Modular Mix-and-Match Complementation of Büchi Automata

Vojtěch Havlena[1(✉)] , Ondřej Lengál[1(✉)] , Yong Li[2,3(✉)] ,
Barbora Šmahlíková[1(✉)] , and Andrea Turrini[3,4(✉)]

[1] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
ihavlena@fit.vut.cz, lengal@vut.cz, xsmahl00@vut.cz
[2] Department of Computer Science, University of Liverpool, Liverpool, UK
liyong@liverpool.ac.uk
[3] State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing, People's Republic of China
turrini@ios.ac.cn
[4] Institute of Intelligent Software, Guangzhou, Guangzhou, People's Republic of China

Abstract. Complementation of nondeterministic Büchi automata (BAs) is an important problem in automata theory with numerous applications in formal verification, such as termination analysis of programs, model checking, or in decision procedures of some logics. We build on ideas from a recent work on BA determinization by Li *et al.* and propose a new modular algorithm for BA complementation. Our algorithm allows to combine several BA complementation procedures together, with one procedure for a subset of the BA's strongly connected components (SCCs). In this way, one can exploit the structure of particular SCCs (such as when they are inherently weak or deterministic) and use more efficient specialized algorithms, regardless of the structure of the whole BA. We give a general framework into which partial complementation procedures can be plugged in, and its instantiation with several algorithms. The framework can, in general, produce a complement with an Emerson-Lei acceptance condition, which can often be more compact. Using the algorithm, we were able to establish an exponentially better new upper bound of $O(4^n)$ for complementation of the recently introduced class of elevator automata. We implemented the algorithm in a prototype and performed a comprehensive set of experiments on a large set of benchmarks, showing that our framework complements well the state of the art and that it can serve as a basis for future efficient BA complementation and inclusion checking algorithms.

1 Introduction

Nondeterministic Büchi automata (BAs) [8] are an elegant and conceptually simple framework to model infinite behaviors of systems and the properties they are expected to satisfy. BAs are widely used in many important verification tasks, such as termination analysis of programs [30], model checking [54], or as the underlying formal model of decision procedures for some logics (such as S1S [8] or a fragment of the first-order logic over Sturmian words [31]). Many of these applications require to perform *complementation* of BAs: For instance, in termination analysis of programs within ULTIMATE AUTOMIZER [30], complementation is used to keep track of the set of paths whose termination still needs to be proved. On the other hand, in model checking[5] and decision

---

[5] Here, we consider model checking w.r.t. a specification given in some more expressive logic, such as S1S [8], QPTL [50], or HyperLTL [12], rather than LTL [44], where negation is simple.

procedures of logics, complement is usually used to implement negation and quantifier alternation. Complementation is often the most difficult automata operation performed here; its worst-case state complexity is $O((0.76n)^n)$ [48,2] (which is tight [55]).

In these applications, efficiency of the complementation often determines the overall efficiency (or even feasibility) of the top-level application. For instance, the success of ULTIMATE AUTOMIZER in the Termination category of the International Competition on Software Verification (SV-COMP) [51] is to a large degree due to an efficient BA complementation algorithm [6,11] tailored for BAs with a special structure that it often encounters (as of the time of writing, it has won 6 gold medals in the years 2017–2022 and two silver medals in 2015 and 2016). The special structure in this case are the so-called *semi-deterministic BAs* (SDBAs), BAs consisting of two parts: (i) an initial part without accepting states/transitions and (ii) a deterministic part containing accepting states/transitions that cannot transition into the first part.

Complementation of SDBAs using one from the family of the so-called NCSB algorithms [6,5,11,28] has the worst-case complexity $O(4^n)$ (and usually also works much better in practice than general BA complementation procedures). Similarly, there are efficient complementation procedures for other subclasses of BAs, e.g., (i) *deterministic BAs* (DBAs) can be complemented into BAs with $2n$ states [35] (or into co-Büchi automata with $n + 1$ states) or (ii) *inherently weak BAs* (BAs where in each *strongly connected component* (SCC), either all cycles are accepting or all cycles are rejecting) can be complemented into DBAs with $O(3^n)$ states using the Miyano-Hayashi algorithm [42].

For a long time, there has been no efficient algorithm for complementation of BAs that are highly structured but do not fall into one of the categories above, e.g., BAs containing inherently weak, deterministic, and some nondeterministic SCCs. For such BAs, one needed to use a general complementation algorithm with the $O((0.76n)^n)$ (or worse) complexity. To the best of our knowledge, only recently has there appeared works that exploit the structure of BAs to obtain a more efficient complementation algorithm: (i) The work of Havlena *et al.* [29], who introduce the class of *elevator automata* (BAs with an arbitrary mixture of inherently weak and deterministic SCCs) and give a $O(16^n)$ algorithm for them. (ii) The work of Li *et al.* [37], who propose a BA determinization procedure (into a deterministic Emerson-Lei automaton) that is based on decomposing the input BA into SCCs and using a different determinization procedure for different types of SCCs (inherently weak, deterministic, general) in a synchronous construction.

In this paper, we propose a new BA complementation algorithm inspired by [37], where we exploit the fact that complementation is, in a sense, more relaxed than determinization. In particular, we present a *framework* where one can plug-in different partial complementation procedures fine-tuned for SCCs with a specific structure. The procedures work only with the given SCCs, to some degree *independently* (thus reducing the potential state space explosion) from the rest of the BA. Our top-level algorithm then orchestrates runs of the different procedures in a *synchronous* manner (or completely independently in the so-called *postponed* strategy), obtaining a resulting automaton with potentially a more general acceptance condition (in general an Emerson-Lei condition), which can help keeping the result small. If the procedures satisfy given correctness requirements, our framework guarantees that its instantiation will also be correct. We also propose its optimizations by, e.g., using round-robin to decrease the amount of nondeterminism, using a shared breakpoint to reduce the size and the number of colours for certain class of partial algorithms, and generalize simulation-based pruning of macrostates.

We provide a detailed description of partial complementation procedures for inherently weak, deterministic, and initial deterministic SCCs, which we use to obtain a *new* exponentially better upper bound of $O(4^n)$ for the class of elevator automata (i.e., the same upper bound as for its strict subclass of SDBAs). Furthermore, we also provide two partial procedures for general SCCs based on determinization (from [37]) and the rank-based construction. Using a prototype implementation, we then show our algorithm complements well existing approaches and significantly improves the state of the art.

2   Preliminaries

We fix a finite non-empty alphabet Σ and the first infinite ordinal ω. An (infinite) word w is a function $w\colon \omega \to \Sigma$ where the i-th symbol is denoted as w_i. Sometimes, we represent w as an infinite sequence $w = w_0 w_1 \ldots$ We denote the set of all infinite words over Σ as Σ^ω; an *ω-language* is a subset of Σ^ω.

Emerson-Lei Acceptance Conditions. Given a set $\Gamma = \{0, \ldots, k-1\}$ of k *colours* (often depicted as **0**, **1**, etc.), we define the set of *Emerson-Lei acceptance conditions* $\mathbb{EL}(\Gamma)$ as the set of formulae constructed according to the following grammar:

$$\alpha ::= \mathsf{Inf}(c) \mid \mathsf{Fin}(c) \mid (\alpha \wedge \alpha) \mid (\alpha \vee \alpha) \tag{1}$$

for $c \in \Gamma$. The *satisfaction* relation $\models$ for a set of colours $M \subseteq \Gamma$ and condition α is defined inductively as follows (for $c \in \Gamma$):

$$M \models \mathsf{Fin}(c) \text{ iff } c \notin M, \qquad M \models \alpha_1 \vee \alpha_2 \text{ iff } M \models \alpha_1 \text{ or } M \models \alpha_2,$$
$$M \models \mathsf{Inf}(c) \text{ iff } c \in M, \qquad M \models \alpha_1 \wedge \alpha_2 \text{ iff } M \models \alpha_1 \text{ and } M \models \alpha_2.$$

Emerson-Lei Automata. A (nondeterministic transition-based[6]) *Emerson-Lei automaton* (TELA) over Σ is a tuple $\mathcal{A} = (Q, \delta, I, \Gamma, \mathsf{p}, \mathsf{Acc})$, where Q is a finite set of *states*, $\delta \subseteq Q \times \Sigma \times Q$ is a set of *transitions*[7], $I \subseteq Q$ is the set of *initial* states, Γ is the set of *colours*, $\mathsf{p}\colon \delta \to 2^\Gamma$ is a *colouring function* of transitions, and $\mathsf{Acc} \in \mathbb{EL}(\Gamma)$. We use $p \xrightarrow{a} q$ to denote that $(p, a, q) \in \delta$ and sometimes also treat δ as a function $\delta\colon Q \times \Sigma \to 2^Q$. Moreover, we extend δ to sets of states $P \subseteq Q$ as $\delta(P, a) = \bigcup_{p \in P} \delta(p, a)$. We use $\mathcal{A}[q]$ for $q \in Q$ to denote the automaton $\mathcal{A}[q] = (Q, \delta, \{q\}, \Gamma, \mathsf{p}, \mathsf{Acc})$, i.e., the TELA obtained from $\mathcal{A}$ by setting q as the only initial state. $\mathcal{A}$ is called *deterministic* if $|I| \leq 1$ and $|\delta(q, a)| \leq 1$ for each $q \in Q$ and $a \in \Sigma$. If $\Gamma = \{0\}$ and $\mathsf{Acc} = \mathsf{Inf}(0)$, we call $\mathcal{A}$ a *Büchi automaton* (BA) and denote it as $\mathcal{A} = (Q, \delta, I, F)$ where F is the set of all transitions coloured by 0, i.e., $F = \mathsf{p}^{-1}(\{0\})$. For a BA, we use $\delta_F(p, a) = \{q \in \delta(p, a) \mid \mathsf{p}(p \xrightarrow{a} q) = \{0\}\}$ (and extend the notation to sets of states as for δ). A BA $\mathcal{A} = (Q, \delta, I, F)$ is called *semi-deterministic* (SDBA) if for every accepting transition $(p \xrightarrow{a} q) \in F$, the reachable part of $\mathcal{A}[q]$ is deterministic.

