BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2022
http://excel.fit.vutbr.cz

Efficient Complementation of Elevator Büchi Automata

Barbora Šmahlíková*

Abstract
Büchi automata complementation is an important operation for decision procedures of some logics, proving termination of programs, or model checking of temporal properties. Due to the high space complexity of Büchi automata complementation, it is necessary to look for some optimizations reducing the size of generated state space. In this paper, we identify elevator automata, a class of Büchi automata, which often occurs in practice. Thanks to their specific structure, we are able to reduce the bound on the maximum rank of states in each strongly connected component, which is one of the main causes of a state space blow-up in rank-based complementation. We compare our techniques, implemented as an extension of the tool RANKER, with other state-of-the-art tools and show that with these optimizations, rank-based complementation is competitive to other complementation approaches and can give better results on a large set of benchmarks.

Keywords: Büchi Automata — Elevator Automata – Büchi Complementation

Supplementary Material: Downloadable Code

*xsmahl00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Büchi automata (BA) were introduced in 1960s as an auxiliary tool for a decision procedure of a fragment of a second-order arithmeric [1]. This paper focuses on complementing Büchi automata, which is a crucial operation for decision procedures of various logics, such as the monadic second-order logic S1S [1] or temporal logics ETL and QPTL [2], as well as for language inclusion and equivalence testing. Besides the theoretical point of view, Büchi automata complementation became important also in practice, for example in model checking of temporal properties [3] or termination analysis of programs [4, 5, 6].

The purpose of model checking is to automatically check whether a system meets its specification. Both the system and the specified (temporal) property can be represented by a Büchi automaton. The problem of system verification is then transformed into the problem of language inclusion of these automata. More precisely, a system meets its specification if the language of its Büchi automaton is a subset of the language of the property automaton. Language inclusion check is performed by complementing the property automaton and checking if its intersection with the system automaton is empty.

The idea behind termination analysis of programs [4, 5, 6] is to construct a difference of two Büchi automata — one representing the program and one representing a set of paths with proved termination. These paths can be safely removed from the program automaton. The removal is done using automata difference, which is implemented as an intersection of the program automaton and the complement of the automaton with terminating paths.

Due to the high space complexity of Büchi complementation, different approaches and further optimizations have been introduced since the original construction by Büchi was presented in 1962. Apart from reducing the upper bound of the size of the complemented automaton, there was also an effort to find the theoretical lower bound, finally refined by Yan to $(0.76n)^n$ [7]. We focus on the *rank-based* complementation, which was introduced by Kupferman and Vardi [8], improved with the help of Friedgut [9], and fur-

ther optimized by Schewe [10], whose construction produces the complement with the size matching the lower bound modulo a $\mathcal{O}(n^2)$ polynomial factor.

Even though Schewe's construction asymptotically matches the lower bound of $(0.76n)^n$, it may still generate a lot of unnecessary states and transitions. Optimization heuristics are therefore critical for good performance in practice. In *rank-based* complementation, every state from a set of states reachable over the current input is assigned a number (called its *rank*). The main problem responsible for the generated state space blow-up is the amount of nondeterminism, caused by a lot of possibilities how to assign ranks to a set of states. The number of possibilities depends combinatorially on the maximum rank that can be assigned. It is therefore desirable to reduce the maximum rank as much as possible.

In this paper, we identify elevator automata, a class of Büchi automata occuring often in practice, and present an algorithm which can reduce the maximum rank of each state and thus reduce the generated state space. We implemented our optimizations in the tool RANKER [11] and evaluated our approach on a large set randomly generated BAs and BAs obtained from LTL formulae. Our techniques significantly reduce the generated state space and in some cases can produce an exponentially better result compared to Schewe's optimal construction.

The following presents a part of my research published at *TACAS'22* [12].

2. Preliminaries

An *alphabet* is a nonempty, finite set of symbols. The symbol ω is used to denote the set of non-negative integers $\{0,1,2,3,\ldots\}$. An (infinite) *word* α over alphabet Σ is represented as a function $\alpha\colon \omega \to \Sigma$ where the i-th symbol is denoted as α_i. We abuse notation and sometimes represent α as an infinite sequence $\alpha = \alpha_0\alpha_1\ldots$ We use Σ^ω to denote the set of all infinite words over Σ.

A *Büchi automaton* (BA) is a quintuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ is an alphabet, δ is a transition function $\delta\colon Q \times \Sigma \to 2^Q$, $I \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states.

