# Empirical Performance Investigation of a Büchi Complementation Construction

Daniel Weibel

July 31, 2015

**Abstract**

This will be the abstract.

**Acknowledgements**

# Contents

# Chapter 1

# Introduction

At the beginning of the 1960's, a Swiss logician named Julius Richard Büchi was looking for a way to decide the satisfiability of formulas of the monadic second order logic with one successor (S1S). In his quest, Büchi observed that an S1S formula can be represented by a certain type of finite state automaton that runs on infinite words, such that this automaton accepts a word if and only if the corresponding interpretation satisfies the formula. The proof of this equivalence between S1S formulas and this type of autoamtaon, which is known as *Büchi's Theorem*, led Büchi to his desired decision procedure: to test whether an S1S formula $\varphi$ is satisfiable, translate it to an equivalent automaton $A$, and test whether $A$ is empty (that is, accepts no words at all). If $A$ is empty, then $\varphi$ is unsatisfiable, if $A$ is non-empty, then $\varphi$ is satsifiable. [4]

The type of automaton that Büchi used for solving this logical problem is called *Büchi automaton*. The application of Büchi automata to logic, that was established by Büchi, had a large impact on other fields, especially model checking, which is a technique of formal verification. In particular, Büchi automata allow to solve the model checking question automata-theoretically, which has many advantages [49].

However, there is one operation on Büchi automata that is giving a "headache" to the research community since the introduction of Büchi automata more than 50 years ago, namely the problem of *complementation*. Algorithms for carrying out this operation, although possible[1], turn out to be very complex, in many cases too complex for practical application. Yet, Büchi complementation has a practical application in the automata-theoretic approach to model checking. This discrepancy led to an ongoing quest for finding more efficient *Büchi complementation constructions*, and generally better understanding the complexity of Büchi complementation. The work in this thesis is situated in this area of research.

In this introductory chapter, we will first

## 1.1 Context

### 1.1.1 Büchi Automata and Büchi Complementation

Büchi automata are finite state automata that process words of infinite length, so called $\omega$-words. If $\Sigma$ is the alphabet of a Büchi automaton, then the set of all the possible $\omega$-words that can be generated from this alphabet is denoted by $\Sigma^\omega$. A word $\alpha \in \Sigma^\omega$ is accepted by a Büchi automaton if it results in at least one run that contain infinitely many occurrences of at least one accepting state. A run of a Büchi automaton on a word is an infinite sequence of states. Deterministic Büchi automata have at most one run for each word in $\Sigma^\omega$, whereas non-deterministic Büchi automata may have multiple runs for a word.

The complement of a Büchi automaton $A$ is another Büchi automaton[2] denoted by $\overline{A}$. Both, $A$ and $\overline{A}$, share the same alphabet $\Sigma$. Regarding a word $\alpha \in \Sigma^\omega$, the relation between an automaton and its complement is as follows:

$$\alpha \text{ accepted by } A \iff \alpha \text{ not accepted by } \overline{A}$$

That is, all the words of $\Sigma^\omega$ that are *accepted* by an automaton are *rejected* by its complement, and all the words of $\Sigma^\omega$ that are *rejected* by an automaton are *accepted* by its complement. In other words, there is no single word of $\Sigma^\omega$ that is accepted or rejected by *both* of an automaton and its complement.

A Büchi complementation construction is an algorithm that, given a Büchi automaton, creates the complement of this Büchi automaton. The difficulty of this operation depends on whether the input automaton is determinstic or non-deterministic. The complementation of deterministic Büchi automata is "easy" and can be done in polynomial time and linear space [17]. The complementation of non-deterministic Büchi automata, however, is very complex. The understanding and reduction of its complexity is a domain of active research and lies at the centre of this thesis.

Consequently, when in the following we talk about "Büchi complementation", we specifically mean the complementation of *non-determinstic* Büchi automata. The main problem with the complexity of Büchi complementation is the so-called state growth or state complexity (sometimes also called state blow-up or

[1] Büchi himself has proved that Büchi automata are closed under complementation [4].

[2] Büchi automata are closed under complementation. This has been proved by Büchi [4], who, to this end, described the first Büchi complementation construction in history.

state explosion). This is the number of states of the output automaton in relation to the number of states of the input automaton. In simple words, Büchi complementation constructions produce complements that may be very, very large.

This inhibits the practial applicaion of Büchi complementation, because in this case the limited computing and time resources may not be high enough to accommodate for this high complexity. In the following subsections we highlight an important application that Büchi complementation has in practice, and thereby motivate the research on Büchi complementation and of this thesis.

### 1.1.2 Applications of Büchi Complementation

**Language Containment of $\omega$-Regular Languages**

Büchi complementation is used for testing language containment of $\omega$-regular languages. The $\omega$-regular languages are the class of formal languages that is equivalent to non-deterministic Büchi automata[3]. At this point, we briefly describe the language containment in general, before in turn describing an application of the language containment problem in the next subsection.

Given two $\omega$-regular languages $L_1$ and $L_2$ over alphabet $\Sigma^\omega$ the language containment problem consists in testing whether $L_1 \subseteq L_2$. This is true if all the words of $L_1$ are also in $L_2$, and false if $L_1$ contains at least one word that is not in $L_2$. The way this problem is algorithmically solved is by testing $L_1 \cap \overline{L_2} = \varnothing$. Here, $\overline{L_2}$ denotes the complement language of $L_2$, which means $\overline{L_2}$ contains all the words of $\Sigma^\omega$ that are *not* in $L_2$. The steps for testing $L_1 \cap \overline{L_2} = \varnothing$ are the following:

- Create the complement language $\overline{L_2}$ of $L_2$
- Create the intersection language $L_{1,\overline{2}}$ of $L_1$ and $\overline{L_2}$
- Test whether $L_{1,\overline{2}}$ is empty (that is, contains no words at all)

Thus, the language containment problem is reduced to three operations on languages, *complementation*, *intersection*, and *emptiness testing*. The common way to work with formal languages is not to handle the languages themselves, but more compact structures that represent them, such as automata. In the case of $\omega$-regular languages, these are non-deterministic Büchi automata.

For solving $L_1 \subseteq L_2$, one thus works with two Büchi automata $A_1$ and $A_2$ that represent $L_1$ and $L_2$, respectively. The problem then becomes $L(A_1) \subseteq L(A_2)$, and equivalently, $L(A_1) \cap \overline{L(A_2)} = \varnothing$. This is automata-theoretically solved as $\mathsf{empty}(A_1 \cap \overline{A_2})$, which includes the three following steps:

- Construct the complement automaton $\overline{A_2}$ of $A_2$
- Construct the intersection automaton $A_{1,\overline{2}}$ of $A_1$ and $A_2$
- Test whether $A_{1,\overline{2}}$ is empty (that is, accepts no words at all)

If the final emptiness test on automaton $A_{1,\overline{2}}$ is true, then $L_1 \subseteq L_2$ is true, and if the emptiness test is false, then $L_1 \subseteq L_2$ is false. In this way, the language containment problem of $\omega$-regular languages is reduced to three operations of *complementation*, *intersection*, and *emptiness testing* of non-deterministic Büchi automata. Thus, Büchi complementation is an intergral part of language contaiment of $\omega$-regular languages.

**Automata-Theoretic Model Checking via Language Containment**

In the last subsection, we have seen that Büchi complementation is used for testing language containment of $\omega$-regular languages. In this subsection, we will see what in turn language containment of $\omega$-regular languages is used for. To this end, we describe one important application of it, namely the language containment approach to automata-theoretic model checking. In the following, we first describe basic working of this technique in general, and then point out the significance that Büchi complementation bears for it.

[3]Note that deterministic Büchi automata have a lower expressivity than non-deterministic Büchi automata, and are equivalent to only a subset of the $\omega$-regular languages.

