Performance Investigation of a Subset-Tuple Büchi Complementation Construction

Daniel Weibel

November 2, 2014

Contents

Chapter 1

The Büchi Complementation Problem

1.1 Preliminaries

1.1.1 Büchi Automata

Büchi automata have been introduced in 1962 by Büchi [2] in order to show the decidability of monadic second order logic; over the successor structure of the natural numbers [1].

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 [25] (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 [24].

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 [2]. 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 [24][18]. 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 [4][16]. In the case of Büchi automata, this inequivalence is the main cause that Büchi complementation problem is such a hard problem [15] and until today regarded as unsolved.

1.1.2 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) [13], Rabin (Rabin automata, 1969) [17], Streett (Streett automata, 1982) [21], and Mostowski (parity automata, 1985) [12].

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 [9]. 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	$\{(E_1, F_1), \ldots, (E_r, F_r)\}$, $E_i, F_i \subseteq Q$	$\exists i : \inf(\rho) \cap E_i = \varnothing \wedge \inf(\rho) \cap F_i \neq \varnothing$
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	$c : Q \to \{1, \ldots, k\}$, $k \in \mathbb{N}$	$\min\{c(q) \mid q \in \inf(\rho)\} \bmod 2 = 0$

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 [9], which will be used for certain Büchi complementation construction, as we will see in Section??. 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* [10]. Another proof given in [22]. 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 **??**.

It turned out that also all the other types of the just introduced ω-automata, non-deterministic and determinstic, are equivalent among each other [18][7][6][9][22]. 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 **??**

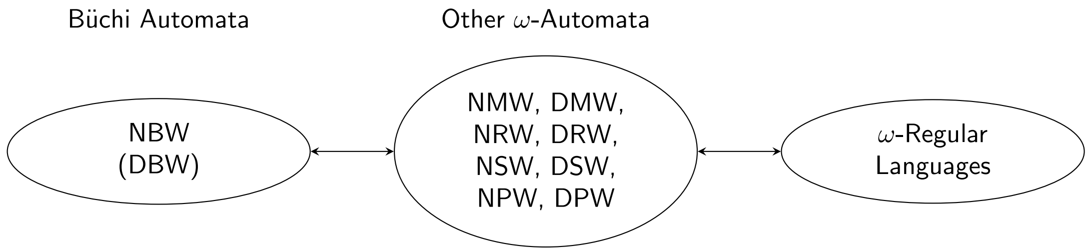


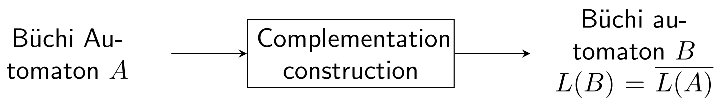
Figure 1.1: Non-deterministic Büchi automata (NBW) are expressively equivalent to Muller, Rabin, Streett, and parity automata (both deterministic and non-deterministic), and to the ω-regular languages. Deterministic Büchi automata (DBW) are less expressive than NBW.

1.1.3 Complementation of Büchi Automata

Büchi automata are closed under complementation. This result has been proved by Büchi himself when he introduced Büchi automata in [2]. 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 [22] (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 [4] (p. 133). A DFA can be complemented by simply switching its accepting and non-accepting states [4] (p. 133). Now, since NFA and DFA are equivalent [4] (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 [8], 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 1, is another alternative way of achievin this same aim.

1.1.4 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 [2] 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 [20]. 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 $n!$ [11]. 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 [26]. 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 $2^{O(n \, log \, n)}$ [26].

Michel's lower bound remained valid for almost two decades until in 2006 Yan showed a new lower bound of $(0.76n)^n$ [26]. 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) [19]. 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 [24] p. 8.

