Complementing Emerson-Lei Elevator Automata

Ondrej Alexaj
Brno University of Technology, Czech Republic

Vojtěch Havlena
Brno University of Technology, Czech Republic

Ondřej Lengál
Brno University of Technology, Czech Republic

Yong Li
Key Laboratory of System Software (Chinese Academy of Sciences), ISCAS, China

Nicolas Mazzocchi
Slovak University of Technology in Bratislava, Slovak Republic

Abstract

Büchi elevator automata naturally appear in several areas of formal methods as a structural expressibly-equivalent subclass of Büchi automata where every strongly connected component is either deterministic or inherently weak. It was shown that this class contains the majority of Büchi automata generated in practical applications, including LTL model-checking and verification of hyperproperties. Moreover, the elevator subclass enables more efficient complementation and determinization algorithms than unrestricted Büchi automata. In this paper, we introduce *Emerson-Lei elevator automata*, which is a generalization of Büchi elevator automata to richer acceptance conditions. We provide a complementation algorithm with a significantly better asymptotic complexity than the best known algorithm for unrestricted Emerson-Lei automata. The practical efficiency of our algorithm is demonstrated by an experimental comparison with the popular state-of-the-art tool Spot. Our work is, to the best of our knowledge, the first step towards practical algorithms for complementing, determinizing, and testing universality and inclusion of Emerson-Lei automata with rich acceptance conditions.

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

Keywords and phrases Emerson-Lei elevator automata, complementation, elevator automata, omega automata, infinite words, omega-regular languages

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2026.27

Related Version *Technical Report*: `http://arxiv.org/abs/2606.26768` [2]

Supplementary Material *Software (Source Code)*: `https://github.com/VeriFIT/kofola`

Funding This work was supported in part by the CAS Project for Young Scientists in Basic Research (Grant No. YSBR-040), the National Natural Science Foundation of China (Grant No. 62102407), the Beijing Natural Science Foundation (Grant No. IS26039), the Czech Science Foundation project 26-22640S, and the FIT BUT internal project FIT-S-26-9011.

1 Introduction

The fundamental framework of automata on infinite words (ω-automata)—such as Büchi automata (BAs), Rabin automata, or parity automata—has proven successful for verifying software properties [25, 26, 45, 27, 52]. Complementation of these automata is an operation used to analyze program termination [27], model-check temporal logics (e.g., QPTL [31], HyperLTL [13]), and decide logics (e.g., S1S [7], or fragments of first-order logic over Sturmian words [28]). Moreover, it is the underlying operation for determining whether the language of one ω-automaton is included in, or equivalent to, that of another. Yet, in the general

Table 1 Upper bounds on the size of $\mathcal{A}^\sharp$, the complement of $\mathcal{A}$ having n states and k colors.

Acceptance condition	State-space of $\mathcal{A}^\sharp$ when $\mathcal{A}$ is unrestricted	State-space of $\mathcal{A}^\sharp$ when $\mathcal{A}$ is elevator
Büchi	$\mathcal{O}(n(0.76n)^n)$ [49]	4^n [21]
gen. Büchi	$\mathcal{O}(n(0.76nk)^n)$ [24]	$(k+3)^n$
Rabin (ℓ pairs)	$\mathcal{O}(n^\ell(0.76n)^{n\ell})$ [24]	$3 \cdot 2^{(3\ell+2)n}$
gen. Rabin (ℓ pairs)	$\mathcal{O}(n^\ell(0.76kn)^{\ell n})$ [24]	$3 \cdot 2^{(3k\ell+2)n}$
Streett (ℓ pairs)	$2^{\Theta(n \log n + nk \log k)}$ [8, 9]	$6 \cdot \ell^n \cdot 2^{(2\ell+2)n}$
Parity (min-odd index ℓ)	$2^{\mathcal{O}(n \log n)}$ [9]	$3 \cdot 2^{(\frac{3}{2}\ell+2)n}$
Emerson-Lei (ℓ atoms)	$\mathcal{O}(n^{2^k}(0.76nk)^{n2^k})$ [24]	$3\ell \cdot 2^{(2\ell+3)n}$

case, complementation is notoriously challenging, and its difficulty has led the verification community to consider automata with structural restrictions, which allow more efficient algorithms. *Elevator automata* are BAs whose maximal strongly connected components (SCCs) are all deterministic or inherently weak (i.e., all runs staying in the SCC are either accepting or rejecting). This subclass was first identified in [23] as a generalization of the well-known semi-deterministic (a.k.a. limit-deterministic) automata [14]. Recent work [1] showed that the vast majority of BAs considered in practical applications, e.g., LTL model checking and verification of hyperproperties, are elevator automata.

Orthogonally, LTL model checking, synthesis, and other applications have raised interest in more sophisticated types of automata. Many types of ω-automata differ exclusively in their acceptance conditions [5]. The most general considered is that of *Emerson-Lei automata* (ELAs) [17], whose acceptance condition is an arbitrary Boolean combination of Fin and Inf predicates. The predicate $\mathsf{Fin}(\mathtt{c})$ denotes that all transitions labeled with the color $\mathtt{c}$ occur only finitely often along an accepting run. Dually, $\mathsf{Inf}(\mathtt{c})$ denotes that some transition labeled with $\mathtt{c}$ occurs infinitely often. For instance, BAs admit a single color $\mathtt{0}$, and their acceptance condition is $\mathsf{Inf}(\mathtt{0})$. The main motivation for using more complex acceptance conditions is the succinctness of the automaton representation, since ELAs can be exponentially more concise than BAs [48] (which is exploited already by some LTL-to-automata translators like `ltl3tela` [42]). Another motivation is algorithmic flexibility and operational costs. It is well known that BAs are not closed under determinization, whereas ELAs are; however, checking emptiness for ELAs is NP-complete, while it is in P for BAs [5].

Generalized Rabin automata (GRAs), whose acceptance condition is $\bigvee_i \mathsf{Fin}(\mathtt{0}_i) \wedge \mathsf{Inf}(\mathtt{1}_i) \wedge \cdots \wedge \mathsf{Inf}(\mathtt{k}_i)$[1], emerged from LTL model checking [36, 18, 10] thanks to their flexibility and conciseness. The practical use of GRAs highlights the need to refine the complexity of complementing sophisticated automata types. Recently, [24] improved the upper-bound when complementing ELAs and subclasses, including GRAs. We aim for more efficient constructions that leverage the elevator structure, both by benefiting from its frequent occurrence and by concisely converting non-elevator automata into elevator ones. For a given elevator automaton, Table 1 compares the upper-bounds on the size of its complement from best known algorithms for automata with unrestricted structure and the upper-bounds for elevator automata provided in this paper. It is clear that exploiting the prevalence of elevator automata can have a substantial impact on reducing the size of the complement automata, in particular, for ELAs, from double-exponential to single-exponential.

[1] I.e., in order for a run to be accepting, there needs to exist a clause $\mathsf{Fin}(\mathtt{0}_j) \wedge \mathsf{Inf}(\mathtt{1}_j) \wedge \cdots \wedge \mathsf{Inf}(\mathtt{k}_j)$ in the disjunction such that the run sees only finitely many occurrences of the color $\mathtt{0}_j$ and infinitely many occurrences of all colors $\mathtt{1}_j, \ldots, \mathtt{k}_j$.

The paper is organized as follows. In Section 2, we introduce Emerson-Lei elevator and semi-deterministic automata. In Sections 3 and 4, we describe the main ideas of our procedure on two special cases of ELAs: semi-deterministic automata with (i) one generalized Rabin pair and (ii) one generalized Streett pair. In Section 5, we present an inductive procedure for semi-deterministic automata with an arbitrary Emerson-Lei condition. In Section 6, we extend our techniques from semi-deterministic to elevator and unrestricted automata. In particular, our conversion treats automata that are "almost elevator" (i.e., where only a few SCCs are not elevator) more efficiently. This is helpful since in non-elevator automata from practice, most SCCs are elevator. In Section 7, we evaluate our complementation procedure, implemented in Kofola [1], and compare its performance to Spot [16]. Our benchmarks include ELAs from [42], randomly generated single-pair GRAs, and ELAs obtained from the translation of randomly generated LTL formulas. We show that the complements constructed by Kofola are often much smaller than those produced by Spot.

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 $w = w_0 w_1 \dots$ We denote the set of all infinite words over Σ as Σ^ω; an *ω-language* is a subset of Σ^ω. We use $\cdot$ for ellipsis, e.g., if interested only in the second component of a triple, we may write the triple as $(\cdot, x, \cdot)$.

2.1 Emerson-Lei Acceptance Conditions

Given a set $\Gamma = \{0, \dots, k-1\}$ of k *colors* (often depicted as 0, 1, etc.), we define the set of *Emerson-Lei acceptance conditions* $\mathbb{EL}(\Gamma)$ as the set of formulas constructed according to the following grammar where c ranges over Γ:

$$\alpha ::= \texttt{true} \mid \texttt{false} \mid \mathsf{Inf}(c) \mid \mathsf{Fin}(c) \mid (\alpha \wedge \alpha) \mid (\alpha \vee \alpha).$$

We denote by $|\alpha|$ the number of atomic conditions contained in α, where multiple occurrences of the same atomic condition are counted multiple times. Given an infinite sequence of sets of colors $\gamma\colon \omega \to 2^\Gamma$ and a condition $\alpha \in \mathbb{EL}(\Gamma)$, the *satisfaction* relation $\models$ is defined inductively as follows (with $c \in \Gamma$):

$$\gamma \models \texttt{true} \qquad \gamma \models \alpha_1 \vee \alpha_2 \text{ iff } \gamma \models \alpha_1 \text{ or } \gamma \models \alpha_2$$
$$\gamma \not\models \texttt{false} \qquad \gamma \models \alpha_1 \wedge \alpha_2 \text{ iff } \gamma \models \alpha_1 \text{ and } \gamma \models \alpha_2$$
$$\gamma \models \mathsf{Fin}(c) \text{ iff } |\{i \mid c \in \gamma(i)\}| < \infty \qquad \gamma \models \mathsf{Inf}(c) \text{ iff } |\{i \mid c \in \gamma(i)\}| = \infty.$$

2.2 Emerson-Lei Automata

A (nondeterministic) transition-based[2] *Emerson-Lei automaton* (ELA) over Σ is a tuple $\mathcal{A} = (\mathcal{Q}, \delta, I, \Gamma, \mathsf{p}, \mathsf{Acc})$, where $\mathcal{Q}$ is a finite set of *states*, $\delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a set of *transitions*, $I \subseteq \mathcal{Q}$ is the set of *initial* states, Γ is the set of *colors*, $\mathsf{p}\colon \delta \to 2^\Gamma$ is a *coloring* of transitions, and $\mathsf{Acc} \in \mathbb{EL}(\Gamma)$ is the acceptance condition. The negation of Acc, i.e., $\neg\mathsf{Acc}$, is obtained by exchanging $\mathsf{Inf}(c)$ with $\mathsf{Fin}(c)$ and $\wedge$ with $\vee$. We use $p \xrightarrow{a} q$ to denote that $(p, a, q) \in \delta$ and sometimes treat δ as a function $\delta\colon \mathcal{Q} \times \Sigma \to 2^\mathcal{Q}$. Moreover, we extend δ to sets of states $P \subseteq \mathcal{Q}$ as $\delta(P, a) = \bigcup_{p \in P} \delta(p, a)$. If $|I| \leq 1$ and $|\delta(q, a)| \leq 1$ for all states $q \in \mathcal{Q}$ and symbols $a \in \Sigma$, then $\mathcal{A}$ is *deterministic*. We define $|\mathcal{A}| = |\mathcal{Q}|$ and use $n = |\mathcal{Q}|$ in complexity bounds.

