Empirical Performance Investigation of a Büchi Complementation Construction

Daniel Weibel

June 13, 2015

Abstract

This will be the abstract.

Acknowledgements

First of all, I would like to thank my supervisors Prof. Dr. Ulrich Ultes-Nitsche and Joel Allred to bring me to the topic of this thesis in the first place, and for the many enlightening discussions and explanations. A very special thank goes to Ming-Hsien Tsai, the main author of the GOAL tool, from the National Taiwan University. Without his very prompt and patient responses to my many questions regarding implementation details of GOAL, the plugin would not have been possible in its current form. I also would like to thank Michael Rolli and Nico Färber from the Grid Admin Team of the University of Bern for their very friendly and helpful responses to my questions about the Linux cluster where the experiments of this thesis have been executed.

Contents

Chapter 1

Performance Investigation of the Fribourg Construction

Contents

In this chapter we come to the core of this thesis, namely to test how the Fribourg construction performs on real test automata. We are interested in two things. First, how the different versions of the Fribourg construction compare to each other. That is, with which optimisations the Fribourg construction is most efficient. Second, we compare the Fribourg construction to other complementation constructions. We can refer ot the first type of investigations as the *internal* tests, and to the second one as the *external* tests.

To do these investigations, we implemented the Fribourg construction as a plugin for an ω-automata manipulation tool called GOAL. GOAL already contains implementations of the most important Büchi complementation constructions. That is, with our plugin, the Fribourg construction lives next to these other constructions in the GOAL tool, and can be easily compared to them. With the plugin in place, we then performed the actual internal and external tests. To do so, we defined a set of test data consisting of totally 22.000 automata. The performance investigation consists then in basically complementing each of these automata with the different constructions, and comparing the results. Our main performance metric is the number of generated states for the output automata. The computations, which due to the complexity of Büchi complementation are quite heavy, were executed on a high-performance computing cluster at the University of Bern, Switzerland.

In this chapter, we are going to describe each of these points, including our concrete experiment setup. The results of the tests are presented and discussed in Section **??**.

1.1 Implementation

1.1.1 GOAL

GOAL stands for Graphical Tool for Omega-Automata and Logics and has been developed at the National University of Taiwan since 2007 [10, 11]. The tool is based on the three pillars, ω-automata, temporal logic formulas, and games. It allows to create instances of each of these types, and manipulate them in a multitude of ways. Relevant for our purposes are the ω-automata capabilities of GOAL.

With GOAL, one can create Büchi, Muller, Rabin, Streett, parity, generalised Büchi, and co-Büchi automata, either by manually defining them, or by having them randomly generated. It is then possible to perform a plethora of operations on these automata. The entirety of provided operations are too many to list, but they include containment testing, equivalence testing, minimisation, determinisation, conversions to other ω-automata types, product, intersection, and, of course, complementation.

All this is accessible by both, a graphical and a command line interface. The graphical interface is shown in Figure 1.1. Automata are displayed in the main editor window of the GUI. They can be freely edited, such as adding or removing states and transitions, and arranging the layout. There are also various layout algorithms for automatically laying out large automata. Most of the functionality provided by the graphical interface is also accessible via a command line mode. This makes it suitable for automating the execution of operations.

For storing automata, GOAL defines an own XML-based file format, called GOAL File Format, usually indicated by the file extension gff.

An important design concept of GOAL is modularity. GOAL uses the Java Plugin Framework (JPF) [1], a library for building modular and extensible Java applications. A JPF application defines so-called extension points for which extensions are provided. These extensions contain the actual functionality of the application. Extensions and extension points are bundled in plugins, the main building block of a JPF application. It is therefore possible to extend an existing JPF application by bundling a couple of new extensions for existing extensions points in a new plugin, and installing this plugin into the existing application. On the next start of the application, the new functionality will be included, all without requiring to recompile the existing application or to even have its source code.

GOAL provides a couple of extensions points, such as *Codec*, *Layout*, or *Complementation Construction*. An extension for *Codec*, for example, allows to add the handling of a new file format which GOAL can

[1] http://jpf.sourceforge.net/

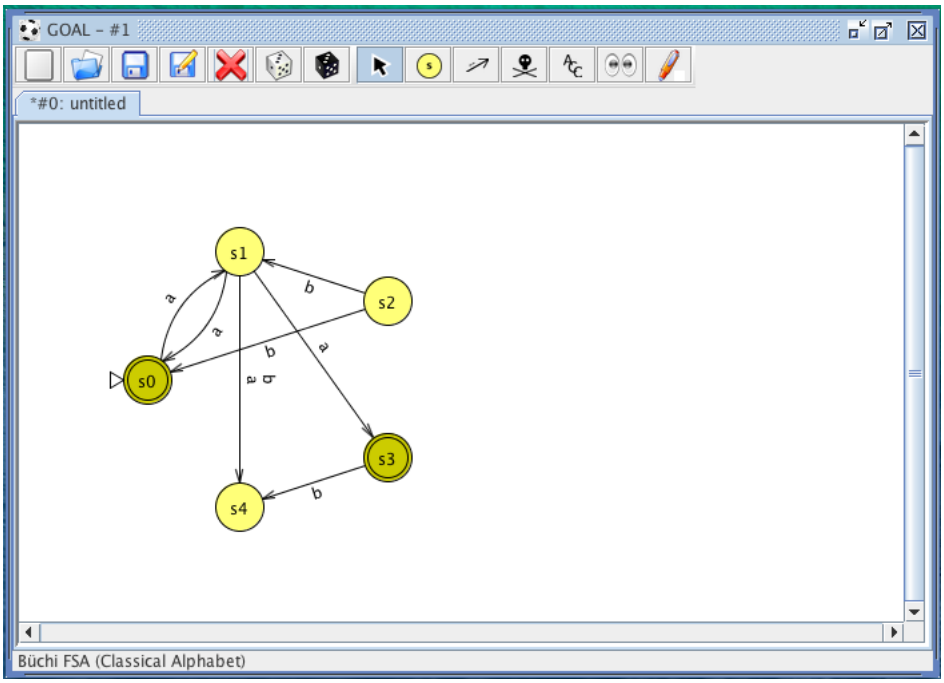


Figure 1.1: Graphical interface of GOAL.

Table 1.1: The complementation constructions implemented in GOAL (version 2014-11-17).

Name	Command line	Reference
Ramsey-based construction	ramsey	Sistla, Vardi, Wolper (1987) [8]
Safra's construction	safra	Safra (1988) [6]
Modified Safra's construction	modfiedsafra	Althoff (2006) [1]
Muller-Schupp construction	ms	Muller, Schupp (1995) [4]
Safra-Piterman construction	piterman	Piterman (2007) [5]
Via weak alternating parity automaton	wapa	Thomas (1999) [9]
Via weak alternating automaton	waa	Kupferman, Vardi (2001) [3]
Rank-based construction	rank	Schewe (2009) [7]
Slice-based construction (preliminary)	slice -p	Vardi, Wilke (2007) [12]
Slice-based construction	slice	Kähler, Wilke (2008) [2]

read from and write to. With an extension for *Layout* one can add a new layout algorithm for laying out automata in the graphical interface. And an extension to Complementation Construction allows to add a new complementation construction to GOAL. This is how we added the Fribourg construction to GOAL, as we will further explain in Section 1.1.2.

There are a couple of Büchi complementation constructions pre-implemented in GOAL. Table 1.1 summarises them, showing for each one its name on the graphical interface and in the command line mode, and the reference to the paper introducing it. As can be seen, the most important representants of all the four approaches (Ramsey-based, determinisation-based, rank-based, and slice-based, see Chapter **??**) are present. In addition to the listed constructions, GOAL also contains Kurshan's construction and classic complementation. These are for complementing DBW and NFA/DFA, respectively, and thus not relevant to us.

One of the constructions can be set as the default complementation construction. It is then possible to invoke this construction with the shortcut Ctrl-Alt-C. Furthermore, the default complementation constructions will be used for the containment and equivalence operations on Büchi automata, as they include complementation.

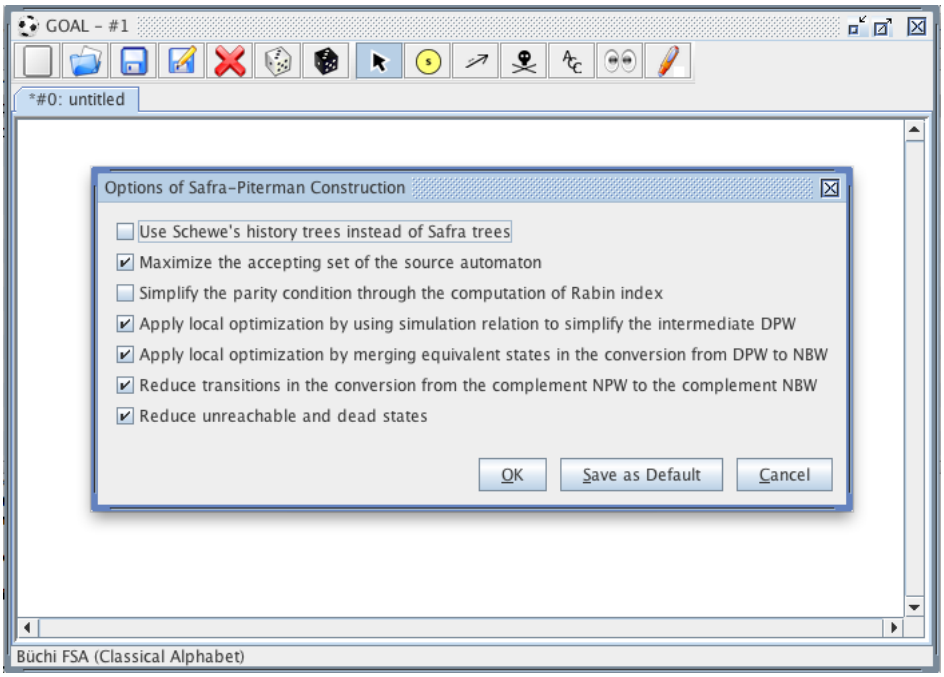


Figure 1.2: Complementation constructions in GOAL can have a set user-selectable options. Here the options of the Safra-Piterman construction.