A *run* of $\mathcal{A}$ from $q \in Q$ on an input word w is an infinite sequence $\rho\colon \omega \to Q$ that starts in q and respects δ, i.e., $\rho_0 = q$ and $\forall i \geq 0\colon \rho_i \xrightarrow{w_i} \rho_{i+1} \in \delta$. Let $\inf_\delta(\rho) \subseteq \delta$ denote the set of transitions occurring in ρ infinitely often and $\inf_\Gamma(\rho) = \bigcup\{\mathsf{p}(x) \mid x \in$

---

[6] We only consider transition-based acceptance in order to avoid cluttering the paper by always dealing with accepting states *and* accepting transitions. Extending our approach to state/transition-based (or just state-based) automata is straightforward.

[7] Note that some authors use a more general definition of TELAs with $\delta \subseteq Q \times \Sigma \times 2^\Gamma \times Q$; we only use them as the output of our algorithm, where the simpler definition suffices.

$\inf_\delta(\rho)\}$ be the set of infinitely often occurring colours. A run ρ is *accepting* in $\mathcal{A}$ iff $\inf_\Gamma(\rho) \models \mathsf{Acc}$ and the *language* of $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is defined as the set of words $w \in \Sigma^\omega$ for which there exists an accepting run in $\mathcal{A}$ starting with some state in I.

Consider a BA $\mathcal{A} = (Q, \delta, I, F)$. For a set of states $S \subseteq Q$ we use $\mathcal{A}_S$ to denote the copy of $\mathcal{A}$ where accepting transitions only occur between states from S, i.e., the BA $\mathcal{A}_S = (Q, \delta, I, F \cap \delta|_S)$ where $\delta|_S = \{p \xrightarrow{a} q \in \delta \mid p, q \in S\}$. We say that a non-empty set of states $C \subseteq Q$ is a *strongly connected component* (SCC) if every pair of states of C can reach each other and C is a maximal such set. An SCC of $\mathcal{A}$ is *trivial* if it consists of a single state that does not contain a self-loop and *non-trivial* otherwise. An SCC C is *accepting* if it contains at least one accepting transition and *inherently weak* iff either (i) every cycle in C contains a transition from F or (ii) no cycle in C contains any transitions from F. An SCC C is *deterministic* iff the BA $(C, \delta|_C, \{q\}, \emptyset)$ for any $q \in C$ is deterministic. We denote inherently weak components as IWCs, accepting deterministic components that are not inherently weak as DACs (deterministic accepting), and the remaining accepting components as NACs (nondeterministic accepting). A BA $\mathcal{A}$ is called an *elevator automaton* if it contains no NAC.

We assume that $\mathcal{A}$ contains no accepting transition outside its SCCs (no run can cycle over such transitions). We use δ_{SCC} to denote the restriction of δ to transitions that do not leave their SCCs, formally, $\delta_{\mathrm{SCC}} = \{p \xrightarrow{a} q \in \delta \mid p \text{ and } q \text{ are in the same SCC}\}$. A *partition block* $P \subseteq Q$ of $\mathcal{A}$ is a nonempty union of its accepting SCCs, and a *partitioning* of $\mathcal{A}$ is a sequence $P_1, \ldots, P_n$ of pairwise disjoint partition blocks of $\mathcal{A}$ that contains all accepting SCCs of $\mathcal{A}$. Given a P_i, let $\mathcal{A}_{P_i}$ be the BA obtained from $\mathcal{A}$ by removing colours from transitions outside P_i. The following fact serves as the basis of our decomposition-based complementation procedure.

Fact 1. $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_{P_1}) \cup \ldots \cup \mathcal{L}(\mathcal{A}_{P_n})$

The complement (automaton) of a BA $\mathcal{A}$ is a TELA that accepts the complement language $\Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$ of $\mathcal{L}(\mathcal{A})$. In the paper, we call a state and a run of a complement automaton a *macrostate* and a *macrorun*, respectively.

3   A Modular Complementation Algorithm

In a nutshell, the main idea of our BA complementation algorithm is to first decompose a BA $\mathcal{A}$ into several partition blocks according to their properties, and then perform complementation for each of the partition blocks (potentially using a different algorithm) independently, using either a *synchronous* construction, synchronizing the complementation algorithms for all partition blocks in each step, or a *postponed* construction, which complements the partition blocks independently and combines the partial results using automata product construction. The decomposition of $\mathcal{A}$ into partition blocks can either be trivial—i.e., with one block for each accepting SCC—, or more elaborate, e.g., a partitioning where one partition block contains all accepting IWCs, another contains all DACs, and each NAC is given its own partition block. In this way, one can avoid running a general complementation algorithm for unrestricted BAs with the state complexity upper bound $O((0.76n)^n)$ and, instead, apply the most suitable complementation procedure for each of the partition blocks. This comes with three main advantages:

1. The complementation algorithm for each partition block can be selected differently in order to exploit the properties of the block. For instance, for partition blocks

with IWCs, one can use complementation based on the breakpoint (the so-called Miyano-Hayashi) construction [42] with $O(3^n)$ macrostates (cf. Sec. 4.1), while for partition blocks with only DACs, one can use an algorithm with the state complexity $O(4^n)$ based on an adaptation of the NCSB construction [6,5,11,28] for SDBAs (cf. Sec. 4.2). For NACs, one can choose between, e.g., rank- [34,21,48,10,24,29] or determinization-based [46,43,45] algorithms, depending on the properties of the NACs (cf. Sec. 6).

2. The different complementation algorithms can focus only on the respective blocks and do not need to consider other parts of the BA. This is advantageous, e.g., for rank-based algorithms, which can use this restriction to obtain tighter bounds on the considered ranks (even tighter than using the refinement in [29]).

3. The obtained automaton can be more compact due to the use of a more general acceptance condition than Büchi [47]—in general, it can be a conjunction of any $\mathbb{EL}$ conditions (one condition for each partition block), depending on the output of the complementation procedures; this can allow a more compact encoding of the produced automaton allowed by using a mixture of conditions. E.g., a deterministic BA can be complemented with constant extra generated states when using a co-Büchi condition rather than a linear number of generated states for a Büchi condition (see Sec. 5.1).

Those partial complementation algorithms then need to be orchestrated by a top-level algorithm to produce the complement of $\mathcal{A}$.

One might regard our algorithm as an optimization of an approach that would for each partition block P obtain a BA $\mathcal{A}_P$, complement $\mathcal{A}_P$ using the selected algorithm, and perform the intersection of all obtained $\mathcal{A}_P$'s (which would, however, not be able to get the upper bound for elevator automata that we give in Sec. 4.3). Indeed, we also implemented the mentioned procedure (called the *postponed* approach, described in Sec. 5.2) and compared it to our main procedure (called the *synchronous* approach).

3.1 Basic Synchronous Algorithm

In this section, we describe the basic *synchronous* top-level algorithm. Then, in Sec. 4, we provide its instantiation for elevator automata and give a new upper bound for their complementation; in Sec. 5, we discuss several optimizations of the algorithm; and in Sec. 6, we give a generalization for unrestricted BAs. Let us fix a BA $\mathcal{A} = (Q, \delta, I, F)$ and, w.l.o.g., assume that $\mathcal{A}$ is *complete*, i.e., $|I| > 0$ and all states $q \in Q$ have an outgoing transition over all symbols $a \in \Sigma$.

