Reducing (to) the Ranks:
Efficient Rank-based Büchi Automata Complementation

Vojtěch Havlena ORCID
Faculty of Information Technology, Brno University of Technology, Czech Republic

Ondřej Lengál ORCID
Faculty of Information Technology, Brno University of Technology, Czech Republic

Abstract

This paper provides several optimizations of the rank-based approach for complementing Büchi automata. We start with Schewe's theoretically optimal construction and develop a set of techniques for pruning its state space that are key to obtaining small complement automata in practice. In particular, the reductions (except one) have the property that they preserve (at least some) so-called *super-tight runs*, which are runs whose ranking is *as tight as possible*. Our evaluation on a large benchmark shows that the optimizations indeed significantly help the rank-based approach and that, in a large number of cases, the obtained complement is the smallest from those produced by state-of-the-art tools for Büchi complementation.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Büchi automata, rank-based complementation, super-tight runs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.9

Related Version *Full Version*: `https://arxiv.org/abs/2010.07834`

1 Introduction

Büchi automata (BA) complementation remains an intensively studied problem since 1962, when Büchi introduced the automata model over infinite words as a foundation for a decision procedure of a fragment of a second-order arithmetic [7]. Since then, efficient BA complementation became an important task from both theoretical and practical side. It is a crucial operation in some approaches for termination analysis of programs [12, 18, 9] as well as in decision procedures concerning reasoning about programs and computer systems, such as S1S [7] or the temporal logics ETL and QPTL [34].

Büchi launched a hunt for an optimal and efficient complementation technique with his doubly exponential complementation approach [7]. A couple of years later, Safra proposed a complementation via deterministic Rabin automata with an $n^{\mathcal{O}(n)}$ *upper bound* of the size of the complement. Simultaneously with finding an efficient complementation algorithm, another search for the theoretical *lower bound* was under way. Michel showed in [28] that a lower bound of the size of a complement BA is $n!$ (approx. $(0.36n)^n$). This result was further refined by Yan to $(0.76n)^n$ in [40]. From the theoretical point of view, it seemed that the problem was already solved since Safra's construction asymptotically matched the lower bound. From the practical point of view, however, a factor in the exponent has a great impact on the size of the complemented automaton (and, consequently, also affects the performance of real-world applications). This gap became a topic of many works [22, 13, 39, 19, 41]. The efforts finally led to the construction of Schewe in [33] producing complement BAs whose sizes match the lower bound modulo a $\mathcal{O}(n^2)$ polynomial factor.

Schewe's construction stores in a macrostate partial information about all runs over some word in an input BA. In order to track the information about all runs, a macrostate contains a set of states representing a single level in a run DAG of some word with a number assigned to each state representing its rank. The number of macrostates (and hence the size of the complement) is combinatorially related to the maximum rank that occurs in macrostates.

Although the construction of Schewe is worst-case optimal, it may in practice still generate a lot of states that are not necessary. In this work, we propose novel optimizations that (among others) reduce this maximum considered rank. We build on the novel notion of a *super-tight run*, i.e., a run in the complement that uses as small ranks as possible. The macrostates not occurring in some super-tight run can be safely removed from the automaton. Further, based on reasoning about super-tight runs, we are able to reduce the maximum rank within a macrostate. In particular, we reduce the maximum considered ranking using a reasoning about the deterministic support of an input automaton or by a relation based on direct simulation implying rank ordering computed *a priori* from the input automaton. The developed optimizations give, to the best of our knowledge, **the most competitive BA complementation procedure**, as witnessed by our experimental evaluation.

These optimizations require some additional computational cost, but from the perspective of BA complementation, their cost is still negligible and, as we show in our experimental evaluation, their effect on the size of the output is often profound, in many cases by one or more orders of magnitude. Rank-based complementation with our optimizations is now competitive with other approaches, in a large number of cases (21 %) obtaining a strictly smaller complement than *any other existing tool* and in the majority of cases (63 %) obtaining an automaton at least as small as the smallest automaton provided by any other tool.

2 Preliminaries

We fix a finite nonempty alphabet Σ and the first infinite ordinal $\omega = \{0, 1, \ldots\}$. For $n \in \omega$, by $[n]$ we denote the set $\{0, \ldots, n\}$. An (infinite) word α 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 also represent α as an infinite sequence $\alpha = \alpha_0\alpha_1 \ldots$ The suffix $\alpha_i\alpha_{i+1}\ldots$ of α is denoted by $\alpha_{i:\omega}$. We use Σ^ω to denote the set of all infinite words over Σ. Furthermore, for a total function $f\colon X \to Y$ and a partial function $h\colon X \rightharpoonup Y$, we use $f \lhd h$ to denote the total function $g\colon X \to Y$ defined as $g(x) = h(x)$ when $h(x)$ is defined and $g(x) = f(x)$ otherwise. Moreover, we use $\mathrm{img}(f)$ to denote the *image* of f, i.e., $\mathrm{img}(f) = \{f(x) \in Y \mid x \in X\}$ and for a set $C \subseteq X$ we use $f_{|C}$ to denote the *restriction* of f to C, i.e., $f_{|C} = f \cap (C \times Y)$.

Büchi automata. A (nondeterministic) *Büchi automaton* (BA) over Σ is a quadruple $\mathcal{A} = (Q, \delta, I, F)$ where Q is a finite set of *states*, δ is a *transition function* $\delta\colon Q \times \Sigma \to 2^Q$, and $I, F \subseteq Q$ are the sets of *initial* and *accepting* states respectively. 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 $\delta(P, a) = \bigcup_{p\in P} \delta(p, a)$. We use $\delta^{-1}(q, a)$ to denote the set $\{s \in Q \mid s \xrightarrow{a} q \in \delta\}$. For a set of states S we define *reachability* from S as $\mathit{reach}_\delta(S) = \mu Z.\, S \cup \bigcup_{a\in\Sigma} \delta(Z, a)$. 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 $\inf(\rho)$ denote the states occurring in ρ infinitely often. We say that ρ is *accepting* iff $\inf(\rho) \cap F \neq \emptyset$. A word α is accepted by $\mathcal{A}$ from a state $q \in Q$ if there is an accepting run ρ of $\mathcal{A}$ 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 $\mathcal{L}(\mathcal{A}) = \mathcal{L}_\mathcal{A}(I)$. For a pair of states p and q in $\mathcal{A}$, we use $p \subseteq_\mathcal{L} q$ to denote $\mathcal{L}_\mathcal{A}(p) \subseteq \mathcal{L}_\mathcal{A}(q)$. $\mathcal{A}$ is complete iff for every state q and symbol a, it holds that $\delta(q, a) \neq \emptyset$. In this paper, we fix a BA $\mathcal{A} = (Q, \delta, I, F)$.

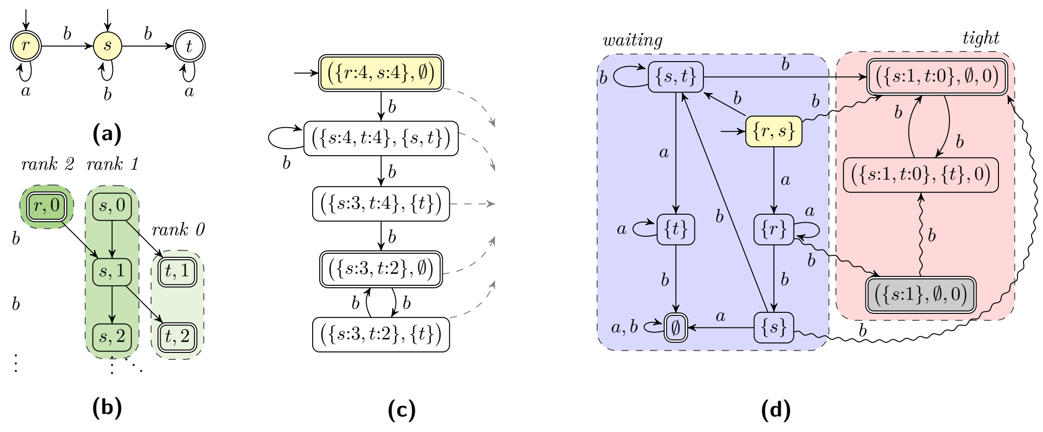


Figure 1 (a) $\mathcal{A}_{ex}$. (b) The run DAG of $\mathcal{A}_{ex}$ over b^ω. (c) A part of $\mathrm{KV}(\mathcal{A}_{ex})$. (d) Schewe($\mathcal{A}_{ex}$); we highlight the *waiting* and the *tight* parts. Delay (Section 4.1) will remove the 4 wobbly transitions and macrostate $(\{s{:}1\}, \emptyset, 0)$.

