



Simple Complementation of Generalized Büchi Automata*

David Dokoupil and Jan Strejček

Masaryk University, Brno, Czech Republic
{`xdokoup1, strejcek`}@fi.muni.cz

Abstract

Transition-based generalized Büchi automata (TGBAs) can be substantially more compact than equivalent Büchi automata, and they are produced or processed by most state-of-the-art ω -automata tools. Research on complementation algorithms for ω -automata focuses mainly on standard Büchi automata, with a few recent exceptions. We present a complementation construction applicable directly to TGBAs based on the idea of “simple and optimal” Büchi automata complementation by Allred and Ultes-Nitsche. We also provide an upper bound on the size of the complement. The bound is asymptotically better than the upper bounds of other complementation approaches applicable to TGBAs. Experimental evaluation shows that on complex random TGBAs with many acceptance sets, our complementation clearly outperforms the complementation processes available in state-of-the-art tools including Spot and Kofola.

1 Introduction

Automata over infinite words have applications in many areas of computer science, for example, in decision procedures for several logics [11, 24], LTL model checking [43], and control synthesis [36]. Various kinds of automata have been considered, ranging from the basic kinds like Büchi, Rabin, or parity automata, to automata with generic acceptance called Emerson-Lei automata [17, 5]. More complex kinds of automata can often represent languages with smaller automata, which leads to faster executions of automata-based algorithms.

Complementation is a fundamental automata operation. Over the last sixty years, several conceptually different approaches to the complementation of Büchi automata have been introduced and gradually improved, namely *Ramsey-based complementation* [11, 41, 10], *determinization-based complementation* [37, 35, 31], and *rank-based complementation* [26, 42, 38, 19, 22, 21]. There also exist complementation constructions that do not follow any of these approaches [30, 32, 3]. Further, there are complementation constructions designed specifically for Büchi automata with certain structural or other properties which can be more efficient than general complementation constructions. For example, there are specific constructions for inherently-weak [34], deterministic [29], semi-deterministic [9, 13], or unambiguous [33, 18] automata. There is also a technique that combines some of the mentioned constructions by applying the most appropriate constructions to individual automata components [20].

*This work has been supported by the Czech Science Foundation grant GA26-22640S.

The research on complementation of more general automata classes attracted substantially less attention. Rank-based complementation has been extended to *generalized Büchi automata* (GBAs) [28, 23], to Streett and Rabin automata [27, 12], and recently also to generalized Rabin automata and some subclasses of Emerson-Lei automata [23]. Further, there also exists a complementation procedure for *transition-based generalized Büchi automata* (TGBAs) where a given TGBA is first semi-determinized [8], and a complementation procedure for parity automata [40]. A very recent contribution to the field is a complementation technique for Emerson-Lei automata [2] that generalizes the complementation technique [20] for Büchi automata mentioned above: a given automaton is transformed into an equivalent *elevator* automaton and its complement is then constructed from the complements of individual components of this elevator automaton, which are obtained using selected complementation constructions.

In this paper, we extend the “simple and optimal” complementation construction [3] designed for Büchi automata to TGBAs. A TGBA contains finitely many sets of accepting transitions and its run is accepting if and only if it uses some transition of each of these sets infinitely often. This automata class is natively supported by many tools producing or consuming automata, including Spot [16], LTL3BA [6], and Seminor [8].

Further, we show that for a TGBA with n states and k sets of accepting transitions, our construction produces a complement automaton with $\mathcal{O}((\frac{kn}{2.718})^n)$ states. This is asymptotically better than the best upper bound on the size of a TGBA complement known so far, namely $\mathcal{O}(n(0.76kn)^n)$ [23]. In comparison, if we first degeneralize a given TGBA into an equivalent *transition-based Büchi automaton* (TBA) with kn states and then apply the “simple and optimal” complementation [3], we would get the upper bound $\mathcal{O}((0.7645kn)^{kn})$.

Our complementation construction produces TBAs, which are TGBAs with a single set of accepting transitions. A TBA can be easily transformed into an equivalent Büchi automaton with at most twice as many states. As every GBA can be seen as a TGBA, the upper bound $\mathcal{O}((\frac{kn}{2.718})^n)$ holds also for the complementation of GBAs and it is very close to the lower bound for GBA complementation $\Omega((\frac{kn}{e})^n)$ identified by Schewe and Varghese [39].

We have implemented the presented complementation construction in a tool named SIMCO. We compared our tool against the following state-of-the-art tools which use different approaches to TGBA complementation. Among the following tools, SIMCO is the only one to complement TGBAs directly. As far as we know, constructions [28, 23] have not been implemented.

- AUTFILT of the library Spot [16] first checks if the input TGBA is very weak. If this is the case, it complements the automaton using dualization (replacing nondeterministic choice with universal transitions) and dealternation (done by the powerset construction). In the opposite case, the TGBA is transformed into an equivalent deterministic parity automaton which is then complemented.
- KOFOLA [20] decomposes a given automaton into strongly connected components, each component is processed by the best-fitting construction, and the produced automata are joined into a complement.
- SEMINATOR 2 [8] first transforms a given TGBA into an equivalent semi-deterministic TBA and then applies the complementation method for semi-deterministic automata [9].
- Finally, our tool SIMCO implements the described construction working directly on TGBAs and also the complementation process where a given TGBA is first transformed into an equivalent TBA, which is then complemented basically by the original complementation algorithm by Allred and Ultes-Nitsche [3].

For the experimental evaluation, we used randomly generated TGBAs and “practical” TGBAs obtained from formulae of the LTL-synthesis competition SYNTCOMP [25]. The experimental evaluation shows that on randomly generated automata, our tool produces smaller complements than the other tools, in some cases even by two orders of magnitude. On “practical” automata, the results were not conclusive.

Contribution The main contributions of this paper are the following.

- We generalize the complementation construction of Büchi automata by Allred and Ultes-Nitsche [3] to TGBAs, including the key concept of *greedy* runs and correctness arguments.
- We derive an asymptotic upper bound on the size of the produced complement, which is lower than the bounds of current TGBA complementation techniques.
- We present two practical optimizations of the generalized complementation technique, namely *level jumping* and *component merging*. The first one is inspired by similar optimizations used in TGBA degeneralization [4] and semideterminization [8]. The latter one is a straightforward generalization of an optimization introduced with the original complementation for Büchi automata [3].
- The TGBA complementation and the optimizations are implemented in a publicly available open-source tool called SIMCO, which was experimentally evaluated together with the established tools mentioned above.

All these items were previously presented in the master’s thesis of the first author [14]. The presentation given here is heavily revised, simplifies the notation, and streamlines both the correctness arguments and the complexity analysis. The experimental evaluation has been redone with newer versions of the tools, a larger set of randomly generated automata, and extended with the automata obtained from SYNTCOMP.

Structure of the paper Section 2 recalls the basic automata terminology. Section 3 then presents the complementation construction by Allred and Ultes-Nitsche [3] adjusted to TBAs. The TGBA complementation and its complexity analysis are described in Sections 4 and 5, respectively. Section 6 presents two optimizations of the complementation construction. Section 7 is devoted to the implementation and experimental evaluation of the TGBA complementation technique. The paper ends with the conclusion in Section 8.

2 Preliminaries

In this section we recall some basic terms and notation, including the definitions of a *transition-based generalized Büchi automaton* (TGBA) and a *transition-based Büchi automaton* (TBA).

Let s be a finite sequence of elements of an arbitrary type. We use $|s|$ to denote its length, and ε to denote the sequence of length 0. For any infinite sequence $s = s_0s_1\dots$ and $i \geq 0$, by $s_{<i}$ we denote its prefix $s_0\dots s_{i-1}$ of length i .

Definition 1. A transition-based generalized Büchi automaton (TGBA) is a tuple $\mathcal{G} = (Q, \Sigma, \delta, q_{in}, \mathcal{T})$, where Q is a finite set of states, Σ is a finite set of symbols called alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $q_{in} \in Q$ is a distinguished initial state, and $\mathcal{T} = \{\textcircled{0}, \textcircled{1}, \dots, \textcircled{m}\}$ is a nonempty finite set. Each $\textcircled{i} \in \mathcal{T}$ is an acceptance set of accepting transitions $\textcircled{i} \subseteq \delta$. If $\mathcal{T}$ is a singleton, the automaton is also called a transition-based Büchi

automaton (TBA) and instead of $\mathcal{T} = \{\bullet\}$ we identify $\mathcal{T}$ with $\bullet$. An automaton is called deterministic if for each $p \in Q$ and $a \in \Sigma$ there exists at most one q satisfying $(p, a, q) \in \delta$.

We sometimes use δ as a function $\delta : Q \times \Sigma \rightarrow 2^Q$ defined as $\delta(p, a) = \{q \mid (p, a, q) \in \delta\}$. A finite word v over Σ is a finite sequence of symbols $v = v_0v_1 \dots v_{n-1} \in \Sigma^*$. An (infinite) word w over Σ is an infinite sequence $w = w_0w_1 \dots \in \Sigma^\omega$. We extend the function δ to sets of states and to finite words. Formally, we define $\delta : 2^Q \times \Sigma^* \rightarrow 2^Q$ recursively as

- $\delta(S, \varepsilon) = S$ and
- $\delta(S, va) = \bigcup_{p \in \delta(S, v)} \delta(p, a)$ for any $v \in \Sigma^*$ and $a \in \Sigma$.

Finally, for each set $\mathfrak{i} \in \mathcal{T}$ we define an auxiliary function $\delta_{\mathfrak{i}} : 2^Q \times \Sigma \rightarrow 2^Q$ that follows only the transitions in $\mathfrak{i}$, i.e., $\delta_{\mathfrak{i}}(S, a) = \bigcup_{p \in S} \{q \mid (p, a, q) \in \mathfrak{i}\}$.

A finite run of $\mathcal{G}$ over a nonempty finite word $v = v_0v_1 \dots v_n \in \Sigma^{n+1}$ is a sequence of transitions $\sigma = (q_0, v_0, q_1)(q_1, v_1, q_2) \dots (q_n, v_n, q_{n+1}) \in \delta^{n+1}$ such that $q_0 = q_{in}$. By σ_o we denote the last state q_{n+1} of the finite run.