### Basics

The language containment approach to automata-theoretic model checking is an approach to automata-theoretic model checking, which is an approach to model checking, which in turn is an approch to formal verification [**?**]. Figure 1.1 shows the branch of the family of formal verification techniques that contains the language containment approach to automata-theoretic model checking.

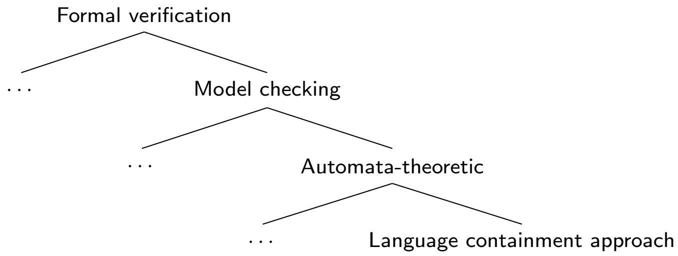


Figure 1.1: Branch of the family of formal verification techniques that contains the language containment approach to automata-theoretic model checking.

Formal verification is the use of mathematical techniques for proving the correctness of a system (software of hardware) with respect to a specified property [**?**]. A typical example is to verify that a program is deadlock-free (in which case the property would be "deadlock-freeness"). In general, formal verification techniques consist of the following three parts [**?**]:

1. A framework for modelling the system to verify
2. A framework for specifying the property to be verified
3. A verification method for testing whether the system satisfies the property

For the language containment approach to automata-theoretic model checking, the frameworks for points 1 and 2 are Büchi automata representing $\omega$-regular languages. The verification method (point 3) is testing language containment of the system automaton's language in the property automaton's language. In some more detail, this works as follows. [49][**?**]

The system $s$ to verify is modelled as a Büchi automaton, say $S$. This Büchi automaton represents an $\omega$-regular language $L(S)$, and each word of $L(S)$ represents in turn a possible *computation trace* of the system. A computation trace is an infinite[4] sequence of "situations" that the system is in at any point in time. Such a situation consists of a finite amount of information of, for example, the values of variables, registers, or buffers. The observation that such a trace can be represented as a word of an $\omega$-regular langauges comes from the fact that it can be represented intuitively as a linear Kripke structure (which in turn is an interpretation for a temporal logic formula that can also be used to represent computations), which in turn can be represented by a word of a language whose alphabet is ranges over the powerset of the atomic propositions of the Kripke strucutre. A work that explains these intimate relations between computation, temporal logic, formal languages, and automata in more detail is [49]. In simple words, the language $L(S)$, represented by the system automaton $S$, represents everything that the system *can* do.

Similarly, a property $p$ to be verified is represented as a Büchi automaton, say $P$, which represents the $\omega$-regular language $L(P)$, whose words represent computation traces. These computation traces are all the computations of a system like $s$ that satisfy the property $p$. If for example $p$ is "deadlock-freeness", then the words of $L(P)$ represent all the possible computation traces that do *not* contain a deadlock. In this way, the language $L(P)$ represents everything that the system is *allowed* to do, with respect to a certain property.

The verification step is finally done by testing $L(S) \subseteq L(P)$. If this is true, then everything that the system *can* do is *allowed* to do, and the system satisfies the property $p$. If the language containment test

[4] The infinity of computation traces suggests that this type of formal verification (and model checking in general) is used for systems that are not expected to terminate and may run indefinitely. This type of systems is called *reactive* systems. They contrast with systems that are expected to terminate and produce a result. For this latter type of systems other formal verification techniques than model checking are used. See for example [**?**] and [**?**] for works that cover the formal verification of both types of systems.

is false, then the system *can* do a computation that is *not allowed* to do, and the system does not satisify the property $p$.

Summarising, the language containment approach to automata-theoretic model checking requires language containment of $\omega$-regular languages, which, as we have seen, requires Büchi complementation. In the following, we will highlight the particular importance of Büchi complementation for this type of formal verification.

### 1.1.3 Significance of Büchi Complementation

The complexity of Büchi complementation makes the just described model checking approach nearly inapplicable in practice [**?**]. According to [**?**], there are so far no tools that include this approach. This is unfortunate, because the other Büchi operations for language containment, intersection and emptiness testing, have highly efficient solutions [**?**] (cf. [49]), and thus Büchi complementation is the only bottleneck. Existing practical applications are thus forced to circumvent the need for Büchi complementation. This is possible, has however certain disadvantages as we will see in the following.

One way to circumvent the complementation of non-deterministic Büchi automata is to specify the property as a deterministic Büchi automaton [**?**][**?**]. As we have mentioned, the complementation of deterministic Büchi automata has an efficient solution. The disadvantage of this approach is, however, that the property automaton may become exponentially larger, and that it is generally more complicated and less intuitive to represent a language as a deterministic automaton [**?**].

Another way is to use a different model checking approach altogether, which leads us back to the essence of model checking. In basic model checking, the property to be verified is represented as a formula $\varphi$ of a temporal logic (typically LTL). The system to verify is represnted as a Kripke structure $K$, which serves as an interpretation of the formula $\varphi$. The verification step consists in checking whether $K$ is a *model* of $\varphi$. An interpretation $K$ is a model of a formula $\varphi$, if every state of the interpretation *satisifes* the formula, written as $K \models \varphi$. This test of modelhood is the reason that this verification approach is called *model checking* [49].

The modelhood test can be done automata-theoretically without the need for Büchi complementation [**?**] (in Figure 1.1, this would be a sibling to the language containment approach). The Kripke structure $K$ is translated to a Büchi automaton $A_K$. The formula $\varphi$ is negated and translated to the Büchi automaton $A_{\neg\varphi}$. Finally, one tests $\mathsf{empty}(A_K \cap A_{\neg\varphi})$. This correponds to teh language containment test $L(K) \subseteq L(\varphi)$, which is equivalent to the modelhood test $K \models \varphi$. The trick is that the complementation of the property, that is required for the language containment test, is pushed off from the complementation of a Büchi automaton to the negation of a temporal logic formula, which is trivial. This approach is used, for example, by the SPIN model checker [**?**]. The disadvantage is that the typically used temporal logic LTL is less expressive than Büchi automata, and hence the breadth of properties that can be expressed is limited. It has been stated that the expressivity of LTL is unsufficient for industrial verification applications [**?**].

For more information on model checking, as well as other formal verification techniques, we refer to the following works: [**?**][**?**][**?**].

As can be seen from these elaborations, having efficient procedures for Büchi complementation would be of great practical value. Even though handling the "worst-cases" will forever be unefficient,

## 1.2 Motivation

In the previous section we have seen that Büchi complementation is complex, and that it would be of practical value to better understand this complexity. In this section, we highlight the need for looking at this complexity in a way that has not received much attention in the past, namely empirically rather than theoretically.

In the following, we first present the traditional way of analysing the worst-case performance of complementation constructions, and then describe the empirical way for investigating their actual peformance.

This includes a review of the work that has been done so far. Note that we are using the terms complexity and performance interchangeably, and they both mean basically state growth.

### 1.2.1 Theoretical Investigation of Worst-Case Performance

The traditional performance measure for Büchi complementation constructions is their *worst-case state growth*[5]. This is the maximum number of states the construction *can* generate, in relation to the number of states of the input automaton.

For example, the initial complementation construction by Büchi (1962) [4] has a worst-case state growth of $2^{2^{O(n)}}$ does not mean that it produces a larger complement than Schewe's construction, for this concrete example. It might well be smaller. In fact, worst-case state complexities only allow to adequately deduce something about the specific worst-cases, and not about all the other automata. From a practical point of view, these worst cases are however not interesting, as their application is impracticable anyway (at least starting from a certain input automaton size). , where $n$ is the number of states of the input automaton. At this point, two short comments. First, the state growth is often not given as an exact function, but uses the big-O notation. Second, for notating state growths, we will consistenly use the variable $n$, whose meaning is the number of states of the input automaton. This means, for example, that for an input automaton with 8 states, the maximum number of states that the output automaton of Büchi's construction can have is $1.16 \times 10^{77}$ (if assuming the concrete function $2^{2^n}$).