Simulation. The *(maximum) direct simulation* on $\mathcal{A}$ is the relation $\preceq_{di} \subseteq Q \times Q$ defined as the largest relation s.t. $p \preceq_{di} q$ implies (i) $p \in F \Rightarrow q \in F$ and (ii) $p \xrightarrow{a} p' \in \delta \Rightarrow \exists q' \in Q \colon q \xrightarrow{a} q' \in \delta \wedge p' \preceq_{di} q'$ for each $a \in \Sigma$. Note that $\preceq_{di}$ is a preorder and $\preceq_{di} \subseteq \subseteq_{\mathcal{L}}$ [27].

3 Complementing Büchi Automata

In this section we first describe the basic rank-based complementation algorithm proposed by Kupferman and Vardi in [22] and then its optimization presented by Schewe in [33]. After that, we present some results related to runs with the minimal ranking. Missing proofs for this and the following section can be found in [16].

3.1 Run DAGs

In this section, we recall the terminology from [33] (which is a minor modification of the terminology from [22]), which we use heavily in the paper. We fix the definition of the *run DAG* of $\mathcal{A}$ over a word α to be a DAG (directed acyclic graph) $\mathcal{G}_\alpha = (V, E)$ of vertices V and edges E where

- $V \subseteq Q \times \omega$ s.t. $(q, i) \in V$ iff there is a run ρ of $\mathcal{A}$ from I over α with $\rho_i = q$,
- $E \subseteq V \times V$ s.t. $((q, i), (q', i')) \in E$ iff $i' = i + 1$ and $q' \in \delta(q, \alpha_i)$.

Given $\mathcal{G}_\alpha$ as above, we will write $(p, i) \in \mathcal{G}_\alpha$ to denote that $(p, i) \in V$. We call (p, i) *accepting* if p is an accepting state. $\mathcal{G}_\alpha$ is *rejecting* if it contains no path with infinitely many accepting vertices. A vertex $v \in \mathcal{G}_\alpha$ is *finite* if the set of vertices reachable from v is finite, *infinite* if it is not finite, and *endangered* if v cannot reach an accepting vertex.

We assign ranks to vertices of run DAGs as follows: Let $\mathcal{G}_\alpha^0 = \mathcal{G}_\alpha$ and $j = 0$. Repeat the following steps until the fixpoint or for at most $2n + 1$ steps, where $n = |Q|$.

- Set $\mathit{rank}_\alpha(v) := j$ for all finite vertices v of $\mathcal{G}_\alpha^j$ and let $\mathcal{G}_\alpha^{j+1}$ be $\mathcal{G}_\alpha^j$ minus the vertices with the rank j.
- Set $\mathit{rank}_\alpha(v) := j + 1$ for all endangered vertices v of $\mathcal{G}_\alpha^{j+1}$ and let $\mathcal{G}_\alpha^{j+2}$ be $\mathcal{G}_\alpha^{j+1}$ minus the vertices with the rank $j + 1$.
- Set $j := j + 2$.

For all vertices v that have not been assigned a rank yet, we assign $rank_\alpha(v) := \omega$. See Figure 1a for an example BA $\mathcal{A}_{ex}$ and Figure 1b for the run DAG of $\mathcal{A}_{ex}$ over b^ω.

3.2 Basic Rank-Based Complementation

The intuition in rank-based complementation algorithms is that states in the complemented automaton $\mathcal{C}$ *track* all runs of the original automaton $\mathcal{A}$ on the given word and the possible *ranks* of each of the runs. Loosely speaking, an accepting run of a complement automaton $\mathcal{C}$ on a word $\alpha \notin \mathcal{L}(\mathcal{A})$ represents the run DAG of $\mathcal{A}$ over α (in the complement, each state in a *macrostate* is assigned a *rank*)[1].

The complementation procedure works with the notion of level rankings of states of $\mathcal{A}$, originally proposed in [22, 13]. For $n = |Q|$, a *(level) ranking* is a function $f \colon Q \to [2n]$ such that $\{f(q_f) \mid q_f \in F\} \subseteq \{0, 2, \ldots, 2n\}$, i.e., f assigns even ranks to accepting states of $\mathcal{A}$. We use $\mathcal{R}$ to denote the set of all rankings and $odd(f)$ to denote the set of states given an odd ranking by f, i.e. $odd(f) = \{q \in Q \mid f(q) \text{ is odd}\}$. For a ranking f, the *rank* of f is defined as $rank(f) = \max\{f(q) \mid q \in Q\}$. We use $f \leq f'$ iff for every state $q \in Q$ we have $f(q) \leq f'(q)$ and $f < f'$ iff $f \leq f'$ and there is a state $p \in Q$ with $f(p) < f'(p)$.

The simplest rank-based procedure, called KV, constructs the BA $\mathrm{KV}(\mathcal{A}) = (Q', \delta', I', F')$ whose components are defined as follows [22]:

- $Q' = 2^Q \times 2^Q \times \mathcal{R}$ is a set of *macrostates* denoted as (S, O, f),
- $I' = \{I\} \times \{\emptyset\} \times \mathcal{R}$,
- $(S', O', f') \in \delta'((S, O, f), a)$ iff
  - $S' = \delta(S, a)$,
  - for every $q \in S$ and $q' \in \delta(q, a)$ it holds that $f'(q') \leq f(q)$, and
  - $O' = \begin{cases} \delta(S, a) \setminus odd(f') \text{ if } O = \emptyset, \\ \delta(O, a) \setminus odd(f') \text{ otherwise, and} \end{cases}$
- $F' = 2^Q \times \{\emptyset\} \times \mathcal{R}$.

The macrostates (S, O, f) of $\mathrm{KV}(\mathcal{A})$ are composed of three components. The S component tracks all runs of $\mathcal{A}$ over the input word in the same way as determinization of an NFA. The O component, on the other hand, tracks all runs whose rank has been even since the last cut-point (a point where $O = \emptyset$). The last component, f, assigns every state in S a rank. Note that the f component is responsible for the nondeterminism of the complement (and also for the content of the O component). A run of $\mathrm{KV}(\mathcal{A})$ is accepting if it manages to empty the O component of states occurring on the run infinitely often. We often merge S and f components and use, e.g., $(\{r{:}4, s{:}4\}, \emptyset)$ to denote the macrostate $(\{r, s\}, \emptyset, \{r \mapsto 4, s \mapsto 4\})$ (we also omit ranks of states not in S). See Figure 1c for a part of $\mathrm{KV}(\mathcal{A}_{ex})$ that starts in $(\{r{:}4, s{:}4\}, \emptyset)$ and keeps ranks as high as possible (the whole automaton is prohibitively large to be shown here—the implementation of KV in GOAL [37] outputs a BA with 98 states). Note that in order to accept the word b^ω, the accepting run needs to nondeterministically decrease the rank of the successor of s (the transition $(\{s{:}4, t{:}4\}, \{s, t\}) \xrightarrow{b} (\{s{:}3, t{:}4\}, \{t\})$).

In the worst case, KV constructs a BA with approximately $(6n)^n$ states [22].

[1] This is not entirely true; there may be more accepting runs of $\mathcal{C}$ over α, with ranks assigned to states of $\mathcal{A}$ that are higher than the ranks in the run DAG. There will, however, be a *minimum* run of $\mathcal{C}$ that matches the run DAG (in the terminology of Section 3.4, such a run corresponds to a *super-tight run*).

3.3 Optimal Rank-Based Complementation

Friedgut, Kupferman, and Vardi observed in [13] that the KV construction generates macrostates with many rankings that are not strictly necessary in the loop part of the lasso for an accepting run on a word. Their optimization is based on composing the complement automaton from two parts: the first part (called by us the *waiting* part) just tracks all runs of $\mathcal{A}$ over the input word (in a similar manner as in a determinized NFA) and the second part (the *tight* part) in addition tracks the rank of each run in a similar manner as the KV construction, with the difference that the rankings are *tight*. For a set of states $S \subseteq Q$, we call f to be S-*tight* if (i) it has an odd rank r, (ii) $\{f(s) \mid s \in S\} \supseteq \{1, 3, \ldots, r\}$, and (iii) $\{f(q) \mid q \notin S\} = \{0\}$. A ranking is *tight* if it is Q-tight; we use $\mathcal{T}$ to denote the set of all tight rankings.

An optimal algorithm whose space complexity matches the theoretical lower bound $\mathcal{O}((0.76n)^n)$ was given by Schewe in [33, Section 3.1]. We denote this algorithm as Schewe. Apart from the optimization from [13], in Schewe, macrostates of the tight part contain one additional component, i.e., a macrostate has the form (S, O, f, i), where the last component $i \in \{0, 2, \ldots, 2n-2\}$, for $n = |Q|$, denotes the rank of states that are in O. Then, at a cut-point (when O is being reset), O is not filled with all states having an even rank, but only those whose rank is i (at every cut-point, i changes to $i+2$ modulo the rank of f).

