Sky Is Not the Limit

Tighter Rank Bounds for Elevator Automata in Büchi Automata Complementation

Vojtěch Havlena, Ondřej Lengál✉, and Barbora Šmahlíková

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

Abstract. We propose several heuristics for mitigating one of the main causes of combinatorial explosion in rank-based complementation of Büchi automata (BAs): unnecessarily high bounds on the ranks of states. First, we identify *elevator automata*, which is a large class of BAs (generalizing semi-deterministic BAs), occurring often in practice, where ranks of states are bounded according to the structure of strongly connected components. The bounds for elevator automata also carry over to general BAs that contain elevator automata as a sub-structure. Second, we introduce two techniques for refining bounds on the ranks of BA states using data-flow analysis of the automaton. We implement out techniques as an extension of the tool Ranker for BA complementation and show that they indeed greatly prune the generated state space, obtaining significantly better results and outperforming other state-of-the-art tools on a large set of benchmarks.

1 Introduction

Büchi automata (BA) complementation has been a fundamental problem underlying many applications since it was introduced in 1962 by Büchi [8,17] as an essential part of a decision procedure for a fragment of the second-order arithmetic. BA complementation has been used as a crucial part of, e.g., termination analysis of programs [13,20,10] or decision procedures for various logics, such as S1S [8], the first-order logic of Sturmian words [33], or the temporal logics ETL and QPTL [38]. Moreover, BA complementation also underlies BA inclusion and equivalence testing, which are essential instruments in the BA toolbox. Optimal algorithms, whose output asymptotically matches the lower bound of $(0.76n)^n$ [43] (potentially modulo a polynomial factor), have been developed [37,1]. For a successful real-world use, asymptotic optimality is, however, not enough and these algorithms need to be equipped with a range of optimizations to make them behave better than the worst case on BAs occurring in practice.

In this paper, we focus on the so-called *rank-based* approach to complementation, introduced by Kupferman and Vardi [24], further improved with the help of Friedgut [14], and finally made optimal by Schewe [37]. The construction stores in a macrostate partial information about all runs of a BA $\mathcal{A}$ over some word α. In addition to tracking states that $\mathcal{A}$ can be in (which is sufficient, e.g., in the determinization of NFAs), a macrostate also stores a guess of the rank of each of the tracked states in the *run DAG* that captures all these runs. The guessed ranks impose restrictions on how the future of a state might look like (i.e., when $\mathcal{A}$ may accept). The number of macrostates in the complement

depends combinatorially on the maximum rank that occurs in the macrostates. The constructions in [24,14,37] provides only coarse bounds on the maximum ranks.

A way of decreasing the maximum rank has been suggested in [15] using a PSpace (and, therefore, not really practically applicable) algorithm (the problem of finding the optimal rank is PSpace-complete). In our previous paper [19], we have identified several basic optimizations of the construction that can be used to refine the *tight-rank upper bound* (TRUB) on the maximum ranks of states. In this paper, we push the applicability of rank-based techniques much further by introducing two novel lightweight techniques for refining the TRUB, thus significantly reducing the generated state space.

Firstly, we introduce a new class of the so-called *elevator automata*, which occur quite often in practice (e.g., as outputs of natural algorithms for translating LTL to BAs). Intuitively, an elevator automaton is a BA whose strongly connected components (SCCs) are all either inherently weak[1] or deterministic. Clearly, the class substantially generalizes the popular inherently weak [6] and semi-deterministic BAs [11,3,4]). The structure of elevator automata allows us to provide tighter estimates of the TRUBs, not only for elevator automata *per se*, but also for BAs where elevator automata occur as a sub-structure (which is even more common). Secondly, we propose a lightweight technique, inspired by data flow analysis, allowing to propagate rank restriction along the skeleton of the complemented automaton, obtaining even tighter TRUBs. We also extended the optimal rank-based algorithm to transition-based BAs (TBAs).

We implemented our optimizations within the Ranker tool [18] and evaluated our approach on thousands of hard automata from the literature (15 % of them were elevator automata that were not semi-deterministic, and many more contained an elevator sub-structure). Our techniques drastically reduce the generated state space; in many cases we even achieved exponential improvement compared to the optimal procedure of Schewe and our previous heuristics. The new version of Ranker gives a smaller complement in the majority of cases of hard automata than other state-of-the-art tools.

2 Preliminaries

Words, functions. 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 $\{0, \ldots, n\}$. For $i \in \omega$ we use $\llbracket i \rrbracket$ to denote the largest even number smaller of equal to i, e.g., $\llbracket 42 \rrbracket = \llbracket 43 \rrbracket = 42$. 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 $\alpha = \alpha_0\alpha_1 \ldots$ We use Σ^ω to denote the set of all infinite words over Σ. For a (partial) function $f\colon X \to Y$ and a set $S \subseteq X$, we define $f(S) = \{f(x) \mid x \in S\}$. Moreover, for $x \in X$ and $y \in Y$, we use $f \triangleleft \{x \mapsto y\}$ to denote the function $(f \setminus \{x \mapsto f(x)\}) \cup \{x \mapsto y\}$.

Büchi automata. A (nondeterministic transition/state-based) *Büchi automaton* (BA) over Σ is a quadruple $\mathcal{A} = (Q, \delta, I, Q_F \cup \delta_F)$ 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. 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 $q \in \delta(p, a)$.

[1] An SCC is inherently weak if it either contains no accepting states or, on the other hand, all cycles of the SCC contain an accepting state.

Moreover, we extend δ to sets of states $P \subseteq Q$ as $\delta(P,a) = \bigcup_{p\in P} \delta(p,a)$, and to sets of symbols $\Gamma \subseteq \Sigma$ as $\delta(P,\Gamma) = \bigcup_{a\in\Gamma} \delta(P,a)$. We define the inverse transition function as $\delta^{-1} = \{p \xrightarrow{a} q \mid q \xrightarrow{a} p \in \delta\}$. 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 BA $(Q, \delta, \{q\}, Q_F \cup \delta_F)$.

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 $\inf_Q(\rho)$ denote the states occurring in ρ infinitely often and $\inf_\delta(\rho)$ denote the transitions occurring in ρ infinitely often. The run ρ is called *accepting* iff $\inf_Q(\rho) \cap Q_F \neq \emptyset$ or $\inf_\delta(\rho) \cap \delta_F \neq \emptyset$.

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 $\mathcal{L}(\mathcal{A}) = \mathcal{L}_\mathcal{A}(I)$. We say that a state $q \in Q$ is *useless* iff $\mathcal{L}_\mathcal{A}(q) = \emptyset$. If $\delta_F = \emptyset$, we call $\mathcal{A}$ *state-based* and if $Q_F = \emptyset$, we call $\mathcal{A}$ *transition-based*. In this paper, we fix a BA $\mathcal{A} = (Q, \delta, I, Q_F \cup \delta_F)$.

3 Complementing Büchi automata

In this section, we describe a generalization of the rank-based complementation of state-based BAs presented by Schewe in [37] to our notion of transition/state-based BAs. Proofs can be found in [16].

3.1 Run DAGs

First, we recall the terminology from [37] (which is a minor modification of the one in [24]), which we use in the paper. Let the *run DAG* of $\mathcal{A}$ over a word α be a DAG (directed acyclic graph) $\mathcal{G}_\alpha = (V, E)$ containing vertices V and edges E such that

- $V \subseteq Q \times \omega$ 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 $(p,i) \in V$. A vertex $(p,i) \in V$ is called *accepting* if p is an accepting state and an edge $((q,i),(q',i')) \in E$ is called *accepting* if $q \xrightarrow{\alpha_i} q'$ is an accepting transition. 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 it cannot reach an accepting vertex or an accepting edge.

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 $rank_\alpha(v) \leftarrow j$ 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 $rank_\alpha(v) \leftarrow j+1$ 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 $j \leftarrow j+2$.

For all vertices v that have not been assigned a rank yet, we assign $rank_\alpha(v) \leftarrow \omega$.