The synchronous algorithm works with partial complementation algorithms for BA's partition blocks. Each such algorithm Alg is provided with a structural condition $\varphi_{\texttt{Alg}}$ characterizing partition blocks it can complement. For a BA $\mathcal{B}$, we use the notation $\mathcal{B} \models \varphi$ to denote that $\mathcal{B}$ satisfies the condition φ. We say that Alg is a *partial complementation algorithm for a partition block P* if $\mathcal{A}_P \models \varphi_{\texttt{Alg}}$. We distinguish between Alg, a general algorithm able to complement a partition block of a given type, and $\texttt{Alg}_P$, its instantiation for the partition block P. Each instance $\texttt{Alg}_P$ is required to provide the following:

- $\texttt{T}^{\texttt{Alg}_P}$ — the type of the macrostates produced by the algorithm;
- $\texttt{Colours}^{\texttt{Alg}_P} = \{0, \ldots, k^{\texttt{Alg}_P} - 1\}$ — the set of used colours;
- $\texttt{Init}^{\texttt{Alg}_P} \in 2^{\texttt{T}^{\texttt{Alg}_P}}$ — the set of initial macrostates;
- $\texttt{Succ}^{\texttt{Alg}_P} : (2^Q \times \texttt{T}^{\texttt{Alg}_P} \times \Sigma) \to 2^{\texttt{T}^{\texttt{Alg}_P} \times \texttt{Colours}^{\texttt{Alg}_P}}$ — a function returning the successors of a macrostate such that $\texttt{Succ}^{\texttt{Alg}_P}(H, M, a) = \{(M_1, \alpha_1), \ldots, (M_k, \alpha_k)\}$, where H is the set of all states of $\mathcal{A}$ reached over the same word, M is the $\texttt{Alg}_P$'s

macrostate for the given partition block, a is the input symbol, and each (M_i, α_i) is a pair (*macrostate*, *set of colours*) such that M_i is a successor of M over a w.r.t. H and α_i is a set of colours on the edge from M to M_i (H helps to keep track of *new* runs coming into the partition block); and
- $\mathtt{Acc}^{\mathtt{Alg}_P} \in \mathbb{BL}(\mathtt{Colours}^{\mathtt{Alg}_P})$ — the acceptance condition.

Let $P_1, \ldots, P_n$ be a partitioning of $\mathcal{A}$ (w.l.o.g., we assume that $n > 0$), and $\mathtt{Alg}^1, \ldots, \mathtt{Alg}^n$ be a sequence of algorithms such that $\mathtt{Alg}^i$ is a partial complementation algorithm for P_i. Furthermore, let us define the following auxiliary *renumbering* function λ as $\lambda(c, j) = c + \sum_{i=1}^{j-1} |\mathtt{Colours}^{\mathtt{Alg}_{P_i}^i}|$, which is used to make the colours and acceptance conditions from the partial complementation algorithms disjoint. We also lift λ to sets of colours in the natural way, and also to $\mathbb{BL}$ conditions such that $\lambda(\varphi, j)$ has the same structure as φ but each atom $\mathsf{Inf}(c)$ is substituted with the atom $\mathsf{Inf}(\lambda(c, j))$ (and likewise for Fin atoms). The synchronous complementation algorithm then produces the TELA $\textsc{ModCompl}(\mathtt{Alg}_{P_1}^1, \ldots, \mathtt{Alg}_{P_n}^n, \mathcal{A}) = (Q^C, \delta^C, I^C, \Gamma^C, \mathsf{p}^C, \mathsf{Acc}^C)$ with components defined as follows (we use $[S_i]_{i=1}^n$ to abbreviate $S_1 \times \cdots \times S_n$):

- $Q^C = 2^Q \times [\mathtt{T}^{\mathtt{Alg}_{P_i}^i}]_{i=1}^n$,
- $\Gamma^C = \{0, \ldots, \lambda(k^{\mathtt{Alg}_{P_n}^n} - 1, n)\}$,
- $I^C = \{I\} \times [\mathtt{Init}^{\mathtt{Alg}_{P_i}^i}]_{i=1}^n$,
- $\mathsf{Acc}^C = \bigwedge_{i=1}^n \lambda(\mathsf{Acc}^{\mathtt{Alg}_{P_i}^i}, i)$,[8] and
- δ^C and p^C are defined such that if

$$((M_1', \alpha_1), \ldots, (M_n', \alpha_n)) \in [\mathtt{Succ}^{\mathtt{Alg}_{P_i}^i}(H, M_i, a)]_{i=1}^n,$$

then δ^C contains the transition $t \colon (H, M_1, \ldots, M_n) \xrightarrow{a} (\delta(H, a), M_1', \ldots, M_n')$, coloured by $\mathsf{p}^C(t) = \bigcup\{\lambda(\alpha_i, i) \mid 1 \leq i \leq n\}$, and δ^C is the smallest such a set.

In order for ModCompl to be correct, the partial complementation algorithms need to satisfy certain properties, which we discuss below.

For a structural condition φ and a BA $\mathcal{B} = (Q, \delta, I, F)$, we define $\mathcal{B} \models_P \varphi$ iff $\mathcal{B} \models \varphi$, P is a partition block of $\mathcal{B}$, and $\mathcal{B}$ contains no accepting transitions outside P. We can now provide the correctness condition on $\mathtt{Alg}$.

Definition 1. *We say that $\mathtt{Alg}$ is* correct *if for each BA $\mathcal{B}$ and partition block P such that $\mathcal{B} \models_P \varphi_{\mathtt{Alg}}$ it holds that $\mathcal{L}(\textsc{ModCompl}(\mathtt{Alg}_P, \mathcal{B})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$.*

The correctness of the synchronous algorithm (provided that each partial complementation algorithm is correct) is then established by Theorem 1.

Theorem 1. *Let $\mathcal{A}$ be a BA, $P_1, \ldots, P_n$ be a partitioning of $\mathcal{A}$, and $\mathtt{Alg}^1, \ldots, \mathtt{Alg}^n$ be a sequence of partial complementation algorithms such that $\mathtt{Alg}^i$ is correct for P_i. Then, we have $\mathcal{L}(\textsc{ModCompl}(\mathtt{Alg}_{P_1}^1, \ldots, \mathtt{Alg}_{P_n}^n, \mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.*

4   Modular Complementation of Elevator Automata

In this section, we first give partial algorithms to complement partition blocks with only accepting IWCs (Sec. 4.1) and partition blocks with only DACs (Sec. 4.2). Then, in Sec. 4.3, we show that using our algorithm, the upper bound on the size of the complement of elevator BAs is in $O(4^n)$, which is *exponentially better* than the known upper bound $O(16^n)$ established in [29].

---

[8] If we drop the condition that $\mathcal{A}$ is complete, we also need to add an *accepting sink state* (representing the case for $H = \emptyset$) with self-loops over all symbols marked by a new colour ⓢ, and enrich Acc^C with $\ldots \vee \mathsf{Inf}(\text{ⓢ})$.

4.1 Complementation of Inherently Weak Accepting Components

First, we introduce a partial algorithm MH with the condition φ_{MH} specifying that all SCCs in the partition block P are *accepting* IWCs. Let P be a partition block of $\mathcal{A}$ such that $\mathcal{A}_P \models \varphi_{\text{MH}}$. Our proposed approach makes use of the Miyano-Hayashi construction [42]. Since in accepting IWCs, all runs are accepting, the idea of the construction is to accept words such that all runs over the words eventually leave P.

Therefore, we use a pair (C, B) of sets of states as a macrostate for complementing P. Intuitively, we use C to denote the set of all runs of $\mathcal{A}$ that are in P (C for "*check*"). The set $B \subseteq C$ represents the runs being inspected whether they leave P at some point (B for "*breakpoint*"). Initially, we let $C = I \cap P$ and also sample into breakpoint all runs in P, i.e., set $B = C$. Along reading an ω-word w, if all runs that have entered P eventually leave P, i.e., B becomes empty infinitely often, the complement language of P should contain w (when B becomes empty, we sample B with all runs from the current C). We formalize MH_P as a partial procedure in the framework from Sec. 3.1 as follows:

- $\mathtt{T}^{\text{MH}_P} = 2^P \times 2^P$, $\mathtt{Colours}^{\text{MH}_P} = \{\textcolor{magenta}{\mathbf{0}}\}$, $\mathtt{Init}^{\text{MH}_P} = \{(I \cap P, I \cap P)\}$,
- $\mathtt{Acc}^{\text{MH}_P} = \mathsf{Inf}(\textcolor{magenta}{\mathbf{0}})$, and $\mathtt{Succ}^{\text{MH}_P}(H, (C, B), a) = \{((C', B'), \alpha)\}$ where
  - $C' = \delta(H, a) \cap P$,
  - $B' = \begin{cases} C' & \text{if } B^\star = \emptyset \text{ for } B^\star = \delta(B, a) \cap C', \\ B^\star & \text{otherwise, and} \end{cases}$ • $\alpha = \begin{cases} \{\textcolor{magenta}{\mathbf{0}}\} & \text{if } B^\star = \emptyset \text{ and} \\ \emptyset & \text{otherwise.} \end{cases}$

We can see that checking whether w is accepted by the complement of P reduces to check whether B has been cleared infinitely often. Since every time when B becomes empty, we emit the colour $\textcolor{magenta}{\mathbf{0}}$, we have that w is not accepted by $\mathcal{A}$ within P if and only if $\textcolor{magenta}{\mathbf{0}}$ occurs infinitely often. Note that the transition function $\mathtt{Succ}^{\text{MH}_P}$ is deterministic, i.e., there is exactly one successor.