Complementation constructions in GOAL can define a set of options that can be set by the user. In the graphical interface this is done at the start of the operations via a dialog window, in the command line mode the options are specified as command line arguments. Figure 1.2 shows the options dialog of the Safra-Piterman construction. Complementation options allow to play with different configurations and variants of a construction, and we will make use of them for including the optimisations presented in Chapter **??** to our implementation of the Fribourg construction.

For most complementation constructions (all listed in Table 1.1 except the Ramsey-based construction) there is also a version for step-by-step execution. In this case, the constructions define so-called steps and stages, through which the user can iterate independently. This is a great way for understanding how a complementation construction works, and for investigating specific cases in order to potentially further improve the construction.

1.1.2 Implementation of the Construction

We implemented the Fribourg construction, including its optimisations, in Java as a plugin for GOAL. This means that after installing out plugin to an existing GOAL installation[2], the Fribourg construction will be an integral part of GOAL and can be used in the same way as any other pre-existing complementation construction.

To keep the Fribourg construction flexible, we made use of options. The three optimisations described in Section **??** are presented to the user as selectable options. Additionally, we included several further options. Table 1.2 lists them all. For convenience, we use for each options a short code name, which is also used as the option name in the command line mode.

The first three items in Table 1.2, m1, m2, and r2c, correspond to the optimisations M1, M2, and R2C, described in Section **??**. As the M2 optimisation requires M1, our implementation makes sure that the m2 option can only be selected if also the m1 option is selected. The `c` option, for making the input automaton complete before starting the actual construction, is intended to be used with the `r2c` option. In this way, the R2C optimisation can be forced to apply. This idea results from previous work that investigated whether making the input automaton complete plus the application of the R2C optimisation

[2] As the plugin interfaces of GOAL have recently changed, the can be used only for GOAL versions 2014-11-17 and newer.

Table 1.2: The options for the Fribourg construction.

Code	Description
m1	Component merging optimisation
m2	Single 2-coloured component optimisation
r2c	Deleting states with rightmost colour 2, if automaton is complete
c	Make input automaton complete
macc	Maximise accepting states of input automaton
r	Remove unreachable and dead states from output automaton
rr	Remove unreachable and dead states from input automaton
b	Use the "bracket notation" for state labels

brings an improvement over the bare Fribourg constructoin [**?**]. The result was negative, that is, the construction performs worse with this variant on practical cases. Also note that using the c option alone, very likely decreases the performance of the construction, because the automaton is made bigger if it is not complete.

The macc and r options are common among the other complementation constructions in GOAL. The first one, macc, maximises the accepting set of the input automaton. That means, it makes as many states accepting as possible without changing the automaton's language. This should help to make the complement automaton smaller. The r options prunes unreachable and dead states from the complement automaton. Unreachable states are states that cannot be reached from the initial states, and dead states are states from where no accepting state can be reached. Clearly, all the runs containing an unreachable or dead state are not accepting, and thus these states can be removed from the automaton without changing its language. The complement automaton can in this way be made smaller. The rr option in turn removes the unreachable and dead states from the *input* automaton. That is, it makes the input automaton smaller, before the actual construction starts, what theoretically results in smaller complement automaton.

Finally, the b option affects just the display of the state labels of the complement automaton. It uses an alternative notation which uses different kinds of brackets, instead of the explicit colour number, to indicate the colours of sets. In particular, 2-coloured sets are indicated by square brackets, 1-coloured sets by round parenthesis, and 0-coloured sets by curly braces. Sets of states of the upper part of the automaton are enclosed by circumflexes. This notation, although being very informal, has proven to be very convenient during the development of the construction.

When we developed the plugin, we aimed for a complete as possible integration with GOAL. We integrated the Friboug construction in the graphical, as well as in the command line interface. We added a step-by-step execution of the construction in the graphical interface. We provided that customised option configurations can be persistently saved, and reset to the defaults at any time. We also integrated the Fribourg construction in the GOAL preferences menu so that it can be selected as the default complementation construction. In this way, it can be invoked with a key-shortcut and it will also be used for the containment and equivalence operations. Our goal is that once the plugin is installed, the Fribourg construction is as seamlessly integrated in GOAL as all the other pre-existing complementation construction.

The complete integration allows us to publish the plugin so that it can be used by other GOAL users. At the time of this writing, the plugin is accessible at http://goal.s3.amazonaws.com/Fribourg.tar.gz and also over the GOAL website[3]. The installation is done by simply extracting the archive file and copying the contained folder to the plugins/ folder in the GOAL system tree. No compilation is necessary. The same plugin and the same installation procedure works FOR Linux, Mac OS X, Microsoft Windows, and other operating systems that run GOAL.

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

[3] http://goal.im.ntu.edu.tw/

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.

1.1.3 Verification of the Implementation

See UBELIX/jobs/2014-11-25

Can we do a complement-equivalence test with all the 11,000 automata of size 15 in the test set?

Of course it is needed to test whether our implementation produces correct results. That is, are the output automata really the complements of the input automata? We chose doing so with an empirical approach, taking one of the pre-existing complementation constructions in GOAL as the "ground truth". We can then perform what we call complementation-equivalence tests. We take a random Büchi automaton and complement it with the ground-truth construction. We then complement the same automaton with our implementation of the Fribourg construction, and check whether the two complement automata are equivalent. Provided that the ground-truth construction is correct, we can show in this way that our construction is correct for this specific case.

We performed complementation-equivalence tests for the Fribourg construction with different option combinations. In particular, we tested the configurations m1, m1+m2, c+r2c, macc, r, rr, and the construction without any options. For each configuration we tested 1000 random automata of size 4 and with an alphabet of size 2 to 4. As the ground-truth construction we chose the Safra-Piterman construction. In all cases the complement of the Fribourg construction was equivalent to the complement of the Safra-Piterman construction.

Doubtlessly, it would be desirable to test more, and especially bigger and more diverse automata. However, by doing so one would quickly face practical problems due to long complementation times with bigger automata and larger alphabets, and high memory usage. For our current purpose, however, the tests we did are enough for us to be confident that our implementation is correct.

1.2 Test Data

The core of this thesis is to empirically investigate the performance of the Fribourg construction. To this end, we "feed" concrete automata to the Fribourg construction and analyse the results. This set of automata is our3 test data and is of crucial importance for the validity of the performance investigation. Test data should not be biased in favour or unfavour of one of the tested algorithms. Ideally, publicly available, or even standardised, test data is used, that has been previously used in other experiments. Furthermore, test data should cover interesting cases that reveal important aspects about the algorithm.

For our investigation we chose two test sets that are publicly known or available and cover very different scenarios. While one of them consists of 11,000 automata and tries to cover a broad range of "everyday" complementation problems, the other one consists of just 4 automata, that are however very special. We call the former one the GOAL test set and the latter on the Michel test set, and we are going to describe them in the following two sections.

1.2.1 GOAL Test Set

Introduction

The GOAL test set has been created by Tsai et al. for the experiments described in the the paper *State of Büchi Complementation* [**?**]. In this paper, the authors carry out empirical performance investigations of Büchi complementation constructions that are very similar to our own investigations. The experiments were run with the GOAL tool, hence the name GOAL test set. The idea leading the design of this test set was to generate a large set of complementation problems with various difficulty levels.

The test set consists of 11,000 automata of size 15, and with an alphabet size of 2. Within the given constraints, the 11,000 automata were generated at random. The test set is furthermore structured into

110 classes that are combinations of two parameters, the so called transition density and the acceptance density. Simply speaking, the transition density determines the number of transitions in an automaton, and the acceptance density determines the number of accepting states.

More precisely, the transition density t is defined as follows. Let n be the number of states of automaton A, and t its transition density. Then A contains tn transitions for each symbol of the alphabet[4]. That is, if one of our test set automata with 15 states and the alphabet $0, 1$ has a transition density of 2, then it contains exactly 30 transtions for symbol 0 and 30 transitions for symbol 1. Consequently, each state has on average two outgoing and two incoming transitions for each symbol. Therefore, the transition density can also be seen as the average number of outgoing and incoming transitons per alphabet symbol that a state has.

The acceptance density a is defined as the ratio of accepting states to non-accepting states in the automaton. That is, if automaton A has n states and an acceptance density of a, then it contains an accepting states[5]. The acceptance density is thus bound to a number between 0 and 1. For example, if an automaton of our test set has an acceptance density of 0.1, then it has 2 accepting states, and if an automaton has an accepting density of 1, then all of its states are accepting states.

In the GOAL test set there are 11 transiton densities and 10 acceptance densities. The transition densities range from 1 to 3 in steps of 0.2, and the acceptance densities range from 0.1 to 1 in steps of 0.1. Thus, we have

$$t' = \{1.0,\ 1.2,\ 1.4,\ 1.6,\ 1.8,\ 2.0,\ 2.2,\ 2.4,\ 2.6,\ 2.8,\ 3.0\}$$
$$a' = \{0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9,\ 1.0\}$$

The 110 classes result from the Cartesian product of t' and a'. Each class consequently contains 100 automata. In this thesis, we will often present these classes as a 11×10 matrix where the rows represent the transition densities t', and the columns represent the acceptance densities a'.

There is a second GOAL test set that is similar to the one that we just described, except that the automata sizes are 20 instead of 15. Tsai et al. [**?**] did their experiments with both, the size-15 (as the primary), and the size-20 (as the scondary) test set. For our own experiments we will however not use the size-20 test set. First, we think that repeating the experiments with the size-20 test set would add only a limited amount of insight. It might be more beneficial to do further tests with a more diverse type of test data, like for example the Michel test set that we describe in the next section. Furthermore, the bigger automata sizes in the size-20 test set are likely to cause practical problems in terms of computing and time resources (see Section 1.3.4). We decided therefore to restrict ourselves to the size-15 test set. When we refer to *the* GOAL test set throughout the rest of this thesis, we specifically mean the size-15 test set.

The GOAL test sets (both, size-15 and size-20) are publicly available on the GOAL website via the following link: http://goal.im.ntu.edu.tw/wiki/lib/exe/fetch.php?media=goal:ciaa2010_automata.tar.gz.

Analysis

We analysed some properties of the GOAL test set that might be of importance for the interpretation of the results of our Fribourg construction. In particular, we tested how many automata of the GOAL test set are *complete*, *universal*, or *empty*.

