From LTL to unambiguous Büchi automata via disambiguation of alternating automata

Simon Jantsch¹ · David Müller¹ · Christel Baier¹ · Joachim Klein¹

Received: 6 March 2020 / Accepted: 4 June 2021 / Published online: 15 December 2021
© The Author(s) 2021

Abstract

Due to the high complexity of translating linear temporal logic (LTL) to deterministic automata, several forms of “restricted” nondeterminism have been considered with the aim of maintaining some of the benefits of deterministic automata, while at the same time allowing more efficient translations from LTL. One of them is the notion of *unambiguity*. This paper proposes a new algorithm for the generation of unambiguous Büchi automata (UBA) from LTL formulas. Unlike other approaches it is based on a known translation from *very weak alternating automata* (VWAA) to NBA. A notion of unambiguity for alternating automata is introduced and it is shown that the VWAA-to-NBA translation preserves unambiguity. Checking unambiguity of VWAA is determined to be PSPACE-complete, both for the explicit and symbolic encodings of alternating automata. The core of the LTL-to-UBA translation is an iterative disambiguation procedure for VWAA. Several heuristics are introduced for different stages of the procedure. We report on an implementation of our approach in the tool Duggi and compare it to an existing LTL-to-UBA implementation in the SPOT tool set. Our experiments cover model checking of Markov chains, which is an important application of UBA.

Keywords ω -Automata · Unambiguous Büchi automata · Linear temporal logic · Verification · Alternating automata

We thank the anonymous reviewers for many valuable comments and suggestions. The authors are supported by the DFG through the Collaborative Research Centers CRC 912 (HAEC), the DFG Grant 389792660 as part of TRR 248, the DFG-project BA-1679/12-1, the Cluster of Excellence EXC 2050/1 (CeTI, Project ID 390696704, as part of Germany’s Excellence Strategy), and the Research Training Group QuantLA (GRK 1763).

✉ Simon Jantsch
simon.jantsch@tu-dresden.de
David Müller
david.mueller2@tu-dresden.de
Christel Baier
christel.baier@tu-dresden.de
Joachim Klein
joachim.klein@tu-dresden.de

¹ Technische Universität, Dresden, Germany

1 Introduction

Translations from linear temporal logic (LTL) to non-deterministic Büchi automata (NBA) have been studied intensively as they are a core ingredient in the classical algorithmic approach to LTL model checking [4,10,45]. While it is known that LTL formulas can be exponentially more succinct in comparison to equivalent NBA, these translations have been optimized and thus made LTL-to-NBA translations viable in practice. In particular, [12,19] offer a tableau-based approach which is implemented in the tool SPOT [16]. Another approach exploits *very weak alternating automata* (VWAA) [18], where LTL3BA [3] is the leading tool currently.

For some applications NBA cannot be used directly. In probabilistic model checking, for example, the standard approach requires the construction of a deterministic automaton. Determinization of NBA is more intricate than for automata over finite words [36,38], however, and translations from LTL to deterministic automata over infinite words induce a double exponential blow up in the worst case [27,28]. A subclass of NBA that eludes some of these problems are *unambiguous* Büchi automata (UBA) [11]. Unambiguity allows non-deterministic branching but requires that each input word has at most one accepting run. LTL can be translated into equivalent UBA with a single exponential blow-up [45].

A prominent case in which unambiguity can be utilized is the universality check (“Is every word accepted?”) for automata on finite words. It is PSPACE-complete for arbitrary non-deterministic finite automata (NFA), but in P for unambiguous NFA [42]. Universality and language inclusion are in P for subclasses of UBA [8,23], but the complexity is open for general UBA. A promising application of UBA is the verification of Markov chains against ω -regular specifications. This problem is PSPACE-hard for arbitrary NBA [43], but in P if the specification is a UBA [5]. Thus, using UBA yields a single-exponential algorithm for LTL model checking of Markov chains, whereas using deterministic automata always involves a double-exponential lower bound in time complexity.

The translation from LTL to NBA by Vardi and Wolper [45] produces *separated* automata, a subclass of UBA where the languages of the states are pairwise disjoint. Their construction did not specifically intend to produce separated automata and is asymptotically optimal for the LTL-to-NBA translation. Separated automata (and hence also UBA) can express all ω -regular languages [9], but UBA may be exponentially more succinct [8]. LTL-to-NBA translations have been studied intensively [14,17–19]. The generation of UBA from LTL formulas, however, has not received much attention so far.

We are aware of three approaches targeted explicitly at generating UBA or subclasses. The approach by Couvreur et al. [13] adapts the algorithm of [45], but still generates separated automata. LTL-to-UBA translations that attempt to exploit the advantages of UBA over separated automata have been presented by Benedikt et al. [7] and Duret-Lutz [15]. They are based on tableau-based LTL-to-NBA algorithms ([19] in the case of [7,12] in the case of [15]) and rely on transformations of the form $\varphi \vee \psi \rightsquigarrow \varphi \vee (\neg\varphi \wedge \psi)$ to enforce that splitting disjunctive formulas generates states with disjoint languages, which ensures unambiguity. To the best of our knowledge, the only available tool that includes a LTL-to-UBA translation (the one of [15]) is `ltl2tgb`, which is part of SPOT.

Contribution We introduce a new LTL-to-UBA construction.¹ It adapts the construction of Gastin and Oddoux [18], and exploits the advantages of the VWAA representation of LTL formulas. More concretely, the main contributions are as follows. We introduce a notion of

¹ This paper is an extended version of the conference paper [24].

Fig. 1 The LTL-to-UBA translation, which applies a sequence of unambiguity checks and disambiguation transformations. We use $\text{trim}(\mathcal{G}_A \otimes \mathcal{G}_A)$ (the self-product of $\mathcal{G}_A$ after removing states with empty language) to check whether unambiguity is achieved or more iterations are necessary

Fig. 2 Overview of the general LTL-to-UBA generation algorithm. The LTL simplification step, the actual LTL-to-UBA translation step, and the automaton post processing step can be combined freely. We propose novel rewriting rules for LTL and a LTL-to-UBA translation, both implemented in our tool *Duggi*

unambiguity for alternating automata, and show that the typical steps of translating VWAA into NBA preserve unambiguity (Sect. 3.1). Additionally we show that checking unambiguity of VWAA is PSPACE-complete, independently of whether the transitions are represented symbolically or explicitly (Sect. 3.2). The hardness proof for the explicit case uses a reduction from the emptiness problem for explicitly represented VWAA, whose PSPACE-hardness (Theorem 3.2) is an interesting result independent of unambiguity.

We describe an iterative disambiguation procedure for VWAA that relies on intermediate unambiguity checks to identify states causing ambiguity and local transformations to remove it (Sect. 4). The local transformations exploit alternating branching. Figure 1 gives an overview of our LTL-to-UBA algorithm. We enhance the main construction by new LTL rewrite rules specifically targeted at producing UBA (see Fig. 2), and heuristics both for VWAA disambiguation and shrinking of the automata as produced by the VWAA to NBA translation (Sect. 5). Finally, we report on an implementation of our construction in the tool *Duggi* and compare it to the existing LTL-to-UBA translator *ltl2tgba*. We also compare *Duggi* with *ltl2tgba* in the context of Markov chain analysis under LTL specifications (Sect. 6). We begin by recalling some standard definitions for LTL and alternating automata on ω -words, and the LTL to NBA translation via VWAA that we use as a starting point (Sect. 2).

2 Preliminaries

This section introduces our notation and standard definitions. The set of infinite words over a finite alphabet Σ is denoted by Σ^ω and we write $w[i]$ to denote the i th position of an infinite word $w \in \Sigma^\omega$, and $w[i..]$ to denote the suffix $w[i]w[i+1] \dots$. LTL is defined using $\mathcal{U}$ (“Until”) and $\bigcirc$ (“Next”). We use the syntactic derivations $\Diamond$ (“Finally”), $\Box$ (“Globally”), and $\mathcal{R}$ (“Release”) (see [4,20] for details), where the release operator satisfies $\varphi \mathcal{R} \psi \equiv \neg(\neg\varphi \mathcal{U} \neg\psi)$ for all LTL-formulas φ, ψ .

Alternating automata on infinite words An alternating ω -automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, \delta, \iota, \Phi)$ where Q is a non-empty, finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function, $\iota \subseteq 2^Q$ is the set of initial state sets and Φ is the acceptance condition. We write $\mathcal{A}(Q_0)$ for the automaton $\mathcal{A}$ with initial states $\iota = \{Q_0\}$. Analogously, for a single initial state q_0 , we write $\mathcal{A}(q_0)$.

Remark 2.1 The other standard way to define alternating automata is via a symbolic transition function and a symbolic initial condition [44]. In this case, the transition function is a function $\Delta : Q \times \Sigma \rightarrow \mathcal{B}^+(Q)$, where $\mathcal{B}^+(Q)$ is the set of positive Boolean formulas over Q as atomic propositions. Analogously, ι is replaced by a Boolean condition $\alpha \in \mathcal{B}^+(Q)$ to define an initial condition. Intuitively, the set of successor sets $\delta(q, a)$ is meant to correspond to the minimal models of the formula $\Delta(q, a)$. Using this idea one can construct Δ from δ and vice versa. Formulas false and true can be modeled by $\emptyset$ (no transition exists) and $\{\emptyset\}$ (a single transition with no constraints on the suffix word). The explicit definition has been proved to be useful for LTL to NBA translations [3,18].

Runs of ω -automata A run of $\mathcal{A}$ for $w \in \Sigma^\omega$ is a directed acyclic graph (DAG) (V, E) [31], where

1. $V \subseteq Q \times \mathbb{N}$, and $E \subseteq \bigcup_{0 \leq l} (Q \times \{l\}) \times (Q \times \{l+1\})$,
2. $\{q : (q, 0) \in V\} \in \iota$,
3. for all $(q, l) \in V : \{q' : ((q, l), (q', l+1)) \in E\} \in \delta(q, w[l])$,
4. for all $(q, l) \in V \setminus (Q \times \{0\})$ there is a q' such that $((q', l-1), (q, l)) \in E$.

We define $V(i) = \{s : (s, i) \in V\}$, called the i th layer of V and $E(q, k) = \{q' : ((q, k), (q', k+1)) \in E\}$, the successors of q in the k th layer. A run is called accepting if every infinite path in it meets the acceptance condition. A *run-prefix* (of length k) is a DAG (V, E) with $V \subseteq Q \times \{0, \dots, k-1\}$ satisfying the conditions above with the restriction that 3. is only required for all $(q, l) \in V$ with $l < k-1$.

Acceptance conditions We distinguish between Büchi, generalized Büchi and co-Büchi acceptance conditions. A Büchi condition is denoted by $\text{Inf}(Q_f)$ for a set $Q_f \subseteq Q$. An infinite path $\pi = q_0 q_1 \dots$ meets $\text{Inf}(Q_f)$ if $Q_f \cap \text{inf}(\pi) \neq \emptyset$, where $\text{inf}(\pi)$ denotes the set of infinitely occurring states in π . A co-Büchi condition is denoted by $\text{Fin}(Q_f)$ and π meets $\text{Fin}(Q_f)$ if $Q_f \cap \text{inf}(\pi) = \emptyset$. An infinite path π meets a generalized Büchi condition $\bigwedge_{Q_f \in F} \text{Inf}(Q_f)$ if it meets $\text{Inf}(Q_f)$ for all $Q_f \in F$. A *transition-based* acceptance condition uses sets of transitions $T \subseteq Q \times \Sigma \times Q$ instead of sets of states to define acceptance of paths. For example, the acceptance condition of a *transition-based generalized Büchi* automaton (t-GBA) is $\bigwedge_{T_f \in F} \text{Inf}(T_f)$, where the $T_f \in F$ are sets of transitions and $\text{Inf}(T_f)$ is defined analogously to the case of state-based acceptance. A word is accepted by $\mathcal{A}$ if there exists an accepting run for it. We denote the set of accepted words of $\mathcal{A}$ by $\mathcal{L}(\mathcal{A})$.

Configurations of ω -automata We call a subset $C \subseteq Q$ a configuration and say that C is reachable if there exists a run prefix of $\mathcal{A}$ with final layer C . A configuration C is said to be reachable from a state q , if C is a reachable configuration of $\mathcal{A}(q)$. Analogously, $C' \subseteq Q$ is reachable from $C \subseteq Q$, if C' is a reachable configuration of $\mathcal{A}(C)$. A configuration C is *reachable via* $u \in \Sigma^*$ if there is a run-prefix (V, E) for u with final layer C . We extend this notion to reachability from states and configurations via finite words in the expected way. We define $\mathcal{L}(C) = \mathcal{L}(\mathcal{A}(C))$, and write $\mathcal{L}_{\mathcal{A}}(C)$ if the underlying automaton is not clear from the context.