We define the *rank of* α, denoted as $rank(\alpha)$, as $\max\{rank_\alpha(v) \mid v \in \mathcal{G}_\alpha\}$ and the *rank of* $\mathcal{A}$, denoted as $rank(\mathcal{A})$, as $\max\{rank(w) \mid w \in \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})\}$.

Lemma 1. *If $\alpha \notin \mathcal{L}(\mathcal{A})$, then $rank(\alpha) \leq 2|Q|$.*

3.2 Rank-Based Complementation

In this section, we describe a construction for complementing BAs developed in the work of Kupferman and Vardi [24]—later improved by Friedgut, Kupferman, and Vardi [14], and by Schewe [37]—extended to our definition of BAs with accepting states and transitions (see [19] for a step-by-step introduction). The construction is based on the notion of tight level rankings storing information about levels in run DAGs. For a BA $\mathcal{A}$ and $n = |Q|$, a *(level) ranking* is a function $f : Q \to [2n]$ such that $f(Q_F) \subseteq \{0, 2, \ldots, 2n\}$, i.e., f assigns even ranks to accepting states of $\mathcal{A}$. For two rankings f and f' we define $f \overset{a}{\underset{S}{\ominus\!\!\rightarrow}} f'$ iff for each $q \in S$ and $q' \in \delta(q, a)$ we have $f'(q') \leq f(q)$ and for each $q'' \in \delta_F(q, a)$ it holds $f'(q'') \leq \lfloor\!\lfloor f(q) \rfloor\!\rfloor$. The set of all rankings is denoted by $\mathcal{R}$. 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 we use $f < f'$ iff $f \leq f'$ and there is a state $q \in Q$ with $f(q) < f'(q)$. For a set of states $S \subseteq Q$, we call f to be *S-tight* if (i) it has an odd rank r, (ii) $f(S) \supseteq \{1, 3, \ldots, r\}$, and (iii) $f(Q \setminus S) = \{0\}$. A ranking is *tight* if it is Q-tight; we use $\mathcal{T}$ to denote the set of all tight rankings.

The original rank-based construction [24] uses macrostates of the form (S, O, f) to track all runs of $\mathcal{A}$ over α. The f-component contains guesses of the ranks of states in S (which is obtained by the classical subset construction) in the run DAG and the O-set is used to check whether all runs contain only a finite number of accepting states. Friedgut, Kupferman, and Vardi [14] improved the construction by having f consider only tight rankings. Schewe's construction [37] extends the macrostates to (S, O, f, i) with $i \in \omega$ representing a particular even rank such that O tracks states with rank i. At the cut-point (a macrostate with $O = \emptyset$) the value of i is changed to $i + 2$ modulo the rank of f. Macrostates in an accepting run hence iterate over all possible values of i. Formally, the complement of $\mathcal{A} = (Q, \delta, I, Q_F \cup \delta_F)$ is given as the (state-based) BA $\textsc{Schewe}(\mathcal{A}) = (Q', \delta', I', Q'_F \cup \emptyset)$, whose components are defined as follows:

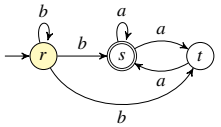
- $Q' = Q_1 \cup Q_2$ where
  - $Q_1 = 2^Q$ and
  - $Q_2 = \{(S, O, f, i) \in 2^Q \times 2^Q \times \mathcal{T} \times \{0, 2, \ldots, 2n-2\} \mid f \text{ is } S\text{-tight}, O \subseteq S \cap f^{-1}(i)\}$,
- $I' = \{I\}$,
- $\delta' = \delta_1 \cup \delta_2 \cup \delta_3$ where
  - $\delta_1 : Q_1 \times \Sigma \to 2^{Q_1}$ 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
    - $S' = \delta(S, a)$,
    - $f \overset{a}{\underset{S}{\ominus\!\!\rightarrow}} f'$,
    - $rank(f) = rank(f')$,
    - and
      - if $O = \emptyset$ then $i' = (i + 2) \mod (rank(f') + 1)$ and $O' = f'^{-1}(i')$, and
      - if $O \neq \emptyset$ then $i' = i$ and $O' = \delta(O, a) \cap f'^{-1}(i)$; and
- $Q'_F = \{\emptyset\} \cup ((2^Q \times \{\emptyset\} \times \mathcal{T} \times \omega) \cap Q_2)$.

We call the part of the automaton with states from Q_1 the *waiting* part (denoted as Waiting), and the part corresponding to Q_2 the *tight* part (denoted as Tight).

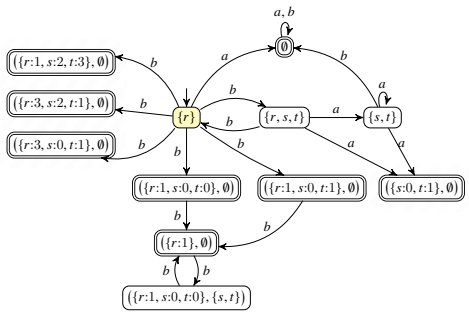
Theorem 2. *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\textsc{Schewe}(\mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.*

The space complexity of Schewe's construction for BAs matches the theoretical lower bound $O((0.76n)^n)$ given by Yan [43] modulo a quadratic factor $O(n^2)$. Note that our extension to BAs with accepting transitions does not increase the space complexity of the construction.

Example 3. Consider the BA $\mathcal{A}$ over $\{a, b\}$ given in Fig. 1a. A part of $\textsc{Schewe}(\mathcal{A})$ is shown in Fig. 1b (we use $(\{s{:}0, t{:}1\}, \emptyset)$ to denote the macrostate $(\{s, t\}, \emptyset, \{s \mapsto 0, t \mapsto 1\}, 0)$). We omit the i-part of each macrostate since the corresponding values are 0 for all macrostates in the figure. Useless states are covered by grey stripes. The full automaton contains even more transitions from $\{r\}$ to useless macrostates of the form $(\{r{:}\cdot, s{:}\cdot, t{:}\cdot\}, \emptyset)$. □



(a) BA $\mathcal{A}$ over $\{a, b\}$



(b) A part of $\textsc{Schewe}(\mathcal{A})$

Fig. 1: Schewe's complementation

From the construction of $\textsc{Schewe}(\mathcal{A})$, we can see that the number of states is affected mainly by sizes of macrostates and by the maximum rank of $\mathcal{A}$. In particular, the upper bound on the number of states of the complement with the maximum rank r is given in the following lemma.

Lemma 4. *For a BA $\mathcal{A}$ with sufficiently many states n such that $rank(\mathcal{A}) = r$ the number of states of the complemented automaton is bounded by $2^n + \frac{(r+m)^n}{(r+m)!}$ where $m = \max\{0, 3 - \lceil \frac{r}{2} \rceil\}$.*

From Lemma 1 we have that the rank of $\mathcal{A}$ is bounded by $2|Q|$. Such a bound is often too coarse and hence $\textsc{Schewe}(\mathcal{A})$ may contain many redundant states. Decreasing the bound on the ranks is essential for a practical algorithm, but an optimal solution is PSpace-complete [15]. The rest of this paper therefore proposes a framework of lightweight techniques for decreasing the maximum rank bound and, in this way, significantly reducing the size of the complemented BA.