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

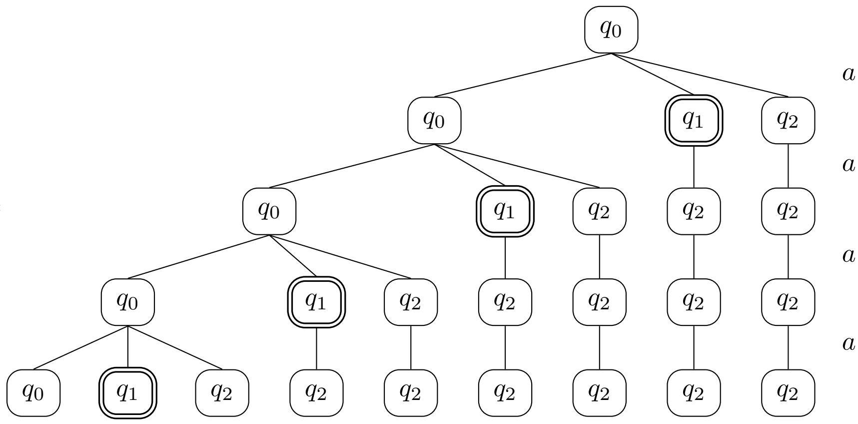


Figure 1.2: Automaton A and the first five levels of the run tree of the runs of A on the word a^ω.

1.2 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.

1.2.1 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 **??** 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.

(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 [14]. 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.

1.2.2 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

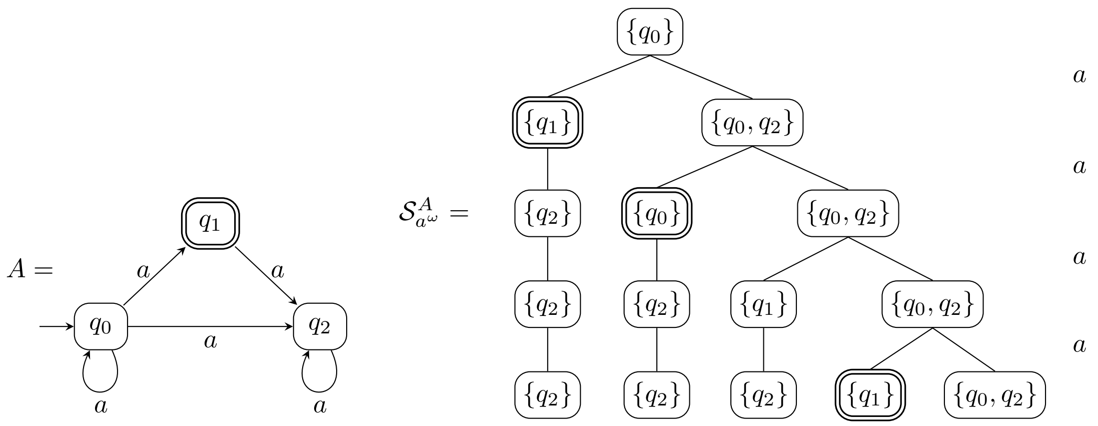


Figure 1.3: Automaton A and the first five levels of the split tree of the runs of A on the word a^ω.

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 [23]. Generally, the idea of treating accepting and non-accepting states separately is important in the run analyis of Büchi automata.

1.2.3 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 **??** 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 [5], cf. [3]. 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 [14]. Formal definitions os split trees can be found in [5][3].

1.2.4 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.

1.2.5 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 **??** shows the run DAG of automaton A on the word $w = a^\omega$.

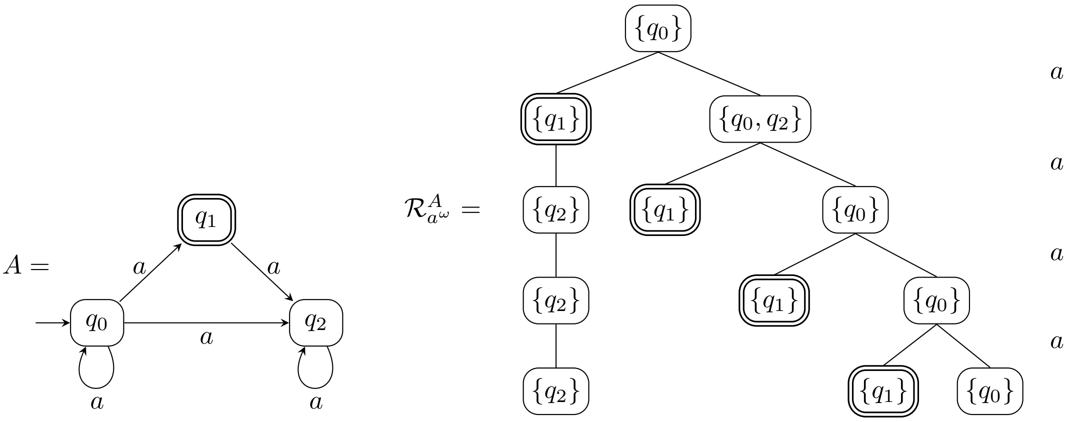


Figure 1.4: Automaton A and the first five levels of the left-to-right reduced split tree of the runs of A on the word a^ω.