An infinite run (or simply a run) of $\mathcal{G}$ over an infinite word $w = w_0w_1 \dots \in \Sigma^\omega$ is an infinite sequence of transitions $\rho = (q_0, w_0, q_1)(q_1, w_1, q_2) \dots \in \delta^\omega$ such that $q_0 = q_{in}$. Let inf_ρ be the set of transitions that appear infinitely many times in ρ . The run is *accepting* if $\text{inf}_\rho \cap \mathfrak{i} \neq \emptyset$ for each $\mathfrak{i} \in \mathcal{T}$. We say that $\mathcal{G}$ *accepts* w if there exists an accepting run of $\mathcal{G}$ over w . Finally, we define the *language* of $\mathcal{G}$ as the set of all infinite words over Σ accepted by $\mathcal{G}$, i.e.,

$$L(\mathcal{G}) = \{w \in \Sigma^\omega \mid \exists \text{ accepting run of } \mathcal{G} \text{ over } w\}.$$

An automaton $\mathcal{C}$ is a *complement automaton* (complement for short) of $\mathcal{G}$ if it uses the same alphabet Σ and $L(\mathcal{C}) = \Sigma^\omega \setminus L(\mathcal{G})$. The process of creating the complement automaton is called *complementation*.

3 Complementation of TBAs

This section describes the complementation construction by Allred and Ultes-Nitsche [3] translated into the transition-based setting. While the original construction handles state-based Büchi automata with a total transition function (i.e., each state has a successor under each alphabet symbol), we adjust the construction to TBAs with an arbitrary transition relation.

For the whole section, we consider $\mathcal{B} = (Q, \Sigma, \delta, q_{in}, \bullet)$ to be the input TBA for the complementation. The cornerstone of the complementation construction is the definition of *greedy runs* over an infinite word w . An infinite run is *greedy* if each of its prefixes is greedy. Intuitively, we say that a finite run σ over v is greedy if there is no other finite run over v leading to the same state that takes accepting transitions sooner than σ .

Definition 2. A score function *scr* is the function assigning to each finite run $\sigma = \sigma_0\sigma_1 \dots \sigma_{n-1} \in \delta^n$ of $\mathcal{B}$ the binary number $\text{scr}(\sigma) = \gamma_0\gamma_1 \dots \gamma_{n-1} \in \{0, 1\}^n$ of the same length defined as

$$\gamma_i = \begin{cases} 1 & \text{if } \sigma_i \in \bullet, \\ 0 & \text{otherwise.} \end{cases}$$

Further, let $\leq_2$ be the standard ordering on integers in big-endian binary representation (the leftmost digit is the most significant). A finite run σ of $\mathcal{B}$ over v is greedy if for each finite run σ' of $\mathcal{B}$ over v it holds

$$\sigma_o = \sigma'_o \implies \text{scr}(\sigma') \leq_2 \text{scr}(\sigma).$$

An infinite run ρ of $\mathcal{B}$ is greedy if each nonempty finite prefix σ of ρ is greedy.

Figure 1 provides an example of a TBA and its three finite runs α, β, λ over aa . The runs α and β both end in q_0 , but since $\text{scr}(\alpha) >_2 \text{scr}(\beta)$, the run α is greedy and β is not. The run λ is greedy because it is the only run over aa that ends in the state q_1 .

It can be proven that a TBA $\mathcal{B}$ accepts a word w if and only if there exists a greedy accepting run of $\mathcal{B}$ over w . We prove an analogous statement for TGBAs in the next section. Thanks to this observation, the complementation construction can track only greedy runs or, more precisely, finite greedy runs as prefixes of potential infinite greedy runs.

The complement automaton consists of two parts, the *upper automaton* and the *lower automaton*, connected by *transfer transitions*. Intuitively, if no greedy run over the input word is accepting, then the complement automaton waits in the upper part until all infinite greedy runs stop using accepting transitions. Then it nondeterministically moves to the lower part and checks that whenever a tracked finite greedy run uses an accepting transition, it cannot be extended into an infinite greedy run over the input word. We first construct the upper and lower automata, which are both deterministic. We mark the components of the upper and lower automata by accents ‘ $\hat{}$ ’ and ‘ $\check{}$ ’, respectively.

The only task of the upper automaton is to track finite greedy runs over the read part of the input word. The *upper automaton* is the TBA $\hat{\mathcal{B}} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_{in}, \emptyset)$ with empty acceptance set. The elements of $\hat{Q}$ are called *macrostates*. Intuitively, a macrostate contains the states of Q reachable over the read input and divided into groups according to the score of finite greedy runs leading to them. The groups are called *components*, and they are ordered by the scores of the corresponding finite greedy runs. Formally, a macrostate $\hat{q} = \hat{q}[0]\hat{q}[1]\dots\hat{q}[k-1]$ is a sequence of $k \geq 0$ components, where each component $\hat{q}[i]$ is a subset of Q . Components of each macrostate of $\hat{Q}$ are nonempty and pairwise disjoint. The first component $\hat{q}[0]$ corresponds

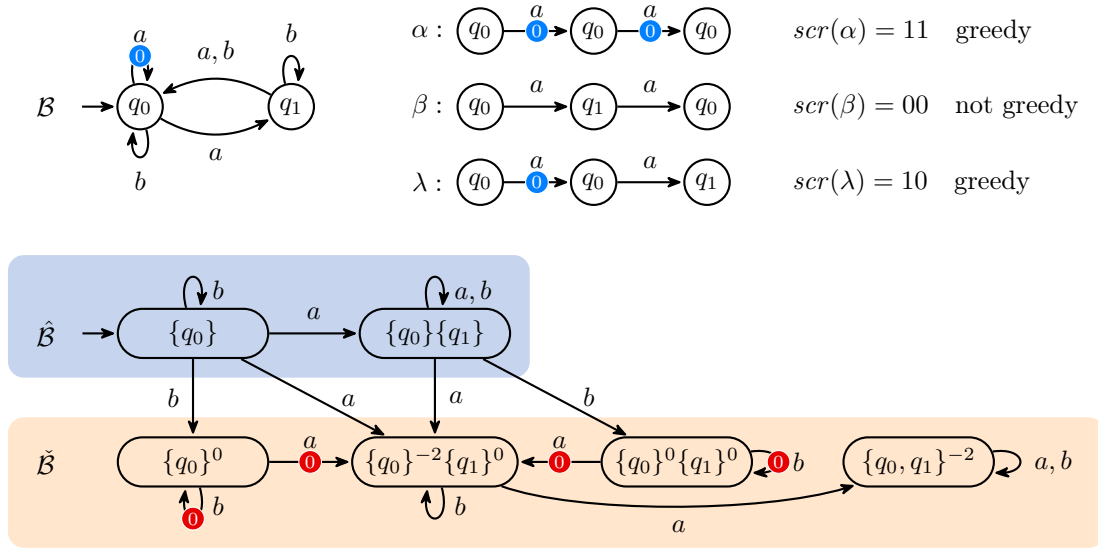


Figure 1: An example TBA $\mathcal{B}$ (top left), all finite runs of $\mathcal{B}$ over aa with their respective scores and greediness status (top right), and the complement of $\mathcal{B}$ (bottom) with the upper automaton marked by the blue area, the lower automaton by the orange area, and transfer transitions between them (unreachable parts of the complement are omitted)

to the highest score. By $\triangleleft \hat{q}[i] = \bigcup_{j=0}^{i-1} \hat{q}[j]$ we denote the union of the components preceding $\hat{q}[i]$. The initial macrostate $\hat{q}_{in} = \{q_{in}\}$ contains just one component with the initial state of $\mathcal{B}$.

For each $\hat{p} \in \hat{Q}$ and $a \in \Sigma$, there is one transition $(\hat{p}, a, \hat{q}) \in \hat{\delta}$. We construct $\hat{q}$ with use of an interim macrostate $\dot{q}$. First, the successors of states in each component $\hat{p}[i]$ are divided into two components $\dot{q}[2i], \dot{q}[2i+1]$, where $\dot{q}[2i]$ contains the successors obtained by an accepting transition (and thus with a higher score), and $\dot{q}[2i+1]$ contains the remaining successors. At the same time, we remove from each component $\dot{q}[j]$ the states of $\triangleleft \dot{q}[j]$ as these are reachable by another finite run with a higher score. Formally, $\dot{q}$ has twice as many components as $\hat{p}$, where

$$\dot{q}[i] = \begin{cases} \delta_{\bullet}(\hat{p}[\frac{i}{2}], a) \setminus \triangleleft \dot{q}[i] & \text{if } i \text{ is even,} \\ (\delta(\hat{p}[\frac{i-1}{2}], a) \setminus \delta_{\bullet}(\hat{p}[\frac{i-1}{2}], a)) \setminus \triangleleft \dot{q}[i] = \delta(\hat{p}[\frac{i-1}{2}], a) \setminus \triangleleft \dot{q}[i] & \text{otherwise.} \end{cases}$$

Second, $\hat{q}$ is constructed from $\dot{q}$ by removing all empty components.

An example of a construction of an upper automaton is provided in Figure 1. For example, consider the transition from $\hat{q}_{in} = \{q_0\}$ over a . The single component $\{q_0\}$ generates $\dot{q}[0] = \delta_{\bullet}(\{q_0\}, a) \setminus \triangleleft \dot{q}[0] = \{q_0\}$ and $\dot{q}[1] = \delta(\{q_0\}, a) \setminus \triangleleft \dot{q}[1] = \{q_1\}$. The resulting macrostate $\hat{q} = \{q_0\}\{q_1\}$ reflects that the greedy run over a to q_0 has a higher score than the one to q_1 .