Different constructions exhibit different worst-case state growths, and one of the main objectives of construction creators is to reduce this number. For example, the much more recent construction by Schewe (2009) [35] has a worst-case state growth as low as $(0.76n)^n + n^2$. Given an input automaton with 8 states, the maximum number of states of the output automaton is approximately 119.5 million.

A related objective of research is the quest for the theoretical worst-case state growth of Büchi complementation *per se*. A first result of $n!$ has been proposed in 1988 by Michel [20]. He proved that there exists a family of automata whose complement *cannot* have less than $n!$ states (these automata are known as Michel automata, and we will use them as part of the test data for our experiments). This proves a *lower bound* for the fundamental worst-case complexity of Büchi complementation, as it is not known whether the Michel automata are the real worst cases, or if there are even worse cases. Indeed, in 2007, Yan [55] proved a new higher lower bound of $(0.76n)^n$ (Michel's $n!$ corresponds to approximately $(0.36n)^n$ [55]). The worst-case state growth of a concrete construction naturally serves as an *upper bound* to a known lower bound. Given Schewe's number $(0.76n)^n n^2$, the lower bound of $(0.76n)^n$ by Yan is regarded as "sharp", as the gap between the lower and upper bound is very narrow, and consequently, the lower bound canot rise much anymore.

Many construction developers aim at bringing the worst-case state growth of their construction close to the currently known lower bound. It goes so far that a construction matching this lower bound is regarded as "optimal".

### 1.2.2 Need for Empirical Investigation of Actual Performance

Worst-case state growths are interesting from a theoretical point of view, but they are poor guides to the actual performance of a construction [42]. For example, if we have a concrete automaton of, say, 15 states, and we complement it with Schewe's construction, the fact that the worst-case state complexity is $(0.76n)^n n^2$ does not reveal anything about how the construction will perform on this concrete automaton. In any case, we are not expecting the complement to have 1.6 quintillion ($1.6 \times 10^{18}$) states (which would be the worst case), because this would most likely be practically infeasible.

Furthermore, if a construction has a higher worst-case state growth than another, it does not mean that it performs worse on a concrete case. In fact, worst-case state complexities only allow to adequately deduce the performance on the worst-case automata, but not on all the other automata. However, from a practical point of view, these worst cases are not interesting, as their application in practice is anyway infeasible [**?**] (at least starting from a certain input automaton size).

[5] As mentioned previously, also known as state complexity, state blow-up, or state explosion.

From a practical perspective we are interested how constructions perform on automata as they could occur in a concrete application of Büchi complementation, such as automata-theoretic model checking. This may include questions like the following. What is a reasonable complement size to expect for the given automaton with $n$ states? Are there generally easier and harder automata? What are the factors that make an automaton especially easy or hard? How does the performance of different constructions on the same automata vary? Are there constructions which are better suited for a certain type of automata than other constructions?

Questions like this can be attempted to answer by empirical peformance investigations. As its two most important elements this includes an *implementation* of the investigated constructions and *test data*. With test data, we mean a set of concrete automata on which the implementations of the constructions are run. The analysis is then done on the generated complement automata.

There have been relatively few empirical attempts in the history of Büchi complementation [42], compared to the number of theoretical works. In the following, we give (non-exhaustive) list of empirical works in the past that illustrate the approach, and also show the line of research in which the work of this thesis is situated.

**1995** Tasiran et al. [**?**] create an efficient implementation of Safra' construction[33] (determinisation-based) and used it for for automata-theoretic model checking tasks with the HSIS verification tool [**?**]. They state that they could easily complement property automata with some hundreds of states, however, they do not provide a statistical evaluation of the results.

**2003** Gurumurthy et al. [**?**] implement Friedgut, Kupferman, and Vardi's construction [16] (rank-based) along with various optimisations that they propose as a part of the tool Wring [**?**]. They complement 1000 small automata, generated by translation from LTL formulas, and evaluate exeuction time, and number of states and transitions of the complement for the different versions of the construction.

**2006** Althoff et al. [2] implement Safra's [33] and Muller and Schupp's [24] determinisation constructions[6] in a tool called OmegaDet, applied them on the Michel automata with 2 to 6 states, and compared the number of states of the determinised output automata.

**2008** Tsay et al. [45] carry out a first comparative experiment with the publicly available[7] GOAL tool [44][45][46][43]. They include the constructions by Safra [33] (determinisation-based), Piterman [28] (determinisation-based), Thomas [41] (WAPA[8]), and Kupferman and Vardi[16] (rank-based or WAA[9]). These constructions are pre-implemented in GOAL. As the test data, they use 300 Büchi automata, translated from LTL formulas, with an average size of 5.4 states. They evaluate and compare execution times, as well as number of states and transitions of the complements.

**2009** Kamarkar and Chakraborty [**?**] propose an improvement of Schewe's construction [35] (rank-based) and implement it, as well as Schewe's original construction, on top of the model checker NuSMV [**?**][**?**]. They run the constructions on 12 test automata and compare the sizes of the complements. Furthermore, they run the same tests with the constructions by Kupferman, and Vardi [16] (rank-based or WAA) and Piterman [28] (determinisation-based) within GOAL, and compare the results to the ones of their implementation of Schewe's construction.

**2010** Tsai et al. [42] (paper entitled "State of Büchi Compelentation") carry out another experiment with GOAL. They compare the constructions by Piterman [28] (determinisation-based), Schewe [35] (rank-based), and Vardi and Wilke [52] (slice-based), with various optimsiations that they propose in the same paper. As the test data, they use 11,000 randomly generated automata with 15 states and an alphabet size of 2. The test set is organised into 110 automata classes that consist of the combinations of 11 transition densities and 10 accceptance densities. This test set is repeatedly used in subsequent work (including in this thesis), and we will refer to it as the GOAL test set (because it has been generated with the GOAL tool). Tsai et al. provide sophisticated evaluation of the states of the complements for all the tested constructions and construction versions.

[6]These determinisation constructions transform a non-deterministic Büchi automaton to a deterministic Rabin automataon (see Section **??**), however, the are used as the base for determinisation-based complementation constructions.

[7]http://goal.im.ntu.edu.tw/wiki/doku.php

[8]Via **W**eak **A**lternating **P**arity **A**utomaton

[9]Via **W**eak **A**lternating **A**utomaton

**2010** Breuers [?] proposes an improvement for the construction by Sistla, Vardi, and Wolper [37] (Ramsey-based), and creates an implementation of it. He generates his own test data (inspired by the work of Tsai et al. [42]) consisting of *easy*, *medium*, and *hard* automata, based on different transition density and acceptance density values. He evaluates the complement sizes produced by the construction for autoamta of sizes 5, 10, and 15 of all these difficulty categories.

**2012** Breuers et al. [3] wrap the implementation of their improvement of Sistla, Vardi, and Wolper's construction [37] in the publicly available tool Alekto[10], and and run it on the GOAL test set. They compare the generated complement sizes, as well as the number of aborted complementation tasks (due to exceeding resource requirements) to the corresponding result for different constructions on the same test set by Tsai et al. [42].

**2013** Göttel [8] creates a C implementation of the Fribourg construction [1], including the R2C optimisation (see Chapter **??**), and executes it on the GOAL test set, as well as on the Michel automata with 3 to 6 states. He analyses the resulting complement sizes and execution times separately for each of the 110 classes that the GOAL test set consists of. The Fribourg construction[11] is a slice-based complementation construction that is being developed at the university of Fribourg, and which lies at the heart of this thesis. The entire Chapter **??** of this thesis is dedicated to explaining the Fribourg construction.