[2] Extending our approach to state-based or mixed state and transition-based acceptance is straightforward.

Table 2 Popular acceptance conditions

Name	Classic	Generalized
Büchi	$\mathrm{Inf}(0)$	$\bigwedge_{j=0}^{k-1} \mathrm{Inf}(j)$
co-Büchi	$\mathrm{Fin}(0)$	$\bigvee_{j=0}^{k-1} \mathrm{Fin}(j)$
Rabin pair	$\mathrm{Fin}(B) \wedge \mathrm{Inf}(G)$	$\mathrm{Fin}(B) \wedge \bigwedge_{\ell=0}^{m-1} \mathrm{Inf}(G_\ell)$
Rabin	$\bigvee_{j=0}^{k-1} \mathrm{Fin}(B_j) \wedge \mathrm{Inf}(G_j)$	$\bigvee_{j=0}^{k-1} (\mathrm{Fin}(B_j) \wedge \bigwedge_{\ell=0}^{m_j-1} \mathrm{Inf}(G_{j,\ell}))$
Streett pair	$\mathrm{Inf}(G) \vee \mathrm{Fin}(B)$	$\mathrm{Inf}(G) \vee \bigvee_{\ell=0}^{m-1} \mathrm{Fin}(B_\ell)$
Streett	$\bigwedge_{j=0}^{k-1} \mathrm{Inf}(G_j) \vee \mathrm{Fin}(B_j)$	$\bigwedge_{j=0}^{k-1} (\mathrm{Inf}(G_j) \vee \bigvee_{\ell=0}^{m_j-1} \mathrm{Fin}(B_{j,\ell}))$
Parity	$\mathrm{Fin}(0) \wedge (\mathrm{Inf}(1) \vee (\mathrm{Fin}(2) \wedge (\mathrm{Inf}(3) \vee (\mathrm{Fin}(4) \wedge \ldots))))$	

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 $\forall i \geq 0\colon \rho(i) \xrightarrow{w_i} \rho(i+1) \in \delta$. We define the infinite sequence of sets of colors $\mathsf{p}(\rho, w)\colon \omega \to 2^\Gamma$ by $\mathsf{p}(\rho, w)(i) = \mathsf{p}(\rho(i), w_i, \rho(i+1))$. A run ρ on w is *accepting* wrt an acceptance condition α iff $\mathsf{p}(\rho, w) \models \alpha$. 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 ρ over w in $\mathcal{A}$ starting from some state in I such that $\mathsf{p}(\rho, w) \models \mathsf{Acc}$. Popular acceptance conditions can be expressed in this more general framework as given in Table 2. Furthermore, we use the syntactic sugar $\mathcal{A} = (Q, \delta, I, F)$ with $F \subseteq \delta$ being a set of *accepting transitions* to denote a (transition-based) Büchi automaton (BA) that would be defined using the ELA definition above as $(Q, \delta, I, \{0\}, \{\tau \mapsto \emptyset \mid \tau \in \delta \setminus F\} \cup \{\tau \mapsto \{0\} \mid \tau \in F\}, \mathsf{Inf}(0))$.

2.3 Emerson-Lei Elevator and Semi-deterministic Automata

Let $\mathcal{A} = (Q, \delta, I, \Gamma, \mathsf{p}, \mathsf{Acc})$ be an ELA. A non-empty set of states $R \subseteq Q$ is a *strongly connected component* (SCC) of $\mathcal{A}$ iff there exists a path of a non-zero length between every two states of R and R does not admit a superset with this property.[3] Let R be an SCC of $\mathcal{A}$. We use $\delta_R = \delta \cap (R \times \Sigma \times R)$ to denote the set of transitions internal to R and $\mathsf{p}_R = \{\tau \mapsto \mathsf{p}(\tau) \mid \tau \in \delta_R\}$ for the restriction of p to δ_R. We define $\mathcal{A}_R = (R, \delta_R, \emptyset, \Gamma, \mathsf{p}_R, \mathsf{Acc})$ to be the ELA obtained by restricting $\mathcal{A}$ to states from R and simplifying the acceptance condition in the standard way.[4] Observe that $\mathcal{A}_R$ does not have any initial state (this is not a problem since we only talk about its structure). The component R is *inherently weak* iff for some state $q \in R$, either (i) all runs in $\mathcal{A}_R$ from q are accepting or (ii) no run in $\mathcal{A}_R$ from q is accepting (we note that the particular choice of q is irrelevant). R is an *elevator component* if it is one of the following kinds: (i) inherently weak, (ii) deterministic (i.e., $\mathcal{A}_R$ is deterministic), or (iii) generalized co-Büchi (i.e., the acceptance condition of $\mathcal{A}_R$ is generalized co-Büchi). An *Emerson-Lei elevator automaton* (ELEA) is an ELA whose all SCCs are elevator SCCs.

Let $\mathcal{A}[q]$ for $q \in Q$ denote the ELA obtained from $\mathcal{A} = (Q, \delta, q, \Gamma, \mathsf{p}, \mathsf{Acc})$ by removing states and transitions unreachable from q (and simplifying p accordingly). We call $\mathcal{A}$ an *Emerson-Lei semi-deterministic automaton* (ELSA) if, for every state $q \in Q$, one of the following holds: (i) q does not belong to any SCC, (ii) q belongs to an SCC R such that the acceptance condition of $\mathcal{A}_R$ is `false`, or (iii) $\mathcal{A}[q]$ is deterministic, i.e., all runs starting from q are deterministic. Intuitively (and simplifying a bit), runs in ELSAs start in nondeterministic

[3] Note that all our SCCs are non-trivial and maximal.

[4] That is, if a color c does not have an occurrence in $\mathcal{A}_R$, all occurrences of $\mathsf{Inf}(c)$ in Acc are substituted by `false` and all occurrences of $\mathsf{Fin}(c)$ are substituted by `true`; the resulting formula is then simplified using identity and annihilation rules to remove all non-essential occurrences of `true` and `false`.

non-accepting components but after they have seen some color for the first time, they cannot make nondeterministic choices any more—every accepting run needs to stay in some deterministic accepting SCC. If $\mathcal{A}$ is an ELSA, we let $\mathcal{Q}_n \subseteq \mathcal{Q}$ denote the states reachable only from SCCs with the acceptance condition `false` (i.e., *non-accepting* states), and $\mathcal{Q}_d$ the remaining states (i.e., states in *deterministic* accepting SCCs).

3 Complementing an ELSA with a Generalized Rabin Pair

We will introduce our complementation algorithm for ELEAs in several steps, describing the main ideas on the simpler case of semi-deterministic automata (Sections 3–5) and lifting the techniques to elevator automata in Section 6. For semi-deterministic automata, we also start with describing the main techniques using simpler acceptance conditions: generalized Rabin pair (this section) and generalized Streett pair (Section 4). Then, in Section 5 we lift the ideas to ELEAs with an arbitrary acceptance condition.

Let us start with the algorithm for complementing an ELSA $\mathcal{A} = (\mathcal{Q}, \delta, I, \Gamma, \mathsf{p}, \mathsf{Acc})$ with a single generalized Rabin pair acceptance condition $\mathsf{Acc} = \mathsf{Fin}(0) \wedge \mathsf{Inf}(1) \wedge \cdots \wedge \mathsf{Inf}(k)$. Our construction can be viewed as a combination of the Miyano-Hayashi construction [44] and the NCSB algorithm for complementing semi-deterministic BAs [4]. In the following, let $\mathcal{Q}_n$ and $\mathcal{Q}_d$ denote states in the nondeterministic and deterministic parts, respectively.

For ELSAs, every accepting run eventually enters a deterministic SCC and remains there. Hence, it suffices to decide acceptance once the run is in the deterministic SCC. Based on this observation, to complement $\mathsf{Fin}(0)$, it suffices to check if runs in $\mathcal{Q}_d$ visit color 0 infinitely often. Following the Miyano-Hayashi construction [44], we use a pair (R, B) of sets of states: $R \subseteq \mathcal{Q}_d$ ("reach") tracks all current runs in $\mathcal{Q}_d$ (i.e., all possible states where $\mathcal{A}$ can be after reading a given prefix of the input word), and $B \subseteq R$ ("breakpoint") contains those runs that have not yet seen 0. Whenever a run in B sees 0, it is removed from B. When B becomes empty, it is reset to R, thereby restarting the inspection of all runs, including newly entered ones. Thus, B is emptied infinitely often iff all runs in R see 0 infinitely often.

To complement a condition $\mathsf{Inf}(j)$, the NCSB construction [4] uses a triple (C, S, B) to detect whether a run in $\mathcal{Q}_d$ sees j only finitely often, i.e., eventually stops seeing j. Intuitively, (C, B) plays a role similar to (R, B) in the Miyano-Hayashi construction: the set C ("check") tracks all current runs in $\mathcal{Q}_d$, while $B \subseteq C$ ("breakpoint") contains those runs that will eventually be moved out of C to S. In addition, the set $S \subseteq C$ ("safe") collects runs that are guessed to never see j again. At each step, runs in C are nondeterministically assigned either to remain in C or move to S. Since $B \subseteq C$, the runs that move from C to S are also removed from B. When B becomes empty, all inspected runs have been moved to S (or cannot proceed) and we restart the inspection by setting $B = C$. If a run placed in S subsequently sees j, the previous guess was incorrect and that branch will fail. Since the runs in B are deterministic, the number of tracked runs in B does not increase. Moreover, there exists a correct guess for moving all runs in C to S. It is shown in [4] that B becomes empty infinitely often iff all runs in C see j only finitely often.

We now present a construction for complementing an ELSA with a single generalized Rabin pair: $\mathsf{Acc} = \mathsf{Fin}(0) \wedge \mathsf{Inf}(1) \wedge \cdots \wedge \mathsf{Inf}(k)$. Our approach generalizes the Miyano-Hayashi construction and the NCSB construction to this setting. The goal is to ensure that every run over a word w violates at least one of these conditions. We use a set N to track runs in the nondeterministic part $\mathcal{Q}_n$. For runs in $\mathcal{Q}_d$, instead of using a pair (R, B) for Fin and separate triples (C_j, S_j, B_j) for each $\mathsf{Inf}(j)$-condition, we share structures: a single set C tracks runs for all Inf-conditions, and a single breakpoint set B is used globally.