A *run* of $\mathcal{A}$ on a word α is an infinite sequence $\rho = q_0q_1q_2\ldots$ such that $q_0 \in I$ and $q_{i+1} \in \delta(q_i, \alpha_i)$ for every $i \geq 0$. Let $\inf(\rho)$ denote the set of states occuring infinitely often in the run ρ of $\mathcal{A}$ on a word α. The run ρ is called *accepting* iff $\inf(\rho) \cap F \neq \emptyset$. The word α is *accepted* by $\mathcal{A}$ if there exists an accepting run ρ of $\mathcal{A}$ on α. The set of all words accepted by $\mathcal{A}$ is called the *language* of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$.

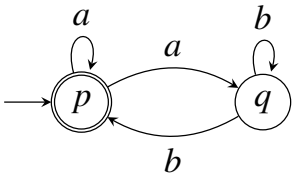


Figure 1. Büchi automaton $\mathcal{A}_{ex}$

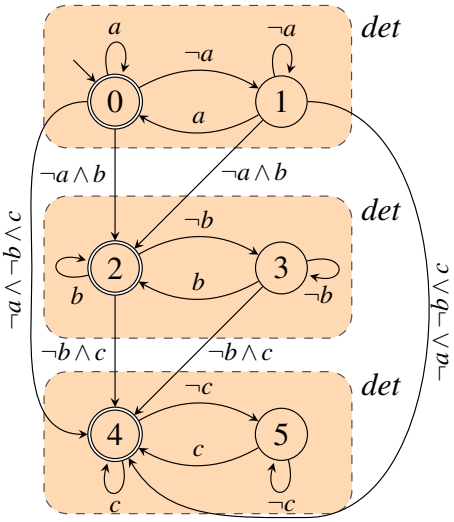


Figure 2. Elevator automaton for LTL formula $GF(a \vee GF(b \vee GFc))$

Figure 1 shows an example of Büchi automaton $\mathcal{A}_{ex} = (Q, \Sigma, \delta, I, F)$ with $Q = \{p,q\}$, $\Sigma = \{a,b\}$, $I = \{p\}$, $F = \{p\}$, and $\delta = \{p \xrightarrow{a} p, p \xrightarrow{a} q, q \xrightarrow{b} q, q \xrightarrow{b} p\}$.

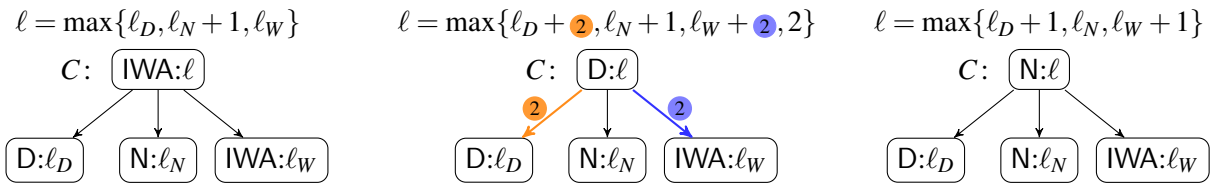
A *strongly connected component* (SCC) of $\mathcal{A}$ is a maximal set of states $C \subseteq Q$ such that for any pair of states $q, q' \in C$ it holds that q is reachable from q' and q' is reachable from q. The notation $\delta_{|S}$ for $S \subseteq Q$ is used to denote the restriction of the transition function $\delta \cap (S \times \Sigma \times S)$.

Let C be an SCC of a given Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ and $\mathcal{A}_{|C} = (C, \Sigma, \delta_{|C}, I \cap C, F \cap C)$. We say that C is *deterministic* iff the BA $\mathcal{A}_{|C}$ is deterministic, *non-accepting* iff $C \cap F = \emptyset$, *inherently weak accepting* iff every cycle in the transition diagram of $\mathcal{A}_{|C}$ contains an accepting state $q_F \in F$, and *trivial* iff $|C| = 1$ and $\delta_{|C} = \emptyset$.

3. Complementing Büchi Automata

Büchi automata complementation is a demanding and complex problem and researchers have been searching for more efficient complementation techniques since the original algorithm by Büchi [1] was introduced in 1960s. There are several approaches for complementing BAs, for example Ramsey-based [2, 13, 14], determinization-based [15, 16] or slice-based [17] complementation approach.