The lower automaton is constructed in a similar way. It tracks finite greedy runs and checks that whenever some of these runs use accepting transitions, they do not extend to infinite greedy runs. To this end, we add to each component a *level* $l \in \{0, -1, -2\}$. The components with level 0 are *clear* components. They track finite greedy runs of the original automaton that did not use any accepting transition since the computation entered the lower automaton. The second category are *tainted* components with level -1 or -2 . They track finite greedy runs that passed an accepting transition after the lower automaton was entered. The complement automaton accepts only if every such greedy run eventually terminates, i.e., it cannot be prolonged under the current input symbol or it ceases to be greedy. This is checked by a *breakpoint* construction. Components with level -1 and -2 are called *on-hold* and *terminal*, respectively. If any terminal component has a nonempty successor, all newly tainted components become on-hold. When all terminal components disappear, the lower automaton marks the transition as accepting and all on-hold components become terminal.

The *lower automaton* $\tilde{\mathcal{B}} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_{in}, \bullet)$ is a TBA with the set $\tilde{Q}$ of macrostates of the form $\tilde{q}[0]\tilde{q}[1] \dots \tilde{q}[k-1]$, where $k \geq 0$ and each component $\tilde{q}[i] \in 2^Q \times \{0, -1, -2\}$ contains a set of states referred to as $\tilde{q}[i].S$ and a level referred to as $\tilde{q}[i].\ell$. Again, in each macrostate $\tilde{q} \in \tilde{Q}$, the sets $\tilde{q}[i].S$ are nonempty and pairwise disjoint. By $\triangleleft \tilde{q}[i] = \bigcup_{j=0}^{i-1} \tilde{q}[j].S$ we denote the union of the state sets in the components preceding $\tilde{q}[i]$. For brevity, we sometimes write the levels as the superscript of the state set of each component. Since the initial state $\tilde{q}_{in}$ of the lower automaton is ignored in the final complement construction, it can be chosen arbitrarily.¹

For each $\tilde{p} \in \tilde{Q}$ and $a \in \Sigma$, there is one transition $(\tilde{p}, a, \tilde{q}) \in \tilde{\delta}$, which is again constructed with use of an interim macrostate $\dot{q}$. The macrostate $\dot{q}$ has twice as many components as $\tilde{p}$. Their state sets are defined as in the upper automaton with one change: tainted components are not split as all greedy runs they represent have to terminate anyway. Instead, all relevant successors of states in a tainted component $\tilde{p}[i]$ are in $\dot{q}[2i].S$, while $\dot{q}[2i+1].S$ is empty and $\dot{q}[2i+1]$ will be removed in $\tilde{q}$. Formally, we set

$$\dot{q}[i].S = \begin{cases} \delta_{\bullet}(\tilde{p}[\frac{i}{2}].S, a) \setminus \triangleleft \dot{q}[i] & \text{if } i \text{ is even and } \tilde{p}[\frac{i}{2}].\ell = 0, \\ \delta(\tilde{p}[\frac{i}{2}].S, a) \setminus \triangleleft \dot{q}[i] & \text{otherwise.} \end{cases}$$

¹Note that the complementation construction assumes that both $\hat{Q}$ and $\tilde{Q}$ contain all possible macrostates. A practical implementation constructs only the macrostates reachable from $\hat{q}_{in}$ via $\hat{\delta}$, $\tilde{\delta}$, and transfer transitions.

To simplify the level assignment, we define a predicate $\text{isBP}(\tilde{p}, a)$ indicating whether the transition from $\tilde{p}$ over a is a breakpoint. This happens when all terminal components of $\tilde{p}$ turn into empty components during this transition. Formally, $\text{isBP}(\tilde{p}, a)$ is true iff for all components $\tilde{p}[i]$ satisfying $\tilde{p}[i].\ell = -2$, it holds that $\tilde{q}[2i].S = \emptyset$. The level of each component $\tilde{q}[i]$ is defined as

$$\tilde{q}[i].\ell = \begin{cases} -2 & \text{if } \tilde{p}[\lfloor \frac{i}{2} \rfloor].\ell = -1 \wedge \text{isBP}(\tilde{p}, a), \\ -2 & \text{if } \tilde{p}[\lfloor \frac{i}{2} \rfloor].\ell = 0 \wedge i \text{ is even} \wedge \text{isBP}(\tilde{p}, a), \\ -1 & \text{if } \tilde{p}[\lfloor \frac{i}{2} \rfloor].\ell = 0 \wedge i \text{ is even} \wedge \neg \text{isBP}(\tilde{p}, a), \\ \tilde{p}[\lfloor \frac{i}{2} \rfloor].\ell & \text{otherwise.} \end{cases}$$

Again, $\tilde{q}$ is obtained from $\tilde{q}$ by removing all components with an empty set of states. The accepting transitions are those with a breakpoint, i.e., $\bullet = \{(\tilde{p}, a, \tilde{q}) \in \check{\delta} \mid \text{isBP}(\tilde{p}, a)\}$.

Finally, the complement automaton consists of the upper and lower automata, and *transfer transitions* leading from the upper to the lower automaton. Given a macrostate $\hat{q} \in \hat{Q}$, by $\text{zero}(\hat{q})$ we denote the macrostate of $\check{Q}$ arising from $\hat{q}$ by extending all its components with level 0. Formally, the complement automaton is the TBA $\mathcal{C} = (Q_{\mathcal{C}}, \Sigma, \delta_{\mathcal{C}}, \hat{q}_{in}, \bullet)$ with states $Q_{\mathcal{C}} = \hat{Q} \uplus \check{Q}$ and transition relation $\delta_{\mathcal{C}} = \hat{\delta} \cup \check{\delta} \cup \{(\hat{p}, a, \tilde{q}) \mid \hat{p} \in \hat{Q}, a \in \Sigma, \tilde{q} \in \check{\delta}(\text{zero}(\hat{p}), a)\}$.

We illustrate the construction on the complement in Figure 1. For example, consider the transition of the lower automaton from $\tilde{q} = \{q_0\}^0$ over a . We get $\tilde{q}[0].S = \{q_0\}$ and $\tilde{q}[1].S = \{q_1\}$. Since $\tilde{q}$ has no terminal components, $\text{isBP}(\tilde{q}, a)$ holds trivially. The assigned levels are $\tilde{q}[0].\ell = -2$ and $\tilde{q}[1].\ell = 0$, giving us the target state $\{q_0\}^{-2}\{q_1\}^0$. The transition is accepting. As $\text{zero}(\{q_0\}) = \{q_0\}^0$, there is also a transfer transition over a leading from $\{q_0\} \in \hat{Q}$ to $\{q_0\}^{-2}\{q_1\}^0 \in \check{Q}$.

4 Complementation of TGBAs

We now extend the TBA complementation to TGBAs, which constitutes one of our main contributions. We use $\mathcal{G} = (Q, \Sigma, \delta, q_{in}, \mathcal{T})$ as the input for the complementation.

The definition of greedy runs for TGBAs is also based on a score function. While the score function for TBAs records every accepting transition in a given run, the score function for TGBAs is rather based on the following observation. A run $\rho_0\rho_1 \dots \in \delta^\omega$ of a TGBA with sets of accepting transitions $\mathcal{T} = \{\bullet, \dots, \bullet\}$ is accepting if and only if there exists an infinite increasing sequence of integers $0 \leq n_0 < n_1 < n_2 < \dots$ such that for each $j \geq 0$ it holds $\rho_{n_j} \in \bullet$ where $i = j \bmod |\mathcal{T}|$. The score function tracks the progress towards acceptance by recording the first occurrence of a transition from $\bullet$ in the run, then the first succeeding occurrence of a transition from $\bullet$, and so on. Once it records the occurrence of $\bullet$, it looks again for the first succeeding occurrence of a transition in $\bullet$, and the situation repeats. Later we refer to this moment by saying that the run completed a full pass through $\mathcal{T}$. We define the score function for the considered TGBA $\mathcal{G}$ as follows.

Definition 3. A score function scr is the function assigning to each finite run $\sigma = \sigma_0\sigma_1 \dots \sigma_{n-1} \in \delta^n$ of $\mathcal{G}$ the $(n+1)$ -ary number $\text{scr}(\sigma) = \gamma_0\gamma_1 \dots \gamma_{n-1} \in \{0, \dots, n\}^n$ of the same length defined as

$$\gamma_0 = \begin{cases} 1 & \text{if } \sigma_0 \in \bullet, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \quad \gamma_i = \begin{cases} \gamma_{i-1} + 1 & \text{if } \sigma_i \in \bullet \text{ for } l = \gamma_{i-1} \bmod |\mathcal{T}|, \\ \gamma_{i-1} & \text{otherwise,} \end{cases}$$

for each $1 \leq i < n$. Let $\leq_{n+1}$ be the standard ordering on integers in big-endian $(n+1)$ -ary

representation (the leftmost digit is the most significant). A finite run σ of $\mathcal{G}$ over v is greedy if for each finite run σ' of $\mathcal{G}$ over v it holds

$$\sigma_o = \sigma'_o \implies \text{scr}(\sigma') \leq_{n+1} \text{scr}(\sigma).$$

An infinite run ρ of $\mathcal{G}$ is greedy if each nonempty finite prefix σ of ρ is greedy.

Figure 2 provides an example of a TGBA and its four finite runs $\alpha, \beta, \lambda, \mu$ over aab . The runs α, β, λ end in q_0 . The runs α and β have the same score even though α contains one more transition from $\textcircled{0}$. The run λ contains one transition from $\textcircled{1}$, but since no transition from $\textcircled{0}$ occurs before it, the score does not increment. The runs α, β have a higher score than λ and are greedy, while λ is not. The run μ is the only run over aab leading to q_1 and thus it is greedy.

Theorem 1. Let $\mathcal{G}$ be a TGBA and $w \in \Sigma^\omega$ be an infinite word. An accepting run of $\mathcal{G}$ over w exists if and only if a greedy accepting run of $\mathcal{G}$ over w exists.