Accordingly, a macrostate in the complement automaton is a tuple $(N, R, C, S_1, \ldots, S_k, B)$. Intuitively, runs in $\mathcal{Q}_n$ are stored in N. Upon entering $\mathcal{Q}_d$, a run is nondeterministically placed in R (to violate $\mathsf{Fin}(\mathbf{0})$) or in C (to violate some Inf-condition). All runs under inspection within $R \cup C$ are initially placed in B. For a run in $C \cap B$ that is guessed to violate $\mathsf{Inf}(\mathbf{j})$, it is moved to S_j and removed from B. For runs in $R \cap B$, we remove runs visiting $\mathbf{0}$ from B. When B becomes empty, all runs in $R \cup C$ have been inspected and B is reset to $R \cup C$ again. Thus, if B is emptied infinitely often and no branch is rejected due to a wrong guess, every run over w violates at least one condition of the generalized Rabin pair.

Formally, we define the complement BA $\mathcal{A}^\sharp = (\mathcal{Q}^\sharp, \delta^\sharp, I^\sharp, \mathsf{Acc}^\sharp)$ where the following holds:

- The set $\mathcal{Q}^\sharp$ of macrostates consists of tuples $(N, R, C, S_1, \ldots, S_k, B) \in 2^{\mathcal{Q}_n} \times (2^{\mathcal{Q}_d})^{k+3}$ with the restrictions that (i) $R \cap (C \cup S_1 \cup \cdots \cup S_k) = \emptyset$ (no run is tracked by more than one procedure), (ii) $C \cap S_i = \emptyset$ for all $1 \leq i \leq k$ (safe runs do not need to be checked any more), (iii) $S_i \cap S_j = \emptyset$ for all $1 \leq i < j \leq k$ (safe runs need to break only one condition), and (iv) $B \subseteq R \cup C$ (breakpoint tracks only relevant runs).
- The set $I^\sharp \subseteq \mathcal{Q}^\sharp$ of initial macrostates is defined by $I^\sharp = \{(I \cap \mathcal{Q}_n, R, C, \emptyset, \ldots, \emptyset, B) \mid R \cap C = \emptyset, B = R \cup C = I \cap \mathcal{Q}_d\}$. That is, initially, all runs starting in $\mathcal{Q}_d$ are nondeterministically partitioned into R and C, and all of them are placed in the set B.
- For a macrostate $(N, R, C, S_1, \ldots, S_k, B) \in \mathcal{Q}^\sharp$ and an input symbol a, we have that $(N', R', C', S'_1, \ldots, S'_k, B') \in \delta^\sharp((N, R, C, S_1, \ldots, S_k, B), a)$ if the following holds:
  - $N' = \delta(N, a) \cap \mathcal{Q}_n$ (The set N tracks runs in the nondeterministic component.),
  - $\delta(N, a) \cap \mathcal{Q}_d \subseteq R' \cup C' \cup S'_1 \cup \cdots \cup S'_k$ (Every run entering the deterministic component from N must be tracked by either the Fin-violating or the Inf-violating procedure.),
  - $\delta(S_j, a) \subseteq S'_j$ and $S'_j \setminus \delta(S_j, a) \subseteq \delta(C, a)$ for each $\mathbf{1} \leq \mathbf{j} \leq \mathbf{k}$ (Each set S_j stores the runs that are guessed to avoid color $\mathbf{j}$ from now on. Accordingly, $\mathbf{j}$-safe runs stay $\mathbf{j}$-safe and, additionally, runs from C can become $\mathbf{j}$-safe.)
  - $\mathbf{j} \notin \mathsf{p}(q, a, \delta(q, a))$ for each $\mathbf{1} \leq \mathbf{j} \leq \mathbf{k}$ and $q \in S_j$ (Every run already contained in S_j must avoid color $\mathbf{j}$ on the current transition.),
  - $B' = R' \cup C'$ if $B = \emptyset$, otherwise $B' \subseteq \delta(B, a)$ (Restart inspection for all runs in $R \cup C$ when B is empty.); moreover, each run leaving B must be correctly discharged:
    - if a run from $B \cap R$ is removed from B', then it must have seen color $\mathbf{0}$, i.e., for all $q \in B \cap R$ such that $\delta(q, a) \notin B'$, we have $\mathbf{0} \in \mathsf{p}(q, a, \delta(q, a))$;
    - if a run from $B \cap C$ is removed from B', then its successor must be placed into some safe set S'_j, i.e., for all $q' \in \delta(B \cap C, a) \setminus B'$, we have $q' \in S'_j$ for some $\mathbf{1} \leq \mathbf{j} \leq \mathbf{k}$.
  - The set R tracks runs that are guessed to violate $\mathsf{Fin}(\mathbf{0})$, i.e., runs that see the color $\mathbf{0}$ infinitely often. Hence, R is closed under deterministic successors and may additionally receive runs entering $\mathcal{Q}_d$ from N: $\delta(R, a) \subseteq R', R' \setminus \delta(R, a) \subseteq \delta(N, a) \cap \mathcal{Q}_d$.
  - The set C tracks runs that will violate one of the Inf-conditions. Such runs evolve deterministically and are moved to some safe set, and C may additionally receive runs entering $\mathcal{Q}_d$ from N: $\delta(C, a) \setminus (S'_1 \cup \ldots \cup S'_k) \subseteq C'$ and $C' \setminus \delta(C, a) \subseteq \delta(N, a) \cap \mathcal{Q}_d$.
- The Büchi acceptance condition requires the breakpoint set to be reset infinitely often. Thus, $\mathsf{Acc}^\sharp = \{(\cdot, \ldots, \cdot, B) \xrightarrow{a} (\cdot, \ldots, \cdot, \cdot) \in \delta^\sharp \mid B = \emptyset\}$.[5]

[5] We remind the reader of the abuse of notation where the acceptance condition for BAs is a set of accepting transitions (Section 2.2).

Note that the construction does not allow merging of runs tracked by different procedures, e.g., in the case the successors of a state $r \in R$ and a state in $c \in C$ are the same state q, the construction cannot continue. This is not a problem since, due to nondeterminism, there will also be a run where the predecessor of c that was nondeterministically put to C is put to R instead, and also a run where the predecessor of r that was nondeterministically put to R is put to C instead (and similarly for merging runs between different S-sets).

▶ **Theorem 1.** $\mathcal{L}(\mathcal{A}^\sharp) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$

Proof (sketch). Every word $w \in \mathcal{L}(\mathcal{A}^\sharp)$ is not accepted by $\mathcal{A}$. Indeed, along any accepting macrorun of $\mathcal{A}^\sharp$, the set B becomes empty infinitely often, which implies that all runs in $R \cup C$ violate at least one condition of the generalized Rabin pair.

Conversely, let $w \notin \mathcal{L}(\mathcal{A})$. Then every run of $\mathcal{A}$ over w violates at least one condition of the generalized Rabin pair. Hence, there exists a correct nondeterministic guess that assigns each run to the set corresponding to the condition it violates, at the earliest point. Under this guess, the macrorun of $\mathcal{A}^\sharp$ over w does not terminate. Moreover, every run in B is eventually removed—either by visiting 0 or by being moved to some S_i—so that B becomes empty infinitely often. Therefore, the macrorun is accepting in $\mathcal{A}^\sharp$, i.e., $w \in \mathcal{L}(\mathcal{A}^\sharp)$. ◀

Let us now consider the size of $\mathcal{A}^\sharp$. We can deduce the bound on the number of states of $\mathcal{A}^\sharp$ by checking, for each state $q \in \mathcal{Q}$, in which internal sets of a macrostate $\mathcal{M} = (N, R, C, S_1, \ldots, S_k, B)$ it can be and give each such a choice a number $f(q)$ as follows:

- If q is not present in $\mathcal{M}$ at all, we assign $f(q) = 1$.
- If $q \in \mathcal{Q}_n \cap N$ or $q \in \mathcal{Q}_d \cap R \cap B$, we assign $f(q) = 2$ (i.e., we can merge N and $R \cap B$ into one set W and split it into N and $R \cap B$ anytime as $N = W \cap \mathcal{Q}_n$ and $R \cap B = W \cap \mathcal{Q}_d$).
- If $q \in R \setminus B$, we let $f(q) = 3$, if $q \in C \cap B$, we let $f(q) = 4$, if $q \in C \setminus B$, we let $f(q) = 5$.
- If $q \in S_j$ for $1 \leq$ j $\leq$ k, we assign $f(q) = 5 + j$.

The total number of functions $f \colon \mathcal{Q} \to \{1, \ldots, 5 + k\}$ is $(5 + k)^n$ where $n = |\mathcal{Q}|$.

▶ **Theorem 2.** *The size of $\mathcal{A}^\sharp$ is bounded by $(5 + k)^n$.*

We contrast the obtained state complexity $(5 + k)^n$ to the best upper bound we are aware of for unrestricted generalized Rabin pair ELAs, which is $\mathcal{O}(nk^n(0.76n)^n)$ [24]. Exploiting the automaton structure can indeed have a profound effect on the hardness of complementation.

Note that if the ELSA has the generalized Büchi condition (i.e., it does not contain Fin), we can remove the R part of the macrostate. Then we can omit considering $R \cap B$ and $R \setminus B$ in the complexity analysis and we obtain the size of the output bounded by $(3 + k)^n$.

3.1 Optimization using Safe Run Fall-Through

One practical issue of the procedure given above is a high degree of nondeterminism, since at every step, (i) newly incoming runs to $\mathcal{Q}_d$ need to be moved nondeterministically to either the R-part or the $(C, S_1, \ldots, S_k)$-part of the macrostate and, at the same time, (ii) runs in B need to either stay in B or nondeterministically move to some S_i (for every such run, all S_i's need to be considered) causing a high out-degree of the constructed macrostates. We give a modification of the procedure that deals with point (ii) as follows:

1. We either move all runs from B to S_1 or let them all stay in B.
2. If a run in any S_j sees j, we move it to S_{j+1} (unless $j = k$ in which case we do not generate the successor).
3. If a run in S_i is merging with a run in S_j for $j > i$, we keep the run in S_j.

Intuitively, we nondeterministically decide a point from which some Inf condition is violated by a run and then try, one by one, to find which Inf condition it is. We give more details about the construction in the inductive procedure in Section 5.3.

4 Complementing an ELSA with a Generalized Streett Pair

Next, we give our algorithm for complementing an ELSA $\mathcal{A} = (Q, \delta, I, \Gamma, \mathsf{p}, \mathsf{Acc})$ with a single generalized Streett pair condition $\mathsf{Acc} = \mathsf{Inf}(0) \vee \mathsf{Fin}(1) \vee \cdots \vee \mathsf{Fin}(k)$, which can be seen as a dual of the generalized Rabin pair condition described in the previous section.

For a generalized Streett pair, it suffices to ensure that every run violates *all* conditions in Acc. As before, our algorithm builds on top of the Miyano-Hayashi [44] (to handle the Fin conditions) and NCSB [4] (to handle the Inf conditions). A macrostate in the complement automaton has the structure (N, CR, S, B, z), where $N \subseteq Q_n$ tracks runs in the nondeterministic part and $CR, S, B \subseteq Q_d$ track runs in the deterministic parts; z is used as a scheduling counter for checking that all $k+1$ acceptance conditions are invalidated in a round robin manner. In essence, the algorithm runs one instance of NCSB and k instances of the Miyano-Hayashi algorithm in sequence, reusing the sets in the macrostate between them. In the macrostate, CR is used to hold the content of the C-set of NCSB (when $z = 0$) or the R_j-set of the Miyano-Hayashi algorithm for invalidating $\mathsf{Fin}(j)$ (when $z = j > 0$).