Properties of ω -automata The *underlying graph* of $\mathcal{A}$ has vertices Q and edges $\{(q, q') : \exists a \in \Sigma. \exists S \in \delta(q, a). q' \in S\}$. We say that $\mathcal{A}$ is *very weak* if every strongly connected component of its underlying graph consists of a single state and it has co-Büchi acceptance. We write $q \rightarrow^* s$ if s is reachable from q in the underlying graph. If $|C_0| = 1$ for every $C_0 \in \iota$ and $|C_\delta| = 1$ for every $C_\delta \in \delta(q, a)$ with $(q, a) \in Q \times \Sigma$, we call $\mathcal{A}$ non-deterministic. As a non-deterministic automaton has only singleton successor sets, its runs are infinite sequences of states. Finally, an automaton $\mathcal{A}$ is *trimmed* if $\mathcal{L}(q) \neq \emptyset$ holds for every state q in $\mathcal{A}$, and we write $\text{trim}(\mathcal{A})$ for the automaton that we get by removing all states with empty language in $\mathcal{A}$. For the non-alternating automata types that we consider, $\text{trim}(\mathcal{A})$ can be computed in linear time using standard graph algorithms.

Remark 2.2 Let $\mathcal{A}$ be an alternating automaton. Let $C_1, C_2 \in \delta(a, q)$ be two different successor sets for some state q of $\mathcal{A}$ and $a \in \Sigma$ and let $C_1 \subset C_2$. Then we can remove C_2 from $\delta(a, q)$, as from any accepting run starting in C_2 we can construct an accepting run starting in C_1 by restricting the states. Hence, we will from now on assume that no two successor sets satisfy strict inclusion. In general, this remark does not apply to non-deterministic automata as they satisfy $|C| = 1$ for all C, a, q such that $C \in \delta(a, q)$. However, in the translations we consider t-GBA states correspond to configurations of an alternating automaton. This fact allows removing transitions in a similar way, which is an optimization used in [3, 18].

From LTL to NBA The translation of LTL to NBA via VWAA consists of the three translation steps of LTL to VWAA, VWAA to transition-based generalized Büchi automata (t-GBA), and t-GBA to NBA [18]. We use the standard translation from LTL to VWAA where the states of the VWAA correspond to subformulas of φ and the transition relation follows the Boolean structure of the state and the LTL expansion laws for $\mathcal{U}$ and $\mathcal{R}$ (see [34, 44]):

$$\varphi \mathcal{U} \psi = \psi \vee (\varphi \wedge \bigcirc(\varphi \mathcal{U} \psi)) \quad \text{and} \quad \varphi \mathcal{R} \psi = \psi \wedge (\varphi \vee \bigcirc(\varphi \mathcal{R} \psi))$$

As in [18], we define the transitions of the VWAA as a function $\delta : Q \rightarrow 2^{2^\Sigma \times 2^Q}$. Hence, δ maps a state into a set of pairs (A, S) with $A \subseteq \Sigma$ and $S \subseteq Q$. Intuitively, A represents the symbols for which a transition to the successor set S is possible. The construction takes an LTL formula in which all negations appear in front of atomic propositions (*positive normal form*) as input. If all dual operators are included in the syntax (where $\mathcal{R}$ is the dual of $\mathcal{U}$), LTL formulas can be transformed into an equivalent formula in positive normal form of at most twice the length.

Definition 2.1 Let φ be an LTL formula in positive normal form over AP . We define the co-Büchi ω -automaton $\mathcal{A}_\varphi$ as a tuple $(Q, \Sigma, \delta, \bar{\varphi}, \text{Fin}(Q_f))$ where $\Sigma = 2^{AP}$, Q is the set of subformulas of φ , $Q_f = \{\psi_1 \mathcal{U} \psi_2 : \psi_1 \mathcal{U} \psi_2 \in Q\}$ and δ is defined as follows:

$$\begin{aligned} \delta(\text{true}) &= \{(\Sigma, \emptyset)\} & \delta(\text{false}) &= \emptyset \\ \delta(a) &= \{(\Sigma_a, \emptyset)\} & \delta(\neg a) &= \{(\Sigma_{\neg a}, \emptyset)\} \\ \delta(\varphi \wedge \psi) &= \delta(\varphi) \otimes \delta(\psi) & \delta(\varphi \vee \psi) &= \delta(\varphi) \cup \delta(\psi) \\ \delta(\bigcirc \varphi) &= \{(\Sigma, \{\psi\}) : \psi \in \bar{\varphi}\} \end{aligned}$$

$$\begin{aligned} \delta(\varphi \mathcal{U} \psi) &= \delta(\psi) \cup (\delta(\varphi) \otimes \{(\Sigma, \{\varphi \mathcal{U} \psi\})\}) \\ \delta(\varphi \mathcal{R} \psi) &= \delta(\psi) \otimes (\delta(\varphi) \cup \{(\Sigma, \{\varphi \mathcal{R} \psi\})\}) \end{aligned}$$

where

$$\begin{aligned}
 M_1 \otimes M_2 &= \{(u \cap v, Q_1 \cup Q_2) : \\
 &\quad (u, Q_1) \in M_1, (v, Q_2) \in M_2\} \quad \bar{\varphi} = \{\varphi\} \text{ if } \varphi \text{ is temporal} \\
 \Sigma_a &= \{m \in \Sigma : a \in m\} \quad \overline{\varphi \wedge \psi} = \{\varphi' \wedge \psi' : \varphi' \in \bar{\varphi}, \psi' \in \bar{\psi}\} \\
 \Sigma_{\neg a} &= \Sigma \setminus \Sigma_a \quad \overline{\varphi \vee \psi} = \bar{\varphi} \cup \bar{\psi}
 \end{aligned}$$

The operator $\otimes$ is used to model conjunctions. For two sets of pairs of the form $2^\Sigma \times 2^Q$ this amounts to including all pairs $(A_1 \cap A_2, S_1 \cup S_2)$, where (A_1, S_1) and (A_2, S_2) come from the respective sets. This represents a conjunction as it includes only transitions whose symbol is included in both A_1 and A_2 and requires the suffix word to be accepted from both S_1 and S_2 . Further down, the operation $\otimes$ is defined on sets of configurations in an analogous way.

The translation from LTL to VWAA has been refined in [3] to produce smaller automata with less non-determinism. The introduced optimizations do not alter or simplify our disambiguation algorithm and we use them in our implementation.

Lemma 2.1 [3,18] *Let φ and $\mathcal{A}_\varphi$ be as above. Then, $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}_\varphi)$.*

A VWAA $\mathcal{A}$ can be transformed into a t-GBA by a powerset-like construction, where the non-deterministic choices of $\mathcal{A}$ are captured by non-deterministic choices of the t-GBA, and the universal choices are captured by the powerset.

Definition 2.2 Let $\mathcal{A} = (Q, \Sigma, \delta, \iota, \text{Fin}(Q_F))$ be a VWAA. The t-GBA $\mathcal{G}_\mathcal{A}$ is the tuple $(2^Q, \Sigma, \delta', \iota, \bigwedge_{f \in Q_F} \text{Inf}(T_f))$, where

- $\delta'(C, a) = \bigotimes_{q \in C} \delta(q, a)$, where $T_1 \otimes T_2 = \{C_1 \cup C_2 : C_1 \in T_1, C_2 \in T_2\}$
- $T_f = \{(C, a, C') : f \notin C' \text{ or there exists } Y \in \delta(f, a) \text{ and } f \notin Y \subseteq C'\}$

The acceptance condition of $\mathcal{G}_\mathcal{A}$ intuitively makes sure that if a rejecting co-Büchi state f is included in a run infinitely often, then it also has a non-looping transition infinitely often.

Remark 2.3 We have not included an optimization from [18] which removes “subsumed” transitions, for example because the configurations corresponding to successor states are in a subset relation (see also Remark 2.2). The reason is that it breaks the strong correspondence of runs of $\mathcal{A}$ and $\mathcal{G}_\mathcal{A}$ (Lemma 3.3) that we use later for our complexity results (Lemma 3.4 and Lemma 3.1). However, Lemma 3.1 holds also when including the optimization, and so the proposed translation from unambiguous VWAA to UBA is correct with both definitions.

Theorem 2.1 (Theorem 2 of [18]) *Let $\mathcal{A}$ be a VWAA and $\mathcal{G}_\mathcal{A}$ be as in Definition 2.2. Then, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G}_\mathcal{A})$.*

The size of $\mathcal{G}_\mathcal{A}$ may be exponential in $|Q|$ and the number of Büchi conditions of $\mathcal{G}_\mathcal{A}$ is $|Q_F|$. Often a Büchi automaton with a (non-generalized) Büchi acceptance is desired. For this step we follow the construction of [18], which translates $\mathcal{G}_\mathcal{A}$ into an NBA $\mathcal{N}_{\mathcal{G}_\mathcal{A}}$ of at most $(|Q_F| + 1) \cdot 2^{|Q|}$ reachable states.

Definition 2.3 (Degeneralization) Let $\mathcal{G} = (Q, \Sigma, \delta, Q_0, \text{Inf}(T_1) \wedge \dots \wedge \text{Inf}(T_n))$ be a t-GBA. Then we define $\mathcal{N}_\mathcal{G}$ to be $(Q \times \{0, \dots, n\}, \Sigma, \delta_\mathcal{N}, Q_0 \times \{0\}, Q \times \{n\})$ where

$$\delta_\mathcal{N}(\langle q, i \rangle, a) = \{\langle q', i' \rangle : q' \in \delta(q, a) \text{ and } i' = \text{next}(i, q \xrightarrow{a} q')\} \text{ and}$$

Fig. 3 A VWAA (a) and t-GBA (b) for the formula $\Box\Diamond a$, as produced by the LTL to VWAA and VWAA to t-GBA translations respectively. The alphabet here is $\{a, \emptyset\}$, represented by a and $\neg a$ in the drawing above. The outgoing arrow of state $\Diamond a$ represents the transition to the empty successor set ($\delta(\Diamond a, a) = \{\emptyset\}$). Accepting transitions in the t-GBA are marked with a blue dot

$$\text{next}(i, q \xrightarrow{a} q') = \begin{cases} \max(\{i \leq j \leq n : \forall k \in \{i+1, \dots, j\}. q \xrightarrow{a} q' \in T_k\}) & \text{if } i \neq n \\ \max(\{0 \leq j \leq n : \forall k \in \{1, \dots, j\}. q \xrightarrow{a} q' \in T_k\}) & \text{if } i = n \end{cases}$$

This construction creates copies of $\mathcal{G}$ for every Büchi acceptance set $\text{Inf}(T_i)$ and switches from copy i to copy i' if and only if the current transition satisfies all acceptance sets $\text{Inf}(T_j)$ with $i < j \leq i'$. Therefore:

Theorem 2.2 (Theorem 3 of [18]) *Let $\mathcal{G}$ and $\mathcal{N}_{\mathcal{G}}$ be as above. Then $\mathcal{L}(\mathcal{G}) = \mathcal{L}(\mathcal{N}_{\mathcal{G}})$.*

3 Unambiguous VWAA

In this section we introduce a new notion of unambiguity for alternating automata and show that unambiguous VWAA are translated to UBA by the translation presented in Sect. 2. We define unambiguity in terms of configurations of the alternating automaton, which are strongly related to the states of the resulting NBA in this translation.

Definition 3.1 (*Unambiguity for alternating automata*) An alternating automaton is unambiguous if it has no distinct configurations C_1, C_2 that are reachable for the same word $u \in \Sigma^*$ and such that $\mathcal{L}(C_1) \cap \mathcal{L}(C_2) \neq \emptyset$.

The standard definition of unambiguity requires an automaton to have at most one accepting run for any word. In our setting runs are DAG's and we do allow multiple accepting runs for a word, as long as they agree on the configurations that they reach for each prefix. In this sense it is a weaker notion. However, the notions coincide on non-deterministic automata as the edge relation of the run is then induced by the sequence of visited states.

3.1 The VWAA to t-GBA translation preserves unambiguity

Let $\mathcal{A} = (Q, \Sigma, \delta, \iota, \text{Fin}(Q_f))$ be a fixed VWAA for the rest of this section. We now show that if $\mathcal{A}$ is unambiguous, then the translation presented in Sect. 2 yields an unambiguous Büchi automaton (Theorem 3.1). To prove this, we establish a mapping from accepting runs r of the generalized Büchi automaton $\mathcal{G}_{\mathcal{A}}$ to accepting runs ρ of $\mathcal{A}$ such that the sequence of layers of ρ is r (Lemma 3.1).

The lemma depends on the assumption that $\mathcal{A}$ is unambiguous. It is complicated by the way that accepting transitions are defined in $\mathcal{G}_{\mathcal{A}}$: they require that a final state q_f must have the option to choose a non-looping transition infinitely often (see Definition 2.2).