## 1.3 Aim and Scope

The aim of this thesis is an in-depth empirical performance investigation of the Fribourg construction. As mentioned, the Fribourg construction is a Büchi complementation construction that is being developed at the University of Fribourg [1]. By empirically investigating the behaviour of this specific construction, we want to follow up the track of empirical research that we have outlined in the last section.

This thesis is certainly not sufficient to describe the performance of the Fribourg construction in its entiretey, or in a way that is adequate to be relied on in industiral applications. Neither this thesis can answer general questions about the observed behaviour of Büchi complementation. Rather, we see this piece of work as a mosaic stone that we add to the very complex and multi-faceted picture of the complexity of Büchi complementation.

The empirical performance investigation will include testing of different versions of the construction, and comparison with other complementation constructions...

Aim: empirical performance investigation of a specific Büchi complementaiton construction, comparison with other constructions

Scope: two test sets, relatively small automata, no real world or "typical" examples,

## 1.4 Overview

[10] http://www.automata.rwth-aachen.de/research/Alekto/

[11] The authors of the constructions use the name *subset-tuple construction* (see [1]), however, in this thesis, we will use the name *Fribourg construction*.

# Chapter 2

# Background

## Contents

In this chapter we treat several topics that serve as a background for the rest of the thesis. In particular, the goal of this chapter is to set the stage for our description of the Fribourg construction in Chapter **??**, and the setup of our empirical performance investigation of the Fribourg construction in Chapter **??**, as well as its results in Chapter **??**.

In Section 2.1 we summarise some aspects about Büchi automata, as well as about other types of automata on infinite words, that are relevant to the purpose of Büchi complementation. In Section 2.2, we describe the principal techniques for run analysis of non-deterministic automata. This topic has a particular relation to Büchi complementation, and one of the run analysis techniques, reduced split trees, is the core of the Fribourg construction. In Section 2.3, we provide a review of proposed Büchi complementation constructions from the introduction of Büchi automata in 1962 until today. We organise the presentation of these constructions along the four main Büchi complementation approaches Ramsey-based, determinisation-based, rank-based, and slice-based. Some of the constructions that we describe in this section will be used in the performance investigation of the Fribourg construction in Chapter **??**.

## 2.1 Büchi Automata and Other Omega-Automata

Büchi automata are a type of $\omega$-automata. These are fninite state automata that run on infinite words (so-called $\omega$-words). Externally, $\omega$-automata look the same as the traditional finite state automata on finite words. It is possible to interpret any such automaton on finite words as an $\omega$-automaton, and vice versa.

The difference between $\omega$-automata and automata on finite words is their acceptance condition. An automaton on finite words accepts a word, if after finishing reading it, the automaton is in an accepting state. For $\omega$-automata, this acceptance condition is not possible, because an $\omega$-automaton never finishes reading a word (because the word "never ends"). Instead, the acceptance condition of $\omega$-automata is defined on the set of the so-called *infinitely recurring states*. We are going to describe this concept in Subsection 2.1.1 below.

In Subsection 2.1.1 of this section, wer first treat Büchi automata, and in Subsection 2.1.2 the principal other types of $\omega$-automata, in particular, Muller, Rabin, Streett, and parity automata. In the latter subsection, we also introduce a shorthand notation for different types of $\omega$-automata that we will use throughout the thesis.

Note that in the entire section we omit overly formal notation and proofs of any kind, because it is not necessary for the aim of this thesis. More comprehensive and formally rigorous treatments of $\omega$-automata can be found, for example, in the works by Thomas [39, 40], or Wilke [53].

### 2.1.1 Büchi Automata

Below we summarise the aspects of Büchi automata that are most significant for our purposes, including the acceptance condition of Büchi automata, the expressivity of non-deterministic and deterministic Büchi automata, and basics about the complementation of non-deterministic and deterministic Büchi automata. In the course of this, we always stress the difference between deterministic and non-deterministic Büchi automata, as this is one of the main sources for the intricacy of Büchi complementation [25].

#### Definition and Acceptance Condition

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

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

- $F$: a set of accepting states, $F \subseteq Q$

Note that this is the same definition as for non-deterministic finite state automata on finite words [10]. The difference between Büchi automata and automata on finite words is only the acceptance condition of Büchi automata that we describe below. For deterministic Büchi automata, the definition is similar to the above one, but with a different transition function $\delta$ that returns none or a single state, instead of a set of states

An important concept in automata theory the notion of a *run* of an automaton on a given word. A run $\rho$ of automaton $A$ on word $\alpha$ is a sequence of states $qinQ$ that $A$ visits in the process of reading $\alpha$. For finite words, the length of a run is finite as well, however, for $\omega$-words, the length of a run may be infinite. Note that deterministic automata have at most one run for a given word, whereas non-deterministic automata may have multiple possible runs for the same word.

The Büchi acceptance condition decides whether a run $\rho$ is accepting or non-accepting. This in turn determines the acceptance or non-acceptance of a word: a word $\alpha$ is accepted by an automaton $A$, if and only if it has at least one accepting run in $A$. The decision whether a run $\rho$ is accepting or non-accepting is based on the set of *infinitely recurring states* of $\rho$ that we denote by $\mathsf{inf}(\rho)$. This set contains all the states that occur infinitely often in $\rho$. In particular, the Büchi acceptance condition is as follows:

$$\text{Run } \rho \text{ is accepting} \iff \mathsf{inf}(\rho) \text{ contains at least one accepting state}$$

That is, a run is accepting, if and only if the run contains at least one accepting state infintely often. Formally, this can be written as $\mathsf{inf}(\rho) \cap F \neq \varnothing$.

An intuitive way for describing the Büchi acceptance condition has been given by Vardi [49]. If we imagine the automaton having a green light that blinks whenever the automaton visits an accepting state, then the run is accepting if we observe the green light blinking infinitely many times. The fact that there are only finitely man accepting states, but the light blinks infinitely many times, implies that at least one accepting state is being visited infinitely often.

#### Expressivity

A particularity of Büchi automata is that deterministic and non-deterministic automata are *not* expressively equivalent. In particular, the class of languages corresponding to the deterministic Büchi automata is a strict subset of the class of languages corresponding to the non-deterministic automata. This result has been proved by Büchi himself in his 1962 paper [4].

This contrasts, for example, with finite state automata on finite words. In this case, non-deterministic and deterministic automata are expressively equivalent, and consequently every non-deterministic automaton can be turned into an equivalent deterministic automaton. With Büchi automata, however, this is not possible, because there exist languages that can be expressed by a non-deterministic Büchi automaton, but not by a deterministic one. An example of such a language is $(0 + 1)^*1^\omega$, that is, the language of all words of 0 and 1 ending with an infinite sequence of 1. A non-deterministic automaton representing this language, cannot be turned into an equivalent deterministic automaton [49, 32]. Because of this fact, we say that Büchi automata can *in general* not be determinised to other Büchi automata. This fact has implications on the complementation of non-deterministic Büchi automata, as we will see below.

The class of languages that is equivalent to the *non-deterministic* Büchi automata is the class of *$\omega$-regular languages*. A formal description of the $\omega$-regular languages can be found, for example, in [39, 40, 53]. Regarding deterministic Büchi automata, consequently the set of languages that is equivalent to them is a strict subset of the $\omega$-regular languages.

#### Complementation

Non-deterministic Büchi automata are closed under complementation. This means that the complement of every non-deterministic Büchi automaton is another non-deterministic Büchi automaton. This result has been proved by Büchi in his 1962 paper [4][1]. Deterministic Büchi automata, on the other hand, are not closed under complementation [39]. In particular, this means that the complement of a deterministic Büchi automaton is still a Büchi automaton, however, possibly a non-deterministic one.

