Complementing Büchi Automata with Ranker

Vojtěch Havlena, Ondřej Lengál✉, and Barbora Šmahlíková
ihavlena@fit.vut.cz, lengal@vut.cz, xsmahl00@vut.cz

Faculty of Information Technology,
Brno University of Technology,
Czech Republic

CAV Artifact Evaluation ★ Available

CAV Artifact Evaluation ★ ★ ★ Reusable

Abstract. We present the tool Ranker for complementing Büchi automata (BAs). Ranker builds on our previous optimizations of rank-based BA complementation and pushes them even further using numerous heuristics to produce even smaller automata. Moreover, it contains novel optimizations of specialized constructions for complementing (i) inherently weak automata and (ii) semi-deterministic automata, all delivered in a robust tool. The optimizations significantly improve the usability of Ranker, as shown in an extensive experimental evaluation with real-world benchmarks, where Ranker produced in the majority of cases a strictly smaller complement than other state-of-the-art tools.

1 Introduction

Büchi automata (BA) complementation is an essential operation in the toolbox of automata theory, logic, and formal methods. It has many applications, e.g., implementing negation in decision procedures of some logics (such as the monadic second-order logic S1S [1,2], the temporal logics EPTL and QPTL [3], or the first-order logic over Sturmian words [4]), proving termination of programs [5,6,7], or model checking of temporal properties [8]. BA complementation also serves as the foundation stone of algorithms for checking inclusion and equivalence of ω-regular languages. In all applications of BAs, the number of states of a BA affects the overall performance. The many uses of BA complementation, as well as the challenging theoretical nature of the problem, has incited researchers to develop a number of different approaches, e.g., *determinization-based* [9,10,11], *rank-based* [12,13,14], or *Ramsey-based* [1,15], some of them [16,14] producing BAs with the number of states asymptotically matching the lower bound $(0.76n)^n$ of Yan [17]. Despite their theoretical optimality, for many real-world cases the constructions create BAs with a lot of unnecessary states, so optimizations making the algorithms efficient in practice are needed.

We present Ranker, a robust tool for complementing (transition-based) BAs. Ranker uses several complementation approaches based on properties of the input BA: it combines an optimization of the rank-based procedure developed in [18,19,20] with specialized (and further optimized) procedures for complementing semi-deterministic BAs [21], inherently weak BAs [22,23], and elevator BAs [19]. An extensive experimental evaluation on a wide range of automata occurring in practice shows that Ranker can obtain a smaller complement in the majority of cases compared to the other state-of-the-art tools.

Contribution. We describe a major improvement of Ranker [18,19], turning it from a prototype into a robust tool. We list the particular optimizations below.

– We extended the original BA complementation procedure with improved deelevation (cf. [19]) and advanced automata reductions.
– We also equipped Ranker with specialized constructions tailored for widely-used semi-deterministic and inherently weak automata.
– On top of that, we propose novel optimizations of the original NCSB construction for semi-deterministic BAs and a simulation-based optimization of the Miyano-Hayashi algorithm for complementing inherently weak automata.

All of these improvements are pushing the capabilities of Ranker, and also of practical BA complementation itself, much further.

2 Büchi Automata

Words, functions. We fix a finite nonempty alphabet Σ and the first infinite ordinal $\omega = \{0, 1, \ldots\}$. An (infinite) word α is 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 $\alpha = \alpha_0\alpha_1 \ldots$ Σ^ω denotes the set of all infinite words over Σ.

Büchi automata. A (nondeterministic transition/state-based) *Büchi automaton* (BA) over Σ is a quintuple $\mathcal{A} = (Q, \delta, I, Q_F, \delta_F)$ where Q is a finite set of *states*, $\delta\colon Q \times \Sigma \to 2^Q$ is a *transition function*, $I \subseteq Q$ is the sets of *initial* states, and $Q_F \subseteq Q$ and $\delta_F \subseteq \delta$ are the sets of *accepting states* and *accepting transitions* respectively. $\mathcal{A}$ is called deterministic if $|I| \leq 1$ and $|\delta(q, a)| \leq 1$ for each $q \in Q$ and $a \in \Sigma$. We sometimes treat δ as a set of transitions $p \xrightarrow{a} q$, for instance, we use $p \xrightarrow{a} q \in \delta$ to denote that $q \in \delta(p, a)$. Moreover, we extend δ to sets of states $P \subseteq Q$ as $\delta(P, a) = \bigcup_{p \in P} \delta(p, a)$. The notation $\delta|_S$ for $S \subseteq Q$ is used to denote the restriction of the transition function $\delta \cap (S \times \Sigma \times S)$. Moreover, for $q \in Q$, we use $\mathcal{A}[q]$ to denote the automaton $(Q, \delta, \{q\}, Q_F, \delta_F)$.

