuring state and the second element does not (Rabin condition), or if for all pairs the first elements do not contain an infinitely occuring state or all the second elements do contain an infinitely occuring state (Streett condition).

The parity condition assigns a number (color) to each state. A run is accpted if and only if the infinitely often occuring state with the smallest number has an even number.

At this point we will start using a notation for the different types of ω-automata that has been used in [27] and [42]. It consists of a three-letter acronymes of the form $\{D, W\} \times \{B, M, R, S, P\} \times W$. The first letter, D or N specifies whether the automaton is deterministic or non-deterministc. The second letter is the initial letter of the automaton type, that is, B for Büchi, M for Muller, R for Rabin, S for Streett, and P for parity automata. The third letter specifies on which the automaton runs, and is in our case always W meaning "words". Thus, throughout this thesis we will use DBW for deterministic Büchi automata, NBW for non-deterministic Büchi automata, DMW for deterministic Muller automata, and so on.

Regarding the expressivity of Muller, Rabin, Streett, and parity automata, it turned out that, like non-deterministic Büchi automata, they are equivalent to the ω-regular languages [39]. However, unlike Büchi automata, for Muller, Rabin, Streett, and parity automata this equivalence holds for deterministic *and* non-deterministic automata. That is, unlike Büchi automata, these automata *can* be determinised. In summary, there is thus an equivalence between NBW, DMW, NMW, DRW, NRW, DSW, NSW, DPW, NPW, and the ω-regular languages. Only the DBW, as a special case, has a different expressivity, which is a strict subset of the expressivities of the other automata types.

1.2 Run Analysis in Non-Deterministic Büchi Automata

There are two basic ways for arraning all runs of a non-deterministic Büchi automaton: run DAGs and trees [53]

1.3 Büchi Complementation Constructions

Since the introduction of Büchi automata in 1962, many constructions for complementing non-deterministic Büchi automata have been proposed.

1.3.1 Ramsey-Based Approach

The Ramsey-based approach has its name from a Ramsey-based combinatorial argument that is used in the complementation constructions. Ramsey was a British mathematician who lived at the beginning of the 20th century and founded a branch of combinatorics called the Ramsey theory [9].

Common to the Ramsey-based complementation constructions is that they stay completely within in the framework of Büchi automata. That is, they do not include intermediate automata of different types, as for example the determinization-based constructions. Rather, Ramsey-based constructions construct the complement automata directly by combinatorial operations on the states and transitions.

Büchi, 1962

The first Büchi complementation construction at all was described by Büchi himself, along with the introduction of Büchi automata in 1962 [4]. This complemenation construction is a Ramsey-based construction. It involves a combinatorial argument based on work by Ramsey [31]. The construction is complicated, and has a doubly-exponential worst-case state complexity of $2^{2^{O(n)}}$ [51]. This means that if we assume, for example, the concrete complexity to be 2^{2^n}, then an automaton with 6 states may result in a complement with at most 2^{2^6} states, which is more than 18 quintillions (18 billion billions).

The complexity of this worst-case is very high, and it would probably be impossible to complement such a worst-case automaton in practice. This is why all the subsequent complementation constructions, until today, have the goal to reduce this worst-case complexity. In this way, the worst-case state complexity became the main measure of performance for Büchi complementation constructions.

Sistla, Vardi, and Wolper, 1985

Another Ramsey-based construction has been introduced by Sistla, Vardi, and Wolper in 1987 [37] (first published in 1985 [36]). It is an improvement of Büchi's construction and the first one that involves only an exponential, instead of a doubly-exponential, worst-case state complexity. The complexity of this construction has been calculated to be $O\left(2^{4n^2}\right)$ (see [33][26]).

The Ramsey-based approach is the oldest of the four approaches and it was particularly

1.3.2 Determinization-Based Approach

The determinization-based complementation constructions proceed by converting an NBW to a deterministic automaton, complementing the deterministic automaton, and finally converting the complement automaton back to an NBW. The deterministic automaton cannot be a DBW (because NBW and DBW are not equivalent), however it can be a DMW, DRW, DSW, or DPW.

The idea behind this approach is that the complementation of deterministic ω-automata is easier than the complementation of non-deterministic ω-automata. The complementation problem is then in fact reduced to conversions between different types of automata. From these conversions, the conversion from the initial NBW to a deterministic ω-automaton is the most difficult and crucial one.

Safra, 1988

The first determinisation-based complementation construction has been described by Safra in 1988 [33]. Safra's main work was actually a determinisation construction for converting an NBW to a DRW. This is what today is known as *Safra's construction*. Safra then describes complementation as a possible application of his determinisation construction. He also presents the additional conversions that are needed for the entire complementation construction. The conversion steps of Safra's complementation procedure are as follows.

1. NBW $\longrightarrow$ DRW (Safra's construction)
2. DRW $\longrightarrow$ $\overline{\text{DSW}}$ (complementation)
3. $\overline{\text{DSW}}$ $\longrightarrow$ $\overline{\text{DRW}}$
4. $\overline{\text{DRW}}$ $\longrightarrow$ $\overline{\text{NBW}}$

The complementation step from a DRW to a DSW that accepts the complement language can be trivially done by interpreting the Rabin acceptance condition as a Streett acceptance condition. This is possible, because these two acceptance conditions are the negations of each other (see Section 1.1.2. The conversions from DSW to DRW, and from DRW to NBW are not of major difficulty or complexity, and are described by Safra in [33] (Lemma 3 and Lemma 5).

The core is the conversion from NBW to DRW (Safra's construction). This construction is basically a modified subset construction. That is, the output automaton is built up from an initial state step-by-step by adding new states and transitions. The main difference to the subset construction is that in Safra's construction, the output-states consist of trees of subsets of input-states, rather than just of subsets of input-states. These trees of subsets of states are called *Safra trees*. The details of the construction are rather intricate, but well described in [33]. The deterministic automaton that results from Safra's construction can then be interpreted as a Rabin automaton.

The state growth of Safra's construction is $2^{O(n \log n)}$, where n is the number of states of the input automaton. The additional conversions (DSW to DRW, and DRW to NBW) have a lower state complexity than this, so that the overall complexity of the entire complementation procedure is still $2^{O(n \log n}$.

Muller and Schupp, 1995

Most other determinisation-based complementation constructions are based on improvements of Safra's construction. One of them is the construction for converting NBW to DRW proposed in 1995 by Muller an Schupp. This construction is said to be simpler and more intuitive than Safra's construction [32], however, often produces larger output automata in practice [2]. The theoretically caluclated state complexity of the Muller-Schupp construction is $2^{O(n \log n)}$, that is, similar to Safra's construction. A comparison of the Muller-Schupp construction and Safra's construction can be found in [2].

Piterman, 2007

Another improvement of Safra's construction has been proposed in 2007 by Piterman from EPF Lausanne [28] (first presented at a conference in 2006 [27]). This construction converts a NBW to a DPW, rather than a DRW. Piterman's construction uses a more compact version of Safra trees, which allows it to produce smaller output automata. The concrete worst-case state growth of Piterman's construction is $2n^n n!$, opposed to $12^n n^{2n}$ of Safra's construction [28]. Complementation with Piterman's construction is done in the following steps.

1. NBW $\longrightarrow$ DPW (Piterman's construction)
2. DPW $\longrightarrow$ $\overline{\text{DPW}}$ (complementation)
3. $\overline{\text{DPW}}$ $\longrightarrow$ $\overline{\text{NBW}}$

The complementation step from a DPW to a DPW accepting the complement language can be trivially done by, for example, increasing the number of each state by 1. The conversion from a DPW to an NBW can alse be done without major complexity [42].

1.3.3 Rank-Based Approach

The rank-based approach was the third of the four proposed main complementation approaches. It does neither include Ramsey theroy, nor determinisation. Rather, it is based on run analysis with run DAGs. The link of run analysis with run DAGs to complementation is as follows. A run DAG allows to summarise all the possible runs of an automaton on a specific word. If all these runs are rejecting, then we say that the entire run DAG is rejecting. In this case, the automaton does not accept the word, and consequently, the complement automaton must accept this word. Conversely, if one or more runs in the run DAG are not rejecting, then the entire run DAG is not rejecting. In this case, the automaton accepts the word, and consequently, the complement automaton must no accept this word.

The information of whether a run DAG is rejecting or not is expressed with so-called ranks. These are numbers that are assigned to the vertices of a run DAG, one rank per vertex. These ranks are assigned in a way that each run of a run DAG eventually gets trapped in a rank. From this information it is then possible to deduce whether the run DAG is rejecting or not. This in turn determines whether the complement automaton must accept the given word, or not.

This entire analysis of run DAGs with ranks is included in a subset construction. This means that the individual run DAGs are not constructed explicitly for each word, but rather implicitly "on-the-fly" within the complement automaton under construction. From a practical point of view, this means that rank-based constructions proceed in a subset construction based fashion. That is, the construction of the complement automaton is started with an initial state, and then step-by-step, successor states are added. Each output state consists of subsets of input-states.

Klarlund, 1991

The first rank-based construction has been proposed in 1991 by Klarlund [12]. However, Klarlund used the term *progress measure* instead of *rank*. This is because he looked at the ranks as a measure for the "progress" of a run towards the satisfaction of a certain property. The term *rank* has, to the best of our knowledge, been introduced by Thomas in 1999 [41]. Klarlund also did not mention run DAGs, but they are implicit in his description of the construction. The construction works as described above by performing a modified subset construction.

Kupferman and Vardi, 1997/2001

This construction by Kupferman and Vardi has been published as a preliminary conference version in 1997 [15], and as a journal version in 2001 [16]. Both publications are entitled "Weak Alternating Automata Are Not That Weak". The idea of the construction described by Kupferman and Vardi is the same as Klarlund's construction from 1991 [12]. However, Kupferman and Vardi provide two different descriptions for this idea.

The first description does not use run DAGs and ranks, but rather converst the input automaton to a weak alternating automaton, which is complemented, and then converted back to a non-deterministic Büchi automata. Weak alternating automata (WAA) have been introduced in 1986 by Muller, Saoudi, and Schupp [23]. Kupferman and Vardi state that this construction is conceptually simpler and easier implementable than Klarlund's construction [12]. This first version of Kupferman and Vardi's construction is described in both, the publications from 1997 [15] and 2001 [16].

The second description in turn is rank-based, as described above, and works in the subset construction fashion without intermediate automata. Kupferman and Vardi point out that this version of the construction is identical to Klarlund's construction. What changes is just the terminology, for example "ranks" instead of "progress measure". This second version of Kupferman and Vardi's construction is to the best of our knowledge only described in the publication from 2001 [16], however, we are not sure, because we could not access the publication from 1997[15].

The automata produced by the two versions of Kupferman and Vardi's construction are identical. The worst-case state complexity has been calculated to be approximately $(6n)^n$ [35][50].

Thomas, 1999

This construction by Thomas [41] is based on the WAA construction by Kupferman and Vardi from 1997 [15]. It uses the concept of ranks, but does not proceed in the subset construction manner, as Klarlund's construction [12] and Kupferman and Vardi's second version [16]. Rather, it transforms the input NBW to an intermediate automaton, complements it, and converts the result back to an NBW. That is, it proceeds in a similar fashion as Kupferman and Vardi's first version [15]. The type of the intermediate automaton is a weak alternating parity automaton (WAPA), that is, a weak alternating automaton with the parity acceptance condition.

Friedgut, Kupferman, and Vardi, 2006

In 2006, Friedgut, Kupferman, and Vardi published a paper entitled "Büchi Complementation Made Tighter" [7] (a preliminary version of the paper has appeared in 2004 [6]). There, they describe an improvement to the second (rank-based) version of Kupferman and Vardi's construction from 2001 [16]. The improvement consists in the so-called *tight ranking*, a more sophisticated ranking function. It allows to massively reduce the worst-case state complexity of the construction to $(0.96n)^n$.