As we already mentioned, the algorithmic difficulty and complexity of complementation is very different for deterministic and non-deterministic automata. For deterministic Büchi automata, there exists a simple procedure, introduced in 1987 by Kurshan [17], that can complement a deterministic Büchi automaton to a non-deterministic Büchi automaton in polynomial time and linear space.

For non-deterministic Büchi automata, however, there exists no easy solution. The main reason is that Büchi automata can in general not be determinised. If they could be determinised, then a solution would be to transform a non-deterministic Büchi automaton to an equivalent deterministic one, and complement the deterministic Büchi automaton with Kurshan's construction. This is by the wy the approach that is used for the complementation of non-deterministic automata on finite words: determinise a non-deterministic automaton with the subset construction [29], and then trivially complement the deterministic automaton by making the accepting states non-accepting, and vice versa. Unfortunately, for Büchi automata such a simple procedure is not possible, and this can be seen as the main reason that Büchi complementation is such a hard problem [25].

### 2.1.2 Other Omega-Automata

After the introduction of Büchi automata in 1962, several other types of $\omega$-automata have been introduced. These automata differ from Büchi automata only in their acceptance condition, that is, in the way they decide whether a run $\rho$ is accepting or non-accepting. All these acceptance condition are however based on the set of infintely recurring states $\mathsf{inf}(\rho)$ of $\rho$. The following are the most important of these alternative $\omega$-automata along with the year of their introduction.

- Muller automata (1963) [22]
- Rabin automata (1969) [30]
- Streett automata (1982) [38]
- Parity automata (1985) [21]

Some of these automata types are used in complementation constructions, especially in determinisation-based complementation constructions (see Section 2.3). Table 2.1 lists the Muller, Rabin, Streett, and parity acceptance conditions, along with the Büchi acceptance condition for comparison.

| Type | Definitions | Condition |
|---|---|---|
| Muller | $U \subseteq 2^Q$ | $\mathsf{inf}(\rho) \in U$ |
| Rabin | $\{(U_1, V_1), \ldots, (U_r, V_r)\}$, $U_i, V_i \subseteq Q$ | $\exists i : \mathsf{inf}(\rho) \cap U_i = \varnothing \wedge \mathsf{inf}(\rho) \cap V_i \neq \varnothing$ |
| Streett | $\{(U_1, V_1), \ldots, (U_r, V_r)\}$, $U_i, V_i \subseteq Q$ | $\forall i : \mathsf{inf}(\rho) \cap U_i \neq \varnothing \vee \mathsf{inf}(\rho) \cap V_i = \varnothing$ |
| Parity | $\pi : Q \to \{1, \ldots, k\}$, $k \in \mathbb{N}$ | $\mathsf{min}(\{\pi(q) \mid q \in \mathsf{inf}(\rho)\}) \bmod 2 = 0$ |
| Büchi | | $\mathsf{inf}(\rho) \cap F \neq \varnothing$ |

Table 2.1: Acceptance conditions of Muller, Rabin, Streett, parity, and Büchi automata. Note that $Q$ denotes the set of states and $F$ the set of accepting states of the corresponding automaton.

Below, we briefly explain the acceptance conditions of Muller, Rabin, Streett, and parity automata in words. More detailed descriptions of these conditions can be found, for example, in [**?**], or [53].

**Muller acceptance condition** Includes a set $U$ of subsets of states. A run $\rho$ is accepting if and only if $\mathsf{inf}(\rho)$ equals one of the pre-defined subsets in $U$. The Muller acceptance condition is the most general one, and the Rabin, Streett, parity, and Büchi acceptance conditions can be expressed as Muller acceptance conditions [**?**, 53].

[1] Actually, the proof of closure under complementation of non-deterministic Büchi automata was a necessary step for Büchi to prove the equivalence of Büchi automata and S1S formulas (Büchi's Theorem). In order to prove this closure under complementation, Büchi described the first Büchi complementation construction in history.

**Rabin acceptance condition** Inludes a list of pairs $(U, V)$ where $U$ and $V$ are subsets of states. A run $\rho$ is accepting if and only if *there is* at least one pair for which $U$ does not contain any states of $\mathsf{inf}(\rho)$ *and* $V$ contains at least one state of $\mathsf{inf}(\rho)$. A pair $(U, V)$ is called a Rabin pair.

**Streett acceptance condition** Inludes a list of pairs $(U, V)$ where $U$ and $V$ are subsets of states. A run $\rho$ is accepting if and only if *for all* the pairs either $U$ contains at least one state of $\mathsf{inf}(\rho)$ *or* $V$ does not contain any states of $\mathsf{inf}(\rho)$. A pair $(U, V)$ is called a Streett pair. Note that the Streett condition is the dual of the Rabin condition. This means that if we have two identical Rabin and Streett automata with an identical list of pairs, then the Streett automaton accepts the complement language of the Rabin automaton [13].

**Parity acceptance condition** Assigns a number to each of the states of the automaton. A run $\rho$ is accepting if and only if the smallest-numbered element of $\mathsf{inf}(\rho)$ has an even number. The numbers that are assigned to the states are sometimes called colours [18].

Regarding the expressivity of these automata, it turns out that they are all equivalent to non-deterministic Büchi automata, and thus to the $\omega$-regular languages [53]. This holds for non-determnistic *and* deterministic automata of these types. That means that, unlike Büchi automata, deterministic and non-deterministic Muller, Rabin, Streett, and parity automata are expressively equivalent.

At this point we introdue a notation that we will occasionally use for denoting different types of $\omega$-automata. This notation has been used by Piterman [27] and later by Tsai et al. [42]. It consists of three-letter acronymes of the form

$$\{N, D\} \times \{B, M, R, S, P\} \times W$$

The first letter specifies whether the automaton is non-deterministic $(N)$ or deterministic $(D)$. The second letter stands for the acceptance condition: $B$ for Büchi, $M$ for Muller, $R$ for Rabin, $S$ for Streett, and $P$ for parity. The last letter specifies on which structure the automaton runs. In our case these are always words, thus the last letter is always $W$. For example, NBW means non-deterministic Büchi automaton, DBW means deterministic Büchi automaton, NMW means non-deterministic Muller automaton, DMW means deterministic Muller automaton, and so on.

## 2.2 Run Analysis of Non-Deterministic Automata

As mentioned, in a deterministic automaton, every input word has at most one run. In a non-deterministic automaton, however, every input word may have multiple runs. The analysis of the different runs of a non-deterministic automaton on a given input word is called *run analysis* and is central to the Büchi complementation problem.

The reason that run analysis is central to Büchi complementation is as follows. In a Büchi complementation task, we are given a non-deterministic Büchi automaton $A$, and we attempt to construct its complement $B$. For constructing $B$ we have in fact to decide for every word $\alpha$ whether $B$ must accept it or not. This decision is intrinsically tied to the entirety of the runs of $A$ on $\alpha$ in the following way:

$$B \text{ accepts } \alpha \iff \textit{All} \text{ the runs of } A \text{ on } \alpha \text{ are } \textit{non-accepting}$$

$B$ must accept a word $\alpha$, if and only if *all* the runs of $A$ on $\alpha$ are non-accepting. If, for example, $A$ has 10 runs on $\alpha$, and 9 of them are non-accepting and one is accepting, then $A$ still accepts the word $\alpha$, and consequently, $B$ must not accept it. Only if all of the 10 runs of $A$ on $\alpha$ are non-accepting, $A$ does not accept $\alpha$, and consequently $B$ must accept $\alpha$. This means that for constructing the complement $B$, we need to consider *all* the runs of the input automaton $A$ on specific words, and the way this can be done is by run analysis on $A$.