A *run* of $\mathcal{A}$ from $q \in Q$ on an input word α is an infinite sequence $\rho\colon \omega \to Q$ that starts in q and respects δ, i.e., $\rho_0 = q$ and $\forall i \geq 0\colon \rho_i \xrightarrow{\alpha_i} \rho_{i+1} \in \delta$. Let $\inf_{Q,\delta}(\rho) \subseteq Q \cup \delta$ denote the set of states and transitions occurring in ρ infinitely often. The run ρ is called *accepting* iff $\inf_{Q,\delta}(\rho) \cap (Q_F \cup \delta_F) \neq \emptyset$. A word α is *accepted by $\mathcal{A}$ from a state* $q \in Q$ if $\mathcal{A}$ has an accepting run ρ on α from q, i.e., $\rho_0 = q$. The set $\mathcal{L}_\mathcal{A}(q) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha \text{ from } q\}$ is called the *language* of q (in $\mathcal{A}$). Given a set of states $R \subseteq Q$, we define the language of R as $\mathcal{L}_\mathcal{A}(R) = \bigcup_{q \in R} \mathcal{L}_\mathcal{A}(q)$ and the language of $\mathcal{A}$ as $\mathcal{L}(\mathcal{A}) = \mathcal{L}_\mathcal{A}(I)$. If $\delta_F = \emptyset$, we call $\mathcal{A}$ *state-based* and if $Q_F = \emptyset$, we call $\mathcal{A}$ *transition-based*.

A *co-Büchi automaton* (co-BA) $\mathcal{C}$ is the same as a BA except the definition of when a run is accepting: a run ρ of $\mathcal{C}$ is *accepting* iff $\inf_{Q,\delta}(\rho) \cap (Q_F \cup \delta_F) = \emptyset$.

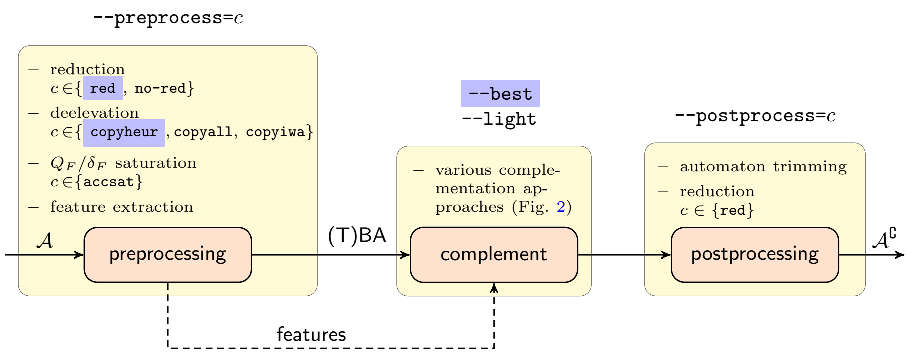


Fig. 1: Overview of the architecture of Ranker with the most important command-line options. Default settings are highlighted in blue.

Automata types. Let $\mathcal{A} = (Q, \delta, I, Q_F, \delta_F)$ be a BA. $C \subseteq Q$ is a *strongly connected component* (SCC) of $\mathcal{A}$ if for any pair of states $q, q' \in C$ it holds that q is reachable from q' and q' is reachable from q. C is *maximal* (MSCC) if it is not a proper subset of another SCC. An MSCC is *non-accepting* if it contains no accepting state and no accepting transition. We say that an SCC C is *inherently weak accepting* (IWA) iff *every cycle* in the transition diagram of $\mathcal{A}$ restricted to C contains an accepting state or an accepting transition. We say that an SCC C is *deterministic* iff $(C, \delta|_C, \emptyset, \emptyset, \emptyset)$ is deterministic. $\mathcal{A}$ is *inherently weak* (IW) if all its MSCCs are inherently weak accepting or non-accepting, and *weak* if for states q, q' that belong to the same SCC, $q \in Q_F$ iff $q' \in Q_F$. $\mathcal{A}$ is *semi-deterministic* (SDBA) if $\mathcal{A}[q]$ is deterministic for every $q \in Q_F \cup \{p \in Q \mid s \xrightarrow{a} p \in \delta_F, s \in Q, a \in \Sigma\}$. Finally, $\mathcal{A}$ is called *elevator* if all its MSCCs are inherently weak accepting, deterministic, or non-accepting.

3 Architecture

Ranker [24] is a publicly available command line tool, written in C++, implementing several approaches for complementation of (transition/state-based) Büchi automata. As an input, Ranker accepts BAs in the HOA [25] or the simpler `ba` [26] format. The architecture overview is shown in Fig. 1. An input automaton is first adjusted by various structural preprocessing steps to an intermediate equivalent automaton with a form suitable for a complementation procedure. Based on the intermediate automaton type, a concrete complementation procedure is used. The result of the complementation is subsequently polished by postprocessing steps, yielding an automaton on the output. In the following text, we provide details about the internal blocks of Ranker's architecture.

3.1 Preprocessing and Postprocessing

Before an input BA is sent to the complementation block itself, it is first transformed into a form most suitable for a concrete complementation technique.

On top of that as a part of preprocessing, we identify structural features that are further used to enabling/disabling certain optimizations during the complementation. After the complementation, the resulting automaton is optionally reduced in a postprocessing step. Ranker provides several options of preprocessing/postprocessing that are discussed below.