Fig. 4 A VWAA (a) and the corresponding t-GBA (b) as produced by the translation from VWAA to t-GBA. The run of the t-GBA that stays in the state $\{q, s, t\}$ forever is accepting. However, there is no accepting run of the VWAA with the same layers, since staying forever in q leads to a rejecting run

Thus, there might exist runs of $\mathcal{G}_A$ that represent runs of $\mathcal{A}$ where q_f has this option infinitely often, but never takes the “good” transition. An example for this behavior can be seen in Fig. 4. Here, the t-GBA state $\{q, s, t\}$ with its self-loop admits an infinite accepting run, whereas an accepting run of the VWAA is not allowed to visit q infinitely often. The reason that the self loop of the initial state of the t-GBA is accepting is that q has the option not to loop while staying in the set $\{q, s, t\}$, namely to move to s .

A way of dealing with this phenomenon is to exclude the “unnecessary” transitions of the t-GBA that correspond to the case when q has the option not to loop, but does not use it. The optimization discussed in Remark 2.3 achieves this, and using an optimized version of the translation would let us prove the following lemma without the precondition that $\mathcal{A}$ is unambiguous. However, we are happy to make this assumption as our goal is to show that the unambiguity is preserved, which holds also for the unoptimized version of the translation.

Lemma 3.1 *If $\mathcal{A}$ is unambiguous, then for every accepting run $r = Q_0Q_1 \dots$ of $\mathcal{G}_A$ for $w \in \Sigma^\omega$ there exists an accepting run $\rho = (V, E)$ of $\mathcal{A}$ for w such that $Q_i = V(i)$ for all $i \geq 0$.*

Proof The proof goes by induction over the number of elements in the set Q_f .

Base case: $Q_f = \emptyset$. We define $V = \{(q, i) : q \in Q_i\}$. $Q_0 \in \iota$ must hold as r is a run of $\mathcal{G}_A$ and thus the initial condition is satisfied. For every transition $Q_i \xrightarrow{w[i]} Q_{i+1}$ we know that $Q_{i+1} \in \bigotimes_{q \in Q_i} \delta(q, w[i])$ (as $\delta'(Q_i, w[i]) = \bigotimes_{q \in Q_i} \delta(q, w[i])$ by Definition 2.2) and we define the edges between $V(i)$ and $V(i+1)$ to match the corresponding successor sets. The result is a run of $\mathcal{A}$ for w and it is accepting as $Q_f = \emptyset$, and hence any run is accepting.

Now consider $Q_f = Q'_f \cup \{q_f\}$. Let $\mathcal{A} = (Q, \Sigma, \delta, \iota, \text{Fin}(Q_f))$ and $\mathcal{A}'$ be $\mathcal{A}$ where q_f is not marked final (i.e. $\mathcal{A}' = (Q, \Sigma, \delta, \iota, \text{Fin}(Q'_f))$). Let $r = Q_0Q_1 \dots$ be an accepting run of $\mathcal{G}_A$ for w . By Definition 2.2 the only difference between $\mathcal{G}_{\mathcal{A}'}$ and $\mathcal{G}_A$ is that $\mathcal{G}_A$ has the additional acceptance set $\mathcal{T}_{q_f}$, which adds an obligation for runs to be accepting. This implies that any accepting run of $\mathcal{G}_A$ is also an accepting run of $\mathcal{G}_{\mathcal{A}'}$. So r is an accepting run of $\mathcal{G}_{\mathcal{A}'}$ and we make use of the induction hypothesis to get an accepting run $\rho = (V, E)$ of $\mathcal{A}'$ such that for all i : $Q_i = V(i)$. If we can show that ρ is also an accepting run of $\mathcal{A}$ for w we are done.

So suppose that it is not accepting. Then there exists a rejecting path π through ρ that ultimately stabilizes on a state $s \in Q_f$. But s can only be q_f , as any other rejecting path would contradict the fact that ρ is an accepting run of $\mathcal{A}'$. Hence there is some k such that for all $j > k$: $q_f \in V(j)$ and $q_f \in E(q_f, j)$.

As $q_f \in Q_f$ we know that there are infinitely many i such that $(Q_i, w[i], Q_{i+1}) \in \mathcal{T}_{q_f}$. Hence, there must exist a triple (S, a, S') such that for infinitely many $i > k$: $Q_i = S$, $Q_{i+1} = S'$, $w[i] = a$ and $(S, a, S') \in \mathcal{T}_{q_f}$. Furthermore, for infinitely many of these i the

edges of ρ between S and S' will be the same. We fix an edge relation $e : S \rightarrow 2^{S'}$ between S and S' in ρ that occurs infinitely often and set:

$$J = \{j \in \mathbb{N} : S = V(j), S' = V(j+1), w[j] = a \text{ and } E(q, j) = e(q) \text{ for all } q \in S\}$$

It follows that $S' = \bigcup_{q \in S} e(q)$ and J is infinite. By the definition of $\mathcal{T}_{q_f}$ in all these positions the state q_f has a successor configuration Y such that $Y \in \delta(q_f, a)$, $q_f \notin Y$ and $Y \subseteq S'$.

We claim that the run that chooses Y instead of $e(q_f)$ as successor set of q_f on these positions also has the property $i: Q_i = V(i)$. To this end, using the fact that $\mathcal{A}$ is unambiguous, we show:

$$S' = Y \cup \underbrace{\left(\bigcup_{q \in S \setminus \{q_f\}} e(q) \right)}_{S''}$$

Suppose that this equality does not hold. Then S'' is strictly contained in S' , as $Y \subseteq S'$ and $e(q) \subseteq S'$ for every $q \in S$. But then for all $j \in J$ we can construct an accepting run ρ_j of $\mathcal{A}$ for w such that all pairs of runs in $\{\rho_j : j \in J\}$ differ on some layer. We construct $\rho_j = (V', E')$ by mimicking ρ up to position j and in position j we choose Y as successor configuration of q_f . For all following positions $k > j$ such that $k \in J$ we also choose Y as successor configuration of q_f , given that $q_f \in V'(k)$. We have $q_f \notin E'(q_f, k)$ for all $k \in J$ such that $k \geq j$, and thus no infinite path stabilizes on q_f . Furthermore, all infinite paths of ρ_j share a suffix with some infinite path in ρ , and hence do not visit any state $s \in Q'_f$ infinitely often. Therefore ρ_j is an accepting run of $\mathcal{A}$ on w .

The two accepting runs ρ_j, ρ_h with $j, h \in J$ and $j < h$ differ on depth $j+1$ as $\rho_j(j+1) = S''$ and $\rho_h(j+1) = S'$. This contradicts the unambiguity of $\mathcal{A}$.

Thus we conclude that $S' = S''$. We get an accepting run $\rho' = (V', E')$ by mimicking ρ for all positions $k \notin J$, and choosing Y as successor set for q_f in all positions $j \in J$. The property $Q_i = V'(i)$ holds for all $i \in \mathbb{N}$ as it holds for ρ and the layers of ρ and ρ' agree. $\square$

A direct consequence of Lemma 3.1 is that if $\mathcal{A}$ is unambiguous, then so is $\mathcal{G}_{\mathcal{A}}$, as multiple accepting runs of $\mathcal{G}_{\mathcal{A}}$ for some word would imply multiple accepting runs of $\mathcal{A}$ for the same word but with different layers.

Corollary 3.1 *If $\mathcal{A}$ is unambiguous, then $\mathcal{G}_{\mathcal{A}}$ is unambiguous.*

Furthermore, the degeneralization construction also preserves unambiguity:

Lemma 3.2 *Let $\mathcal{G}_{\mathcal{A}}$ be an unambiguous t-GBA and $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$ the result of degeneralizing $\mathcal{G}_{\mathcal{A}}$. Then, $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$ is unambiguous.*

Proof The degeneralization construction in [18] makes $|Q_f| + 1$ copies of $\mathcal{G}_{\mathcal{A}}$ (see Definition 2.3). As the next copy is uniquely determined by the current state and word label, the runs of $\mathcal{G}_{\mathcal{A}}$ and $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$ are in one-to-one correspondance. Hence, if $\mathcal{G}_{\mathcal{A}}$ is unambiguous, then so is $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$. $\square$

Theorem 3.1 *Let $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$ be the NBA for $\mathcal{A}$, obtained by the translation from Sect. 2. If $\mathcal{A}$ is unambiguous, then $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$ is unambiguous.*

Proof Follows from Corollary 3.1 and Lemma 3.2. $\square$

3.2 Complexity of unambiguity checking for VWAA

We now show that deciding whether a VWAA is unambiguous is PSPACE-complete. For this question it is important to consider how VWAA are represented (see also Remark 2.1). Alternating automata can be represented *symbolically* using positive Boolean formulas to encode the transition relation (e.g. [31,35,44]). In our presentation (as in [3,18]) we use an *explicit* encoding, where the transition relation is given as a function into a set of sets of states. Although the representations can be easily transformed into one another, generally there is an exponential blow up when going from the symbolic to the explicit setting.

For containment in PSPACE the difference between the encodings is minor, and in both cases the argument is that a certain accepting lasso in the self-product of the corresponding t-GBA, which exists if and only if the automaton is not unambiguous, can be guessed in polynomial space. We first show that for every accepting run of $\mathcal{A}$, we find a matching accepting run of $\mathcal{G}_{\mathcal{A}}$.

Lemma 3.3 *For every accepting run $\rho = (V, E)$ of $\mathcal{A}$ for $w \in \Sigma^\omega$ there exists an accepting run $r = Q_0 Q_1 \dots$ of $\mathcal{G}_{\mathcal{A}}$ for w , such that $Q_i = V(i)$ for all $i \geq 0$.*

Proof We show that $r = V(0)V(1) \dots$ is an accepting run of $\mathcal{G}_{\mathcal{A}}$. As ρ is a run of $\mathcal{A}$ we get $V(0) \in \iota$ and hence r satisfies the initial condition of a t-GBA run.

We show that for all $i : V(i+1) \in \bigotimes_{q \in V(i)} \delta(q, w[i])$. As ρ is a run of $\mathcal{A}$, every $q \in V(i)$ must have successors $E(q, i) \subseteq V(i+1)$ such that $E(q, i) \in \delta(q, w[i])$. Furthermore, each $q' \in V(i+1)$ must have a predecessor in $V(i)$ which implies that $\bigcup_{q \in V(i)} E(q, i) = V(i+1)$. By definition of $\bigotimes$ we get $V(i+1) \in \bigotimes_{q \in V(i)} \delta(q, w[i])$ and thus $(V(i), w[i], V(i+1))$ is a transition of $\mathcal{G}_{\mathcal{A}}$.

To see that r is accepting assume, for contradiction, that there exists $q_f \in Q_f$ and $k \in \mathbb{N}$ such that for all $n \geq k : (V(n), w[n], V(n+1)) \notin \mathcal{T}_{q_f}$. Then, by the definition of $\mathcal{T}_{q_f}$ it follows that for all $n > k : q_f \in V(n)$ and there is no $Y \in \delta(q_f, w[n])$ such that $q_f \notin Y$ and $Y \subseteq V(n+1)$. This implies, however, that infinitely often q_f is in the set of successors of q_f . But then ρ is not accepting, which contradicts the assumption. $\square$

Lemma 3.4 *$\mathcal{A}$ is unambiguous if and only if $\mathcal{G}_{\mathcal{A}}$ is unambiguous.*

Proof The direction from left to right follows by Corollary 3.1

To show the other direction, assume that $\mathcal{A}$ is ambiguous. Then, there exists a word $w \in \Sigma^\omega$ and two accepting runs $\rho_1 = (V_1, E_1)$, $\rho_2 = (V_2, E_2)$ of $\mathcal{A}$ for w such that for some $i \in \mathbb{N} : V_1(i) \neq V_2(i)$. By Lemma 3.3, there exist two accepting runs r_1, r_2 of $\mathcal{G}_{\mathcal{A}}$ for w such that $r_1(i) \neq r_2(i)$, and hence $\mathcal{G}_{\mathcal{A}}$ is ambiguous. $\square$

Proposition 3.1 *Unambiguity of explicit VWAA can be decided in PSPACE.*

Proof By the theorem of Savitch, we know that NPSPACE=PSPACE. We give an NPSPACE (in the size of $\mathcal{A}$) algorithm for checking whether $\mathcal{G}_{\mathcal{A}}$ is unambiguous, which is enough by Lemma 3.4.