Formally, $\text{Schewe}(\mathcal{A}) = (Q', \delta', I', F')$ is constructed as follows:

- $Q' = Q_1 \cup Q_2$ where
  - $Q_1 = 2^Q$ and
  - $Q_2 = \{(S, O, f, i) \in 2^Q \times 2^Q \times \mathcal{T} \times \{0, 2, \ldots, 2n-2\} \mid f \text{ is } S\text{-tight}, O \subseteq S \cap f^{-1}(i)\}$,
- $I' = \{I\}$,
- $\delta' = \delta_1 \cup \delta_2 \cup \delta_3$ where
  - $\delta_1 : Q_1 \times \Sigma \to 2^{Q_1}$ such that $\delta_1(S, a) = \{\delta(S, a)\}$,
  - $\delta_2 : Q_1 \times \Sigma \to 2^{Q_2}$ such that $\delta_2(S, a) = \{(S', \emptyset, f, 0) \mid S' = \delta(S, a), f \text{ is } S'\text{-tight}\}$, and
  - $\delta_3 : Q_2 \times \Sigma \to 2^{Q_2}$ such that $(S', O', f', i') \in \delta_3((S, O, f, i), a)$ iff
    - $S' = \delta(S, a)$,
    - for every $q \in S$ and $q' \in \delta(q, a)$ it holds that $f'(q') \leq f(q)$,
    - $\mathit{rank}(f) = \mathit{rank}(f')$,
    - and
      - $i' = (i+2) \mod (\mathit{rank}(f') + 1)$ and $O' = f'^{-1}(i')$ if $O = \emptyset$ or
      - $i' = i$ and $O' = \delta(O, a) \cap f'^{-1}(i)$ if $O \neq \emptyset$, and
- $F' = \{\emptyset\} \cup ((2^Q \times \{\emptyset\} \times \mathcal{T} \times \omega) \cap Q_2)$.

We call the part of $\text{Schewe}(\mathcal{A})$ with the states in Q_1 the *waiting* part and the part with the states in Q_2 the *tight* part (an accepting run in $\text{Schewe}(\mathcal{A})$ simulates the run DAG of $\mathcal{A}$ over a word w by *waiting* in Q_1 until it can generate *tight rankings* only; then it moves to Q_2). See Figure 1d for $\text{Schewe}(\mathcal{A}_{ex})$. Note that in order to accept the word b^ω, the accepting run needs to nondeterministically move from the waiting to the tight part.

▶ **Theorem 1. ([33, Corollary 3.3])** *Let $\mathcal{B} = \text{Schewe}(\mathcal{A})$. Then $\mathcal{L}(\mathcal{B}) = \overline{\mathcal{L}(\mathcal{A})}$.*

In the following, we assume that $\text{Schewe}(\mathcal{A})$ contains only the states and transitions reachable from I'. We use Schewe as the base algorithm in the rest of the paper.

3.4 Super-Tight Runs

Let $\mathcal{B} = \text{Schewe}(\mathcal{A})$. Each accepting run of $\mathcal{B}$ on $\alpha \in \mathcal{L}(\mathcal{B})$ is *tight*, i.e., the rankings of macrostates it traverses in Q_2 are tight (this follows from the definition of Q_2). In this section, we show that there exists a *super-tight run* of $\mathcal{B}$ on α, which is, intuitively, a run that uses as little ranks as possible. Our optimizations in Section 4 are based on preserving super-tight runs of $\mathcal{B}$.

Let $\rho = S_0 \dots S_m(S_{m+1}, O_{m+1}, f_{m+1}, i_{m+1})(S_{m+2}, O_{m+2}, f_{m+2}, i_{m+2}) \dots$ be an accepting run of $\mathcal{B}$ over a word $\alpha \in \Sigma^\omega$. Given a macrostate (S_k, O_k, f_k, i_k) for $k > m$, we define its *rank* as $\mathit{rank}((S_k, O_k, f_k, i_k)) = \mathit{rank}(f_k)$. Further, we define the *rank of the run* ρ as $\mathit{rank}(\rho) = \min\{\mathit{rank}((S_k, O_k, f_k, i_k)) \mid k > m\}$. Let $\mathcal{G}_\alpha$ be the run DAG of $\mathcal{A}$ over α and rank_α be the ranking of vertices in $\mathcal{G}_\alpha$. We say that the run ρ is *super-tight* if for all $k > m$ and all $q \in S_k$, it holds that $f_k(q) = \mathit{rank}_\alpha(q, k)$. Intuitively, super-tight runs correspond to runs whose ranking faithfully copies the ranks assigned in $\mathcal{G}_\alpha$ (from some position m corresponding to the transition from the waiting to the tight part of $\mathcal{B}$).

▶ **Lemma 2.** *Let $\alpha \in \mathcal{L}(\mathcal{B})$. Then there is a super-tight accepting run ρ of $\mathcal{B}$ on α.*

Let $\rho = S_0 \dots S_m(S_{m+1}, O_{m+1}, f_{m+1}, i_{m+1})(S_{m+2}, O_{m+2}, f_{m+2}, i_{m+2}) \dots$ be a run and consider a macrostate (S_k, O_k, f_k, i_k) for $k > m$. We call a set $C_k \subseteq S_k$ a *tight core of a ranking* f_k if $f_k(C_k) = \{1, 3, \dots, \mathit{rank}(f_k)\}$ and $f_k|_{C_k}$ is injective (i.e., every state in the tight core has a unique odd rank). Moreover, C_k is a *tight core of a macrostate* (S_k, O_k, f_k, i_k) if it is a tight core of f_k. We say that an infinite sequence $\tau = C_{m+1}C_{m+2}\dots$ is a *trunk* of run ρ if for all $k > m$ it holds that C_k is a tight core of $\rho(k)$ and there is a bijection $\theta : C_k \to C_{k+1}$ s.t. if $\theta(q_k) = q_{k+1}$ then $q_{k+1} \in \delta(q_k, \alpha_k)$. We will, in particular, be interested in trunks of super-tight runs. In these runs, a trunk (there can be several) *represents runs of $\mathcal{A}$ that keep the super-tight ranks of ρ*. The following lemma shows that every state in any tight core in a trunk of such a run has at least one successor with the same rank.

▶ **Lemma 3.** *Let $\rho = S_0 \dots S_m(S_{m+1}, O_{m+1}, f_{m+1}, i_{m+1}) \dots$ be an accepting super-tight run of $\mathcal{B}$ on α. Then there is a trunk $\tau = C_{m+1}C_{m+2}\dots$ of ρ and, moreover, for every $k > m$ and all states $q_k \in C_k$, it holds that there is a state $q_{k+1} \in C_{k+1}$ such that $f_k(q_k) = f_{k+1}(q_{k+1})$.*

4 Optimized Complement Construction

In this section, we introduce our optimizations of Schewe that are key to producing small complement automata in practice.

4.1 Delaying the Transition from Waiting to Tight

Our first optimization of the construction of the complement automaton reduces the number of nondeterministic transitions between the *waiting* and the *tight* part. This optimization is inspired by the idea of *partial order reduction* in model checking [14, 38, 29]. In particular, since in each state of the waiting part, it is possible to move to the tight part, we can arbitrarily delay such a transition (but need to take it eventually) and, therefore, significantly reduce the number of transitions (and, as our experiments later show, also significantly reduce the number of reachable states in Q_2).

Speaking in the terms of partial order reduction, when constructing the waiting part of the complement BA, given a macrostate $S \in Q_1$ and a symbol $a \in \Sigma$, we can set $\theta_2 \subseteq \delta_2$ such that $\theta_2(S, a) := \emptyset$ if the *cycle closing condition* holds and $\theta_2(S, a) := \delta_2(S, a)$ otherwise. Informally, the *cycle closing condition* (often denoted as **C3**) holds for S and a if the successor of S over a in the waiting part does not close a cycle where the transition to the tight part would be infinitely often delayed. Practically, it means that when constructing Q_1, we need

Algorithm 1 The DELAY construction

```
    Input: A Büchi automaton $\mathcal{A} = (Q, I, \delta, F)$
    Output: A Büchi automaton $\mathcal{C}$ s.t. $\mathcal{L}(\mathcal{C}) = \overline{\mathcal{L}(\mathcal{A})}$
 1  $\mathcal{S} \leftarrow \{I\}$, $\mathcal{Q}_1 \leftarrow \{I\}$, $\theta_2 \leftarrow \emptyset$, $(\cdot, \delta_1 \cup \delta_2 \cup \delta_3, I', F') \leftarrow$ SCHEWE($\mathcal{A}$);
 2  while $\mathcal{S} \neq \emptyset$ do
 3      Take a waiting-part macrostate $R \subseteq Q$ from $\mathcal{S}$;
 4      foreach $a \in \Sigma$ do
 5          if $\exists T \in \delta_1(R, a)$ s.t. $R \xrightarrow{a} T$ closes a cycle in $Q_1$ then
 6              $\theta_2 \leftarrow \theta_2 \cup \{R \xrightarrow{a} U \mid U \in \delta_2(R, a)\}$;
 7          foreach $T \in \delta_1(R, a)$ s.t. $T \notin Q_1$ do
 8              $\mathcal{S} \leftarrow \mathcal{S} \cup \{T\}$;
 9              $Q_1 \leftarrow \mathcal{Q} \cup \{T\}$;
10  $Q_2 \leftarrow reach_{\delta_3}(\mathrm{img}(\theta_2))$;
11  return $\mathcal{C} = (Q_1 \cup Q_2, \delta_1 \cup \theta_2 \cup \delta_3, I', F' \cap Q_2)$;
```

to check whether successors of a macrostate close a cycle in the so-far generated part of Q_1. We give the construction in Algorithm 1 and refer to it as DELAY. Using this optimisation on the example in Figure 1d, we would remove the b-transitions from $\{r, s\}$ and $\{s\}$ to the macrostate $(\{s{:}1, t{:}0\}, \emptyset, 0)$ and also the macrostate $(\{s{:}1\}, \emptyset, 0)$ (including the transitions incident with it).