Schewe, 2009

In 2009, Schewe presented another improvement to the construction by Friedgut, Kupferman, and Vardi from 2006 [35]. His paper is entitled "Büchi Complementation Made Tight", which hints at the relation to the paper by Friedgut, Kupferman, and Vardi [7]. Schewe's improvement consists in a further refinement of the construction, in particular the use of turn-wise tests in the cut-point construction step. This improvement allows to further reduce the worst-case state complexity of the construction to $(0.76\,(n+1))^{n+1}$. This coincides, modulo a polynomial factor, with the lower bound for the state complexity of Büchi complementation of $(0.76n)^n$ that has been previously established by Yan in 2006 [54][55].

This result narrows down the possible range for the real worst-case state complexity of Büchi complementation considerably. It cannot be lower than the lower bound of $(0.76n)^n$ by Yan, and it cannot be higher than the complexity of Schwewe's construction of $(0.76\,(n+1))^{n+1}$. For this reason, we say that the proven worst-case complexity of a specific construction serves as an upper bound for the actual complexity of the problem.

1.3.4 Slice-Based Approach

The slice-based approach was the last approach that has been proposed. Its idea is very similar to the rank-based approach, but the main difference is the use of reduced split trees instead of run DAGs. The basic idea is to look at a state of the output automaton under construction as a horizontal level of a reduced split tree. Based on this, for each alphabet symbol, the succeeding level of the reduced split tree is determined, which results in a new state in the output automaton. These levels of reduced split trees are called *slices*, hence the name slice-based approach.

Like rank-based constructions, slice-based construction are essentially enhanced subset constructions. The slice-based constructions, however, include two runs of a subset construction, where the second one is typically more sophisticated than the first one.

Vardi and Wilke, 2007

The first slice-based Büchi complementation construction has been proposed in 2007 by Vardi and Wilke [52]. In this work, the authors review translations from various logics, including monadic second order logic of one successor (S1S), to ω-automata. They devise the slice-based complementation construction as a by-product of a determinisation construction for Büchi automata that they also introduce in this work.

Vardi and Wilke use left-to-right reduced split trees for their construction. That means, accepting states are put to the left of non-accepting states, and only the left most occurrence of each state is kept. The construction works by two passes of the enhanced subset construction. The first one (initial phase) is as described above. The second one (repetition phase), does additionally include decorations of the vertices of the reduced split trees (subsets) consisting of the three labels *inf*, *die*, and *new*. These decoration serves to keep track of the criterion that a word is rejected if and only if all of the branches of the corresponding reduced split tree contain only a finite number of left-turns. The worst-case state complexity of Vardi and Wilke's construction is $(3n)^n$ [52].

The slice-based construction by Vardi and Wilke is very similar to the Fribourg construction that we describe in Chapter 2. An obvious difference is that the Fribourg construction uses right-to-left, rather than left-to-right, reduced split trees. However, this is an arbitrary choice, and has no influence on the result of the constructions. Another difference is that the transition from the initial phase to the repetition phase is handled quite differently by Vardi and Wilke, than for the corresponding automata parts in the Fribourg construction.

Kähler and Wilke, 2008

The slice-based construction by Kähler and Wilke from 2008 [11] is a generalisation of the construction by Vardi and Wilke from 2007 [52]. Kähler and Wilke proposed a construction idea that can be used for both, complementation and disambiguation. Consequently, this construction is less efficient than Vardi and Wilke's construction. It has a worst-case state complexity of $4(3n)^n$ [42].

A comparison of the rank-based and slice-based complementation approaches has been done by Fogarty, Kupferman, Wilke, and Vardi [5]. In this work, the authors also describe a translation of the slice-based construction by Kähler and Wilke [11] to a rank-based construction.

1.4 Empirical Performance Investigations

1.5 Preliminaries

1.5.1 Büchi Automata

Büchi automata have been introduced in 1962 by Büchi [4] in order to show the decidability of monadic second order logic; over the successor structure of the natural numbers [3].

he had proved the decidability of the monadic-second order theory of the natural numbers with successor func- tion by translating formulas into finite automata [52] (p. 1)

Büchi needed to create a complementation construction (proof the closure under complementation of Büchi automata) in order to prove Büchi's Theorem.

Büchi's Theorem: S1S formulas and Büchi automata are expressively equivalent (there is a NBW for every S1S formula, and there is a S1S formula for every NBW).

Definitions

Informally speaking, a Büchi automaton is a finite state automaton running on input words of infinite length. That is, once started reading a word, a Büchi automaton never stops. A word is accepted if it results in a run (sequence of states) of the Büchi automaton that includes infinitely many occurrences of at least one accepting state.

More formally, a Büchi automaton A is defined by the 5-tuple $A = (Q, \Sigma, q_0, \delta, F)$ with the following components.

- Q: a finite set of states
- Σ: a finite alphabet
- q_0: an initial state, $q_0 \in Q$
- δ: a transition function, $\delta : Q \times \Sigma \to 2^Q$
- F: a set of accepting states, $F \in 2^Q$

We denote by Σ^ω the set of all words of infinite length over the alphabet Σ. A Büchi automaton runs on the elements of Σ^ω. In the following, we define the acceptance behaviour of a Büchi automaton A on a word $\alpha \in \Sigma^\omega$.

- A *run* of Büchi automaton A on a word $\alpha \in \Sigma^\omega$ is a sequence of states $q_0 q_1 q_2 \dots$ such that q_0 is A's initial state and $\forall i \geq 0 : q_{i+1} \in \delta(q_i, \alpha_i)$
- $\inf(\rho) \in 2^Q$ is the set of states that occur infinitely often in a run ρ
- A run ρ is accepting if an only if $\inf(\rho) \cap F \neq \varnothing$
- A Büchi automaton A accepts a word $\alpha \in \Sigma^\omega$ if and only if there is an accepting run of A on α

The set of all the words that are accepted by a Büchi automaton A is called the *language* $L(A)$ of A. Thus, $L(A) \subseteq \Sigma^\omega$. On the other hand, the set of all words of Σ^ω that are rejected by A is called the *complement language* $\overline{L(A)}$ of A. The complement language can be defined as $\overline{L(A)} = \Sigma^\omega \setminus L(A)$.

Büchi automata are closed under union, intersection, concatenation, and complementation [49].

Continued/discontinued runs

A deterministic Büchi automaton (DBW) is a special case of a non-determnistic Büchi automaton (NBW). A Büchi automaton is a DBW if $|\delta(q, \alpha)| = 1, \forall q \in Q, \forall \alpha \in \Sigma$. That is, every state has for every alphabet symbol exactly one successor state. A DBW can also be defined directly by replacing the transition function $\delta : Q \times \Sigma \to 2^Q$ with $\delta : Q \times \Sigma \to Q$ in the above definition.

Expressiveness

It has been showed by Büchi that NBW are expressively equivalent the ω-regular languages [4]. That means that every language that is recognised by a NBW is a ω-regular language, and on the other hand, for every ω-regular language there exists a NBW recognising it.

However, this equivalence does not hold for DBW (Büchi showed it too). There are ω-regular languages that cannot be recognised by any DBW. A typical example is the language $(0 + 1)^*1^\omega$. This is the language of all infinite words of 0 and 1 with only finitely many 0. It can be shown that this language can be recognised by a NBW (it is thus a ω-regular language) but not by a DBW [49][32]. The class of languages recognised by DBW is thus a strict subset of ω-regular languages recognised by NBW. We say that DBW are less expressive than NBW.

An implication of this is that there are NBW for which no DBW recognising the same language exists. Or in other words, there are NBW that cannot be converted to DBW. Such an inequivalence is not the case, for example, for finite state automata on finite words, where every NFA can be converted to a DFA with the subset construction [10][29]. In the case of Büchi automata, this inequivalence is the main cause that Büchi complementation problem is such a hard problem [25] and until today regarded as unsolved.

1.5.2 Other ω-Automata

After the introduction of Büchi automata in 1962, several other types of ω-automata have been proposed. The best-known ones are by Muller (Muller automata, 1963) [22], Rabin (Rabin automata, 1969) [30], Streett (Streett automata, 1982) [38], and Mostowski (parity automata, 1985) [21].

All these automata differ from Büchi automata, and among each other, only in their acceptance condition, that is, the condition for accepting or rejecting a run ρ. We can write a general definition of ω-automata that covers all of these types as $(Q, \Sigma, q_0, \delta, Acc)$. The only difference to the 5-tuple defining Büchi automata is the last element, Acc, which is a general acceptance condition. We list the acceptance condition of all the different ω-automata types below [18]. Note that again a run ρ is a sequence of states, and $\inf(\rho)$ is the set of states that occur infinitely often in run ρ.

Type	**Definitions**	**Run ρ accepted if and only if...**
Büchi	$F \subseteq Q$	$\inf(\rho) \cap F \neq \varnothing$
Muller	$F \subseteq 2^Q$	$\inf(\rho) \in F$
Rabin	$\{(E_1, F_1), \dots, (E_r, F_r)\}$, $E_i, F_i \subseteq Q$	$\exists i : \inf(\rho) \cap E_i = \varnothing \wedge \inf(\rho) \cap F_i \neq \varnothing$
Streett	$\{(E_1, F_1), \dots, (E_r, F_r)\}$, $E_i, F_i \subseteq Q$	$\forall i : \inf(\rho) \cap E_i \neq \varnothing \vee \inf(\rho) \cap F_i = \varnothing$
Parity	$c : Q \to \{1, \dots, k\}$, $k \in \mathbb{N}$	$\min\{c(q) \mid q \in \inf(\rho)\} \bmod 2 = 0$

In the Muller acceptance condition, the set of infinitely occuring states of a run ($\inf(\rho)$) must match a predefined set of states. The Rabin and Streett conditions use pairs of state sets, so-called accepting pairs. The Rabin and Streett conditions are the negations of each other. This allows for easy complementation of deterministic Rabin and Streett automata [18], which will be used for certain Büchi complementation construction, as we will see in Section1.8. The parity condition assigns a number (color) to each state and accepts a run if the smallest-numbered of the infinitely often occuring states has an even number. For all of these automata there exist non-deterministic and deterministic versions, and we will refer to them as NMW, DMW (for non-deterministic and deterministic Muller automata), and so on.

In 1966, McNaughton made an important proposition, known as *McNaughton's Theorem* [19]. Another proof given in [39]. It states that the class of languages recognised by deterministic Muller automata are the ω-regular languages. This means that non-deterministic Büchi automata and deterministic Muller automata are equivalent, and consequently every NBW can be turned into a DMW. This result is the base for the determinisation-based Büchi complementation constructions, as we will see in Section 1.8.2.

It turned out that also all the other types of the just introduced ω-automata, non-deterministic and determinstic, are equivalent among each other [32][14][13][18][39]. This means that all the ω-automata mentioned in this thesis, with the exception of DBW, are equivalent and recognise the ω-regular languages. This is illustrated in Figure 1.1

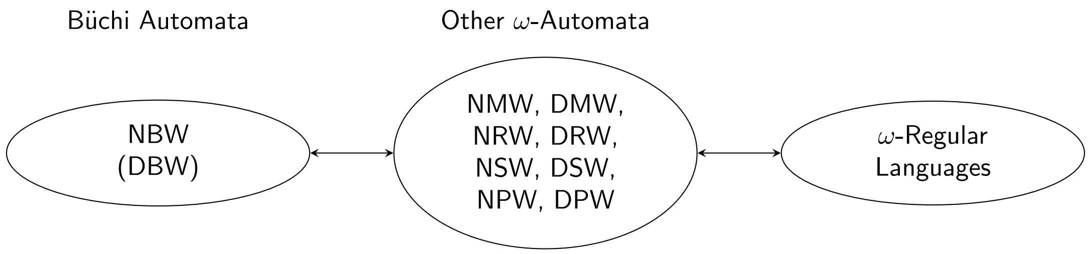


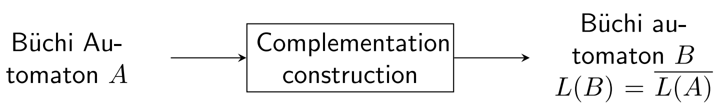
Figure 1.1: Non-deterministic Büchi automata (NBW) are expressively equivalent to Muller, Rabin, Streett, and parity automata (both deterministic and non-deterministic), and to the ω-regular languages. Deterministic Büchi automata (DBW) are less expressive than NBW.