Lemma 1. *The partial algorithm* MH *is correct.*

4.2 Complementation of Deterministic Accepting Components

In this section, we give a partial algorithm CSB with the condition φ_{CSB} specifying that a partition block P consists of *DACs*. Let P be a partition block of $\mathcal{A}$ such that $\mathcal{A}_P \models \varphi_{\text{CSB}}$. Our approach is based on the NCSB family of algorithms [6,11,5,28] for complementing SDBAs, in particular the NCSB-MaxRank construction [28]. The algorithm utilizes the fact that runs in DACs are deterministic, i.e., they do not branch into new runs. Therefore, one can check that a run is non-accepting if there is a time point from which the run does not see accepting transitions any more. We call such a run that does not see accepting transitions any more *safe*. Then, an ω-word w is not accepted in P iff all runs over w in P either (i) leave P or (ii) eventually become safe.

For checking point (i), we can use a similar technique as in algorithm MH, i.e., use a pair (C, B). Moreover, to be able to check point (ii), we also use the set S that contains runs that are supposed to be *safe*, resulting in macrostates of the form (C, S, B)[9]. To make sure that all runs are deterministic, we will use δ_{SCC} instead of δ when computing the successors of S and B since there may be nondeterministic jumps between different DACs in P; we will not miss any run in P since if a run moves between DACs of P, it

---

[9] In contrast to MH, here we use $C \cup S$ rather than C to keep track of all runs in P.
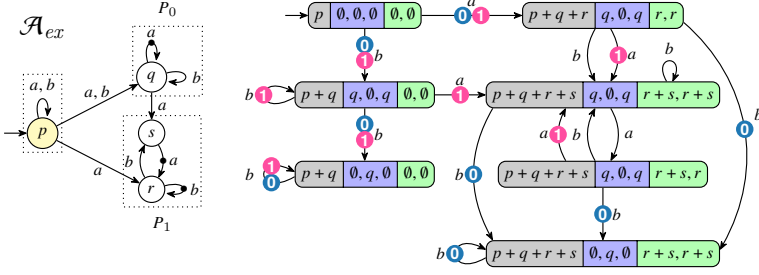
Fig. 1: Left: BA $\mathcal{A}_{ex}$ (dots represent accepting transitions). Right: the outcome of $\text{MODCOMPL}(\text{CSB}_{P_0}, \text{MH}_{P_1}, \mathcal{A}_{ex})$ with $\text{Acc}\colon \text{Inf}(\textcircled{0}) \wedge \text{Inf}(\textcircled{1})$. States are given as $(H, (C_0, S_0, B_0), (C_1, B_1))$; to avoid too many braces, sets are given as sums.

can be seen as the run leaving P and a new run entering P. Since a run eventually stays in one SCC, this guarantees that the run will not be missed.

We formalize CSB_P in the top-level framework as follows:

- $\text{T}^{\text{CSB}_P} = 2^P \times 2^P \times 2^P$, $\text{Init}^{\text{CSB}_P} = \{(I \cap P, \emptyset, I \cap P)\}$,
- $\text{Colours}^{\text{CSB}_P} = \{\textcircled{0}\}$, $\text{Acc}^{\text{CSB}_P} = \text{Inf}(\textcircled{0})$, and
- $\text{Succ}^{\text{CSB}_P}(H, (C, S, B), a) = U$ such that
  - if $\delta_F(S, a) \neq \emptyset$, then $U = \emptyset$ (Runs in S must be *safe*),
  - otherwise U contains $((C', S', B'), c)$ where
    * $S' = \delta_{\text{SCC}}(S, a) \cap P$, $C' = (\delta(H, a) \cap P) \setminus S'$,
    * $B' = \begin{cases} C' & \text{if } B^\star = \emptyset \text{ for } B^\star = \delta_{\text{SCC}}(B, a), \\ B^\star & \text{otherwise, and} \end{cases}$ * $c = \begin{cases} \{\textcircled{0}\} & \text{if } B^\star = \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$

    Moreover, in the case $\delta_F(B, a) = \emptyset$, then U also contains $((C'', S'', C''), \{\textcircled{0}\})$ where $S'' = S' \cup B'$ and $C'' = C' \setminus S''$.

Intuitively, when $\delta_F(B, a) \cap \delta_{\text{SCC}}(B, a) = \emptyset$, we make the following guess: (i) either the runs in B all become safe (we move them to S) or (ii) there might be some unsafe runs (we keep them in B). Since the runs in B are deterministic, the number of tracked runs in B will not increase. Moreover, if all runs in B are eventually safe, we are guaranteed to move all of them to S at the right time point, e.g., the maximal time point where all runs are safe since the number of runs is finite.

As mentioned above, w is not accepted within P iff all runs over w either (i) leave P or (ii) become safe. In the context of the presented algorithm, this corresponds to (i) B becoming empty infinitely often and (ii) $\delta_F(S, a)$ never seeing an accepting transition. Then we only need to check if there exists an infinite sequence of macrostates $\hat{\rho} = (C_0, S_0, B_0) \ldots$ that emits $\textcircled{0}$ infinitely often.

Lemma 2. *The partial algorithm* CSB *is correct.*

It is worth noting that when the given partition block P contains all DACs of $\mathcal{A}$, we can still use the construction above, while the construction in [28] only works on SDBAs.

Example 1. In Fig. 1, we give an example of the run of our algorithm on the BA $\mathcal{A}_{ex}$. The BA contains three SCCs, one of them (the one containing p) non-accepting (therefore,

it does not need to occur in any partition block). The partition block P_0 contains a single DAC, so we can use algorithm CSB, and the partition block P_1 contains a single accepting IWC, so we can use MH. The resulting $\text{MODCOMPL}(\text{CSB}_{P_0}, \text{MH}_{P_1}, \mathcal{A}_{ex})$ uses two colours, **🔵0** from CSB and **🔴1** from MH. The acceptance condition is $\text{Inf}(🔵) \wedge \text{Inf}(🔴)$.   □

4.3 Upper-bound for Elevator Automata Complementation

We now give an upper bound on the size of the complement generated by our algorithm for elevator automata, which significantly improves the best previously known upper bound of $O(16^n)$ [29] to $O(4^n)$, the same as for SDBAs, which are a strict subclass of elevator automata [6] (we note that this upper bound cannot be obtained by a determinization-based algorithm, since determinization of SDBAs is in $\Omega(n!)$ [17,40]).

Theorem 2. *Let $\mathcal{A}$ be an elevator automaton with n states. Then there exists a BA with $O(4^n)$ states accepting the complement of $\mathcal{L}(\mathcal{A})$.*

Proof (Sketch). Let Q_W be all states in accepting IWCs, Q_D be all states in DACs, and Q_N be the remaining states, i.e., $Q = Q_W \uplus Q_D \uplus Q_N$. We make two partition blocks: $P_0 = Q_W$ and $P_1 = Q_D$ and use MH and CSB respectively as the partial algorithms, with macrostates of the form $(H, (C_0, B_0), (C_1, S_1, B_1))$. For each state $q_N \in Q_N$, there are two options: either $q_N \notin H$ or $q_N \in H$. For each state $q_W \in Q_W$, there are three options: (i) $q_W \notin C_0$, (ii) $q_W \in C_0 \setminus B_0$, or (iii) $q_W \in C_0 \cap B_0$. Finally, for each $q_D \in Q_D$, there are four options: (i) $q_D \notin C_1 \cup S_1$, (ii) $q_D \in S_1$, (iii) $q_D \in C_1 \setminus B_1$, or (iv) $q_D \in C_1 \cap B_1$. Therefore, the total number of macrostates is $2 \cdot 2^{|Q_N|} \cdot 3^{|Q_W|} \cdot 4^{|Q_D|} \in O(4^n)$ where the initial factor 2 is due to degeneralization from two to one colour (the two colours can actually be avoided by using our shared breakpoint optimization from Sec. 5.4).   □

5 Optimizations of the Modular Construction

In this section, we propose optimizations of the basic modular algorithm. In Sec. 5.1, we give a partial algorithm to complement initial partition blocks with DACs. Further, in Sec. 5.2, we propose the postponed construction allowing to use automata reduction on intermediate results. In Sec. 5.3, we propose the round-robin algorithm alleviating the problem with the explosion of the size of the Cartesian product of partial successors. In Sec. 5.4, we provide an optimization for partial algorithms that are based on the breakpoint construction, and, finally, in Sec. 5.5, we show how to employ simulation to decrease the size of macrostates in the synchronous construction.

5.1 Complementation of Initial Deterministic Partition Blocks

Our first optimization is an algorithm CoB for a subclass of partition blocks containing DACs. In particular, the condition φ_{CoB} specifies that the partition block P is deterministic and can be reached only deterministically in $\mathcal{A}$ (i.e., $\mathcal{A}_P$ after removing redundant states is deterministic). Then, we say that P is an *initial deterministic* partition block. The algorithm is based on complementation of deterministic BAs into co-Büchi automata.