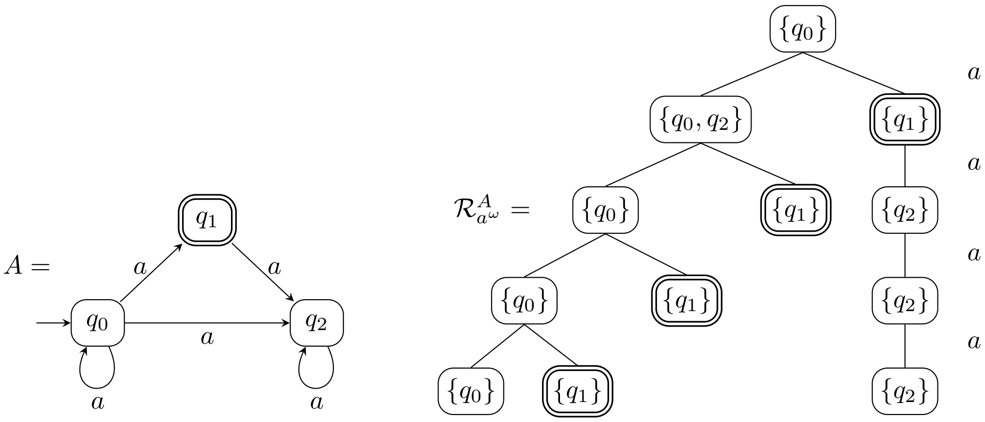


Figure 1.5: Automaton A and the first five levels of the left-to-right reduced split tree of the runs of A on the word a^ω.

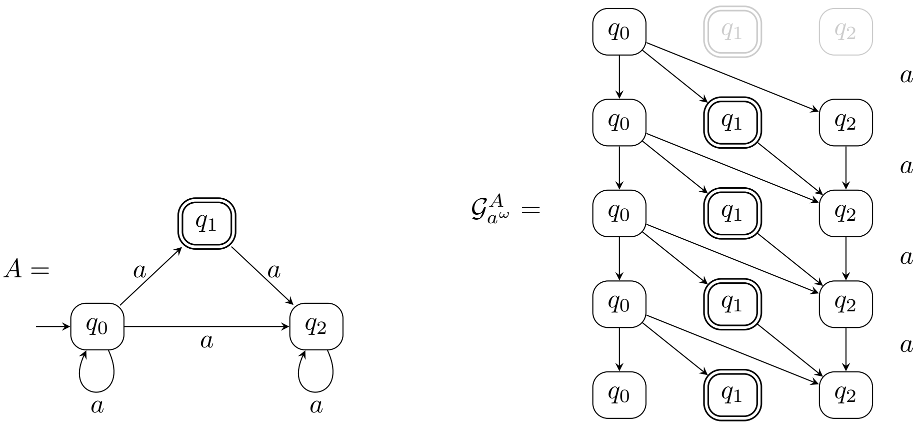


Figure 1.6: Automaton A and the first five levels of the run DAG of the runs of A on the word a^ω.

1.3 Review of Büchi Complementation Constructions

1.3.1 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 [1].

1.3.2 Determinisation-Based Approaches

1.3.3 Rank-Based Approaches

1.3.4 Slice-Based Approaches

1.4 Empirical Performance Investigations

Chapter 2

The Fribourg Construction

The Fribourg construction draws from several ideas: the subset construction, run analysis based on reduced split trees, and Kurshan's construction [8] for complementing DBW. Following the classification we used in Section **??**, it is a slice-based construction. Some of its formalisations are similar to the slice-based construction by Vardi and Wilke [25], 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.