An automaton is complete if every state has at least one outgoing transition for every symbol of the alphabet. Universality means that an automaton accepts *every* word that can be generated from its alphabet. Emptiness is the contrary of universality and means that an automaton does not accept *any* word (except the empty word ϵ).

To know how many automata of the test set are complete is relevant because the R2C optimisation of the Fribourg construction only works on complete input automata. Universality and emptiness are interesting because their respective complement sizes have a lower bound of 1. The complement of a universal

[4] In the case tn is not an integer, it is rounded up to the next integer.
[5] Again, if an is not an integer, it is rounded up to the next integer.

automaton is an empty automaton, which can be represented by a single state, and the complement of an empty automaton is a universal automaton which can be represented by a single state as well. In this way we can get an idea about the efficiency of the Fribourg construction in terms of produced unreachable and dead states, an aspect that we will investigate with the R option (see Section 1.3.

GOAL provides a command for testing emptiness of an automaton. However, it does not provide commands for testing completeness and universality. We therefore implemented these commands on our own as a separate GOAL plugin (ch.unifr.goal.util, also available as described in Appendix A). The tests were executed in the execution environment as described in Section 1.3.4. The overall results are as follows.

- 990 of the 11,000 automata are complete (9%)
- 6,796 of teh 11,000 automata are universal (61.8%)
- 63 of the 11,000 automata are empty (0.6%)

We furthermore tested how these numbers are distributed over the 110 transition/acceptance density classes.

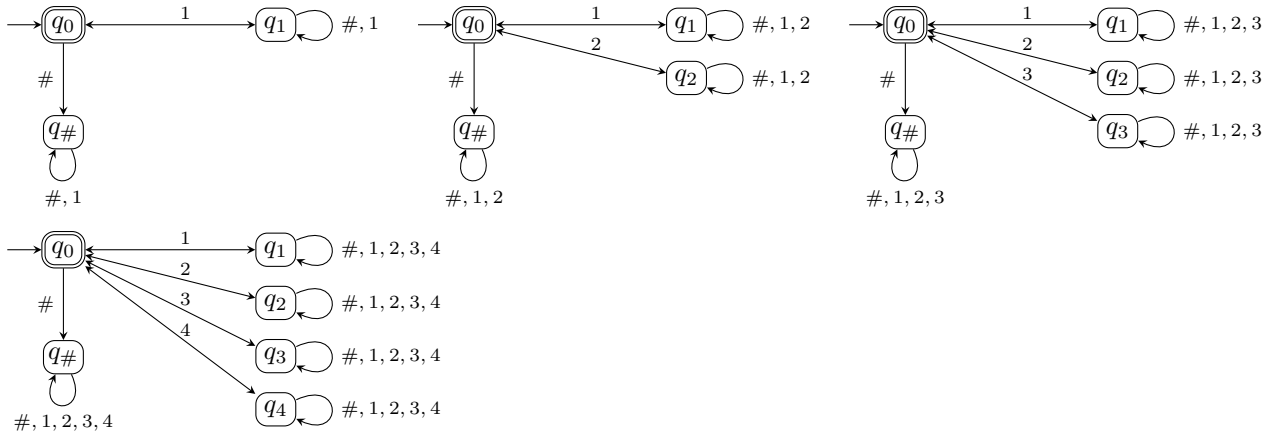
We furthermore analysed how these two properties are distributed over the 110 classes of transition/acceptance density combinations. The results follow below in table form and as a three-dimensional visualisation.

We also tested the automata for emptiness, that is, the contrary of universality. From the 11,000 automata, only 63 are empty, which is a percentage of 0.57%. As one could expect, most of them are concentrated in the classes with low transition and acceptance densities. For example, 44 of these empty automata are in the classes with a transition density of 1.0.

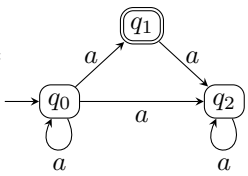
1.2.2 Michel Test Set

Besides the test set of the GOAL automata, we did all the tests also on the first four Michel automata:

- Michel N1: 3 states, 7 transitions, 1 accepting state
- Michel N2: 4 states, 14 transitions, 1 accepting state
- Michel N3: 5 states, 23 transitions, 1 accepting state
- Michel N4: 6 states, 34 transitions, 1 accepting state



$A =$



These are the Michel automata that can be processes with our implementation and on our execution environment. Complementing Michel automata N5 and above would already by far exceed our available computing and time resources.

1.3 Experimental Setup

1.3.1 Internal Tests

In the internal tests our aim is to compare different versions of the Fribourg construction. A specific version of the Fribourg construction is composed of a combination of options. Options can be the

As presented in Section **??**, there are three optimisations to the Fribourg construction:

- R2C: if the input automaton is complete, remove all states whose rightmost colour is 2
- M1: merge certain adjacent sets within a state
- M2: reduce 2-coloured sets (requires M1)

Furthermore, our GOAL plugin includes, among others, the following options:

- C: make the input automaton complete (by adding a sink state)
- R: remove unreachable and dead states from the output automaton

The versions of the Fribourg construction that we chose for our internal tests consist of combinations of these five options. According to the nature of the two test sets (the GOAL test set and the Michel automata), we chose two different sets of versions for the two test sets. We are now first going to describe the setup for the GOAL test set and then the one for the Michel test set.

GOAL Test Set

For the tests on the GOAL test set, we chose the following eight versions of the Fribourg construction:

1. Fribourg
2. Fribourg+R2C
3. Fribourg+R2C+C
4. Fribourg+M1
5. Fribourg+M1+M2
6. Fribourg+M1+R2C
7. Fribourg+M1+R2C+C
8. Fribourg+R

The first version is the plain Fribourg construction without any optimisations or options. The next two versions are devoted to investigate the R2C optimisation. Version 2 applies the optimsation only to the automata which happen to be complete (and as we have seen in Section **??** these are 9%). Version 3, on the other hand, makes all input automata preliminarily complete by adding a sink state, so that the R2C optimisation can be applied to *all* automata. The question here is, does it pay off to increase the size of the automata by one (for adding the sink state) but then being able to apply R2C, or is it better to not add the extra state but then not applying R2C neither?

Similar investigations about the R2C optimisation of the Fribourg construction have been made by Göttel [**?**]. In particular, he compared Version 1 in our above listing with Version 3. His results were that the mean complement sizes of Version 3 are higher than in Version 1. He evaluated, however, only the mean values, but looking closely at the results suggests that the median values could be in favour of Version 3. Therefore, we decided to reinvestigate this question. Indeed, in our own results the median complement sizes of Version 3 are considerably lower than the ones of Version 1 and Version 2. We will further elaborate on this point in Chapter 2.

Versions 4 and 5 in our above listing are for investigating the M1 and M2 optimisations. As M2 requires M1, there are only these two possible combinations. Versions 6 and 7 then enhance the "better" one of Version 4 and 5 with R2C and its alternative R2C+C. As we will see in Chapter 2, the better one of Version 4 and 5 in terms of median complement sizes is Version 4. That is, the application of M2 results in a decline, rather than a gain, in performance compared to the application of M1 alone. We have to note at this point that such results are always specific to the used the test set, and not universally valid.

With a different test set, Version 5 might indeed be better than Version 4. As we will see in the next section, this is the case for our alternative test set consisting of the first four Michel automata.

Version 8, finally, is again the plain Fribourg construction, but this time the output automata are reduced by removing their unreachable and dead states. Comparing the results of Version 8 with Version 1 gives an idea of how many unreachable and dead states the Fribourg construction produces. This is inspired by the paper of the GOAL authors [**?**] in which the number of unreachable and dead states is one of the main metrics for assessing the performance of a construction.

Michel Test Set

The versions we tested for the Michel test set are the following:

1. Fribourg
2. Fribourg+R2C
3. Fribourg+M1
4. Fribourg+M1+M2
5. Fribourg+M1+M2+R2C
6. Fribourg+R

The rationale for choosing these versions is basically the same as for the GOAL test set with the following differences. First, Michel automata are complete, thus it is not necessary to apply the C option. Rather, R2C will automatically apply to all automata. Second, the aim of Version 5 is again to enhance the better one of Versions 3 and 4 with R2C. However, contrarily to the GOAL test set Fribourg+M1+M2 is better than Fribourg+M1 for the Michel automata. That is why Version 5 is Fribourg+M1+M2+R2C.

1.3.2 External Tests

In the so called external tests we compare the best version of the Fribourg construction with different complementation constructions, again for both the GOAL test set and the Michel test set. The concrete constructions we compared for the external tests are the following.

1. Piterman+EQ+RO
2. Slice+P+RO+MADJ+EG
3. Rank+TR+RO
4. Fribourg+M1+R2C (GOAL test set) and Fribourg+M1+M2+R2C (Michel test set)

For the alternative constructions, we chose the Piterman, Slice, and Rank construction. These constructions are representative for three of the four main complementation approaches, determinization-based, slice-based, and rank-based. The fourth complementation approach is Ramsey-based and there is an implementation of the Ramsey construction in GOAL. However, in preliminary tests, we realised that this construction is not performant enough to be used on our test sets within our time and memory constraints. The authors of GOAL came to a similar conclusion when they made similar experiments on the GOAL test set [**?**]. In their case, the Ramsey construction could not complete any of the 11,000 automata within the set time and memory constraints, and they went on to do the result analysis without the Ramysey construction.

The same applies to all the other constructions that are implemented in GOAL. We did preliminary tests with all of them and saw that only the three mentioned constructions, Piterman, Slice, and Rank, can be resonably used on the GOAL test set[6].

The three chosen constructions also have optimisations in their implementations in GOAL. To have a fair comparison to the best version of the Fribourg construction, which also uses optimisations, we activated the optimisation of these constructions as well. In particular, we chose those optimisations which are set as default in the GOAL GUI for each construction.

[6] The Safra construction would also have been possible, but the Safra construction is similar to the Piterman construction.

We use Fribourg+M1+R2C for the GOAL test set and Fribourg+M1+M2+R2C for the Michel test set. This is because these two versions are the most performant ones for the respective test sets.

1.3.3 Time and Memory Limits

We defined a time limit of 600 seconds CPU time and a memory limit of 1 GB per complementation task in the GOAL test set. That means, if the complementation of a single automaton is not finished after 600 seconds CPU time or uses more than 1 GB memory, the task is aborted and marked as a *timeout* or *memory excess*.