Preprocessing. The following are the most important settings for preprocessing:

- *Reduction*: In order to obtain a smaller automaton, reduction using *direct simulation* [27] can be applied (`--preprocess=red`). Moreover, if the input automaton is IW or SDBA, we transform it into a transition-based BA, which might be smaller (we only do local modifications and merge two states if they have the same successors while moving the acceptance condition from states to transitions entering accepting states). We, however, do not use this strategy for other BAs, because despite their possibly more compact representation, this reduction limits the effect of some optimizations used in the rank-based complementation procedure (the presence of accepting states allows to decrease the rank bound, cf. [19]).
- *Deelevation* [19]: For elevator automata, Ranker supports a couple of deelevation strategies (extending a basic version introduced in [19]). Roughly speaking, deelevation makes a copy of MSCCs such that each copied MSCC becomes a terminal component (i.e., no run can leave it) and accepting states/transitions are removed from the original component (we call this the *deelevation* of the component). Deelevation increases the number of states but decreases the rank bounds for rank-based complementation. Ranker offers several strategies that differ on which components are deelevated:
  - `--preprocess=copyall`: Every component is deelevated.
  - `--preprocess=copyiwa`: Only IWA components are deelevated.
  - `--preprocess=copyheur`: This option combines two modifications applied in sequence: (i) If the input BA is not IW and the rank bound estimation [19] of the BA is at least 5, then all MSCCs with an accepting state/transition are deelevated (the higher rank bound indicates a longer sequence of components, for which deelevation is likely to be benefical). (ii) If on all paths from all initial states of the intermediate BA, the first non-trivial MSCC is non-accepting, then we partially determinize the initial part of the BA (up to the first non-trivial MSCCs); this reduces sizes of macrostates obtained in rank-based complementation.
- *Saturation of accepting states/transitions*: Since a higher number of accepting states and transitions can help the rank-based complementation procedure, Ranker can (using `--preprocess=accsat`) saturate accepting states/transitions in the input BA (while preserving the language). This is, however, not always beneficial; for instance, saturation can break the structure for elevator rank estimation (cf. [19]).
- *Feature extraction*: During preprocessing, we extract features of the BA that can help the complementation procedure in the second step. The features are, e.g., the type of the BA, rank bounds for individual states [19], or settings of particular optimizations from [18] (e.g., for deterministic automata with

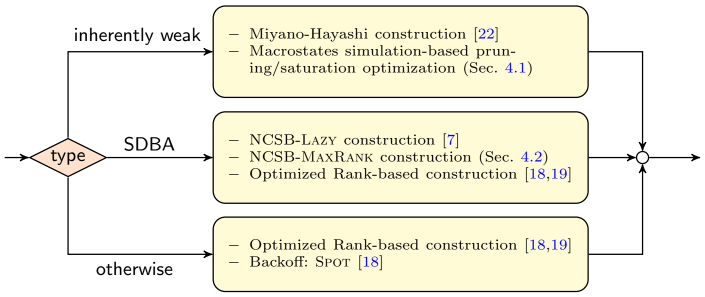

inherently weak
– Miyano-Hayashi construction [22]
– Macrostates simulation-based pruning/saturation optimization (Sec. 4.1)
type
SDBA
– NCSB-Lazy construction [7]
– NCSB-MaxRank construction (Sec. 4.2)
– Optimized Rank-based construction [18,19]
otherwise
– Optimized Rank-based construction [18,19]
– Backoff: Spot [18]

Fig. 2: Overview of complementation approaches used in Ranker.

a smaller rank bound, it is counter-productive to use techniques reducing the rank bound based on reasoning about the waiting part).

Postprocessing. After the complementation procedure finishes, Ranker removes useless states and optionally applies simulation reduction (`--postprocess=red`).

3.2 Complementation Approaches

Based on the automaton type, Ranker uses several approaches for complementation (cf. Fig. 2). These are, ordered by decreasing priority, the following:

- *Inherently weak BAs*: For the complementation of inherently weak automata, both the Miyano-Hayashi construction [22] and its optimization of adjusting macrostates (described in Sec. 4.1), are implemented. The construction converts an input automaton into an intermediate equivalent co-Büchi automaton, which is then complemented. The implemented optimizations adjust macrostates of the Miyano-Hayashi construction according to a direct simulation relation. By default (`--best`), the Miyano-Hayashi construction and the optimization of pruning simulation-smaller states from macrostates are used and the smaller result is output. For the option `--light`, only the optimized construction is used.
- *Semi-deterministic BA*: For SDBAs, Ranker by default (`--best`) uses both an NCSB-based [21] procedure and an optimized rank-based construction with advanced rank estimation [18,19]; the smaller result is picked. The particular NCSB-based procedure used is NCSB-MaxRank from Sec. 4.2 (Ranker also contains an implementation of NCSB-Lazy from [7], which can be turned on using `--ncsb-lazy`, but usually gives worse results). For the option `--light`, only NCSB-MaxRank is used.
- *Otherwise*: For BAs with no special structure, Ranker uses the optimized rank-based complementation algorithm from [18,19] with Spot as the backoff [18] (i.e., Ranker can determine when the input has a structure that is bad for the rank-based procedure and use another approach). Particular optimizations are selected according to the features of the input BA (e.g., the number of states or the structure of the automaton).

4 Optimizations of the Constructions

In this section, we provide details about new optimizations of complementation of inherently weak and semi-deterministic automata implemented in Ranker. Proofs of their correctness can be found in the technical report [28].

4.1 Macrostates Adjustment for Inherently Weak Automata

For complementing IW automata, Ranker uses a method based on the Miyano-Hayashi construction (denoted as MiHay) [22]: In the first step, accepting states of an input IW BA $\mathcal{A}$ are saturated to obtain a language-equivalent weak automaton $\mathcal{W} = (Q, \delta, I, Q_F, \emptyset)$ (we remove accepting transitions because they do not provide any advantage for IW automata). In the second step, $\mathcal{W}$ is converted to the equivalent co-Büchi automaton $\mathcal{C} = (Q, \delta, I, Q'_F = Q \setminus Q_F, \emptyset)$ by swapping accepting and non-accepting states. Finally, the Miyano-Hayashi construction is used to obtain the complement (state-based) BA.