In this paper, we focus on *rank-based* comple-



(a) C is IWA **(b)** C is D **(c)** C is N

Figure 3. Rules for assigning types and rank bounds to SCCs. The symbols 2 and 2 are interpeted as 0 if all the corresponding edges from the components having rank ℓ_D and ℓ_W, respectively, are deterministic; otherwise they are interpreted as 2. Transitions between two components C_1 and C_2 are deterministic if the BA $(C, \delta|_C, \emptyset, \emptyset)$ is deterministic for $C = \delta(C_1, \Sigma) \cap (C_1 \cup C_2)$.

mentation, which was first introduced by Kupferman and Vardi [8] with the space complexity $2^{\mathcal{O}(n \log n)}$, then improved by Kupferman, Vardi, and Friedgut [9] to $\mathcal{O}((0.96n)^n)$ and made asymptotically optimal by Schewe [10]. The space complexity of Schewe's construction matches the theoretical lower bound $\mathcal{O}((0.76n)^n)$ given by Yan [7] modulo a quadratic factor $\mathcal{O}(n^2)$.

In order to show the reason of a blow-up in the generated state space of the complement, we have to first introduce some necessary definitions. For a given Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, a *level ranking* is a function $f\colon Q \to \{0, 1, \ldots, 2|Q|\}$ such that $\{f(q_f) \mid q_f \in F\} \subseteq \{0, 2, \ldots, 2|Q|\}$, i.e., f maps all accepting states of $\mathcal{A}$ to even ranks. Given a set of states $S \subseteq Q$, a (level) ranking $f\colon Q \to \{0, 1, \ldots, 2|Q|\}$ is called *S-tight* if it has an odd rank r, $\{f(s) \mid s \in S\} \supseteq \{1, 3, \ldots, r\}$, and $\{f(q) \mid q \notin S\} = \{0\}$. A ranking is *tight* if it is Q-tight. We use $\mathcal{T}$ to denote the set of all tight level rankings.

Let $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ be a BA. The algorithm from [10], denoted as SCHEWE, constructs its complement $\mathcal{C} = (Q_1 \cup Q_2, \Sigma, \delta', I', F')$ with macrostates of the following form:

- $Q_1 = 2^Q$, and
- $Q_2 = \{(S, O, f, i) \in 2^Q \times 2^Q \times \mathcal{T} \times \{0, 2, \ldots, 2n-2\} \mid f \text{ is } S\text{-tight}, O \subseteq S \cap f^{-1}(i)\}$.

When we want to compute successors of a state in Q_2 (which is also in Q_2) over a given symbol, all three components S, O and i of a macrostate are given deterministically. The high amount of nondeterminism is caused by the f-component, i.e., by the number of all possible tight level rankings of a successive macrostate. For a given macrostate, the number of possible tight rankings rises combinatorially with the macrostate's maximum rank. More precisely, for a given set of n states and a maximum rank r, it corresponds to $r!$ multiplied by the Stirling number of the second kind $S(n, r)$ [9] - the number of ways to partition a set of n labeled objects (states) into r nonempty unlabeled subsets (ranks). A maximum rank sufficient for each state is $2|Q| - 1$ [8]. Some optimizations that can reduce the amount of nondeterminism were already presented in [18].

4. Elevator Automata

In this section, we introduce *elevator automata*, a large class of Büchi automata with a particular structure, that can be effectively complemented by setting bounds on the maximum rank for states in every SCC.

A Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ is an *elevator* (Büchi) automaton if for every SCC C of $\mathcal{A}$ it holds that C is (i) deterministic, (ii) inherently weak accepting (IWA), or (iii) non-accepting. Elevator automata occur often in practice, for example a lot of BAs obtained from LTL formulae. An example of a BA obtained from LTL formula $GF(a \vee GF(b \vee GFc))$ is shown in Figure 2. This automaton consists of three deterministic components.

We present an algorithm that assigns each SCC a label of the form TYPE:*rank* with the type of SCC and the bound on the maximum rank of its states. The assignment is performed from terminal SCCs (i.e., SCCs from which it cannot be reached to any other SCC) towards SCCs with initial states. More precisely, a label can be assigned to SCC C only if (i) C is terminal, or (ii) a label was already assigned to all SCCs reachable from C. We consider the following types of SCCs: inherently weak accepting (IWA), deterministic (D), and non-accepting (N). Note that there can be more options to assign a type for some SCCs. The algorithm assigns the type that is most suitable in terms of keeping the rank bound as low as possible. This can be different for every SCC, depending on the labels of its successors.