These limits correspond to the ones used by the experiments of the GOAL authors [**?**]. However, from their paper it is not clear if their time limit is in CPU time or wallclock time. But since they used different computing nodes, our results regarding the number of timeouts will anyway differ from theirs.

The reason for these limits is simply the restricted amount of time and memory resources that we have available for the experiments. In an ideal world, we would let every complementation task run to its end, no matter how long it takes and how much memory it uses. This would give a perfectly unbiased picture of the results. By setting time and memory limits, we basically exclude the most difficult automata from the experiment. However, as mentioned, the practical reasons of limited time and computing power force as to make this compromise.

The timeout and memory limit of 1 GB apply just to the automata of the GOAL test set. For the Michel test set we did not set a timeout because we wanted each one of the four automata to finish. The longest complementation task of a Michel automaton consequently durated 109,810 seconds which is about 28 hours. We set a very high memory limit of 14 GB for the Michel test set to avoid memory excesses as we wanted each automaton to successfully complete. The number of 14 GB is determined by the physically available memory on the used computing nodes. All Michel automata successfully completed with this amount of memory.

The timeout was implemented by the means of the ulimit Bash builtin[7], which allows to set a maximum time after which running processes are killed. The memory limit was implemented by setting the maximum size of the Java heap, which can be done by the -Xmx option to the Java Virtual Machine (JVM). The heap is the main memory area of Java and the place where all the objects reside. Note that since our memory limit defines actually the size of the Java heap, the total amount of memory used by the process is higher than our limit, as Java has some other memory areas, for example for the JVM itself. However, this is a rather constant amount of memory and independent from the current automaton, so it does not disturb the relative comparisons of the results.

The presence of aborted complementation task require the consideration of the so called effective samples in the result analysis, as introduced in the experiment paper of the GOAL authors. The effective samples are those automata which have been successfully completed by *all* constructions that are to be compared to each other. Imagine two constructions A and B where A is successful complementing all the automata, whereas B has timeouts or memory excesses at 100 of the automata. If we would now take, for example, the median complement sizes of the two result sets without first extracting the effective samples, then B is likely be assessed as too good relative to A, because B's results do not include the 100 automata at which it failed, and which are thus likely to have large complement sizes with B. The same 100 automata would however be included in the results of A. Therefore, all the result analysis of the experiments with the GOAL test sets, that we present in Chapter 2, are based on the effective samples of the result sets.

1.3.4 Execution Environment

We executed the experiments on a high performance computing (HPC) computer cluster called UBELIX at the University of Bern[8]. This cluster consists of different types of Linux-driven HPC computing nodes, and is managed by Oracle Grid Engine[9] (formerly known as Sun Grid Engine) version 6.2. Oracle Grid

[7] http://linux.die.net/man/1/bash
[8] http://ubelix.unibe.ch
[9] http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html

Engine is a so called load scheduler that is responsible for automatically distributing computing tasks to computing nodes.

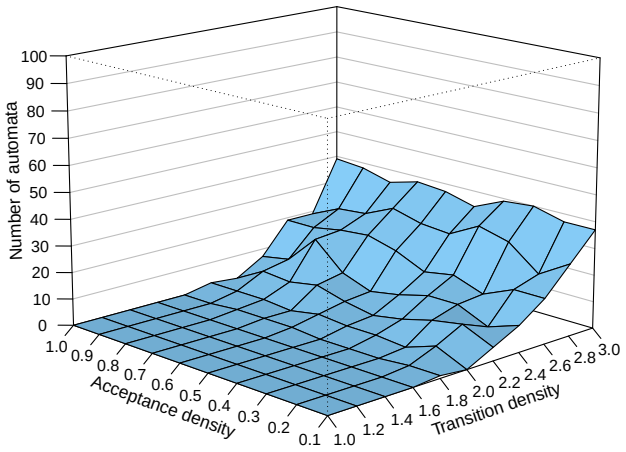
The basic workflow of working on the cluster is to prepare a so called job and specify the resources that the job requires for running (time, memory, number of CPU cores, and so on). Then the job can be submitted to the grid engine. The grid engine first puts incoming jobs in a queue and then automatically dispatches them to suitable computing nodes as soon as the required capacity is available.

A job in our case is the execution of a construction (or version of the Fribourg construction) over an entire test set. Thus, we were running 22 jobs in total. We arranged that all jobs were run on identical nodes with the following specifications:

- Processor: Intel Xeon E5-2665 2.40GHz (64 bit)
- CPU cores: 16
- Operating System: Red Hat Enterprise Linux 6.6

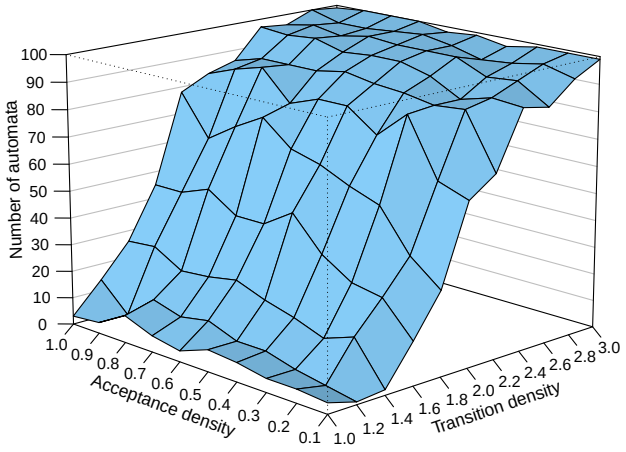
Since we use the execution time (CPU time) as a secondary metrics, next to the complement sizes, it is important that all complementation tasks are executed on the same type of hardware. GOAL is multithreaded and thus the jobs may use multiple CPU cores. The number of CPU cores a job may use is not restricted (up to the number of available cores on the node, in our case 16), but our observation is that the jobs use a rather small number of cores (2–3). Note that the measurement of the execution time is not affected by the number of cores a process uses, as the CPU times are measured separately on each core and then added together.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	0	0	0	0	0	0	0	0	0	0
1.2	0	0	0	0	0	0	0	0	0	0
1.4	0	0	0	0	0	0	0	0	0	0
1.6	0	0	0	0	0	0	0	0	0	0
1.8	1	1	0	0	0	1	1	1	0	0
2.0	0	5	1	2	2	3	1	2	2	2
2.2	5	10	8	5	3	5	8	6	7	1
2.4	10	6	11	11	8	6	10	20	9	7
2.6	17	17	12	16	14	19	22	21	19	19
2.8	27	20	29	32	26	27	30	25	24	19
3.0	37	37	40	39	34	37	38	35	38	39



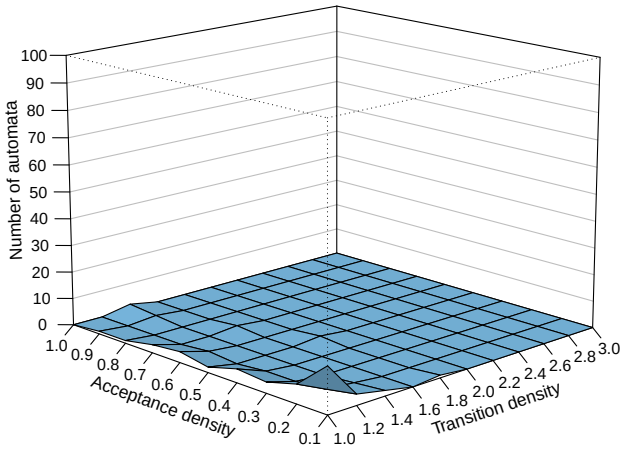
(a) Completeness

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	4	5	5	7	8	4	6	10	4	3
1.2	1	3	5	8	8	12	10	13	4	14
1.4	2	17	13	17	20	24	22	21	27	26
1.6	16	28	30	37	49	42	42	49	45	45
1.8	31	40	55	59	64	67	76	70	63	78
2.0	60	64	85	75	83	83	79	90	87	83
2.2	67	87	86	88	89	91	89	89	89	86
2.4	88	89	86	92	95	95	94	97	96	97
2.6	86	93	92	97	97	97	98	96	98	96
2.8	94	97	95	94	97	99	98	97	97	100
3.0	99	99	99	97	99	98	100	100	100	99



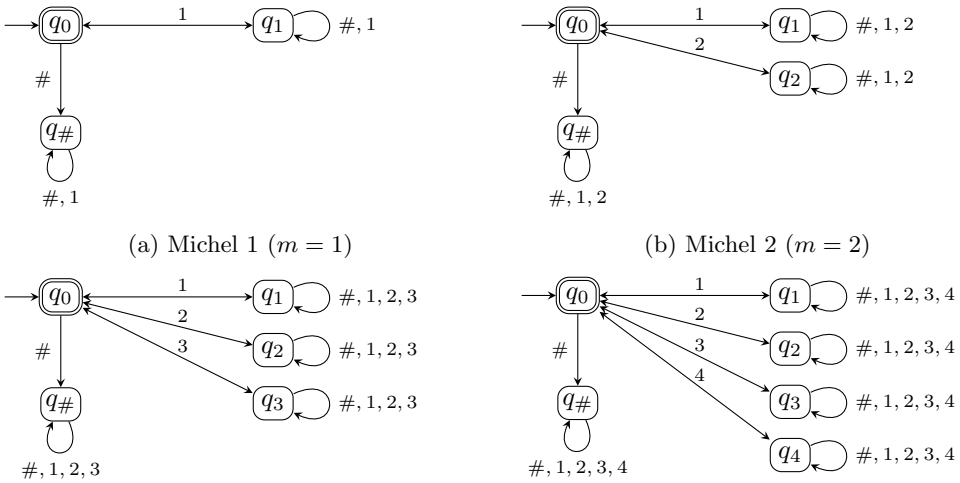
(b) Universality

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1.0	17	7	4	5	2	4	3	1	1	0
1.2	4	2	1	1	0	1	0	0	0	0
1.4	2	1	0	0	0	0	0	0	1	2
1.6	0	0	0	0	0	0	1	0	0	0
1.8	1	0	0	0	1	0	0	0	0	0
2.0	0	0	0	0	0	0	0	0	0	0
2.2	0	0	0	0	0	0	0	0	0	0
2.4	0	0	0	0	0	0	0	0	0	0
2.6	0	0	0	0	0	0	0	0	0	0
2.8	0	0	0	0	0	0	0	0	0	0
3.0	0	0	0	0	0	0	0	0	0	0



(c) Emptiness

Figure 1.3: Completeness and universality in the GOAL test set.