Initially, $B = CR$ and $z = 0$, indicating that the condition $\mathsf{Inf}(0)$ is currently under inspection. The breakpoint B contains exactly those runs that are being checked for violating the currently scheduled condition. When $z = 0$, for violating the $\mathsf{Inf}(0)$ condition, we need to nondeterministically move all runs from B to S eventually (runs in S are not allowed to see 0 any more) before switching to the next condition. For $z = j \geq 1$, we are inspecting the condition $\mathsf{Fin}(j)$. In this phase, a run remains in B until it sees color j, at which point it is removed from B, witnessing a violation of the Fin-condition. When B becomes empty, we increment z and switch to the next condition. At every switch, we re-sample $B = CR$.

This cyclic inspection ensures that every run is repeatedly checked against each condition. If B becomes empty infinitely often, then for every run and for every condition in Acc, there is a point at which the run is verified to violate that condition. Hence, all runs violate all conditions, and thus the generalized Streett pair condition.

Formally, we define the complement BA $\mathcal{A}^\sharp = (Q^\sharp, \delta^\sharp, I^\sharp, \mathsf{Acc}^\sharp)$ as follows.

- The set $Q^\sharp$ of macrostates contains tuples $(N, CR, S, B, z) \in 2^{Q_n} \times (2^{Q_d})^3 \times \{0, \ldots, k\}$ with $S \subseteq CR$ and $B \subseteq CR$.
- $I^\sharp = \{(I \cap Q_n, I \cap Q_d, \emptyset, I \cap Q_d, 0)\}$ contains a single initial macrostate (we start with checking the Inf condition).
- For a macrostate (N, CR, S, B, z) and $a \in \Sigma$, $(N', CR', S', B', z') \in \delta^\sharp((N, CR, S, B, z), a)$ if the following holds:
  - $N' = \delta(N, a) \cap Q_n$ (N tracks the runs in the non-deterministic component.),
  - $CR' = \delta(N \cup CR, a) \cap Q_d$ (CR tracks all runs in Q_d and those that came from Q_n.),
  - $0 \notin \mathsf{p}(q, a, \delta(q, a))$ for each $q \in S$ (Safe states cannot see 0 any more.),
  - if $B \neq \emptyset$ (We keep tracking the current condition.) then $z' = z$ and
    - if $z = 0$ (we are tracking $\mathsf{Inf}(0)$):
      - $S' = \delta(S, a)$ and $B' = \delta(B, a) \setminus S'$ (We wait with moving B to S.) or
      - $S' = \delta(S \cup B, a)$ and $B' = \emptyset$ (We move runs in B to S.),
    - if $z = j > 0$ (we are tracking $\mathsf{Fin}(j)$), then $S' = \delta(S, a)$ and $B' = \delta(\{q \in B \mid j \notin \mathsf{p}(q, a, \delta(q, a))\}, a)$ (We remove from B runs that see j.),

- if $B = \emptyset$ then $S' = \delta(S, a)$, $z' = (z + 1) \bmod (k + 1)$, and $B' = CR' \setminus S'$ (if $z' = 0$) or $B' = S'$ (if $z > 0$).

- $\mathsf{Acc}^\sharp = \{(\cdot, \ldots, \cdot, B) \xrightarrow{a} (\cdot, \ldots, \cdot, \cdot) \in \delta^\sharp \mid B = \emptyset\}$ (We need to reset B infinitely often).

Note that the construction has (unlike many other complementation constructions for omega automata) a quite high degree of determinism: each macrostate has for every symbol at most two successors (when $z = 0$) or at most one successor (when $z > 0$). We also note that we are sampling the breakpoint B with CR only when entering the $\mathsf{Inf}(0)$ part of the algorithm, when moving to Fin conditions, we sample B with S. We could also sample B with CR, but this would make the complexity of the procedure worse.

▶ **Theorem 3.** $\mathcal{L}(\mathcal{A}^\sharp) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$

Proof (sketch). In a macrorun over a word $w \in \mathcal{L}(\mathcal{A}^\sharp)$, the set B becomes empty infinitely often. This means that the counter z cycles through all conditions. Hence, every run is repeatedly inspected and shown to violate each condition in Acc. Thus, all runs in $\mathcal{A}$ violate all conditions, and so $w \notin \mathcal{L}(\mathcal{A})$.

Conversely, let $w \notin \mathcal{L}(\mathcal{A})$. Then every run over w violates all conditions in Acc. There exists a correct nondeterministic guess such that, in each phase, every run in B is eventually removed—either by being moved to S or by witnessing the corresponding Fin-condition—so that B becomes empty infinitely often. Since all conditions are inspected cyclically, each violation is eventually verified. Hence, the macrorun is accepting, and $w \in \mathcal{L}(\mathcal{A}^\sharp)$. ◀

The following theorem gives bounds on the size of $\mathcal{A}^\sharp$ (obtained in a similar manner as the bounds in Theorem 2 and proved in [2]).

▶ **Theorem 4.** *The size of $\mathcal{A}^\sharp$ is bounded by $(k + 1) \cdot 4^n$.*

Note that if there is no Inf condition, i.e., the acceptance condition is generalized co-Büchi, then we do not need the S part of the macrostate; $\mathcal{A}^\sharp$ then has at most $k \cdot 3^n$ states.

5 Inductive Procedure for Complementing Deterministic Components

This section generalizes the constructions of Sections 3 and 4 to arbitrary acceptance conditions. Let $\mathcal{A} = (\mathcal{Q}_n \cup \mathcal{Q}_d, \delta, I, \Gamma, \mathsf{p}, \mathsf{Acc})$ be a semi-deterministic ELA. To complement $\mathcal{A}$, we require that every run entering $\mathcal{Q}_d$ satisfies $\psi = \neg\mathsf{Acc}$. As before, we use a set N to track the runs in $\mathcal{Q}_n$ and a set C for runs entering $\mathcal{Q}_d$. Unlike generalized Rabin and Streett pairs, ψ may contain both conjunctions and disjunctions, which complicates the run tracking. To address this, we distinguish two modes: *sampling* and *checking*. In the sampling mode, we nondeterministically guess, for each run in $\mathcal{Q}_d$, which disjuncts of each $\vee$-subformula in ψ it will satisfy. Accordingly, each subformula of ψ is associated with a set of runs expected to satisfy it. In the checking mode, we verify these guesses. For a condition $\mathrm{Inf}(k)$, we use a pair (H, B), where H contains the runs guessed to satisfy the condition and $B \subseteq H$ is a breakpoint set used to verify that these runs visit k infinitely often. Concretely, a run is removed from B once it visits k, and whenever B becomes empty, it is reset to H. For a condition $\mathrm{Fin}(\ell)$, it suffices to ensure that runs in S do not visit ℓ.

To implement this, we employ *tree-macrostates* that mirror the syntactic structure of ψ. Formally, a tree-macrostate M_ψ is a tree constructed by $\mathsf{Tr}(r, e, m_1, m_2)$, where the root label r is either $\wedge$, $\vee$, or a basic condition of the form $\mathrm{Inf}(k)$ or $\mathrm{Fin}(\ell)$, e is a node label, and m_1, m_2 are the tree-macrostates corresponding to the subformulas (or the empty tree NIL if absent). For internal nodes, e is undefined (denoted by $-$). For leaf nodes, e stores sets

Algorithm 1 Sampling successor runs $\texttt{Succ}(M_\psi, a, C, f)$

if $\psi = Fin(\ell)$ *with* $e = S_{Fin(\ell)}$ **then**
  $L = \emptyset$ if $\exists t = s \xrightarrow{a} s' \in \delta$ for $s \in S \cup C$ s.t $\ell \in \mathsf{p}(t)$ and otherwise $L = \{\delta(S \cup C, a)\}$;
  $M = \{\mathsf{Tr}(\mathrm{Fin}(\ell), L_{\mathrm{Fin}(\ell)}, \mathsf{NIL}, \mathsf{NIL})\}$;
else if $\psi = Inf(k)$ *with* $e = (H, B)_{Inf(k)}$ **then**
  $L = (\delta(H \cup C, a), \delta(H \cup C, a))$ if $f = \top$ otherwise $L = (\delta(H, a), \delta(B, a) \setminus \{t \in \delta \mid k \in \mathsf{p}(t)\})$;
  $M = \{\mathsf{Tr}(\mathrm{Inf}(k), L_{\mathrm{Inf}(k)}, \mathsf{NIL}, \mathsf{NIL})\}$;
else if $\psi = \varphi_1 \wedge \varphi_2$ **then**
  $M = \{\mathsf{Tr}(\wedge, -, m_1, m_2) \mid m_1 \in \texttt{Succ}(M_{\varphi_1}, a, C, f), m_2 \in \texttt{Succ}(M_{\varphi_2}, a, C, f)\}$;
else if $\psi = \varphi_1 \vee \varphi_2$ **then**
  $M = \{\mathsf{Tr}(\vee, -, m_1, m_2) \mid m_1 \in \texttt{Succ}(M_{\varphi_1}, a, C_1, f), m_2 \in \texttt{Succ}(M_{\varphi_2}, a, C_2, f), C = C_1 \cup C_2\}$; `// Nondet. split of` C
return M

of runs: $e = (H, B)_{\mathrm{Inf}(k)} \subseteq \mathcal{Q}_d^2$ for Inf-conditions and $e = S_{\mathrm{Fin}(\ell)} \subseteq \mathcal{Q}_d$ for Fin-conditions. We write a leaf node simply as its label e.

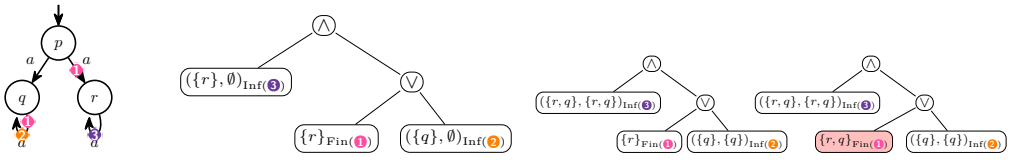
Intuitively, in the sampling mode, we ensure that each run satisfies ψ by propagating it through the tree. At a leaf, a run is assigned to H (for Inf-conditions) or to S (for Fin-conditions). If $\psi = \varphi_1 \wedge \varphi_2$, the run must satisfy both subformulas and is therefore assigned to both subtrees M_{φ_1} and M_{φ_2}. If $\psi = \varphi_1 \vee \varphi_2$, the run must satisfy at least one subformula, and is thus nondeterministically assigned to either M_{φ_1} or M_{φ_2}.

In the checking mode, the construction proceeds as in the previous sections. The key idea is that breakpoint B is emptied infinitely often iff all runs are correctly guessed and satisfy ψ.