Our optimizations of the MiHay procedure are inspired by optimizations of the determinization algorithm for automata over finite words [29] and by saturation of macrostates in rank-based BA complementation procedure [20], where simulation relations are used to adjust macrostates in order to obtain a smaller automaton. We modify the original construction by introducing an *adjustment function* that modifies obtained macrostates, either to obtain *smaller* macrostates (for *pruning* strategy) or *larger* macrostates (for *saturating* strategy; the hope is that *more* original macrostates map to *the same* saturated macrostate). Formally, given a co-BA $\mathcal{C}$ and an *adjustment function* $\theta\colon 2^Q \to 2^Q$, the construction MiHay_θ gives the (deterministic, state-based) BA $\text{MiHay}_\theta(\mathcal{C}) = (Q', \delta', I', Q'_F, \emptyset)$, whose components are defined as follows:

- $Q' = 2^Q \times 2^Q$,
- $I' = \{(\theta(I), \theta(I) \setminus Q'_F)\}$,
- $\delta'((S, B), a) = (S', B')$ where
  - $S' = \theta(\delta(S, a))$,
  - and
    - $B' = S' \setminus Q'_F$ if $B = \emptyset$ or
    - $B' = (\delta(B, a) \cap S') \setminus Q'_F$ if $B \neq \emptyset$, and
- $F' = 2^Q \times \{\emptyset\}$.

Intuitively, the construction tracks in the S-component all runs over a word and uses the B-component to check that each of the runs sees infinitely many accepting states from Q'_F (by a cut-point construction). The original MiHay procedure can be obtained by using identity for the adjustment function, $\theta = \mathrm{id}$.

In the following, we use $\preceq_{di}^{\mathcal{W}}$ and $\preceq_{f}^{\mathcal{C}}$ to denote a *direct simulation* on $\mathcal{W}$ and a *fair simulation* on $\mathcal{C}$ respectively (see, e.g., [30] for more details; in particular, $p \preceq_f^{\mathcal{C}} q$ iff for every trace of $\mathcal{C}$ from state p over α with finitely many accepting states, there exists a trace from q with finitely many accepting states over α).

Let $\sqsubseteq\, \subseteq Q \times Q$ be a relation on the states of $\mathcal{C}$ defined as follows: $p \sqsubseteq q$ iff (i) $p \preceq_f^{\mathcal{C}} q$, (ii) q is reachable from p in $\mathcal{C}$, and (iii) either p is not reachable from q in $\mathcal{C}$ or $p = q$. The two adjustment functions $pr, sat \colon 2^Q \to 2^Q$ are then defined for each $S \subseteq Q$ as follows:

– *pruning*: $pr(S) = S'$ where $S' \subseteq S$ is the lexicographically smallest set (given a fixed ordering on Q) such that $\forall q \in S \exists q' \in S' \colon q \sqsubseteq q'$ and
– *saturating*: $sat(S) = \{p \in Q \mid \exists q \in Q \colon p \preceq_f^{\mathcal{C}} q\}$.

Informally, pr removes simulation-smaller states and sat saturates a macrostate with all simulation-smaller states.[1] The correctness of the constructions is summarized by the following theorem:

Theorem 1. *For a co-BA $\mathcal{C}$, $\mathcal{L}(\textsc{MiHay}_{sat}(\mathcal{C})) = \mathcal{L}(\textsc{MiHay}_{pr}(\mathcal{C})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{C})$.*

In Ranker, we approximate a fair simulation $\preceq_f^{\mathcal{C}}$ by a direct simulation $\preceq_{di}^{\mathcal{W}}$ (which is easier to compute); the correctness holds due to the following lemma:

Lemma 2. *Let $\mathcal{W} = (Q, \delta, I, Q_F, \emptyset)$ be a weak BA and $\mathcal{C} = (Q, \delta, I, Q'_F = Q \setminus Q_F, \emptyset)$ be a co-BA. Then $\preceq_{di}^{\mathcal{W}} \subseteq \preceq_f^{\mathcal{C}}$.*

4.2 NCSB-MaxRank Construction

The structure of semi-deterministic BAs allows to use more efficient complementation techniques. From the point of view of rank-based complementation, the maximum rank of semi-deterministic automata can be bounded by 3. If a rank-based complementation procedure based on *tight rankings* (such as [18,19]) is used to complement an SDBA, it can suffer from having too many states due to the presence of the *waiting* part (intuitively, runs wait in the waiting part of the complement until they can see only tight rankings, then they jump to the *tight* part where they can accept, cf. [13,14,18] for more details). Furthermore, the information about ranks of individual runs may sometimes be more precise than necessary, which disables merging some runs. The NCSB construction [21] overcomes these issues by not considering the waiting part and keeping only rough information about the ranks. As a matter of fact, NCSB and the rank-based approach are not comparable due to tight-rankings and additional techniques restricting the ranking functions [18,19], taking into account structural properties of the automaton, which is why Ranker in the default setting tries both rank-based and NCSB-based procedures for complementing SDBAs.

An issue of the NCSB algorithm is a high degree of nondeterminism of the constructed BA (and therefore also a higher number of states). The NCSB-Lazy construction [7] improves the original algorithm with postponing the nondeterministic choices, which usually produces smaller results. Even the NCSB-Lazy construction may, however, suffer in some cases from generating too many successors. We propose an improvement of the original NCSB algorithm, inspired

[1] It has been brought to our attention by Alexandre Duret-Lutz that a strategy similar to *pruning* with direct simulation has been implemented in Spot's [31] determinization and, moreover, generalized in [32] to also work in some cases *within* SCCs.

by the MaxRank construction in rank-based complementation from [18] (which is inspired by [14, Section 4]), hence called the NCSB-MaxRank construction, reducing the number of successors of any macrostate and symbol to at most two.