3.3 Tight Rank Upper Bounds

Let $\alpha \notin \mathcal{L}(\mathcal{A})$. For $\ell \in \omega$, we define the ℓ-th *level* of $\mathcal{G}_\alpha$ as $level_\alpha(\ell) = \{q \mid (q, \ell) \in \mathcal{G}_\alpha\}$. Furthermore, we use f_ℓ^α to denote the ranking of level ℓ of $\mathcal{G}_\alpha$. Formally,

$$f_\ell^\alpha(q) = \begin{cases} rank_\alpha((q, \ell)) & \text{if } q \in level_\alpha(\ell), \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We say that the ℓ-th level of $\mathcal{G}_\alpha$ is *tight* if for all $k \geq \ell$ it holds that (i) f_k^α is tight, and (ii) $rank(f_k^\alpha) = rank(f_\ell^\alpha)$. Let $\rho = S_0 S_1 \ldots S_{\ell-1}(S_\ell, O_\ell, f_\ell, i_\ell) \ldots$ be a run on a word

α in Schewe($\mathcal{A}$). We say that ρ is a *super-tight run* [19] if $f_k = f_k^\alpha$ for each $k \geq \ell$. Finally, we say that a mapping $\mu\colon 2^Q \to \mathcal{R}$ is a *tight rank upper bound (TRUB) wrt* α iff

$$\exists \ell \in \omega\colon level_\alpha(\ell) \text{ is tight} \wedge (\forall k \geq \ell\colon \mu(level_\alpha(k)) \geq f_k^\alpha). \tag{2}$$

Informally, a TRUB is a ranking that gives a conservative (i.e., larger) estimate on the necessary ranks of states in a super-tight run. We say that μ is a TRUB iff μ is a TRUB wrt all $\alpha \notin \mathcal{L}(\mathcal{A})$. We abuse notation and use the term TRUB also for a mapping $\mu'\colon 2^Q \to \omega$ if the mapping $inner(\mu')$ is a TRUB where $inner(\mu')(S) = \{q \mapsto m \mid m = \mu'(S) \dot{-} 1 \text{ if } q \in Q_F \text{ else } m = \mu'(S)\}$ for all $S \in 2^Q$. ($\dot{-}$ is the *monus* operator, i.e., minus with negative results saturated to zero.) Note that the mappings $\mu_t = \{S \mapsto (2|S \setminus Q_F| \dot{-} 1)\}_{S \in 2^Q}$ and $inner(\mu_t)$ are trivial TRUBs.

The following lemma shows that we can remove from Schewe($\mathcal{A}$) macrostates whose ranking is not covered by a TRUB (in particular, we show that the reduced automaton preserves super-tight runs).

Lemma 5. *Let μ be a TRUB and $\mathcal{B}$ be a BA obtained from Schewe($\mathcal{A}$) by replacing all occurrences of Q_2 by $Q_2' = \{(S, O, f, i) \mid f \leq \mu(S)\}$. Then, $\mathcal{L}(\mathcal{B}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.*

4 Elevator Automata

In this section, we introduce *elevator automata*, which are BAs having a particular structure that can be exploited for complementation and semi-determinization; elevator automata can be complemented in $O(16^n)$ (cf. Lemma 10) space instead of $2^{O(n \log n)}$, which is the lower bound for unrestricted BAs, and semi-determinized in $O(2^n)$ instead of $O(4^n)$ (cf. [16]). The class of elevator automata is quite general: it can be seen as a substantial generalization of semi-deterministic BAs (SDBAs) [11,5]. Intuitively, an elevator automaton is a BA whose strongly connected components are all either deterministic or inherently weak.

Let $\mathcal{A} = (Q, \delta, I, Q_F \cup \delta_F)$. $C \subseteq Q$ 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 C is *trivial* iff $|C| = 1$ and $\delta|_C = \emptyset$. The *condensation* of $\mathcal{A}$ is the DAG $cond(\mathcal{A}) = (\mathcal{M}, \mathcal{E})$ where $\mathcal{M}$ is the set of $\mathcal{A}$'s MSCCs and $\mathcal{E} = \{(C_1, C_2) \mid \exists q_1 \in C_1, \exists q_2 \in C_2, \exists a \in \Sigma\colon q_1 \xrightarrow{a} q_2 \in \delta\}$. An MSCC is *non-accepting* if it contains no accepting state and no accepting transition, i.e., $C \cap Q_F = \emptyset$ and $\delta|_C \cap \delta_F = \emptyset$. The *depth* of $(\mathcal{M}, \mathcal{E})$ is defined as the number of MSCCs on the longest path in $(\mathcal{M}, \mathcal{E})$.

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. C is *inherently weak* if it is either non-accepting or IWA, and $\mathcal{A}$ is inherently weak if all of its MSCCs are inherently weak. $\mathcal{A}$ is *deterministic* iff $|I| \leq 1$ and $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. An SCC $C \subseteq Q$ is *deterministic* iff $(C, \delta|_C, \emptyset, \emptyset)$ is deterministic. $\mathcal{A}$ is a *semi-deterministic BA* (SDBA) if $\mathcal{A}[q]$ is deterministic for every $q \in Q_F \cup \{p \in Q \mid s \xrightarrow{a} p \in \delta_F, s \in Q, a \in \Sigma\}$, i.e., whenever a run in $\mathcal{A}$ reaches an accepting state or an accepting transition, it can only continue deterministically.

$\mathcal{A}$ is an *elevator (Büchi) automaton* iff for every MSCC C of $\mathcal{A}$ it holds that C is (i) deterministic, (ii) IWA, or (iii) non-accepting. In other words, a BA is an elevator automaton iff every nondeterministic SCC of $\mathcal{A}$ that contains an accepting state or transition is inherently weak. An example of an elevator automaton obtained from the LTL formula $\mathsf{GF}(a \vee \mathsf{GF}(b \vee \mathsf{GF}c))$ is shown in Fig. 2. The BA consists of three connected deterministic components. Note that the automaton is neither semi-deterministic nor unambiguous.

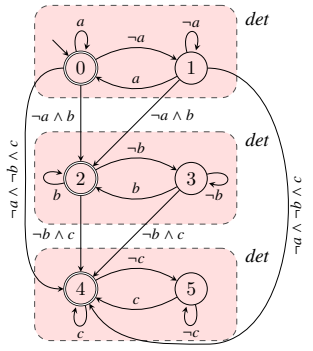


Fig. 2: The BA for LTL formula $\mathsf{GF}(a \vee \mathsf{GF}(b \vee \mathsf{GF}c))$ is elevator

The rank of an elevator automaton $\mathcal{A}$ does not depend on the number of states (as in general BAs), but only on the number of MSCCs and the depth of $cond(\mathcal{A})$. In the worst case, $\mathcal{A}$ consists of a chain of deterministic components, yielding the upper bound on the rank of elevator automata given in the following lemma.

Lemma 6. *Let $\mathcal{A}$ be an elevator automaton such that its condensation has the depth d. Then $rank(\mathcal{A}) \leq 2d$.*

4.1 Refined Ranks for Elevator Automata

Notice that the upper bound on ranks provided by Lemma 6 can still be too coarse. For instance, for an SDBA with three linearly ordered MSCCs such that the first two MSCCs are non-accepting and the last one is deterministic accepting, the lemma gives us an upper bound on the rank 6, while it is known that every SDBA has the rank at most 3 (cf. [5]). Another examples might be two deterministic non-trivial MSCCs connected by a path of trivial MSCCs, which can be assigned the same rank.

Instead of refining the definition of elevator automata into some quite complex list of constraints, we rather provide an algorithm that performs a traversal through $cond(\mathcal{A})$ and assigns each MSCC a label of the form $\boxed{type{:}rank}$ that contains (i) a type and (ii) a bound on the maximum rank of states in the component. The types of MSCCs that we consider are the following:

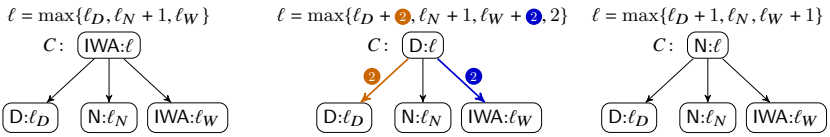
T**:** *trivial* components,
IWA**:** *inherently weak accepting* components,
D**:** *deterministic* (potentially accepting) components, and
N**:** *non-accepting* components.

Note that the type in an MSCC is not given *a priori* but is determined by the algorithm (this is because for deterministic non-accepting components, it is sometimes better to treated them as D and sometimes as N, depending on their neighbourhood). In the following, we assume that $\mathcal{A}$ is an elevator automaton without useless states and, moreover, all accepting conditions on states and transitions not inside non-trivial MSCCs are removed (any BA can be easily transformed into this form).

We start with terminal MSCCs C, i.e., MSCCs that cannot reach any other MSCC:

T1: If C is IWA, then we label it with $\boxed{\text{IWA:0}}$.