Formally, let $\mathcal{A} = (\mathcal{Q}_n \cup \mathcal{Q}_d, \delta, I, \Gamma, \mathsf{p}, \mathsf{Acc})$ be a semi-deterministic ELA whose negated acceptance condition is ψ. We define an NBA $\mathcal{A}^\sharp = (\mathcal{Q}^\sharp, \delta^\sharp, I^\sharp, \mathsf{Acc}^\sharp)$ as follows:

- The set of macrostates $\mathcal{Q}^\sharp$ contains macrostates of the form $(N, C, \mathcal{M}_\psi)$ where $N \subseteq \mathcal{Q}_n$, $C \subseteq \mathcal{Q}_d$, and $\mathcal{M}_\psi$ is a tree-macrostate.
- The initial set of macrostates is $I^\sharp = \{(I \cap \mathcal{Q}_n, I \cap \mathcal{Q}_d, \mathcal{M}_\psi^0)\}$, where $\mathcal{M}_\psi^0$ has the same syntactic structure as ψ and in particular, $M^0_{\mathrm{Fin}(\ell)} = \mathsf{Tr}(\mathrm{Fin}(\ell), \emptyset_{\mathrm{Fin}(\ell)}, \mathsf{NIL}, \mathsf{NIL})$ for each $\mathrm{Fin}(\ell)$ and $M^0_{\mathrm{Inf}(k)} = \mathsf{Tr}(\mathrm{Inf}(k), (\emptyset, \emptyset)_{\mathrm{Inf}(k)}, \mathsf{NIL}, \mathsf{NIL})$ for each $\mathrm{Inf}(k)$. Intuitively, all deterministic runs are initially placed in the C-set, and the automaton has not yet entered the sampling-and-checking mode.
- The definition of $\delta^\sharp$ and $\mathsf{Acc}^\sharp$ will be defined in detail below.

To construct $\delta^\sharp$ and $\mathsf{Acc}^\sharp$, we first introduce the notions of successors and satisfiability of a given tree-macrostate. We define the successor construction for tree-macrostates (see Algorithm 1). Let $\mathcal{M}_\varphi$ be a tree-macrostate for a subtree φ of ψ, $a \in \Sigma$, $C \subseteq \mathcal{Q}_d$, and $f \in \{\bot, \top\}$ a mode flag, where $\top$ denotes sampling mode. The set of successor tree-macrostates is defined inductively as specified in Algorithm 1. Observe that, for Fin-leaves,



(a) Automaton **(b)** Some tree-macrostate $\mathcal{M}_\psi$ **(c)** Tree-macrostates of $\texttt{Succ}(\mathcal{M}_\psi, a, \{q\}, \top)$

Figure 1 Succ computation with the negated condition $\psi = \mathrm{Inf}(3) \wedge (\mathrm{Fin}(1) \vee \mathrm{Inf}(2))$.

Algorithm 2 Successor computation $\delta^\sharp((N, C, \mathcal{M}_\psi), a)$

```
1 N' = δ(N, a) ∩ Q_n;
2 if C = ∅ ∧ IsSat(M_ψ) then
3 |   R := {(N', δ(N, a) ∩ Q_d, M'_ψ) | M'_ψ ∈ Succ(M_ψ, a, ∅, ⊥)};
4 else
5 |   C' := δ(C, a);
6 |   R := {(N', C', M'_ψ) | M'_ψ ∈ Succ(M_ψ, a, ∅, ⊥)};
7 |   R := R ∪ {(N', ∅, M'_ψ) | M'_ψ ∈ Succ(M_ψ, a, C, ⊤)};
8 return R
```

the presence of a ℓ-labelled transition eliminates all successors. Moreover, the flag f is only relevant for Inf-leaves: if $f = \top$, the breakpoint set B is resampled; if $f = \bot$, it continues to track runs that have not visited k.

▶ **Example 5.** Consider the automaton in Figure 1(a) with negated acceptance condition $\psi = \text{Inf}(3) \wedge (\text{Fin}(1) \vee \text{Inf}(2))$. Consider the tree-macrostate $\mathcal{M}_\psi$ in Figure 1(b), with $C = \{q\}$ and $f = \top$. At the $\wedge$-node, both C and f are forwarded to the two children. For the Inf(3)-leaf, we have $H = \{r\}$ and $B = \emptyset$. Its successors are trees $\mathsf{Tr}(\text{Inf}(3), (\{r, q\}, \{r, q\})_{\text{Inf}(3)}, \mathsf{NIL}, \mathsf{NIL})$, since $\delta(H \cup C, a) = \delta(\{r, q\}, a) = \{r, q\}$ and $f = \top$ (hence B is resampled). At the $\vee$-node, C is nondeterministically split into $C_1 \cup C_2$. If $C_1 = \emptyset$ and $C_2 = \{q\}$, then at the Fin(1)-leaf, we have $\delta(\{r\}, a) = \{r\}$, as no state in $S \cup C_1 = \{r\}$ admits a 1-colored transition. The Inf(2)-leaf resamples its breakpoint (since $f = \top$) and returns trees $\mathsf{Tr}(\text{Inf}(2), (\{q\}, \{q\})_{\text{Inf}(2)}, \mathsf{NIL}, \mathsf{NIL})$, because $\delta(H \cup C_2, a) = \delta(\{q\}, a) = \{q\}$.

If $C_1 = \{q\}$ and $C_2 = \emptyset$, then at the Fin(1)-leaf it obtains $S \cup C_1 = \{r, q\}$. Since $q \xrightarrow{a} q$ is 1-colored, it returns $\emptyset$, and this branch yields no successor.

We say that a tree-macrostate is *satisfied* if all its Inf-leaves have empty breakpoints. Formally, we define IsSat inductively over subtrees of ψ by $\texttt{IsSat}(\mathcal{M}_{\varphi_1 \star \varphi_2}) = \texttt{IsSat}(\mathcal{M}_{\varphi_1}) \wedge \texttt{IsSat}(\mathcal{M}_{\varphi_2})$ with $\star \in \{\wedge, \vee\}$, $\texttt{IsSat}(M_{\text{Fin}(\ell)}) = \top$, and $\texttt{IsSat}(M_{\text{Inf}(k)}) \iff B = \emptyset$ where B is the breakpoint set of $M_{\text{Inf}(k)}$.

Now we have all the ingredients to construct $\delta^\sharp$ and $\mathsf{Acc}^\sharp$. Intuitively, $\mathcal{A}^\sharp$ operates in two alternating phases that repeat infinitely often: First, in the *sampling* mode, the construction samples runs entering $\mathcal{Q}_d$ by maintaining a set C. This mode must repeat infinitely often to ensure that every run is eventually tracked and checked against ψ. Second, in the *checking* mode, the sampled runs are verified to ensure that all of them satisfy ψ. Formally, $\delta^\sharp$ is defined by Algorithm 2. Its behavior depends on the value of C. When $C = \emptyset$ and $\texttt{IsSat}(\mathcal{M}_\psi)$ (i.e., checking mode has been successfully completed), an accepting mark is emitted and C is replenished with all states from $N \cap \mathcal{Q}_d$ to begin a fresh sampling cycle. Otherwise, the automaton is in the checking phase, meaning that some breakpoint set is not yet empty. In this case, we proceed nondeterministically: either the successors are computed without resampling, i.e., $f = \bot$, thereby continuing the verification of the currently tracked runs (see Line 6), or we choose to start a new sampling cycle by initiating resampling (see Line 7), which nondeterministically assigns all current runs in C into leaf nodes (see Line 7). Finally, the accepting condition of $\mathcal{A}^\sharp$ is $\mathsf{Acc}^\sharp = \{(N, C, \mathcal{M}_\psi) \xrightarrow{a} (N', C', \mathcal{M}'_\psi) \in \delta^\sharp \mid C = \emptyset, \texttt{IsSat}(\mathcal{M}_\psi)\}$. Let ψ^{DNF} be the disjunctive normal form of ψ. Then $\mathsf{Acc}^\sharp$ ensures that each run of the input word either never enters the deterministic component or, after entering, satisfies some conjunct of ψ^{DNF} by visiting infinitely colours of all Inf-leaves while avoiding colours of all Fin-leaves. It follows that $\mathcal{L}(\mathcal{A}^\sharp) \subseteq \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$. Conversely, for every word not accepted by

$\mathcal{A}$, there exists a correct nondeterministic choice for runs expected to satisfy a disjunct in ψ^{DNF}. Along such a macrorun, the breakpoint sets associated with all Inf-leaves are cleared infinitely often, while no Fin-conditions are violated. Hence, $\Sigma^\omega \setminus \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}^\sharp)$.

Therefore, we obtain our main result:

▶ **Theorem 6.** $\mathcal{L}(\mathcal{A}^\sharp) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

5.1 Tree-Macrostate Normalization

Two runs may merge along a word. Consequently, after computing successors over a letter, a successor tree-macrostate may contain redundant run tracking, in particular when the same state appears in multiple subtrees of a $\vee$-node and is checked independently. To avoid having multiple representations of semantically the same state, we keep only the rightmost occurrence and enforce disjointness among subtrees of each $\vee$-node. Intuitively, once two runs merge, one can be discarded, as its finite prefix does not affect the satisfaction of any acceptance condition. This normalization preserves correctness. See [2] for details.

5.2 Shared Breakpoint Optimization

In the base construction, each Inf-leaf maintains its own breakpoint, leading to a state-space factor of $|2^{\mathcal{Q}_d}|^k$ for a subtree with k Inf-leaves. We can apply a standard round-robin technique to track one Inf-leaf at a time using a shared breakpoint, reducing the factor to $k \cdot |2^{\mathcal{Q}_d}|$. Concretely, the construction cycles through the Inf-leaves, verifying each leaf in turn before proceeding to the next, and restarting the cycle thereafter. This optimization preserves correctness while avoiding independent breakpoint tracking.

To implement this, we attach a context $ctx = (h, j, \mathbb{L}, B_{sh})$ to selected nodes, which maintain the shared breakpoint. The phase $h \in \{W, R, P\}$ indicates whether the cycle is waiting for the next sampling, resampling the shared breakpoint for the current leaf, or processing the current breakpoint. The sequence $\mathbb{L} = \langle u_1, \ldots, u_k \rangle$ orders the Inf-leaves, j denotes the currently tracked leaf, and $B_{sh} \subseteq \mathcal{Q}_d$ is the shared breakpoint. Individual breakpoints are removed under this optimization. A tree-macrostate may contain multiple such contexts; details are given in [2].

5.3 OR-FIN Optimization

In the base construction, `Succ` for a $\vee$-node nondeterministically partitions the states in C across its children, yielding $2^{|C|}$ successors. The *OR-FIN optimization* avoids this blowup when the left child φ_1 is an `IsORF` subtree, i.e., a $\vee$-composition of Fin-leaves. The key idea is to propagate violating states sequentially: a Fin-leaf Fin filters violating states/runs and passes them to the next Fin-leaf. These states/runs are eventually either absorbed by an Inf-leaf or cause rejection if they reach the root. This is correct because a run needs to satisfy at least one branch of the $\vee$-subtree; thus, it suffices to check the Fin-leaves one after another. If a run violates all of them, it is correctly rejected; otherwise, it is accepted as soon as one condition is met. This eliminates the need for exponential branching while preserving correctness. Details of the construction are provided in [2].