The algorithm guesses an accepting lasso in the self product of $\mathcal{G}_{\mathcal{A}}$: a path that reaches a state (C_1, C_2) such that $C_1 \neq C_2$ and such that there exists an accepting loop starting in (C'_1, C'_2) reachable from (C_1, C_2) . The length of the prefix and recurring parts of such a lasso can be assumed to be at most $|Q| \cdot 2^{2|Q|}$. To guess a successor of a state (Q_1, Q_2) in $\mathcal{G}_{\mathcal{A}} \otimes \mathcal{G}_{\mathcal{A}}$ for a symbol $a \in \Sigma$ the algorithm first guesses a successor configuration of Q_1 and a in $\mathcal{A}$. This can be done by guessing a set $S_q \in \delta(q, a)$ for each $q \in Q_1$ and then taking $Q'_1 = \bigcup_{q \in Q_1} S_q$ as successor configuration of Q_1 . The same is done to guess a successor

configuration Q'_2 of Q_2 . By the definition of $\mathcal{G}_A$ (Definition 2.2), (Q'_1, Q'_2) is an α -successor of (Q_1, Q_2) in $\mathcal{G}_A \otimes \mathcal{G}_A$. While guessing the loop the algorithm additionally remembers which of the acceptance sets have already been visited. Deciding in which acceptance sets $\mathcal{T}_f$ the transition $Q_1 \xrightarrow{a} Q'_1$ belongs to can be done in polynomial time.

The algorithm first guesses $p, r \leq |Q| \cdot 2^{2|Q|}$ (using $O(\log(|Q|) + |Q|)$ bits each), and then guesses a lasso with prefix of length p and recurring part of length r . It returns “yes” if the path is indeed a lasso as above and all acceptance sets have been visited on the loop, and otherwise returns “no”.

The algorithm needs to remember before the loop that it has visited a state (C_1, C_2) with $C_1 \neq C_2$, at the beginning of the loop the state (C'_1, C'_2) , the current state in $\mathcal{G}_A \otimes \mathcal{G}_A$ and, for the loop, which of the acceptance sets in $\mathcal{T}$ have already been satisfied. Hence, its space requirements are polynomial in $|A|$. $\square$

The proof of PSPACE membership of unambiguity checking for symbolic VWAA is analogous, using the additional observation that a minimal model of the transition formula can be guessed in PSPACE. To keep the notations consistent we have decided to spell out the proof for the explicit version of VWAA. To show PSPACE-hardness, we reduce the problem of deciding VWAA emptiness to the problem of deciding VWAA unambiguity (Lemma 3.5). For the reduction, we define the union of two VWAA.

Definition 3.2 Let $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, \iota_1, \text{Fin}(F_1))$ and $\mathcal{A}_2 = (Q_2, \Sigma, \Delta_2, \iota_2, \text{Fin}(F_2))$ be two VWAA. The union automaton $\mathcal{A}_1 \cup \mathcal{A}_2 = (Q, \Sigma, \Delta, \iota, \text{Fin}(F))$ is defined as follows:

- $Q = (Q_1 \times \{1\}) \cup (Q_2 \times \{2\})$
- $\Delta((q, i), a) = \{S \times \{i\} : S \in \Delta_i(q, a)\}$
- $\iota = \{I \times \{i\} : I \in \iota_i, i \in \{1, 2\}\}$
- $F = (F_1 \times \{1\}) \cup (F_2 \times \{2\})$

The union automaton satisfies $\mathcal{L}(\mathcal{A}_1 \cup \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

Lemma 3.5 *There is a polynomial reduction from the VWAA emptiness problem to the VWAA unambiguity problem.*

Proof Let $\mathcal{A}$ be a VWAA. An accepting run of $\mathcal{A}$ corresponds to two distinct accepting runs of $\mathcal{A} \cup \mathcal{A}$ and thus we have:

$$\mathcal{L}(\mathcal{A}) = \emptyset \text{ iff } \mathcal{A} \cup \mathcal{A} \text{ is unambiguous.}$$

For the symbolic representation this directly yields PSPACE-hardness, as the standard translation from LTL to symbolic VWAA is linear [34,44], and the satisfiability problem of LTL is PSPACE-hard [4, Theorem 5.48]. However, the size of the explicit encoding of VWAA may not be polynomial in the size of the LTL formula. Hence we cannot directly use the translation from LTL to show PSPACE-hardness in the same way. We now give a proof of PSPACE-hardness of the emptiness problem of explicitly encoded VWAA, which is an interesting result independent of our main goal to show that checking unambiguity is PSPACE-hard.

A reduction from SAT First we show that checking non-emptiness of explicit VWAA is NP-hard by encoding 3-SAT. The construction will be used later to encode QBF. For Boolean formulas φ over variables $\mathcal{V}$, we will write $\mathcal{V}' \models \varphi$ (with $\mathcal{V}' \subseteq \mathcal{V}$) to mean that the interpretation which assigns exactly the variables in $\mathcal{V}'$ to true is a model of φ . Let $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_n$ be a propositional formula in 3-CNF over the variables $\{z_1, \dots, z_m\}$

Fig. 5 Sketch of $\text{VWAA } \mathcal{A}^{\text{SAT}}(\varphi)$ for checking satisfiability of $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_n$. All transitions between states C_i and $(\neg)z_j^1$ are labeled by the symbol $\#_s$. The constructions of the chains between states $(\neg)z_j^1$ and E is shown in Fig. 6

Fig. 6 The chain $\mathbf{z}_j$ for the positive literal z_j requires that the j th symbol is a 1. Each dashed edge in Fig. 5 represents the corresponding chain

and denote by $\text{lit}(C_j)$ the set containing the three literals of C_j , in particular we have $\text{lit}(C_j) \subseteq \{z_k, \neg z_k : 1 \leq k \leq m\}$.

The main idea is to first universally select all clauses, and then letting each clause C_j choose one of its literals non-deterministically. Consistency of these choices is checked by enforcing for a literal $(\neg)z_j$ that the symbol of the j th position in the suffix word is 1 (resp. 0).

We construct a $\text{VWAA } \mathcal{A}^{\text{SAT}}(\varphi) = (Q^{\text{SAT}}, \Sigma, \delta^{\text{SAT}}, \iota^{\text{SAT}}, \text{true})$ over the alphabet $\Sigma = \{0, 1, \#_i, \#_s\}$ as sketched in Fig. 5. The acceptance condition true implies that all words that have any run are accepted. It has states $Q^{\text{SAT}} = \{I, C_1, \dots, C_n, E\} \cup \{z_j^k, \neg z_j^k : 1 \leq k, j \leq m\}$, where I is the single initial state (that is, we define $\iota^{\text{SAT}} = \{I\}$). The transition function of initial and clause states is defined as follows:

$$\delta^{\text{SAT}}(I, \#_i) = \{C_1, \dots, C_n\} \quad \delta^{\text{SAT}}(C_j, \#_s) = \{l^1 : l \in \text{lit}(C_j)\}$$

For all other symbols, these states have no transitions. Each state $(\neg)z_j^1$ is the beginning of a chain of m states that enforces the j th symbol of the following word to be 1 (resp. 0). Hence, states $(\neg)z_j^k$ have a transition to $(\neg)z_j^{k+1}$ (or to E , if $k = m$) for both symbols 0 and 1, *except if $j = k$* . The state z_j^j has only the transition labeled by 1, and $\neg z_j^j$ has only the transition labeled by 0. So, for example, $\delta^{\text{SAT}}(z_2^1, 0) = \delta^{\text{SAT}}(z_2^1, 1) = \{z_2^2\}$, but $\delta^{\text{SAT}}(z_2^2, 0) = \emptyset$, whereas $\delta^{\text{SAT}}(z_2^2, 1) = \{z_2^3\}$. Figure 6 shows how these chains are constructed. Finally, we have $\delta^{\text{SAT}}(E, \#_i) = \{\emptyset\}$. This means that state E accepts all words starting with $\#_i$, and this is pictured by an arrow with no successor states in Fig. 5. Symbols $\#_i$ and $\#_s$ play no immediate role for the NP-hardness proof, but will be used later for coordination purposes in the PSPACE-hardness proof.

Proposition 3.2

$$\mathcal{L}(\mathcal{A}^{\text{SAT}}(\varphi)) = \{\#_i \#_s b_1 \dots b_m \#_i : \{z_i : b_i = 1\} \models \varphi\} \Sigma^\omega$$

Proof “ $\subseteq$ ”: Take a word $w \in \mathcal{L}(\mathcal{A}^{\text{SAT}}(\varphi))$ and let $\rho = (V, E)$ be an accepting run of $\mathcal{A}^{\text{SAT}}(\varphi)$ for w . By construction of $\mathcal{A}^{\text{SAT}}(\varphi)$, $w = \#_i \#_s b_1 \dots b_m \#_i \Sigma^\omega$, for some $b_1 \dots b_m \in \{0, 1\}^m$. We have $V(1) = \{C_1, \dots, C_n\}$ and $V(2)$ corresponds to a set of literals such that for each $j \in \{1, \dots, n\}$: there exists $l \in \text{lit}(C_j)$ such that $l^1 \in V(2)$. By construction of the chain of states $(\neg)z_j^1 \dots (\neg)z_j^m$ we get that if $z_j^1 \in V(2)$, then $b_j = 1$, and analogously for $\neg z_j^1$. This implies that $V(2)$ contains no pair of contradicting literals. Hence, $\{z_i : b_i = 1\}$ is a model of φ .

“ $\supseteq$ ”: Let $w = \#_i \#_s b_1 \dots b_m \#_i w'$ such that $\{z_i : b_i = 1\} \models \varphi$. We construct an accepting run of $\mathcal{A}^{\text{SAT}}(\varphi)$ for w by letting the state C_j move to some state l^1 such that $l \in \text{lit}(C_j)$ and $\{z_i : b_i = 1\} \models l$. Such a literal l must exist for each C_j , as $\{z_i : b_i = 1\} \models \varphi$. By construction, the sequence $b_0 \dots b_m$ is accepted from all these states l^1 . This implies that an accepting run can be constructed for w . $\square$

Corollary 3.2 $\mathcal{L}(\mathcal{A}^{\text{SAT}}(\varphi)) \neq \emptyset$ if and only if φ is satisfiable.

The size of Q^{SAT} is $n+2m^2+2$, where the $2m^2$ corresponds to the states $\{z_j^k, \neg z_j^k : 1 \leq k, j \leq m\}$. The number of transition sets of each state is bounded by $|Q^{\text{SAT}}|$, and hence the total size of the automaton is polynomial in φ .

Proposition 3.3 The non-emptiness problem of explicit VWAA without loops is NP-complete.

Proof NP-hardness follows from Corollary 3.2 and it remains to show that the problem is also in NP. The length of runs of a VWAA $\mathcal{A}$ without loops is bounded by the maximal path-length of the automaton. This implies that the size of its runs is bounded by $|\mathcal{A}|^2$, as the width of its runs is also at most linear in $|\mathcal{A}|$. So we can non-deterministically guess a run of the automaton and verify in polynomial time that it is accepting. $\square$

Remark 3.1 Several papers have studied the complexity of fragments of LTL [1,6,37,39,40]. In [40] it was shown that satisfiability for LTL($\diamond$) is NP-complete, while it is PSPACE-complete already for LTL($\diamond, \square, \bigcirc$). NP-hardness for most standard fragments is a direct consequence of the fact that they include propositional logic.

The transition structure of explicit VWAA however only allows one, informally, to encode Boolean formulas in *disjunctive normal form* directly. This kind of restriction has, to the best of our knowledge, not been considered before from a complexity point of view. Under it, NP-hardness does not follow directly from NP-hardness of 3-SAT. The argument that non-emptiness of VWAA with no loops is in NP is similar as for satisfiability in the fragment LTL($\bigcirc$) (see [39]).

PSPACE-hardness Now we reduce QBF-truth to explicit VWAA emptiness. The main idea is to enumerate all valuations of universally quantified variables and at the same time checking whether for each of these a consistent valuation of the existential variables can be found that satisfy the (quantifier-free) matrix of the QBF-formula. To check the second condition we use the automaton $\mathcal{A}^{\text{SAT}}$. Now we show how to implement the consistent enumeration of valuations using a polynomial-size very weak alternating automaton.

We let the $+1$ operation on sequences of the form $\{0, 1\}^m$ be the standard incrementation of fixed size binary numbers with overflow, in particular $1^m + 1 = 0^m$. The encoding is such that the the first position contains the most significant bit, and the m th position

contains the least significant bit. Given $X \in \{0, 1\}^m$, we will denote the k th position of X by $X(k)$. The following lemma relates QBF-truth to the existence of a word satisfying certain conditions, and we will use it to reduce the problem to VWAA-emptiness. Observe that it is enough to consider QBF-formulas with alternating quantifiers as we can always add “dummy” quantifiers. For example: $\forall x \forall y. x \vee y \equiv \forall x \exists z_1 \forall y \exists z_2. x \vee y$. To emphasize that a given Boolean formula f contains only free variables from $\{x_1, \dots, x_n\}$ we write $f(x_1, \dots, x_n)$, and as before, we write $\mathcal{V}' \models f(x_1, \dots, x_n)$ if the interpretation that assigns exactly the variables in $\mathcal{V}'$ to true is a model of $f(x_1, \dots, x_n)$.