Formally, for a given SDBA $\mathcal{A} = (Q_1 \uplus Q_2, \delta = \delta_1 \uplus \delta_2 \uplus \delta_t, I, Q_F, \delta_F)$ where Q_2 are the states reachable from an accepting state or transition and Q_1 is the rest, $\delta_1 = \delta_{|Q_1}$, $\delta_2 = \delta_{|Q_2}$, and δ_t is the transition function between Q_1 and Q_2, we define NCSB-MaxRank$(\mathcal{A}) = (Q', I', \delta', Q'_F, \emptyset)$ to be the (state-based) BA whose components are the following:

- $Q' = \{(N, C, S, B) \in 2^{Q_1} \times 2^{Q_2} \times 2^{Q_2 \setminus Q_F} \times 2^{Q_2} \mid B \subseteq C\}$,
- $I' = \{(Q_1 \cap I, Q_2 \cap I, \emptyset, Q_2 \cap I)\}$,
- $\delta' = \gamma_1 \cup \gamma_2$ where the successors of a macrostate (N, C, S, B) over $a \in \Sigma$ are defined such that if $\delta_F(S, a) \neq \emptyset$ then $\delta'((N, C, S, B), a) = \emptyset$, else
  - $\gamma_1((N, C, S, B), a) = \{(N', C', S', B')\}$ where
    - $N' = \delta_1(N, a)$,
    - $S' = \delta_2(S, a)$,
    - $C' = (\delta_t(N, a) \cup \delta_2(C, a)) \setminus S'$, and
    - $B' = C'$ if $B = \emptyset$, otherwise $B' = \delta_2(B, a) \cap C'$.
  - If $B' \cap Q_F = \emptyset$, we also set $\gamma_2((N, C, S, B), a) = \{(N', C^\bullet, S^\bullet, B^\bullet)\}$ with
    - $B^\bullet = \emptyset$,
    - $S^\bullet = S' \cup B'$, and
    - $C^\bullet = C' \setminus S^\bullet$,

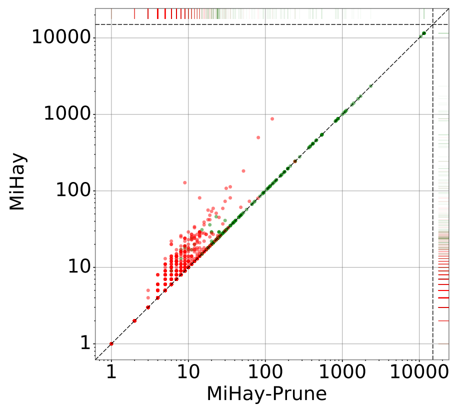
    else $\gamma_2((N, C, S, B), a) = \emptyset$.
- $Q'_F = \{(N, C, S, B) \in Q' \mid B = \emptyset\}$.

Intuitively, NCSB-MaxRank provides at most two choices for each macrostate: either keep all states in B or move all states from B to S (if B contains no accepting state). If a word is not accepted by $\mathcal{A}$, it will be safe to put all states from B to S at some point. The construction is in fact incomparable to the original NCSB algorithm [21] (in particular due to the condition $C' \subseteq \delta_2(C \setminus Q_F, a)$, which need not hold in NCSB-MaxRank). Correctness of the construction is given by the following theorem.

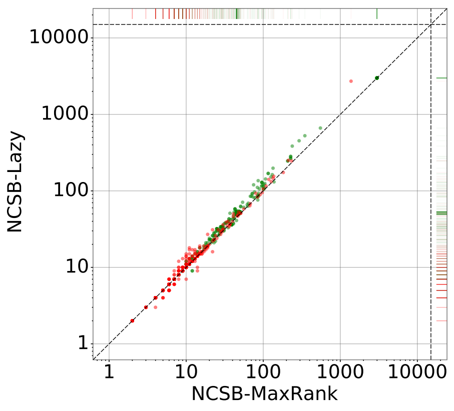
Theorem 3. *Let $\mathcal{A}$ be an SDBA. Then $\mathcal{L}(\text{NCSB-MaxRank}(\mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.*

5 Experimental Evaluation

We compared the improved version of Ranker presented in this paper with other state-of-the-art tools, namely, Goal [33] (implementing Piterman [10], Safra [9], and Fribourg [16]), Spot 2.9.3 [31] (implementing Redziejowski's algorithm [11]), Seminator 2 [34], LTL2dstar 0.5.4 [35], Roll [36], and the previous version of Ranker from [19], denoted as $\text{Ranker}_{\text{Old}}$. All tools were set to the mode where they output a state-based BA. The correctness of our implementation was tested using Spot's `autcross` on all of BAs from our benchmarks. 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, using a 5-minute timeout. Axes in plots are logarithmic. An artifact that allows reproduction of the results is available as [37].



