Empirical Performance Investigation of a Büchi Complementation Construction

Daniel Weibel

July 23, 2015

Abstract

This will be the abstract.

Acknowledgements

Contents

Chapter 1

Introduction

At the beginning of the 1960s, a Swiss logician named Julius Richard Büchi at Michigan University was looking for a way to prove the decidability of the satisfiability of monadic second order logic with one successor (S1S). Büchi applied a trick that truly founded a new paradigm in the application of logic to theoretical computer science. He thought of interpretations of a S1S formula as infinitly long words of a formal language and designed a type of finite state automaton that accepts such a word if and only if the interpretation it represents satisfies the formula. After proving that every S1S formula can be translated to such an automaton and vice versa (Büchi's Theorem), the satisfiabilty problem of an S1S formula could be reduced to testing the non-emptiness of the corresponding automaton.

This special type of finite state automaton was later called Büchi automaton.

1.1 Context of Study

1.1.1 Büchi Automata and Büchi Complementation

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

The complement of a Büchi automaton A is another Büchi automaton[1] and is denoted by $\overline{A}$. Both, A and $\overline{A}$, share the same alphabet Σ. Regarding any 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 that are *accepted* by an automaton are *rejected* by its complement, and all the words that are *rejected* by an automaton are *accepted* by its complement. In other words, there is no single word that is either accepted or rejected by *both* of an automaton and its complement.

The complementation of Büchi automata, in particular non-deterministic Büchi automata, is commonly known as "Büchi complementation" or the "Büchi complementation problem". It is a very complex problem because it exhibits a very high state growth, which is sometimes even called state explosion (in the following, we will use the terms state growth, state explosion, and state complexity interchangeably). State growth denotes the relation of the number of states of a complement $\overline{A}$ (output of the complementation construction) to the number of states of the automaton A (input to the complementation construction). This relation is for worst-case automata exponential, even for an ideal complementation construction[2]. Even though the state growth that existing complementation constructions produce for many non-worst-case automata is not nearly as high as the worst case, it may still be very high. This is a serious problem, because Büchi complementation has important practical applications (as we will see next), and it is the reason that the quest for more efficient and more practical Büchi complementation constructions is still an active research topic today.

1.1.2 Language Containment

An important application of Büchi complementation is language containment of ω-regular languages. The ω-regular languages form the class of languages that is equivalent to non-deterministic Büchi automata. The language containment problem consists in determining whether $L_1 \subseteq L_2$, that is, whether a language L_1 is *contained* in another langauge L_2. This is true if every word of L_1 is also in L_2.

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

[2]Yan proved in 2007 a lower bound for the worst-case state growth of Büchi complementation of $(0.76n)^n$, where n is the number of states of the initial automaton [55].

The way $L_1 \subseteq L_2$ is commonly resolved is by testing $L_1 \cap \overline{L_2} = \varnothing$. Here, $\overline{L_2}$ denotes the complement language of L_2. This means, we have to create the intersection language, say $L_{1,\overline{2}}$ of L_1 and the complement of L_2, and then test whether $L_{1,\overline{2}}$ is empty (that is, contains no words at all). If $L_{1,\overline{2}}$ is empty, then there is no word of L_1 that is not also in L_2, and $L_1 \subseteq L_2$ is true. If $L_{1,\overline{2}}$ is non-empty, then there is at least one word of L_1 that is not in L_2, and $L_1 \subseteq L_2$ is false.

With this procedure, we in fact reduce the language containment problem to three operations on languages: complementation, intersection, and emptiness testing. By translating the languages L_1 and L_2 to equivalent automata A_1 and A_2, and mainpulating the automata instead of the languages, the problem becomes $L(A_1 \cap \overline{A_2}) = \varnothing$. That is, we complement A_2, create the intersection automaton $A_{1,\overline{2}}$ of A_1 and the complement of A_2, and test whether the language of $A_{1,\overline{2}}$ is empty, which is done by directly testing the automaton $A_{1,\overline{2}}$ for emptiness.

In this way, we reduce the language containment problem of ω-regular languages to three operations on non-deterministic Büchi automata: complementation, intersection, and emptiness testing. Büchi complementation is thus an integral part of the language contaiment problem. However, this does not yet answer our initial question of what is a *concrete* and *practical* application of Büchi complementation. To answer this question, we will in the following describe one important application of langauge containment of ω-regular languages.