Lemma 3.6 *Let $\theta = \forall x_1 \exists y_1 \dots \forall x_m \exists y_m \varphi$, where φ is a 3-CNF formula. Then, θ is valid if and only if there exists a sequence $X_1 Y_1 \dots X_{2^m} Y_{2^m}$, with $X_i, Y_i \in \{0, 1\}^m$ for all $i \leq 2^m$, such that:*

- (1) $X_1 = 0^m$ and $X_{i+1} = X_i + 1$ for all $i < 2^m$
- (2) $Y_i(k) \neq Y_{i+1}(k) \implies X_i(k) \neq X_{i+1}(k)$ for all $i < 2^m, k \leq m$
- (3) $\{x_k : X_i(k) = 1, k \leq m\} \cup \{y_k : Y_i(k) = 1, k \leq m\} \models \varphi$ for all $i \leq 2^m$

Proof We first observe that θ is valid if and only if there exist Boolean formulas $f_j(x_1, \dots, x_j)$ for each $1 \leq j \leq m$ such that

$$\forall x_1 \forall x_2 \dots \forall x_m \varphi[y_1/f_1(x_1), y_2/f_2(x_1, x_2), \dots, y_m/f_m(x_1, \dots, x_m)] \text{ is valid} \quad (*)$$

The intuition is that f_j determines the value of variable y_j , given the valuation of the preceding universally quantified variables (see [25, Lemma 1]).

“ $\implies$ ”: If θ is valid, there exist Boolean formulas $f_j(x_1, \dots, x_j)$ satisfying (*). Using these formulas, we construct a sequence $X_1 Y_1 \dots X_{2^m} Y_{2^m}$ satisfying (1-3). We define $X_1 X_2 \dots X_{2^m}$ to be the sequence that starts with 0^m and is increased by one in each step. This satisfies (1). Now, the j th entry of Y_i is defined as follows:

$$Y_i(j) = \begin{cases} 1 & \text{if } \{x_k : k \leq j, X_i(k) = 1\} \models f_j(x_1, \dots, x_j) \\ 0 & \text{otherwise} \end{cases}$$

It depends on f_j and the first j entries of X_i . To show (2), we note that if $Y_i(k) \neq Y_{i+1}(k)$ holds, then $X_i(k') \neq X_{i+1}(k')$ must hold for some $k' \leq k$, as the k th position of Y_i (resp. Y_{i+1}) depends only on these values. But if $X_i(k') \neq X_{i+1}(k')$ for some $k' \leq k$, then already $X_i(k) \neq X_{i+1}(k)$ by the fact that $X_{i+1} = X_i + 1$ (in a binary counter, the value of a higher bit changes only if all values of lower bits change). For (3), assume that $\{x_k : X_i(k) = 1, k \leq m\} \cup \{y_k : Y_i(k) = 1, k \leq m\} \not\models \varphi$ holds for some $i \leq 2^m$. By definition of Y_i , it follows that

$$\{x_k : X_i(k) = 1, k \leq m\} \not\models \varphi[y_1/f_1(x_1), y_2/f_2(x_1, x_2), \dots, y_m/f_m(x_1, \dots, x_m)]$$

This contradicts the fact that the formulas f_j satisfy (*).

“ $\Leftarrow$ ”: Let $X_1 Y_1 \dots X_{2^m} Y_{2^m}$ be a sequence satisfying (1–3). We show that Boolean formulas $f_j(x_1, \dots, x_j)$ exist (for $1 \leq j \leq m$) which satisfy (*). To this end, we first show that formulas f_j exist satisfying:

$$\text{for all } i \leq 2^m, j \leq m : Y_i(j) = 1 \iff (\{x_k : k \leq j, X_i(k) = 1\} \models f_j(x_1, \dots, x_j))$$

Such f_j exist if and only if there are no $i, i' \leq 2^m$ such that $Y_i(j) \neq Y_{i'}(j)$ but $X_i(k) = X_{i'}(k)$ for all $k \leq j$. But such i, i' cannot exist, as the positions in which the first j (most significant) bits of X remain unchanged form a consecutive subsequence of $X_1 \dots X_{2^m}$, and by (2),

(a) $\mathcal{C}(j : \neg \text{switch})$. Checks that one of the positions $j+1, j+2, \dots, m$ has symbol 1. Runs for words that do not satisfy this condition end up in the rejecting trap state at the bottom-right.

(b) $\mathcal{C}(j : \text{switch})$. Checks that all of positions $j+1, j+2, \dots, m$ have symbol 0.

(c) $\mathcal{C}^X(j : *)$, with $* \in \{0, 1\}$ ($\mathcal{C}^Y(j : *)$ is defined similarly). This gadget checks that the j -th (resp. $m+j$ -th for Y) symbol of the suffix is $*$. Furthermore, the word should be in $\{0, 1\}^{2m} \#_i \#_s$. The transition labeled by $\#_i \#_s$ represents two transitions with another state in between.

Fig. 7 The check-gadgets accept words starting with a sequence in $\{0, 1\}^{2m}$ that satisfy certain additional conditions

$Y_i(j) \neq Y_{i+1}(j)$ implies $X_i(j) \neq X_{i+1}(j)$. Now to show that $(*)$ holds we use that the sequence $X_1, \dots, X_{2^m}$ enumerates all sequences in $\{0, 1\}^m$ by (1), and for all these we have:

$$\{x_k : X_i(k) = 1, k \leq m\} \models \varphi[y_1/f_1(x_1), y_2/f_2(x_1, x_2), \dots, y_m/f_m(x_1, \dots, x_m)]$$

by (3) and the above property of formulas f_j . $\square$

Condition (3) in the above lemma requires to check that the variable interpretation induced by $X_i Y_i$ satisfies φ for all i . For this, we will use a slight adaptation of the construction $\mathcal{A}^{\text{SAT}}(\varphi)$.

It remains to show how conditions (1) and (2) can be implemented in a VWAA. We construct the automaton $\mathcal{A}^{\text{count}} = (Q^{\text{count}}, \Sigma, \delta^{\text{count}}, \iota^{\text{count}}, \text{true})$ that accepts words of the form $\#_i \#_s X_1 Y_1 \#_i \#_s X_2 Y_2 \dots \in (\#_i \#_s \{0, 1\}^{2m})^\omega$, such that the sequence $X_1 Y_1 X_2 Y_2 \dots X_{2^m} Y_{2^m}$ satisfies these conditions. The main idea is the following. There are $2m$ control-gadgets, one for each position of the sequences $X_i Y_i$. In each round, each of them moves to one of two states, corresponding to the possible values $\{0, 1\}$ for the position in this round. Check-gadgets are used to make sure that the choices are consistent with the conditions (1–2). For example, position j of sequence X should change its value with respect to the previous round if and only if all positions $j+1$ through m of X change their value from 1 to 0 in this round. In particular, the m th position of sequence X should change its value in every round. This will make sure that the sequence $X_1 X_2 \dots$ simulates a binary counter (condition (1)). Similarly, position j of Y should only change its value if the position j of X changes its value (condition (2)).

Check-gadgets We will use the following check-gadgets (for $0 \leq j \leq m$):

- $C^X(j : *)$ checks that the j th position is $*$, for $*$ $\in \{0, 1\}$,
- $C^Y(j : *)$ checks that the $m+j$ th position is $*$, for $*$ $\in \{0, 1\}$,
- $C(j : \text{switch})$ requires positions $j+1$ through m to be 0 (defined for $j < m$) and
- $C(j : \neg \text{switch})$ requires one of positions $j+1$ to m to be 1 (defined for $j < m$).

Gadget $C(m : \text{switch})$ is constructed to accept all words, while $C(m : \neg \text{switch})$ accepts the empty language. This makes sure that the last bit of sequences X_i always alternates. Additionally to the above conditions the check-gadgets enforce that the suffix word starts with a sequence in $\{0, 1\}^{2m} \#_s$. The symbol $\#_s$ is used to trigger the next round. Figure 7 shows how they are realized in VWAA with at most $2m+2$ states each.

Control-gadgets The control-gadgets $\mathcal{Q}_j^X$ and $\mathcal{Q}_j^Y$ determine the value of position j of sequence X_i (resp. Y_i) for all i . They use the check-gadgets to ensure that the choices of values are consistent with conditions (1) and (2) of Lemma 3.6. Each control-gadget $\mathcal{Q}_j^X$ has three states: the “main” state together with the two states $\text{pos}^X(j)$ and $\text{neg}^X(j)$ which are used to remember the last value of position j in sequences X (and analogously for Y). Figure 8 shows how these gadgets are constructed. Transitions moving to initial states of check gadgets are not drawn, but the conditions that a transition is dependent upon are indicated as labels on the edges. For example, $\#_s(X(j) : 1, \text{switch})$ means that the label of the transition is $\#_s$ and initial states of $C^X(j : 1)$ and $C^X(j : \text{switch})$ are included in the successor set, additionally to the drawn successors. The label $\#_s(\neg \text{switch})$ in $\mathcal{Q}_j^X$ means that only $C^X(j : \neg \text{switch})$ is included additionally. Including the check-gadgets in the universal successor set implies that the transition can only be taken if the suffix word is also accepted by the corresponding gadgets. In the following we let gadget names represent their initial states and write a set as second argument to the transition function if all symbols in the set have exactly the same transitions.

$$\begin{aligned}
 \delta^{\text{count}}(\mathcal{Q}_j^X, \Sigma \setminus \{\#_s\}) &= \{\{\mathcal{Q}_j^X\}\} \\
 \delta^{\text{count}}(\mathcal{Q}_j^X, \#_s) &= \{\{\mathcal{Q}_j^X, C(j : \neg \text{switch})\}, \\
 &\quad \{\mathcal{Q}_j^X, C(j : \text{switch}), C^X(j : 1), \text{pos}^X(j)\}, \\
 &\quad \{\mathcal{Q}_j^X, C(j : \text{switch}), C^X(j : 0), \text{neg}^X(j)\}\} \\
 \delta^{\text{count}}(\text{pos}^X(j), \Sigma \setminus \{\#_s\}) &= \{\{\text{pos}^X(j)\}\} \\
 \delta^{\text{count}}(\text{pos}^X(j), \#_s) &= \{\{\text{pos}^X(j), C(j : \neg \text{switch}), C^X(j : 1)\}, \\
 &\quad \{C(j : \text{switch}), C^X(j : 0)\}\} \\
 \delta^{\text{count}}(\text{pos}^Y(j), \Sigma \setminus \{\#_s\}) &= \{\{\text{pos}^Y(j)\}\} \\
 \delta^{\text{count}}(\text{pos}^Y(j), \#_s) &= \{\{\text{pos}^Y(j), C(j : \neg \text{switch}), C^Y(j : 1)\}, \\
 &\quad \{C(j : \text{switch})\}\}
 \end{aligned}$$

The transitions of states $\mathcal{Q}_j^Y$, $\text{neg}^X(j)$ and $\text{neg}^Y(j)$ are analogous and hence excluded here. The important properties of gadgets $\mathcal{Q}_j^X$ and $\mathcal{Q}_j^Y$ are:

- All $\#_s$ -transitions with $\text{pos}^X(j)$ in their successor set also include $C^X(j : 1)$ (and analogously for neg and 0, and the Y -gadget).
- The only non-looping transition of $\text{pos}^X(j)$ (resp. $\text{neg}^X(j)$) includes $C^X(j : 0)$ (resp. $C^X(j : 1)$) and $C(j : \text{switch})$ (in contrast to the Y -gadget).

Fig. 8 The gadgets Q_j^X and Q_j^Y control the value of the j th bit of sequences X_i/Y_i . The current value of the bit is stored in the states pos and neg . To make sure that the value is switched whenever all subsequent bits are zero, check gadgets are included in the successor sets of transitions labeled by $\#_s$, as indicated by the transition labels

The second item holds only for X -states and ensures that if the switch-condition holds for a suffix word, then state Q_j^X can only take the $\#_s$ -transition to the state in $\text{pos}^X(j)/\text{neg}^X(j)$ which was not part of the previous layer of the run-DAG. This implies that the value of position j in sequence X changes. In this situation the j th position of Y is allowed to change its value, but is not required to do so. The first item above shows that in no accepting run-DAG the states $\text{pos}^X(j)$ and $\text{neg}^X(j)$ can appear on the same layer, as this would imply that $C^X(j : 1)$ and $C^X(j : 0)$ would also appear in the same layer, but these two gadgets accept disjoint languages (the analogous statement holds for Y -states).

The state space $\mathcal{Q}^{\text{count}}$ of $\mathcal{A}^{\text{count}}$ is the union of the states of the gadgets, together with a state used for initialisation:

$$\{Q_j^X, Q_j^Y, \text{pos}^X(j), \text{neg}^X(j), \text{pos}^Y(j), \text{neg}^Y(j) : 1 \leq j \leq m\} \quad (\text{control-gadgets})$$

$$\bigcup \{ \text{states}(C^{X/Y}(j : *)) : 0 \leq j \leq m, * \in \{0, 1, (\neg) \text{ switch}\} \} \quad (\text{check-gadgets})$$

$$\{start\} \quad (\text{initialisation})$$

The state *start* forces the first symbols to be $\#_i \#_s$ and then moves to the initial state of $C(0 : \text{switch})$, which in turn forces the suffix word to begin with 0^m . That is, the transition function of *start* is: $\delta^{\text{count}}(start, \#_i \#_s) = \{C(0 : \text{switch})\}$. To enforce the sequence of two symbols we formally require another state.