Proof. As the implication ‘ $\Leftarrow$ ’ holds trivially, we focus on the opposite one. Let w be a word accepted by $\mathcal{G}$ and Ψ_0 be the set of all accepting runs of $\mathcal{G}$ over w . For each $i > 0$, we set

$$\Psi_i = \{\rho \mid \rho \in \Psi_0 \text{ and } \rho_{<i} \text{ is a finite greedy run}\}.$$

The definition of greedy runs implies that every nonempty prefix of a finite greedy run is also greedy. Hence, we have a sequence $\Psi_0 \supseteq \Psi_1 \supseteq \Psi_2 \supseteq \dots$ of sets of accepting runs over w . We prove that all these sets are nonempty. For the sake of contradiction, assume that Ψ_i is the first empty set in the sequence. We know that Ψ_0 is nonempty because $w \in L(\mathcal{G})$. Hence, $i > 0$.

Let ρ be an arbitrary run from Ψ_{i-1} . As Ψ_i is empty, we know that $\rho_{<i}$ is not greedy. Hence, there exists another finite run over the same word as $\rho_{<i}$ ending in the same state with a strictly higher score. Since the set of finite runs over $w_{<i}$ ending in $(\rho_{<i})_o$ is finite, there exists a finite run in this set with the maximal score. Let σ be such a run. Note that σ is greedy by definition. If we replace in run ρ the prefix $\rho_{<i}$ by σ , we get an accepting run ρ' over w . As $\rho'_{<i} = \sigma$ is a finite greedy run, we know that $\rho' \in \Psi_i$, which is a contradiction with $\Psi_i = \emptyset$.

Next we construct a tree T with the root ε (0-th level) such that its nodes on levels $i > 0$ are the prefixes $\rho_{<i}$ of runs $\rho \in \Psi_i$. A parent of each node $\rho_{<i}$ on level i is its prefix $\rho_{<i-1}$. As the relation δ is finite, T is finitely branching. Further, at each level $i > 0$, we remove from the tree the nodes that do not have the maximum score among all nodes on the level i . Crucially, every node with the maximum score at level i must be a child of a node with the maximum score at level $i - 1$ (from the big-endian ordering of scores). Let T^* be the pruned

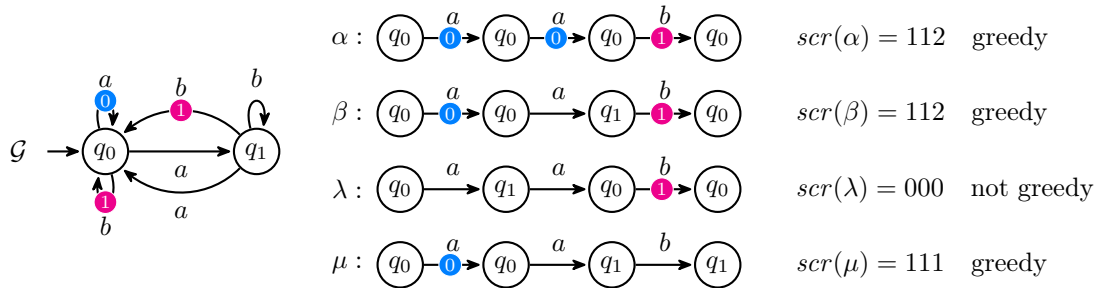


Figure 2: An example TGBA $\mathcal{G}$ with $\mathcal{T} = \{\textcircled{0}, \textcircled{1}\}$ (left) and all runs of $\mathcal{G}$ over aab with their respective scores and greediness status (right)

tree. Note that tree T^* is infinite as every level of the original tree has to have some nodes with the maximum score. König's lemma implies that T^* has an infinite branch. Let ρ^* be an infinite run corresponding to this branch. Then ρ^* is greedy because each of its nonempty prefixes $(\rho^*)_{<i}$ is a prefix of a run in Ψ_i . Since all nodes of T are prefixes of accepting runs, the last digit of the maximum score at each level grows without any bound. As prefixes of ρ^* achieve the maximum score at every level, the last digit of their scores is also unbounded, meaning that ρ^* visits each acceptance set infinitely often. Thus, ρ^* is an accepting greedy run over w . $\square$

The structure of the complement is the same as in the TBA case: it consists of two deterministic automata, the *upper* and the *lower*, connected by *transfer transitions*. The purpose of these automata remains unchanged. However, both automata now use components extended with levels in order to track finite greedy runs. We use the accents ‘ $\hat{\cdot}$ ’ and ‘ $\check{\cdot}$ ’ as previously.

The *upper automaton* is the TBA $\hat{\mathcal{G}} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_{in}, \emptyset)$, where elements of $\hat{Q}$ are *macrostates*. A macrostate $\hat{q} = \hat{q}[0]\hat{q}[1] \dots \hat{q}[k-1]$ is a sequence of $k \geq 0$ components, where each component $\hat{q}[i]$ is a pair of a state set $\hat{q}[i].S \subseteq Q$ and a level $\hat{q}[i].\ell \in \{0, 1, \dots, |\mathcal{T}| - 1\}$. The state sets of the components in each macrostate $\hat{q}$ are pairwise disjoint and nonempty. We again use $\triangleleft \hat{q}[i] = \bigcup_{j=0}^{i-1} \hat{q}[j].S$ to denote the set of states contained in the components preceding $\hat{q}[i]$. The initial macrostate $\hat{q}_{in}$ consists of one component $\{q_{in}\}$ with level 0, written also as $\{q_{in}\}^0$.

The level of a component says which set of accepting transitions would increase the score of the finite greedy runs represented by the component. The successors obtained by a transition of this set form a separate component with the level increased by one. If the new level would be $|\mathcal{T}|$, the level is *reset* to 0. For each macrostate $\hat{p} \in \hat{Q}$ and each symbol $a \in \Sigma$, there is one transition $(\hat{p}, a, \hat{q}) \in \hat{\delta}$. We construct the target macrostate $\hat{q}$ with use of an interim macrostate $\check{q}$ with twice as many components as $\hat{p}$. The components of $\check{q}$ are defined as

$$\begin{aligned} \check{q}[i].S &= \begin{cases} \delta_{\bullet}(\hat{p}[\frac{i}{2}].S, a) \setminus \triangleleft \check{q}[i] & \text{if } i \text{ is even, where } l = \hat{p}[\frac{i}{2}].\ell, \\ \delta(\hat{p}[\frac{i-1}{2}].S, a) \setminus \triangleleft \check{q}[i] & \text{otherwise,} \end{cases} \\ \check{q}[i].\ell &= \begin{cases} (\hat{p}[\frac{i}{2}].\ell + 1) \bmod |\mathcal{T}| & \text{if } i \text{ is even,} \\ \hat{p}[\frac{i-1}{2}].\ell & \text{otherwise.} \end{cases} \end{aligned}$$

The macrostate $\hat{q}$ is obtained from $\check{q}$ by removing all components with an empty set of states.

The *lower automaton* is the TBA $\check{\mathcal{G}} = (\check{Q}, \Sigma, \check{\delta}, \check{q}_{in}, \mathbf{0})$, where elements of $\check{Q}$ are again macrostates. These macrostates are defined as in the lower automaton of the TBA complement with a single difference: the levels range over $\{-2, -1, 0, \dots, |\mathcal{T}| - 1\}$. The components with nonnegative levels are called *clear* and they function as in the upper automaton until the level is reset to 0. This moment is analogous to passing an accepting transition in the TBA setting. Hence, the complement automaton accepts only if all reset components of the lower automaton eventually terminate. Such components are called *tainted*, and instead of resetting their level to 0, we set it to -1 or -2 using the same breakpoint approach as in the TBA complement. We do not specify the initial state $\check{q}_{in}$ as it plays no role in the complement.

For each $\check{p} \in \check{Q}$ and $a \in \Sigma$ we construct one transition $(\check{p}, a, \check{q}) \in \check{\delta}$ with use of an interim macrostate $\check{q}$, which has twice as many components as $\check{p}$. The definition of $\check{q}[i].S$ combines the principle of the upper automaton $\hat{\mathcal{G}}$ with the fact that tainted components are not split:

$$\check{q}[i].S = \begin{cases} \delta_{\bullet}(\check{p}[\frac{i}{2}].S, a) \setminus \triangleleft \check{q}[i] & \text{if } i \text{ is even and } \check{p}[\frac{i}{2}].\ell \geq 0, \text{ where } l = \check{p}[\frac{i}{2}].\ell, \\ \delta(\check{p}[\frac{i}{2}].S, a) \setminus \triangleleft \check{q}[i] & \text{otherwise.} \end{cases}$$

In the level assignment, we use the predicate $\text{isBP}(\check{p}, a)$ as defined in the TBA complementation. We introduce two new predicates: $\text{isReset}(\check{p}, i)$ is true iff i is even and $\check{p}[\frac{i}{2}].\ell = |\mathcal{T}| - 1$, and $\text{isLevelUp}(\check{p}, i)$ is true iff i is even and $0 \leq \check{p}[\frac{i}{2}].\ell < |\mathcal{T}| - 1$. The levels $\check{q}[i].\ell$ are defined as

$$\check{q}[i].\ell = \begin{cases} -2 & \text{if } \check{p}[\lfloor \frac{i}{2} \rfloor].\ell = -1 \wedge \text{isBP}(\check{p}, a), \\ -2 & \text{if } \text{isReset}(\check{p}, i) \wedge \text{isBP}(\check{p}, a), \\ -1 & \text{if } \text{isReset}(\check{p}, i) \wedge \neg \text{isBP}(\check{p}, a), \\ \check{p}[\frac{i}{2}].\ell + 1 & \text{if } \text{isLevelUp}(\check{p}, i), \\ \check{p}[\lfloor \frac{i}{2} \rfloor].\ell & \text{otherwise.} \end{cases}$$

Again, $\check{q}$ is obtained from $\check{q}$ by removing all components with an empty set of states. The accepting transitions $\textcircled{0} = \{(\check{p}, a, \check{q}) \in \check{\delta} \mid \text{isBP}(\check{p}, a)\}$ are defined as in the TBA complement.