1.1.3 Automata-Theoretic Model Checking via Language Containment

The language containment approach to automata-theoretic model checking is an approach to automata-theoretic model checking, which is an approach to general 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.

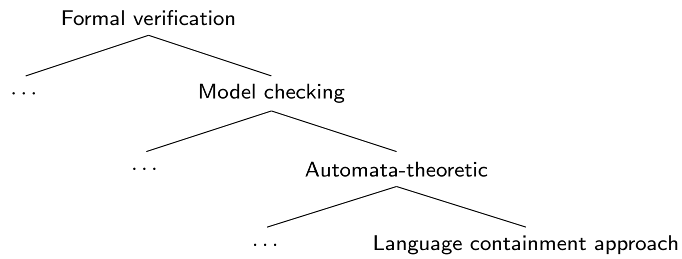

Figure 1.1: Branch of the family of formal verification techniques that contains the language containment approach to automata-theoretic model checking.

Formal verification is the use of mathemtical techniques for proving the correctness of a system (software of hardware) with respect to a specified set of properties [**?**]. A typical example is to verify that a program is deadlock-free. 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 properties that the system must satisfy
3. A verification method for testing whether the system satisfies the properties

For the language containment approach to automata-theoretic model checking, the frameworks for modelling the system to verify and for specifying the properties to be verified are both Büchi automata. The verification method is to test language containment of the languages corresponding to the two Büchi automata. In more detail, the approach works as we explain in the following.

The system s to be verified is modelled as a Büchi automaton, say S. Each word of the language $L(S)$ of S corresponds to a possible computation trace of the system s. A computation trace is an infinite sequence of "combinations of properties" of the system. Such properties can be for example variable values or statuses of individual processes. Each element of a computation trace corresponds to a point in time during the execution of the system[3]. The language $L(S)$ represents thus everything that the system *can* do.

A property p to be verified is represented as a non-deterministic Büchi automaton, say P. The words of the language $L(P)$ of P also represent computation traces. In particular, the language $L(P)$ represents all the possible computation traces that do satisfy the property p. If for example p is "deadlock-freeness", then $L(P)$ contains all the possible computation traces of a system like s that are deadlock-free. In other words, the language $L(P)$ represents everything that a system is *allowed* to do, with respect to a property p.

The verification step then consists in testing $L(S) \subseteq L(P)$, that is, whether everything that the system *can* do is contained in everything that the system is *allowed* to do. If this is true, then every possible computation trace of the system satisfies the property p, because it is contained in $L(P)$. If p means deadlock-freeness, then we can conclude that the system is deadlock-free. If the language containment test returns a negative result, then there must be at least one computation trace of the system that does not satisfiy the property p, because it is not in $L(P)$. In that case, this computation trace (however improbable it is) leads, for example, to a deadlock, and we have to conclude that the system is not deadlock-free.

As we have seen, the

Property p as automaton P. $L(P)$ defines everything that is *allowed* to do with respect to satisfiying the property p.

Automata-theoretic model checking is an approach to model checking, which in turn is an approach to formal verification. Formal verification means the use of mathemtical techniques for proving the correctness of a system (software of hardware) with respect to a specification [**?**]. A typical example is to prove that a system has no deadlocks.

The langauge containment approach to automata-theoretic model checking works as follows. The system, whose correctness is to prove, is represented as a Büchi automaton, say S. This automaton S defines a language $L(S)$, and the words of this language correspond to all the possible computation traces that the system can produce.

On the other hand, the property that the system must satisfy (for example, deadlock-freeness) is represented as another Büchi automaton, say P. The words of the language $L(P)$ correspond to all the possible computation traces that satisfy the property.

With these two representations in place, the verification step is done by testing whether $L(S) \subseteq L(P)$. That is, whether the language defined by the system automaton S is contained in the language defined by the property automaton P. As we have seen, this problem is solved by testing whether $L(S) \cap L(\overline{P}) = \varnothing$. This is in turn algorithmically done by testing $L(S \cap \overline{P}) = \varnothing$, which includes the following three steps:

1. Construct the complement $\overline{P}$ of the property automaton P
2. Construct the intersection of S and $\overline{P}$, that is, $A_{S,\overline{P}} = S \cap \overline{P}$
3. Test whether $A_{S,\overline{P}}$ is empty

If the emptiness test is positive, then $L(S) \subseteq L(P)$ is true, and the system satsifies the property (for example, deadlock-freeness) with all its possible computation traces. If the emptiness test is negative, then $L(S) \subseteq L(P)$ is false, and there is at least one computation trace of the system that violates the property.