▶ **Lemma 4.** *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\text{DELAY}(\mathcal{A})) = \mathcal{L}(\text{SCHEWE}(\mathcal{A}))$. Moreover, for every accepting super-tight run of SCHEWE($\mathcal{A}$) on α, there is an accepting super-tight run of DELAY($\mathcal{A}$) on α.*

Since DELAY does not affect the rankings in the macrostates and only delays the transition from the waiting to the tight part, we can freely use it as the base algorithm instead of SCHEWE in all following optimizations.

4.2 Successor Rankings

Our next optimization is used to reduce the maximum considered ranking of a macrostate in the tight part of $\mathcal{B} = \text{SCHEWE}(\mathcal{A})$. For a given macrostate, the number of tight rankings that can occur within the macrostate rises combinatorially with the macrostate's maximum rank (in particular, the number of tight rankings for a given set of states corresponds to the Stirling number of the second kind of the maximum rank [13]). It is hence desirable to reduce the maximum considered rank as much as possible.

The idea of our optimization called SUCCRANK is the following. Suppose we have a macrostate (S, O, f, i) from the tight part of $\mathcal{B}$. Further, assume that the maximum number of non-accepting states in the S-component of a macrostate that is infinitely often reachable from (S, O, f, i) is $\lceil S \rceil$. Then, we know that a super-tight accepting run that goes through (S, O, f, i) will never need a rank higher than $2\lceil S \rceil - 1$ (any accepting state will be assigned an even rank, so we can omit them). Therefore, if the rank of f is higher than $2\lceil S \rceil - 1$, we can safely discard (S, O, f, i) (since there will be a super-tight accepting run that goes over (S, O', f', i') with $f' < f$). This part of the optimization is called *coarse*.

Moreover, let $q \in S$ and let $\lfloor \{q\} \rfloor$ be the smallest size of a set of states (again without accepting states) reachable from q over some (infinite) word infinitely often. Then, we know that those states will have a rank bounded by the rank of $f(q)$, so there are only (at most)

$\lceil S \rceil - \lfloor \{q\} \rfloor$ states whose rank can be higher than $f(q)$. Therefore, the rank of f, which is tight, can be at most $f(q) + 2(\lceil S \rceil - \lfloor \{q\} \rfloor)$. We call this part of the optimization *fine*.

We now formalize the intuition. Let us fix a BA $\mathcal{A} = (Q, \delta, I, F)$. Then, let us consider a BA $R_{\mathcal{A}} = (2^Q, \delta_R, \emptyset, \emptyset)$, with $\delta_R = \{R \xrightarrow{a} S \mid S = \delta(R, a)\}$, which is tracking *reachability* between set of all states of $\mathcal{A}$ (we only focus on its structure and not the language). Note that $R_{\mathcal{A}}$ is deterministic and complete. Further, given $S \subseteq Q$, let us use $SCC(S) \subseteq 2^{2^Q}$ to denote the set of all *strongly connected components reachable from* S *in* $R_{\mathcal{A}}$. We will use *inf-reach*(S) to denote the set of states $\bigcup SCC(S)$, i.e., the set of states such that there is an infinite path in $R_{\mathcal{A}}$ starting in S that passes through a given state infinitely many times. We define the maximum and minimum sizes of macrostates reachable infinitely often from S:

$$\lceil S \rceil = \max\{|R \setminus F| : R \in \textit{inf-reach}(S)\} \qquad \text{and} \qquad \lfloor S \rfloor = \min\{|R \setminus F| : R \in \textit{inf-reach}(S)\}.$$

For a macrostate (S, O, f, i), we define $\varphi_{coarse}((S, O, f, i)) \overset{\text{def}}{\equiv} \textit{rank}(f) \leq 2\lceil S \rceil - 1$. If (S, O, f, i) does not satisfy φ_{coarse}, we can omit it from the output of Schewe$(\mathcal{A})$ (as allowed by Lemma 5). See Figure 2a for an example of such a macrostates. For instance, macrostate $(\{r{:}3, t{:}1\}, \emptyset, 0)$ can be removed since its rank is 3 and $\lceil \{r, t\} \rceil = 1$, so $3 \not\leq 2\lceil \{r, t\} \rceil - 1$.

Moreover, we also define the condition

$$\varphi_{\textit{fine}}((S, O, f, i)) \overset{\text{def}}{\equiv} \textit{rank}(f) \leq \min\{f(q) + 2(\lceil S \rceil - \lfloor \{q\} \rfloor) \mid q \in S\}. \tag{1}$$

Again, we can omit (S, O, f, i) if it does not satisfy $\varphi_{\textit{fine}}$. See Figure 2b for an example of such a macrostate. Note that the rank of $(\{r{:}1, s{:}5, t{:}3\}, \emptyset, 0)$ is 5, $\lceil \{r, s, t\} \rceil = 3$ and $\lfloor \{r\} \rfloor = 2$, $\lfloor \{s\} \rfloor = 1$, $\lfloor \{t\} \rfloor = 0$. Then, $\min\{f(r) + 2(3-2), f(s) + 2(3-1), f(t) + 2(3-0)\} = \min\{1+2, 5+4, 3+6\} = 3$, so the macrostate does not satisfy $\varphi_{\textit{fine}}$ and can be removed.

We emphasize that φ_{coarse} and $\varphi_{\textit{fine}}$ are incomparable. For example, the macrostates removed due to φ_{coarse} in Figure 2a satisfy $\varphi_{\textit{fine}}$ (since, e.g., $3 \leq \min\{3+2(1-1), 1+2(1-0)\}$) and the macrostate removed due to $\varphi_{\textit{fine}}$ in Figure 2b satisfies φ_{coarse} (since $5 \leq 2 \cdot 3 - 1$).

Putting the conditions together, we define the predicate

$$\textsc{SuccRank}((S, O, f, i)) \overset{\text{def}}{\equiv} \varphi_{coarse}((S, O, f, i)) \wedge \varphi_{\textit{fine}}((S, O, f, i)). \tag{2}$$

We abuse notation and use SuccRank$(\mathcal{A})$ to denote the output of Schewe$(\mathcal{A}) = (Q', \delta', I', F')$ where the states from the tight part of Q' are restricted to those that satisfy SuccRank.

▶ **Lemma 5.** *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\textsc{SuccRank}(\mathcal{A})) = \mathcal{L}(\textsc{Schewe}(\mathcal{A}))$.*

4.3 Rank Simulation

The next optimization RankSim is a modification of optimization Purge$_{di}$ from [8]. Intuitively, Purge$_{di}$ is based on the fact that if a state p is directly simulated by a state r, i.e., $p \preceq_{di} r$, then any macrostate (S, O, f, i) where $f(p) > f(r)$ can be safely removed (intuitively, any run from p can be simulated by a run from r, where the run from r may contain more accepting states and so needs to decrease its rank more times). Purge$_{di}$ is compatible with Schewe but, unfortunately, it is incompatible with the MaxRank construction (one of our further optimizations introduced in Section 4.5) since in MaxRank, several runs are represented by one *maximal* run (w.r.t. the ranks) and removing such a run would also remove the smaller runs. We, however, change the condition and obtain a new reduction, which is incomparable with Purge$_{di}$ but compatible with MaxRank.

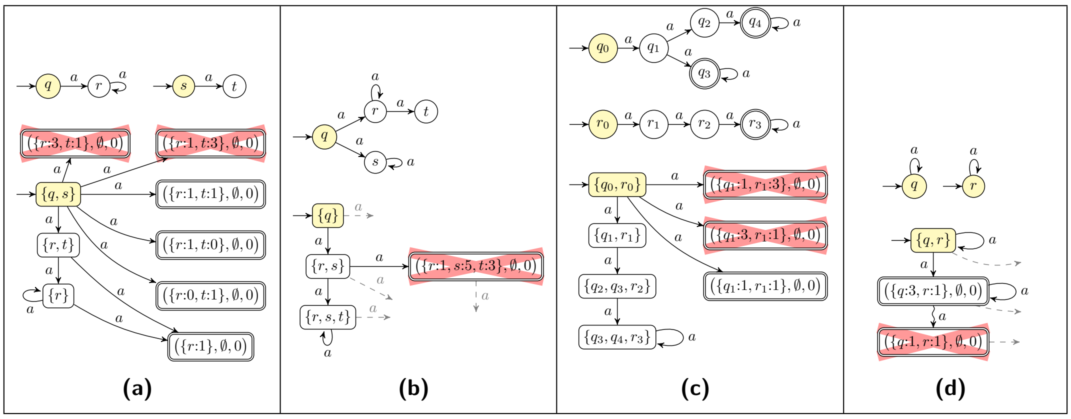


Figure 2 (a) Illustration of SuccRank reduction (φ_{coarse}), focusing on the transitions from the waiting to the tight part. (b) Illustration of SuccRank reduction (φ_{fine}), focusing on one particular macrostate. (c) Illustration of RankSim$'$. (d) Illustration of RankRestr.