1.5.3 Complementation of Büchi Automata

Büchi automata are closed under complementation. This result has been proved by Büchi himself when he introduced Büchi automata in [4]. Basically, this means that for every Büchi automata A, there exists another Büchi automaton B that recognises the complement language of A, that is, $L(B) = \overline{L(A)}$.

It is interesting to see that this closure does not hold for the specific case of DBW. That means that while for every DBW a complement Büchi automaton does indeed exist, following from the above closure property for Büchi automata in general, this automaton is not necessarily a DBW. The complement of a DBW may be, and often is, as we will see, a NBW. This result is proved in [39] (p. 15).

The problem of Büchi complementation consists now in finding a procedure (usually called a construction) that takes as input any Büchi automaton A and outputs another Büchi automaton B with $L(B) = \overline{L(A)}$, as shown below.



For complementation of automata in general, construction usually differ depending on whether the input automaton A is deterministic or non-deterministic. Complementation of deterministic automata is often simpler and may sometimes even provide a solution for the complementation of the non-deterministic ones.

To illustrate this, we can briefly look at the complementation of the ordinary finite state automata on finite words (FA). FA are also closed under complementation [10] (p. 133). A DFA can be complemented by simply switching its accepting and non-accepting states [10] (p. 133). Now, since NFA and DFA are equivalent [10] (p. 60), a NFA can be complemented by converting it to an equivalent DFA first, and then complement this DFA. Thus, the complementation construction for DFA provides a solution for the complementation of NFA.

Returning to Büchi automata, the case is more complicated due to the inequivalence of NBW and DBW. The complementation of DBW is indeed "easy", as was the complementation of DFA. There is a construction, introduced in 1987 by Kurshan [17], that can complement a DBW to a NBW in polynomial time. The size of the complement NBW is furthermore at most the double of the size of the input DBW.

If now for every NBW there would exist an equivalent DBW, an obvious solution to the general Büchi complementation problem would be to transform the input automaton to a DBW (if it is not already a

DBW) and then apply Kurshan's construction to the DBW. However, as we have seen, this is not the case. There are NBW that cannot be turned into equivalent DBW.

Hence, for NBW, other ways of complementing them have to be found. In the next section we will review the most important of these "other ways" that have been proposed in the last 50 years since the introduction of Büchi automata. The Fribourg construction, that we present in Chapter **??**, is another alternative way of achievin this same aim.

1.5.4 Complexity of Büchi Complementation

Constructions for complementing NBW turned out to be very complex. Especially the blow-up in number of states from the input automaton to the output automaton is significant. For example, the original complementation construction proposed by Büchi [4] involved a doubly exponential blow-up. That is, if the input automaton has n states, then for some constant c the output automaton has, in the worst case, c^{c^n} states [37]. If we set c to 2, then an input automaton with six states would result in a complement automaton with about 18 quintillion (18×10^{18}) states.

Generally, state blow-up functions, like the c^{c^n} above, mean the absolute worst cases. It is the maximum number of states a construction *can* produce. For by far most input automata of size n a construction will produce much fewer states. Nevertheless, worst case state blow-ups are an important (the most important?) performance measure for Büchi complementation constructions. A main goal in the development of new constructions is to bring this number down.

A question that arises is, how much this number can be brought down? Researchers have investigated this question by trying to establish so called lower bounds. A lower bound is a function for which it is proven that no state blow-up of any construction can be less than it. The first lower bound for Büchi complementation has been established by Michel in 1988 at $n!$ [20]. This means that the state blow-up of any Büchi complementation construction can never be less than $n!$.

There are other notations that are often used for state blow-ups. One has the form $(xn)^n$, where x is a constant. Michel's bound of $n!$ would be about $(0.36n)^n$ in this case [54]. We will often use this notation, as it is convenient for comparisons. Another form has 2 as the base and a big-O term in the exponent. In this case, Michel's $n!$ would be $2^{O(n\,log\,n)}$ [54].

Michel's lower bound remained valid for almost two decades until in 2006 Yan showed a new lower bound of $(0.76n)^n$ [54]. This does not mean that Michel was wrong with his lower bound, but just too reserved. The best possible blow-up of a construction can now be only $(0.76n)^n$ and not $(0.36n)^n$ as believed before. In 2009, Schewe proposed a construction with a blow-up of exactly $(0.76n)^n$ (modulo a polynomial factor) [35]. He provided thus an upper bound that matches Yan's lower bound. The lower bound of $(0.76n)^n$ can thus not rise any further and seems to be definitve.

Maybe mention note on exponential complexity in [49] p. 8.

1.6 Run Analysis

A deterministic automaton has exactly one run on every word. A non-deterministic automaton, on the other hand, may have multiple runs on a given word. The analysis of all runs of a word, in some form or another, an integral part of Büchi complemenation constructions. Remember that a non-deterministic automaton accepts a word if there is *at least one* accepting run. Consequently, a word is rejected if only if *all* the runs are rejecting. That is, if B is the complement Büchi automaton of A, then B has to accept a word w if and only if *all* the runs of A on w are rejecting. For constructing the complement B, we have thus to consider all the possible runs of A on every word.

There are two main data structures that are used for analysing the runs of a non-deterministic automaton on a word. These are trees and DAGs (directed acyclic graphs) [53]. In this section, we present both of them. We put however emphasis on trees, as they are used by the subset-tuple construction presented in Chapter **??**.

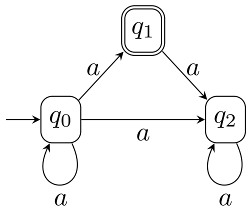


Figure 1.2: Example NBW A that will be used in different places throughout this thesis. The alphabet of A consists of the single symbol a, consequently, A can only process the single ω-word a^ω. This word is rejected by A, so the automaton is empty.

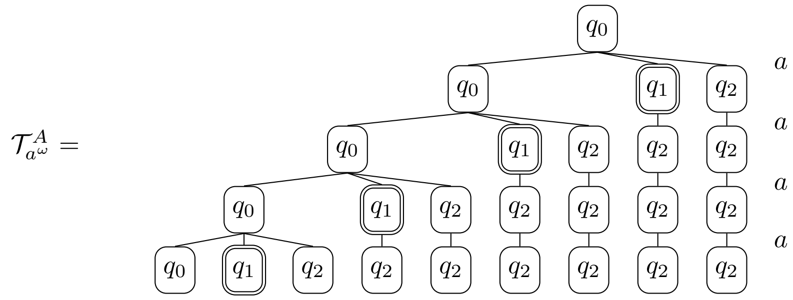


Figure 1.3: Automaton A and the first five levels of the run tree of the runs of A on the word a^ω.

1.6.1 From Run Trees to Split Trees

The one tree data-structure that truly represents *all* the runs of an automaton on a word are run trees. The other variants of trees that we present in this section are basically derivations of run trees that sacrifice information about individual runs, by merging or discarding some of them, at the benefit of becoming more concise. Figure 1.8 shows the first few levels of the run tree of the example automaton A from Figure 1.2 on the word a^ω.

In a run tree, every vertex represents a single state and has a descendant for every a-successor of this state, if a is the current symbol of the word. A run is thus represented as a branch of the run tree. In particular, there is a one-to-one mapping between branches of a run tree and runs of the automaton on the given word.

We mentioned that the other tree variants that we talk about in this section, split trees and reduced split trees, make run trees more compact by not keeping information about individual runs anymore. They thereby relinquish the one-to-one mapping between branches of the tree and runs. Let us look at one extreme of this aggregation of runs which is done by the subset construction. This will motivate the definition of split trees, and at the same time shows why the subset construction fails for determinising NBW [1].

For determinising an automaton A, the subset construction in effect merges all the diverse runs of A on word w to one single run by merging all the states on a level of the corresponding run tree to one single state. This state will be a state of the output automaton B, and is labelled with the set of A-states it includes. Figure 1.4 shows this effect with our example automaton from Figure 1.2 and the word a^ω.

Clearly, this form of tree created by the subset construction is the most concise form a run tree can be brought to. However, almost all information about individual runs in A has been lost. All that can be said by looking at the structure in Figure 1.4 is that there must be at least one continued A-run on a^ω(all the other runs visiting the other A-states on each level, might be discontinued). But which states a possible continued run visits cannot be deduced.

This lack of identification A-runs is the reason why the subset construction fails for determinising Büchi automata. Note that a B-state of the subset construction is accepting if the set of A-states it represents contains at least one accepting A-state. For our example, this means that the state $\{q_0, q_1, q_2\}$ is accepting

[1]The NBW that *can* be turned into DBW.

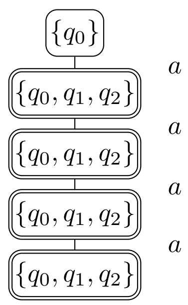


Figure 1.4: Automaton A and the first five levels of the run tree of the runs of A on the word a^ω.

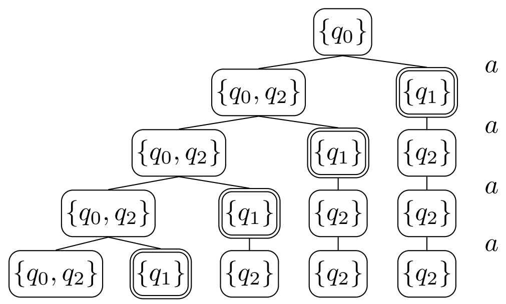


Figure 1.5: Automaton A and the first five levels of the split tree of the runs of A on the word a^ω.

(this is also indicated in Figure 1.4). This state is visited infinitely often by the unified run on a^ω. Hence, the DBW B, resulting from applying the subset construction to the NBW A, accepts a^ωwhile A does not accept it.

By looking closer at the trees in Figure 1.4 and 1.8, the reason for this problem becomes apparent. If we look for example at the second level of the subset-construction tree we can deduce that there must be an A-run that visits the accepting A-state q_1. Let us call this run r_{q_1}. However, at the third level, we cannot say anything about r_{q_1} anymore, whether it visits one of the non-accepting states or again q_1 on the third level, or whether it even ended at the second level. In turn, what we know on the third level in our example is that there is again an A-run, r'_{q_1}, that visits q_1. However, whether r'_{q_1} is r_{q_1}, and in turn the future of r'_{q_1} cannot be deduced. In our example we end up with the situation that there are infintiely many visits to q_1 in the unifed B-run, but we don't know if the reaon for this are one or more A-runs that visit q_1 infinitely often, or infinitely man A-runs where each one visits q_1 only finitely often (the way it is in our example). In the first case, it would be correct ot accept the B-run, in the second case however it would be wrong as the input automaton A does not accept the word. The subset construction does not distinguish these two cases and hence the determinised automaton B may accept words that the input automaton A rejects. In general, the language of an output DBW of the subset construction is a superset of the language of the input NBW.

This raises the question how the subset construction can be minimally modified such that the output automaton is equivalent to the input automaton. One solution is to not mix accepting and non-accepting A-states in the B-states. That is, instead of creating one B-state that contains all the A-states, as in the subset construction, one creates two B-states where one contains the accepting A-states and the other the non-accepting A-states. Such a construction has been formalised in [48]. The output automaton B is then not deterministic, but it is equivalent to A. The type of run analysis trees that correspond to this refined subset construction are split trees. Figure 1.9 shows the first five levels of the split tree of our example automaton A on the word a^ω.