As can be seen from these three steps, the verification problem is reduced to three operations on non-deterministic Büchi automata: (1) complementation, (2) intersection, and (3) emptiness testing. It turns out that intersection and emptiness testing have efficient solutions [49], whereas

Where Büchi complementation is used and why it is important

[3] 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.

- What are Büchi automata (very short)
- What is Büchi complementation (very short)
- Application of Büchi complementation (longer)
  - Main usage in anguage containment: $L_1 \subseteq L_2$ done by testing whether $L_1 \cap \overline{L_2} = \varnothing$
    * In terms of automata: $L(A) \subseteq L(A')$ by testing $L(A) \cap L(\overline{A'}) = \varnothing$, that is A' must be complemented
  - Important application of language containment: language containment approach to automata-theoretic model checking
    * Model system as Büchi automaton M
    * Represent specification properties as Büchi automaton P
    * Test $L(M) \subseteq L(P)$, that is, $L(M) \cap L(\overline{P}) = \varnothing$
    * Need to complement Büchi automaton P, which is very difficult. Alternatives:
      · Specify property as deterministic Büchi automaton (complementation is easy). Disadvantage: DBW less expressive, less intuitive, larger automata
      · Directly represent negation of properites as Büchi automaton. Disadvantage: difficult
      · Different approach to automata-theoretic model checking: specify properties as LTL formulas, negate them, and translate to Büchi automaton, model system as labelled transition system and translate to Büchi automaton (used by SPIN). Disadvantage: LTL is less expressive than Büchi automata
    * Importance of more efficient Büchi complementation: so far no tool includes complementation of Büchi automata [**?**]

A Büchi complementation construction takes as input a Büchi automaton A and produces as output another Büchi automaton B which accepts the complement language of the input automaton A. Complement language denotes the "contrary" language, that is, B must *accept* (over a given alphabet) every word that A *does not* accept, and must in turn *not accept* every word that A *accepts*.

Büchi automata are finite automata (that is, having a finite number of states) which operate on infinite words (that is, words that "never end"). Operating on infinite words, they belong thus to the category ω-automata. An important application of Büchi automata is in model checking which is a formal system verification technique. There, they are used to represent both, the description of the system to be checked for the presence of a correctness property, and (the negation of) this correctness property itself.

In one approach to model checking, the correctness property is directly specified as a Büchi automaton One approach to model checking requires that the Büchi automaton representing the correctness property is complemented. It is here that the problem of Büchi complementation has one of its practical applications.

1.1.4 Stating the problem, reason the research is worth tackling

Regarding the state complexity of Büchi complementation constructions, only the worst-case state growths have been investigated. However, they are a poor guide to actual peformance of constructions [42]. Need for empirical complexity investigations to see the *actual* performance of complementation constructions.

The complementation of non-deterministic Büchi automata is hard. It has been proven to have an exponential lower bound in the number of generated states [cite]. That is, the number of states of the output automaton is, in the worst case, an exponential function of the number of states of the input automaton. However, since the introduction of Büchi automata in the 1960's, significant process in reducing the complexity (in other words, the degree of exponentiality) of the Büchi complementation problem has been made. Some numbers [list complexities of the different constructions].

1.1.5 Aim and Scope

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,

1.1.6 Overview

Appendix A

Plugin Installation and Usage

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

Appendix B

Median Complement Sizes of the GOAL Test Set

Bla bla bla

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

(a) Fribourg

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

(b) Fribourg+R2C

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

(c) Fribourg+R2C+C

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

(d) Fribourg+M1

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

(e) Fribourg+M1+M2

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

(f) Fribourg+M1+R2C

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

(g) Fribourg+M1+R2C+C

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

(h) Fribourg+R

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

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

(a) Piterman+EQ+RO

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

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

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

Appendix C

Execution Times

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

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

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

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

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

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

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

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

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

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

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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