Consider the following relation of *odd-rank simulation* on Q defined such that $p \preceq_{ors} r$ iff

$$\forall \alpha \in \Sigma^\omega, \forall i \geq 0\colon (\mathit{rank}_\alpha(p,i) \text{ is odd} \wedge \mathit{rank}_\alpha(r,i) \text{ is odd}) \Rightarrow \mathit{rank}_\alpha(p,i) \leq \mathit{rank}_\alpha(r,i). \quad (3)$$

Intuitively, if $p \preceq_{ors} r$ holds, then in any super-tight run and a macrostate (S, O, f, i) in such a run, if $p, r \in S$ and both $f(p)$ and $f(r)$ are odd, then it needs to hold that $f(p) \leq f(r)$. Such a reasoning can also be applied transitively ($\preceq_{ors}$ is by itself not transitive): if, in addition, $t \in S$, the rank $f(t)$ is odd, and $r \preceq_{ors} t$, then it also needs to hold that $f(p) \leq f(t)$.

Formally, given a ranking f, let $\preceq^f_{ors}$ be a modification of $\preceq_{ors}$ defined as

$$p \preceq^f_{ors} r \overset{\text{def}}{\equiv} f(p) \text{ is odd} \wedge f(r) \text{ is odd} \wedge p \preceq_{ors} r \quad (4)$$

and $\preceq^{fT}_{ors}$ be its transitive closure. We use $\preceq^{fT}_{ors}$ to define the following condition:

$$\text{RankSim}((S, O, f, i)) \overset{\text{def}}{\equiv} \forall p, r \in S\colon p \preceq^{fT}_{ors} r \Rightarrow f(p) \leq f(r). \quad (5)$$

Abusing the notation, let RankSim($\mathcal{A}$) denotes the output of Schewe($\mathcal{A}$) $= (Q', \delta', I', F')$ where states from the tight part of Q' are restricted to those that satisfy RankSim.

▶ **Lemma 6.** *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\text{RankSim}(\mathcal{A})) = \mathcal{L}(\text{Schewe}(\mathcal{A}))$.*

From the definition of $\preceq_{ors}$, it is not immediate how to compute it, since it is defined over all infinite runs of $\mathcal{A}$ over all infinite words. The computation of a rich under-approximation of $\preceq_{ors}$ will be the topic of the rest of this section. We first note that $\preceq_{di} \subseteq \preceq_{ors}$, which is a consequence of the following lemma.

▶ **Lemma 7** (Lemma 7 in [8]). *Let $p, r \in Q$ be such that $p \preceq_{di} r$ and $\mathcal{G}_\alpha = (V, E)$ be the run DAG of $\mathcal{A}$ over α. For all $i \geq 0$, $((p, i) \in V \wedge (r, i) \in V) \Rightarrow \mathit{rank}_\alpha(p, i) \leq \mathit{rank}_\alpha(r, i)$.*

We extend $\preceq_{di}$ into a relation $\preceq_R$, which is computed statically on $\mathcal{A}$, and then show that $\preceq_R \subseteq \preceq_{ors}$. The relation $\preceq_R$ is defined recursively as the smallest binary relation over Q s.t. (i) $\preceq_{di} \subseteq \preceq_R$ and (ii) for $p, r \in Q$, if $\forall a \in \Sigma : (\delta(p, a) \setminus F) \preceq^{\forall\forall}_R (\delta(r, a) \setminus F)$, then $p \preceq_R r$. Here, $S_1 \preceq^{\forall\forall}_R S_2$ holds iff $\forall x \in S_1, \forall y \in S_2 : x \preceq_R y$. The relation $\preceq_R$ can then be computed using a standard *worklist* algorithm, starting from $\preceq_{di}$ and adding pairs of states for which condition 2 holds until a fixpoint is reached.

▶ **Lemma 8.** *We have $\preceq_R \subseteq \preceq_{ors}$.*

Putting it all together, we modify (5) by substituting $\preceq_{ors}^{fT}$ with $\preceq_{R}^{fT}$, which denotes the transitive closure of $\preceq_{R}^{f}$, where $\preceq_{R}^{f}$ is a relation defined (by modifying (4)) as

$$p \preceq_{R}^{f} r \overset{\text{def}}{\equiv} f(p) \text{ is odd} \wedge f(r) \text{ is odd} \wedge p \preceq_{R} r. \tag{6}$$

Because $\preceq_R \subseteq \preceq_{ors}$, Lemma 6 still holds. We denote the modification of RankSim that uses $\preceq_{R}^{fT}$ instead of $\preceq_{ors}^{fT}$ as RankSim′.

▶ **Example 9.** Consider the BA $\mathcal{A}$ (top) and the part of Schewe($\mathcal{A}$) (bottom) in Figure 2c. Note that $r_2 \preceq_{di} q_2$ and $q_2 \preceq_{di} r_2$ so $r_2 \preceq_R q_2$ and $q_2 \preceq_R r_2$. From the definition of $\preceq_R$, we can deduce that $r_1 \preceq_R q_1$ (since $\{r_2\} \preceq_R^{\forall\forall} \{q_2\}$) and $q_1 \preceq_R r_1$ (since $\{q_2\} \preceq_R^{\forall\forall} \{r_2\}$). Note that $q_1 \not\preceq_{di} r_1$). As a consequence and due to the odd ranks of q_1 and r_1, we can eliminate the macrostates $(\{q_1{:}1, r_1{:}3\}, \emptyset, 0)$ and $(\{q_1{:}3, r_1{:}1\}, \emptyset, 0)$.

4.4 Ranking Restriction

Another optimization, called RankRestr, restricts ranks of successors of states with an odd rank. In particular, in a super-tight run, every odd-ranked state has a successor with the same rank (this follows from the construction of the run DAG). Let $\mathcal{A}$ be a BA and $\mathcal{B} = \text{Schewe}(\mathcal{A}) = (Q, \delta_1 \cup \delta_2 \cup \delta_3, I, F)$. We define the following restriction on transitions:

$$\begin{aligned}&\text{RankRestr}((S,O,f,i) \xrightarrow{a} (S',O',f',i')) \overset{\text{def}}{\equiv}\\ &\qquad \forall q \in S : f(q) \text{ is odd } \Rightarrow (\exists q' \in \delta(q,a) : f'(q') = f(q)).\end{aligned} \tag{7}$$

We abuse notation and use RankRestr($\mathcal{A}$) to denote $\mathcal{B}$ with transitions from δ_3 restricted to those that satisfy RankRestr. See Figure 2d for an example of a transition (and a newly unreachable macrostate) removed using RankRestr.

▶ **Lemma 10.** *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\text{RankRestr}(\mathcal{A})) = \mathcal{L}(\text{Schewe}(\mathcal{A}))$.*

4.5 Maximum Rank Construction

Our next optimization, named MaxRank, has the biggest practical effect. We introduce it as the last one because it depends on our previous optimizations (in particular SuccRank and RankSim′). It is a modified version of Schewe's "Reduced Average Outdegree" construction [33, Section 4], named Schewe$_{\text{RedAvgOut}}$, which may omit some runs, the so-called *max-rank* runs, that are essential for our other optimizations (we discuss the particular issue later).[2]

The main idea of MaxRank is that a set of runs of $\mathcal{B} = \text{Schewe}(\mathcal{A})$ (including super-tight runs) that assign different ranks to non-trunk states is represented by a single, "maximal," not necessarily super-tight (but having the same rank), run in $\mathcal{C} = \text{MaxRank}(\mathcal{A})$. We call such runs *max-rank runs*. More concretely, when moving from the waiting to the tight part, $\mathcal{C}$ needs to correctly guess a rank that is needed on an accepting run and the first tight core of a trunk of the run. The ranks of the rest of states are made maximal. Then, the tight part of $\mathcal{C}$ contains for each macrostate and symbol at most two successors: one via η_3 and one via η_4. Loosely speaking, the η_3-successor keeps all ranks as high as possible, while

[2] We believe that this property was not originally intended by the author, since it is not addressed in the proof. As far as we can tell, the construction is correct, although the original argument of the proof in [33] needs to be corrected.

the η_4-successor decreases the rankings of all non-accepting states in O (and can therefore help emptying O, which is necessary for an accepting run).