In this section, we present the principal run analysis techniques for non-deterministic Büchi automata [53]. We start with *run DAGs* (DAG stands for directed acyclic graphs) in Section 2.2.1. Then in Sections **??**, 2.2.3, and 2.2.4, we present three techniques based on trees with increasing sophistication, namely *run trees*, *split trees*, and *reduced split trees*. The last of the tree techniques, reduced split trees, lies at the heart of the Fribourg construction that we describe in Chapter **??**.

In the following subsections, we will give examples for the different run analysis techniques that are based on the non-deterministic Büchi automaton in Figure 2.1. Note that the alphabet of this automaton is $\Sigma = \{a\}$, and thus the only $\omega$-word in $\Sigma^\omega$ is $a^\omega$. However, the automaton has mutliple (infinitely many) runs for this word.

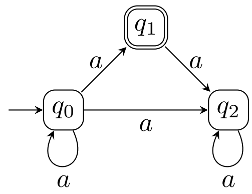


Figure 2.1: Non-deterministic Büchi automaton $A$ used as example autoamton in this section.

### 2.2.1 Run DAGs

Run DAGs arrange all the runs of an automaton $A$ on a word in a directed acyclic graph (DAG). This graph can be thought of as a matrix with rows and columns. The rows are called levels, and each column corresponds to a state of $A$. Consequently, the width (that is, number of columns) of a run DAG equals the number of states of $A$. Each level $i$ (starting from 0) corresponds to the situation after reading the first $i$ symbols of the word. Figure 2.2 shows the first five levels of the run DAG of the example automaton $A$ in Figure 2.1 on the word $a^\omega$.

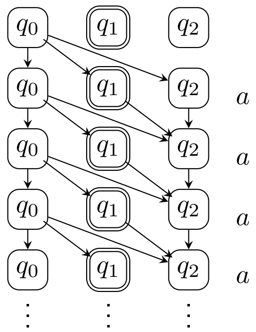


Figure 2.2: First few levels of the run DAG for the runs of automaton $A$ (Figure 2.1) on the word $a^\omega$.

Note that throughout this thesis, we are using rectangles with rounded corners for states of automata, and rectangles with sharp corners for vertices of graphs nodes of trees. In graphs and trees, we will however indicate vertices or nodes that correspond to accepting automaton states with double-lined rectangles.

As can be seen in Figure 2.2, every path of a run DAG corresponds to a run of the automaton on the given word. A run DAG is a structure that is able to represent an infinte number of runs by keeping a finite width. The number of levels of a run DAG is infinite for $\omega$-words. A formal description of run DAGs can be found, for example, in [5].

Run DAGs are the basic structure for the rank-based complementation constructions, that we review in Section 2.3.3. Regarding this application, the fact that run DAGs have finite width is important, because the levels of run DAGs are mapped to states of the output automaton in these complementation constructions.

### 2.2.2 Run Trees

Trees in general are, after DAGs, the second structure that is used for run analysis. Run trees are the most basic of these tree structures. A run tree is basically a direct unwinding of all the runs of an automaton on a word as a tree. Figure 2.3 shows the first five levels of the run tree for the automaton $A$ in Figure 2.1 on the word $a^\omega$.

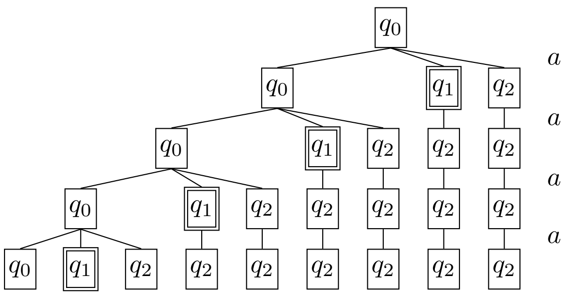


Figure 2.3: First five levels of the run tree for the runs of automaton $A$ (Figure 2.1) on the word $a^\omega$.

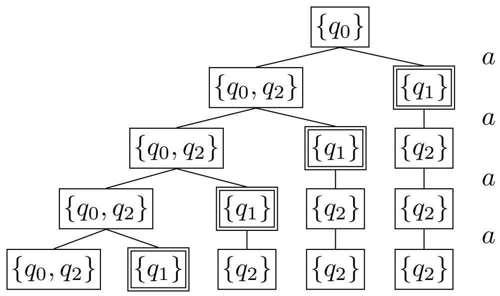


Figure 2.4: First five levels of the split tree for the runs of automaton $A$ (Figure 2.1) on the word $a^\omega$.

As can be see in Figure 2.3, there is a one-to-one mapping of paths in a run tree (from the root to the leaves) to runs of the corresponding automaton. This means that if the automaton has an infinite number of runs on a given word, then the corresponding run tree has an infinite maximum width. This makes run trees impractical to be used in Büchi complementation. The following tree techniques, split trees and reduced split trees, sacrifice a part of the information about individual runs, with the benefit of making the tree structure more compact.s

### 2.2.3 Split Trees

A split tree is basically a run tree where the accepting and non-accepting children of every node are merged. Consequently, a node of a split tree does not represent a single state of the automaton, but a set of states. The reduction of the number of children to at most two, makes the split tree furthermore a binary tree. Figure 2.4 shows the first five levels of the split tree of the automaton $A$ (Figure 2.1) on the word $a^\omega$.

Split trees sacrifice a part of the information about individual runs. For example, in Figure 2.4, we see that there must be a run that starts in $q_0$ (root node) and is in $q_0$ after reading four symbols (leftmost node on fifth level). However, the split tree does not provide the exact sequence of states. All that it reveals is that the sequence is $q_0 q_?^1 q_?^2 q_?^3 q_0$, where each $q_?^i$ is either $q_0$ or $q_2$. The run tree in Figure 2.3, on the other hand, shows the actual sequence unambiguously, which is $q_0q_0q_0q_0q_0$.

However, the split tree still provides an important piece of information, namely that the sequence $q_0 q_?^1 q_?^2 q_?^3 q_0$, whathever it actually looks like, does not contain any accepting states, because $q_?^i$ can only be $q_0$ or $q_2$, which are both non-accepting. It turns out that, with regard to the application of run analysis to Büchi complementation, this information is sufficient.

Split trees in fact embody a modified subset construction that does not mix accepting and non-accepting states. The result of applying such a construction to a non-deterministic automaton is an equivalent non-deterministic automaton whose degree of non-determinism is reduced to two (which means that each state has at most two successors for a given alphabet symbol). Such a construction has been described in [48].

A formal description of split trees can be found, for example, in [52] or [5]. Split trees are clearly more compact than run trees. However, their width may still grow infinitely. In the next section, we present a further reduction of the information contained in the tree, that bounds the maximum width of the tree to a finite number.

### 2.2.4 Reduced Split Trees

Reduced split trees are split trees that sacrifice even more information. However, the information that is retained is still sufficient for Büchi complementation. The rule is that on each level, going from right to left or from left to right, only the first occurrence of each state is kept, and the others are omitted. This bounds the maximum width of the tree to the number of states of the corresponding automaton.

The direction in which the first occurrence of a state is determined depends on whether right-to-left or left-to-right split trees are used. In *right-to-left split trees*, the accepting child is put to the right of the non-accepting child. The split tree in Figure 2.4 is thus a right-to-left split tree. In *left-to-right split trees*, on the other hand, the accepting child is put to the left of the non-accepting child. In the literature both versions are used. For example, Vardi and Wilke [52] describe the left-to-right version, whereas Fogarty et al. [5] describe the right-to-left version. In this thesis, we will use exclusively the right-to-left version.

In a right-to-left reduced split tree, the levels are processed from right to left. This means that only the rightmost occurrence of each state on a level is kept and the others are omitted. Figure 2.5 shows the first five levels of the reduced split tree of the automaton $A$ (Figure 2.1) on the word $a^\omega$.