The initial states of $\mathcal{A}^{\text{count}}$ are: $\iota^{\text{count}} = \{start\} \cup \{Q_j^X, Q_j^Y : 1 \leq j \leq m\}$. This represents the conjunction of all control-gadgets, together with the state *start*. Hence, words accepted by $\mathcal{A}^{\text{count}}$ need to be accepted by all the control-gadgets, which check precisely the conditions (1) and (2) of Lemma 3.6 for each bit:

Lemma 3.7 *I. If $w \in \mathcal{L}(\mathcal{A}^{\text{count}})$, then w is of the form*

$$w = \#_i \#_s X_1 Y_1 \#_i \#_s X_2 Y_2 \dots \in (\#_i \#_s \{0, 1\}^{2m})^\omega$$

and the sequence $X_1 Y_1 X_2 Y_2 \dots X_{2m} Y_{2m}$ satisfies conditions (1) and (2) of Lemma 3.6.

II. Conversely, if $X_1 Y_1 X_2 Y_2 \dots X_{2m} Y_{2m}$ satisfies conditions (1) and (2) of Lemma 3.6, then $(\#_i \#_s X_1 Y_1 \#_i \#_s X_2 Y_2 \dots X_{2m} Y_{2m})^\omega$ is accepted by $\mathcal{A}^{\text{count}}$.

Proof I. Let ρ be an accepting run of $\mathcal{A}^{\text{count}}$. We show that the layers of ρ are an infinite repetition of $2m+2$ -length sequences of the form: $I_1 \xrightarrow{\#_i} I_2 \xrightarrow{\#_s} R_1 \dots R_{2m}$. The arrows indicate that layers I_1 and I_2 will contain states that only allow the corresponding symbols. We will use superscripts to indicate which block a layer belongs to. For example, R_1^j is the layer R_1 in the j th block, with $1 \leq j$. We show the following properties of the blocks for all $1 \leq j$ by induction:

- (a) For all $1 \leq k \leq m$: R_1^j contains either $\text{pos}^X(k)$ or $\text{neg}^X(k)$, but not both, and analogously for Y -states.
- (b) For all $1 \leq k \leq m$: R_1^j contains $C^X(k : 1)$ iff R_1^j contains $\text{pos}^X(k)$ and R_1^j contains $C^X(k : 0)$ iff R_1^j contains $\text{neg}^X(k)$ (and analogously for Y -states).
- (c) For all $1 \leq k \leq m$: $R_1^{j+1} \cap \{\text{pos}^X(k), \text{neg}^X(k)\} \neq R_1^j \cap \{\text{pos}^X(k), \text{neg}^X(k)\}$ iff for all $k < k' \leq m$: $\text{neg}^X(k') \in R_1^{j+1}$. Furthermore, $R_1^{j+1} \cap \{\text{pos}^Y(k), \text{neg}^Y(k)\} \neq R_1^j \cap \{\text{pos}^Y(k), \text{neg}^Y(k)\}$ implies for all $k < k' \leq m$: $\text{neg}^X(k') \in R_1^{j+1}$.

First, we observe that all states Q_k^X and Q_k^Y are present in every layer of every run, as these have only looping transitions and are included in the single initial set.

(a) For $j = 1$, observe that *start* forces the word to begin with $\#_i \#_s 0^m$. The only $\#_s$ -transition of Q_k^X that does not lead to a state in $\{\text{pos}^X(k), \text{neg}^X(k)\}$ includes the initial state of $C(k : \neg \text{switch})$. But this gadget requires one of the first m positions of the suffix to be 1, which is not the case. We show that all subsequent layers of the run contain exactly one of states $\{\text{pos}^X(k), \text{neg}^X(k)\}$ for all k . The reason is that all non-looping transitions of these states are labeled by $\#_s$ and contain the check-gadget $C^X(k : \text{switch})$ in their successor set. Whenever this transition is taken in a valid run, state Q_k^X must take a transition that again includes some state in $\{\text{pos}^X(k), \text{neg}^X(k)\}$, as the alternative includes the check-gadget $C^X(k : \neg \text{switch})$, which cannot appear on the same layer as $C^X(k : \text{switch})$. The analogous reasoning holds for the Y -states.

(b) First, observe that all $\#_s$ -transitions which include $\text{pos}^X(k)$ as successor also include $C^X(k : 1)$, and analogously for $\text{neg}^X(k)$. It follows that (b) holds for $j = 1$, as by (a) these states are included in the third layer of the run. Furthermore, by induction hypothesis, the $2 + (2m+2) \cdot k$ th (for $k > 0$) transition must be labeled by $\#_s$ as gadgets $C^X(k : 1)$ and $C^X(k : 0)$ force the $2m+2$ th following symbol to be $\#_s$. Again, the reasoning for Y -states is analogous.

(c) Intuitively, the statement is that the value chosen for position k in either sequence X or Y may only change from one block to the next if all subsequent positions k' are 0 in the X -block. For X -sequences, the other direction should hold as well. Observe that all $\#_s$ -transitions of states $\text{neg}^{X/Y}(k)$ and $\text{pos}^{X/Y}(k)$ that are not looping include the gadget $C(k : \text{switch})$, which checks exactly that positions $k+1, \dots, m$ in the suffix word are 0. By (b), this implies that for $k < k' \leq m$ we have $\text{neg}^X(k') \in R_1^{j+1}$. To see that the other direction holds for X , observe that the non-looping $\#_s$ -transitions of $\text{pos}^X(k)$ and $\text{neg}^X(k)$ force state Q_k^X to move the other state by including gadgets $C^X(k : 0)$ or $C^X(k : 1)$, respectively.

It remains to argue that the fact that all runs have the above structure implies that the corresponding sequences $X_1 Y_1 \dots X_{2^m} Y_{2^m}$ satisfy conditions (1) and (2) of Lemma 3.6. We have already seen that any accepted word begins with $\#_i \#_s 0^m$. By (a) and (b) we have $X_j(k) = 1$ iff $\text{pos}^X(k) \in R_1^j$, and analogously for Y -states. By (c), $X_j(k) \neq X_{j+1}(k)$ iff $X_{j+1}(k+1) \dots X_{j+1}(m) = 0 \dots 0$ for all j , which characterizes a binary counter. By the second part of (c), it follows that $Y_j(k) \neq Y_{j+1}(k)$ implies $X_j(k) \neq X_{j+1}(k)$. The fact that each sequence $X_i Y_i$ is followed by $\#_i \#_s$ is guaranteed by the fact that one of $C^X(k : 0)$ and $C^X(k : 1)$ is included in R_1^j for $1 \leq j$ by (b). These states guarantee that the $2m+1$ and $2m+2$ symbols are $\#_i \#_s$ by construction.

II. Suppose that $X_1 Y_1 \dots X_{2^m} Y_{2^m}$ satisfies conditions (1) and (2) of Lemma 3.6. Then an accepting run for $(\#_i \#_s X_1 Y_1 \dots \#_i \#_s X_{2^m} Y_{2^m})^\omega$, which is structured as above, can be constructed. Essentially, states in the control-gadgets loop for all symbols except $\#_s$, for which they decide on whether to move to $\text{pos}^{X/Y}(k)$ or $\text{neg}^{X/Y}(k)$ depending on the value of $X_j(k)/Y_j(k)$ and whether $X_j(k+1) \dots X_j(m) = 0 \dots 0$ or not. Using conditions (1) and (2) it can be shown a transition is always possible that satisfies the constraints of the included check-gadgets. $\square$

Adapting $\mathcal{A}^{\text{SAT}}$. We now adapt $\mathcal{A}^{\text{SAT}}(\varphi)$ (Fig. 5) such that it checks a sequence of interpretations against the formula φ , and not just a single one. This is achieved by adapting the transitions of state I such that it loops for all symbols:

$$\delta_r^{\text{SAT}}(I, \Sigma \setminus \{\#_i\}) = \{\{I\}\} \quad \text{and} \quad \delta_r^{\text{SAT}}(I, \#_i) = \{\{I, C_1, \dots, C_n\}\}$$

For all other states, δ_r^{SAT} is defined as δ^{SAT} . We let $\mathcal{A}_r^{\text{SAT}}(\varphi)$ be defined as $\mathcal{A}^{\text{SAT}}(\varphi)$, but using δ_r^{SAT} as transition function. The construction of $\mathcal{A}_r^{\text{SAT}}(\varphi)$ implicitly uses a specific order of the variables $z_1, \dots, z_k$, which determines the order in which their values are enumerated in the accepted words. The variables used in φ are $x_1, \dots, x_m, y_1, \dots, y_m$ and we construct $\mathcal{A}_r^{\text{SAT}}(\varphi)$ using this order. Then we get:

Lemma 3.8 *I. If $\mathcal{A}_r^{\text{SAT}}(\varphi)$ accepts a word $w \in (\#_i \#_s \{0, 1\}^{2m})^\omega$, then the corresponding sequence $X_1 Y_1 \dots X_{2^m} Y_{2^m}$ satisfies condition (3) of Lemma 3.6.*

II. If $X_1 Y_1 X_2 Y_2 \dots X_{2^m} Y_{2^m}$ satisfies condition (3) of Lemma 3.6, then $\mathcal{A}_r^{\text{SAT}}(\varphi)$ accepts $(\#_i \#_s X_1 Y_1 \dots X_{2^m} Y_{2^m})^\omega$.

Proof I. As all transitions of state I are looping, it is present in every layer of any run and it follows by a direct extension of Proposition 3.2 that whenever symbols $\#_i \#_s$ appear in an accepted word, then the following sequence in $\{0, 1\}^{2m}$ induces a variable interpretation satisfying φ .

II. For all j , the sequence $X_j Y_j$ fully specifies an interpretation of variables in φ . We let $\mathcal{I}(j)$ be the set of literals interpreted by true in this interpretation. Consider the following sequence of layers:

$$\{I\} \rightarrow \{I, C_1, \dots, C_n\} \rightarrow \{I, l^1 : l \in \mathcal{I}(1)\} \rightarrow \dots \rightarrow \{I, l^{2^m} : l \in \mathcal{I}(1)\} \rightarrow \{I, E\}$$

We claim that extending this forever by using interpretations $\mathcal{I}(j)$, for $j \geq 0$, yields an accepting run of $\mathcal{A}_r^{\text{SAT}}(\varphi)$ for $(\#_i \#_s X_1 Y_1 \dots X_{2^m} Y_{2^m})^\omega$. This is because for all j the literals in $\mathcal{I}(j)$ are non-contradictory and every clause C_k has a successor in $\{l^1 : l \in \mathcal{I}(j)\}$, as $\mathcal{I}(j)$ is a satisfying interpretation of φ by assumption. $\square$

Final construction We have seen how to construct automata that check conditions (1) and (2) of Lemma 3.6 ($\mathcal{A}^{\text{count}}$) and condition (3) ($\mathcal{A}_r^{\text{SAT}}$). The automaton $\mathcal{A}^{\text{QBF}}(\theta)$, for $\theta = \forall x_1 \exists y_1 \dots \forall x_m \exists y_m \varphi$ is constructed by taking the disjoint union of states and transitions of $\mathcal{A}^{\text{count}}$ and $\mathcal{A}_r^{\text{SAT}}(\varphi)$, with the single initial set $\{\{I, \text{start}\} \cup \{Q_j^X, Q_j^Y : 1 \leq j \leq m\}\}$, the union of the two single initial sets of $\mathcal{A}_r^{\text{SAT}}(\varphi)$ and $\mathcal{A}^{\text{count}}$. As for the other automata, the acceptance condition is true, which means that any valid run is also accepting. By taking the union of the single initial sets, the words accepted by $\mathcal{A}^{\text{QBF}}(\theta)$ are exactly those that are accepted by both $\mathcal{A}^{\text{count}}$ and $\mathcal{A}_r^{\text{SAT}}(\varphi)$. Together, Lemmas 3.6, 3.7 and 3.8 let us conclude:

Lemma 3.9 *The formula $\theta = \forall x_1 \exists y_1 \dots \forall x_m \exists y_m \varphi$ is valid if and only if $\mathcal{L}(\mathcal{A}^{\text{QBF}}(\theta))$ is not empty.*

Proof “ $\implies$ ”: By Lemma 3.6 there exists a sequence $X_1 Y_1 \dots X_{2^m} Y_{2^m}$ satisfying conditions (1-3) of the lemma. It follows from Lemmas 3.7 and 3.8 that the sequence $(\#_i \#_s X_1 Y_1 \dots \#_i \#_s X_{2^m} Y_{2^m})^\omega$ is accepted by both $\mathcal{A}^{\text{count}}$ and $\mathcal{A}_r^{\text{SAT}}(\varphi)$, and hence also by $\mathcal{A}^{\text{QBF}}(\theta)$.