The algorithm CoB_P is formalized below:

- $\text{T}^{\text{CoB}_P} = P \cup \{\emptyset\}$, $\text{Init}^{\text{CoB}_P} = I \cap P$, $\text{Colours}^{\text{CoB}_P} = \{🔵\}$, $\text{Acc}^{\text{CoB}_P} = \text{Fin}(🔵)$,

– $\text{Succ}^{\text{CoB}_P}(H, q, a) = \{(q', \alpha)\}$ where

- $q' = \begin{cases} r & \text{if } \delta(H, a) \cap P = \{r\} \text{ and} \\ \emptyset & \text{otherwise,} \end{cases}$ • $\alpha = \begin{cases} \{\textbf{0}\} & \text{if } q \xrightarrow{a} q' \in F \text{ and} \\ \emptyset & \text{otherwise.} \end{cases}$

Intuitively, all runs reach P deterministically, which means that over a word w, at most one run can reach P (so $|\text{Init}^{\text{CoB}_P}| = 1$). Thus, we have $|\delta(H, w_j) \cap P| = 1$ for some $j \geq 0$ if there is a run over w to P, corresponding to $\delta(H, a) \cap P = \{r\}$ in the construction. To check whether w is not accepted in P, we only need to check whether the run from $r \in P$ over w visits accepting transitions only finitely often. We give an example of complementation of a BA containing an initial deterministic partition block in [27].

Lemma 3. *The partial algorithm* CoB *is correct.*

5.2 Postponed Construction

The modular synchronous construction from Sec. 3.1 utilizes the assumption that in the simultaneous construction of successors for each partition block over a, if one partial macrostate M_i does not have a successor over a, then there will be no successor of the $(H, M_1, \ldots, M_n)$ macrostate in δ^C as well. This is useful, e.g., for inclusion testing, where it is not necessary to generate the whole complement. On the other hand, if we need to generate the whole automaton, a drawback of the proposed modular construction is that each partial complementation algorithm itself may generate a lot of useless states. In this section, we propose the *postponed construction*, which complements the partition blocks (with their surrounding) independently and later combines the intermediate results to obtain the complement automaton for $\mathcal{A}$. The main advantage of the postponed construction is that one can apply automata reduction (e.g., based on removing useless states or using simulation [13,18,1,9]) to decrease the size of the intermediate automata.

In the postponed construction, we use product-based BA intersection operation (i.e., for two TELAs $\mathcal{B}_1$ and $\mathcal{B}_2$, a product automaton $\mathcal{B}_1 \cap \mathcal{B}_2$ satisfying $\mathcal{L}(\mathcal{B}_1 \cap \mathcal{B}_2) = \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$[10]). Further, we employ a function Red performing some language-preserving reduction of an input TELA. Then, the postponed construction for an elevator automaton $\mathcal{A}$ with a partitioning $P_1, \ldots, P_n$ and a sequence $\text{Alg}^1, \ldots, \text{Alg}^n$ where Alg^i is a partial complementation algorithm for P_i, is defined as follows:

$$\text{PostpCompl}(\text{Alg}^1_{P_1}, \ldots, \text{Alg}^n_{P_n}, \mathcal{A}) = \bigcap_{i=1}^{n} \text{Red}\left(\text{ModCompl}(\text{Alg}^i_{P_i}, \mathcal{A}_{P_i})\right). \quad (2)$$

The correctness of the construction is then summarized by the following theorem.

Theorem 3. *Let $\mathcal{A}$ be a BA, $P_1, \ldots, P_n$ be a partitioning of $\mathcal{A}$, and $\text{Alg}^1, \ldots, \text{Alg}^n$ be a sequence of partial complementation algorithms such that Alg^i is correct for P_i. Then, $\mathcal{L}(\text{PostpCompl}(\text{Alg}^1_{P_1}, \ldots, \text{Alg}^n_{P_n}, \mathcal{A})) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.*

5.3 Round-Robin Algorithm

The proposed basic synchronous approach from Sec. 3.1 may suffer from the combinatorial explosion because the successors of a macrostate are given by the Cartesian product of all successors of the partial macrostates. To alleviate this explosion, we propose

---

[10] Alternatively, one might also avoid the product and generate linear-sized *alternating* TELA, but working with those is usually much harder and not used in practice.

a *round-robin* top-level algorithm. Intuitively, the round-robin algorithm actively tracks runs in only one partial complementation algorithm at a time (while other algorithms stay passive). The algorithm periodically changes the active algorithm to avoid starvation (the decision to leave the active state is, however, fully directed by the partial complementation algorithm). This can alleviate an explosion in the number of successors for algorithms that generate more than one successor (e.g., for rank-based algorithms where one needs to make a nondeterministic choice of decreasing ranks of states in order to be able to accept [34,21,48,10,24,29]; such a choice needs to be made only in the active phase while in the passive phase, the construction just needs to make sure that the run is consistent with the given ranking, which can be done deterministically).

The round-robin algorithm works on the level of *partial complementation round-robin algorithms*. Each instance of the partial algorithm provides *passive types* to represent partial macrostates that are passive and *active types* to represent currently active partial macrostates. In contrast to the basic partial complementation algorithms from Sec. 3.1, which provide only a single successor function, the round-robin partial algorithms provide several variants of them. In particular, `SuccPass` returns (passive) successors of a passive partial macrostate, `Lift` gives all possible active counterparts of a passive macrostate, and `SuccAct` returns successors of an active partial macrostate. If `SuccAct` returns a partial macrostate of the passive type, the round-robin algorithm promotes the next partial algorithm to be the active one. For instance, in the round-robin version of CSB, the passive type does not contain the breakpoint and only checks that safe runs stay safe, so it is deterministic. Due to space limitations, we give a formal definition and more details about the round-robin algorithm in [27].

5.4 Shared Breakpoint

The partial complementation algorithms CSB and MH (and later RNK defined in Sec. 6) use a breakpoint to check whether the runs under inspection are accepting or not. As an optimization, we consider merging of breakpoints of several algorithms and keeping only a single breakpoint for all supported algorithms. The top-level algorithm then needs to manage only one breakpoint and emit a colour only if this sole breakpoint becomes empty. This may lead to a smaller number of generated macrostates since we synchronize the breakpoint sampling among several algorithms. The second benefit is that this allows us to generate fewer colours (in the case of elevator automata complemented using algorithms CSB and MH, we get only one colour).

5.5 Simulation Pruning

Our construction can be further optimized by a simulation (or other compatible) relation for pruning macrostates.[II] A simulation is, broadly speaking, a relation $\preccurlyeq \subseteq Q \times Q$ implying language inclusion of states, i.e., $\forall p, q \in Q: p \preccurlyeq q \implies \mathcal{L}(\mathcal{A}[p]) \subseteq \mathcal{L}(\mathcal{A}[q])$. Intuitively, our optimization allows to remove a state p from a macrostate M if there is also a state q in M such that (i) $p \preccurlyeq q$, (ii) p is not reachable from q, and (iii) p is smaller than q in an arbitrary total order over Q (this serves as a tie-breaker for

---

[II] This optimization can be seen as a generalization of the simulation-based pruning techniques that appeared, e.g., in [41,28] in the context of concrete determinization/complementation procedures. Here, we generalize the technique to all procedures that are based on run tracking.

simulation-equivalent mutually unreachable states). The reason why p can be removed is that its behaviour can be completely mimicked by q. In our construction, we can then, roughly speaking, replace each call to the functions $\delta(U, a)$ and $\delta_F(U, a)$, for a set of states U, by $pr(\delta(U, a))$ and $pr(\delta_F(U, a))$ respectively in each partial complementation algorithm, as well as in the top-level algorithm, where $pr(S)$ is obtained from S by pruning all eligible states. The details are provided in [27].

6   Modular Complementation of Non-Elevator Automata

A non-elevator automaton $\mathcal{A}$ contains at least one NAC, besides possibly other IWCs or DACs. To complement $\mathcal{A}$ in a modular way, we apply the techniques seen in Sec. 4 to its DACs and IWCs, while for its NACs we resort to a general complementation algorithm `Alg`. In theory, rank- [34], slice- [32], Ramsey- [50], subset-tuple- [2], and determinization- [46] based complementation algorithms adapted to work on a single partition block instead of the whole automaton are all valid instantiations of `Alg`. Below, we give a high-level description of two such algorithms: rank- and determinization-based.

Rank-based partial complementation algorithm. Working on each NAC independently benefits the complementation algorithm even if the input BA contains only NACs. For instance, in rank-based algorithms [34,21,48,33,10,24,29], the fact whether all runs of $\mathcal{A}$ over a given ω-word w are non-accepting is determined by *ranks* of states, given by the so-called *ranking functions*. A ranking function is a (partial) function from Q to ω. The main idea of rank-based algorithms is the following: (i) every run is initially nondeterministically assigned a rank, (ii) ranks can only decrease along a run, (iii) ranks need to be even every time a run visits an accepting transition, and (iv) the complement automaton accepts iff all runs eventually get trapped in odd ranks[12]. In the standard rank-based procedure, the initial assignment of ranks to states in (i) is a function $Q \rightharpoonup \{0, \dots, 2n - 1\}$ for $n = |Q|$. Using our framework, we can, however, significantly restrict the considered ranks in a partition block P to only $P \rightharpoonup \{0, \dots, 2m - 1\}$ for $m = |P|$ (here, it makes sense to use partition blocks consisting of single SCCs). One can further reduce the considered ranks using the techniques introduced in, e.g., [24,29].