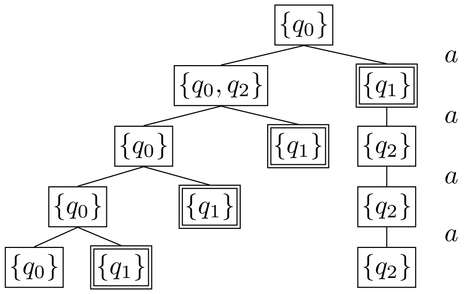


Figure 2.5: First five levels of the reduced split tree for the runs of automaton $A$ (Figure 2.1) on the word $a^\omega$.

As can be seen in Figure 2.5, each level contains at most one occurrence of each of the states $q_0$, $q_1$, and $q_2$. Comparing the reduced split tree in Figure 2.5 with the split tree in Figure 2.4 reveals which states have been omitted in the reduced split tree. The root level and level 1 are similar in both trees. On level 2, the state $q_2$ in the leftmost node is omitted. This is because there is already a $q_2$ farther to the right, namely in the rightmost node of level 2. In level 3, there are two omissions of $q_2$ for the same reason. One of them causes an entire node to disappear, because this node contained $q_2$ as its only state. Finally, on level 4, there are three omissions of $q_2$.

In the following, we illustrate what this omission of states entails and why the resulting "truncated" run analysis is still usabel for Büchi complementaton. By omitting states, we obviously omit runs from the tree. This omission of runs is targeted at so-called *re-joining runs*. Re-joining runs are runs that afer a certain number of steps end up in the same state again. For example, the automaton from Figure 2.1 has seven re-joining runs that after four steps end up in the state $q_2$. This can be easily seen in the corresponding run tree in Figure 2.3. A reduced split tree keeps exactly one of a group of re-joining runs and removes the others. This can be seen in Figure 2.5 which contains only one run from the root to $q_2$ in four steps.

The run that is kept is named *rightmost run*. The name refers to the fact that this run is the first of a group of re-joining runs that makes a *right turn* in the corresponding path of the tree. A right turn means the transition from a parent to an accepting child, which are situated to the right of the non-accepting child, hence the name[2]. The rightmost run of a group of re-joining runs has the following crucial property:

$$\text{Any re-joining run is accepting} \Longrightarrow \text{The rightmost re-joining run is accepting}$$

[2]Note that everything we explain here is also valid for left-to-right split trees, but one has to exchange *right*, *rightmost*, and *right turn* with *left*, *leftmost*, and *left turn*, respectively.

That is, if a group of re-joining runs contains an accepting run, then the rightmost run is accepting too. On the other hand, if the rightmost run is non-accepting, then all the other re-joining runs are non-accepting as well. A proof of this relation can be found, for example, in [52][3].

Practically, this means that we can reduce the existential question about the presence of acceptance in a group of runs to the rightmost run. Remember that for Büchi complementation we are interested in exactly such an existential question: is there an accepting run of the automaton $A$ on the word $\alpha$? If yes (it does not matter how many accepting runs there are), then $A$ accepts $\alpha$ and the complement must not accept it. If no, then $A$ does not accept $\alpha$ and the complement must accept it. By considering only the rightmost runs, we can still reliably answer this question.

Thus, reduced split trees include a tremendous reduction of the number of runs to be analysed, but still retain all the needed information for Büchi complementation. Most importantly, this reduction of runs bounds the width of the trees to a finite number, what makes them usable in Büchi complementation constructions, such as the Fribourg construction (see Chapter **??**), and other slice-based constructions (see Section 2.3.4).

Note that an alternative name for rightmost run is *greedy run* [1]. This name refers to the fact that the rightmost run is the first of a group of re-joining runs that visits an accepting state, what makes it "greedy". It is interesting to note that reduced split trees are related to Muller-Schupp trees that are used in the Büchi determinisation construction by Muller and Schupp [24] (cf. [52, 5]). As a last remark, formal descriptions of reduced split trees can be found, for example, in [52] and [53] (left-to-right version), or [5] (right-to-left version).

This concludes our presentation of run analysis techniques. In the next section we will review the most prominent Büchi complementation constructions that have been proposed over the past 50 years. Many of them are based on the run analysis techniques that we have described in the present section.

## 2.3 Review of Büchi Complementation Constructions

Since the introduction of Büchi automata in 1962, many different Büchi complementation constructions have been proposed. In this section, we will list and briefly review some of the most important of them. To this end, we will organise the constructions into four approaches that have also been used in

| | | |
|---|---|---|
| Ramsey-based approach | | |
| Büchi | 1962 | [4] |
| Sistla, Vardi, and Wolper | 1987 | [36, 37] |
| Determinisation-based approach | | |
| Safra | 1988 | [33, 34] |
| Muller and Schupp | 1995 | [24] |
| Piterman | 2007 | [27, 28] |
| Rank-based approach | | |
| Klarlund | 1991 | [12] |
| Kupferman and Vardi | 1997/2001 | [15, 16] |
| Thomas | 1999 | [41] |
| Friedgut, Kupferman, and Vardi | 2006 | [6, 7] |
| Schewe | 2009 | [35] |
| Slice-based approach | | |
| Vardi and Wilke | 2007 | [52] |
| Kähler and Wilke | 2008 | [11] |

[3] Lemma 2.6

### 2.3.1 Ramsey-Based Approach

Constructions of the Ramsey-based approach are based on a branch of combinatorics called Ramsey theory [9], hence the name Ramsey-based approach[4]. Constructions of this approach proceed by directly constructing the complement automaton out of the input automaton (this is unlike determinisation-based constructions, as we will see in the next subsection). The Ramsey-based approach is the oldest approach and includes the earliest Büchi complementation constructions, including Büchi's own construction [4].

#### Büchi (1962)

The first Büchi complementation construction in history, described by Büchi in his 1962 paper [4], is based on a Ramsey-based combinatorial argument from [31]. The construction is rather complicated [51], and has a doubly-exponential worst-case state growth of $2^{2^{O(n)}}$ [50]. This blow-up is enormous, for example, given an automaton with 9 states, the maximum size of the complement is $1.3 \times 10^{154}$ states, tremendously more as there are atoms in the observable universe[5].

However, as mentioned in our discussion about the use of worst-case state growth as a performane measure for complementation constructions in Section 1.2, this number applies only to the worst case, which is a very extraordinary special case. Furthermore, we believe that Büchi's aim was not to create an *efficient* construction, but rather to prove that such a construction exists in the first place. Because for Büchi this served as a proof that Büchi automata are closed under complementation, which was in turn required to prove Büchi's Theorem [52].

#### Sistla, Vardi, and Wolper (1987)

In 1987, Sistla, Vardi, and Wolper proposed an improvement of Büchi's construction [37][6]. This construction was the first to include only an exponential, rather than doubly-exponential, worst-case state growth. The state growth function of the construction is $O(16^{n^2})$ [37]. Thus, given an automaton with 9 states, the maximum size of the complement is $3.4 \times 10^{97}$ states.

### 2.3.2 Determinisation-Based Approach

The determinisation-based approach achieves complementation by chaining several conversions between different types of $\omega$-automata. The most important of these conversions is the transformation of the non-deterministic Büchi automaton to a deterministic $\omega$-automaton of a different type. Thus, the first step is to determinise the input automaton, hence the name determinisation-based approach.

Note that in this context *determinisation* refers to the conversion of a Büchi automaton to a *different type* of $\omega$-automata, such as Muller, Rabin, Streett, or partity (see 2.1.2). As mentioned in Section 2.1.1, the determinisation of a Büchi automaton to a deterministic Büchi automaton is in general not possible, because determnistic Büchi automata are less expressive than non-deterministic Büchi automata.

This approach harnesses the fact that the complementation of deterministic automata is generally easier than the complementation of nono-deterministic automata. This means that once the non-deterministic Büchi automaton is turned into a deterministic $\omega$-automaton, it can be easily complemented, and the result can then be converted back to a non-deterministic Büchi automaton.