T2: Else if C is deterministic accepting, we label it with $\boxed{\text{D:2}}$.



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

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

(Note that the previous two options are complete due to our requirements on the structure of $\mathcal{A}$.) When all terminal MSCCs are labelled, we proceed through $cond(\mathcal{A})$, inductively on its structure, and label non-terminal components C based on the rules defined below.

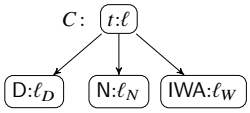


Fig. 4: Structure of elevator ranking rules

The rules are of the form that uses the structure depicted in Fig. 4, where children nodes denote already processed MSCCs. In particular, a child node of the form $k{:}\ell_k$ denotes an aggregate node of *all* siblings of the type k with ℓ_k being the maximum rank of these siblings. Moreover, we use $\text{typemax}\{e_D, e_N, e_W\}$ to denote the type $j \in \{\mathsf{D}, \mathsf{N}, \mathsf{IWA}\}$ for which $e_j = \max\{e_D, e_N, e_W\}$ where e_i is an expression containing ℓ_i (if there are more such types, j is chosen arbitrarily). The rules for assigning a type t and a rank ℓ to C are the following:

I1: If C is trivial, we set $t = \text{typemax}\{\ell_D, \ell_N, \ell_W\}$ and $\ell = \max\{\ell_D, \ell_N, \ell_W\}$.
I2: Else if C is IWA, we use the rule in Fig. 3a.
I3: Else if C is deterministic accepting, we use the rule in Fig. 3b.
I4: Else if C is deterministic and non-accepting, we try both rules from Figs. 3b and 3c and pick the rule that gives us a smaller rank.
I5: Else if C is nondeterministic and non-accepting, we use the rule in Fig. 3c.

Then, for every MSCC C of $\mathcal{A}$, we assign each of its states the rank of C. We use $\chi\colon Q \to \omega$ to denote the rank bounds computed by the procedure above.

Lemma 7. *χ is a TRUB.*

Using Lemma 5, we can now use χ to prune states during the construction of Schewe($\mathcal{A}$), as shown in the following example.

Example 8. As an example, consider the BA $\mathcal{A}$ in Fig. 1a. The set of MSCCs with their types is given as

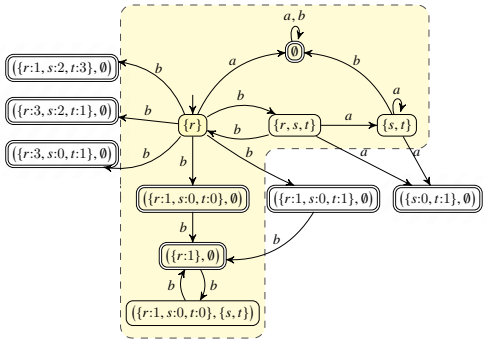


Fig. 5: A part of Schewe($\mathcal{A}$). The TRUB computed by elevator rules is used to prune states outside the yellow area.

$\{\{r\}{:}\mathsf{N}, \{s,t\}{:}\mathsf{IWA}\}$ showing that BA $\mathcal{A}$ is an elevator. Using the rules **T1** and **I4** we get the TRUB $\chi = \{r{:}1, s{:}0, t{:}0\}$. The TRUB can be used to prune the generated states as shown in Fig. 5. □

4.2 Efficient Complementation of Elevator Automata

In Section 4.1 we proposed an algorithm for assigning ranks to MSCCs of an elevator automaton $\mathcal{A}$. The drawback of the algorithm is that the maximum obtained rank is not bounded by a constant but by the depth of the condensation of $\mathcal{A}$. We will, however, show that it is actually possible to change $\mathcal{A}$ by at most doubling the number of states and obtain an elevator BA with the rank at most 3.

Intuitively, the construction copies every non-trivial MSCC C with an accepting state or transition into a component $C^\bullet$, copies all transitions going into states in C to also go into the corresponding states in $C^\bullet$, and, finally, removes all accepting conditions from C. Formally, let $\mathcal{A} = (Q, \delta, I, Q_F \cup \delta_F)$ be a BA. For $C \subseteq Q$, we use $C^\bullet$ to denote a unique copy of C, i.e., $C^\bullet = \{q^\bullet \mid q \in C\}$ s.t. $C^\bullet \cap Q = \emptyset$. Let $\mathcal{M}$ be the set of MSCCs of $\mathcal{A}$. Then, the *deelevated* BA $\textsc{DeElev}(\mathcal{A}) = (Q', \delta', I', Q'_F \cup \delta'_F)$ is given as follows:

- $Q' = Q \cup Q^\bullet$,
- $\delta' : Q' \times \Sigma \to 2^{Q'}$ where for $q \in Q$
  - $\delta'(q,a) = \delta(q,a) \cup (\delta(q,a))^\bullet$ and
  - $\delta'(q^\bullet, a) = (\delta(q,a) \cap C)^\bullet$ for $q \in C \in \mathcal{M}$;
- $I' = I$, and
- $Q'_F = Q^\bullet_F$ and $\delta'_F = \{q^\bullet \xrightarrow{a} r^\bullet \mid q \xrightarrow{a} r \in \delta_F\} \cap \delta'$.

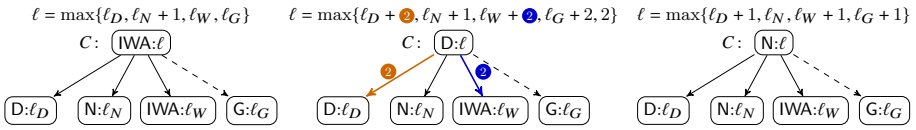
It is easy to see that the number of states of the deelevated automaton is bounded by $2|Q|$. Moreover, if $\mathcal{A}$ is elevator, so is $\textsc{DeElev}(\mathcal{A})$. The construction preserves the language of $\mathcal{A}$, as shown by the following lemma.

Lemma 9. *Let $\mathcal{A}$ be a BA. Then, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\textsc{DeElev}(\mathcal{A}))$.*

Moreover, for an elevator automaton $\mathcal{A}$, the structure of $\textsc{DeElev}(\mathcal{A})$ consists of (after trimming useless states) several non-accepting MSCCs with copied terminal deterministic or IWA MSCCs. Therefore, if we apply the algorithm from Section 4.1 on $\textsc{DeElev}(\mathcal{A})$, we get that its rank is bounded by 3, which gives the following upper bound for complementation of elevator automata.

Lemma 10. *Let $\mathcal{A}$ be an elevator automaton with suffficiently many states n. Then the language $\Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$ can be represented by a BA with at most $O(16^n)$ states.*

The complementation through $\textsc{DeElev}(\mathcal{A})$ gives a better upper bound than the rank refinement from Section 4.1 applied directly on $\mathcal{A}$, however, based on our experience, complementation through $\textsc{DeElev}(\mathcal{A})$ behaves worse in many real-world instances. This poor behaviour is caused by the fact that the complement of $\textsc{DeElev}(\mathcal{A})$ can have a larger Waiting and macrostates in Tight can have larger S-components, which can yield more generated states (despite the rank bound 3). It seems that the most promising approach would to be a combination of the approaches, which we leave for future work.



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

Fig. 6: Rules assigning types and rank bounds for non-elevator automata.

4.3 Refined Ranks for Non-Elevator Automata

The algorithm from Section 4.1 computing a TRUB for elevator automata can be extended to compute TRUBs even for general non-elevator automata (i.e., BAs with nondeterministic accepting components that are not inherently weak). To achieve this generalization, we extend the rules for assigning types and ranks to MSCCs of elevator automata from Section 4.1 to take into account general non-deterministic components. For this, we add into our collection of MSCC types *general* components (denoted as G). Further, we need to extend the rules for terminal components with the following rule:

T3: Otherwise, we label C with $\boxed{\mathsf{G}{:}2|C \setminus Q_F|}$.