5.4 Complexity Analysis

Let us now analyse the state complexity of our construction. Consider an ELSA $\mathcal{A}$ with n states $\mathcal{Q} = \mathcal{Q}_n \cup \mathcal{Q}_d$ and an Emerson-Lei condition Acc, together with its complement $\mathcal{A}^\sharp$

constructed by the inductive procedure above. Let $\psi = \neg\mathsf{Acc}$ with x being the number of Fin atoms, and y being the number of Inf atoms. We define d as the maximum number of Inf-leaves in a $\wedge$-only subtree of ψ, and m as the number of $\wedge$-only maximal subtrees (i.e., $\wedge$-only subtrees not contained inside larger $\wedge$-only subtrees) of ψ with an Inf-leaf. Moreover, $m \leq y$ refers to the number of contexts generated by the shared breakpoint optimization (Section 5.2), and d is the maximum number of leaves that contexts may schedule. Observe that x, y, m, and d are upper-bounded by the number of atoms of Acc.

The states of $\mathcal{A}^\sharp$ hold a set $N \subseteq \mathcal{Q}_n$, a set $C \subseteq \mathcal{Q}_d$, and a tree-macrostate $\mathcal{M}_\psi$. Without optimization, $\mathcal{M}_\psi$ holds x sets $S \subseteq \mathcal{Q}_d$ (one per Fin-leaf), and y pairs of sets $(S, B) \subseteq \mathcal{Q}_d^2$ (one per Inf-leaf). With the shared breakpoint optimization, $\mathcal{M}_\psi$ holds $x + y$ sets $S \subseteq \mathcal{Q}_d$ (one per leaf), and m contexts. Each context keeps a phase status $p \in \{W, R, P\}$, a counter $j \in \{1, \ldots, d\}$, a set of at most d identifiers among Inf-leaves $\mathbb{L}$, and a breakpoint $B_{sh} \subseteq \mathcal{Q}_d$. Note that $\mathbb{L}$ is fully determined by the formula, and thus it is independent of the size of $\mathcal{A}^\sharp$.

We establish an upper bound on the number of states of $\mathcal{A}^\sharp$ by determining, for each state of $\mathcal{A}$, the internal subsets of a macrostate in which it may appear. This analysis assumes the uses of the shared breakpoint optimization. Given a state $q \in \mathcal{Q}$ of $\mathcal{A}$, the possibilities are that (I) q belongs in N and no other subsets, (II) q belongs in N and some other subsets: (II.i) q belongs in C or not, (II.ii) for each Fin-leaf holding the set $S \subseteq \mathcal{Q}_d$, q belongs in S or not, (II.iii) for each Inf-leaf holding the pair of sets $(S, B) \subseteq \mathcal{Q}_d^2$, q belongs in S or not, and (II.iv) for each node holding the shared context B_{sh}, q belongs in S or not. Item (I) identifies a single isolated case, all other cases are captured by the subitems of (II). Item (II.i) describes two subcases, i.e., possibilities that stacks with the other subitems of (II). Item (II.ii) considers all subsets of Fin-leaves, thus describing 2^x subcases. Item (II.iii) is analogous for Inf-leaves, thus describing 2^y subcases. Finally, Item (II.iv) is analogous for roots of $\wedge$-only maximal subtrees of ψ with a Inf-leaf, thus describing 2^m subcases. Observe that case where q belongs in no subsets (including N) is captured by (II). Since this subset appearance is independent for all states of $\mathcal{A}$, we have $(1 + 2 \cdot 2^y \cdot 2^x \cdot 2^m)^n$ cases so far. To get the full picture, it remains to include the data carried by each context, namely, the phase and the counter. We take this information into account through copies of the previous reasoning. Hence, $\mathcal{A}^\sharp$ admits at most

$$3 \cdot \max\{d, 1\} \cdot (1 + 2 \cdot 2^y \cdot 2^x \cdot 2^m)^n$$

macrostates. Indeed, there are three phase statuses and a counter that takes at most d distinct values. We derive upper bounds for Rabin, generalized Rabin, and Parity acceptance conditions from this general analysis. We can further reduce this bound when a certain property holds. First, we assume that sets appearing in Fin-leaves are disjoint. This situation is fulfilled when all Fin-leaves of ψ are in a subtree without $\wedge$-nodes. So, this assumption holds true for ψ when the acceptance condition of $\mathcal{A}$ is generalized co-Büchi, Streett pair, generalized Streett pair, Streett, and generalized Streett, for instance. We get $3 \cdot \max\{d, 1\} \cdot (1 + 2 \cdot 2^y \cdot (x + 1) \cdot 2^m)^n$ because Item (II.ii) now captures the subcases where q appears in one of the Fin-leaves, or none of them. Second, we assume that sets appearing in Fin-leaves are disjoint and additionally the whole negated acceptance condition ψ has no conjunction. Thus, the shared breakpoint optimization has no effect here and breakpoint sets appear on each Inf-leaf. In particular, $0 \leq d \leq 1$ (0 if ψ has no Inf-leaf, 1 otherwise) and $m = y$. This assumption holds true for ψ when the acceptance condition of $\mathcal{A}$ is Büchi, generalized Büchi, co-Büchi, Rabin pair, and generalized Rabin pair. We get $3 \cdot \max\{d, 1\} \cdot (1 + 2 \cdot 2^y \cdot (x + m + 1))^n$ because Items (II.ii) and (II.iv) now capture together the subcases where q appears in one of the leaves (since $m = y$), or none of them.

Table 3 Detailed complexity analysis

Name	Classic	Generalized
Büchi	$\mathrm{Inf}(0)$ $x=1,\ y=0,\ d=0,\ m=0$ $3\cdot 5^n$	$\bigwedge_{j=0}^{k-1}\mathrm{Inf}(j)$ $x=k,\ y=0,\ d=0,\ m=0$ $3\cdot(2k+3)^n$
co-Büchi	$\mathrm{Fin}(0)$ $x=0,\ y=1,\ d=1,\ m=1$ $3\cdot 9^n$	$\bigvee_{j=0}^{k-1}\mathrm{Fin}(j)$ $x=0,\ y=k,\ d=k,\ m=1$ $3\cdot 2^{(k+3)n}$
Rabin pair	$\mathrm{Fin}(B)\wedge\mathrm{Inf}(G)$ $x=1,\ y=1,\ d=1,\ m=1$ $3\cdot 13^n$	$\mathrm{Fin}(B)\wedge\bigwedge_{i=0}^{\ell-1}\mathrm{Inf}(G_i)$ $x=\ell,\ y=1,\ d=1,\ m=1$ $3\cdot(4\ell+9)^n$
Rabin	$\bigvee_{j=0}^{k-1}\mathrm{Fin}(B_j)\wedge\mathrm{Inf}(G_j)$ $x=k,\ y=k,\ d=1,\ m=k$ $3\cdot 2^{(3k+2)n}$	$\bigvee_{j=0}^{k-1}(\mathrm{Fin}(B_j)\wedge\bigwedge_{i=0}^{\ell-1}\mathrm{Inf}(G_{j,i}))$ $x=k\ell,\ y=k,\ d=1,\ m=k$ $3\cdot 2^{(3k\ell+2)n}$
Streett pair	$\mathrm{Inf}(G)\vee\mathrm{Fin}(B)$ $x=1,\ y=1,\ d=1,\ m=1$ $3\cdot 17^n$	$\mathrm{Inf}(G)\vee\bigvee_{i=0}^{\ell-1}\mathrm{Fin}(B_i)$ $x=1,\ y=\ell,\ d=\ell,\ m=1$ $3\cdot 2^{(\ell+4)n}$
Streett	$\bigwedge_{j=0}^{k-1}\mathrm{Inf}(G_j)\vee\mathrm{Fin}(B_j)$ $x=k,\ y=k,\ d=1,\ m=k$ $6\cdot k^n\cdot 2^{(2k+2)n}$	$\bigwedge_{j=0}^{k-1}(\mathrm{Inf}(G_j)\vee\bigvee_{i=0}^{\ell-1}\mathrm{Fin}(B_{j,i}))$ $x=k,\ y=k\ell,\ d=\ell,\ m=k$ $6\cdot k^n\cdot 2^{(2k\ell+2)n}$
Parity	$\mathrm{Fin}(0)\wedge(\mathrm{Inf}(1)\vee(\mathrm{Fin}(2)\wedge(\mathrm{Inf}(3)\vee(\mathrm{Fin}(4)\wedge\ldots\mathrm{Inf}(k))\ldots))))$ $(\mathrm{Fin}(0)\wedge\mathrm{Inf}(1))\vee(\mathrm{Fin}(0,2)\wedge\mathrm{Inf}(1,3))\vee\ldots\vee(\mathrm{Fin}(0,\ldots,k\text{-}1)\wedge\mathrm{Inf}(1,\ldots,k))$ $x=\frac{k}{2},\ y=\frac{k}{2},\ d=1,\ m=\frac{k}{2}$ $3\cdot 2^{(\frac{3}{2}k+2)n}$	

Upper bounds for popular acceptance conditions are given in Table 3 (a simplified version was already given in Table 1). We note that we could obtain better bounds for the concrete constructions in Sections 3 and 4 because of more fine-tuned construction and analysis.

6 Complementation of an ELEA and a General ELA

Modular construction of an ELEA. To lift the complementation procedure from semi-deterministic to elevator automata, we employ the modular complementation approach [21] generalized to the ELA setting. The modular construction decomposes complementation of the whole automaton $\mathcal{A}$ into complementation algorithms for individual SCCs or groups of SCCs called partitions. Each partial complementation algorithm specifies its own macrostates, transition function, coloring, and acceptance condition. The transition function takes into account runs entering the partition from outside. The inductive construction from Section 5 fits naturally into this framework: the macrostates of the partial algorithm take the form $(C, \mathcal{M}_\psi)$, where the N component is omitted since incoming runs are tracked by the top-level modular algorithm. Concretely, in Algorithm 2 the set N' represents the newly incoming runs supplied by the top-level algorithm, and $\mathcal{Q}_d$ denotes the states of the deterministic partition. To obtain a complete complementation algorithm for ELEAs, we additionally plug the Miyano-Hayashi algorithm [44] into the modular construction to handle inherently-weak partitions and the same algorithm with round robin (similar to the procedure in Section 4) to handle generalized co-Büchi partitions with the complexity $\mathcal{O}(3^n)$ and $\mathcal{O}(k\cdot(3^n))$ respectively. An example of the modular construction is shown in [2].