#### Safra (1988)

The first determinisation-based complementation construction has been described in 1988 by Safra [33, 34]. Safra's main work is actually a construction for converting a non-deterministic Büchi automaton

[4]Ramsey was a British mathematician who lived at the beginning of the 20th century

[5]Assuming a number of $10^{80}$ atoms in the observable universe, according to http://www.wolframalpha.com/input/?i=number+of+atoms+in+the+universe.

[6]This paper was preliminarily published in 1985 [36].

(NBW) to a deterministic Rabin automaton (DRW)[7]. This determinisation construction is what today commonly is known as *Safra's construction*. However, Safra also describes a series further conversions that, when chained together, result in a complementation construction. The complete series of conversions is 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. $\overline{\text{DRW}}$ $\longrightarrow$ $\overline{\text{NBW}}$

That means, the DRW resulting from Safra's determinisation construction is complemented to a DSW, which is converted to a DRW, which in turn is converted to an NBW. The resulting NBW is the complement of the input NBW. Note that a horizontal indicates that the automaton accepts the complement language of the automata without a bar.

Safra's construction is in fact a modified subset construction that guarantees the equivalence of the input and output automata[8]. The main difference of Safra's construction to the classical subset construction is that the states of the output automaton are labelled by trees of subsets of states, rather than just subsets of states. These trees are called *Safra trees*. Detailed analyses and explanations of Safra's construction can be found, next to Safra's papers [33, 34], in [32], [13], [2], or [?].

The complementation step from the DRW to the complement DSW can be trivially done by interpreting the Rabin acceptance condition as a Streett acceptance condition [13]. This works, because, as mentioned in Section 2.1.2, the two acceptance conditions are the duals of each other. The final conversions from DSW to DRW, and DRW to NBW are described by Safra in [33, 34] or alternatively in [13].

The entire construction from an NBW to a complement NBW has a worst-case state complexity of $2^{O(n \log n)}$. With this result, Safra's complementation construction matched the lower bound of $n!$ that has been introduced in the same year by Michel [20]. However, as pointed out by Vardi [50], the big-$O$ notation hides a large gap between the two functions, and Safra's complementation construction is thus not as "optimal" as it seems. Thus, the quest for finding Büchi complementation constructions with an even lower worst-case state complexity continued.

#### Muller and Schupp (1995)

In 1995, Muller and Schupp proposed and improvement of Safra's determinisation construction [24]. This construction can be used at the place of Safra's original construction in the conversation chain that we described above for Safra's construction. Muller and Schupp's construction uses *Muller-Schupp trees* instead of Safra trees. While the principles of the two constructions are very similar, it is said that Muller and Schupp's construction is simpler and more intuitive than Safra's construction [32]. However, the drawback is that in many concrete cases Muller and Schupp's construction produces larger output automata than Safra's construction [2]. The worst-case state complexity of Muller and Schupp's construction is, similarly to Safra's construction, $2^{O(n \log n)}$. A detailed comparison of the determinisation constructions by Muller and Schupp, and Safra can be found in [2].

#### Piterman (2007)

A further improvement of Safra's determinisation construction has been proposed by Piterman from EPF Lausanne in 2007 [28][9]. The main difference is that Piterman's construction converts the input Büchi automaton to a deterministic parity automaton (DPW), rather than a deterministic Rabin automaton (DRW). Furthermore, Piterman uses a more compact version of Safra trees, what results by trend in smaller output automata. Also in terms of worst-case complexity, Piterman's construction is more efficient

[7] This is the notation for $\omega$-automata types that we introduced in Section 2.1.2.

[8] Safra shows in his papers [33, 34] nicely that applying the classical subset construction to an NBW $A$ results in a DBW $B$ that may accept some words that are not accepted by $A$. This means that the subset construction is not *sound* with respect to Büchi automata.

[9] A preliminary version of this paper has appeared in 2006 [27].

than Safra's construction. The blow-up in Piterman's construction from the NBW to the DPW is $2n^n n!$, whereas in Safra's construction the blow-up from the NBW to the DRW is $12^n n^{2n}$ [27, 28].

Since Piterman's determinisation construction produces a DPW, it entails different conversions for complementation than Safra's construction. In particular, the procedure is as follows [42]:

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

The complementation of the DPW can be trivially done by increasing the number assigned to each state by one [42] (see Section 2.1.2 for an explanation of the parity acceptance condition). The complemented DPW can then be converted directly to an NBW [42].

### 2.3.3 Rank-Based Approach

The rank-based approach is based on run analysis with run DAGs (we presented run DAGs in Section 2.2.1). The basic "mechanics" of rank-based construction is similar to the subset construction, that is, the output automaton is constructed state by state, by adding to every state a successor state for every symbol of the alphabet. In rank-based constructions, these states are derived from levels of a run DAG. The idea is that the run analysis with run DAGs is interweaved with the construction, so that the states of the output automaton represent levels of "virtual" run DAGs.

As we have seen in Section 2.2, the purpose of run analysis is to find out whether *all* runs of an input automaton $A$ on a word $\alpha$ are non-accepting. Because in this case, the complement $B$ must accept $\alpha$, and in all other cases it must not accept it. In rank-based constructions this is achieved by the means of natural numbers called *ranks* that are assigned to the vertices of a run DAG (hence the name rank-based approach). These ranks are assigned in such a way that vertices corresponding to *accepting* states only get *even* ranks. Furthermore the ranks along paths of the run DAG are monotonically decreasing. The effect of this is that every path gets eventually trapped in a rank. If for a given path this trapped rank is odd, it means that the corresponding run does not visit any accepting states anymore starting from a certain point, and thus is non-accepting. If all the paths of a run DAG get trapped in an odd rank, then all the runs of the automaton on the given word are non-accepting. This situation is called *odd-ranking* and indicates that the complement must accept the word.

This basic idea is the same for all rank-based construction and described in many places, for example, [5, 16, 7, 50, **?**, 35]. The individual rank-based constructions differ mainly in details of how the ranking is done.

#### Klarlund (1991)

Klarlund's construction [12] rather foreshadowed the rank-based constructions than being one itself. It was one of the first constructions that was neither Ramsey-based nor determinisation-based [12, 16]. Its fundamental idea is very similar to the later rank-based constructions, but it does not use the term "rank" and is not explicitely based on run DAGs. However, Klarlund uses so-called *progress measures*, which have a very similar role as the ranks of later constructions. This construction served as the base for Kupferman and Vardi's rank-based construction in 1997 [15]. The worst-case state growth of Klarlund's construction is $2^{O(n \log n)}$

#### Kupferman and Vardi (1997/2001)

This construction has been published for the first time in 1997 [15] and later again in 2001 [16]. Both publications are entitled "Weak Alternating Automata Are Not That Weak". Kupferman and Vardi's construction basically pick up Klarlund's construction and provide a different description for it [16]. Actually, they even provide two descriptions for the same construction, one based on weak alternating automata (WAA), and one truly rank-based.

The first description includes several conversions. It starts by interpreting the input NBW as a universal co-Büchi automaton (UCW) that accepts the complement language. This UCW is then converted to a WAA, which is in turn converted to an NBW. The second description is the typical rank-based construction that we described in the introduction to this section. Kupferman and Vardi state that the results produced by their two versions and of Klarlund's construction are identical, however, their descriptions make the construction easier to understand and implement. [16].

**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 $(0.96n)^n$.

Tight rankings: description in [5] [50]

**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.

### 2.3.4 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 $\omega$-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 **??**. 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.

# 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 | $\approx$ 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 | $\approx$ 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 | $\approx$ 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 $\omega$-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 $\omega$-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. $\omega$-Automata. In J.-E. Pin, ed., *Handbook of Automata Theory*. European Mathematical Society. To appear, 2015.

[54] Q. Yan. Lower Bounds for Complementation of $\omega$-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.