Before we give the construction, let us first provide some needed notation. We now use $(S, O, f, i) \leq (S, O, g, i)$ to denote that $f \leq g$ and similarly for $<$ (note that non-ranking components of the macrostates need to match).

The construction is then formally defined as $\textsc{MaxRank}(\mathcal{A}) = (Q_1 \cup Q_2, \eta, I', F')$ with $\eta = \delta_1 \cup \eta_2 \cup \eta_3 \cup \eta_4$ such that $Q_1, Q_2, I', F', \delta_1$ are the same as in $\textsc{Schewe}$. Let $\mathcal{B} = \textsc{Delay}(\mathcal{A}) = (\cdot, \delta_1 \cup \theta_2 \cup \delta_3, \cdot, \cdot)$ where δ_1, θ_2, and δ_3 are defined as in $\textsc{Delay}$. We define an auxiliary transition function that uses our previous optimizations as follows:

$$\Delta^\bullet(q, a) = \{q' \mid q' \in \theta_2(q, a) \wedge \textsc{RankSim}'(q') \wedge \textsc{SuccRank}(q')\}. \tag{8}$$

(We note that q is from the waiting and q' is from the tight part of $\mathcal{B}$.) Given a macrostate (S, O, f, i) and $a \in \Sigma$, we define the maximal successor ranking $f'_{max} = \textit{max-rank}((S, O, f, i), a)$ as follows. Consider $q' \in \delta(S, a)$ and the rank $r = \min\{f(s) \mid s \in \delta^{-1}(q', a) \cap S\}$. Then

- $f'_{max}(q') := r - 1$ if r is odd and $q' \in F$ and
- $f'_{max}(q') := r$ otherwise.

Let δ_3 be the transition function of the tight part of $\textsc{Schewe}(\mathcal{A})$. We can now proceed to the definition of the missing components of $\textsc{MaxRank}(\mathcal{A})$:

- $\eta_2(S, a) := \{(S', \emptyset, f', 0) \in \Delta^\bullet(S, a) \mid (S', \emptyset, f', 0)$ is a maximal element of $\leq$ in $\Delta^\bullet(S, a)\}$.
- $\eta_3((S, O, f, i), a)$: Let $f'_{max} = \textit{max-rank}((S, O, f, i), a)$. Then, we set
  - $\eta_3((S, O, f, i), a) := \{(S', O', f'_{max}, i')\}$ when $(S', O', f'_{max}, i') \in \delta_3((S, O, f, i), a)$ (i.e., if f'_{max} is tight; note that, in general, the result of *max-rank* may not be tight) and
  - $\eta_3((S, O, f, i), a) := \emptyset$ otherwise.
- $\eta_4((S, O, f, i), a)$: Let $\eta_3((S, O, f, i), a) = \{(S', P', h', i')\}$ and let
  - $f' = h' \lhd \{u \mapsto h'(u) - 1 \mid u \in P' \setminus F\}$ and
  - $O' = P' \cap f'^{-1}(i')$.

  Then, if $i' \neq 0$, we set $\eta_4((S, O, f, i), a) := \{(S', O', f', i')\}$, else we set $\eta_4((S, O, f, i), a) := \emptyset$.

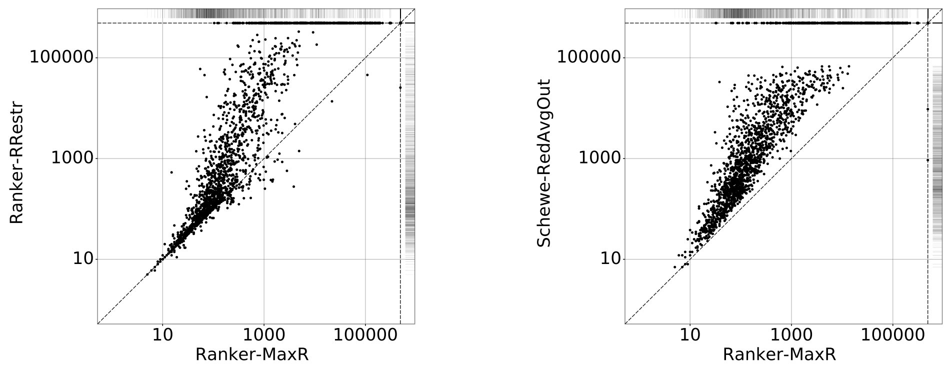
$\textsc{MaxRank}$ differs from $\textsc{Schewe}_{\textsc{RedAvgOut}}$ in the definition of η_2 and η_4. In particular, in the η_4 of $\textsc{Schewe}_{\textsc{RedAvgOut}}$ (named γ_4 therein), the condition that only non-accepting states ($u \in P' \setminus F$) decrease rank is omitted. Instead, the rank of all states in P' is decreased by one, which might create a "false ranking" (not an actual ranking since an accepting state is given an odd rank), so the target macrostate is omitted from the complement. Due to this, some max-rank runs may also be removed. Our construction preserves max-rank runs, which makes the proof of the theorem significantly more involved.

▶ **Theorem 11.** *Let $\mathcal{A}$ be a BA and $\mathcal{C} = \textsc{MaxRank}(\mathcal{A})$. Then $\mathcal{L}(\mathcal{C}) = \overline{\mathcal{L}(\mathcal{A})}$.*

Note that $\textsc{MaxRank}$ is incompatible with $\textsc{RankRestr}$ since $\textsc{RankRestr}$ optimizes the transitions in the tight part of the complement BA, which are abstracted in $\textsc{MaxRank}$.

4.6 Backing Off

Our final optimization, called $\textsc{BackOff}$, is a strategy for guessing when our optimized rank-based construction is likely (despite the optimizations) to generate too many states and when it might be helpful to give up and use a different complementation procedure instead. We evaluate this after the initial phase of $\textsc{Schewe}$, which constructs δ_2 (η_2 in $\textsc{MaxRank}$, θ_2 in $\textsc{Delay}$; we will just use δ_2 now), finishes. We provide a set of pairs



(a) Ranker$_{\text{MaxR}}$ vs Ranker$_{\text{RRestr}}$ **(b)** Ranker$_{\text{MaxR}}$ vs Schewe$_{\text{RedAvgOut}}$

Figure 3 Evaluation of the effectiveness of our optimizations on the generated state space (axes are logarithmic). The horizontal and vertical dashed lines represent timeouts.

$\{(\mathit{StateSize}_j, \mathit{RankMax}_j)\}_{j\in\mathcal{J}}$ for an index set $\mathcal{J}$ (obtained experimentally) and check (after δ_2 is constructed) that for all $(S, O, f, i) \in \mathrm{img}(\delta_2)$ and all $j \in \mathcal{J}$ it holds that either $|S| < \mathit{StateSize}_j$ or $\mathit{rank}(f) < \mathit{RankMax}_j$. If for some (S, O, f, i) and j the condition does not hold, we terminate the construction and execute a different, *surrogate*, procedure.

5 Experimental Evaluation

Used tools and evaluation environment. We implemented the optimizations described in the previous sections in a tool called Ranker [17] in C++ (we tested the correctness of our implementation using Spot's `autcross` on all BAs in our benchmark). We compared our complementation approach with other state-of-the-art tools, namely, GOAL [37] (including the Fribourg plugin [1]), Spot 2.9.3 [11], Seminator 2 [4], LTL2dstar 0.5.4 [21], and ROLL [24]. All tools were set to the mode where they output an automaton with the standard state-based Büchi acceptance condition. We note that some of the tools are aimed at complementing more general flavours of ω-automata, such as Seminator 2 focusing on generalized transition-based Büchi automata. 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. The timeout was set to 5 minutes.

Dataset. The source of our main benchmark are the 11,000 BAs used in [36], which were randomly generated using the Tabakov-Vardi approach [35] over a two letter alphabet, starting from 15 states and with various different parameters (see [36] for more details). In preprocessing, the automata were reduced using a combination of Rabit [27] and Spot's `autfilt` (using the `-high` simplification level) and converted to the HOA format [2]. From this set, we removed automata that are (i) semi-deterministic, (ii) inherently weak, or (iii) unambiguous, since for these kinds of automata there exist more efficient complementation procedures than for unrestricted BAs [3, 4, 5, 26]. Moreover, we removed BAs with an empty language or empty language of complement. We were left with **2,393** *hard* automata.