For the following algorithm, we assume that an elevator automaton contains no useless states (there is therefore no terminal non-accepting SCC).

For a terminal SCC C, we assign the following label:

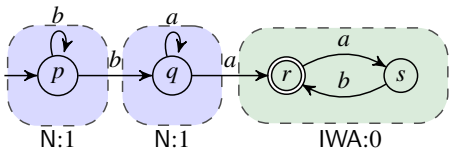


Figure 4. Elevator automaton $\mathcal{A}_{el}$

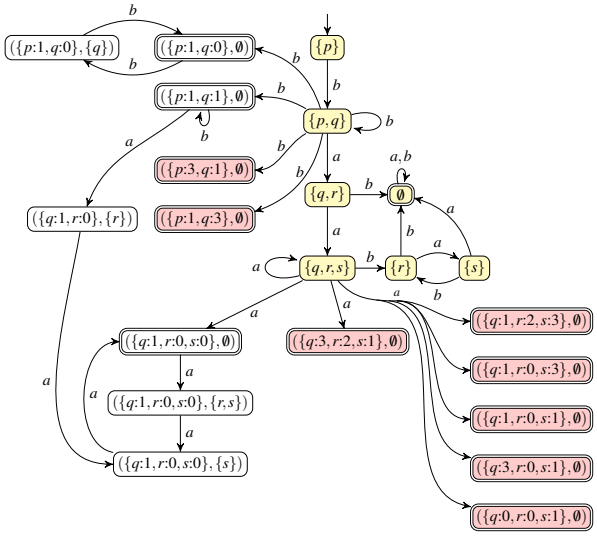


Figure 5. Complement of $\mathcal{A}_{el}$. Our procedure will not generate the red states (and their successors, which are not shown in the figure).

1. IWA:0 if C is inherently weak accepting,
2. D:2 otherwise.

For non-terminal SCCs, we use the corresponding rules from Figure 3. Children nodes denote already processed successive SCCs. In particular, a child node of the form k:ℓ_k denotes an aggregate of all siblings of the type k with ℓ_k being the maximum rank of these siblings. For a non-terminal SCC C, the rules for assigning a label are the following:

1. If C is trivial, we try both rules from Figure 3a and Figure 3c and use the one with smaller rank.
2. Else if C is IWA, we use the rule in Figure 3a.
3. Else if C is deterministic accepting, we use the rule in Figure 3b.
4. Else if C is deterministic and non-accepting, we use one of the rules in Figure 3b and Figure 3c that gives us a smaller rank.
5. Else if C is nondeterministic and non-accepting, we use the rule in Figure 3c.

The maximum ranks of each SCC is then assigned to all its states and macrostates with higher ranks are not generated. Figure 4 shows an elevator automaton $\mathcal{A}_{el}$ with assigned label for each SCC. Complement of $\mathcal{A}_{el}$ is in Figure 5. Red macrostates are not generated because value assigned to some state is higher than the rank bound on maximum rank.

The algorithm for elevator automata can also be extended to non-elevator BAs, i.e., BAs containing a nondeterministic accepting SCC that is not inherently weak (see [12]).