(a) MiHay$_{pr}$ vs MiHay



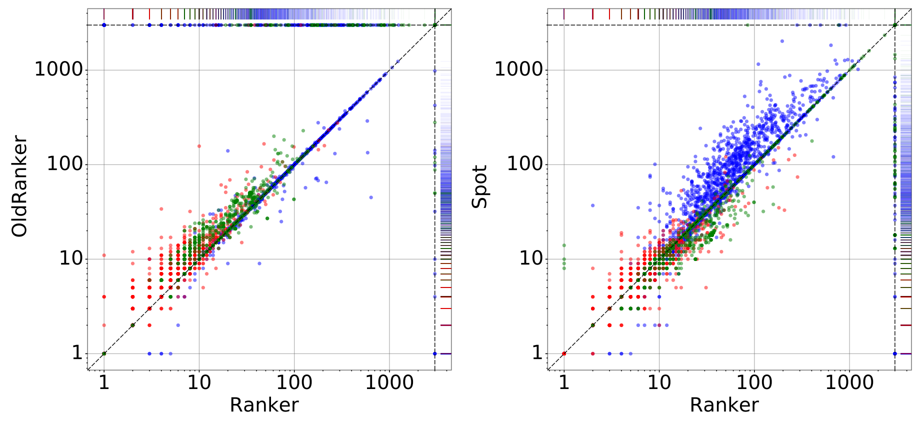
(b) NCSB-MaxRank vs NCSB-Lazy

Fig. 3: Evaluation of the effect of our optimizations for IW and SDBA automata.

Datasets. We use automata from the following three datasets: (i) `random` containing 11,000 BAs over a two letter alphabet used in [38], which were randomly generated via the Tabakov-Vardi approach [39], starting from 15 states and with various parameter settings; (ii) `LTL` with 1,721 BAs over larger alphabets (up to 128 symbols) used in [34], obtained from LTL formulae from literature (221) or randomly generated (1,500), (iii) `Automizer` containing 906 BAs over larger alphabets (up to 2^{35} symbols) used in [7], which were obtained from the Ultimate Automizer tool (all benchmarks are available at [40]). Note that we included `random` in order to simulate applications that cannot easily generate BAs of one of the easier fragments (unlike, e.g., Ultimate Automizer, which generates in most cases SDBAs) and have thus, so far, not been seriously considered by the community due to the lack of practically efficient BA complementation approaches (e.g., the automata-based S1S decision procedure [1]). All automata were preprocessed using Spot's `autfilt` (using the `--high` simplification level), and converted to the HOA format [25]. We also removed trivial one-state BAs. In the end, we were left with 4,533 (`random`, **blue** data points), 1,716 (`LTL`, **red** data points), and 906 (`Automizer`, **green** data points) automata. We use `all` to denote their union (7,155 BAs).

5.1 Effect of the Proposed Optimizations

In the first part of the experimental evaluation, we measured the effect of the proposed optimizations from Sec. 4 on the size of the generated state space, i.e., sizes of output automata without any postprocessing. This use case is motivated by language inclusion and equivalence checking, where the size of the generated state space directly affects the performance of the algorithm. We carried out the evaluation on `LTL` and `Automizer` benchmarks (we use `both` to denote their union) since most of the automata there are either IW or SDBAs.



(a) Ranker vs Ranker$_{\text{Old}}$ (b) Ranker vs Spot

Fig. 4: Comparison of the complement size obtained by Ranker, Ranker$_{\text{Old}}$, and Spot (horizontal and vertical dashed lines represent timeouts).

Table 1: Effects of our optimizations for IW and SDBA automata. Sizes of output BAs are given as "both (LTL : Automizer)".

method	mean	median
MiHay$_{pr}$	43.4 (7.3 : 140.7)	7 (5 : 21)
MiHay	46.1 (10.9 : 141.3)	7 (6 : 23)
NCSB-MaxRank	30 (20.3 : 38.3)	12 (8 : 28)
NCSB-Lazy	35.7 (25.1 : 44.8)	13 (9 : 32)

The first experiment compares the number of states generated by the original MiHay and by the macrostates-pruning optimization MiHay$_{pr}$ from Sec. 4.1 on inherently weak BAs (948 BAs from LTL and 360 BAs from Automizer = 1,308 BAs). Note that we omit MiHay$_{sat}$ as it is overall worse than MiHay$_{pr}$. The scatter plot is shown in Fig. 3a and statistics are in the top part of Table 1. We can clearly see that the optimization works well, substantially decreasing both the mean and the median size of the output BAs.

The second experiment compares the size of the state space generated by NCSB-Lazy [7] and NCSB-MaxRank from Sec. 4.2 on 735 SDBAs (that are not IW) from LTL (328 BAs) and Automizer (407 BAs). We omit a comparison with the original NCSB [21] procedure, since NCSB-Lazy behaves overall better [7]. The results are in Fig. 3b and the bottom part of Table 1. Again, both the mean and the median are lower for NCSB-MaxRank. The scatter plot shows that the effect of the optimization is stronger when the generated state space is larger (for BAs where the output had ≥ 150 states, our optimization was never worse).