Selection of Optimizations. We use two settings of Ranker with different optimizations turned on. Since the RankRestr and MaxRank optimizations are incompatible, the main difference between the settings is which one of those two they use. The particular optimizations used in the settings are the following:

Table 1 Statistics for our experiments. The upper part compares different optimizations of the rank-based procedure (no postprocessing). The lower part compares our approach with other methods (with postprocessing). "BO" denotes the BackOff optimization. In the left-hand side of the table, the column "**med.**" contains the median, "**std. dev**" contains the standard deviation, and "**TO**" contains the number of timeouts (5 mins). In the right-hand side of the table, we provide the number of cases where our tool (Ranker$_{\text{MaxR}}$ without postprocessing in the upper part and with postprocessing in the lower part) was strictly better ("**wins**") or worse ("**losses**"). The "**(TO)**" column gives the number of times this was because of the timeout of the loser. Approaches implemented in GOAL are labelled with ⚽.

method	max	mean	med.	std. dev	TO	wins	(TO)	losses	(TO)
Ranker$_{\text{MaxR}}$	319 119	8 051.58	185	28 891.4	360	—	—	—	—
Ranker$_{\text{RRestr}}$	330 608	9 652.67	222	32 072.6	854	1810	(495)	109	(1)
Schewe$_{\text{RedAvgOut}}$ ⚽	67 780	5 227.3	723	10 493.8	844	2030	(486)	3	(2)
Ranker$_{\text{MaxR}}$	1 239	61.83	32	103.18	360	—	—	—	—
Ranker$_{\text{MaxR}}$+BO	1 706	73.65	33	126.8	17	—	—	—	—
Piterman ⚽	1 322	88.30	40	142.19	12	1 069	(3)	469	(351)
Safra ⚽	1 648	99.22	42	170.18	158	1 171	(117)	440	(319)
Spot	2 028	91.95	38	158.13	13	907	(6)	585	(353)
Fribourg ⚽	2 779	113.03	36	221.91	78	996	(51)	472	(333)
LTL2dstar	1 850	88.76	41	144.09	128	1 156	(99)	475	(331)
Seminator 2	1 772	98.63	33	191.56	345	1 081	(226)	428	(241)
ROLL	1 313	21.50	11	57.67	1 106	1 781	(1 041)	522	(295)

$$\textsc{Ranker}_{\textsc{MaxR}} = \textsc{Delay} + \textsc{SuccRank} + \textsc{RankSim}' + \textsc{MaxRank}$$
$$\textsc{Ranker}_{\textsc{RRestr}} = \textsc{Delay} + \textsc{SuccRank} + \textsc{RankSim}' + \textsc{RankRestr} + \textsc{Purge}_{di}$$

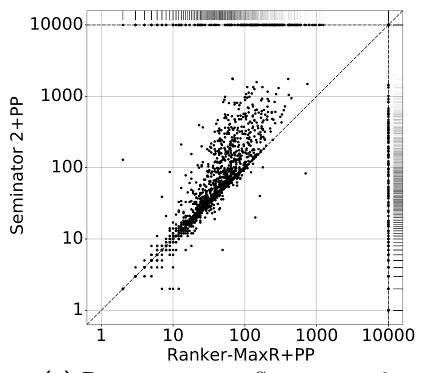
(The Purge$_{di}$ optimization is from [8].) Note that the two settings include all optimizations compatible with MaxRank and RankRestr respectively. Due to space constraints, we cannot give a detailed analysis of the effect of individual optimizations on the size of the obtained complement automaton. Let us, at least, give a bird's-eye view. The biggest effect has MaxRank, followed by Delay—their use is key to obtaining a small state space. The rest of the optimizations are less effective, but they still remove a significant number of states.

5.1 Comparison of Rank-Based Procedures

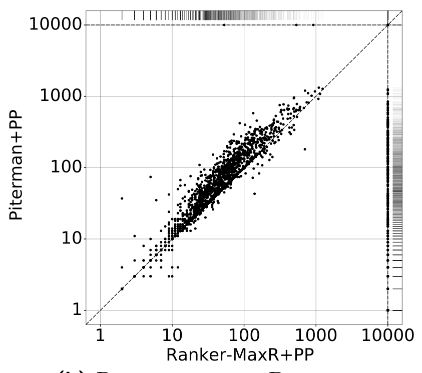
First, we evaluated how our optimizations reduce the generated state space, i.e., we compared the sizes of complemented BAs with no postprocessing. Such a use case represents applications like testing inclusion or equivalence of BAs, where postprocessing the output is irrelevant.

More precisely, we first compared the sizes of automata produced by our settings Ranker$_{\text{MaxR}}$ and Ranker$_{\text{RRestr}}$ to see which of them behaves better (cf. Figure 3a) and then we compared Ranker$_{\text{MaxR}}$, which had better results, with the Schewe$_{\text{RedAvgOut}}$ procedure implemented in GOAL (parameters `-m rank -tr -ro`). Scatter plots of the results are given in Figure 3b and summarizing statistics in the upper part of Table 1.

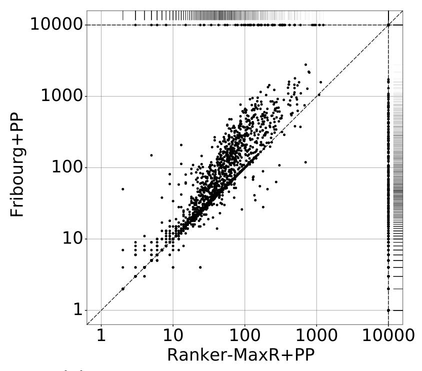
We note that although Ranker$_{\text{MaxR}}$ produces in the vast majority of cases (1,810) smaller automata than Ranker$_{\text{RRestr}}$, in a few cases (109) Ranker$_{\text{RRestr}}$ still outputs a smaller result (in 1 case this is due to the timeout of Ranker$_{\text{MaxR}}$). The comparison with Schewe$_{\text{RedAvgOut}}$ shows that our optimizations indeed have a profound effect on the size of the generated state space. Although the mean and maximum size of complements produced by Ranker$_{\text{MaxR}}$ and Ranker$_{\text{RRestr}}$ are larger than those of Schewe$_{\text{RedAvgOut}}$, this is because for cases where the complement would be large, the run of Schewe$_{\text{RedAvgOut}}$ in GOAL timeouted before it could produce a result. Therefore, the median is a more meaningful indicator, and it is significantly (3–4×) lower for both Ranker$_{\text{MaxR}}$ and Ranker$_{\text{RRestr}}$.



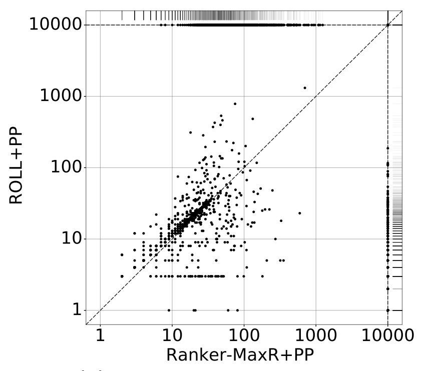
(a) RANKER$_{\text{MAXR}}$ VS SEMINATOR 2



(b) RANKER$_{\text{MAXR}}$ VS PITERMAN



(c) RANKER$_{\text{MAXR}}$ VS FRIBOURG



(d) RANKER$_{\text{MAXR}}$ VS ROLL

Figure 4 Comparison of the sizes of the BAs constructed using our optimized rank-based construction and other approaches. Timeouts are on the dashed lines.

5.2 Comparison with Other Approaches

Further, we evaluated the complements produced by RANKER$_{\text{MAXR}}$ and other approaches. In this setting, we focused on the size of the output BA *after* postprocessing (we, again, used `autfilt` with simplification `-high`; we denote this using "+PP"). We evaluated the following algorithms: SAFRA [32], its optimization PITERMAN [30] the optimization implemented in LTL2DSTAR [21], FRIBOURG [1], SPOT (Redziejowski's algorithm [31]), ROLL's learning-based algorithm [25], and a semideterminization-based algorithm [3] in SEMINATOR 2.

In Figure 4, we give scatter plots of selected comparisons; we omitted the results for SAFRA, SPOT, and LTL2DSTAR, which on average performed slightly worse than PITERMAN. We give summarizing statistics in the lower part of Table 1 and the run times in Table 2.

Let us now discuss the data in the lower part of Table 1. In the left-hand side, we can see that the mean and median size of BAs obtained by RANKER$_{\text{MAXR}}$ are both the lowest with the exception of ROLL. ROLL implements a learning-based approach, which means that it works on the level of the *language* of the input BA instead of the *structure*. Therefore, it can often find a much smaller automaton than other approaches. Its practical time complexity, however, seems to grow much faster with the number of states of the output BA than other approaches (cf. Table 2). RANKER$_{\text{MAXR}}$ by itself had more timeouts than other approaches, but when used with the BACKOFF strategy, is on par with PITERMAN and SPOT.