Finally, the complement of $\mathcal{G}$ is the TBA $\mathcal{C} = (Q_{\mathcal{C}}, \Sigma, \delta_{\mathcal{C}}, \hat{q}_{in}, \textcircled{0})$, where $Q_{\mathcal{C}} = \hat{Q} \uplus \check{Q}$ (macrostates of each part are formally distinct) and $\delta_{\mathcal{C}} = \hat{\delta} \cup \check{\delta} \cup \{(\hat{p}, a, \check{q}) \mid \hat{p} \in \hat{Q}, a \in \Sigma, \check{q} \in \check{\delta}(\hat{p}, a) \text{ where } \hat{p} \text{ is the same macrostate as } \check{p} \text{ but in } \check{Q}\}$. The last set of the union defining $\delta_{\mathcal{C}}$ represents transfer transitions.

Theorem 2. *Let $\mathcal{G}$ be a TGBA. Then $L(\mathcal{C}) = \Sigma^{\omega} \setminus L(\mathcal{G})$.*

Proof (Sketch). The definition of $\delta_{\mathcal{C}}$ guarantees that all greedy runs of $\mathcal{G}$ over w are properly tracked by each run of $\mathcal{C}$ over w . Indeed, every successor of a state $q \in \hat{p}[i].S$ lies in one of the two components derived from $\hat{p}[i]$, or in an earlier component (the same holds for $\check{p}$). Therefore, no finite greedy run is lost. Further, a finite greedy prefix $\rho_{<i}$ of an infinite greedy run ρ is never absorbed into an earlier component, since that would require a finite run over the same word as $\rho_{<i}$ with a higher score ending in the same state, contradicting the greediness of ρ . Hence, the component tracking a finite prefix of an infinite greedy run ρ is always derived from the component tracking the prefix of ρ shorter by one step. Moreover, the level of each component of the upper automaton and each clear component of the lower automaton equals the last digit of the score of the finite greedy runs tracked by the component modulo $|\mathcal{T}|$, which follows from the definition of $\hat{\delta}$ and $\check{\delta}$ by induction on the length of the tracked runs.

First, let $w \in \Sigma^{\omega}$ be a word accepted by $\mathcal{G}$. We prove $w \notin L(\mathcal{C})$. By Theorem 1 there exists an accepting greedy run ρ of $\mathcal{G}$ over w . For the sake of contradiction let $\bar{\rho}$ be an accepting run of $\mathcal{C}$ over w . We have that ρ is tracked by $\bar{\rho}$. From the definition of $\mathcal{C}$, there must exist a position j such that $\bar{\rho}_j$ is a transfer transition, otherwise $\bar{\rho}$ could not reach any accepting transitions, as all accepting transitions are located in the lower automaton. Further, there must exist $k > j$ such that the component tracking the prefix $\rho_{<k}$ becomes tainted because ρ eventually completes a full pass through $\mathcal{T}$. Recall that tainted components are not split and as ρ is infinite, this component has one nonempty successor at every step. If some successor of this component in $\bar{\rho}$ becomes terminal, all its future successors are terminal as well, and $\bar{\rho}$ never reaches a breakpoint from then on. In the other case the tainted components tracking prefixes of ρ never become terminal, which means that no breakpoint occurs in the future. In both cases, $\bar{\rho}$ cannot be an accepting run, which is a contradiction.

Second, assume $w \notin L(\mathcal{G})$. We construct an accepting run $\bar{\rho}$ of $\mathcal{C}$ over w . By Theorem 1, no accepting greedy run of $\mathcal{G}$ over w exists, and hence every infinite greedy run over w completes only finitely many full passes through $\mathcal{T}$. This moreover happens uniformly: there exists $j > 0$ such that each infinite greedy run ρ over w completes all its full passes through $\mathcal{T}$ within its prefix $\rho_{<j}$. To show that such j exists, we focus on the run of the upper deterministic automaton over w . More precisely, we focus on components in this run that track prefixes of some infinite

greedy runs ρ of $\mathcal{G}$ over w . We call such components *surviving*. The first paragraph of this proof implies that each surviving component has at least one successor component that is also surviving. Thus, the number of surviving components in a macrostate cannot decrease by any transition of $\hat{\delta}$. Further, each macrostate contains at most $|Q|$ components. Therefore, the number of surviving components in the macrostates of the run of the upper automaton stabilizes at some point and in each future step, every surviving component has exactly one surviving successor. If there existed an infinite chain of subsequent surviving components such that the component level increases infinitely many times, there would be an infinite greedy run of $\mathcal{G}$ completing infinitely many full passes through $\mathcal{T}$. This run would be accepting, contradicting $w \notin L(\mathcal{G})$. Hence, there are at most $|Q|$ infinite chains of subsequent surviving components, and the component level in each chain remains unchanged after some number of steps. We can choose j as the maximum of these numbers of steps.

Let $\bar{\rho}$ be the run of $\mathcal{C}$ over w that follows $\hat{\delta}$ for the first j steps and then takes the transfer transition $\bar{\rho}_j$ to the lower automaton. The construction of $\mathcal{C}$ implies that $\bar{\rho}$ is fully determined by j and all its transitions after $\bar{\rho}_j$ are transitions of $\check{\delta}$. A component in the macrostates of $\bar{\rho}$ becomes tainted only if the finite greedy runs tracked by this component complete a full pass through $\mathcal{T}$. By the choice of j , no such finite greedy run can be extended into an infinite greedy run. Therefore, each tainted component eventually becomes empty.

If a macrostate $\check{\rho}$ of $\bar{\rho}$ contains no tainted components, the transition from $\check{\rho}$ is a breakpoint and thus accepting. If $\check{\rho}$ contains some tainted components, at least one of them is terminal. All terminal components eventually become empty and all newly tainted components are on-hold until then. When the last terminal component of $\check{\rho}$ becomes empty, a breakpoint is reached and the corresponding transition is accepting. Hence, an accepting transition is reached from each $\check{\rho}$ of $\bar{\rho}$. This means that $\bar{\rho}$ is an accepting run of $\mathcal{C}$ over w and $w \in L(\mathcal{C})$. $\square$

5 Complexity Analysis

We now analyze the complexity of our TGBA complementation. In particular, we focus on the number of states of the complement.

First, we recall some relevant combinatorial concepts. A *partition* of an arbitrary set S is a decomposition into nonempty pairwise disjoint sets $S_0, S_1, \dots, S_l$ such that their union is S . A *Stirling partition number* $\left\{ \begin{smallmatrix} m \\ p \end{smallmatrix} \right\} = \sum_{i=0}^p \frac{(-1)^{p-i} \cdot i^m}{(p-i)! \cdot i!}$ is the number of ways a set of m elements can be divided into p nonempty pairwise disjoint subsets. The number of partitions of a set with m elements is given by the *m -th Bell number* $B_m = \sum_{p=0}^m \left\{ \begin{smallmatrix} m \\ p \end{smallmatrix} \right\}$. Lastly, the *m -th ordered Bell number* $a_m = \sum_{p=0}^m p! \cdot \left\{ \begin{smallmatrix} m \\ p \end{smallmatrix} \right\}$ is the number of ways a set of m elements can be partitioned when the order of the parts is considered.

In the rest of this section, $n = |Q|$ stands for the number of states of the TGBA $\mathcal{G}$ and $k = |\mathcal{T}|$ for the number of its acceptance sets. We construct an upper bound on the number of macrostates in the lower automaton. In our computation, we ignore the empty macrostate $\check{q} = \varepsilon$ as it does not play any role in the asymptotic complexity. The same bound will apply also to the macrostates in the upper automaton.

Each nonempty macrostate of the lower automaton contains $1 \leq m \leq n$ states, chosen from the n states of $\mathcal{G}$ in $\binom{n}{m}$ ways. The m states are divided into $p \leq m$ components which form a partition of the m states. The p components are ordered in one of $p!$ ways and each component

has an assigned level ranging over $\{-2, -1, \dots, k-1\}$. Hence, we get

$$|\check{Q}| \leq \sum_{m=1}^n \binom{n}{m} \sum_{p=0}^m p! \left\{ \begin{matrix} m \\ p \end{matrix} \right\} (k+2)^p.$$

To simplify the expression, we employ the m -th λ -colored Bell number a_m^λ as introduced by Allred and Ultes-Nitsche in [3]. The number itself is defined as

$$a_m^\lambda = \sum_{p=0}^m p! \left\{ \begin{matrix} m \\ p \end{matrix} \right\} \lambda^p, \quad \text{with} \quad a_m^\lambda \approx \frac{\sqrt{2\pi m}}{(\lambda+1) \cdot \ln(\frac{\lambda+1}{\lambda})} \cdot \left(\frac{1}{\ln(\frac{\lambda+1}{\lambda})} \cdot m \right)^m$$

as its asymptotic approximation. By using an additional approximation (based on Taylor expansion, which holds asymptotically as $\lambda \rightarrow \infty$) $\ln(\frac{\lambda+1}{\lambda}) = \ln(1 + \frac{1}{\lambda}) \approx \frac{1}{\lambda}$ we simplify the approximation to

$$a_m^\lambda \approx \frac{\sqrt{2\pi m} \cdot \lambda}{(\lambda+1)} \cdot \left(\frac{\lambda}{e} \cdot m \right)^m.$$

Substituting the m -th λ -colored Bell number in our bounds gives us

$$|\check{Q}| \leq \sum_{m=1}^n \binom{n}{m} \cdot a_m^{k+2}.$$

Further, Allred and Ultes-Nitsche [3] showed that

$$\sum_{m=1}^n \binom{n}{m} \cdot a_m^\lambda = \frac{\lambda+1}{\lambda} \cdot a_n^\lambda - 1.$$

If we replace a_n^λ by its approximation, we obtain

$$\sum_{m=1}^n \binom{n}{m} a_m^\lambda \approx \frac{\lambda+1}{\lambda} \cdot \frac{\sqrt{2\pi n} \cdot \lambda}{(\lambda+1)} \cdot \left(\frac{\lambda}{e} \cdot n \right)^n = \sqrt{2\pi n} \cdot \left(\frac{\lambda \cdot n}{e} \right)^n.$$

Finally, substituting the expression above into our bound on $|\check{Q}|$ results in

$$|\check{Q}| \leq \sqrt{2\pi n} \left(\frac{(k+2) \cdot n}{e} \right)^n.$$

Therefore, we conclude that the upper bound on the number of states of the automaton produced by the presented TGBA complementation is

$$|Q_C| \in \mathcal{O} \left(n^{\frac{1}{2}} \left(\frac{(k+2) \cdot n}{e} \right)^n \right).$$

We can relax the bound by observing that $e \approx 2.71828 > 2.718$. Thus, $\frac{1}{e} < \frac{1}{2.718}$, which allows us to discard the factor outside the exponential to obtain

$$|Q_C| \in \mathcal{O} \left(\left(\frac{(k+2) \cdot n}{2.718} \right)^n \right).$$

Alternatively, for sufficiently large k , the same gap between e and 2.718 is large enough to absorb both the polynomial factor and the reduction of $k + 2$ to k in the base, yielding

$$|Q_C| \in \mathcal{O} \left(\left(\frac{kn}{2.718} \right)^n \right).$$

The paper [39] shows that the complementation problem for (state-based) generalized Büchi automata has a lower bound of at least $\Omega \left(\left(\frac{kn}{e} \right)^n \right)$. Therefore, our construction is near-optimal with respect to this established lower bound².