Let us see why the splitted subset construction produces output automata that are equivalent to the input automata. For this equivalence to hold, a branch of a reduced split tree must include infinitely many accepting vertices if and only if there is an A-run that visits at least one accepting A-state infinitely often. For an infinite branch of a split tree, there must be at least one continued A-run. If this infinite branch includes infinitely many accepting vertices, then this A-run must infinitely many times go trough an accepting A-state. This is certain, because an accepting vertex in a split tree contains *only* accepting

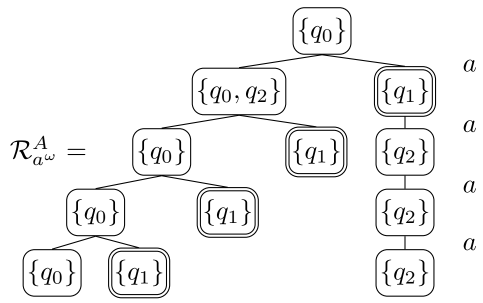


Figure 1.6: Automaton A and the first five levels of the reduced split tree of the runs of A on the word a^ω.

A-states. Since there are only finitely many accepting A-states, the A-run must visit at least one of them infinitely often. On the other hand, if an A-run includes infinite visits to an accepting state, then this results in a branch of the split tree with infinitely many accepting vertices, since every A-run must be "contained" in a branch of the split tree.

Split trees can be seen as run trees where some of the branches are contracted to unified branches. In particular, a split tree unifies as many branches as possible, such that the resulting tree still correctly represents the Büchi acceptance of all the runs included in a unified branch. This can form the basis for constructions that transform an NBW to another equivalent NBW. Split trees are for example the basis for Muller-Schupp trees in Muller and Schupp's Büchi determinisation construction [24], cf. [2].

1.6.2 Reduced Split Trees

It turns out that split trees can be compacted even more. The resulting kind of tree is called reduced split tree. In a reduced split tree, each A-state occurs at most once on every level. Figure 1.6 shows the reduced split tree corresponding to the split tree in Figure 1.9. As can be seen, only one occurrence of each A-state on each level is kept, the other are discarded. To allow this, however, the order of the accepting and non-accepting siblings in the tree matters. Either the accepting child is always put to the right of the non-accepting child (as in our example in Figure 1.6, or vice versa. We call the former variant a right-to-left reduced split tree, and the latter a left-to-right reduced split tree. In this thesis, we will mainly adopt the right-to-left version.

A reduced split tree is constructed like a split tree, with the following restrictions.

- For determining the vertices on level $n+1$, the parent vertices on level n have to be processed from right to left
- From every child vertex on level $n+1$, subtract the A-states that occur in some vertex to the right of it on level $n+1$
- Put the accepting child to the right of the non-accepting child on level $n+1$

A very important property of reduced split trees is that they have a fixed width. The width of a tree is the maximal number of vertices on a level. For reduced split trees, this is the number of states of the input automaton A. As we will see, the subset-tuple construction (like other slice-based constructions) uses levels of a reduced split tree as states othe output automaton, and the limited size of these levels ensures an upper bound on the number of states these constructions can create.

By deleting A-states from a level of a reduced split tree, we actually delete A-runs that reach the same A-state on the same substring of the input word. For example, in the split tree in Figure 1.9 we see that there are at least for A-runs on the string *aaaa* from the inital state q_0 to q_2. The reduced split tre in Figure 1.6, however, contains only one run on *aaaa* from q_0 to q_2, namely the rightmost branch of the tree. The information about all the other runs is lost. This single run that is kept is very special and, as we will see shorty, it represents the deleted runs. We will call this run the *greedy run*. The reason for calling it greedy is that it visits an accepting state earlier than any of the deleted runs. In a right-to-left reduced split tree, the greedy run is always the rightmost of the runs from the root to a certain A-state

$\mathcal{G}^A_{a^\omega} =$

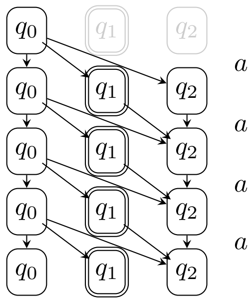


Figure 1.7: Automaton A and the first five levels of the run DAG of the runs of A on the word a^ω.

on a certain level. In left-to-right reduced split tree, the greedy run would in turn be the leftmost of these runs.

We mentioned that the greedy run somehow represents the deleted runs. More precisely, the relation is as follows and has been proved in [52]: if any of the deleted runs is a prefix of a run that is Büchi-accepted (that is, an infinite run visiting infinitely many accepting A-states), then the greedy run is so too. That means that if the greedy cannot be expanded to a Büchi-accepting run, then none of the deleted runs could be either. Conversely, if any of the deleted runs could become Büchi-accepting, then the greedy run can so too. So, the greedy run is sufficient to indicate the existence or non-existence of a Büchi-accepting run with this prefix, and it is safe to delete all the other runs.

1.6.3 Run DAGs

DAGs (directed acyclic graphs) are, after trees, the second form for analysing the runs of a non-deterministic automaton on a given word. A run DAG has the form of a matrix with one column for each A-state and a row for each position in the word. The directed edges go from the vertices on one row to the vertices on the next row (drawn below) according to the transitions in the automaton on the current input symbol. Figure 1.12 shows the first five rows of the run DAG of the example automaton in Figure 1.2 on the word a^ω.

Like run trees, run DAGs represent all the runs of an automaton on a given word. However, run DAGs are more compact than run trees. The rank-based complementation constructions are based on run DAGs.

1.7 Run Analysis

In a deterministic automaton every word has exactly one run. In a non-deterministic automaton, howevever, a given word may have multiple runs. The analysis of the different runs of a given word on an automaton plays an important role in the complementation of Büchi automata. There are several techniques for analysing the runs of a word that we present in this section.

1.7.1 Run Trees

The simplest of run analyis technique is the run tree. A run tree is a direct unfolding of all the possible runs of an automaton A on a word w. Each vertex v in the tree represents a state of A that we denote by $\sigma(v)$. The descendants of a vetex v on level i are vertices representing the successor states of $\sigma(n)$ on the symbol $w(i+1)$ in A. In this way, every branch of the run tree originating in the root represents a possible run of automaton A on word w.

Figure 1.8 shows an example automaton A and the first five levels of the run tree for the word $w = a^\omega$ (infinite repetitions of the symbol a). Each branch from the root to one of the leaves represents a possible way for reading the first four positions of w. On the right, as a label for all the edges on the corresponding level, is the symbol that causes the depicted transitions.

$\mathcal{T}^A_{a^\omega} =$

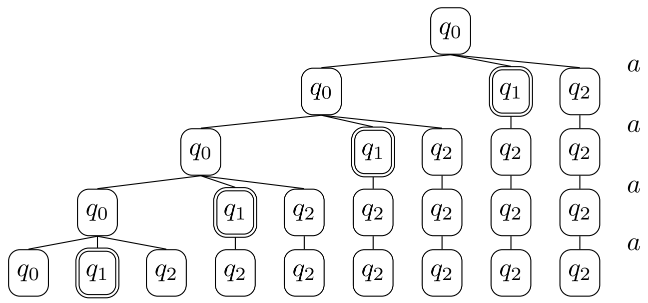


Figure 1.8: Automaton A and the first five levels of the run tree of the runs of A on the word a^ω.

(A does not accept any word, it is empty. The only word it could accept is a^ω which it does not accept.)

We define by the width of a tree the maximum number of vertices occurring at any level [24]. Clearly, for ω-words the width of a run tree may become infinite, because there may be an infinite number of levels and each level may have more vertices than the previous one.

1.7.2 Failure of the Subset-Construction for Büchi Automata

Run trees allow to conveniently reveal the cause why the subset construction does not work for determinising Büchi automata, which in turn motivates the basic idea of the next run analysis technique, split trees.

Applying the subset construction to the same NBW A used in the previous example, we get the automaton A' shown in Figure **??**. Automaton A' is indeed a DBW but it accepts the word a^ω which A does not accept. If we look at the run tree of A on word a^ω, the subset construction merges the individual states occuring at level i of the tree to one single state s_i, which is accepting if at least one of its components is accepting. Equally, the individual transitions leading to and leaving from the individual components of s_i are merged to a unified transition. The effect of this is that we lose all the information about these indiviudal transitions. This fact is depicted in Figure **??**. For the NFA acceptance condition this does not matter, but for NBW it is crucial because the acceptance condition depends on the history of specific runs. In the example in Figure **??**, a run ρ of A visiting the accepting state q_1 can never visit an accepting state anymore even though the unified run of which ρ is part visits q_1 infinitely often. But the latter is achieved by infinitely many different runs each visiting q_1 just once.

It turns out that enough information about individual runs to ensure the Büchi acceptance condition could be kept, if accepting and non-accepting state are not mixed in the subset construction. Such a constructio has bee proposed in [48]. Generally, the idea of treating accepting and non-accepting states separately is important in the run analyis of Büchi automata.

1.7.3 Split Trees

Split trees can be seen as run trees where the accepting and non-accepting descendants of a node n are aggregated in two nodes. We will call the former the *accepting child* and the latter the *non-accepting child* of n. Thus in a split tree, every node has at most two descendants (if either the accepting or the non-accepting child is empty, it is not added to the tree), and the nodes represent sets of states rather than individual states. Figure 1.9 shows the first five levels of the split tree of automaton A on the word a^ω.

The order in which the accepting and non-accepting child are

The notion of split trees (and reduced split trees, see next section) has been introduced by Kähler and Wilke in 2008 for their slice-based complementation construction [11], cf. [5]. However, the idea of separating accepting from non-accepting states has already been used earlier, for example in Muller and Schupp's determinisation-based complementation construction from 1995 [24]. Formal definitions os split trees can be found in [11][5].

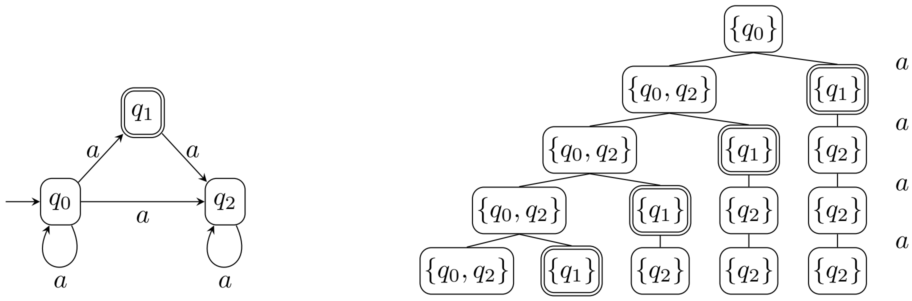


Figure 1.9: Automaton A and the first five levels of the split tree of the runs of A on the word a^ω.

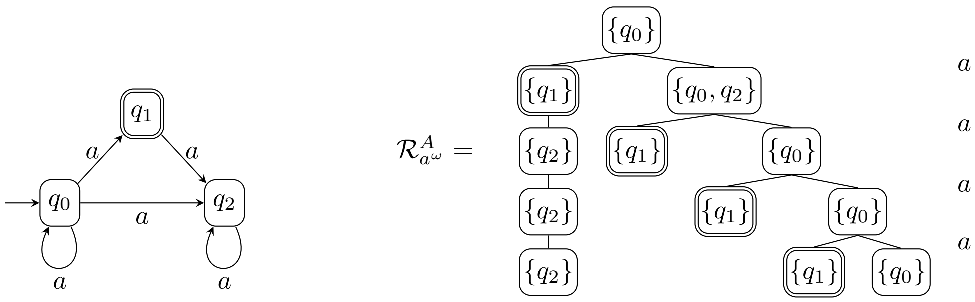


Figure 1.10: Automaton A and the first five levels of the left-to-right reduced split tree of the runs of A on the word a^ω.