Moreover, we adjust the rules for assigning a type t and a rank ℓ to C to the following (the rule **I1** is the same as for the case of elevator automata):

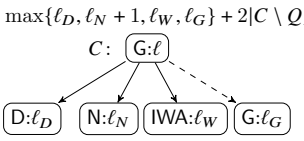


Fig. 7: C is G

I2–I5: *(We replace the corresponding rules for their counterparts including general components from Fig. 6).*

I6: Otherwise, we use the rule in Fig. 7.

Then, for every MSCC C of a BA $\mathcal{A}$, we assign each of its states the rank of C. Again, we use $\chi : Q \to \omega$ to denote the rank bounds computed by the adjusted procedure above.

Lemma 11. *χ is a TRUB.*

5 Rank Propagation

In the previous section, we proposed a way, how to obtain a TRUB for elevator automata (with generalization to general automata). In this section, we propose a way of using the structure of $\mathcal{A}$ to refine a TRUB using a propagation of values and thus reduce the size of Tight. Our approach uses *data flow analysis* [32] to reason on how ranks and rankings of macrostates of Schewe($\mathcal{A}$) can be decreased based on the ranks and rankings of the *local neighbourhood* of the macrostates. We, in particular, use a special case of *forward analysis* working on the *skeleton* of Schewe($\mathcal{A}$), which is defined as the BA $\mathcal{K}_\mathcal{A} = (2^Q, \delta', \emptyset, \emptyset)$ where $\delta' = \{R \xrightarrow{a} S \mid S = \delta(R, a)\}$ (note that we are only interested in the structure of $\mathcal{K}_\mathcal{A}$ and

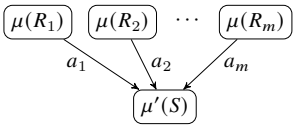


Fig. 8: Rank propagation flow

not its language; also notice the similarity of $\mathcal{K}_\mathcal{A}$ with Waiting). Our analysis refines a rank/ranking estimate $\mu(S)$ for a macrostate S of $\mathcal{K}_\mathcal{A}$ based on the estimates for its predecessors $R_1, \ldots, R_m$ (see Fig. 8). The new estimate is denoted as $\mu'(S)$.

More precisely, $\mu\colon 2^Q \to \mathbb{V}$ is a function giving each macrostate of $\mathcal{K}_\mathcal{A}$ a value from the domain $\mathbb{V}$. We will use the following two value domains: (i) $\mathbb{V} = \omega$, which is used for estimating *ranks* of macrostates (in the *outer macrostate analysis*) and (ii) $\mathbb{V} = \mathcal{R}$, which is used for estimating *rankings* within macrostates (in the *inner macrostate analysis*). For each of the analyses, we will give the *update function* $up\colon (2^Q \to \mathbb{V}) \times (2^Q)^{m+1} \to \mathbb{V}$, which defines how the value of $\mu(S)$ is updated based on the values of $\mu(R_1), \ldots, \mu(R_m)$. We then construct a system with the following equation for every $S \in 2^Q$:

$$\mu(S) = up(\mu, S, R_1, \ldots, R_m) \quad \text{where } \{R_1, \ldots, R_m\} = \delta'^{-1}(S, \Sigma). \tag{3}$$

We then solve the system of equations using standard algorithms for data flow analysis (see, e.g., [32, Chapter 2]) to obtain the fixpoint μ^*. Our analyses have the important property that if they start with μ_0 being a TRUB, then μ^* will also be a TRUB.

As the initial TRUB, we can use a trivial TRUB or any other TRUB (e.g., the output of elevator state analysis from Section 4).

5.1 Outer Macrostate Analysis

We start with the simpler analysis, which is the *outer macrostate analysis*, which only looks at sizes of macrostates. Recall that the rank r of every super-tight run in Schewe($\mathcal{A}$) does not change, i.e., a super tight run stays in Waiting as long as needed so that when it jumps to Tight, it takes the rank r and never needs to decrease it. We can use this fact to decrease the maximum rank of a macrostate S in $\mathcal{K}_\mathcal{A}$. In particular, let us consider all cycles going through S. For each of the cycles c, we can bound the maximum rank of a super-tight run going through c by $2m - 1$ where m is the smallest number of non-accepting states occurring in any macrostate on c (from the definition, the rank of a tight ranking does not depend on accepting states). Then we can infer that the maximum rank of any super-tight run going through S is bounded by the maximum rank of any of the cycles going through S (since S can never assume a higher rank in any super-tight run). Moreover, the rank of each cycle can also be estimated in a more precise way, e.g. using our elevator analysis.

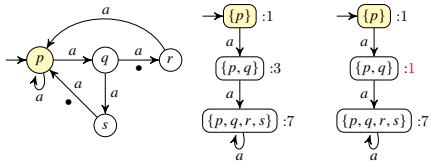
Since the number of cycles in $\mathcal{K}_\mathcal{A}$ can be large[2], instead of their enumeration, we employ data flow analysis with the value domain $\mathbb{V} = \omega$ (i.e, for every macrostate S of $\mathcal{K}_\mathcal{A}$, we remember a bound on the maximum rank of S) and the following update function:

$$up_{out}(\mu, S, R_1, \ldots, R_m) = \min\{\mu(S), \max\{\mu(R_1), \ldots, \mu(R_m)\}\}. \tag{4}$$

Intuitively, the new bound on the maximum rank of S is taken as the smaller of the previous bound $\mu(S)$ and the largest of the bounds of all predecessors of S, and the new value is propagated forward by the data flow analysis.

[2] $\mathcal{K}_\mathcal{A}$ can be exponentially larger than $\mathcal{A}$ and the number of cycles in $\mathcal{K}_\mathcal{A}$ can be exponential to the size of $\mathcal{K}_\mathcal{A}$, so the total number of cycles can be double-exponential.

Example 12. Consider the BA $\mathcal{A}_{ex}$ in Fig. 9a. When started from the initial TRUB $\mu_0 = \{\{p\} \mapsto 1, \{p,q\} \mapsto 3, \{p,q,r,s\} \mapsto 7\}$ (Fig. 9b), outer macrostate analysis decreases the maximum rank estimate for $\{p,q\}$ to 1, since $\min\{\mu_0(\{p,q\}, \max\{\mu_0(\{p\})\}\} = \min\{3,1\} = 1$. The estimate for $\{p,q,r,s\}$ is not affected, because $\min\{7, \max\{1,7\}\} = 7$ (Fig. 9c). □



(a) $\mathcal{A}_{ex}$ (b) μ_0 (c) μ^*_{out}

Fig. 9: Example of outer macrostate analysis. (a) $\mathcal{A}_{ex}$ (• denotes accepting transitions). The initial TRUB μ_0 in (b) is refined to μ^*_{out} in (c).

Lemma 13. *If μ is a TRUB, then $\mu \lhd \{S \mapsto up_{out}(\mu, S, R_1, \ldots, R_m)\}$ is a TRUB.*

Corollary 14. *When started with a TRUB μ_0, the outer macrostate analysis terminates and returns a TRUB μ^*_{out}.*

5.2 Inner Macrostate Analysis

Our second analysis, called *inner macrostate analysis*, looks deeper into super-tight runs in Schewe($\mathcal{A}$). In particular, compared with the outer macrostate analysis from the previous section—which only looks at the *ranks*, i.e., the bounds on the numbers in the rankings—, inner macrostate analysis looks at how the *rankings* assign concrete values to the *states* of $\mathcal{A}$ *inside the macrostates*.

Inner macrostate analysis is based on the following. Let ρ be a super-tight run of Schewe($\mathcal{A}$) on $\alpha \notin \mathcal{L}(\mathcal{A})$ and (S, O, f, i) be a macrostate from Tight. Because ρ is super-tight, we know that the rank $f(q)$ of a state $q \in S$ is bounded by the ranks of the predecessors of q. This holds because in super-tight runs, the ranks are only *as high as necessary*; if the rank of q were higher than the ranks of its predecessors, this would mean that we may wait in Waiting longer and only jump to q with a lower rank later.