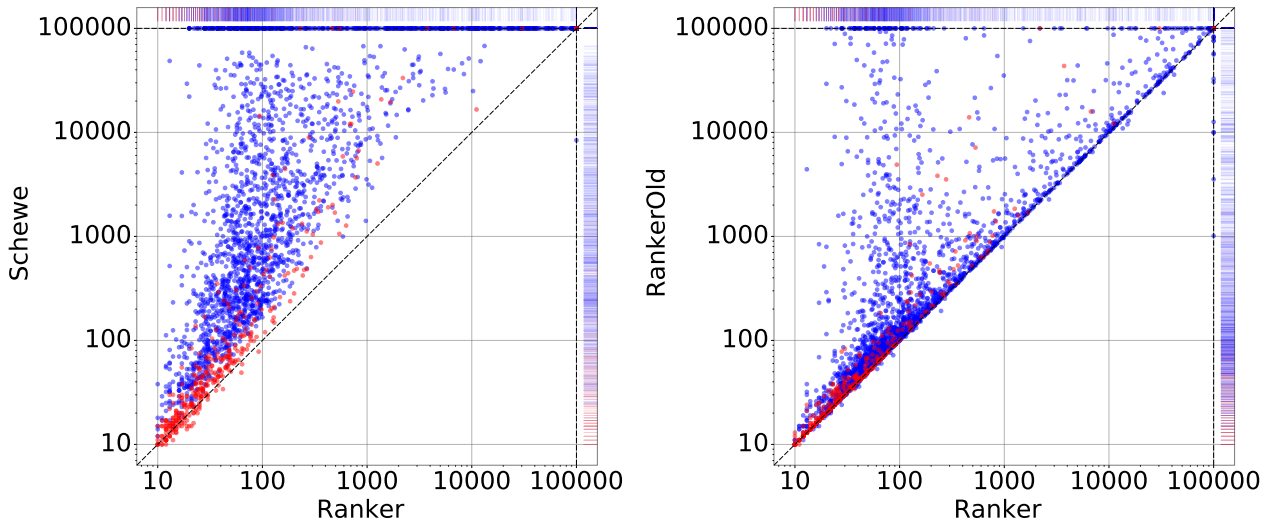
The algorithm is based on a special structure of the *run DAGs* [8] - directed acyclic graphs capturing all runs of an input BA over a given (infinite) word. For all BAs with n states, setting maximum rank for each state to $2|n| - 1$ is sufficient. This can be seen from a ranking procedure presented in [8]. However, for elevator automata, we are able to significantly reduce the maximum considered rank for each component because of the structure of run DAGs. Whole reasoning is shown in [12].

5. Experiments

The optimization for complementing elevator automata described in Section 4 was implemented as an extension of the tool RANKER [11] in C++. We used two datasets for our experiments: (i) `random` with 11 000 BAs over a two letter alphabet used in [19], and (ii) `LTL` containing 1 721 BAs over larger alphabets (up to 128 symbols) used in [20], obtained from LTL formulae from literature or randomly generated. The automata were preprocessed using RABIT [21] and SPOT's `autfilt` (using the `--high` simplification level), trnasformed to state-based acceptance BAs (if they were not already), and converted to the HOA format [22]. From this set, we removed automata that were (i) semi-deterministic, (ii) inherently weak, (iii) unambiguous, or (iv) have an empty language, since for these automata types there exist more efficient complementation procedures than for unrestricted BAs [23, 20, 24, 25]. In the end, we were left with 2 592 (`random`) and 414 (`LTL`) *hard* automata. We use `all` to denote their union (3 006 BAs). Of these hard automata, 458 were elevator automata.

We compared RANKER with other state-of-the-art tools for BA complementation, namely, GOAL [26] (implementing PITERMAN [16], SCHEWE [10], SAFRA [15], and FRIBOURG [27]), SPOT 2.9.3 [28] (implementing Redziejowski's algorithm [29]), SEMINATOR 2 [20], LTL2DSTAR 0.5.4 [30], ROLL [31], and the previous version of RANKER (denoted as $\text{RANKER}_{\text{OLD}}$) with the heuristics from [18]. The experimental evaluation was performed on a 64-bit GNU/LINUX DEBIAN workstation with an Intel(R) Xeon(R) CPU E5-2620 running at 2.40 GHz with 32 GiB of RAM and using a timeout of 5 minutes.

Our first experiment the effectiveness of our heuristics for reducing the generated state space by comparing the sizes of complemented BAs without postpro-



(a) RANKER vs SCHEWE **(b)** RANKER vs RANKER$_{\text{OLD}}$

Figure 6. Comparison of the state space generated by our optimizations and other rank-based procedures (horizontal and vertical dashed lines represent timeouts). Blue data points are from `random` and red data points are from `LTL`. Axes are logarithmic.

cessing. These results are useful for applications where postprocessing is not needed, for example language inclusion or equivalence checking. Figure 6a compares the generated state space with SCHEWE (the version Reduced Average Outdegree from [10]) implemented in GOAL. Comparison with RANKER$_{\text{OLD}}$ is in Figure 6b. We can see that the improvement was exponential for many automata in both cases.