Elevatorization of ELAs. Any ELA can be transformed into an equivalent ELEA, which lifts our techniques to the general case. The conversion treats each SCC independently. In particular, nondeterministic accepting components (NACs) must be semi-determinized [14] since they violate the elevator condition. Because semi-determinization yields only inherently-weak and deterministic components, the resulting automaton is an ELEA. Let $\mathcal{A}$ be an ELA with acceptance condition Acc. For each NAC, Acc can be simplified to contain only colors appearing in the NAC. This yields a smaller local condition that is used in place of Acc when applying the semi-determinization construction of [29] to the corresponding NAC. After semi-determinization, a copy of the NAC without colors is connected to a deterministic part that emits a fresh color ⊤, and $\mathsf{Inf}(⊤)$ becomes the acceptance condition of this semi-deterministic component. To prevent interference between SCCs of $\mathcal{A}$, we disable Acc within all semi-determinized components by marking them with a fresh color ⊥, and adopt $(\mathsf{Acc} \wedge \mathsf{Fin}(⊥)) \vee \mathsf{Inf}(⊤)$ as the global acceptance condition. The semi-determinization of [29] runs in $\mathcal{O}(2^{|\varphi|+|Q|})$, where Q is the set of states and φ is the simplified acceptance condition of the given SCC. Details are given in [2].

7 Experimental Evaluation

Implementation. We extended Kofola [1] with direct support for ELA complementation and integrated the inductive construction into its modular framework. We evaluate four configurations of the inductive construction: (i) Kofola-Ind: the base inductive construction without any optimizations; (ii) Kofola-Ind-SHB-Root: includes the shared breakpoint optimization, where a single context is maintained at the root of each tree-macrostate; (iii) Kofola-Ind-SHB-Subtree: includes the shared breakpoint optimization, where a context is maintained within each subtree consisting solely of ∧-nodes and Inf-leaves (allowing multiple contexts per tree-macrostate); and (iv) Kofola-Ind-FOR: employs the OR-FIN optimization.

Used tools and environment. We experimentally evaluated the inductive construction and compared it with the state-of-the-art tool Spot [16] (we use a custom build with the maximum number of colors increased from 32 to 128) and its variant Spot-High, which applied Spot's High reduction on the output. All experiments were conducted on an Ubuntu GNU/Linux 24.04 virtual machine with 64 GiB of RAM, running on an AMD EPYC 9124 CPU server under the Proxmox Virtual Environment. The timeout was 120 s. The correctness of results was validated by cross-checking the outputs using Spot's `autcross`.

Benchmarks. Our evaluation is based on the following three data sets (available at [37]):

- **Existing LTL benchmarks.** We used a collection of LTL formulas from the literature, taken from the benchmarks used in the evaluation of `ltl3tela` [42]. The benchmark set is available in the `ltl3tela` repository [41] and is based on formulas generated by `genltl` as well as formulas collected from prior work. After filtering out trivial cases, this resulted in 1,378 instances. These automata were then transformed into elevator automata. We denote this benchmark as **LTL-Lit**.
- **Randomly generated automata.** We generated 315 automata with a single generalized Rabin pair acceptance and transformed them into elevator automata while preserving the generalized Rabin pair acceptance form. We denote this benchmark as **GRA**.
- **Random LTL formulas.** We used Spot's `randltl` with parameters `--weak-fairness -l --seed=42128971 -n -1 --simplify=0 a b c d e` to generate random LTL formu-

Table 4 Statistics for complementation. The column **unsolved** shows the number of automata for which complementation failed (out of 2,693), including timeouts and errors such as running out of resources or exceeding the maximum number of acceptance sets. The columns **average**, **median**, and **total states** give the average, median, and total numbers of states of the resulting automata (across successful runs). The column **time** contains the total runtime (in seconds).

tool	unsolved	average	median	total states	time
Kofola-Ind-FOR	40	2,436	25	6,463,028	2,828
Kofola-Ind	247	537	21	1,312,689	3,629
Kofola-Ind-SHB-Root	226	639	26	1,576,427	3,502
Kofola-Ind-SHB-Subtree	242	547	23	1,339,994	3,066
Spot	156	15,058	39	38,202,295	673
Spot-High	159	11,136	15	28,218,675	1,167

las. The generated formulas were subsequently translated into automata in the `HOA` format using `ltl3tela` [42]. We used `autfilt` to remove trivial automata (such as deterministic, weak, empty, or terminal ones), and kept only elevator automata. This way, we obtained 1,000 instances. We denote this benchmark as **LTL-Rand**.

Discussion. We give the results comparing sizes of outputs of our four configurations and Spot in Figure 2 and Table 4. Figure 2a evaluates the OR-FIN optimization (Section 5.3) against the baseline; it shows that the optimization is mostly advantageous and essential for solving some benchmarks (it solves 207 more benchmarks than the baseline; we note that in a few cases, the baseline, however, could solve some benchmarks that the optimization could not). The second plot, in Figure 2b, evaluates the two shared breakpoint optimizations; it shows that although Kofola-Ind-SHB-Root solves slightly more benchmarks, Kofola-Ind-SHB-Subtree almost always outputs a slightly smaller automaton (if it finishes). The last plot, Figure 2c, evaluates our best configuration, Kofola-Ind-FOR, against Spot-High; on a part of the inputs, the two tools give comparable results (Spot-High in many cases giving slightly smaller outputs), however, on more challenging inputs, Kofola-Ind-FOR mostly dominates Spot-High by a large margin and can solve 119 more benchmarks.

In Table 4, we provide statistical data for our experiments. We highlight Kofola-Ind-FOR, which solved the most benchmarks by a large margin (the second one was Spot with 116 less solved benchmarks). Although the average size of the output is larger than other configurations of Kofola, this is mostly because Kofola-Ind-FOR can complement many of the more challenging inputs (this also holds for the median and the total number of states). On the other hand, even with the more finished runs, Kofola-Ind-FOR was the fastest configuration of Kofola (but still not as good as Spot—which is quite optimized—when we

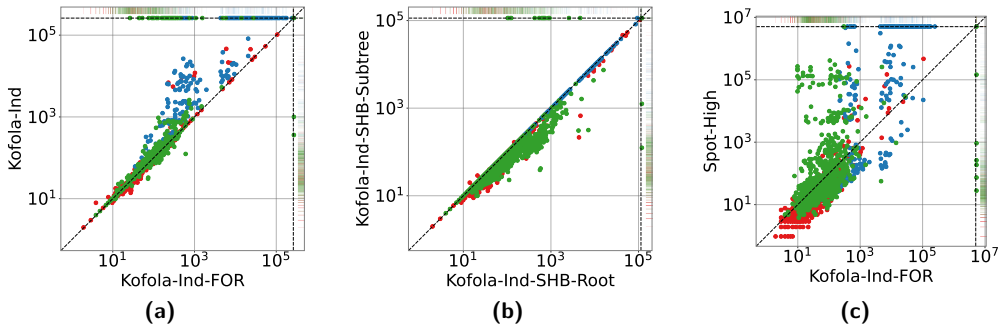


Figure 2 Scatter-plots comparing sizes of complemented automata (in the number of states). Colors denote benchmarks: • **GRA**, • **LTL-Rand**, and • **LTL-Lit**.

compare the runtime on only finished instances). We emphasize the total number of states generated by Kofola-Ind-FOR: roughly 4.4× less than Spot-High despite solving many more cases.

8 Related Work

Complementation of finite automata on infinite words is a problem addressed since the trailblazing work of Büchi [7]. For the basic model of Büchi automata, the lower bound for the problem is $\Omega((0.76n)^n)$ established using the full automata technique in [53] by Yan (improving the previous $\Omega(n!)$ lower bound of Michel [43]). A multitude of approaches for complementing BAs have been proposed in the literature, e.g., Ramsey-based [6, 7, 50], rank-based [20, 23, 22, 11, 51, 32, 49], determinization-based [47, 46, 38], slice-based [30], and others [3, 21, 39]. Some of these approaches [49, 3] even match the lower bound (modulo a small polynomial factor). More efficient complementation algorithms for structural subclasses of BAs were also considered, e.g., for inherently-weak ($\frac{2}{3}3^n$) [44], deterministic ($n+1$ with output being a co-Büchi automaton or $2n$ with output being a BA) [35], semideterministic (4^n) [4, 12], elevator (a GBA with 4^n states and two colors) [23, 21], or unambiguous (4^n) [40, 19] BAs.

The landscape is considerably less explored for richer acceptance conditions (we recommend Boker's excellent survey [5] and its accompanying web page for more details and references). For generalized BAs (GBAs) with n states and k acceptance sets, [34] gives a complementation algorithm with the complexity $2^{\mathcal{O}(n \log kn)}$. The GBA complementation algorithm given in [24] produces a result as a BA with $\mathcal{O}(n(0.76nk)^n)$ states and [15] claims to improve over this. Co-Büchi automata can be complemented into BAs with $\frac{2}{3}3^n$ states [44, 5]. For a Rabin automaton with n states and ℓ pairs, the algorithm in [33] produces a complement BA with $\mathcal{O}(\ell \cdot 3^n \cdot (2n+1)^{n\ell})$ states and the one in [24] produces a GBA with $\mathcal{O}(n^\ell(0.76n)^{n\ell})$ states and ℓ colours. Streett automata can be complemented into BAs with $2^{\mathcal{O}(n\ell \log n\ell)}$ states using [33] and into BAs with $2^{\mathcal{O}(n \log n + n\ell \log \ell)}$ states [9]. Parity automata can be complemented into BAs with $2^{\mathcal{O}(n \log n)}$ states. In [48], it is shown that the lower bound for the number of states of a complement of an ELA is 2^{2^n} (even if the output can also be an ELA), and the same work shows that the upper bound is $2^{2^{\mathcal{O}(n)}}$ (not considering the complexity of the acceptance condition as a parameter). This was refined in [24], which gave an algorithm that produces a GBA with 2^k colors and $\mathcal{O}(n^{2^k}(0.76nk)^{n2^k})$ states, where n is the number of states of the input ELA and k is the number of colors that occur in the acceptance condition. All of our upper bounds are better.

Elevator automata were introduced in [23] in the context of rank-based complementation as a subclass of BAs with a more efficient complementation procedure ($\mathcal{O}(16^n)$). Since then, it was shown that there exists on optimal determinization procedure ($\mathcal{O}(n!)$) [38] and an efficient complementation procedure ($\mathcal{O}(4^n)$) [21] for them. A recent paper [1] shows that the vast majority of BAs considered in practical scenarios are elevator automata.

References

1 Ondrej Alexaj, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, Yong Li, and Nicolas Mazzocchi. Kofola 1.0: A modular approach to ω-regular complementation and inclusion checking. In *CAV'26 (to appear)*, 2026.

2 Ondrej Alexaj, Vojtěch Havlena, Ondřej Lengál, Yong Li, and Nicolas Mazzocchi. Complementing Emerson-Lei elevator automata (technical report). *CoRR*, abs/2606.26768, 2026. URL: `http://arxiv.org/abs/2606.26768`, `arXiv:2606.26768`.

3 Joël D. Allred and Ulrich Ultes-Nitsche. A simple and optimal complementation algorithm for Büchi automata. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 46–55. ACM, 2018. `doi:10.1145/3209108.3209138`.

4 František Blahoudek, Matthias Heizmann, Sven Schewe, Jan Strejček, and Ming-Hsien Tsai. Complementing semi-deterministic Büchi automata. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 770–787. Springer, 2016. URL: `https://doi.org/10.1007/978-3-662-49674-9_49`, `doi:10.1007/978-3-662-49674-9_49`.