Let us introduce some necessary notation. Let $f, f' \in \mathcal{R}$ be rankings (i.e., $f, f' \colon Q \to \omega$). We use $f \sqcup f'$ to denote the ranking $\{q \mapsto \max\{f(q), f'(q)\} \mid q \in Q\}$, and $f \sqcap f'$ to denote the ranking $\{q \mapsto \min\{f(q), f'(q)\} \mid q \in Q\}$. Moreover, we define *max-succ-rank*$^a_S(f) = \max_{\leq}\{f' \in \mathcal{R} \mid f \mathrel{\ominus\!\!\rightarrow}^a_S f'\}$ and a function $dec \colon \mathcal{R} \to \mathcal{R}$ such that $dec(\theta)$ is the ranking θ' for which

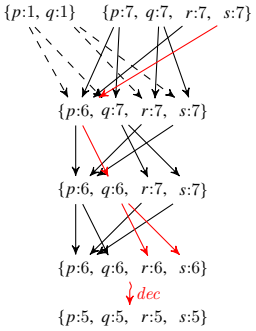
$$\theta'(q) = \begin{cases} \theta(q) \dot{-} 1 & \text{if } \theta(q) = rank(\theta) \text{ and } q \notin Q_F, \\ \llbracket \theta(q) \dot{-} 1 \rrbracket & \text{if } \theta(q) = rank(\theta) \text{ and } q \in Q_F, \\ \theta(q) & \text{otherwise.} \end{cases} \tag{5}$$

Intuitively, *max-succ-rank*$^a_S(f)$ is the (pointwise) maximum ranking that can be reached from macrostate S with ranking f over a (it is easy to see that there is a unique such maximum ranking) and $dec(\theta)$ decreases the maximum ranks in a ranking θ by one (or by two for even maximum ranks and accepting states).

The analysis uses the value domain $\mathbb{V} = \mathcal{R}$ (i.e., each macrostate of $\mathcal{K}_{\mathcal{A}}$ is assigned a ranking giving an upper bound on the rank of each state in the macrostate) and the update function up_{in} given in the right-hand side of the page. Intuitively, up_{in}

updates $\mu(q)$ for every $q \in S$ to hold the maximum rank compatible with the ranks of its predecessors. We note line Line 6, which makes use of the fact that we can only consider tight rankings (whose rank is odd), so we can decrease the estimate using the function dec defined above.

```
1 up_in(μ, S, R_1, ..., R_m):
2     foreach 1 ≤ i ≤ m and a ∈ Σ do
3         if δ(R_i, a) = S then
4             g_i^a ← max-succ-rank^a_{R_i}(μ(R_i))
5     θ ← μ(S) ⊓ ⨆{g_i^a | g_i^a is defined};
6     if rank(θ) is even then θ ← dec(θ);
7     return θ;
```



Example 15. Let us continue in Section 5.1 and perform inner macrostate analysis starting with the TRUB $\{\{p{:}1\}, \{p{:}1, q{:}1\}, \{p{:}7, q{:}7, r{:}7, s{:}7\}\}$ obtained from μ^*_{out}. We show three iterations of the algorithm for $\{p, q, r, s\}$ in the right-hand side (we do not show $\{p, q\}$ except the first iteration since it does not affect intermediate steps). We can notice that in the three iterations, we could decrease the maximum rank estimate to $\{p{:}6, q{:}6, r{:}6, s{:}6\}$ due to the accepting transitions from r and s. In the last of the three iterations, when all states have the even rank 6, the condition on Line 6 would become true and the rank of all states would be decremented to 5 using dec. Then, again, the accepting transitions from r and s would decrease the rank of p to 4, which would be propagated to q and so on. Eventually, we would arrive to the TRUB $\{p{:}1, q{:}1, r{:}1, s{:}1\}$, which could not be decreased any more, since $\{p{:}1, q{:}1\}$ forces the ranks of r and s to stay at 1. □

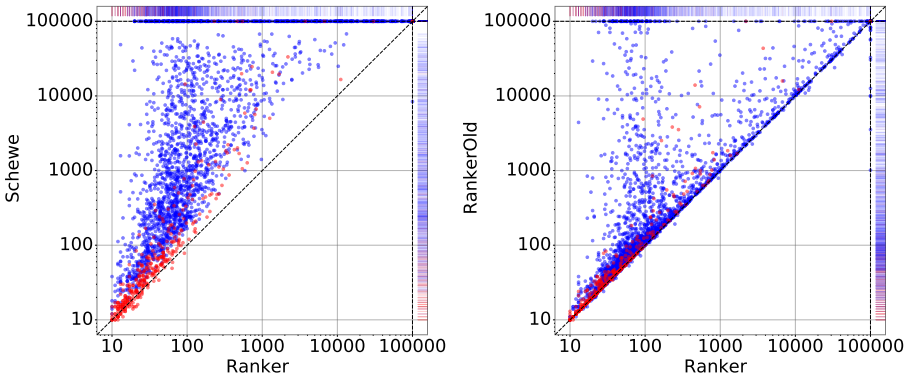
Lemma 16. *If μ is a TRUB, then $\mu \lhd \{S \mapsto up_{in}(\mu, S, R_1, \ldots, R_m)\}$ is a TRUB.*

Corollary 17. *When started with a TRUB μ_0, the inner macrostate analysis terminates and returns a TRUB μ^*_{in}.*

6 Experimental Evaluation

Used tools and evaluation environment. We implemented the techniques described in the previous sections as an extension of the tool Ranker [18] (written in C++). Speaking in the terms of [19], the heuristics were implemented on top of the Ranker$_{\text{MaxR}}$ configuration (we refer to this previous version as Ranker$_{\text{Old}}$). We tested the correctness of our implementation using Spot's `autcross` on all BAs in our benchmark. We compared modified Ranker with other state-of-the-art tools, namely, Goal [41] (implementing Piterman [34], Schewe [37], Safra [36], and Fribourg [1]), Spot 2.9.3 [12] (implementing Redziejowski's algorithm [35]), Seminator 2 [4], LTL2dstar 0.5.4 [23], and Roll [26]. All tools were set to the mode where they output an automaton with the standard state-based Büchi acceptance condition. The experimental evaluation was performed on a 64-bit GNU/Linux Debian workstation with an Intel(R) Xeon(R) CPU E5-2620 running at 2.40 GHz with 32 GiB of RAM and using a timeout of 5 minutes.

Datasets. As the source of our benchmark, we use the two following datasets: (i) `random` containing 11,000 BAs over a two letter alphabet used in [40], which were randomly



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

Fig. 10: Comparison of the state space generated by our optimizations and other rank-based procedures (horizontal and vertical dashed lines represent timeouts). Blue data points are from `random` and red data points are from `LTL`. Axes are logarithmic.

generated via the Tabakov-Vardi approach [39], starting from 15 states and with various different parameters; (ii) `LTL` with 1,721 BAs over larger alphabets (up to 128 symbols) used in [4], which were obtained from LTL formulae from literature (221) or randomly generated (1,500). We preprocessed the automata using RABIT [30] and SPOT's `autfilt` (using the `--high` simplification level), transformed them to state-based acceptance BAs (if they were not already), and converted to the HOA format [2]. From this set, we removed automata that were (i) semi-deterministic, (ii) inherently weak, (iii) unambiguous, or (iv) have an empty language, since for these automata types there exist more efficient complementation procedures than for unrestricted BAs [5,4,6,28]. In the end, we were left with **2,592** (`random`) and **414** (`LTL`) *hard* automata. We use `all` to denote their union (**3,006** BAs). Of these hard automata, 458 were elevator automata.

6.1 Generated State Space

In our first experiment, we evaluated the effectiveness of our heuristics for pruning the generated state space by comparing the sizes of complemented BAs without postprocessing. This use case is directed towards applications where postprocessing is irrelevant, such as inclusion or equivalence checking of BAs.