(a) Michel 1 ($m = 1$)

(b) Michel 2 ($m = 2$)

(c) Michel 3 ($m = 3$)

(d) Michel 4 ($m = 4$)

Figure 1.4: The Michel automata with $m = \{1, \ldots, 4\}$, an alphabet size of $m + 1$, and $m + 2$ states.

Chapter 2

Results and Discussion

Contents

2.1 Internal Tests

2.1.1 GOAL Test Set

First of all, let us see how many unsuccessful complementation tasks there were with the internal tests on the GOAL test set. Table 2.1 shows the number of timeouts and memory excesses for each of the eight tested versions of the Fribourg construction.

Construction	Timeouts	Memory excesses
Fribourg	48	0
Fribourg+R2C	30	0
Fribourg+R2C+C	54	0
Fribourg+M1	2	0
Fribourg+M1+M2	1	0
Fribourg+M1+R2C	1	0
Fribourg+M1+R2C+C	8	0
Fribourg+R	48	0

Table 2.1: Number of timeouts and memory excesses in the internal tests on the GOAL test set.

As we can see in the table, there were no memory excesses, that is, none of the 11,000 complementation tasks required more than 1 GB memory (actually, Java heap size). On the other hand, there is quite a number of timeouts. If we extract the effective samples from these result sets, we get a set of 10,939 automata. That means that these 10,939 automata have been successfully completed by all the versions of the Fribourg construction, whereas the remaining 61 automata (0.55%) provoked a timeout in at least one of the versions.

Our main analysis is now about the sizes of the complements of these 10,939 automata. With size we mean the number of states of an automaton. A stripchart as the one in Figure 2.1 is a good way to get a first glance of the complement sizes. Each horizontal strip contains 10,939 dots, each one corresponding to a produced complement automaton. The x-position of a dot indicates the size of the corresponding complement automaton.

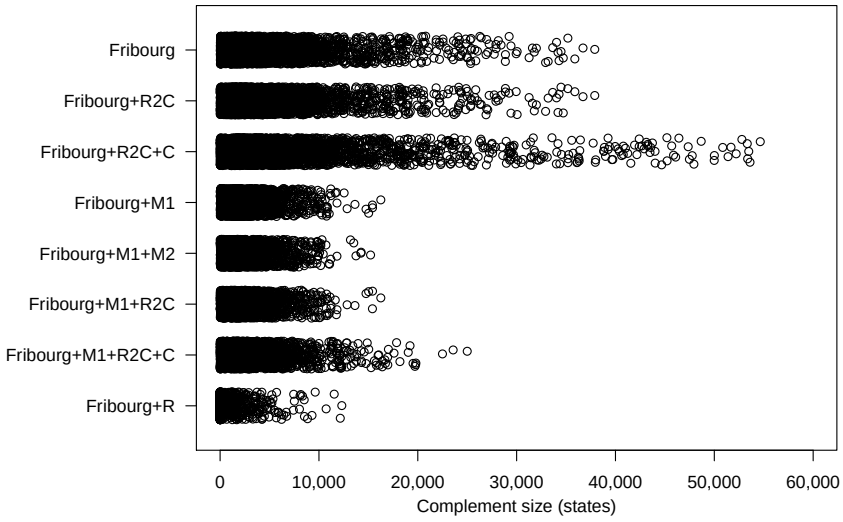


Figure 2.1: Stripchart with the complement sizes of the 10,939 effective samples of the GOAL test set.

The first thing to note is that the distribution of complement sizes is extremely right-skewed (also known as positive-skewed). The peak close to the left end of the x-axis and there is a long tail toward the right. This means that most of the complements are very small and the frequency decreases with increasing

complement size. Finally, there are some few complements which are very large. This distribution implies that the mean is generally higher than the median, because the mean is "dragged" to the right by the few very large complements.

Next, we can compare the distributions of the individual version to each other. While Fribourg and Fribourg+R2C have a similarly long tail, Fribourg+R2C+C has a considerably longer one. This clearly is the effect of increasing the size of 91% of the automata by adding a sink state in order to make them complete. Next, Fribourg+M1, Fribourg+M1+M2, and Fribourg+M1+R2C have similarly long tails that are significantly shorter than the ones of the previous versions. This indicates that the M1 optimisation is effective in reducing the size of otherwise large complements. Fribourg+M1+R2C+C, as expected, again increases the size of the tail due to the C option. Finally, Fribourg+R has a very short tail and an even higher concentration of very small complements. This is because the complements are pruned by removing their unreachable and dead states, and thus at least the complements of the 61.8% universal input automata are reduced to empty automata of size 1.

Having a first impression of the distribution of complement sizes, we can look at some statistics characterising this distribution. Table 2.2 shows the mean complement size along with the classic five-number summary consisting of minimum value, 25th percentile, median, 75th percentile and maximum value for each version of the Fribourg construction. As expected, the mean is always higher than the median. Generally, the median is a more robust metrics than the mean, because it is not affected by the value of outliers. This applies specifically to our distribution where some few very large complements might significantly increase the man whereas they leave the median unaffected. Therefore, the median will be our main metric, and the subsequent analyses will be based on the median.

Construction	Mean	Min.	P25	Median	P75	Max.
Fribourg	2,004.6	2	222.0	761.0	2,175.0	37,904
Fribourg+R2C	1,955.9	2	180.0	689.0	2,127.5	37,904
Fribourg+R2C+C	2,424.6	2	85.0	451.0	2,329.0	54,648
Fribourg+M1	963.2	2	177.0	482.0	1,138.0	16,260
Fribourg+M1+M2	958.0	2	181.0	496.0	1,156.5	15,223
Fribourg+M1+R2C	937.7	2	152.0	447.0	1,118.0	16,260
Fribourg+M1+R2C+C	1,062.6	2	83.0	331.0	1,208.5	25,002
Fribourg+R	136.3	1	1.0	1.0	21.0	12,312

Table 2.2: Statistics of the complement sizes of the 10,939 effective samples of the GOAL test set.

Going through the median values in Table 2.2 we encounter some surprises. To begin with, as expected, there is a decrease from Fribourg (761) to Fribourg+R2C (689). Then, however, there is a significant drop to 451 with Fribourg+R2C+C. This is a surprise insofar as by looking at Figure 2.1, Fribourg+R2C+C seems to have the worst performance at a first glance. Indeed it also has the highest mean, which is due to the group of extremely large complements. The median, however, is very low, even lower than the one of Fribourg+M1 with its significantly shorter tail in Figure 2.1. Also the 25th percentile of Fribourg+R2C+C is with 85 one of the lowest. Going to the other side of the median, however, the 75th percentile (2,329) is the largest of all versions. A possible characterisation of this phenomenon is that the C option (together with R2C) makes small complements smaller, and large complements larger. The diminishment of small complements is far-reaching enough that the median is affected by it and decreased significantly.

The next thing we see in Table 2.2 is that the median of Fribourg+M1 (482) is slightly lower than the median of Fribourg+M1+M2 (496). The same applies to the 25th and 75th percentile. This backs up our statement from Section 1.3.1 that Fribourg+M1 performs better on the GOAL teset set than Fribourg+M1+M2. The difference is rather small (the median increase is by 2.9%), but it is enough to consider Fribourg+M1 as the better of the two versions, and to combine it therefore with R2C and R2C+C.

Fribourg+M1+R2C brings down the median from 482 to 447, with respect to Fribourg+M1. Also the 25th and 75th percentile are decreased. Adding the C option to Fribourg+M1+R2C, again causes the median to drop dramatically, from 447 to 331. The 25th percentile decreases from 152 to 83. The 75th

percentile however increases from 1,118 to 1,208.5. Here we have again the same picture of the effecto of adding the C option that we had before. Namely that small complements are made smaller, and large complements are made larger.

Finally, the last row in Table 2.2 with Fribourg+R shows the extent of unreachable and dead states that the Fribourg construction produces. The median is 1, and a further analysis reveals that the single-state complements go up to the 61st percentile. That is, 61% of the complements have a size of 1. This is likely to correspond to the 61.8% of universal automata in the GOAL test set whose complements can be reduced to empty automata with a single state.

Up to now, we only looked at statistics aggregated over the entire test set. But as we know from Section 1.2.1, the 11,000 automata of the GOAL test set are divided into 110 classes which consist of combinations of 11 transition densities and 10 acceptance densities. The transition densities range from 1 to 3 in steps of 0.2, and the acceptance densities range from 0.1 to 1 in steps of 0.1. In each class there are 100 automata. It is assumable that the median complement sizes are different for different classes. In the next step of the analysis, we want to analyse exactly this point. How do the median complement sizes of the different classes compare to each other, and can we identify a pattern of the classes that result in large and small complements?