5 Udi Boker. Why these automata types? In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 143–163. EasyChair, 2018. URL: `https://doi.org/10.29007/c3bj`, `doi:10.29007/C3BJ`.

6 Stefan Breuers, Christof Löding, and Jörg Olschewski. Improved Ramsey-based Büchi complementation. In Lars Birkedal, editor, *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, pages 150–164. Springer, 2012. URL: `https://doi.org/10.1007/978-3-642-28729-9_10`, `doi:10.1007/978-3-642-28729-9_10`.

7 J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proc. of International Congress on Logic, Method, and Philosophy of Science 1960*. Stanford Univ. Press, Stanford, 1962.

8 Yang Cai and Ting Zhang. A tight lower bound for Streett complementation. In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 339–350. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. URL: `https://doi.org/10.4230/LIPIcs.FSTTCS.2011.339`, `doi:10.4230/LIPICS.FSTTCS.2011.339`.

9 Yang Cai and Ting Zhang. Tight upper bounds for Streett and parity complementation. In Marc 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. URL: `https://doi.org/10.4230/LIPIcs.CSL.2011.112`, `doi:10.4230/LIPICS.CSL.2011.112`.

10 Krishnendu Chatterjee, Andreas Gaiser, and Jan Křetínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, Lecture Notes in Computer Science, pages 559–575. Springer, 2013. URL: `https://doi.org/10.1007/978-3-642-39799-8_37`, `doi:10.1007/978-3-642-39799-8_37`.

11 Yu-Fang Chen, Vojtěch Havlena, and Ondřej Lengál. Simulations in rank-based Büchi automata complementation. In Anthony Widjaja Lin, editor, *Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*,

volume 11893 of *Lecture Notes in Computer Science*, pages 447–467. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-34175-6_23, doi:10.1007/978-3-030-34175-6_23.

12 Yu-Fang Chen, Matthias Heizmann, Ondřej Lengál, Yong Li, Ming-Hsien Tsai, Andrea Turrini, and Lijun Zhang. Advanced automata-based algorithms for program termination checking. In Jeffrey S. Foster and Dan Grossman, editors, *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*, pages 135–150. ACM, 2018. doi:10.1145/3192366.3192405.

13 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *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, pages 265–284. Springer, 2014. URL: https://doi.org/10.1007/978-3-642-54792-8_15, doi:10.1007/978-3-642-54792-8_15.

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

15 David Dokoupil. Efficient complementation of generalized Büchi automata. Master's thesis, Faculty of Informatics, Masaryk University, 2026. URL: https://is.muni.cz/th/xu3iz/.

16 Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In Sharon Shoham and Yakir 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. URL: https://doi.org/10.1007/978-3-031-13188-2_9, doi:10.1007/978-3-031-13188-2_9.

17 E. Allen Emerson and Chin-Laung 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.

18 Javier Esparza, Jan Křetínský, and Salomon Sickert. From LTL to deterministic automata - A safraless compositional approach. *Formal Methods Syst. Des.*, 49(3):219–271, 2016. URL: https://doi.org/10.1007/s10703-016-0259-2, doi:10.1007/S10703-016-0259-2.

19 Weizhi Feng, Yong Li, Andrea Turrini, Moshe Y. Vardi, and Lijun Zhang. On the power of finite ambiguity in Büchi complementation. *Inf. Comput.*, 292:105032, 2023. URL: https://doi.org/10.1016/j.ic.2023.105032, doi:10.1016/J.IC.2023.105032.

20 Vojtěch Havlena and Ondřej Lengál. Reducing (to) the ranks: Efficient rank-based Büchi automata complementation. In Serge Haddad and Daniele 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. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2021.2, doi:10.4230/LIPICS.CONCUR.2021.2.

21 Vojtěch Havlena, Ondřej Lengál, Yong Li, Barbora Šmahlíková, and Andrea Turrini. Modular mix-and-match complementation of Büchi automata. In Sriram Sankaranarayanan and Natasha 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 2022, Paris, France, April 22-27, 2023, Proceedings, Part I*, volume 13993 of *Lecture Notes in Computer Science*, pages 249–270. Springer, 2023. URL: https://doi.org/10.1007/978-3-031-30823-9_13, doi:10.1007/978-3-031-30823-9_13.

22 Vojtěch Havlena, Ondřej Lengál, and Barbora Šmahlíková. Complementing Büchi automata with Ranker. In Sharon Shoham and Yakir 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. URL: https://doi.org/10.1007/978-3-031-13188-2_10, doi:10.1007/978-3-031-13188-2_10.

23 Vojtěch Havlena, Ondřej Lengál, and Barbora Šmahlíková. Sky is not the limit: Tighter rank bounds for elevator automata in Büchi automata complementation. In Dana Fisman and Grigore 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. URL: https://doi.org/10.1007/978-3-030-99527-0_7, doi:10.1007/978-3-030-99527-0_7.

24 Vojtěch Havlena, Ondřej Lengál, and Barbora Šmahlíková. Complementation of Emerson-Lei automata. In Parosh Aziz Abdulla and Delia 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. URL: https://doi.org/10.1007/978-3-031-90897-2_5, doi:10.1007/978-3-031-90897-2_5.

25 Matthias Heizmann, Yu-Fang Chen, Daniel Dietsch, Marius Greitschus, Jochen Hoenicke, Yong Li, Alexander Nutz, Betim Musa, Christian Schilling, Tanja Schindler, and Andreas Podelski. Ultimate Automizer and the search for perfect interpolants (competition contribution). In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*, Lecture Notes in Computer Science, pages 447–451. Springer, 2018. URL: https://doi.org/10.1007/978-3-319-89963-3_30, doi:10.1007/978-3-319-89963-3_30.

26 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Software model checking for people who love automata. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, Lecture Notes in Computer Science, pages 36–52. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-39799-8_2, doi:10.1007/978-3-642-39799-8_2.

27 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Termination analysis by learning terminating programs. In Armin Biere and Roderick Bloem, editors, *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, pages 797–813. Springer, 2014. URL: https://doi.org/10.1007/978-3-319-08867-9_53, doi:10.1007/978-3-319-08867-9_53.

28 Philipp Hieronymi, Dun Ma, Reed Oei, Luke Schaeffer, Christian Schulz, and Jeffrey O. Shallit. Decidability for Sturmian words. *Log. Methods Comput. Sci.*, 20(3), 2024. URL: https://doi.org/10.46298/lmcs-20(3:12)2024, doi:10.46298/LMCS-20(3:12)2024.

29 Tobias John, Simon Jantsch, Christel Baier, and Sascha Klüppelholz. Determinization and limit-determinization of Emerson-Lei automata. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, Lecture Notes in Computer Science, pages 15–31. Springer, 2021. URL: https://doi.org/10.1007/978-3-030-88885-5_2, doi:10.1007/978-3-030-88885-5_2.

30 Detlef Kähler and Thomas Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *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, pages 724–735. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-70575-8_59, doi:10.1007/978-3-540-70575-8_59.

31 Yonit Kesten and Amir Pnueli. A complete proof systems for QPTL. In *Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science, San Diego, California, USA, June 26-29, 1995*, pages 2–12. IEEE Computer Society, 1995. doi:10.1109/LICS.1995.523239.

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

33 Orna Kupferman and Moshe Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In Nicolas Halbwachs and Lenore 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. URL: https://doi.org/10.1007/978-3-540-31980-1_14, doi:10.1007/978-3-540-31980-1_14.

34 Orna Kupferman and Moshe Y. Vardi. From complementation to certification. *Theor. Comput. Sci.*, 345(1):83–100, 2005. doi:10.1016/j.tcs.2005.07.021.

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

36 Jan Křetínský and Javier Esparza. Deterministic automata for the (F, G)-fragment of LTL. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, Lecture Notes in Computer Science, pages 7–22. Springer, 2012. URL: https://doi.org/10.1007/978-3-642-31424-7_7, doi:10.1007/978-3-642-31424-7_7.

37 Ondřej Lengál et al. Automata benchmarks. https://github.com/ondrik/automata-benchmarks, 2026.

38 Yong Li, Andrea Turrini, Weizhi Feng, Moshe Y. Vardi, and Lijun Zhang. Divide-and-conquer determinization of Büchi automata based on SCC decomposition. In Sharon Shoham and Yakir 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. URL: https://doi.org/10.1007/978-3-031-13188-2_8, doi:10.1007/978-3-031-13188-2_8.

39 Yong Li, Andrea Turrini, Lijun Zhang, and Sven Schewe. Learning to complement Büchi automata. In Isil Dillig and Jens Palsberg, editors, *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, pages 313–335. Springer, 2018. URL: https://doi.org/10.1007/978-3-319-73721-8_15, doi:10.1007/978-3-319-73721-8_15.

40 Yong Li, Moshe Y. Vardi, and Lijun Zhang. On the power of unambiguity in Büchi complementation. In Jean-François Raskin and Davide Bresolin, editors, *Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2020, Brussels, Belgium, September 21-22, 2020*, EPTCS, pages 182–198, 2020. doi:10.4204/EPTCS.326.12.

41 Juraj Major. LTL3TELA benchmark formulae. https://github.com/jurajmajor/ltl3tela/tree/master/Experiments/formulae, 2019. commit 02092adca492678407bd74b5d3bbe103d8c2b400.

42 Juraj Major, František Blahoudek, Jan Strejček, Miriama Sasaráková, and Tatiana Zbončáková. ltl3tela: LTL to small deterministic or nondeterministic Emerson-Lei automata. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, Lecture Notes in Computer Science, pages 357–365. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-31784-3_21, doi:10.1007/978-3-030-31784-3_21.

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

44 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32(3):321–330, 1984. URL: https://www.sciencedirect.com/science/article/pii/0304397584900495, doi:10.1016/0304-3975(84)90049-5.

45 Reed Oei, Dun Ma, Christian Schulz, and Philipp Hieronymi. Pecan: An automated theorem prover for automatic sequences using Büchi automata. *CoRR*, abs/2102.01727, 2021. URL: https://arxiv.org/abs/2102.01727, arXiv:2102.01727.

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

47 Shmuel Safra. On the complexity of ω-automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 319–327. IEEE Computer Society, 1988. doi:10.1109/SFCS.1988.21948.

48 Shmuel Safra and Moshe Y. Vardi. On ω-automata and temporal logic (preliminary report). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 127–137. ACM, 1989. doi:10.1145/73007.73019.

49 Sven Schewe. Büchi complementation made tight. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. URL: https://doi.org/10.4230/LIPIcs.STACS.2009.1854, doi:10.4230/LIPICS.STACS.2009.1854.

50 A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49(2-3):217–237, 1987.

51 Moshe Y. Vardi. The Büchi complementation saga. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, Lecture Notes in Computer Science, pages 12–22. Springer, 2007. URL: https://doi.org/10.1007/978-3-540-70918-3_2, doi:10.1007/978-3-540-70918-3_2.

52 Moshe Y. Vardi and Pierre Wolper. 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*, pages 332–344. IEEE Computer Society, 1986.

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