1.7.4 Reduced Split Trees

The width of a split tree can still become infinitely large. A reduced split tree limits this width to a finite number with the restriction that on any level a given state may occur at most once. This is in effect the same as saying that if in a split tree there are multiple ways of going from the root to state q, then we keep only one of them.

1.7.5 Run DAGs

A run DAG (DAG stands for directed acyclic graph) can be seen as a graph in matrix form with one column for every state of A and one row for every position of word w. The edges are defined similarly than in run trees. Figure 1.12 shows the run DAG of automaton A on the word $w = a^\omega$.

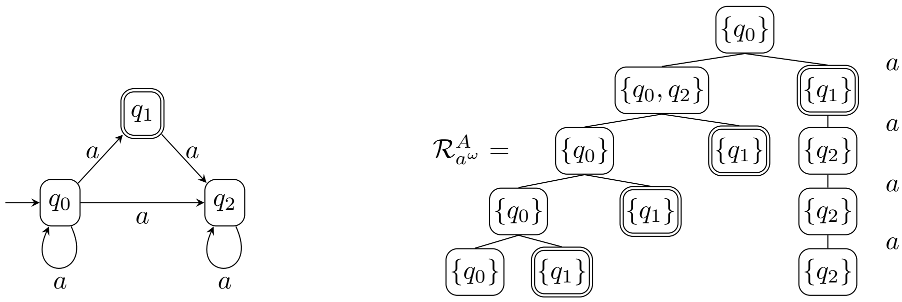


Figure 1.11: Automaton A and the first five levels of the left-to-right reduced split tree of the runs of A on the word a^ω.

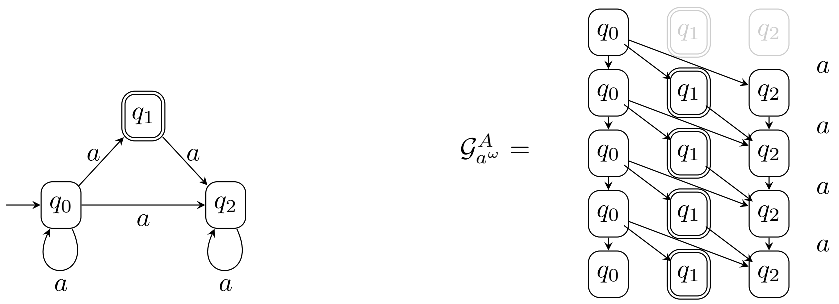


Figure 1.12: Automaton A and the first five levels of the run DAG of the runs of A on the word a^ω.

1.8 Review of Büchi Complementation Constructions

1.8.1 Ramsey-Based Approaches

The method is called Ramsey-based because its correctness relies on a combinatorial result by Ramsey to obtain a periodic decomposition of the possible behaviors of a Büchi automaton on an infinite word [3].

1.8.2 Determinisation-Based Approaches

1.8.3 Rank-Based Approaches

1.8.4 Slice-Based Approaches

1.9 Empirical Performance Investigations

Chapter 2

The Fribourg Construction

Contents

2.1 Basics **23**
2.2 The Construction **23**
2.2.1 Upper Part 23
2.2.2 Lower Part 23
2.3 First Stage: Constructing the Upper Part **24**
2.4 Second Stage: Adding the Lower Part **25**
2.4.1 Construction 25
2.4.2 Meaning and Function of the Colours 27
2.5 Intuition for Correctness **27**
2.6 Optimisations **28**
2.6.1 Removal of Non-Accepting States (R2C) 28
2.6.2 Merging of Adjacent Sets (M1) 28
2.6.3 Reduction of 2-Coloured Sets (M2) 28

In this chapter we describe the Fribourg construction, the Büchi complementation construction developed at the University of Fribourg by Joel Allred and Ulrich Ultes-Nitsche. The construction has been published in 2014 as a technical report entitled “Complementing Büchi Automata with a Subset-tuple Construction” [1].

We do not give a formal description of the Fribourg construction in this chapter, because this has already been done in [1]. Rather, our aim is to give an intuitive and practically oriented description. That means, demonstrating the concrete steps one has to do when sitting with a pencil in front of an automaton to be complemented. Similarly, this chapter does not contain any proofs of the correctness or complexity of the constructions, because they can be found in [1].

This chapter is structured as follows. In Section **??** we present some basic properties of the Fribourg construction and put it in relation with other complementation constructions. In Section **??**, we describe the actual construction which consists of two stages, the construction of the upper part and the construction of the lower part. We present these two stages in separate sections, together with an example. Finally, in Section **??**, we describe three optimisations for the construction, that have the abbreviations R2C, M1, and M2. These optimisations will also be subject to our empirical performance investigation that we describe in the subsequent chapter.

A note on terminology: the authors themselves call their construction “subset-tuple construction”. This is because a state of the output automaton consists of a tuple of subsets of states of the input automaton. However, this is also the case for other constructions. To make our construction more distinguishable from the other constructions, we decided to use the more striking name “Fribourg construction”.

2.1 Basics

The Fribourg construction belongs to the slice-based complementation approach, that we described in Section 1.3.4. That means that it relies on the analysis of runs of the input automaton with reduced split trees.

The Fribourg construction, as described in [1], uses right-to-left reduced split trees. This is different from Vardi and Wilke’s [52] and Kähler and Wilke’s [11] slice-based constructions which use left-to-right reduced split trees. However, this choice is arbitrary, and it would be possible to describe the Fribourg construction with left-to-right reduced split trees. However, in this thesis, we will stick with the right-to-left reduced split trees for the Fribourg construction.

At the heart of the Fribourg construction is a mapping between states of the output automaton and levels of a reduced split tree. These levels are called “slices” according to the terminology of Vardi and Wilke [52]. Figure 2.1 shows how this translation works. A slice consists of a sequence of vertices, where each vertex consists of a subset of states of the initial automaton. Such a slice results in a state consisting of a tuple of subsets, where the subsets correspond to the vertices of the slice in the same order.

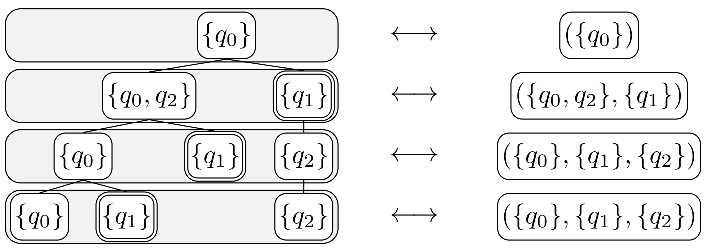


Figure 2.1: Translation from slices of a reduced split tree (shaded boxes on the left) to states of the Fribourg construction (right), and vice versa. A slice is represented as a tuple of subsets of input-states, where each subset corresponds to a vertex on the given level of the reduced split tree.

Upper part, lower part –¿ Kurshan’s construction is a special case of the Fribourg construction

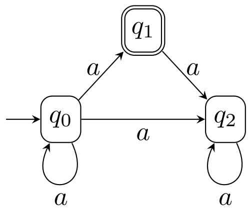


Figure 2.2: Example automaton.

2.2 The Construction

2.2.1 Upper Part

Description

- no accepting states - complete

Example

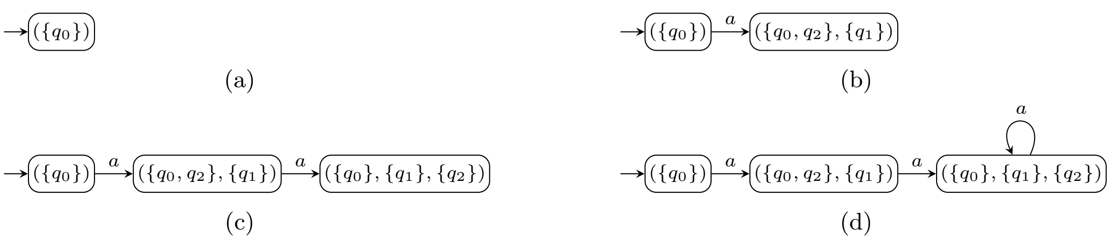


Figure 2.3: The final complement automaton B.

2.2.2 Lower Part

The Fribourg construction draws from several ideas: the subset construction, run analysis based on reduced split trees, and Kurshan's construction [17] for complementing DBW. Following the classification we used in Section 1.8, it is a slice-based construction. Some of its formalisations are similar to the slice-based construction by Vardi and Wilke [52], however, the Fribourg construction has been developed independently. Furthermore, as we will see in Chapter **??**, the empirical performance of Vardi and Wilke's construction and the Fribourg construction differ considerably, in favour of the latter.

Basically, the Fribourg construction proceeds in two stages. First it constructs the so-called upper part of the complement automaton, and then adds to it its so-called lower part. These terms stem from the fact that it is often convenient to draw the lower part below the previously drawn upper part. The partitioning in these two parts is inspired by Kurshan's complementation construction for DBW. The upper part of the Fribourg construction contains no accepting states and is intended to model the finite "start phase" of a run. At every state of the upper part, a run has the non-deterministic choice to either stay in the upper part or to move to the lower part. Once in the lower part, a run must stay there forever (or until it ends if it is discontinued). That is, the lower part models the infinite "after-start phase" of a run. The lower part now includes accepting states in a sophisticated way so that at least one run on word w will be accepted if and only if all the runs of the input NBW on w are rejected.

As it may be apparent from this short summary, the construction of the lower part is much more involved than the construction of the upper part.

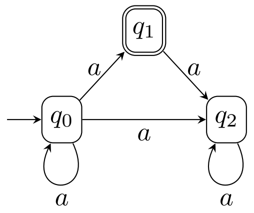


Figure 2.4: Example automaton A

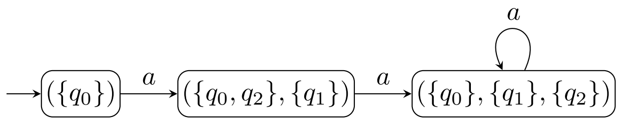


Figure 2.5: Upper part $B'$of example automaton A.

2.3 First Stage: Constructing the Upper Part

The first stage of the subset-tuple construction takes as input an NBW A and outputs a deterministic automaton B'. This $B'$is the upper part of the final complement automaton B of A. The construction of B'can be seen as a modified subset construction. The difference to the normal subset construction lies in the inner structure of the constructed states. While in the subset construction a state consists of a subset of the states of the input automaton, a B'-state in the subset-tuple construction consists of a *tuple of subsets* of A-states. The subsets in a tuple are pairwise disjoint, that is, every A-state occurs at most once in a B'-state. The A-states occurring in a B'-state are the same that would result from the classic subset construction. As an example, if applying the subset construction to a state $\{q_0\}$ results in the state $\{q_0, q_1, q_2\}$, the subset-tuple construction might yield the state $(\{q_0, q_2\}, \{q_1\})$ instead.

The structure of B'-states is determined by levels of corresponding reduced split trees. Vardi, Kähler, and Wilke refer to these levels as *slices* in their constructions [52, 11]. Hence the name slice-based approach. In the following, we will use the terms levels and slices interchangebly. A slice-based construction can work with either left-to-right or right-to-left reduced split trees. Vardi, Kähler, and Wilke use the left-to-right version in their above cited publications. In this thesis, in contrast, we will use right-to-left reduced split trees, which were also used from the beginning by the authors of the subset-tuple construction.

Figure **??** shows how levels of a right-to-left reduced split tree map to states of the subset-tuple construction. In essence, each node of a level is represented as a set in the state, and the order of the nodes determines the order of the sets in the tuple. [INFORMATION ABOUT ACC AND NON-ACC IS NEEDED IN THE LOWER PART BUT IMPLICIT IN THE STATES OF A]. To determine the successor of a state, say $(\{q_0, q_2\}, \{q_1\})$, one can regard this state as level of a reduced split tree, determine the next level and map this new level to a state. In the example of Figure **??**, the successor of $(\{q_0, q_2\}, \{q_1\})$ is determined in this way to $(\{q_0\}, \{q_1\}, \{q_2\})$.