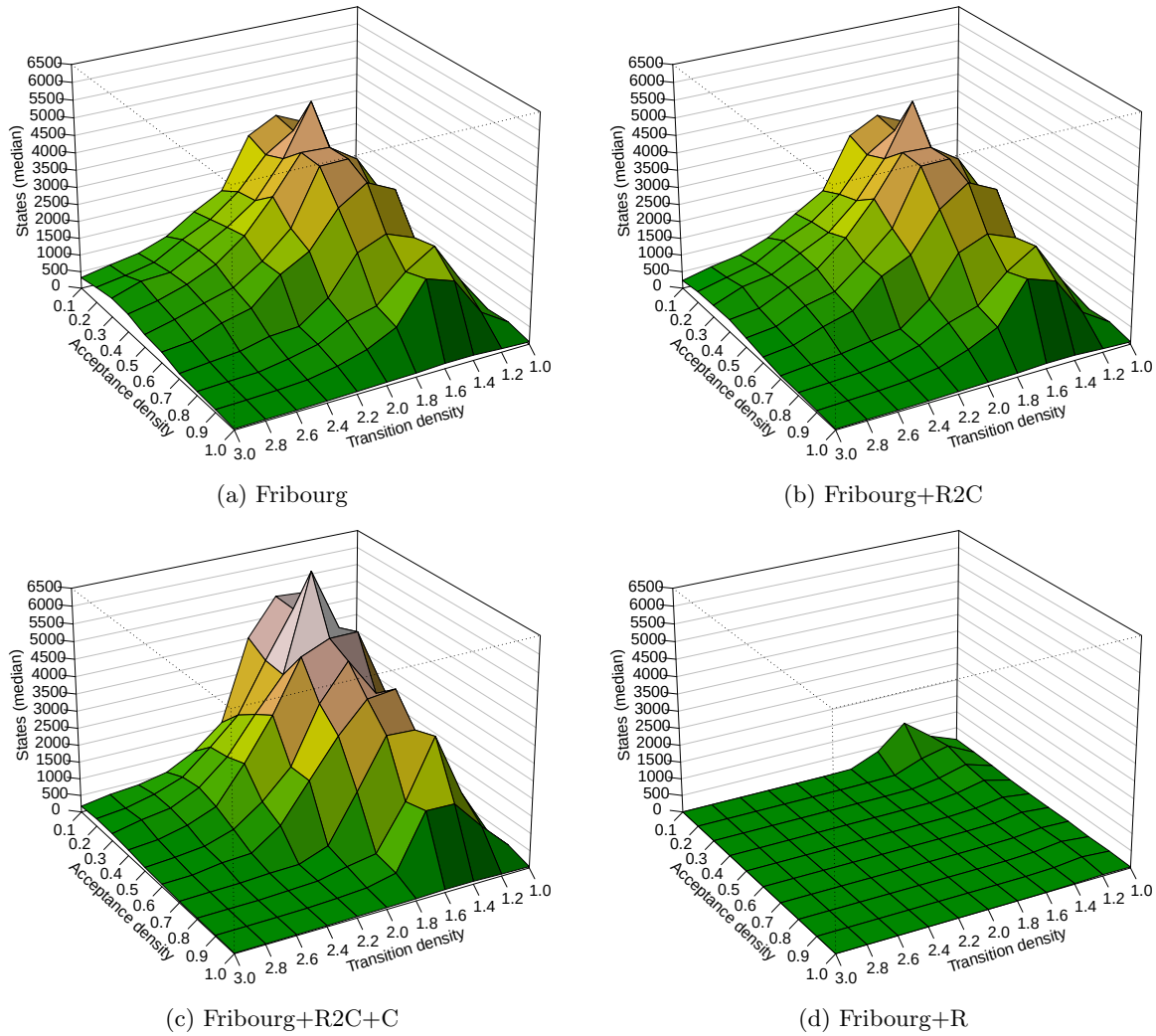
This type of analysis naturally results in three-dimensional data. The classes themselves have two dimensions (the transition densities and the acceptance densities), and each of them has a third dimension consisting of the median complement size of this class. One way to present such data is by, for example, a 11x10 matrix with the rows being the transition densities, the columns the acceptance densities, and the cells the median values. Another way is by so called perspective plots. We used both of these ways in Section 1.2.1 when we analysed the number of complete and universal automata in each class of the GOAL test set. In the present analysis, we will use perspective plots and omit the matrices for space reasons. However, we present the corresponding matrices in Appendix B.2. In this way, the interested reader may consult the exact values of the median complement sizes for each class. In the same way, we restrict ourselves to the median complement sizes as the third dimension. The same analysis could also be done, for example, for the mean, the 25th percentile, or the 75th percentile of complement sizes. However, as mentioned, we consider the median to be the most meaningful statistics, and thus exclusively focus on it for the matter of conciseness.

Figures **??** and **??** show the perspective plots of the median complement sizes for the 110 classes of the GOAL test set. Each crossing of two lines on the xy-plane represents a class, and the z-value (height) of thie crossing indicates the median complement size of this class. As a help for the orientation, if one imagines the corresponding matrices, as they are presented in Appendix B.2, then in the perspective plots we are looking at these matrices from the bottom-right corner. That is, the class that is in the top-left corner of a matrix (transiton density 1.0 and acceptance density 0.1) is the one that is the farthest away from the viewer in the corresponding perspective plot. This orientation holds for all the remaining perspective plots in this thesis. The colours of the squares in the perspective plot (called facets) are a function of the z-values of their four corners and are selected to draw an analogy with topographical terrain maps.

The first thing we notice when looking at the plots in Figure 2.2 and **??** is that there are indeed large differences in the median complement sizes across the transition/acceptance density classes. Figuratively speaking, there is an oblong mountain (or hill) in the northern part of the area in east-west orientation, whose ridge increases in height from east to west and stays high until the western edge of the area. The mountain is located roughly in the area of transition densities between 1.2 and 2.2 and acceptance densities from 0.1 up to 0.9. This means that the automata of these classes are apparently harder to complement than the automata of other classes, as they result more frequently in larger complements.

We will further elaborate on the reasons of these different complement sizes across classes, and also try to characterise easy, medium, and hard automata for the Fribourg construction, at the end of this subsection. For now, we focus on the relative differences of median complement sizes between the different versions of the Fribourg construction.

Looking at Figure 2.2, the perspective plots for Fribourg and Fribourg+R2C are rather similar. The top of the mountain ridge is at between 3,500 and 4,000 states with a single peak of around 4900 states in the class with transition density 1.6 and acceptance density 0.3. From Table 2.2 we can learn that the



(a) Fribourg

(b) Fribourg+R2C

(c) Fribourg+R2C+C

(d) Fribourg+R

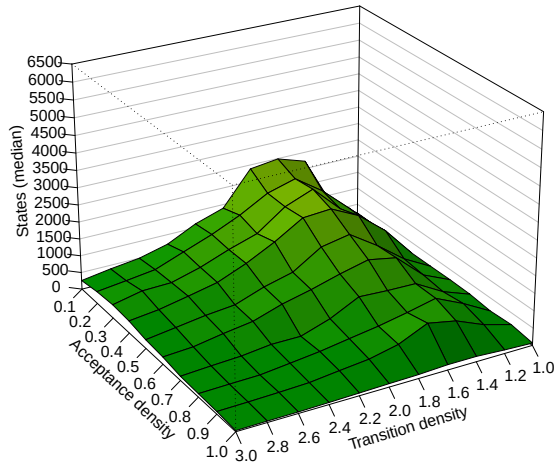
Figure 2.2: Median complement sizes of the 110 transition density/acceptance density classes of the 10,939 effective samples of the GOAL test set.

overall median complement size is 761 for Fribourg and 689 for Fribourg+R2C. These low values might surprise at first as the mountain, which is much higher, seems to dominate. However, by taking a closer look, it becomes apparent that around half of the classes are in rather low terrain (less than 1,000 states). Furthermore, the heights of the mountain peak do not allow to deduce anything about the overall median, because the median is not affected by the actual values of the data points which are greater than the median. This is the main characteristics of the median and applies to all the subsequent perspective plots in this chapter. The means in turn are 2,004.6 for Fribourg and 1955.9 for Fribourg+R2C.

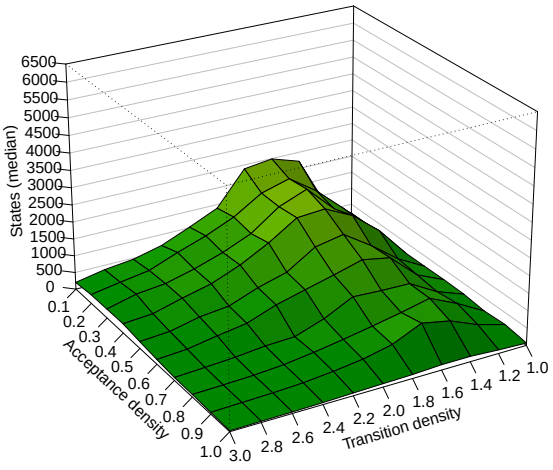
Fribourg+R2C+C in Figure 2.2 (c) has an even considerably higher mountain than the previous two versions. The top of the ridge is at around 5,000 states and the peak at the class with transition density 1.6 and acceptance density 0.3 has close to 6,500 states. This increase in height is however not reflected in the overall median of Fribourg+R2C+C relative to Fribourg+R2C. As we have seen in in Table 2.2, the median of Fribourg+R2C+C is 451, and thus 34.5% lower than the median of Fribourg+R2C. By taking a closer look at the perspective plots, the reason for this can be seen, as the low areas of Fribourg+R2C+C are indeed slightly lower than the low areas of Fribourg+R2C. The significantly higher high areas of Fribourg+R2C+C do not have an effect on the overall median. However, they do have an effect on the overall mean which with 2,424.6 is indeed 24% higher than the one of Fribourg+R2C (1,955.9).

Going from the first three perspective plot in Figure 2.2 to the perspective plot of Fribourg+R is like going from the Swiss Alps to a Dutch polder. The mountain shrinks to a small hillock and the rest of

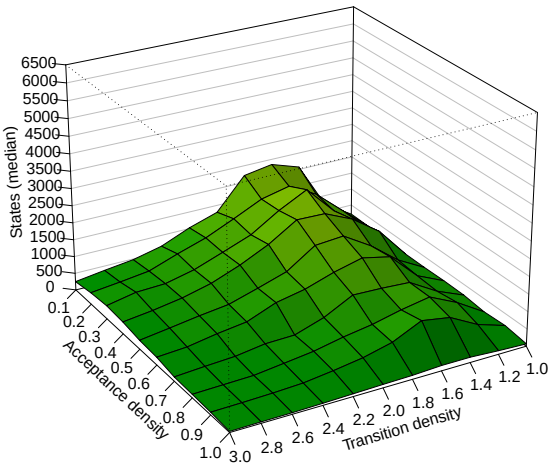
the terrain is low and flat. This is because so many complements of the Fribourg construction can be reduced to very small sizes by removing their unreachable and dead states. A look at the corresponding matrix in Appendix B.2 reveals that 68 of the 110 classes have a median complement size of 1. If we further compare this matrix to the matrix with the number of universal automata in Figure 1.3 (b), we see that all the classes with a median of 1 contain more than 50 universal automata, and the classes with a median greater than 1 contain less than 50 universal automata. There is a total of 100 automata per class. This makes sense as the complements of universal automata are empty automata, and every empty automaton can be reduced to an automaton with a single non-accepting state. Looking at the classes with a median greater than 1, we see that their values are still considerably lower than the ones of the plain Fribourg construction, which indicates that the Fribourg construction generates a large number of unreachable and dead states. However, as already mentioned, this thread of analysis is not the focus of the present thesis, and we leave it for future work.