In order to adapt the rank-based construction as a partial complementation algorithm `RNK` in our framework, we need to extend the ranking functions by a fresh "box state" ∎ representing states outside the partition block. The ranking function then uses ∎ to represent ranks of runs newly coming into the partition block. The box-extension also requires to change the transition in a way that ∎ always represents reachable states from the outside. We provide the details of the construction, which includes the MaxRank optimization from [24], in [27].

Determinization-based partial complementation algorithm. In [52,29] we can see that determinization-based complementation is also a good instantiation of `Alg` in practice, so, we also consider the standard Safra-Piterman determinization [46,43,45] as a choice of `Alg` for complementing NACs. Determinization-based algorithms use a layered subset construction to organize all runs over an ω-word w. The idea is to identify a subset $S \subseteq H$ of reachable states that occur infinitely often along reading w such that between every two occurrences of S, we have that (i) every state in the second occurrence of S can be reached

---

[12] Since we focus on intuition here, we use runs rather than the directed acyclic graphs of runs.

Table 1: Statistics for our experiments. The column **unsolved** classifies unsolved instances by the form *timeouts : out of memory : other failures*. For the cases of VBS we provide just the number of unsolved cases. The columns **states** and **runtime** provide *mean : median* of the number of states and runtime, respectively.

tool	solved	unsolved	states	runtime
KOFOLA$_S$	39,738	89 : 10 : 0	76 : 3	0.32 : 0.03
KOFOLA$_P$	39,750	76 : 11 : 0	86 : 3	0.41 : 0.03
VBS$_+$	39,834	3	78 : 3	0.05 : 0.01
VBS$_-$	39,834	3	96 : 3	0.05 : 0.01

tool	solved	unsolved	states	runtime
COLA	39,814	21 : 0 : 2	80 : 3	0.17 : 0.02
RANKER	38,837	61 : 939 : 0	45 : 4	3.31 : 0.01
SEMINATOR	39,026	238 : 573 : 0	247 : 3	1.98 : 0.03
SPOT	39,827	8 : 0 : 2	160 : 4	0.08 : 0.02

by a state in the first occurrence of S and (ii) every state in the second occurrence is reached by a state in the first occurrence while seeing an accepting transition. According to König's lemma, there must then be an accepting run of $\mathcal{A}$ over w.

The construction initially maintains only one set H: the set of reachable states. Since S as defined does not necessarily need to be H, every time there are runs visiting accepting transitions, we create a new subset C for those runs and remember which subset C is coming from. This way, we actually organize the current states of all runs into a tree structure and do subset construction in parallel for the sets in each tree node. If we find a tree node whose labelled subset, say S', is equal to the union of states in its children, we know the set S' satisfies the condition above and we remove all its child nodes and emit a good event. If such good event happens infinitely often, it means that S' also occurs infinitely often. So in complementation, we only need to make sure those good events only happen for finitely many times. Working on each NAC separately also benefits the determinization-based approach since the number of possible trees will be less with smaller number of reachable states. Following the idea of [37], to adapt for the construction as the partial complementation algorithm, we put all the newly coming runs from other partition blocks in a newly created node without a parent node. In this way, we actually maintain a forest of trees for the partial complementation construction. We denote the determinization-based construction as DET; cf. [37] for details.

7   Experimental Evaluation

To evaluate the proposed approach, we implemented it in a prototype tool KOFOLA [25] (written in C++) built on top of SPOT [16] and compared it against COLA [37], RANKER [28] (v. 2), SEMINATOR [5] (v. 2.0), and SPOT [15,16] (v. 2.10.6), which are the state of the art in BA complementation [29,28,37]. Due to space restrictions, we give results for only two instantiations of our framework: KOFOLA$_S$ and KOFOLA$_P$. Both instantiations use MH for IWCs, CSB for DACs, and DET for NACs. The partitioning selection algorithm merges all IWCs into one partition block, all DACs into one partition block, and keeps all NACs separate. Simulation-based pruning from Sec. 5.5 is turned on, and round-robin from Sec. 5.3 is turned off (since the selected algorithms are quite deterministic). KOFOLA$_S$ employs the *synchronous* and KOFOLA$_P$ employs the *postponed* strategy. We also consider the Virtual Best Solver (VBS), i.e., a virtual tool that would choose the best solver for each single benchmark among all tools (VBS$_+$) and among all tools except both versions of KOFOLA (VBS$_-$). We ran our experiments on an Ubuntu 20.04.4 LTS system running on a desktop machine with 16 GiB RAM and an
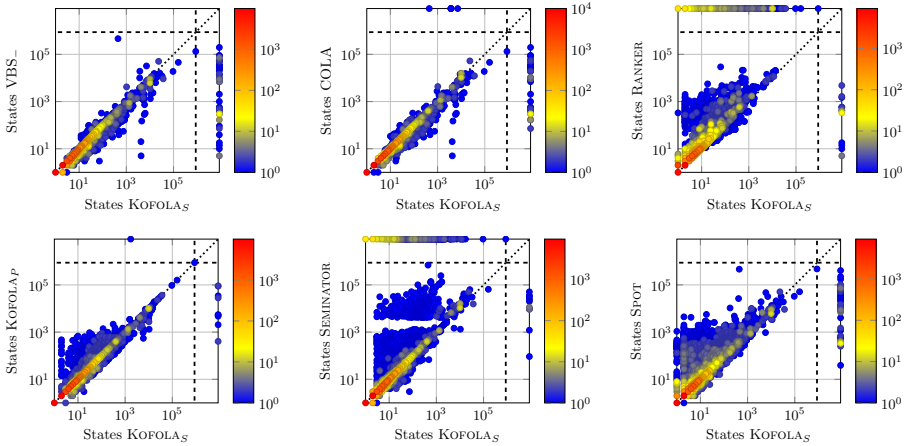
Fig. 2: Scatter plots comparing the numbers of states generated by the tools.

Intel 3.6 GHz i7-4790 CPU. To constrain and collect statistics about the executions of the tools, we used BenchExec [3] and imposed a memory limit of 12 GiB and a timeout of 10 minutes; we used Spot to cross-validate the equivalence of the automata generated by the different tools. An artifact reproducing our experiments is available as [26].

As our data set, we used 39,837 BAs from the automata-benchmarks repository [36] (used before by, e.g., [29,28,37]), which contains BAs from the following sources: (i) randomly generated BAs used in [52] (21,876 BAs), (ii) BAs obtained from LTL formulae from the literature and randomly generated LTL formulae [5] (3,442 BAs), (iii) BAs obtained from Ultimate Automizer [11] (915 BAs), (iv) BAs obtained from the solver for first-order logic over Sturmian words Pecan [31] (13,216 BAs), (v) BAs obtained from an S1S solver [23] (370 BAs), and (vi) BAs from LTL to SDBA translation [49] (18 BAs). From these BAs, 23,850 are deterministic, 6,147 are SDBAs (but not deterministic), 4,105 are elevator (but not SDBAs), and 5,735 are the rest.

In Table 1 we present an overview of the outcomes. Despite being a prototype, Kofola can already complement a large portion of the input automata, with very few cases that can be complemented successfully only by Spot or COLA. Regarding the mean number of states, Kofola_S has the **least mean value** from all tools (except Ranker, which, however, had 1,000 unsolved cases) Moreover, Kofola **significantly decreased the mean number of states** when included into the VBS: from 96 to 78! We consider this to be a strong validation of the usefulness of our approach. Regarding the runtime, both versions of Kofola are rather similar; Kofola is just slightly slower than Spot and COLA but much faster than both Ranker and Seminator (cf. [27]).

In Fig. 2 we present a comparison of the number of states generated by Kofola_S and other tools; we omit VBS_+ since the corresponding plot can be derived from the one for VBS_- (since Ranker and Seminator only output BAs, we compare the sizes of outputs transformed into BAs for all tools to be fair). In the plots, the number of benchmarks represented by each mark is given by its colour; a mark above the diagonal means that Kofola_S generated a BA smaller than the other tool while a mark on the top border means that the other tool failed while Kofola_S succeeded, and symmetrically for the

bottom part and the right-hand border. Dashed lines represent the maximum number of states generated by one of the tools in the plot, axes are logarithmic.

From the results, KOFOLA$_S$ clearly domi-nates state-of-the-art tools that are not based on SCC decomposition (RANKER, SPOT, SEM-INATOR). The outputs are quite comparable to COLA, which also uses SCC decomposition and can be seen as an instantiation of our frame-work. This supports our intuition that working on the single SCCs helps in reducing the size of the final automaton, confirming the validity of our modular mix-and-match Büchi comple-

mentation approach. Lastly, in the figure in the right we compare our algorithm for elevator automata with the one in RANKER (the only other tool with a dedicated algo-rithm for this subclass). Our new algorithm clearly dominates the one in RANKER.
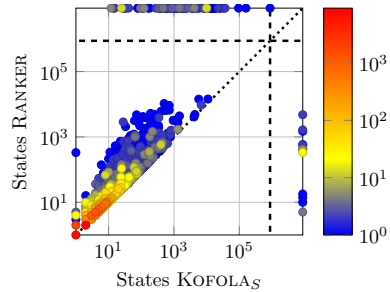
8   Related Work

To the best of our knowledge, we provide the *first general framework* where one can plug-in different BA complementation algorithms while taking advantage of the specific structure of SCCs. We will discuss the difference between our work and the literature.