Our second experiment compares RANKER with other state-of-the-art tools. It compares sizes of output BAs, therefore, each automaton was postprocessed with `autfilt` (simplification level `--high`). Scatter plots are given in Figure 7, where we compare RANKER with SPOT (which had the best results on average from the other tools except ROLL) and ROLL. Summary statistics are in the lower part of Table 1. RANKER has the second lowest mean (after ROLL) and the third lowest median (after SEMINATOR 2 and ROLL). The columns *wins* and *losses* represent the number of cases where RANKER gives strictly better or worse result, respectively. Comparing these columns, we can see that RANKER gives more wins than losses in comparison with any other tool.

In Figure 7a, we can see that in the majority of cases RANKER gives a smaller BA than SPOT, especially for harder BAs (SPOT, however, behaves slightly better on the simpler BAs from `LTL`). The results in Figure 7b do not seem so clear. ROLL uses a learning-based approach—more heavyweight and completely orthogonal to any of the other tools—and can in some cases output a tiny automaton, but does not scale, as observed by the number of timeouts much higher than any other tool.

Regarding runtimes, RANKER is comparable to SEMINATOR 2, but slower than SPOT and LTL2DSTAR (SPOT is the fastest tool). The number of timeouts of RANKER is still higher than of some other tools (especially PITERMAN, SPOT, and FRIBOURG).

6. Conclusions

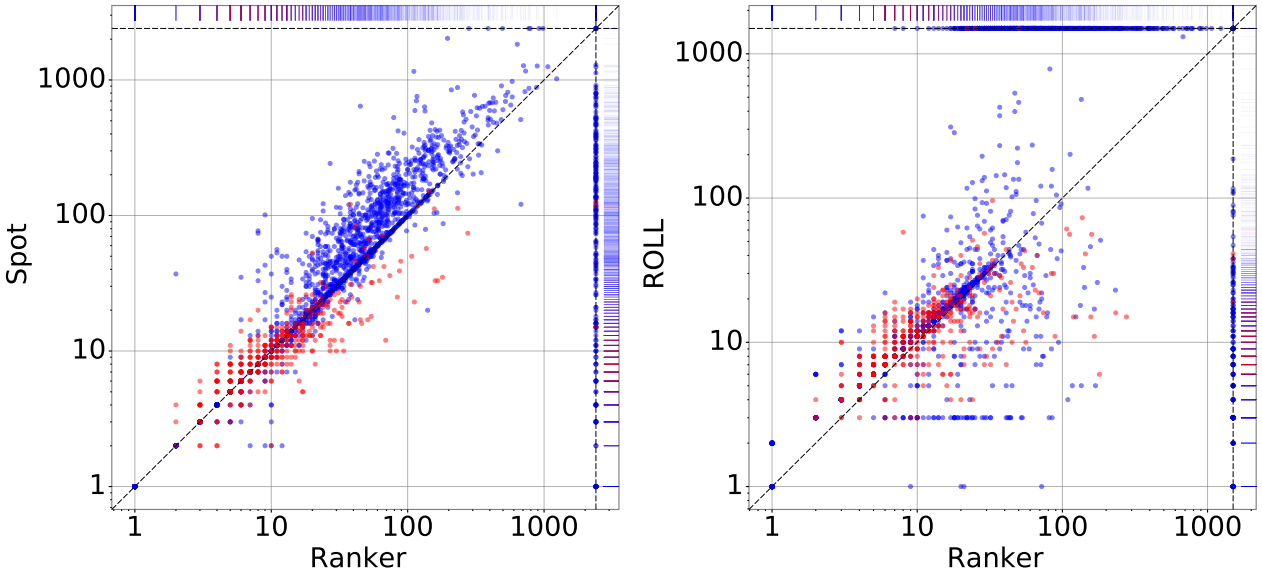
The main problem of rank-based complementation is the high amount of nondeterminism and unnecessarily high bounds on the maximum rank causing state space blow-up. Using the techniques presented in this paper, we are able to significantly reduce the complement size of elevator automata, a large class of Büchi automata occuring often in practice. Moreover, these techniques can also be extended to general Büchi automata containing a deterministic, nonaccepting or inherently weak accepting component. In comparison to other BA complementation approaches, rank-based complementation may be quite uneffective. However, using the optimizations from this paper (along with other optimizations from [18]), rank-based complementation becomes competitive to other approaches and in a lot of cases can give better results than other state-of-the-art tools.

Acknowledgements

I would like to thank my supervisor Ondřej Lengál and my consultant Vojtěch Havlena for their ideas, advice, and support.