We focused on a comparison with two less optimized versions of the rank-based complementation procedure: SCHEWE (the version "Reduced Average Outdegree" from [37] implemented in GOAL under `-m rank -tr -ro`) and its optimization RANKER$_{\text{OLD}}$. The scatter plots in Fig. 10 compare the numbers of states of automata generated by RANKER and the other algorithms and the upper part of Table 1 gives summary statistics. Observe that our optimizations from this paper drastically reduced the generated search space compared with both SCHEWE and RANKER$_{\text{OLD}}$ (the mean for SCHEWE is lower than for RANKER$_{\text{OLD}}$ due to its much higher number of timeouts); from Fig. 10b we can see that the improvement was in many cases *exponential* even when compared with our previous optimizations in RANKER$_{\text{OLD}}$. The median (which is a more meaningful indicator with the presence of timeouts) decreased by 44 % w.r.t. RANKER$_{\text{OLD}}$, and we also reduced the

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

method	mean	median	wins	losses	mean runtime [s]	median runtime [s]	timeouts
Ranker	3812 (4452 : 207)	79 (93 : 26)			7.83 (8.99 : 1.30)	0.51 (0.84 : 0.04)	279 (276 : 3)
Ranker$_{\text{Old}}$	7398 (8688 : 358)	141 (197 : 29)	2190 (2011 : 179)	111 (107 : 4)	9.37 (10.73 : 1.99)	0.61 (1.04 : 0.04)	365 (360 : 5)
Schewe ⚽	4550 (5495 : 665)	439 (774 : 35)	2640 (2315 : 325)	55 (1 : 54)	21.05 (24.28 : 7.80)	6.57 (7.39 : 5.21)	937 (928 : 9)
Ranker	47 (52 : 18)	22 (27 : 10)			7.83 (8.99 : 1.30)	0.51 (0.84 : 0.04)	279 (276 : 3)
Piterman ⚽	73 (82 : 22)	28 (34 : 14)	1435 (1124 : 311)	416 (360 : 56)	7.29 (7.39 : 6.65)	5.99 (6.04 : 5.62)	14 (12 : 2)
Safra ⚽	83 (91 : 30)	29 (35 : 17)	1562 (1211 : 351)	387 (350 : 37)	14.11 (15.05 : 8.37)	6.71 (6.92 : 5.79)	172 (158 : 14)
Spot	75 (85 : 15)	24 (32 : 10)	1087 (936 : 151)	683 (501 : 182)	0.86 (0.99 : 0.06)	0.02 (0.02 : 0.02)	13 (13 : 0)
Fribourg ⚽	91 (104 : 13)	23 (31 : 9)	1120 (1055 : 65)	601 (376 : 225)	17.79 (19.53 : 7.22)	9.25 (10.15 : 5.48)	81 (80 : 1)
LTL2dstar	73 (82 : 21)	28 (34 : 13)	1465 (1195 : 270)	465 (383 : 82)	3.31 (3.84 : 0.11)	0.04 (0.05 : 0.02)	136 (130 : 6)
Seminator 2	79 (91 : 15)	21 (29 : 10)	1266 (1131 : 135)	571 (367 : 204)	9.51 (11.25 : 0.08)	0.22 (0.39 : 0.02)	363 (362 : 1)
Roll	18 (19 : 14)	10 (9 : 11)	2116 (1858 : 258)	569 (443 : 126)	31.23 (37.85 : 7.28)	8.19 (12.23 : 2.74)	1109 (1106 : 3)

number of timeouts by 23 %. Notice that the numbers for the `LTL` dataset do not differ as much as for `random`, witnessing the easier structure of the BAs in `LTL`.

6.2 Comparison with Other Complementation Techniques

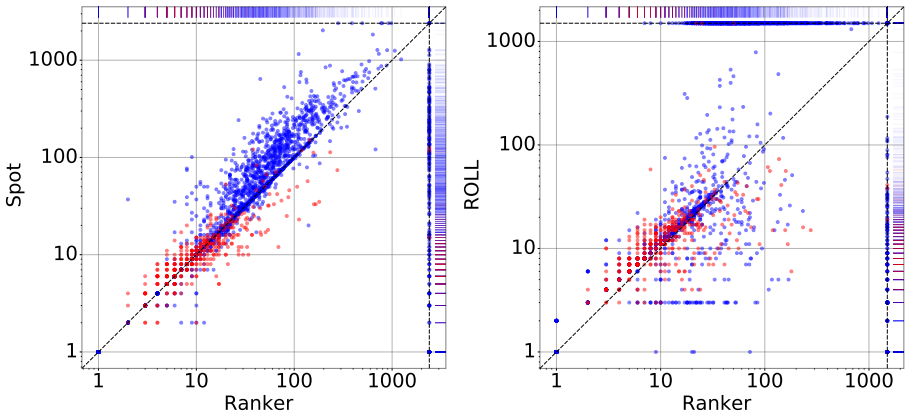
In our second experiment, we compared the improved Ranker with other state-of-the-art tools. We were comparing sizes of output BAs, therefore, we postprocessed each output automaton with `autfilt` (simplification level `--high`). Scatter plots are given in Fig. 11, where we compare Ranker with Spot (which had the best results on average from the other tools except Roll) and Roll, and summary statistics are in the lower part of Table 1. Observe that Ranker has by far the lowest mean (except Roll) and the third lowest median (after Seminator 2 and Roll, but with less timeouts). Moreover, comparing the numbers in columns **wins** and **losses** we can see that Ranker gives strictly better results than other tools (**wins**) more often than the other way round (**losses**).

In Fig. 11a see that indeed in the majority of cases Ranker gives a smaller BA than Spot, especially for harder BAs (Spot, however, behaves slightly better on the simpler BAs from `LTL`). The results in Fig. 11b do not seem so clear. Roll uses a learning-based approach—more heavyweight and completely orthogonal to any of the other tools—and can in some cases output a tiny automaton, but does not scale, as observed by the number of timeouts much higher than any other tool. It is, therefore, positively surprising that Ranker could in most of the cases still obtain a much smaller automaton than Roll.

Regarding runtimes, the prototype implementation in Ranker is comparable to Seminator 2, but slower than Spot and LTL2dstar (Spot is the fastest tool). Implementations of other approaches clearly do not target speed. We note that the number of timeouts of Ranker is still higher than of some other tools (in particular Piterman, Spot, Fribourg); further state space reduction targeting this particular issue is our future work.

7 Related Work

BA complementation remains in the interest of researchers since their first introduction by Büchi in [8]. Together with a hunt for efficient complementation techniques, the effort has been put into establishing the lower bound. First, Michel showed that the lower bound is $n!$ (approx. $(0.36n)^n$) [31] and later Yan refined the result to $(0.76n)^n$ [43].



(a) RANKER vs SPOT (b) RANKER vs ROLL

Fig. 11: Comparison of the complement size obtained by RANKER and other state-of-the-art tools (horizontal and vertical dashed lines represent timeouts). Axes are logarithmic.

The complementation approaches can be roughly divided into several branches. *Ramsey-based complementation*, the very first complementation construction, where the language of an input automaton is decomposed into a finite number of equivalence classes, was proposed by Büchi and was further enhanced in [7]. *Determinization-based complementation* was presented by Safra in [36] and later improved by Piterman in [34] and Redziejowski in [35]. Various optimizations for determinization of BAs were further proposed in [29]. The main idea of this approach is to convert an input BA into an equivalent deterministic automaton with different acceptance condition that can be easily complemented (e.g. Rabin automaton). The complemented automaton is then converted back into a BA (often for the price of some blow-up). *Slice-based complementation* tracks the acceptance condition using a reduced abstraction on a run tree [42,21]. *A learning-based approach* was introduced in [27,26]. Allred and Ultes-Nitsche then presented a novel optimal complementation algorithm in [1]. For some special types of BAs, e.g., deterministic [25], semi-deterministic [5], or unambiguous [28], there exist specific complementation algorithms. *Semi-determinization based complementation* converts an input BA into a semi-deterministic BA [11], which is then complemented [4].