Apart from this special way of determining successor states, the construction of B'proceeds similarly as the subset construction. One small further difference is that if at the end of determining a successor for every state in B', the automaton is not complete, it must be made complete with an *accepting* sink state. The steps for constructing $B'$from A can be summarised as follows.

- Start with the state $(\{q_0\})$ if q_0 is the initial state of A
- Determine for each state in B'a successor for every input symbol
- It at the end $B'$is not complete, make it complete with an accepting sink state

For the example automaton A in Figure 2.4, we would start with $(\{q_0\})$, determine $(\{q_0, q_2\}, \{q_1\})$ as its a-successor, whose a-successor in turn we determine a $(\{q_0\}, \{q_1\}, \{q_2\})$. The a-successor of $(\{q_0\}, \{q_1\}, \{q_2\})$ is $(\{q_0\}, \{q_1\}, \{q_2\})$ again what results in a loop. Figure 2.5 shows the final upper part $B'$of A.

2.4 Second Stage: Adding the Lower Part

The second stage of the subset-tuple construction adds the lower part to the upper part B'. The two parts together form the final complement automaton B. The lower part is constructed by again applying a modified subset construction to the states of the upper part B'. This modified subset construction is an extension of the construction for the upper part. The addition is that each set gets decorated with a colour. These colours later determine which states of the lower part are accepting states.

We divide our discussion of the lower part in two sections. In the following one (2.4.1), we explain the "mechanical" construction of the lower part, the steps that have to be done to arrive at the final complement automaton B. In the next section (2.5) we give the idea and intuition behind the construction and explain why it works.

2.4.1 Construction

As mentioned, every set of the states of the lower part gets a colour. There are three colours and we call them 0, 1, and 2. In the end we have to be able to disinguish the states of the upper part from the states of the lower part. This can be achieved by preliminarily assigning the special colour -1 to every set of the states of the upper part. After that the extended modified subset construction is applied, taking the states of the upper part (except a possible sink state) as the pre-existing states.

At first, the extended modified subset construction determines the successor tuple (without the colours) of an existing state in the same way as the construction of the upper part. We will refer to the state being created as p and to the existing state as p_{pred}. Then, one of the colours 0, 1, or 2 is determined for each set s of p. We denote the colour of s as $c(s)$. The choice of $c(s)$ depends on three factors.

- Whether p_{pred} has a set with colour 2 or not
- The colour of the predecessor set s_{pred} of s
- Whether s is an accepting or non-accepting set

The predecessor set s_{pred} is the set of p_{pred} that in the corresponding reduced split tree is the parent node of the node corresponding to s. Figure 2.6 shows the values of $c(s)$ for all possible situations as two matrices. There is one matrix for the two cases of factor 1 above (p_{pred} has colour 2 or not) and the other two factors are laid out along the rows and columns of either matrix. Note that $c(s_{pred}) = -1$ is only present in the upper matrix, because in this case p_{pred} is a state of the upper part and cannot contain colour 2.

We will use the following notation to denote the colour of s: $\widehat{s}$ if $c(s) = -1$, s if $c(s) = 0$, $\overline{s}$ if $c(s) = 1$, and $\overline{\overline{s}}$ if $c(s) = 2$. Let us look now at a concrete example of this construction. We will add the lower part to the upper part B'in Figure 2.5, and thereby complete the complementation of the example automaton A in Figure 2.4.

First of all, we assign colour -1 all the sets of the states of B'. We might then start processing the state $(\widehat{\{q_0\}})$, let us call it p_{pred}. The resulting successor tuple, without the colours, of p_{pred} is, as in the upper part, $(\{q_0, q_2\}, \{q_1\})$. We now have to determine the colours of the sets $\{q_0, q_2\}$ and $\{q_1\}$. Since p_{pred} does not contain any 2-coloured sets, we need only to consult the upper matrix in Figure 2.6. For $\{q_1\}$, the predecessor set is $\widehat{\{q_1\}}$ with colour -1. Furthermore $\{q_1\}$ is accepting. So, the colour of $\{q_1\}$ is 2, because we end up in the first-row, second-column cell of the upper matrix ($M_1(1, 2)$). The other set, $\{q_0, q_2\}$, in turn is non-accepting, so its colour is 0 ($M_1(1, 1)$). The successor state of $(\widehat{\{q_0\}})$ is thus $(\{q_0, q_2\}, \overline{\overline{\{q_1\}}})$.

We can then continue the construction right with this new state $(\{q_0, q_2\}, \overline{\overline{\{q_1\}}})$, and call it p_{pred} in turn. The successing tuple without the colours of p_{pred} is $(\{q_0\}, \{q_1\}, \{q_2\})$. Since p_{pred} contains a set with colour 2, we have to consult the lower matrix of Figure 2.6 to determine the colours of $\{q_0\}$, $\{q_1\}$, and $\{q_2\}$. For $\{q_2\}$, we end up with colour 2 ($M_2(3, 1)$), because its predecessor set, which is $\overline{\overline{\{q_1\}}}$, has colour 2. $\{q_1\}$ gets colour 1 as it is accepting and its predecessor set, $\{q_0, q_2\}$, has colour 0 ($M_2(1, 2)$). $\{q_0\}$,

Colour of s_{pred}	Colour of s, if s is non-accepting	Colour of s, if s is accepting
-1	**0**	**2**
0	**0**	**2**
1	**2**	**2**

(a) Case A: the predecessor state has *no* 2-coloured components

Colour of s_{pred}	Colour of s, if s is non-accepting	Colour of s, if s is accepting
0	**0**	**1**
1	**1**	**1**
2	**2**	**2**

(b) Case B: the predecessor state *has* 2-coloured components

Figure 2.6: Rules for determining the colour of a component c, based on (1) the colour of the predecessor component c_{pred}, and (2) whether c is an accepting or non-accepting component. There are two set of rules that are shown in the two subfigures: (a) the predecessor state does not have any components with colour 2, and (b) the predecessor state does have one or more components with colour 2.

which has the same predecessor set, gets colour 0, because it is non-accepting ($M_2(1,1)$). The successor state of $(\{q_0, q_2\}, \overline{\overline{\{q_1\}}})$ is thus $(\{q_0\}, \overline{\{q_1\}}, \overline{\overline{\{q_2\}}})$.

The construction continues in this way until every state has been processed. The resulting automaton is shown in Figure 2.7. The last thing that has to be done is to make every state of the lower part that does not contain colour 2 accepting. In our example, this is only one state. The NBW B in Figure 2.7 is the complement of the NBW A in Figure 2.4, such that $L(B) = \overline{L(A)}$. This can be easily verified, since A is empty and B is universal (with regard to the single ω-word a^ω).

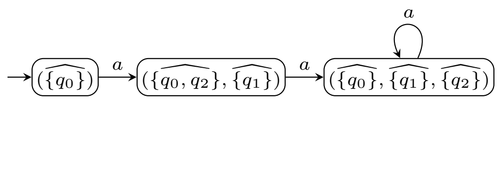
2.4.2 Meaning and Function of the Colours

2.5 Intuition for Correctness

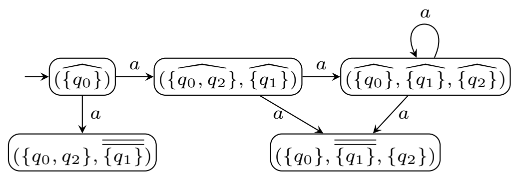
The general relation between a non-deterministic automaton A and its complement B is that a word w is accepted by B, if and only if all the runs of A on w are rejecting. Of course for the subset-tuple construction, as we have just described it above, this is also true. A formal proof can be found in [1]. In this section, in contrast, we try to give an intuitive way to understand this correctness. One one hand, there is the question, if there is an accepting run of B on w, how can we conclude that all the runs of A on w are rejected?

- If there is an accepting run of B on w, how can we conclude that all the runs of A on w are rejected?
- If all the runs of A on w are rejected, how can we conclude that there must be an accepting run of B on w?

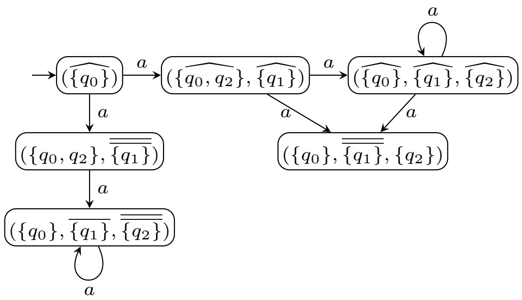
Since this condition is on *all* runs of A, the construction somehow has to keep track of them.



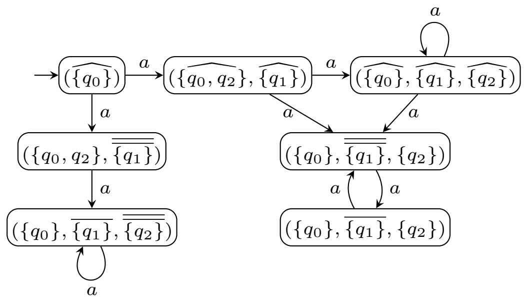
(a) Upper part



(b) Complete

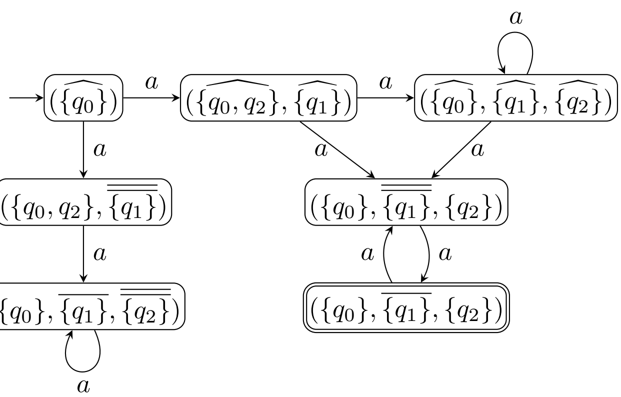


(c) Upper part



(d) Complete

Figure 2.7: The final complement automaton B.



2.6 Optimisations

2.6.1 Removal of Non-Accepting States (R2C)

2.6.2 Merging of Adjacent Sets (M1)

2.6.3 Reduction of 2-Coloured Sets (M2)

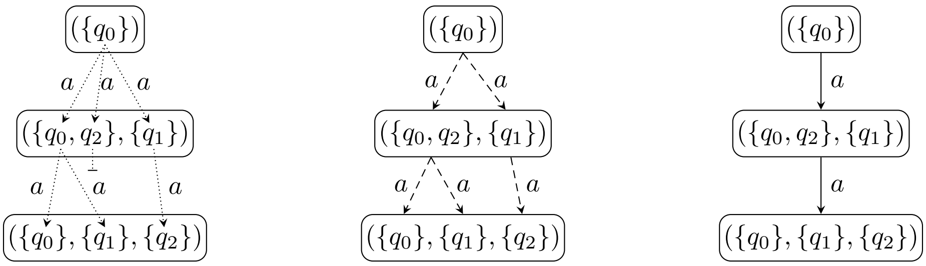


Figure 2.8: Different notions of runs.

Appendix A

Plugin Installation and Usage

Since between the 2014–08–08 and 2014–11–17 releases of GOAL certain parts of the plugin interfaces have changed, and we adapted our plugin accordingly, the currently maintained version of the plugin works only with GOAL versions 2014-11-17 or newer. It is thus essential for any GOAL user to update to this version in order to use our plugin.