“ $\impliedby$ ”: By Lemma 3.7 there exists a word $w \in (\#_i \#_s \{0, 1\})^\omega \in \mathcal{L}(\mathcal{A}^{\text{QBF}}(\theta))$ such that the corresponding sequence $X_1 Y_1 \dots X_{2^m} Y_{2^m}$ satisfies conditions (1) and (2) of Lemma 3.6. As $w \in \mathcal{L}(\mathcal{A}_r^{\text{SAT}}(\varphi))$ holds, the sequence also satisfies condition (3) by Lemma 3.8. The claim follows from Lemma 3.6. $\square$

It remains to argue that the size of $\mathcal{A}^{\text{QBF}}(\theta)$ is polynomial in the size of θ , and that it is very weak.

Lemma 3.10 *$\mathcal{A}^{\text{QBF}}(\theta)$ is very weak and its size is polynomial in the size of θ .*

Proof By construction of $\mathcal{A}^{\text{QBF}}(\theta)$, it is enough to show that automata $\mathcal{A}^{\text{SAT}}(\varphi)$ and $\mathcal{A}^{\text{count}}$ are both very weak, and of polynomial size. For $\mathcal{A}^{\text{SAT}}(\varphi)$ we have already shown this (the minor extension does not increase the automaton substantially). To see that $\mathcal{A}^{\text{count}}$ is very weak, we observe that the control-gadgets (Fig. 8) and the check-gadgets (Fig. 7) do not have any non-trivial loops. Furthermore, the only edges between gadgets go from control-gadgets to check-gadgets, but not back. To see that the size of $\mathcal{A}^{\text{count}}$ is polynomial in the size of θ , we first observe that it includes $2m$ control-gadgets with 3 states each, and $6m$ check-gadgets of at most $2m+2$ states each. Also, the number of transition sets in $\mathcal{A}^{\text{count}}$ is constant for each state and alphabet symbol. $\square$

The following theorem is a consequence of lemmas 3.9 and 3.10 :

Theorem 3.2 *The emptiness problem for explicit VWAA is PSPACE-hard.*

Lemma 3.1, Theorem 3.2 and Lemma 3.5 give us PSPACE-completeness of checking unambiguity for explicitly encoded VWAA. PSPACE-hardness for symbolical VWAA follows from this, but can also be shown directly by using the standard translation from LTL to symbolic VWAA, which is linear [34,44], and the fact that LTL satisfiability is PSPACE-complete. Containment in PSPACE for symbolic VWAA follows essentially by the same proof as in Lemma 3.1.

Theorem 3.3 *Deciding unambiguity for explicit and symbolic VWAA is PSPACE-complete.*

4 Disambiguating VWAA

Our disambiguation procedure is inspired by the idea of “separating” the language of successors for every non-deterministic branching. A disjunction $\varphi \vee \psi$ is transformed into $\varphi \vee (\neg\varphi \wedge \psi)$ by this principle. The rules for $\mathcal{U}$ and $\mathcal{R}$ are derived by applying the disjunction rule to the expansion law of the corresponding operator:

	Expansion law	Adapted expansion law
$\varphi \mathcal{U} \psi$	$\Gamma \equiv \psi \vee (\varphi \wedge \bigcirc \Gamma)$	$\Gamma \equiv \psi \vee (\varphi \wedge \neg\psi \wedge \bigcirc \Gamma)$
$\varphi \mathcal{R} \psi$	$\Gamma \equiv \psi \wedge (\varphi \vee \bigcirc \Gamma)$	$\Gamma \equiv \psi \wedge (\varphi \vee (\neg\varphi \wedge \bigcirc \Gamma))$

These rules are applied by `ltl2tgba` in its tableau-based algorithm to guarantee that the resulting automaton is unambiguous, and have also been proposed in [7].

In our approach we define corresponding transformations for non-deterministic branching in the VWAA. Furthermore, we propose to do this in an “on-demand” manner: instead of applying these transformation rules to every non-deterministic split, we identify ambiguous states during the translation and only apply the transformations to them. This guarantees that we return the automaton produced by the core translation, without disambiguation, in case it is already unambiguous.

The main steps of our disambiguation procedure are the following:

1. A preprocessing step that computes a complement state $\tilde{s}$ for every state s .
2. A procedure that identifies ambiguous states.
3. Local transformations that remove the ambiguity.

Fig. 9 Example of an automaton together with its dual. The $\sim$ -states accept the complement language of the corresponding states on the left. Dualization essentially replaces nondeterministic branching by universal branching, and vice versa, for each state and symbol. The acceptance status of each state is swapped

If no ambiguity is found in step 2, the VWAA is unambiguous. The high-level overview is also depicted in Fig. 1. In what follows we fix a VWAA $\mathcal{A} = (Q, \Sigma, \delta, s_0, \text{Fin}(Q_F))$ which we assume to have a single initial state s_0 .

Complement states The transformations we apply for disambiguation rely on the following precondition: for every state s there exists a state $\tilde{s}$ such that $\mathcal{L}(\tilde{s}) = \overline{\mathcal{L}(s)}$. We will assume that applying the $\sim$ operator twice to a state s again yields s .

The complement states are computed in a preprocessing step and added to the VWAA. Complementing alternating automata can be done by dualizing both the acceptance condition and transition structure, as shown by Muller and Schupp [35]. An example of this transformation is given in Fig. 9. While there is no blow-up in the symbolic representation of alternating automata, it may introduce an exponential number of transitions in the explicit representation. As dualizing the acceptance condition and complementing the set of final states yields an equivalent VWAA, we can keep the co-Büchi acceptance when complementing.

We now consider the automaton $\mathcal{A}'$ that we get after adding complement states. As dualization of the transition structure does not change the underlying graph, and no state s can reach its complement state $\tilde{s}$, we have:

$$\forall s, s_1, s_2. (s_1 \xrightarrow{*} s_2 \iff \tilde{s}_1 \xrightarrow{*} \tilde{s}_2) \quad \text{and} \quad s \not\xrightarrow{*} \tilde{s} \quad (1)$$

It follows from the very weakness of $\mathcal{A}$ that $\mathcal{A}'$ is also very weak. Furthermore, the “strict successor” relation $<$ between pairs of states s_1, s_2 defined by

$$s_1 < s_2 \iff (s_1 \xrightarrow{*} s_2 \wedge s_1 \neq s_2) \quad (2)$$

is a strict partial order. Using $<$, we define the following relation over pairs of states:

$$s_1 < s_2 \iff (s_1 < s_2 \vee \tilde{s}_1 < s_2) \quad (3)$$

So $s_1 < s_2$ holds if s_2 is a strict successor of either s_1 or $\tilde{s}_1$. From Eqs. (1) and (2) we can deduce that $<$ is a strict partial order on Q and satisfies:

$$\forall s_1, s_2. s_1 < s_2 \iff s_1 < \tilde{s}_2 \quad (4)$$

and

$$\forall s_1, s_2. (s_1 \xrightarrow{*} s_2 \wedge s_1 \neq s_2) \implies s_1 < s_2 \quad (5)$$

The existence of a strict partial order that respects reachability in the underlying graph in the sense above implies very weakness. We will use this later to show that very weakness is preserved by the translation. For the remaining part of this section we will assume that complement states have been added to $\mathcal{A}$, that is we take $\mathcal{A}$ to be $\mathcal{A}'$.

Source configurations and source states To characterize ambiguous situations we define *source configurations* and *source states*.

Definition 4.1 (*Source configurations and source states*)

- A *source configuration* of $\mathcal{A}$ is a reachable configuration C such that there exist two different configurations C_1, C_2 that are reachable from C via some $a \in \Sigma$ and $\mathcal{L}(C_1) \cap \mathcal{L}(C_2) \neq \emptyset$.
- A *source state* of a source configuration C as above is a state $s \in C$ with two transitions $S_1, S_2 \in \delta(s, a)$ such that $S_i \subseteq C_i$, for $i \in \{1, 2\}$, $S_1 \neq S_2$ and $(S_1 \cup S_2) \setminus (C_1 \cap C_2) \neq \emptyset$.

The last condition on a source state ensures that either S_1 or S_2 contains a state that is not common to C_1 and C_2 . By Definition 3.1, $\mathcal{A}$ is not unambiguous if a source configuration exists. Also, every source configuration has at least one source state as the following lemma shows.

Lemma 4.1 *Every source configuration of $\mathcal{A}$ has at least one source state.*

Proof Let C be a source configuration as above, with the successor configurations C_1, C_2 for $a \in \Sigma$. By the definition of a run for alternating automata, we have $C_i = \bigcup_{q \in C} E_i(q)$ for some choice $E_i(q) \in \delta(q, a)$ for every state $q \in C$. Hence, all sets $E_i(q)$ satisfy $E_i(q) \subseteq C_i$. If $E_1(q) = E_2(q)$ would hold for all $q \in C$, then $C_1 = C_2$ holds which contradicts the assumption. It remains to show that for some state $q \in C$ we have $(E_1(q) \cup E_2(q)) \setminus (C_1 \cap C_2) \neq \emptyset$. If this were not the case, then $(C_1 \cup C_2) \setminus (C_1 \cap C_2) = \emptyset$, which again implies $C_1 = C_2$. $\square$

Ambiguity check and finding source states For the analysis of source configurations and source states we use the standard product construction $\mathcal{G}_1 \otimes \mathcal{G}_2$, which returns a t-GBA such that $\mathcal{L}(\mathcal{G}_1 \otimes \mathcal{G}_2) = \mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2)$ for two given t-GBA $\mathcal{G}_1$ and $\mathcal{G}_2$. Specifically, we consider the self product $\mathcal{G}_{\mathcal{A}} \otimes \mathcal{G}_{\mathcal{A}}$ of $\mathcal{G}_{\mathcal{A}}$. It helps to identify ambiguity: $\mathcal{G}_{\mathcal{A}}$ is not unambiguous if and only if there exists a reachable state (C_1, C_2) in $\text{trim}(\mathcal{G}_{\mathcal{A}} \otimes \mathcal{G}_{\mathcal{A}})$ with $C_1 \neq C_2$.

The pair of configurations (C_1, C_2) is a witness to ambiguity of $\mathcal{A}$ by Lemma 3.1. We look for a symbol $a \in \Sigma$ and a configuration C such that $(C, C) \xrightarrow{a} (C'_1, C'_2) \rightarrow^* (C_1, C_2)$ is a path in $\text{trim}(\mathcal{G}_{\mathcal{A}} \otimes \mathcal{G}_{\mathcal{A}})$ and $C'_1 \neq C'_2$. Such a configuration must exist as we have assumed that $\mathcal{A}$ has a single initial state q_i , which implies that $\text{trim}(\mathcal{G}_{\mathcal{A}} \otimes \mathcal{G}_{\mathcal{A}})$ has a single initial state $(\{q_i\}, \{q_i\})$. Configuration C is a source configuration and therefore must contain a source state by Lemma 4.1 which we can find by inspecting all pairs of transitions of states in C .

In principle, this approach would work also if we allowed multiple initial states. It would require a simple extension of the definition of source configurations which captures that ambiguity may occur already in the choice of initial states.

Disambiguating a source state The general scheme for disambiguating source states is depicted in Fig. 10.

Assume that we have identified a source state s with successor sets S_1 and S_2 as explained above. The assumption made in Remark 2.2 guarantees $S_1 \not\subseteq S_2$ and $S_2 \not\subseteq S_1$.

We need to distinguish the looping successor sets (i.e. those S_i that contain s) from the non-looping. Technically, we consider two cases: either S_1 or S_2 do not contain s or both sets contain s . In the first case we assume, w.l.o.g., that $s \notin S_1$. The successor set S_2 is split into $|S_1|$ new successor sets $\{(S_2 \cup \{\tilde{s}_1\}) : s_1 \in S_1\}$. The new sets of states are added to $\delta(s, a)$ and the successor set S_2 is removed. If both S_1 and S_2 contain s , we proceed as in the first case but do not add the successor set $S_2 \cup \{\tilde{s}\}$ to $\delta(s, a)$. We call this transformation `local_disamb`,

Fig. 10 Disambiguation scheme for a source state s with successors s_1 and s_2 in the VWAA. Transitions with a successor set of size > 1 are conjoined by a $\bullet$

it takes a tuple $(\mathcal{A}, s, S_1, S_2)$ as above and returns a new VWAA. It preserves very weakness, as the transitions it introduces are between states that already satisfy $s < s'$.