6 Optimizations

This section presents two optimizations of the complementation technique introduced in Section 4. The first technique is called *level jumping* and it is specific to TGBAs. The second technique, called *component merging*, was introduced by Allred and Ultes-Nitsche [3] for TBA complementation, and it can be easily adapted to TGBAs as well. Both techniques lead to the production of smaller (on average) complement automata, and they can be combined together.

The optimization called *level jumping* is inspired by similar optimizations in TGBA degeneralization [4] and semideterminization [8]. Intuitively, in the transition construction of both the upper and the lower automaton, we enable components to increase their level by more than one if they are obtained by a transition lying in several consecutive acceptance sets. Formally, the change is only in the construction of the interim macrostate. Given a macrostate $\hat{p}$ of the upper automaton and $a \in \Sigma$, the interim macrostate $\dot{q}$ is constructed such that for each component $\hat{p}[i]$ it contains $k = |\mathcal{T}| + 1$ components $\dot{q}[ki], \dot{q}[ki + 1], \dots, \dot{q}[ki + |\mathcal{T}|]$. Each $\dot{q}[ki + j]$ is defined as

$$\begin{aligned} \dot{q}[ki + j].S &= \begin{cases} \delta_{l,j}(\hat{p}[i].S, a) \setminus \triangleleft \dot{q}[ki + j] & \text{if } 0 \leq j < |\mathcal{T}|, \text{ where } l = \hat{p}[i].\ell, \\ \delta(\hat{p}[i].S, a) \setminus \triangleleft \dot{q}[ki + j] & \text{otherwise,} \end{cases} \\ \dot{q}[ki + j].\ell &= \begin{cases} (\hat{p}[i].\ell + |\mathcal{T}| - j) \bmod |\mathcal{T}| & \text{if } 0 \leq j < |\mathcal{T}|, \\ \hat{p}[i].\ell & \text{otherwise,} \end{cases} \end{aligned}$$

$$\text{where } \delta_{l,j} = \bigcap_{i \in \{l, (l+1) \bmod |\mathcal{T}|, \dots, (l+|\mathcal{T}|-j-1) \bmod |\mathcal{T}|\}} \textcircled{i}$$

is the set of transitions that are in $|\mathcal{T}| - j$ succeeding acceptance sets starting with $\textcircled{i}$ and counted cyclically.

The construction of the lower automaton is similar. The state sets $\dot{q}[ki + j].S$ of individual components are defined as above, only $\hat{p}$ is replaced by $\check{p}$ and the condition $0 \leq j < |\mathcal{T}|$ is conjoined with $\check{p}[i].\ell \geq 0$ such that tainted components are not divided. For the level assignments, we first replace the predicate $\text{isReset}(\check{p}, i)$ by the predicate $\text{isReset}(\check{p}, i, j)$ valid if and only if

²By replacing 2.718 with a closer decimal underapproximation of e , the bound approaches $\mathcal{O} \left(\left(\frac{kn}{e} \right)^n \right)$, at the cost of requiring a larger threshold on k .

$\check{p}[i].\ell + (|\mathcal{T}| - j) \geq |\mathcal{T}|$. The actual assignment is given by

$$\check{q}[ki + j].\ell = \begin{cases} -2 & \text{if } \check{p}[i].\ell = -1 \wedge \text{isBP}(\check{p}, a), \\ -2 & \text{if } \text{isReset}(\check{p}, i, j) \wedge \text{isBP}(\check{p}, a), \\ -1 & \text{if } \text{isReset}(\check{p}, i, j) \wedge \neg \text{isBP}(\check{p}, a), \\ (\check{p}[i].\ell + |\mathcal{T}| - j) \bmod |\mathcal{T}| & \text{if } \check{p}[i].\ell \geq 0 \wedge \neg \text{isReset}(\check{p}, i, j), \\ \check{p}[i].\ell & \text{otherwise.} \end{cases}$$

To prove the correctness of this optimization, one needs to modify the score function in Definition 3 such that it increases by the value from 0 to $|\mathcal{T}|$ saying how many succeeding acceptance sets the used transition belongs to. Formally, for each finite run $\sigma = \sigma_0 \sigma_1 \dots \sigma_{n-1}$,

- γ_0 is the maximal $0 \leq j \leq |\mathcal{T}|$ such that $\sigma_0 \in \textcolor{blue}{\bullet}$ for each $l \in \{0, 1, \dots, j-1\}$,
- if $\sigma_i \in \textcolor{blue}{\bullet}$ for $l = \gamma_{i-1} \bmod |\mathcal{T}|$, then $\gamma_i = \gamma_{i-1} + j$ where $1 \leq j \leq |\mathcal{T}|$ is the maximal number such that $\sigma_i \in \textcolor{blue}{\bullet}$ for each $l \in \{\gamma_{i-1} \bmod |\mathcal{T}|, (\gamma_{i-1} + 1) \bmod |\mathcal{T}|, \dots, (\gamma_{i-1} + j - 1) \bmod |\mathcal{T}|\}$, and
- if $\sigma_i \notin \textcolor{blue}{\bullet}$ for $l = \gamma_{i-1} \bmod |\mathcal{T}|$, then $\gamma_i = \gamma_{i-1}$.

This modification of the scoring function changes the set of greedy runs such that it again holds that the complement automaton with level jumping tracks the finite greedy runs with the same score in the same component. The rest of the correctness arguments are then similar to the unoptimized case.

Component merging can merge adjacent tainted components in macrostates of the lower automaton. This reduces the number of macrostates in the lower automaton as two macrostates can be transformed into the same one. More precisely, whenever a macrostate $\check{q} \in \check{Q}$ contains two adjacent components $\check{q}[i]\check{q}[i+1]$ satisfying either

$$\check{q}[i].\ell = \check{q}[i+1].\ell = l \text{ and } l \in \{-2, -1\}, \quad \text{or} \quad \check{q}[i].\ell = -1 \text{ and } \check{q}[i+1].\ell = -2,$$

the components are merged into one component with the set of states $\check{q}[i].S \cup \check{q}[i+1].S$ and the level $\min(\check{q}[i].\ell, \check{q}[i+1].\ell)$. This operation is repeated until no more components can be merged. Note that the resulting macrostate does not depend on the merging order. This optimization is correct for the reasons given by Allred and Ultes-Nitsche [3] in the context of TBA complementation.

7 Implementation and Experimental Evaluation

Our complementation construction is implemented in a prototype tool named **SIMCo** (**Simple Complementation**). The tool is written in Python 3.14.4 using the Spot 2.14.5 library [15]. It is available under the GNU GPLv3 license together with the experimental evaluation presented in this section at <https://github.com/DavidDokoupil/SimCo>.

The entry point is the file `complement.py`, which expects a path to a file containing one or more TGBAs in the HOA format [5]. The complements are written to standard output again in the HOA format. By default, input automata are preprocessed using Spot's function `scc_filter`, which removes dead states and also acceptance from transitions outside accepting SCCs. The following flags can be used to enable some optimizations.

- Flags `--jump` and `--merge` enable the optimizations *level jumping* and *component merging* described in Section 6, respectively.

- Flag `--post-process` reduces the produced complement by Spot’s automata simplification technique on the `--low` setting.

The following evaluation uses two benchmark sets, both focused on TGBAs with at least two acceptance sets. The first set called RANDOM contains automata randomly generated by the tool `randaut` of Spot. We used the flags `-A2`, `-A3`, `-A4`, `-A5` to produce TGBAs with two to five acceptance sets. For each of these flags, we generated 200 TGBAs divided into four groups of 50 automata with 2, 3, 4, and 5 atomic propositions. We always used `-Q2..7` to get automata with two to seven states, and `-e0.5` to control the density of the generated automata. In total, we generated 800 random automata. Finally, we called `autfilt` to remove automata representing the empty language. We were left with 491 TGBAs in RANDOM.

The second benchmark set, called SYNTCOMP, is based on the automata translated from LTL formulae used for the LTL-synthesis competition SYNTCOMP [25]. The formulae were converted from TLSF files of the TLSF benchmark repository³ using the tool `syfco`⁴ and default instances for the parametric formulae. The dataset contained 959 unique formulae which we translated using the tool `ltl2tgba` of Spot with a one-minute timeout. Spot was compiled with the flag `--enable-max-accsets=256` to support up to 256 acceptance sets, which has a negative performance impact. We discarded automata representing the empty language and those with a single acceptance set. The final dataset SYNTCOMP contains 209 TGBAs.

We evaluated two configurations of our tool. One referred to simply as SIMCO uses the flags `--jump --merge --post-process`. The second configuration referred to as SIMCO-DEGEN is in fact a pipeline that first degeneralizes a given input TGBA to a TBA using `autfilt -b` and then runs SIMCO with flags `--merge --post-process`. We also evaluated three state-of-the-art complementation tools, namely AUTFLT of the Spot library [15], KOFOLA [1], and SEMINATOR 2 [7]. These tools were run with the same postprocessing configuration. Note that the other tools were not restricted to producing TBAs and may output automata with more general acceptance conditions, which can result in fewer states. The comparison of state counts is therefore favorable to them. We chose not to restrict them, as we regard the ability to output automata with a more general acceptance condition as a genuine strength of those tools.