Appendix B

Median Complement Sizes of the GOAL Test Set

Bla bla bla

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	269	308	254	236	238	297	266	156	207	68
1.2	960	1,407	1,479	2,150	1,152	1,090	942	1,206	718	104
1.4	3,426	2,915	2,752	3,393	2,693	3,265	2,263	2,425	1,844	154
1.6	3,799	3,698	4,901	3,926	3,960	3,655	2,580	1,905	2,124	155
1.8	3,375	3,169	3,420	3,967	3,943	3,132	2,246	1,144	971	114
2.0	1,906	2,261	2,383	2,884	2,354	2,096	1,169	932	568	98
2.2	1,467	1,633	1,795	1,942	1,611	1,640	569	499	330	78
2.4	924	1,232	1,319	1,317	1,056	886	514	314	182	59
2.6	625	763	880	945	828	684	316	175	132	44
2.8	483	584	836	690	575	395	240	151	103	41
3.0	319	450	557	523	367	313	155	116	84	32

(a) Fribourg

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	269	308	254	236	238	297	266	156	207	68
1.2	960	1,407	1,479	2,150	1,152	1,090	942	1,206	718	104
1.4	3,426	2,915	2,752	3,393	2,693	3,265	2,263	2,425	1,844	154
1.6	3,799	3,698	4,901	3,926	3,960	3,655	2,580	1,905	2,124	155
1.8	3,375	3,169	3,420	3,967	3,943	3,093	2,246	1,144	971	114
2.0	1,906	2,184	2,383	2,818	2,354	1,989	1,127	885	568	97
2.2	1,410	1,561	1,639	1,884	1,609	1,588	496	464	284	78
2.4	884	1,200	1,234	1,184	939	806	373	256	165	55
2.6	575	731	815	860	751	575	246	162	114	43
2.8	431	530	672	466	371	274	174	120	85	36
3.0	232	325	344	360	269	169	91	85	53	27

(b) Fribourg+R2C

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	390	438	434	324	328	459	337	204	227	40
1.2	1,576	2,394	2,505	2,996	1,613	1,551	1,166	1,542	1,002	58
1.4	5,007	4,336	4,652	4,877	3,458	3,956	3,169	3,380	1,868	86
1.6	5,067	5,032	6,444	4,868	4,575	3,864	3,211	1,731	1,892	85
1.8	4,016	3,701	3,647	4,523	3,548	3,009	1,808	451	336	62
2.0	1,663	2,276	2,676	3,035	1,925	1,932	464	307	150	54
2.2	989	1,514	1,621	1,826	1,121	846	155	127	93	45
2.4	560	821	919	771	529	267	133	87	55	32
2.6	388	519	524	441	259	219	84	50	41	26
2.8	311	317	396	242	165	95	64	44	33	22
3.0	173	224	211	169	102	72	41	34	27	18

(c) Fribourg+R2C+C

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	225	223	195	181	187	199	189	124	161	68
1.2	731	971	946	1,071	629	562	488	568	388	104
1.4	2,228	1,701	1,543	1,732	1,241	1,287	945	944	727	154
1.6	2,489	2,263	2,331	2,133	1,777	1,443	964	757	889	155
1.8	2,381	2,027	2,009	2,075	1,618	1,243	1,005	592	515	114
2.0	1,390	1,569	1,416	1,573	1,093	1,008	594	464	330	98
2.2	1,118	1,197	1,150	1,151	879	809	317	330	241	78
2.4	712	885	836	809	580	535	316	231	145	59
2.6	498	569	601	627	497	412	217	137	113	44
2.8	391	455	578	456	374	263	173	119	90	41
3.0	258	350	392	354	253	208	119	97	74	32

(d) Fribourg+M1

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	215	213	189	174	175	192	186	121	156	68
1.2	712	914	913	1,075	619	563	526	620	416	104
1.4	2,075	1,620	1,503	1,650	1,254	1,339	1,003	1,006	848	154
1.6	2,344	2,062	2,340	2,016	1,755	1,520	1,053	858	986	155
1.8	2,205	1,873	1,920	2,040	1,689	1,315	1,080	664	598	114
2.0	1,290	1,485	1,405	1,522	1,134	1,044	652	531	392	98
2.2	1,023	1,119	1,092	1,127	868	875	376	359	262	78
2.4	674	849	790	807	617	544	355	251	156	59
2.6	478	549	594	597	510	431	231	147	116	44
2.8	370	439	559	455	382	283	182	124	93	41
3.0	249	341	388	348	260	225	123	101	77	32

(e) Fribourg+M1+M2

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	225	223	195	181	187	199	189	124	161	68
1.2	731	971	946	1,071	629	562	488	568	388	104
1.4	2,228	1,701	1,543	1,732	1,241	1,287	945	944	727	154
1.6	2,489	2,263	2,331	2,133	1,777	1,443	964	757	889	155
1.8	2,381	2,027	2,009	2,075	1,618	1,215	1,005	592	515	114
2.0	1,390	1,513	1,416	1,542	1,093	1,003	594	441	330	97
2.2	1,019	1,156	1,064	1,104	859	785	304	303	221	78
2.4	672	867	789	772	544	478	269	191	139	55
2.6	466	542	572	568	452	348	183	129	99	43
2.8	368	407	480	337	260	197	129	96	75	36
3.0	201	261	266	272	199	136	83	74	50	27

(f) Fribourg+M1+R2C

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	329	303	279	240	229	288	230	157	160	40
1.2	988	1,392	1,356	1,352	751	741	608	704	516	58
1.4	2,939	2,581	2,066	2,190	1,351	1,622	1,132	1,261	932	86
1.6	3,150	2,900	2,842	2,218	1,885	1,563	1,177	821	896	85
1.8	2,782	2,485	2,047	2,180	1,625	1,269	855	395	309	62
2.0	1,338	1,638	1,544	1,566	979	957	349	261	147	54
2.2	838	1,125	993	1,027	667	521	153	125	93	45
2.4	494	700	624	524	296	214	126	87	55	32
2.6	327	434	383	334	212	163	82	50	41	26
2.8	283	273	305	202	144	95	60	44	33	22
3.0	164	200	173	142	92	72	41	34	27	18

(g) Fribourg+M1+R2C+C

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	126	118	97	60	51	52	62	36	48	30
1.2	432	517	345	262	160	126	92	120	109	40
1.4	1,044	331	133	89	45	22	19	31	27	20
1.6	358	24	11	5	4	6	5	3	3	4
1.8	19	5	1	1	1	1	1	1	1	1
2.0	1	1	1	1	1	1	1	1	1	1
2.2	1	1	1	1	1	1	1	1	1	1
2.4	1	1	1	1	1	1	1	1	1	1
2.6	1	1	1	1	1	1	1	1	1	1
2.8	1	1	1	1	1	1	1	1	1	1
3.0	1	1	1	1	1	1	1	1	1	1

(h) Fribourg+R

Figure B.1: Median complement sizes of the 10,939 effective samples of the internal tests on the GOAL test set. The rows (1.0 to 3.0) are the transition densities, and the columns (0.1 to 1.0) are the acceptance densities.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	130	117	109	77	69	61	56	40	40	29
1.2	387	456	352	281	155	136	101	105	75	45
1.4	822	683	394	376	230	204	151	120	105	63
1.6	890	594	458	321	237	178	134	114	113	61
1.8	624	507	324	275	196	136	110	92	89	41
2.0	362	286	211	176	117	103	79	64	59	34
2.2	248	222	124	116	82	73	56	52	50	28
2.4	147	145	114	87	56	48	43	39	35	19
2.6	115	117	67	61	47	42	32	29	29	15
2.8	95	71	52	45	38	29	27	25	23	13
3.0	59	60	47	35	32	27	22	21	20	10

(a) Piterman+EQ+RO

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	171	174	166	124	118	117	100	67	84	35
1.2	622	833	803	877	529	398	320	372	215	53
1.4	2,086	1,618	1,367	1,676	1,065	967	664	682	494	78
1.6	2,465	2,073	2,182	1,959	1,518	1,259	767	545	623	78
1.8	2,310	1,963	1,950	1,988	1,485	1,095	746	418	346	57
2.0	1,318	1,482	1,393	1,461	981	871	434	338	228	50
2.2	1,068	1,145	1,085	1,067	772	747	263	235	158	40
2.4	689	838	809	751	524	466	240	159	93	30
2.6	469	531	555	565	437	360	169	94	71	23
2.8	369	421	536	405	329	224	130	81	58	21
3.0	244	327	360	322	219	176	85	64	49	16

(b) Slice+P+RO+MADJ+EG

Figure B.2: Median complement sizes of the 10,998 effective samples of the external tests without the Rank construction. The rows (1.0 to 3.0) are the transition densities, and the columns (0.1 to 1.0) are the acceptance densities.

Appendix C

Execution Times

Construction	Mean	Min.	P25	Median	P75	Max.	Total	≈ hours
Fribourg	8.5	2.5	3.3	4.9	7.3	586.0	93,351.2	259
Fribourg+R2C	6.6	2.2	2.9	4.2	6.4	219.7	72,545.7	202
Fribourg+R2C+C	8.5	2.2	2.6	3.5	6.4	582.9	93,396.2	259
Fribourg+M1	4.9	2.5	3.2	4.1	5.9	55.1	54,061.3	150
Fribourg+M1+M2	4.6	2.2	2.9	3.8	5.1	38.4	49,848.0	138
Fribourg+M1+R2C	4.4	2.2	2.8	3.6	5.3	42.5	48,572.0	135
Fribourg+M1+R2C+C	5.6	2.5	3.2	4.0	6.5	147.4	60,918.9	169
Fribourg+R	7.5	2.2	3.0	3.9	6.3	470.5	82,387.3	229

Table C.1: Execution times in CPU time seconds for the 10,939 effectvie samples of the GOAL test set.

Construction	Mean	Min.	P25	Median	P75	Max.	Total	≈ hours
Piterman+EQ+RO	3.0	2.2	2.6	2.8	3.0	42.9	21,410.6	59
Slice+P+RO+MADJ+EG	3.7	2.2	2.7	3.2	4.1	36.7	26,398.9	73
Rank+TR+RO	16.0	2.3	2.8	3.7	9.3	443.3	115,563.9	321
Fribourg+M1+R2C	4.0	2.2	2.7	3.1	4.4	410.4	28,970.8	80

Table C.2: Execution times in CPU time seconds for the 7,204 effectvie samples of the GOAL test set.

Construction	Mean	Min.	P25	Median	P75	Max.	Total	≈ hours
Piterman+EQ+RO	3.6	2.2	2.7	2.9	3.4	365.7	39,663.4	110
Slice+P+RO+MADJ+EG	4.3	2.2	2.9	3.7	5.0	42.4	47,418.2	132
Fribourg+M1+R2C	4.7	2.2	2.8	3.6	5.3	410.4	52,149.0	145

Table C.3: Execution times in CPU time seconds for the 10,998 effectvie samples of the GOAL test set without the Rank construction.

Construction	Michel 1	Michel 2	Michel 3	Michel 4	Fitted curve	Std. error
Fribourg	2.3	4.0	88.8	100,976.0	$(1.14n)^n$	0.64%
Fribourg+R2C	2.3	3.4	27.4	27,938.3	$(0.92n)^n$	0.64%
Fribourg+M1	2.2	3.6	17.9	6,508.4	$(0.72n)^n$	0.63%
Fribourg+M1+M2	2.3	3.5	13.8	2,707.4	$(0.62n)^n$	0.62%
Fribourg+M1+M2+R2C	2.5	3.5	10.8	2,332.6	$(0.61n)^n$	0.62%
Fribourg+R	2.4	3.7	86.0	101,809.6	$(1.14n)^n$	0.64%

Table C.4: Execution times in CPU time seconds for the four Michel automata.