Table 1. Statistics for our experiments. The upper part compares various optimizations of the rank-based procedure (no postprocessing). The lower part compares RANKER to other approaches (with postprocessing). The left-hand side compares sizes of complement BAs and the right-hand side runtimes of the tools. The wins and losses columns give the number of times when RANKER was strictly better and worse. The values are given for the three datasets as "`all` (`random` : `LTL`)".

method	mean		median		wins		losses		mean runtime [s]		median runtime [s]		timeouts	
RANKER	3812	(4452 : 207)	79	(93 : 26)					7.83	(8.99 : 1.30)	0.51	(0.84 : 0.04)	279	(276 : 3)
RANKER$_{\text{OLD}}$	7398	(8688 : 358)	141	(197 : 29)	2190	(2011 : 179)	111	(107 : 4)	9.37	(10.73 : 1.99)	0.61	(1.04 : 0.04)	365	(360 : 5)
SCHEWE	4550	(5495 : 665)	439	(774 : 35)	2640	(2315 : 325)	55	(1 : 54)	21.05	(24.28 : 7.80)	6.57	(7.39 : 5.21)	937	(928 : 9)
RANKER	47	(52 : 18)	22	(27 : 10)					7.83	(8.99 : 1.30)	0.51	(0.84 : 0.04)	279	(276 : 3)
PITERMAN	73	(82 : 22)	28	(34 : 14)	1435	(1124 : 311)	416	(360 : 56)	7.29	(7.39 : 6.65)	5.99	(6.04 : 5.62)	14	(12 : 2)
SAFRA	83	(91 : 30)	29	(35 : 17)	1562	(1211 : 351)	387	(350 : 37)	14.11	(15.05 : 8.37)	6.71	(6.92 : 5.79)	172	(158 : 14)
SPOT	75	(85 : 15)	24	(32 : 10)	1087	(936 : 151)	683	(501 : 182)	0.86	(0.99 : 0.06)	0.02	(0.02 : 0.02)	13	(13 : 0)
FRIBOURG	91	(104 : 13)	23	(31 : 9)	1120	(1055 : 65)	601	(376 : 225)	17.79	(19.53 : 7.22)	9.25	(10.15 : 5.48)	81	(80 : 1)
LTL2DSTAR	73	(82 : 21)	28	(34 : 13)	1465	(1195 : 270)	465	(383 : 82)	3.31	(3.84 : 0.11)	0.04	(0.05 : 0.02)	136	(130 : 6)
SEMINATOR 2	79	(91 : 15)	21	(29 : 10)	1266	(1131 : 135)	571	(367 : 204)	9.51	(11.25 : 0.08)	0.22	(0.39 : 0.02)	363	(362 : 1)
ROLL	18	(19 : 14)	10	(9 : 11)	2116	(1858 : 258)	569	(443 : 126)	31.23	(37.85 : 7.28)	8.19	(12.23 : 2.74)	1109	(1106 : 3)



(a) RANKER VS SPOT **(b)** RANKER VS ROLL

Figure 7. Comparison of the complement size obtained by RANKER and other state-of-the-art tools (horizontal and vertical dashed lines represent timeouts). Axes are logarithmic.

My Contribution

The work presented in this paper is a part of my research published at *TACAS'22*, on which we worked together with my supervisor and consultant. In order to not take credit for all the work, I specify what my original contribution is. It is especially an extension of the definition of elevator automata (originally without inherently weak accepting components), and the implementation of the algorithm assigning rank bound to each state of an automaton.

References

[1] J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proc. of International Congress on Logic, Method, and Philosophy of Science 1960*. Stanford Univ. Press, Stanford, 1962.

[2] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49(2-3):217–237, 1987.

[3] Moshe Y. Vardi and Pierre Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE, 1986.

[4] Seth Fogarty and Moshe Y. Vardi. Büchi complementation and size-change termination. In *Proc. of TACAS'09*, pages 16–30. Springer, 2009.

[5] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Termination analysis by learning terminating programs. In *Proc. of CAV'14*, pages 797–813. Springer, 2014.

[6] Yu-Fang Chen, Matthias Heizmann, Ondrej Lengál, Yong Li, Ming-Hsien Tsai, Andrea Turrini, and Lijun Zhang. Advanced automata-based algorithms for program termination checking. In Jeffrey S. Foster and Dan Grossman, editors, *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*, pages 135–150. ACM, 2018.

[7] Qiqi Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. of ICALP'06*, pages 589–600. Springer, 2006.