The breakpoint construction [42] was designed to complement BAs with only IWCs, while our construction treats it as a partial complementation procedure for IWCs and differs in the need to handle incoming states from other partition blocks. The NCSB family of algorithms [6,11,5,28] for SDBAs do not work when there are nondeterministic jumps between DACs; they can, however, be adapted as partial procedures for comple-menting DACs in our framework, cf. Sec. 4.2. In [29], a deelevation-based procedure is applied to elevator automata to obtain BAs with a fixed maximum rank of 3, for which a rank-based construction produces a result of the size in $O(16^n)$. In our work, we exploit the structure of the SCCs much more to obtain an exponentially better upper bound of $O(4^n)$ (the same as for SDBAs). The upper bound $O(4^n)$ for complementing unambiguous BAs was established in [39], which is orthogonal to our work, but seems to be possible to incorporate into our framework in the future.

There is a huge body of work on complementation of general BAs [8,50,7,34,21,22,10,24,29,48,2,46,43,45,5,52,32,53,19,20]; all of them work on the whole graph structure of the input BAs. Our framework is general enough to allow including all of them as partial complementation procedures for NACs. On the contrary, our framework does not directly allow (at least in the synchronous strategy) to use al-gorithms that *do not* work on the structure of the input BA, such as the learning-based complementation algorithm from [38]. The recent determinization algorithm from [37], which serves as our inspiration, also handles SCCs separately (it can actually be seen as an instantiation of our framework). Our current algorithm is, however, more flex-ible, allowing to mix-and-match various constructions, keep SCCs separate or merge them into partition blocks, and allows to obtain the complexity $O(4^n)$, while [37] only allowed $O(n!)$ (which is tight since SDBA determinization is in $\Omega(n!)$ [17,40]).

Regarding the tool SPOT [15,16], it should not be perceived as a single comple-mentation algorithm. Instead, SPOT should be seen as a highly engineered platform

utilizing breakpoint construction for inherently weak BAs, NCSB [6,11] for SDBAs, and determinization-based complementation [46,43,45] for general BAs, while using many other heuristics along the way. SEMINATOR uses semi-determinization [14,4,5] to make sure the input is an SDBA and then uses NCSB [6,11] to compute the complement.

9   Conclusion and Future Work

We have proposed a general framework for BA complementation where one can plug-in different partial complementation procedures for SCCs by taking advantage of their specific structure. Our framework not only obtains an exponentially better upper bound for elevator automata, but also complements existing approaches well. As shown by the experimental results (especially for the VBS), our framework significantly improves the current portfolio of complementation algorithms.

We believe that our framework is an ideal testbed for experimenting with different BA complementation algorithms, e.g., for the following two reasons: (i) One can develop an efficient complementation algorithm that only works for a quite restricted sub-class of BAs (such as the algorithm for initial deterministic SCCs that we showed in Sec. 5.1) and the framework can leverage it for complementation of all BAs that contain such a substructure. (ii) When one tries to improve a general complementation algorithm, they can focus on complementation of the structurally hard SCCs (mainly the nondeterministic accepting SCCs) and do not need to look for heuristics that would improve the algorithm if there were some easier substructure present in the input BA (as was done, e.g., in [29]). From how the framework is defined, it immediately offers opportunities for being used for on-the-fly BA *language inclusion* testing, leveraging the partial complementation procedures present. Finally, we believe that the framework also enables new directions for future research by developing smart ways, probably based on machine learning, of selecting which partial complementation procedure should be used for which SCC, based on their features. In future, we want to incorporate other algorithms for complementation of NACs, and identify properties of SCCs that allow to use more efficient algorithms (such as unambiguous NACs [39]). Moreover, it seems that generalizing the DELAYED optimization from [24] on the top-level algorithm could also help reduce the state space.

References

1. Abdulla, P.A., Chen, Y., Holík, L., Vojnar, T.: Mediating for reduction (on minimizing alternating büchi automata). Theor. Comput. Sci. **552**, 26–43 (2014). https://doi.org/10.1016/j.tcs.2014.08.003, https://doi.org/10.1016/j.tcs.2014.08.003

2. Allred, J.D., Ultes-Nitsche, U.: A simple and optimal complementation algorithm for Büchi automata. In: Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018. pp. 46–55. ACM (2018). https://doi.org/10.1145/3209108.3209138, https://doi.org/10.1145/3209108.3209138

3. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: requirements and solutions. Int. J. Softw. Tools Technol. Transf. **21**(1), 1–29 (2019). https://doi.org/10.1007/s10009-017-0469-y, https://doi.org/10.1007/s10009-017-0469-y

4. Blahoudek, F., Duret-Lutz, A., Klokocka, M., Kretínský, M., Strejcek, J.: Seminator: A tool for semi-determinization of omega-automata. In: Eiter, T., Sands, D. (eds.) LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017. EPiC Series in Computing, vol. 46, pp. 356–367. Easy-Chair (2017). https://doi.org/10.29007/k5nl, https://doi.org/10.29007/k5nl

5. Blahoudek, F., Duret-Lutz, A., Strejcek, J.: Seminator 2 can complement generalized Büchi automata via improved semi-determinization. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12225, pp. 15–27. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_2, https://doi.org/10.1007/978-3-030-53291-8_2

6. Blahoudek, F., Heizmann, M., Schewe, S., Strejček, J., Tsai, M.: Complementing semi-deterministic Büchi automata. In: Chechik, M., Raskin, J. (eds.) 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, https://doi.org/10.1007/978-3-662-49674-9_49

7. Breuers, S., Löding, C., Olschewski, J.: Improved Ramsey-based Büchi complementation. In: Birkedal, L. (ed.) Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7213, pp. 150–164. Springer (2012). https://doi.org/10.1007/978-3-642-28729-9_10, https://doi.org/10.1007/978-3-642-28729-9_10

8. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Mac Lane, S., Siefkes, D. (eds.) The Collected Works of J. Richard Büchi, pp. 425–435. Springer (1990). https://doi.org/10.1007/978-1-4613-8928-6_23, https://doi.org/10.1007/978-1-4613-8928-6_23

9. Bustan, D., Grumberg, O.: Simulation-based Minimization. ACM Transactions on Computational Logic **4**(2), 181–206 (2003)

10. Chen, Y., Havlena, V., Lengál, O.: Simulations in rank-based Büchi automata complementation. In: Lin, A.W. (ed.) 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, https://doi.org/10.1007/978-3-030-34175-6_23

11. Chen, Y., Heizmann, M., Lengál, O., Li, Y., Tsai, M., Turrini, A., Zhang, L.: Advanced automata-based algorithms for program termination checking. In: Foster, J.S., Grossman, D. (eds.) 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, https://doi.org/10.1145/3192366.3192405

12. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8414, pp. 265–284. Springer (2014). https://doi.org/10.1007/978-3-642-54792-8_15, https://doi.org/10.1007/978-3-642-54792-8_15

13. Clemente, L., Mayr, R.: Efficient reduction of nondeterministic automata with application to language inclusion testing. Log. Methods Comput. Sci. **15**(1) (2019). https://doi.org/10.23638/LMCS-15(1:12)2019, https://doi.org/10.23638/LMCS-15(1:12)2019

14. 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, https://doi.org/10.1109/SFCS.1988.21950

15. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 - A framework for LTL and ω-automata manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9938, pp. 122–129 (2016). https://doi.org/10.1007/978-3-319-46520-3_8, https://doi.org/10.1007/978-3-319-46520-3_8

16. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From Spot 2.0 to Spot 2.10: What's new? In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13372, pp. 174–187. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_9, https://doi.org/10.1007/978-3-031-13188-2_9

17. Esparza, J., Kretínský, J., Raskin, J., Sickert, S.: From LTL and limit-deterministic Büchi automata to deterministic parity automata. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10205, pp. 426–442 (2017). https://doi.org/10.1007/978-3-662-54577-5_25, https://doi.org/10.1007/978-3-662-54577-5_25

18. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM J. Comput. **34**(5), 1159–1175 (2005). https://doi.org/10.1137/S0097539703420675, https://doi.org/10.1137/S0097539703420675

19. Fogarty, S., Kupferman, O., Vardi, M.Y., Wilke, T.: Profile trees for Büchi word automata, with application to determinization. Inf. Comput. **245**, 136–151 (2015). https://doi.org/10.1016/j.ic.2014.12.021, https://doi.org/10.1016/j.ic.2014.12.021

20. Fogarty, S., Kupferman, O., Wilke, T., Vardi, M.Y.: Unifying Büchi complementation constructions. Log. Methods Comput. Sci. **9**(1) (2013). https://doi.org/10.2168/LMCS-9(1:13)2013, https://doi.org/10.2168/LMCS-9(1:13)2013