5.2 Comparison with Other Tools

In the second part of the experimental evaluation, we compared Ranker with other state-of-the-art tools for BA complementation. We measured how small output BAs we can obtain, therefore, we compared the number of states after reduction using `autfilt` (with the simplification level `--high`). The scatter plots

Table 2: Statistics for our experiments. The table compares the sizes of complement BAs obtained by RANKER and other approaches (after postprocessing). 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` : `Automizer`)". Approaches in GOAL are labelled with ⚽.

method	mean	median	wins	losses	timeouts
RANKER	38 (44: 9:67)	11 (18: 5:22)			158 (53: 0:105)
RANKER$_{\text{OLD}}$	30 (38:10:32)	12 (18: 6:22)	1554 (356: 650:548)	264 (142: 69:53)	458 (259: 7:192)
PITERMAN ⚽	43 (56:12:38)	14 (19: 8:24)	2881 (1279: 966:636)	392 (263: 68:61)	309 (12: 4:293)
SAFRA ⚽	49 (60:17:56)	15 (18:10:24)	3109 (1348:1117:644)	274 (229: 31:14)	599 (160:30:409)
SPOT	46 (57: 8:66)	11 (18: 5:18)	1347 (935: 339:73)	1057 (327:343:387)	73 (13: 0:60)
FRIBOURG ⚽	49 (68: 8:27)	11 (18: 6:19)	2223 (1177: 503:543)	586 (245:207:134)	399 (93: 2:304)
LTL2DSTAR	44 (56:12:47)	14 (19: 7:24)	2794 (1297: 924:573)	448 (283: 88:77)	288 (130:13:145)
SEMINATOR 2	46 (58: 8:64)	11 (17: 5:21)	1626 (1297: 291:38)	1113 (286:398:429)	419 (368: 1:50)
ROLL	18 (15:11:54)	9 (8: 8:28)	6050 (3824:1551:675)	620 (369:125:126)	1893 (1595: 8:290)

in Fig. 4 compare the numbers of states of automata generated by RANKER, RANKER$_{\text{OLD}}$, and SPOT. Summarizing statistics are given in Table 2. The backoff strategy in RANKER was applied in 278 (264:1:13) cases.

First, observe that RANKER significantly outperforms RANKER$_{\text{OLD}}$, especially in the much lower number of timeouts, which decreased by 65 % (moreover, 66 of the 158 timeouts were due to the timeout of `autfilt` in postprocessing). The higher mean of RANKER compared to RANKER$_{\text{OLD}}$ is also caused by less timeouts). From Table 2, we can also see that RANKER has the smallest mean and median (except ROLL and RANKER$_{\text{OLD}}$, but they have a much higher number of timeouts). RANKER has also the second lowest number of timeouts (SPOT has the lowest). If we look at the number of **wins** and **loses**, we can see that RANKER in majority of cases produces a strictly smaller automaton compared to other tools. In Table 3, see that the run time of RANKER is comparable to the run times of other tools (much better than GOAL and ROLL, comparable with SEMINATOR 2, and a bit worse than SPOT and LTL2DSTAR).

Table 3: Run times of the tools [s] given as "`all` (`random` : `LTL` : `Automizer`)"

method	mean	median
RANKER	3.72 (4.34:0.45:7.30)	0.05 (0.10:0.04:0.08)
RANKER$_{\text{OLD}}$	4.62 (5.33:0.72:9.69)	0.07 (0.19:0.03:0.15)
PITERMAN ⚽	8.06 (6.07:5.95:28.38)	5.12 (4.96:5.08:8.68)
SAFRA ⚽	11.58 (10.41:6.51:38.65)	5.41 (5.32:5.26:9.02)
SPOT	0.64 (0.57:0.02:2.28)	0.02 (0.02:0.01:0.02)
FRIBOURG ⚽	13.13 (14.14:6.06:23.88)	5.69 (6.82:4.92:6.57)
LTL2DSTAR	2.1 (2.25:0.34:5.15)	0.02 (0.02:0.01:0.05)
SEMINATOR 2	4.16 (6.33:0.03:1.88)	0.03 (0.08:0.01:0.03)
ROLL	23.65 (29.82:3.88:49.02)	3.34 (6.19:1.71:17.14)

Acknowledgements. We thank the anonymous reviewers for their useful remarks that helped us improve the quality of the paper, the artifact evaluation committee for their thorough testing of the artifact, and Alexandre Duret-Lutz for useful feedback on an earlier version of the paper. This work was supported by the Czech Ministry of Education, Youth and Sports project LL1908 of the ERC.CZ programme, the Czech Science Foundation project 20-07487S, and the FIT BUT internal project FIT-S-20-6427.