2.0.1 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

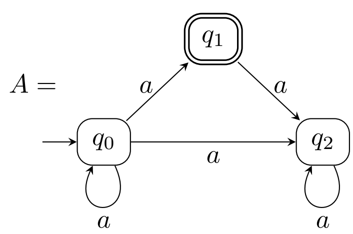


Figure 2.1: Example automaton A

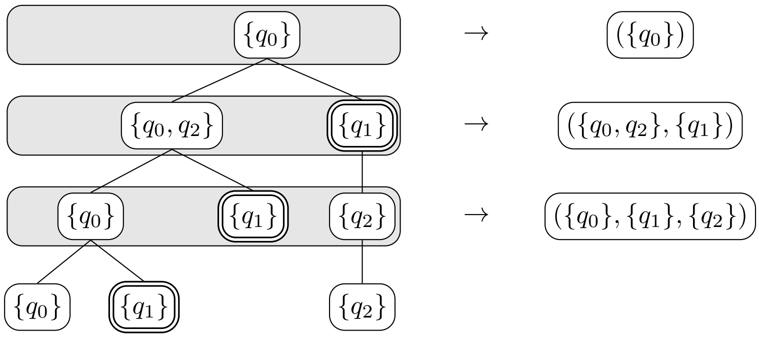


Figure 2.2: Mapping from levels of a reduced split tree, to states of the subset-tuple construction.

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 [25, 5]. 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 1.2 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 1.2, 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 1.1, 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 1.3 shows the final upper part $B'$of A.

2.0.2 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

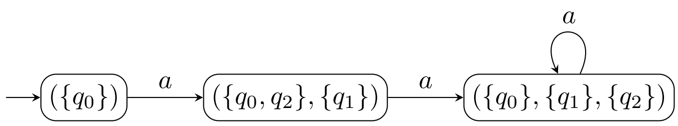


Figure 2.3: Upper part $B'$of example automaton A.

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 (1.0.2) one, 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 (1.0.2) we give the idea and intuition behind the construction and explain why it works.

The Steps for Constructing the Lower Part

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 1.4 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 1.3, and thereby complete the complementation of the example automaton A in Figure 1.1.

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 1.4. 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 $(\{q_0, q_2\}, \overline{\overline{\{q_1\}}})$.

p_{pred} has no sets with colour 2	s non-accepting	s accepting
$c(s_{pred}) = -1$	0	2
$c(s_{pred}) = 0$	0	2
$c(s_{pred}) = 1$	2	2

p_{pred} has set(s) with colour 2	s non-accepting	s accepting
$c(s_{pred}) = 0$	0	1
$c(s_{pred}) = 1$	1	1
$c(s_{pred}) = 2$	2	2

Figure 2.4: Colour rules.

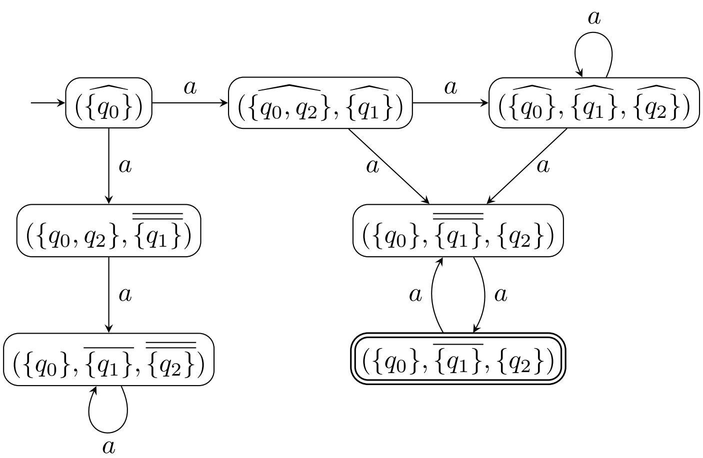


Figure 2.5: The final complement automaton B.

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 1.4 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 ($M_2(1,2)$). $\{q_0\}$, 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 1.5. 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 1.5 is the complement of the NBW A in Figure 1.1, 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).

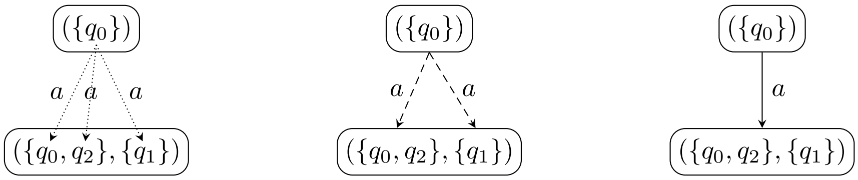


Figure 2.6: Different notions of runs.

Intuition

2.1 Optimisations

2.1.1 Removal of Non-Accepting States (R2C)

2.1.2 Merging of Adjacent Sets (M)

2.1.3 Reduction of 2-Coloured Sets (M2)

Bibliography

[1] 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.

[2] 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.

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

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

[5] 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.

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

[7] 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.

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

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.