[8] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.

[9] Ehud Friedgut, Orna Kupferman, and Moshe Vardi. Büchi complementation made tighter. *International Journal of Foundations of Computer Science*, 17:851–868, 2006.

[10] Sven Schewe. Büchi complementation made tight. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[11] Vojtěch Havlena, Ondřej Lengál, and Barbora Šmahlíková. RANKER, 2021. https://github.com/barbora4/ba-inclusion.

[12] Vojtěch Havlena, Ondřej Lengál, and Barbora Šmahlíková. Sky is not the limit: Tighter rank bounds for elevator automata in Büchi automata complementation. In *Proceedings of TACAS'22*. Springer Verlag, 2022. To appear. An extended version is available at https://arxiv.org/abs/2110.10187.

[13] F. P. Ramsey. On a Problem of Formal Logic. *Proceedings of the London Mathematical Society*, (1):264–286, 1930.

[14] Stefan Breuers, Christof Löding, and Jörg Olschewski. Improved Ramsey-based Büchi complementation. In *Proc. of FOSSACS'12*, pages 150–164. Springer, 2012.

[15] Shmuel Safra. On the complexity of ω-automata. In *Proc. of FOCS'88*, pages 319–327. IEEE, 1988.

[16] Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. of LICS'06*, pages 255–264. IEEE, 2006.

[17] Detlef Kähler and Thomas Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *Proc. of ICALP'08*, pages 724–735. Springer, 2008.

[18] Vojtěch Havlena and Ondřej Lengál. Reducing (To) the Ranks: Efficient Rank-Based Büchi Automata Complementation. In *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969.

[19] Ming-Hsien Tsai, Seth Fogarty, Moshe Y. Vardi, and Yih-Kuen Tsay. State of Büchi complementation. In Michael Domaratzki and Kai Salomaa, editors, *Implementation and Application of Automata*, pages 261–271, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[20] František Blahoudek, Alexandre Duret-Lutz, and Jan Strejček. Seminator 2 can complement generalized Büchi automata via improved semi-determinization. In *Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV'20)*, volume 12225 of *Lecture Notes in Computer Science*, pages 15–27. Springer, July 2020.

[21] Richard Mayr and Lorenzo Clemente. Advanced automata minimization. In *Proc. of POPL'13*, pages 63–74, 2013.

[22] Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi omega-automata format. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 479–486. Springer, 2015.

[23] František Blahoudek, Matthias Heizmann, Sven Schewe, Jan Strejček, and Ming-Hsien Tsai. Complementing semi-deterministic Büchi automata. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as*

Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings, volume 9636 of *Lecture Notes in Computer Science*, pages 770–787. Springer, 2016.

[24] Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625. Springer, 2001.

[25] Yong Li, Moshe Y. Vardi, and Lijun Zhang. On the power of unambiguity in Büchi complementation. In Jean-Francois Raskin and Davide Bresolin, editors, Proceedings 11th International Symposium on *Games, Automata, Logics, and Formal Verification,* Brussels, Belgium, September 21-22, 2020, volume 326 of *Electronic Proceedings in Theoretical Computer Science*, pages 182–198. Open Publishing Association, 2020.

[26] Ming-Hsien Tsai, Yih-Kuen Tsay, and Yu-Shiang Hwang. GOAL for games, omega-automata, and logics. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 883–889, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[27] Joël D. Allred and Ulrich Ultes-Nitsche. A simple and optimal complementation algorithm for Büchi automata. In *Proceedings of the Thirty third Annual IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 46–55. IEEE Computer Society Press, July 2018.

[28] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Étienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *Automated Technology for Verification and Analysis*, pages 122–129, Cham, 2016. Springer International Publishing.

[29] Roman R. Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundam. Informaticae*, 119(3-4):393–406, 2012.

[30] Joachim Klein and Christel Baier. On-the-fly stuttering in the construction of deterministic *omega* -automata. In Jan Holub and Jan Zdárek, editors, *Implementation and Application of Automata, 12th International Conference, CIAA 2007, Prague, Czech Republic, July 16-18, 2007, Revised Selected Papers*, volume 4783 of *Lecture Notes in Computer Science*, pages 51–61. Springer, 2007.

[31] Yong Li, Xuechao Sun, Andrea Turrini, Yu-Fang Chen, and Junnan Xu. ROLL 1.0: ω-regular language learning library. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 365–371. Springer, 2019.