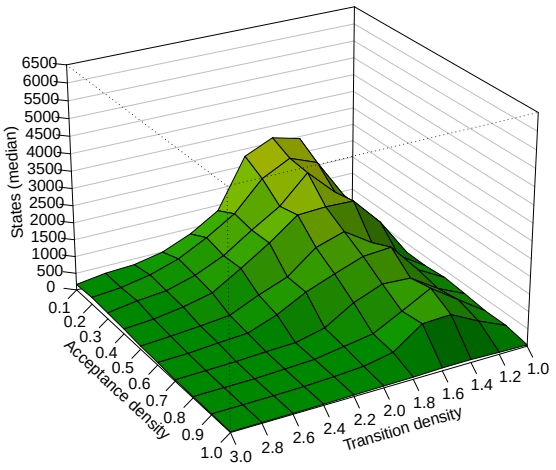
(a) Fribourg+M1



(b) Fribourg+M1+R2C



(c) Fribourg+M1+M2



(d) Fribourg+M1+R2C+C

Figure **??** shows the perspective plots of the remaining four versions of the Fribourg construction, all of which include the M1 optimisation. The first thing we note is that, while the mountain is still there, its height shrinks significantly. For Fribourg+M1, and Fribourg+M1+R2C, the top of the ridge is between is around 2,500 states. This is reflected by the overall means of these two versions compared to their counterparts without the M1 optimisation, Fribourg, and Fribourg+R2C. The decrease of the overall mean from Fribourg to Fribourg+M1 is by 52% (from 2004.6 to 963.2) and from Fribourg+R2C to Fribourg+M1+R2C by 52.1% (from 1955.9 to 937.7). The decreases of the overall medians are by 36.6% (from 761 to 482), and 35.1% (from 689 to 447) for the same two pairs of versions. With this we can confirm that the M1 optimisation brings a significant performance gain.

Regarding the M2 optimisation, we can see that the mountain ridge in the Fribourg+M1+M2 perspective

plot is slightly lower than the one in the Fribourg+M1 perspective plot. The flatland regions, however, seem to not change much. This is reflected by the overall mean of Fribourg+M1+M2 which is slightly lower than in Fribourg+M1 (958.9 opposed to 963.2). The overall median, on the other hand, is higher for Fribourg+M1+M2 than for Fribourg+M1 (496 opposed to 482). An interpretation of this behaviour is that the application of the M2 optimisation results in smaller complements for *some* input automata. These automata are especially the "hard" ones that produce large complements. This positive effect of M2 does however not affect enough input automata, especially not the "easy" automata, as to improve the overall performance of the construction in terms of median complement size. As already stated previously, we consider therefore Fribourg+M1 as the better construction on the GOAL test set than Fribourg+M1+M2.

Finally,Fribourg+M1+R2C+C differs from Fribourg+M1+R2C in a similar way Fribourg+R2C+C differs from Fribourg+R2C. The higher regions get higher and the lower regions get lower, that is, a performance decline on "hard" automata, but a performance gain on easy automata. The performance gain on the easy automata is however effective enough to decrease the overall median from 447 to 331, which is minus 26%.

With 331 states, Fribourg+M1+R2C+C has the lowest median of all the versions (except Fribourg+R which is a special case because it modifies the output of the construction). However, we still declare Fribourg+M1+R2C as the winner on the GOAL test set, mainly for two reasons. First, while Fribourg+M1+R2C+C has a lower median, the mean is still higher (1062.6 to 937.7 which is a plus of 13.3%). This results from the complements of the hard automata, which are larger than with Fribourg+M1+R2C. From a practical point of view, the mean might be relevant, because it relates more directly to the required computing resources than the median. For example the execution CPU time per complementation task, that we also measured along with our experiments, is 25.4% higher for Fribourg+M1+R2C+C than for Fribourg+M1+R2C. The increase in the average execution time per automaton is from 4.44 to 5.57 seconds and in the total execution time from 48,572 seconds ($\approx$ 135 hours) to 60,919 seconds ($\approx$ 169 hours). Fribourg+M1+R2C, on the other hand, has the lowest mean of all versions. The second reason that we choose Fribourg+M1+R2C as the winner and not Fribourg+M1+R2C+C is that the C option is not a real part of the construction. It actually modifies the input automata before the construction starts in order to make them better suited for the construction. Fribourg+M1+R2C, on the other hand, includes only native options, and input and output of the construction are not modified. This should also make it fairer to compare the Fribourg construction to other constructions in the external tests (Section 2.2).

As we have seen, there are big difference in the complement sizes across the different classes of the GOAL test set. Furthermore, there is a certain pattern, namely the mountain in the north-western region of the class matrix. Automata of classes that are in the mountain region seem to be harder to complement than automata from the classes in the flatland region. We attempted to categorise the classes of the GOAL test set into the three groups "easy", "medium", and "hard". To do so, we first averaged the matrices with the median complement sizes of all the eight versions of the Fribourg construction. In this way, we have a mean median complement size for each class. Then we defined two breakpoints that divide the classes into easy, medium, and hard groups. The breakpoints 500 and 1,600 result in an appropriate groups that seem to capture the reality well. The result with these breakpoints can be seen in Figure **??**.

As can be seen in Figure **??**, there are 53 easy, 36 medium, and 21 hard classes. The easy classes are mainly those with extreme values. In particular, all the classes with a low or high transition density of 0.1, or 2.8 and 3.0, and a high acceptance density of 1.0 are easy. Furthermore, there is a "triangle" of easy classes between transition densities 2.0 and 2.6. and acceptance densities 0.5 and 0.9. The higher the transition density, the lower the acceptance density may be so that the corresponding class is still easy. The hard classes are roughly those with a transition density between 1.4 and 1.8 and an acceptance density between 0.1 and 0.6. The medium classes finally are situated as a "belt" around the hard classes.

It is interesting that the extreme values of transition density and acceptance density result in easy automata. With a transition density of 1.0 and an alphabet size of 2, each of the 15 states has on average two outgoing and two incoming transitions[1]. With a transition density of 3.0, each state has on average 6 outgoing and 6 incoming transitions. These low or high connectivity seems to considerably simplify the

[1]The transition density multiplied with the number of states defines the number of transition for each symbol of the alphabet in the automaton, see Section 1.2.1.

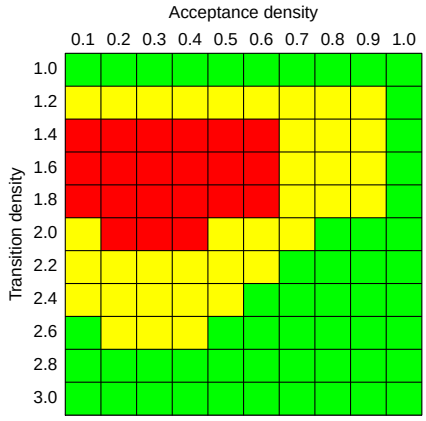


Figure 2.4: Difficulty categories of the transition/acceptance density classes of the GOAL test set. Green: easy; yellow: medium; red: hard.

complementation task. The same applies to a high acceptance density of 1.0, which means that every state is an accepting state[2]. Generally, we can say that automata with high acceptance densities are easier to complement than automata with lower acceptance densities. This also means that the pattern of easy automata at the extreme values of transition and accceptance density, does not apply to to the lower extreme of the acceptance density. Automata with a very low acceptance density of 0.1 are hard to complement—unless they are made easy by a low or high transition density.

Another interesting point is that the hard automata have transition densities between 1.4 and 1.8. It seems that this range of transition densities is the crucial factor in the hardness of a complementation task, and that it is only alleviated by a growing acceptance density. This explains the decline of the mountain ridge from west to east.

Summarising we can say that transition densities between 1.4 and 1.8 produce the hardest complementation tasks, and that to the both sides the difficulty steadily decreases with declining or growing transition density. Furthermore, a growing acceptance density generally implies easier complementation tasks.

2.1.2 Michel Test Set

We complemented the first four Michel automata with the six versions of the Fribourg constructions that are listed in Section 1.3.1. All complementation tasks were successful, as we did not set a timeout or memory limit. Remember that the first four Michel automata have 3, 4, 5, and 6 states, respectively. The sizes of the resulting complements are presented in Figure 2.3 in table form and visualised as a grouped barplot.

Construction	Michel 1	Michel 2	Michel 3	Michel 4	State growth	Std. error
Fribourg	57	843	14,535	287,907	$(1.35n)^n$	0.01%
Fribourg+R2C	33	467	8,271	168,291	$(1.24n)^n$	0.06%
Fribourg+M1	44	448	5,506	81,765	$(1.10n)^n$	0.07%
Fribourg+M1+M2	42	402	4,404	57,116	$(1.03n)^n$	0.12%
Fribourg+M1+M2+R2C	28	269	3,168	43,957	$(0.99n)^n$	0.04%
Fribourg+R	18	95	528	3,315	$(0.64n)^n$	0.35%

Table 2.3: Complement sizes of the first four Michel automata.

Looking at the results reveals two things. First, Michel automata result in very large complements. For example, Michel 4, which has six states, results in a complement with 287,907 states with the plain Fribourg construction. This is a state growth of $(1.354n)^n$. Exactly this state explosion is the aim of

[2]The acceptance density defines the percentage of states that are accepting, see Section 1.2.1 as well.

Michel automata, which have been developed with the goal to crate a set of automata that is extremely hard to complement.

This also shows why we were only able to complement the first four Michel automata. If we assume the same state growth of $(1.354n)^n$ for the fifth Michel automaton, Michel 5, with seven states, then the complement would have around 6.9 million states. The execution time for Michel 4 with the plain Friboug construction was 100,976 CPU time seconds ($\approx$ 180 hours). If we assume a linear relation between the number of generated states and the execution time, then Michel 5 would have an execution time of more than 2.4 million seconds, which is around 279 days. This estimation is still a very conservative, as the relation between complement size and execution time is not linear but seems to be exponential as well (see table with exeuction times in appendix).