The evaluation was done on a CachyOS Linux system equipped with an i7-13700F processor and 64 GB of RAM. We used Spot’s `autcross` to gather statistics and, on a subset of the results, to verify the correctness of the complementation. The correctness check was restricted to the complements produced by SIMCO and was not run on all cases, as it is resource intensive (`autcross` was run with a 16 GB memory limit). For the benchmark RANDOM, 481 results out of 491 were verified. In the remaining 10, `autcross` exceeded the memory limit. For the benchmark SYNTCOMP, we ran the check only on results obtained by SIMCO in under 5 minutes: out of the 209 cases, 92 results were verified, while the remaining ones were either timeouts by SIMCO (86) or cases where `autcross` was unable to complete the check within the memory limit (31). All checks were successful.

In our analysis, we focus solely on the number of states of the complements. As SIMCO is a Python-based prototype, execution time comparisons against highly optimized C++ implementations do not provide a meaningful algorithmic evaluation.

For the benchmark RANDOM, we used a one-minute timeout and a 16 GB memory limit. For the SYNTCOMP benchmark, we increased the timeout to one hour per automaton due to the fact that the input automata are much larger in terms of their state counts or number of acceptance sets. The timeouts and out-of-memory errors are presented in Table 1. The failure

³<https://github.com/SYNTCOMP/benchmarks/tree/master/tlsf>

⁴<https://github.com/reactive-systems/syfco>

Table 1: The numbers of timeout and out-of-memory (MO) cases on the benchmark sets

	SIMCO	SIMCO-DEGEN	AUTFILT	KOFOLA	SEMINATOR 2
Timeouts in RANDOM	0	41	5	9	54
MOs in RANDOM	0	0	0	0	6
Timeouts in SYNTCOMP	97	109	1	1	41
MOs in SYNTCOMP	6	6	30	67	51

rate was a real issue in the SYNTCOMP benchmark as is evident from the 97 timeouts and the 6 out-of-memory errors out of 209 cases.

Figure 3 compares the sizes of complements produced by SIMCO against those produced by the other considered configurations for individual automata. On the RANDOM benchmark, SIMCO is clearly outperforming the others, often by orders of magnitude. The SYNTCOMP benchmark is not as favorable for SIMCO as the previous one, since only SIMCO-DEGEN produces consistently larger complements than SIMCO. The reason for the improved performance of the other tools may be a different structure of the input automata. Many of the automata in SYNTCOMP are deterministic, very weak, or elevator automata, to which the other tools apply optimized complementation procedures. Still, the performance of SIMCO remains competitive and against each tool there exist automata where SIMCO produces a smaller complement. For example, in subplot (e) there are two cases where the differences are substantial.

8 Conclusion

We have shown a direct approach for complementing TGBAs based on the construction by Allred and Ultes-Nitsche [3]. For a TGBA with n states and k acceptance sets, the approach produces a complement with at most $\mathcal{O}\left(\left(\frac{kn}{2.718}\right)^n\right)$ states. This bound is asymptotically better than the bounds for previous complementations and it approaches the lower bound $\Omega\left(\left(\frac{kn}{e}\right)^n\right)$.

The approach has been implemented in a tool called SIMCO. Experimental results on the RANDOM benchmark show a clear advantage of using the direct technique over the state-of-the-art tools. Despite the high timeout rate due to the Python implementation, the results remain competitive on the SYNTCOMP benchmark.

The next step is to move SIMCO out of the prototype stage with a performant compiled implementation, which would enable fairer evaluation against state-of-the-art tools. Such an implementation would also allow potential integration into the Spot library or KOFOLA’s modular framework. Other research directions are to explore further optimizations (e.g., using a simulation relation on input automata states), or to further extend the notion of greedy runs to handle more complex acceptance conditions.

Disclaimer on the use of AI

Portions of the experimental pipeline code were written or modified with assistance from Claude (Anthropic). The implementation of SIMCO itself is the authors’ own work. Claude was also used interactively for feedback on the wording and presentation of the text, including suggested reformulations of passages, which the authors reviewed and revised. All experimental design, benchmark selection, tool configuration, result interpretation, and paper content are the authors’ own work, and the authors take full responsibility for the contents of this paper.

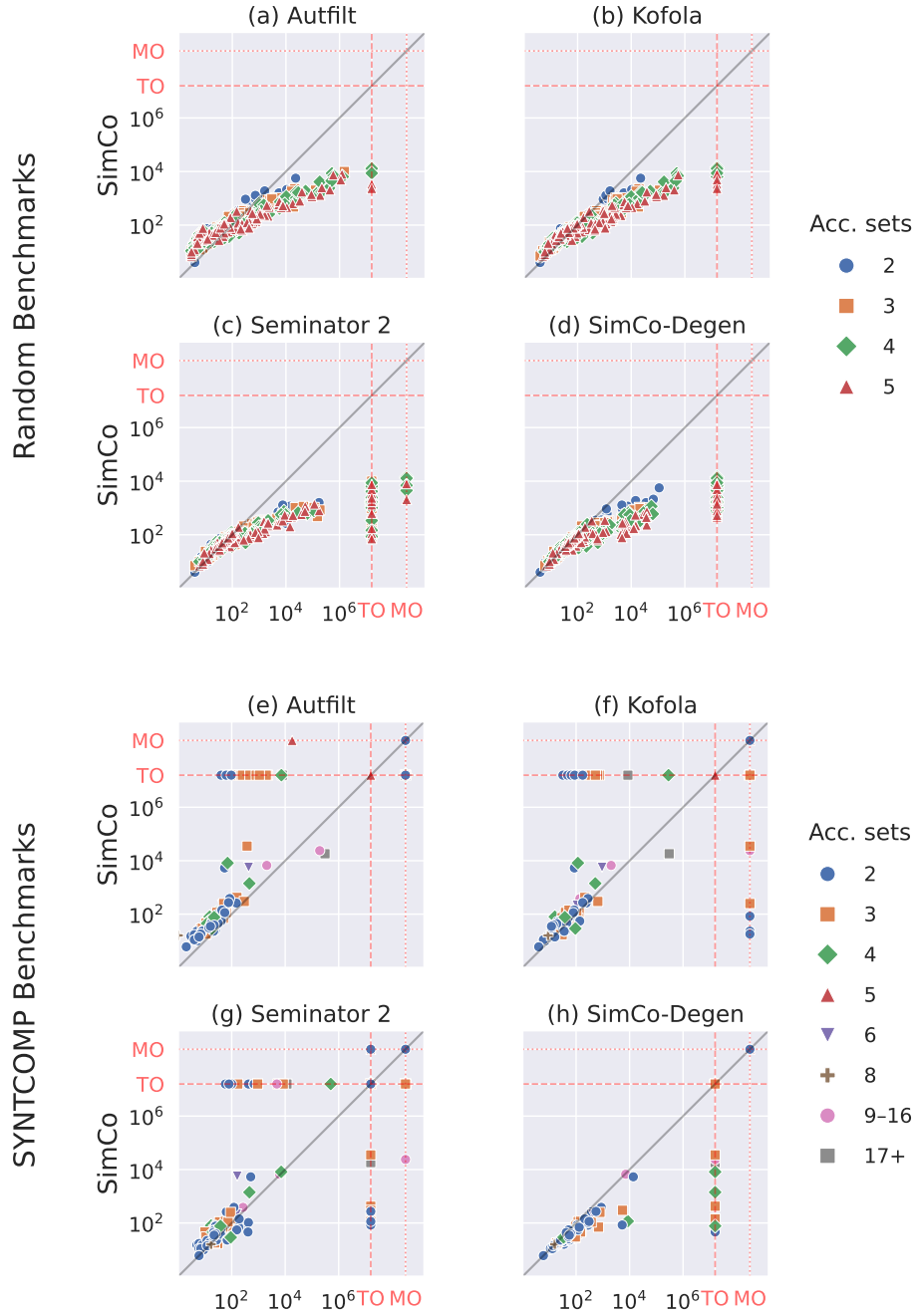


Figure 3: Plots compare the size (i.e., state counts) of complements produced by the considered configurations for individual TGBAs from the sets RANDOM (top) and SYNTCOMP (bottom). The axes are logarithmic. The marks show the number of acceptance sets in the input automata. The labels TO and MO correspond to timeouts and out-of-memory errors, respectively.