Rank-based complementation, studied in [24,15,14,37,22], extends the subset construction for determinization of finite automata by storing additional information in each macrostate to track the acceptance condition of all runs of the input automaton. Optimizations of an alternative (sub-optimal) rank-based construction from [24] going through *alternating Büchi automata* were presented in [15]. Furthermore, the work in [22] introduces an optimization of SCHEWE, in some cases producing smaller automata (this construction is not compatible with our optimizations). As shown in [9], the rank-based construction can be optimized using simulation relations. We identified several heuristics that help reducing the size of the complement in [19], which are compatible with the heuristics in this paper.

Acknowledgements. We thank anonymous reviewers 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. Allred, J.D., Ultes-Nitsche, U.: 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). pp. 46–55. IEEE Computer Society Press (July 2018)
2. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J., Müller, D., Parker, D., Strejček, J.: The Hanoi omega-automata format. In: Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9206, pp. 479–486. Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_31
3. Blahoudek, F., Heizmann, M., Schewe, S., Strejček, J., Tsai, M.H.: Complementing semi-deterministic büchi automata. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 770–787. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
4. Blahoudek, F., Duret-Lutz, A., Strejček, J.: Seminator 2 can complement generalized Büchi automata via improved semi-determinization. In: Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV'20). Lecture Notes in Computer Science, vol. 12225, pp. 15–27. Springer (Jul 2020)
5. Blahoudek, F., Heizmann, M., Schewe, S., Strejček, J., Tsai, M.: Complementing semi-deterministic Büchi automata. In: 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. Lecture Notes in Computer Science, vol. 9636, pp. 770–787. Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_49
6. Boigelot, B., Jodogne, S., Wolper, P.: On the use of weak automata for deciding linear arithmetic with integer and real variables. In: Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2083, pp. 611–625. Springer (2001). https://doi.org/10.1007/3-540-45744-5_50
7. Breuers, S., Löding, C., Olschewski, J.: Improved Ramsey-based Büchi complementation. In: Proc. of FOSSACS'12. pp. 150–164. Springer (2012)
8. 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)
9. Chen, Y., Havlena, V., Lengál, O.: Simulations in rank-based Büchi automata complementation. In: Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11893, pp. 447–467. Springer (2019). https://doi.org/10.1007/978-3-030-34175-6_23
10. 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 the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018. pp. 135–150. ACM (2018). https://doi.org/10.1145/3192366.3192405
11. Courcoubetis, C., Yannakakis, M.: 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. pp. 338–345. IEEE Computer Society (1988). https://doi.org/10.1109/SFCS.1988.21950
12. 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: Automated Technology for Verification and Analysis. pp. 122–129. Springer International Publishing, Cham (2016)

13. Fogarty, S., Vardi, M.Y.: Büchi complementation and size-change termination. In: Proc. of TACAS'09. pp. 16–30. Springer (2009)
14. Friedgut, E., Kupferman, O., Vardi, M.: Büchi complementation made tighter. International Journal of Foundations of Computer Science **17**, 851–868 (2006)
15. Gurumurthy, S., Kupferman, O., Somenzi, F., Vardi, M.Y.: On complementing nondeterministic Büchi automata. In: 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. LNCS, vol. 2860, pp. 96–110. Springer (2003). https://doi.org/10.1007/978-3-540-39724-3_10
16. Havlena, V., Lengál, O., Smahlíková, B.: Sky is not the limit: Tighter rank bounds for elevator automata in Büchi automata complementation (technical report). CoRR **abs/2110.10187** (2021), https://arxiv.org/abs/2110.10187
17. Havlena, V., Lengál, O., Šmahlíková, B.: Deciding S1S: Down the rabbit hole and through the looking glass. In: Proceedings of NETYS'21. pp. 215–222. No. 12754 in LNCS, Springer Verlag (2021). https://doi.org/10.1007/978-3-030-91014-3_15
18. Havlena, V., Lengál, O., Šmahlíková, B.: Ranker (2021), https://github.com/vhavlena/ranker
19. Havlena, V., Lengál, O.: Reducing (To) the Ranks: Efficient Rank-Based Büchi Automata Complementation. In: Proc. of CONCUR'21. LIPIcs, vol. 203, pp. 2:1–2:19. Schloss Dagstuhl, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.CONCUR.2021.2, iSSN: 1868-8969
20. Heizmann, M., Hoenicke, J., Podelski, A.: Termination analysis by learning terminating programs. In: Proc. of CAV'14. pp. 797–813. Springer (2014)
21. Kähler, D., Wilke, T.: Complementation, disambiguation, and determinization of Büchi automata unified. In: Proc. of ICALP'08. pp. 724–735. Springer (2008)
22. Karmarkar, H., Chakraborty, S.: On minimal odd rankings for Büchi complementation. In: Proc. of ATVA'09. LNCS, vol. 5799, pp. 228–243. Springer (2009). https://doi.org/10.1007/978-3-642-04761-9_18
23. Klein, J., Baier, C.: On-the-fly stuttering in the construction of deterministic *omega* -automata. In: Proc. of CIAA'07. LNCS, vol. 4783, pp. 51–61. Springer (2007). https://doi.org/10.1007/978-3-540-76336-9_7
24. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Log. **2**(3), 408–429 (2001). https://doi.org/10.1145/377978.377993
25. Kurshan, R.P.: Complementing deterministic Büchi automata in polynomial time. J. Comput. Syst. Sci. **35**(1), 59–71 (1987). https://doi.org/10.1016/0022-0000(87)90036-5
26. Li, Y., Sun, X., Turrini, A., Chen, Y., Xu, J.: ROLL 1.0: ω-regular language learning library. In: Proc. of TACAS'19. LNCS, vol. 11427, pp. 365–371. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_23
27. Li, Y., Turrini, A., Zhang, L., Schewe, S.: Learning to complement Büchi automata. In: Proc. of VMCAI'18. pp. 313–335. Springer (2018)
28. Li, Y., Vardi, M.Y., Zhang, L.: On the power of unambiguity in Büchi complementation. In: Proc. of GandALF'20. EPTCS, vol. 326, pp. 182–198. Open Publishing Association (2020). https://doi.org/10.4204/EPTCS.326.12
29. Löding, C., Pirogov, A.: New optimizations and heuristics for determinization of büchi automata. In: Automated Technology for Verification and Analysis. pp. 317–333. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-31784-3_18
30. Mayr, R., Clemente, L.: Advanced automata minimization. In: Proc. of POPL'13. pp. 63–74 (2013)
31. Michel, M.: Complementation is more difficult with automata on infinite words. CNET, Paris **15** (1988)
32. Nielson, F., Nielson, H.R., Hankin, C.: Principles of program analysis. Springer (1999). https://doi.org/10.1007/978-3-662-03811-6

33. 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
34. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proc. of LICS'06. pp. 255–264. IEEE (2006)
35. Redziejowski, R.R.: An improved construction of deterministic omega-automaton using derivatives. Fundam. Informaticae **119**(3-4), 393–406 (2012). https://doi.org/10.3233/FI-2012-744
36. Safra, S.: On the complexity of ω-automata. In: Proc. of FOCS'88. pp. 319–327. IEEE (1988)
37. Schewe, S.: Büchi complementation made tight. In: Proc. of STACS'09. LIPIcs, vol. 3, pp. 661–672. Schloss Dagstuhl (2009). https://doi.org/10.4230/LIPIcs.STACS.2009.1854
38. 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), 217–237 (1987)
39. Tabakov, D., Vardi, M.Y.: Experimental evaluation of classical automata constructions. In: Proc. of LPAR'05. pp. 396–411. Springer (2005)
40. Tsai, M.H., Fogarty, S., Vardi, M.Y., Tsay, Y.K.: State of Büchi complementation. In: Implementation and Application of Automata. pp. 261–271. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
41. Tsai, M.H., Tsay, Y.K., Hwang, Y.S.: GOAL for games, omega-automata, and logics. In: Computer Aided Verification. pp. 883–889. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
42. Vardi, M.Y., Wilke, T.: Automata: From logics to algorithms. Logic and Automata **2**, 629–736 (2008)
43. Yan, Q.: Lower bounds for complementation of ω-automata via the full automata technique. In: Automata, Languages and Programming. pp. 589–600. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)