21. Friedgut, E., Kupferman, O., Vardi, M.Y.: Büchi complementation made tighter. Int. J. Found. Comput. Sci. **17**(4), 851–868 (2006). https://doi.org/10.1142/S0129054106004145, https://doi.org/10.1142/S0129054106004145

22. Gurumurthy, S., Kupferman, O., Somenzi, F., Vardi, M.Y.: On complementing nondeterministic Büchi automata. In: Geist, D., Tronci, E. (eds.) 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. Lecture Notes in Computer Science, vol. 2860, pp. 96–110. Springer (2003). https://doi.org/10.1007/978-3-540-39724-3_10, https://doi.org/10.1007/978-3-540-39724-3_10

23. Havlena, V., Lengál, O., Smahlíková, B.: Deciding S1S: down the rabbit hole and through the looking glass. In: Echihabi, K., Meyer, R. (eds.) Networked Systems - 9th International Conference, NETYS 2021, Virtual Event, May 19-21, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12754, pp. 215–222. Springer (2021). https://doi.org/10.1007/978-3-030-91014-3_15, https://doi.org/10.1007/978-3-030-91014-3_15

24. Havlena, V., Lengál, O.: Reducing (to) the ranks: Efficient rank-based Büchi automata complementation. In: Haddad, S., Varacca, D. (eds.) 32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference. LIPIcs, vol. 203, pp. 2:1–2:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPIcs.CONCUR.2021.2, https://doi.org/10.4230/LIPIcs.CONCUR.2021.2

25. Havlena, V., Lengál, O., Li, Y., Šmahlíková, B., Turrini, A.: Kofola (2022), https://github.com/VeriFIT/kofola

26. Havlena, V., Lengál, O., Li, Y., Šmahlíková, B., Turrini, A.: Artifact for the TACAS'23 paper "Modular Mix-and-Match Complementation of Büchi Automata" (Jan 2023). https://doi.org/10.5281/zenodo.7505210, https://doi.org/10.5281/zenodo.7505210

27. Havlena, V., Lengál, O., Li, Y., Šmahlíková, B., Turrini, A.: Modular mix-and-match complementation of Büchi automata (technical report). CoRR **abs/2301.01890** (2023). https://doi.org/10.48550/arXiv.2301.01890, https://doi.org/10.48550/arXiv.2301.01890

28. Havlena, V., Lengál, O., Šmahlíková, B.: Complementing Büchi automata with Ranker. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13372, pp. 188–201. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_10, https://doi.org/10.1007/978-3-031-13188-2_10

29. Havlena, V., Lengál, O., Šmahlíková, B.: Sky is not the limit: Tighter rank bounds for elevator automata in Büchi automata complementation. In: Fisman, D., Rosu, G. (eds.) 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. Lecture Notes in Computer Science, vol. 13244, pp. 118–136. Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_7, https://doi.org/10.1007/978-3-030-99527-0_7

30. Heizmann, M., Hoenicke, J., Podelski, A.: Termination analysis by learning terminating programs. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 797–813. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_53, https://doi.org/10.1007/978-3-319-08867-9_53

31. Hieronymi, P., Ma, D., Oei, R., Schaeffer, L., Schulz, C., Shallit, J.O.: Decidability for Sturmian words. In: Manea, F., Simpson, A. (eds.) 30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference). LIPIcs, vol. 216, pp. 24:1–24:23. Schloss Dagstuhl - Leibniz-Zentrum für

Informatik (2022). https://doi.org/10.4230/LIPIcs.CSL.2022.24, https://doi.org/10.4230/LIPIcs.CSL.2022.24

32. Kähler, D., Wilke, T.: Complementation, disambiguation, and determinization of Büchi automata unified. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games. Lecture Notes in Computer Science, vol. 5125, pp. 724–735. Springer (2008). https://doi.org/10.1007/978-3-540-70575-8_59, https://doi.org/10.1007/978-3-540-70575-8_59

33. Karmarkar, H., Chakraborty, S.: On minimal odd rankings for Büchi complementation. In: Liu, Z., Ravn, A.P. (eds.) Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5799, pp. 228–243. Springer (2009). https://doi.org/10.1007/978-3-642-04761-9_18, https://doi.org/10.1007/978-3-642-04761-9_18

34. 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, https://doi.org/10.1145/377978.377993

35. 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, https://doi.org/10.1016/0022-0000(87)90036-5

36. Lengál, O.: Automata benchmarks (2022), https://github.com/ondrik/automata-benchmarks

37. Li, Y., Turrini, A., Feng, W., Vardi, M.Y., Zhang, L.: Divide-and-conquer determinization of Büchi automata based on SCC decomposition. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13372, pp. 152–173. Springer (2022). https://doi.org/10.1007/978-3-031-13188-2_8, https://doi.org/10.1007/978-3-031-13188-2_8

38. Li, Y., Turrini, A., Zhang, L., Schewe, S.: Learning to complement Büchi automata. In: Dillig, I., Palsberg, J. (eds.) Verification, Model Checking, and Abstract Interpretation - 19th International Conference, VMCAI 2018, Los Angeles, CA, USA, January 7-9, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10747, pp. 313–335. Springer (2018). https://doi.org/10.1007/978-3-319-73721-8_15, https://doi.org/10.1007/978-3-319-73721-8_15

39. Li, Y., Vardi, M.Y., Zhang, L.: On the power of unambiguity in Büchi complementation. In: Raskin, J., Bresolin, D. (eds.) Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2020, Brussels, Belgium, September 21-22, 2020. EPTCS, vol. 326, pp. 182–198 (2020). https://doi.org/10.4204/EPTCS.326.12, https://doi.org/10.4204/EPTCS.326.12

40. Löding, C.: Optimal bounds for transformations of omega-automata. In: Rangan, C.P., Raman, V., Ramanujam, R. (eds.) Foundations of Software Technology and Theoretical Computer Science, 19th Conference, Chennai, India, December 13-15, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1738, pp. 97–109. Springer (1999). https://doi.org/10.1007/3-540-46691-6_8, https://doi.org/10.1007/3-540-46691-6_8

41. Löding, C., Pirogov, A.: New optimizations and heuristics for determinization of Büchi automata. In: Chen, Y., Cheng, C., Esparza, J. (eds.) Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11781, pp. 317–333. Springer

(2019). https://doi.org/10.1007/978-3-030-31784-3_18, https://doi.org/10.1007/978-3-030-31784-3_18

42. Miyano, S., Hayashi, T.: Alternating finite automata on omega-words. Theor. Comput. Sci. **32**, 321–330 (1984). https://doi.org/10.1016/0304-3975(84)90049-5, https://doi.org/10.1016/0304-3975(84)90049-5

43. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. Log. Methods Comput. Sci. **3**(3) (2007). https://doi.org/10.2168/LMCS-3(3:5)2007, https://doi.org/10.2168/LMCS-3(3:5)2007

44. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32, https://doi.org/10.1109/SFCS.1977.32

45. 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, https://doi.org/10.3233/FI-2012-744

46. Safra, S.: On the complexity of omega-automata. In: 29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988. pp. 319–327. IEEE Computer Society (1988). https://doi.org/10.1109/SFCS.1988.21948, https://doi.org/10.1109/SFCS.1988.21948

47. Safra, S., Vardi, M.Y.: On omega-automata and temporal logic (preliminary report). In: Johnson, D.S. (ed.) Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA. pp. 127–137. ACM (1989). https://doi.org/10.1145/73007.73019, https://doi.org/10.1145/73007.73019

48. Schewe, S.: Büchi complementation made tight. In: Albers, S., Marion, J. (eds.) 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings. LIPIcs, vol. 3, pp. 661–672. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2009). https://doi.org/10.4230/LIPIcs.STACS.2009.1854, https://doi.org/10.4230/LIPIcs.STACS.2009.1854

49. Sickert, S., Esparza, J., Jaax, S., Kretínský, J.: Limit-deterministic Büchi automata for linear temporal logic. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9780, pp. 312–332. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_17, https://doi.org/10.1007/978-3-319-41540-6_17

50. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theor. Comput. Sci. **49**, 217–237 (1987). https://doi.org/10.1016/0304-3975(87)90008-9, https://doi.org/10.1016/0304-3975(87)90008-9

51. The SV-COMP Community: International competition on software verification (2022), https://sv-comp.sosy-lab.org/

52. Tsai, M., Fogarty, S., Vardi, M.Y., Tsay, Y.: State of Büchi complementation. Log. Methods Comput. Sci. **10**(4) (2014). https://doi.org/10.2168/LMCS-10(4:13)2014, https://doi.org/10.2168/LMCS-10(4:13)2014

53. Vardi, M.Y., Wilke, T.: Automata: from logics to algorithms. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and Automata: History and Perspectives. Texts in Logic and Games, vol. 2, pp. 629–736. Amsterdam University Press (2008)

54. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986. pp. 332–344. IEEE Computer Society (1986)

55. Yan, Q.: Lower bounds for complementation of omega-automata via the full automata technique. Log. Methods Comput. Sci. **4**(1) (2008). https://doi.org/10.2168/LMCS-4(1:5)2008, https://doi.org/10.2168/LMCS-4(1:5)2008