References

- [1] O. Alexaj, V. Havlena, O. Lengál, and B. Šmahlíková. Kofola. <https://github.com/VeriFIT/kofola>. Commit: 385841890f826e2b83a48058f89a56d6b087b45a.
- [2] O. Alexaj, V. Havlena, O. Lengál, Y. Li, and N. Mazzocchi. Complementing Emerson-Lei elevator automata. In A. Sokolova and P. Totzke, editors, *37th International Conference on Concurrency Theory, CONCUR 2026, Liverpool, UK, September 1-4, 2026*, volume 391 of *LIPIcs*, pages 8:1–8:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. doi:10.4230/lipics.concur.2026.8.
- [3] J. D. Allred and U. Ultes-Nitsche. A simple and optimal complementation algorithm for Büchi automata. In A. Dawar and E. Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'18)*, pages 46–55. ACM, 2018. doi:10.1145/3209108.3209138.
- [4] T. Babiak, T. Badie, A. Duret-Lutz, M. Křetínský, and J. Strejček. Compositional approach to suspension and other improvements to LTL translation. In *Proceedings of the 20th International SPIN Symposium on Model Checking of Software (SPIN'13)*, volume 7976 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2013. doi:10.1007/978-3-642-39176-7_6.
- [5] T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, D. Müller, D. Parker, and J. Strejček. The Hanoi omega-automata format. In D. Kroening and C. 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.
- [6] T. Babiak, M. Křetínský, V. Řehák, and J. Strejček. LTL to Büchi automata translation: Fast and more deterministic. In C. Flanagan and B. König, editors, *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2012. doi:10.1007/978-3-642-28756-5_8.
- [7] F. Blahoudek, A. Duret-Lutz, and J. Strejček. Seminorator 2. <https://github.com/adl/seminorator>. Commit: dbb71b3740e5e5716c22ad8edb2a0b7620bb2219.
- [8] F. Blahoudek, A. Duret-Lutz, and J. Strejček. Seminorator 2 can complement generalized Büchi automata via improved semi-determinization. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2020. doi:10.1007/978-3-030-53291-8_2.
- [9] F. Blahoudek, M. Heizmann, S. Schewe, J. Strejček, and M. Tsai. Complementing semi-deterministic Büchi automata. In M. Chechik and J. 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.
- [10] S. Breuers, C. Löding, and J. Olschewski. Improved Ramsey-based Büchi complementation. In L. Birkedal, editor, *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'12)*, volume 7213 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2012. doi:10.1007/978-3-642-28729-9_10.
- [11] J. R. Büchi. On a decision method in restricted second order arithmetic. In *1960 International Congress for Logic, Methodology and Philosophy of Science*, 1962.
- [12] Y. Cai and T. Zhang. Tight upper bounds for Streett and Parity complementation. In M. Bezem, editor, *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, volume 12 of *LIPIcs*, pages 112–128. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/lipics.csl.2011.112.

- [13] Y.-F. Chen, M. Heizmann, O. Lengál, Y. Li, M.-H. Tsai, A. Turrini, and L. Zhang. Advanced automata-based algorithms for program termination checking. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'18)*, pages 135–150, 2018. doi:[10.1145/3296979.3192405](https://doi.org/10.1145/3296979.3192405).
- [14] D. Dokoupil. Efficient complementation of generalized Büchi automata. Master's thesis, Masaryk University, 2025. URL: <https://is.muni.cz/th/xu3iz/?lang=en>.
- [15] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, and H. Lauko. spot-lib. <https://spot.lre.epita.fr/>. Version: 2.14.5.
- [16] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, and H. Lauko. From Spot 2.0 to Spot 2.10: What's new? In S. Shoham and Y. Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 2022. doi:[10.1007/978-3-031-13188-2_9](https://doi.org/10.1007/978-3-031-13188-2_9).
- [17] E. A. Emerson and C. Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, 1987. doi:[10.1016/0167-6423\(87\)90036-0](https://doi.org/10.1016/0167-6423(87)90036-0).
- [18] W. Feng, Y. Li, A. Turrini, M. Y. Vardi, and L. Zhang. On the power of finite ambiguity in Büchi complementation. *Inf. Comput.*, 292:105032, 2023. doi:[10.1016/j.ic.2023.105032](https://doi.org/10.1016/j.ic.2023.105032).
- [19] V. Havlena and O. Lengál. Reducing (to) the ranks: Efficient rank-based Büchi automata complementation. In S. Haddad and D. Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 2:1–2:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:[10.4230/lipics.concur.2021.2](https://doi.org/10.4230/lipics.concur.2021.2).
- [20] V. Havlena, O. Lengál, Y. Li, B. Šmahlíková, and A. Turrini. Modular mix-and-match complementation of Büchi automata. In S. Sankaranarayanan and N. Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings, Part I*, volume 13993 of *Lecture Notes in Computer Science*, pages 249–270. Springer, 2023. doi:[10.1007/978-3-031-30823-9_13](https://doi.org/10.1007/978-3-031-30823-9_13).
- [21] V. Havlena, O. Lengál, and B. Šmahlíková. Complementing Büchi automata with Ranker. In S. Shoham and Y. Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 188–201. Springer, 2022. doi:[10.1007/978-3-031-13188-2_10](https://doi.org/10.1007/978-3-031-13188-2_10).
- [22] V. Havlena, O. Lengál, and B. Šmahlíková. Sky is not the limit - tighter rank bounds for elevator automata in Büchi automata complementation. In D. Fisman and G. Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part II*, volume 13244 of *Lecture Notes in Computer Science*, pages 118–136. Springer, 2022. doi:[10.1007/978-3-030-99527-0_7](https://doi.org/10.1007/978-3-030-99527-0_7).
- [23] V. Havlena, O. Lengál, and B. Šmahlíková. Complementation of Emerson-Lei automata. In P. A. Abdulla and D. Kesner, editors, *Foundations of Software Science and Computation Structures - 28th International Conference, FoSSaCS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings*, volume 15691 of *Lecture Notes in Computer Science*, pages 88–110. Springer, 2025. doi:[10.1007/978-3-031-90897-2_5](https://doi.org/10.1007/978-3-031-90897-2_5).
- [24] P. Hieronymi, D. Ma, R. Oei, L. Schaeffer, C. Schulz, and J. O. Shallit. Decidability for Sturmian words. *Log. Methods Comput. Sci.*, 20(3), 2024. doi:[10.46298/lmcs-20\(3:12\)2024](https://doi.org/10.46298/lmcs-20(3:12)2024).
- [25] S. Jacobs, G. A. Pérez, R. Abraham, V. Bruyère, M. Cadilhac, M. Colange, C. Delfosse, T. van Dijk, A. Duret-Lutz, P. Faymonville, B. Finkbeiner, A. Khalimov, F. Klein, M. Luttenberger, K. J. Meyer, T. Michaud, A. Pommellet, F. Renkin, P. Schlehuber-Caissier, M. Sakr,

- S. Sickert, G. Staquet, C. Tamines, L. Tentrup, and A. Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021. *Int. J. Softw. Tools Technol. Transf.*, 26(5):551–567, 2024. doi:[10.1007/s10009-024-00754-1](https://doi.org/10.1007/s10009-024-00754-1).
- [26] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001. doi:[10.1145/377978.377993](https://doi.org/10.1145/377978.377993).
- [27] O. Kupferman and M. Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In N. Halbwachs and L. D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 206–221. Springer, 2005. doi:[10.1007/978-3-540-31980-1_14](https://doi.org/10.1007/978-3-540-31980-1_14).
- [28] O. Kupferman and M. Y. Vardi. From complementation to certification. *Theor. Comput. Sci.*, 345(1):83–100, 2005. doi:[10.1016/j.tcs.2005.07.021](https://doi.org/10.1016/j.tcs.2005.07.021).
- [29] R. 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](https://doi.org/10.1016/0022-0000(87)90036-5).
- [30] D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfssdóttir, and I. Walukiewicz, editors, *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5125 of *Lecture Notes in Computer Science*, pages 724–735. Springer, 2008. doi:[10.1007/978-3-540-70575-8_59](https://doi.org/10.1007/978-3-540-70575-8_59).
- [31] Y. Li, A. Turrini, W. Feng, M. Y. Vardi, and L. Zhang. Divide-and-conquer determinization of Büchi automata based on SCC decomposition. In S. Shoham and Y. Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 152–173. Springer, 2022. doi:[10.1007/978-3-031-13188-2_8](https://doi.org/10.1007/978-3-031-13188-2_8).
- [32] Y. Li, A. Turrini, L. Zhang, and S. Schewe. Learning to complement Büchi automata. In I. Dillig and J. Palsberg, editors, *Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'18)*, volume 10747 of *Lecture Notes in Computer Science*, pages 313–335. Springer, 2018. doi:[10.1007/978-3-319-73721-8_15](https://doi.org/10.1007/978-3-319-73721-8_15).
- [33] Y. Li, M. Y. Vardi, and L. Zhang. On the power of unambiguity in Büchi complementation. In J. Raskin and D. Bresolin, editors, *Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2020, Brussels, Belgium, September 21-22, 2020*, volume 326 of *EPTCS*, pages 182–198, 2020. doi:[10.4204/EPTCS.326.12](https://doi.org/10.4204/EPTCS.326.12).
- [34] S. Miyano and T. Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32:321–330, 1984. doi:[10.1016/0304-3975\(84\)90049-5](https://doi.org/10.1016/0304-3975(84)90049-5).
- [35] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 255–264. IEEE Computer Society, 2006. doi:[10.1109/LICS.2006.28](https://doi.org/10.1109/LICS.2006.28).
- [36] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89)*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989. doi:[10.1007/BFb0035790](https://doi.org/10.1007/BFb0035790).
- [37] S. Safra. On the complexity of omega-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 319–327. IEEE Computer Society, 1988. doi:[10.1109/SFCS.1988.21948](https://doi.org/10.1109/SFCS.1988.21948).
- [38] S. Schewe. Büchi complementation made tight. In S. Albers and J. Marion, editors, *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS'09)*, volume 3 of *LIPICs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009. doi:[10.4230/lipics.stacs.2009.1854](https://doi.org/10.4230/lipics.stacs.2009.1854).
- [39] S. Schewe and T. Varghese. Tight bounds for the determinisation and complementation of gen-

- eralised Büchi automata. In S. Chakraborty and M. Mukund, editors, *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, volume 7561 of *Lecture Notes in Computer Science*, pages 42–56. Springer, 2012. doi:[10.1007/978-3-642-33386-6_5](https://doi.org/10.1007/978-3-642-33386-6_5).
- [40] S. Schewe and T. Varghese. Tight bounds for complementing parity automata. In E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 499–510. Springer, 2014. doi:[10.1007/978-3-662-44522-8_42](https://doi.org/10.1007/978-3-662-44522-8_42).
- [41] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987. doi:[10.1016/0304-3975\(87\)90008-9](https://doi.org/10.1016/0304-3975(87)90008-9).
- [42] M. Y. Vardi. The Büchi complementation saga. In W. Thomas and P. Weil, editors, *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *Lecture Notes in Computer Science*, pages 12–22. Springer, 2007. doi:[10.1007/978-3-540-70918-3_2](https://doi.org/10.1007/978-3-540-70918-3_2).
- [43] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society, 1986. URL: https://www.researchgate.net/publication/221442685_An_Automata-Theoretic_Approach_to_Automatic_Program_Verification_Preliminary_Report.