2.2 External Tests

2.2.1 GOAL Test Set

Construction	Timeouts	Memory excesses
Piterman+EQ+RO	2	0
Slice+P+RO+MADJ+EG	0	0
Rank+TR+RO	3,713	83
Fribourg+M1+R2C	1	0

Table 2.4: Number of timeouts and memory excesses.

- With Rank
  - 7,204 effective samples (3,796 uncompleted tasks, 34.51%)
- Without Rank
  - 10,998 effective samples (2 uncompleted tasks, 0.02%)

With Rank

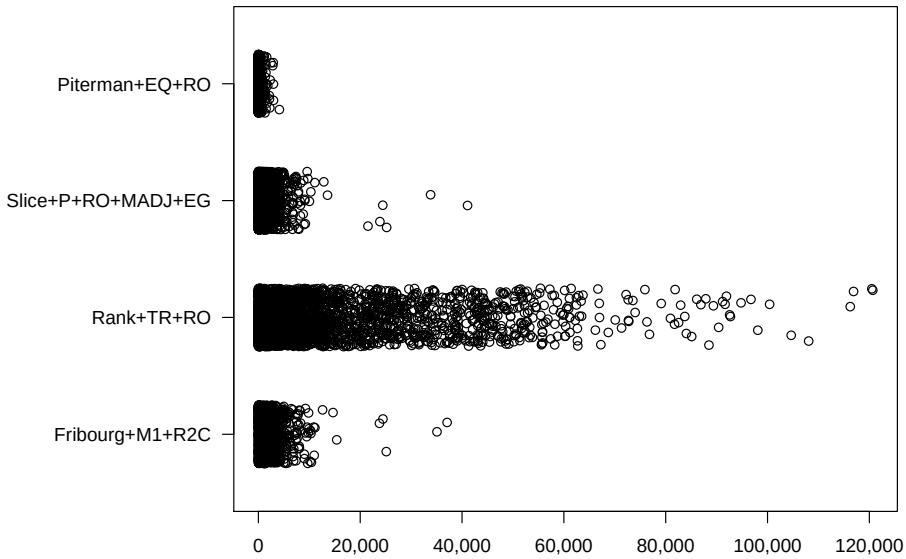


Figure 2.5: Complement sizes of the 7,204 effective samples.

Construction	Mean	Min.	P25	Median	P75	Max.
Piterman+EQ+RO	106.0	1	29.0	58.0	121.0	4,126
Slice+P+RO+MADJ+EG	555.4	2	70.0	202.0	596.0	41,081
Rank+TR+RO	5,255.6	2	81.0	254.5	3,178.2	120,674
Fribourg+M1+R2C	662.9	2	101.0	269.0	754.5	37,068

Table 2.5: Aggregated statistics of complement sizes of the 7,204 effective samples.

Construction	Mean	Min.	P25	Median	P75	Max.
Piterman+EQ+RO	2.97	2.22	2.58	2.78	3.03	42.93
Slice+P+RO+MADJ+EG	3.66	2.21	2.69	3.22	4.07	36.67
Rank+TR+RO	16.04	2.28	2.76	3.71	9.31	443.33
Fribourg+M1+R2C	4.02	2.22	2.69	3.10	4.37	410.37

Table 2.6: Aggregated statistics of the running times of the 7,204 effective samples.

Without Rank

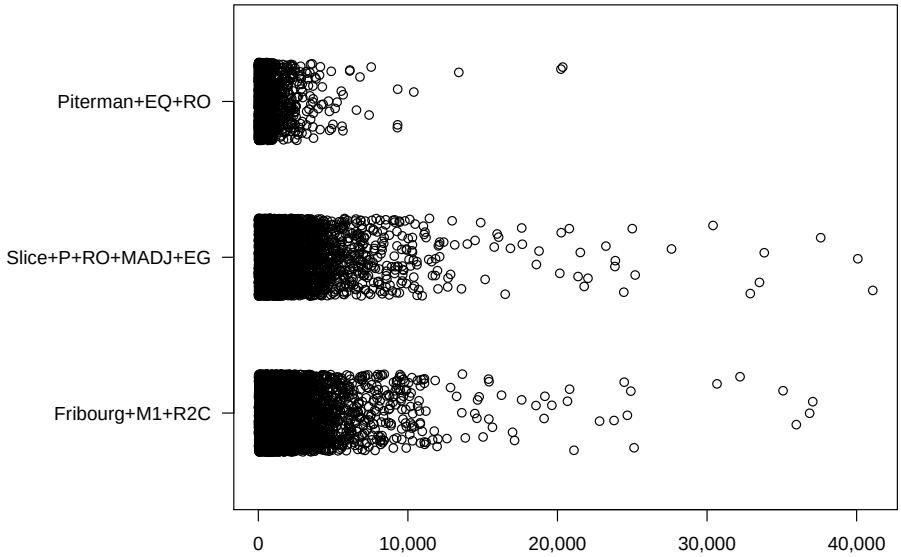


Figure 2.6: Complement sizes of the 10,998 effective samples.

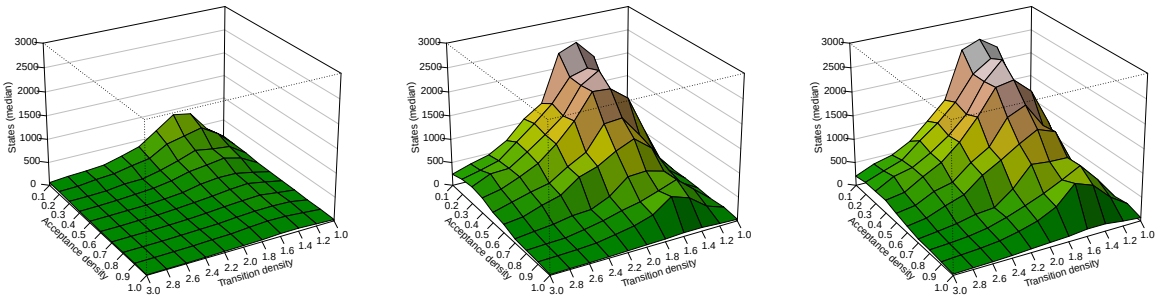
2.2.2 Michel Automata

Construction	Mean	Min.	P25	Median	P75	Max.
Piterman+EQ+RO	209.6	1	38.0	80.0	183.0	20,349
Slice+P+RO+MADJ+EG	949.4	2	120.0	396.0	1,003.0	41,081
Fribourg+M1+R2C	1,017.3	2	153.0	452.0	1,134.0	37,068

Table 2.7: Aggregated statistics of complement sizes of the 10,998 effective samples without Rank.

Construction	Mean	Min.	P25	Median	P75	Max.
Piterman+EQ+RO	3.61	2.22	2.66	2.90	3.38	365.68
Slice+P+RO+MADJ+EG	4.31	2.21	2.94	3.73	5.00	42.37
Fribourg+M1+R2C	4.74	2.22	2.83	3.61	5.31	410.37

Table 2.8: Aggregated statistics of the running times of the 10,998 effective samples without Rank.



(a) Piterman+EQ+RO (b) Slice+P+RO+MADJ+EG (c) Fribourg+M1+R2C

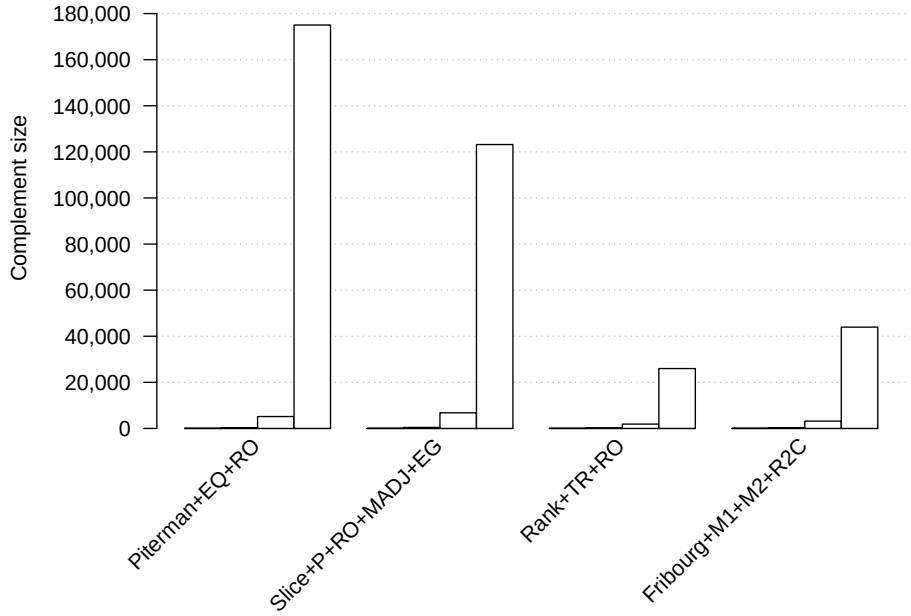
Figure 2.7: Median complement sizes.

Construction	Michel 1	Michel 2	Michel 3	Michel 4
Piterman+EQ+RO	23	251	5,167	175,041
Slice+P+RO+MADJ+EG	35	431	6,786	123,180
Rank+TR+RO	23	181	1,884	25,985
Fribourg+M1+M2+R2C	28	269	3,168	43,957

Table 2.9: Execution times for the first four Michel automata.

Construction	Michel 1	Michel 2	Michel 3	Michel 4
Piterman+EQ+RO	23	251	5,167	175,041
Slice+P+RO+MADJ+EG	35	431	6,786	123,180
Rank+TR+RO	23	181	1,884	25,985
Fribourg+M1+M2+R2C	28	269	3,168	43,957

(a) Complement sizes of the first four Michel automata.



(b) Complement sizes of the first four Michel visualised as a barplot. The bars 1 to 4 of each group correspond to Michel automata 1 to 4.

Figure 2.8: Complement sizes of Michel automata.

Appendix A

Plugin Installation and Usage

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	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.1: 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.

	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.2: 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.

Appendix C

Execution Times

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

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

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

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

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

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

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

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

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

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

Bibliography

[1] 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.

[2] 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.

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

[4] 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.

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

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

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.