Construction	Michel 1	Michel 2	Michel 3	Michel 4	Fitted curve	Std. error
Piterman+EQ+RO	2.5	3.8	42.6	75,917.4	$(1.08n)^n$	0.64%
Slice+P+RO+MADJ+EG	2.3	3.6	11.4	159.5	$(0.39n)^n$	0.38%
Rank+TR+RO	2.2	3.0	6.4	30.0	$(0.29n)^n$	0.18%
Fribourg+M1+M2+R2C	2.5	3.5	10.8	2,332.6	$(0.61n)^n$	0.62%

Table C.5: Execution times in CPU time seconds for the four Michel automata.

Bibliography

[1] J. Allred, U. Ultes-Nitsche. Complementing Büchi Automata with a Subset-Tuple Construction. Tech. rep.. University of Fribourg, Switzerland. 2014.

[2] C. Althoff, W. Thomas, N. Wallmeier. Observations on Determinization of Büchi Automata. In J. Farré, I. Litovsky, S. Schmitz, eds., *Implementation and Application of Automata*. vol. 3845 of *Lecture Notes in Computer Science*. pp. 262–272. Springer Berlin Heidelberg. 2006.

[3] S. Breuers, C. Löding, J. Olschewski. Improved Ramsey-Based Büchi Complementation. In L. Birkedal, ed., *Foundations of Software Science and Computational Structures*. vol. 7213 of *Lecture Notes in Computer Science*. pp. 150–164. Springer Berlin Heidelberg. 2012.

[4] J. R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *Proc. International Congress on Logic, Method, and Philosophy of Science, 1960*. Stanford University Press. 1962.

[5] S. J. Fogarty, O. Kupferman, T. Wilke, et al. Unifying Büchi Complementation Constructions. *Logical Methods in Computer Science*. 9(1). 2013.

[6] E. Friedgut, O. Kupferman, M. Vardi. Büchi Complementation Made Tighter. In F. Wang, ed., *Automated Technology for Verification and Analysis*. vol. 3299 of *Lecture Notes in Computer Science*. pp. 64–78. Springer Berlin Heidelberg. 2004.

[7] E. Friedgut, O. Kupferman, M. Y. Vardi. Büchi Complementation Made Tighter. *International Journal of Foundations of Computer Science*. 17(04):pp. 851–867. 2006.

[8] C. Göttel. Implementation of an Algorithm for Büchi Complementation. BSc Thesis, University of Fribourg, Switzerland. November 2013.

[9] R. L. Graham, B. L. Rothschild, J. H. Spencer. *Ramsey theory*. vol. 20. John Wiley & Sons. 1990.

[10] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley. 2nd edition ed.. 2001.

[11] D. Kähler, T. Wilke. Complementation, Disambiguation, and Determinization of Büchi Automata Unified. In L. Aceto, I. Damgård, L. Goldberg, et al, eds., *Automata, Languages and Programming*. vol. 5125 of *Lecture Notes in Computer Science*. pp. 724–735. Springer Berlin Heidelberg. 2008.

[12] N. Klarlund. Progress measures for complementation of omega-automata with applications to temporal logic. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*. pp. 358–367. Oct 1991.

[13] J. Klein. Linear Time Logic and Deterministic Omega-Automata. *Master's thesis, Universität Bonn*. 2005.

[14] J. Klein, C. Baier. Experiments with Deterministic ω-Automata for Formulas of Linear Temporal Logic. In J. Farré, I. Litovsky, S. Schmitz, eds., *Implementation and Application of Automata*. vol. 3845 of *Lecture Notes in Computer Science*. pp. 199–212. Springer Berlin Heidelberg. 2006.

[15] O. Kupferman, M. Y. Vardi. Weak Alternating Automata Are Not that Weak. In *Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems*. pp. 147–158. IEEE Computer Society Press. 1997.

[16] O. Kupferman, M. Y. Vardi. Weak Alternating Automata Are Not that Weak. *ACM Trans. Comput. Logic.* 2(3):pp. 408–429. Jul. 2001.

[17] R. Kurshan. Complementing Deterministic Büchi Automata in Polynomial Time. *Journal of Computer and System Sciences.* 35(1):pp. 59 – 71. 1987.

[18] C. Löding. Optimal Bounds for Transformations of ω-Automata. In C. Rangan, V. Raman, R. Ramanujam, eds., *Foundations of Software Technology and Theoretical Computer Science.* vol. 1738 of *Lecture Notes in Computer Science.* pp. 97–109. Springer Berlin Heidelberg. 1999.

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

[20] M. Michel. Complementation is more difficult with automata on infinite words. *CNET, Paris.* 15. 1988.

[21] A. Mostowski. Regular expressions for infinite trees and a standard form of automata. In A. Skowron, ed., *Computation Theory.* vol. 208 of *Lecture Notes in Computer Science.* pp. 157–168. Springer Berlin Heidelberg. 1985.

[22] D. E. Muller. Infinite Sequences and Finite Machines. In *Switching Circuit Theory and Logical Design, Proceedings of the Fourth Annual Symposium on.* pp. 3–16. Oct 1963.

[23] D. E. Muller, A. Saoudi, P. E. Schupp. Alternating automata, the weak monadic theory of the tree, and its complexity. In L. Kott, ed., *Automata, Languages and Programming.* vol. 226 of *Lecture Notes in Computer Science.* pp. 275–283. Springer Berlin Heidelberg. 1986.

[24] D. E. Muller, P. E. Schupp. Simulating Alternating Tree Automata by Nondeterministic Automata: New Results and New Proofs of the Theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science.* 141(1–2):pp. 69 – 107. 1995.

[25] F. Nießner, U. Nitsche, P. Ochsenschläger. Deterministic Omega-Regular Liveness Properties. In S. Bozapalidis, ed., *Preproceedings of the 3rd International Conference on Developments in Language Theory, DLT'97.* pp. 237–247. Citeseer. 1997.

[26] J.-P. Pecuchet. On the complementation of Büchi automata. *Theoretical Computer Science.* 47(0):pp. 95 – 98. 1986.

[27] N. Piterman. From Nondeterministic Buchi and Streett Automata to Deterministic Parity Automata. In *Logic in Computer Science, 2006 21st Annual IEEE Symposium on.* pp. 255–264. 2006.

[28] N. Piterman. From Nondeterministic Buchi and Streett Automata to Deterministic Parity Automata. *Logical Methods in Computer Science.* 3(5):pp. 1–21. 2007.

[29] M. Rabin, D. Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development.* 3(2):pp. 114–125. April 1959.

[30] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society.* 141:pp. 1–35. July 1969.

[31] F. P. Ramsey. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society.* s2-30(1):pp. 264–286. 1930.

[32] M. Roggenbach. Determinization of Büchi-Automata. In E. Grädel, W. Thomas, T. Wilke, eds., *Automata Logics, and Infinite Games.* vol. 2500 of *Lecture Notes in Computer Science.* pp. 43–60. Springer Berlin Heidelberg. 2002.

[33] S. Safra. On the Complexity of Omega-Automata. *Journal of Computer and System Science.* 1988.

[34] S. Safra. On the Complexity of Omega-Automata. In *Foundations of Computer Science, 1988., 29th Annual Symposium on.* pp. 319–327. Oct 1988.

[35] S. Schewe. Büchi Complementation Made Tight. In *26th International Symposium on Theoretical Aspects of Computer Science-STACS 2009.* pp. 661–672. 2009.

[36] A. Sistla, M. Vardi, P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In W. Brauer, ed., *Automata, Languages and Programming*. vol. 194 of *Lecture Notes in Computer Science*. pp. 465–474. Springer Berlin Heidelberg. 1985.

[37] A. P. Sistla, M. Y. Vardi, P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*. 49(2–3):pp. 217 – 237. 1987.

[38] R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*. 54(1–2):pp. 121 – 141. 1982.

[39] W. Thomas. Automata on Infinite Objects. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science (Vol. B)*. chap. Automata on Infinite Objects, pp. 133–191. MIT Press, Cambridge, MA, USA. 1990.

[40] W. Thomas. Languages, Automata, and Logic. In G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*. pp. 389–455. Springer Berlin Heidelberg. 1997.

[41] W. Thomas. Complementation of Büchi Automata Revisited. In J. Karhumäki, H. Maurer, G. Păun, et al, eds., *Jewels are Forever*. pp. 109–120. Springer Berlin Heidelberg. 1999.

[42] M.-H. Tsai, S. Fogarty, M. Vardi, et al. State of Büchi Complementation. In M. Domaratzki, K. Salomaa, eds., *Implementation and Application of Automata*. vol. 6482 of *Lecture Notes in Computer Science*. pp. 261–271. Springer Berlin Heidelberg. 2011.

[43] M.-H. Tsai, Y.-K. Tsay, Y.-S. Hwang. GOAL for Games, Omega-Automata, and Logics. In N. Sharygina, H. Veith, eds., *Computer Aided Verification*. vol. 8044 of *Lecture Notes in Computer Science*. pp. 883–889. Springer Berlin Heidelberg. 2013.

[44] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, et al. Goal: A Graphical Tool for Manipulating Büchi Automata and Temporal Formulae. In O. Grumberg, M. Huth, eds., *Tools and Algorithms for the Construction and Analysis of Systems*. vol. 4424 of *Lecture Notes in Computer Science*. pp. 466–471. Springer Berlin Heidelberg. 2007.

[45] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, et al. Goal Extended: Towards a Research Tool for Omega Automata and Temporal Logic. In C. Ramakrishnan, J. Rehof, eds., *Tools and Algorithms for the Construction and Analysis of Systems*. vol. 4963 of *Lecture Notes in Computer Science*. pp. 346–350. Springer Berlin Heidelberg. 2008.

[46] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, et al. Tool support for learning Büchi automata and linear temporal logic. *Formal Aspects of Computing*. 21(3):pp. 259–275. 2009.

[47] Y.-K. Tsay, M.-H. Tsai, J.-S. Chang, et al. Büchi Store: An Open Repository of Büchi Automata. In P. Abdulla, K. Leino, eds., *Tools and Algorithms for the Construction and Analysis of Systems*. vol. 6605 of *Lecture Notes in Computer Science*. pp. 262–266. Springer Berlin Heidelberg. 2011.

[48] U. Ultes-Nitsche. A Power-Set Construction for Reducing Büchi Automata to Non-Determinism Degree Two. *Information Processing Letters*. 101(3):pp. 107 – 111. 2007.

[49] M. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller, G. Birtwistle, eds., *Logics for Concurrency*. vol. 1043 of *Lecture Notes in Computer Science*. pp. 238–266. Springer Berlin Heidelberg. 1996.

[50] M. Vardi. The Büchi Complementation Saga. In W. Thomas, P. Weil, eds., *STACS 2007*. vol. 4393 of *Lecture Notes in Computer Science*. pp. 12–22. Springer Berlin Heidelberg. 2007.

[51] M. Y. Vardi. Buchi Complementation: A Forty-Year Saga. In *5th symposium on Atomic Level Characterizations (ALC'05)*. 2005.

[52] M. Y. Vardi, T. Wilke. Automata: From Logics to Algorithms. In J. Flum, E. Grädel, T. Wilke, eds., *Logic and Automata: History and Perspectives*. vol. 2 of *Texts in Logic and Games*. pp. 629–736. Amsterdam University Press. 2007.

[53] T. Wilke. ω-Automata. In J.-E. Pin, ed., *Handbook of Automata Theory*. European Mathematical Society. To appear, 2015.

[54] Q. Yan. Lower Bounds for Complementation of ω-Automata Via the Full Automata Technique. In M. Bugliesi, B. Preneel, V. Sassone, et al, eds., *Automata, Languages and Programming*. vol. 4052 of *Lecture Notes in Computer Science*. pp. 589–600. Springer Berlin Heidelberg. 2006.

[55] Q. Yan. Lower Bounds for Complementation of omega-Automata Via the Full Automata Technique. *CoRR*. abs/0802.1226. 2008.