In the right-hand side of Table 1, we give the numbers of times where RANKER$_{\text{MAXR}}$ gave strictly smaller and strictly larger outputs respectively. Here, we can see that the output of RANKER$_{\text{MAXR}}$ is often at least as small as the output of the other method (this is not

Table 2 Run times of the tools [s]

method	mean	med.	std. dev
Ranker$_{\text{MaxR}}$	10.21	0.84	28.43
Ranker$_{\text{MaxR}}$+BO	9.40	3.03	16.00
Piterman ⚽	7.47	6.03	8.46
Safra ⚽	15.49	7.03	35.59
Spot	1.07	0.02	8.94
Fribourg ⚽	19.43	10.01	32.76
LTL2dstar	4.17	0.06	22.19
Seminator 2	11.41	0.37	34.97
ROLL	42.63	14.92	67.31

Table 3 Wins and losses for Ranker$_{\text{MaxR}}$+BO

method	wins	(TO)	losses	(TO)
Piterman ⚽	1 160	(4)	112	(9)
Safra ⚽	1 255	(147)	222	(6)
Spot	985	(8)	328	(12)
Fribourg ⚽	1 076	(71)	287	(10)
LTL2dstar	1 208	(118)	272	(7)
Seminator 2	1 236	(333)	253	(5)
ROLL	1 923	(1 096)	360	(7)

in the table, but can be computed as $2,393 -$ **losses**; the losses were caused mostly by timeouts; results with the BackOff strategy would increase the number even more) and often a strictly smaller one (the **wins** column). When comparing Ranker$_{\text{MaxR}}$ with *the best result of any other tool*, it obtained a *strictly smaller* BA in 539 cases (22.5 %) and a BA *at least as small* as the best result of any other tool in 1,518 cases (63.4 %). Lastly, we note that there were four BAs in the benchmark that *no tool* could complement and one BA that *only* Ranker$_{\text{MaxR}}$ was able to complement; there was no such a case for any other tool.

Let us now focus on the run times of the tools in Table 2. GOAL and ROLL are implemented in Java, which adds a significant overhead to the run time (e.g., the fastest run time of GOAL was 3.15 s; it is hard to predict how their performance would change if they were reimplemented in a faster language); the other approaches are implemented in C++.

BackOff. Our BackOff setting in the experiments used the set of constraints $\{(\mathit{StateSize}_1 = 9, \mathit{RankMax}_1 = 5), (\mathit{StateSize}_2 = 8, \mathit{RankMax}_2 = 6)\}$ and Piterman as the surrogate algorithm. The BackOff strategy was executed 873 times and managed to decrease the number of timeouts of Ranker$_{\text{MaxR}}$ from 360 to 17 (row Ranker$_{\text{MaxR}}$+BO in Table 1).

Discussion. The results of our experiments show that our optimizations are key to making rank-based complementation competitive to other approaches in practice. Furthermore, with the optimizations, the obtained procedure in the majority of cases produces a BA at least as small as a BA produced by any other approach, and in a large number of cases *the smallest* BA produced by any existing approach. We emphasize the usefulness of the BackOff heuristic: as there is no clear "best" complementation algorithm—different techniques having different strengths and weaknesses—knowing which technique to use for an input automaton is important in practice. In Table 3, we give a modification of the right-hand side of Table 1 giving wins and losses for Ranker$_{\text{MaxR}}$+BO. It seems that the combination of these two completely different algorithms yields a quite strong competitor.

6 Related Work

The problem of BA complementation has attracted researchers since Büchi's seminal work [7]. Since then, there have appeared several directions of BA complementation approaches. *Ramsey-based complementation* using Büchi's original argument, decomposing the language accepted by an automaton into a finite number of equivalence classes, was later improved in [6]. *Determinization-based complementation* was introduced by Safra in [32], later improved by Piterman in [30]. Determinization-based approaches convert an input BA into an equivalent intermediate deterministic automaton with different accepting condition (e.g. Rabin automaton) that can be easily complemented. The result is then converted back into a BA (often for the price of some blow-up). *Slice-based complementation* uses a reduced abstraction on a run tree to track the acceptance condition [39, 19]. *A learning-based approach* was presented in [25, 24]. A novel optimal complementation algorithm by Allred and Utes-Nitsche was presented in [1]. There are also specific approaches for complementation of special types of BAs, e.g., deterministic [23], semi-deterministic [3], or unambiguous [26]. *Semi-determinization based complementation* then uses a conversion of a standard BA to a semi-deterministic version [10] followed by its complementation [4].

Rank-based complementation, studied in [22, 15, 13, 33, 20], extends the subset construction for determinizing finite automata with additional information kept in each macrostate to track the acceptance condition of all runs of the input automaton. We have described the refinement of the basic procedure from [22] towards [13] and [33] in Section 3. The work in [15] contains optimizations of an alternative (sub-optimal) rank-based construction from [22] that goes through *alternating Büchi automata*. Furthermore, the work in [20] proposes an optimization of SCHEWE that in some cases produces smaller automata (the construction is not compatible with our optimizations). Rank-based construction can be optimized using simulation relations as shown in [8]. Here the direct and delayed simulation relations can be used to prune macrostates that are redundant for accepting a word or to saturate macrostates with simulation-smaller states.

Acknowledgements. We thank reviewers of this and previously submitted versions of the paper for their useful remarks that helped us improve the quality of the paper. This work was supported by the Czech Science Foundation project 20-07487S and the FIT BUT internal project FIT-S-20-6427.

References

1 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.

2 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. doi:10.1007/978-3-319-21690-4_31.

3 Frantisek Blahoudek, Matthias Heizmann, Sven Schewe, Jan Strejcek, 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. doi:10.1007/978-3-662-49674-9_49.

4 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. doi:10.1007/978-3-030-53291-8_2.

5 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. doi:10.1007/3-540-45744-5_50.

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

7 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.

8 Yu-Fang Chen, Vojtech Havlena, and Ondrej Lengál. Simulations in rank-based Büchi automata complementation. In Anthony Widjaja Lin, editor, *Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*, volume 11893 of *Lecture Notes in Computer Science*, pages 447–467. Springer, 2019. `doi:10.1007/978-3-030-34175-6\_23`.

9 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. `doi:10.1145/3192366.3192405`.

10 Costas Courcoubetis and Mihalis Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 338–345. IEEE Computer Society, 1988. `doi:10.1109/SFCS.1988.21950`.

11 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.

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

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

14 Patrice Godefroid. Using partial orders to improve automatic verification methods. In Edmund M. Clarke and Robert P. Kurshan, editors, *Computer Aided Verification, 2nd International Workshop, CAV '90, New Brunswick, NJ, USA, June 18-21, 1990, Proceedings*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185. Springer, 1990. `doi:10.1007/BFb0023731`.

15 Sankar Gurumurthy, Orna Kupferman, Fabio Somenzi, and Moshe Y. Vardi. On complementing nondeterministic Büchi automata. In Daniel Geist and Enrico Tronci, editors, *Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L'Aquila, Italy, October 21-24, 2003, Proceedings*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer, 2003. `doi:10.1007/978-3-540-39724-3\_10`.

16 Vojtěch Havlena and Ondřej Lengál. Reducing (to) the ranks: Efficient rank-based Büchi automata complementation (technical report). *CoRR*, abs/2010.07834, 2020. URL: `https://arxiv.org/abs/2010.07834`, `arXiv:2010.07834`.

17 Vojtěch Havlena and Ondřej Lengál. Ranker, 2021. `https://github.com/vhavlena/ba-inclusion`.

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

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

20 Hrishikesh Karmarkar and Supratik Chakraborty. On minimal odd rankings for Büchi complementation. In Zhiming Liu and Anders P. Ravn, editors, *Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings*, volume 5799 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2009. `doi:10.1007/978-3-642-04761-9\_18`.

21 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. `doi:10.1007/978-3-540-76336-9\_7`.

22 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001. `doi:10.1145/377978.377993`.

23 Robert P. Kurshan. Complementing deterministic Büchi automata in polynomial time. *J. Comput. Syst. Sci.*, 35(1):59–71, 1987. `doi:10.1016/0022-0000(87)90036-5`.

24 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. `doi:10.1007/978-3-030-17462-0\_23`.

25 Yong Li, Andrea Turrini, Lijun Zhang, and Sven Schewe. Learning to complement Büchi automata. In *Proc. of VMCAI'18*, pages 313–335. Springer, 2018.

26 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. `doi:10.4204/EPTCS.326.12`.

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

28 Max Michel. Complementation is more difficult with automata on infinite words. *CNET, Paris*, 15, 1988.

29 Doron A. Peled. All from one, one for all: on model checking using representatives. In Costas Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993. `doi:10.1007/3-540-56922-7\_34`.

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

31 Roman R. Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundam. Informaticae*, 119(3-4):393–406, 2012. `doi:10.3233/FI-2012-744`.

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

33 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. `doi:10.4230/LIPIcs.STACS.2009.1854`.

34 Prasad A. Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *Automata, Languages and Programming*, pages 465–474. Springer, 1985.

35 Deian Tabakov and Moshe Y. Vardi. Experimental evaluation of classical automata constructions. In *Proc. of LPAR'05*, pages 396–411. Springer, 2005.

36 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.

37 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.

38 Antti Valmari. Stubborn sets for reduced state space generation. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1990*, pages 491–515, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

39 Moshe Y. Vardi and Thomas Wilke. Automata: From logics to algorithms. *Logic and Automata*, 2:629–736, 2008.

40 Qiqi Yan. Lower bounds for complementation of ω-automata via the full automata technique. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 589–600, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

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