References

1. Büchi, J.R.: 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. Havlena, V., Lengál, O., Šmahlíková, B.: Deciding S1S: Down the rabbit hole and through the looking glass. In: Proceedings of NETYS'21. Number 12754 in LNCS, Springer (2021) 215–222
3. Sistla, A.P., Vardi, M.Y., Wolper, P.: The Complementation Problem for Büchi Automata with Applications to Temporal Logic. Theoretical Computer Science **49**(2-3) (1987) 217–237
4. Oei, R., Ma, D., Schulz, C., Hieronymi, P.: Pecan: An automated theorem prover for automatic sequences using Büchi automata. CoRR **abs/2102.01727** (2021) https://arxiv.org/abs/2102.01727.
5. Fogarty, S., Vardi, M.Y.: Büchi complementation and size-change termination. In: Proceedings of TACAS'09, Springer (2009) 16–30
6. Heizmann, M., Hoenicke, J., Podelski, A.: Termination analysis by learning terminating programs. In: Proceedings of CAV'14, Springer (2014) 797–813
7. Chen, Y., Heizmann, M., Lengál, O., Li, Y., Tsai, M., Turrini, A., Zhang, L.: Advanced automata-based algorithms for program termination checking. In: Proceedings of PLDI'18, ACM (2018) 135–150
8. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: Proceedings of LICS'86, IEEE (1986) 332–344
9. Safra, S.: On the complexity of ω-automata. In: Proceedings of FOCS'88, IEEE (1988) 319–327
10. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proceedings of LICS'06, IEEE (2006) 255–264
11. Redziejowski, R.R.: An improved construction of deterministic omega-automaton using derivatives. Fundam. Informaticae **119**(3-4) (2012) 393–406
12. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Log. **2**(3) (2001) 408–429
13. Friedgut, E., Kupferman, O., Vardi, M.: Büchi complementation made tighter. International Journal of Foundations of Computer Science **17** (2006) 851–868
14. Schewe, S.: Büchi complementation made tight. In Albers, S., Marion, J., eds.: Proceedings of STACS'09. Volume 3 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009) 661–672
15. Breuers, S., Löding, C., Olschewski, J.: Improved Ramsey-based Büchi complementation. In: Proceedings of FOSSACS'12, Springer (2012) 150–164
16. Allred, J.D., Ultes-Nitsche, U.: A simple and optimal complementation algorithm for Büchi automata. In: Proceedings of LICS'18, IEEE (2018) 46–55
17. Yan, Q.: Lower bounds for complementation of ω-automata via the full automata technique. In: Proceedings of ICALP'06, Springer (2006) 589–600
18. Havlena, V., Lengál, O.: Reducing (To) the Ranks: Efficient Rank-Based Büchi Automata Complementation. In: Proceedings of CONCUR'21. Volume 203 of LIPIcs., Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2021) 2:1–2:19
19. Havlena, V., Lengál, O., Šmahlíková, B.: Sky is not the limit: Tighter rank bounds for elevator automata in Büchi automata complementation. In: Proceedings of TACAS'22. Volume 13244 of LNCS., Springer (2022) 118–136
20. Chen, Y., Havlena, V., Lengál, O.: Simulations in rank-based Büchi automata complementation. In: Proceedings of APLAS'19. Volume 11893 of LNCS., Springer (2019) 447–467

21. Blahoudek, F., Heizmann, M., Schewe, S., Strejček, J., Tsai, M.: Complementing semi-deterministic Büchi automata. In: Proceedings of TACAS'16. Volume 9636 of LNCS., Springer (2016) 770–787
22. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theoretical Computer Science **32**(3) (1984) 321–330
23. Boigelot, B., Jodogne, S., Wolper, P.: On the use of weak automata for deciding linear arithmetic with integer and real variables. In: Proceedings of IJCAR'01. Volume 2083 of LNCS., Springer (2001) 611–625
24. Havlena, V., Lengál, O., Šmahlíková, B.: Ranker (2022) https://github.com/vhavlena/ranker.
25. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J., Müller, D., Parker, D., Strejček, J.: The Hanoi omega-automata format. In: Proceedings of CAV'15. Volume 9206 of LNCS., Springer (2015) 479–486
26. Abdulla, P.A., Chen, Y., Clemente, L., Holík, L., Hong, C.D., Mayr, R., Vojnar, T.: Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In: Proceedings of CAV'10, Springer (2010) 132–147
27. Mayr, R., Clemente, L.: Advanced automata minimization. In: Proceedings of POPL'13. (2013) 63–74
28. Havlena, V., Lengál, O., Šmahlíková, B.: Complementing Büchi automata with Ranker (technical report). CoRR **abs/2206.01946** (2021) https://arxiv.org/abs/2206.01946.
29. Glabbeek, R., Ploeger, B.: Five determinisation algorithms. In: Proceedings of CIAA'08, Springer (2008) 161–170
30. Etessami, K.: A hierarchy of polynomial-time computable simulations for automata. In: Proceedings of CONCUR'02, Springer (2002) 131–144
31. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Proceedings of ATVA'16, Springer (2016) 122–129
32. Löding, C., Pirogov, A.: New optimizations and heuristics for determinization of Büchi automata. In: Proceedings of ATVA'19. Volume 11781 of LNCS., Springer (2019) 317–333
33. Tsai, M.H., Tsay, Y.K., Hwang, Y.S.: GOAL for games, omega-automata, and logics. In: Proceedings of CAV'13, Springer (2013) 883–889
34. Blahoudek, F., Duret-Lutz, A., Strejček, J.: Seminator 2 can complement generalized Büchi automata via improved semi-determinization. In: Proceedings of CAV'20. Volume 12225 of LNCS., Springer (2020) 15–27
35. Klein, J., Baier, C.: On-the-fly stuttering in the construction of deterministic ω-automata. In: Proceedings of CIAA'07. Volume 4783 of LNCS., Springer (2007) 51–61
36. Li, Y., Sun, X., Turrini, A., Chen, Y., Xu, J.: ROLL 1.0: ω-regular language learning library. In: Proceedings of TACAS'19. Volume 11427 of LNCS., Springer (2019) 365–371
37. Havlena, V., Lengál, O., Šmahlíková, B.: Artifact for the CAV'22 submission "Complementing Büchi Automata with Ranker" (May 2022) https://doi.org/10.5281/zenodo.6558229.
38. Tsai, M.H., Fogarty, S., Vardi, M.Y., Tsay, Y.K.: State of Büchi complementation. In: Proceedings of CIAA'11. Volume 6482 of LNCS., Springer (2011) 261–271
39. Tabakov, D., Vardi, M.Y.: Experimental evaluation of classical automata constructions. In: Proceedings of LPAR'05, Springer (2005) 396–411
40. Lengál, O.: Automata benchmarks repository (2022) https://github.com/ondrik/automata-benchmarks/tree/master/omega.