Lemma 4.2 *The transformation `local_disamb` preserves very weakness.*

Proof Let $\mathcal{A}' = \text{local_disamb}(\mathcal{A}, s, S_1, S_2)$ and let $\rightarrow_{\mathcal{A}}^*$ and $\rightarrow_{\mathcal{A}'}^*$ be the reachability relations in the underlying graphs of respective automata. Let $<'$, $<'$ be defined as in Eqs. (2) and (3) respectively but using $\rightarrow_{\mathcal{A}'}^*$. By definition, $<'$ satisfies Eq. (5) (after replacing $\rightarrow^*$ by $\rightarrow_{\mathcal{A}'}^*$). So in order to show that $\mathcal{A}'$ is very weak it is enough to show that $<'$ is a strict partial order. We do this by showing that $<' = <$ holds, which is enough as we may assume that $<$ is a strict partial order.

As the transformation only adds edges to the underlying graph, it is enough to show $<' \subseteq <$. So suppose, for contradiction, that $<' \not\subseteq <$ and take a pair $p, q \in Q$ such that $p <' q$ and $p \not< q$. First, take the case that $p <' q$. As $p \not< q$, we must have $p \rightarrow_{\mathcal{A}}^* s$ and $\tilde{s}_1 \rightarrow_{\mathcal{A}}^* q$ for some $s_1 \in S_1 \setminus \{s\}$. This is because the only added transitions are between s and $\tilde{s}_1$ for all $s_1 \in S_1 \setminus \{s\}$. But then also $p < s_1$, as $s_1 \in S_1 \setminus \{s\}$ and S_1 is a successor set of s . From $\tilde{s}_1 \rightarrow_{\mathcal{A}}^* q$ we get either $\tilde{s}_1 < q$ or $\tilde{s}_1 = q$. We show that both cases yield $p < q$, contradicting the assumption. In the first case we have $s_1 < q$ which gives us $p < q$ by using $p < s_1$ and transitivity. In the second case we have $p < \tilde{q}$ which gives $p < q$ by Eq. (4). The case $\tilde{p} <' q$ is analogous. $\square$

Proposition 4.1 *The transformation `local_disamb` preserves the language of all states of the automaton.*

Proof Let $\mathcal{A}' = \text{local_disamb}(\mathcal{A}, s, S_1, S_2)$. It is enough to show that the language of the state s is preserved, as this is the only state whose transitions change. We make a case distinction on whether s is not included in one of S_1, S_2 (here we assume, w.l.o.g., that it is S_1) or it is included in both.

First assume that $s \notin S_1$. The transformation removes the successor set S_2 and adds the sets $S = \{(S_2 \cup \{\tilde{s}_1\}) : s_1 \in S_1\}$. We claim that a word that has an accepting run starting in S_2 either has an accepting run from S_1 , or from one of the sets in S . So assume that a word w does not have an accepting run starting in S_1 . Then, there exists a state $s_1 \in S_1$ such that w has no accepting run from s_1 . As $\tilde{s}_1$ accepts the complement language of s_1 , there exists an accepting run starting in $\tilde{s}_1$ for w . But then w is also accepted from $(S_2 \cup \{\tilde{s}_1\})$.

Now assume that $s \in S_1 \cap S_2$. Then a word that is accepted in S_2 but not in S_1 has an accepting run from S_2 and some state $\tilde{s}_1$, with $s_1 \in S_1 \setminus \{s\}$. Hence, it is accepted from some of the sets $\{S_2 \cup \{\tilde{s}_1\} : s_1 \in S_1 \setminus \{s\}\}$.

This argument depends on the fact that the languages of states $\tilde{s}_1$ are not changed by the transformation. To show this, it is enough to show that these states cannot reach s . For contradiction, assume that $\tilde{s}_1 \rightarrow_{\mathcal{A}'}^* s$ holds. But then $\tilde{s}_1 < s_1$ holds, as s_1 is a successor of s . This contradicts very weakness and Eq. (4). $\square$

It is not guaranteed that s is not a source state in `local_disamb` $(\mathcal{A}, s, S_1, S_2)$. However, the ambiguity that stems from the non-deterministic choice of transitions $S_1, S_2 \in \delta(a, s)$ is

removed. If s is still a source state it will be identified again for another pair of transitions. After a finite number of iterations all successor sets of s for any symbol in Σ will accept pairwise disjoint languages, in which case s cannot be a source state anymore.

Iterative algorithm Putting things together, our algorithm works as follows: it searches for a triple (s, S_1, S_2) such that s is a source state in $\mathcal{A}$, which is witnessed by the pair of successor configurations S_1, S_2 . This is done using $\mathcal{G}_{\mathcal{A}}$. Then, `local_disamb`($\mathcal{A}, s, S_1, S_2$) is computed, and the algorithm recurses (see Fig. 1). As rebuilding the t-GBA may become costly, in our implementation we identify which part of the t-GBA has to be recomputed due to the changes in $\mathcal{A}$, and rebuild only this part. If no source configuration is found, we know that both $\mathcal{A}$ and $\mathcal{G}_{\mathcal{A}}$ are unambiguous and we can apply degeneralization to obtain a UBA.

Lemma 4.3 *The iterative algorithm terminates and returns an unambiguous VWAA.*

Proof The local transformation replaces a successor set of the source state with at least one larger successor set in every iteration. Hence, the procedure terminates eventually. As the procedure terminates only when no ambiguous configuration pair is found, the VWAA that is returned is unambiguous by Lemma 3.4. $\square$

Complexity of the procedure The VWAA-to-t-GBA translation that we adapt produces a t-GBA $\mathcal{G}_{\mathcal{A}}$ of size at most 2^n for a VWAA $\mathcal{A}$ with n states. In our disambiguation procedure we enlarge $\mathcal{A}$ by adding complement states for every state in the original automaton, yielding a VWAA with $2n$ states. Thus, a first size estimate of $\mathcal{G}_{\mathcal{A}}$ in our construction is 4^n . However, no state in $\text{trim}(\mathcal{G}_{\mathcal{A}})$ can contain both s and $\tilde{s}$ for any state s of $\mathcal{A}$. The reason is that the language of a state in $\mathcal{G}_{\mathcal{A}}$ is the intersection of the languages of the VWAA-states it contains, and $\mathcal{L}(s) \cap \mathcal{L}(\tilde{s}) = \emptyset$. Thus, $\text{trim}(\mathcal{G}_{\mathcal{A}})$ has at most 3^n states.

The amount of ambiguous situations that we identify is bounded by the number of non-deterministic splits in the VWAA, which may be exponential in the length of the input LTL formula. In every iteration we check ambiguity of the new VWAA, which can be done in exponential time. In total, we may need an exponential number of exponential time ambiguity-checks, which is still in exponential time as $2^{O(n)} \cdot 2^{O(n)} = 2^{O(n)}$. Thus, our procedure computes a UBA in time exponential in the length of the formula. In contrast to just *checking* unambiguity of VWAA, we cannot expect to translate VWAA to UBA using only polynomial space as we may need exponential space just to keep the resulting UBA in memory.

5 Heuristics and optimizations

5.1 Heuristics for purely-universal formulas

In this section we introduce alternative disambiguation transformations for special source states representing formulas $\phi\mathcal{U}v$, where v is *purely-universal*. The class of purely-universal formulas is a syntactically defined subclass of LTL-formulas with suffix-closed languages. These transformations reduce the size of the resulting UBA and often produce automata of a simpler structure. The idea is to decide whether v holds whenever moving to a state representing $\phi\mathcal{U}v$ and, if not, finding the *last* position where it does not hold.

Example 5.1 Consider the formula $\Diamond\Box a$. The VWAA for this formula in 11a is ambiguous, as runs for words satisfying $\Box a$ may loop in the initial state for an arbitrary amount of steps before moving to the next state.

Fig. 11 Three VWAA for $\Diamond\Box a$. The automaton in (b) is the result of standard disambiguation and (c) is the result of the modified transformation applied to (a). The automaton in (c) is non-deterministic and has two looping states, whereas (b) is not non-deterministic and has three looping states

In the standard disambiguation transformation the state $\Diamond\neg a$ is added to the self loop of the initial state (Fig. 11b). The automaton in Fig. 11c, on the other hand, makes the following case distinction: either a word satisfies $\Box a$, in which case we move to that state directly, or there is a suffix that satisfies $\neg a$ and $\Box\Box a$. The state φ is used to find the *last* occurrence of $\neg a$, which is unique.

To generalize this idea and identify the situations where it is applicable we use the syntactically defined subclasses of LTL of *purely-universal* (ν), *purely-eventual* (μ) and *alternating* (ξ) formulas [3,17]. We emphasize that not all LTL-formulas belong to one of these classes. In the following definition φ ranges over arbitrary LTL formulas:

$$\begin{aligned}\nu &::= \Box\varphi \mid \nu \vee \nu \mid \nu \wedge \nu \mid \Box\nu \mid \nu\mathcal{U}\nu \mid \varphi\mathcal{R}\nu \mid \Diamond\nu \\ \mu &::= \Diamond\varphi \mid \mu \vee \mu \mid \mu \wedge \mu \mid \Box\mu \mid \varphi\mathcal{U}\mu \mid \mu\mathcal{R}\mu \mid \Box\mu \\ \xi &::= \Box\mu \mid \Diamond\nu \mid \xi \vee \xi \mid \xi \wedge \xi \mid \Box\xi \mid \varphi\mathcal{U}\xi \mid \varphi\mathcal{R}\xi \mid \Diamond\xi \mid \Box\xi\end{aligned}$$

Formulas that fall into these classes define suffix closed (ν), prefix closed (μ) and prefix invariant (ξ) languages respectively:

Lemma 5.1 [3,17] *For all $u \in \Sigma^*$ and $w \in \Sigma^\omega$:*

- If ν is purely-universal, then $uw \models \nu \implies w \models \nu$.
- If μ is purely-eventual, then $w \models \mu \implies uw \models \mu$.
- If ξ is alternating, then $w \models \xi \iff uw \models \xi$.

Let ν be purely-universal. We want to find a formula $\mathbf{g}(\nu)$, called the *goal* of ν , that is simpler than ν and satisfies $\mathbf{g}(\nu) \wedge \Box\nu \equiv \nu$. If ν does not hold initially for some word w we can identify the last suffix $w[i..]$ where it does not hold, given that such an i exists, by checking if $w[i..]$ satisfies $\neg\mathbf{g}(\nu) \wedge \Box\nu$.

It is not clear how to define $\mathbf{g}(\nu)$ for purely-universal formulas of the form $\nu_1 \vee \nu_2$ or $\nu_1\mathcal{U}\nu_2$. We therefore introduce the concept of *disjunction-free* purely-universal formulas in which all occurrences of $\vee$ and $\mathcal{U}$ appear in the scope of some $\Box$. As $\varphi\mathcal{R}\nu \equiv \nu$ if ν is purely-universal, we assume that all occurrences of $\mathcal{R}$ are also in the scope of some $\Box$ for purely-universal formulas.

Lemma 5.2 *Every purely-universal formula ν can be rewritten into a formula $\nu_1 \vee \dots \vee \nu_n$, where ν_i is disjunction-free for all $1 \leq i \leq n$.*

Proof Let v be a purely-universal formula. As a first step we remove all occurrences of $\mathcal{U}$ and $\mathcal{R}$ in v that are not in the scope of some $\Box$ by applying the rules $v_1 \mathcal{U} v_2 \mapsto v_2 \vee (v_1 \wedge \Diamond v_2)$ and $\varphi \mathcal{R} v' \mapsto v'$. These transformation rules preserve the language if v_1, v_2 and v' are purely-universal. Then, we proceed by induction on the structure of v . In the case that $v = \Box\varphi$ it already has the desired structure. For the cases $\vee, \wedge, \bigcirc$ and $\Diamond$ we apply the induction hypothesis to the subformulas and then lift the disjunction over the corresponding operators. $\square$

Disjunction-free purely-universal formulas have a natural notion of “goal”.

Definition 5.1 Let v be a disjunction-free and purely-universal formula. We define $g(v)$ inductively as follows:

$$\begin{aligned} g(\Box\varphi) &= \varphi & g(\bigcirc v) &= \bigcirc g(v) \\ g(v_1 \wedge v_2) &= g(v_1) \wedge g(v_2) & g(\Diamond v) &= \text{true} \end{aligned}$$

The reason for defining $g(\Diamond v)$ as **true** is that $\Diamond v$ is an alternating formula and checking its validity can thus be temporarily suspended. Indeed, the definition satisfies the equivalence that we aimed for:

Lemma 5.3 Let v be disjunction-free and purely-universal. Then $g(v) \wedge \bigcirc v \equiv v$.

Proof We show the claim by induction on the structure of v (v_1, v_2 and v' are assumed to be purely-universal).

- $v = \Box\varphi$: By instantiation we get the $\varphi \wedge \bigcirc\Box\varphi \equiv \Box\varphi$.
- $v = \bigcirc v'$: By induction hypothesis we have $g(v') \wedge \bigcirc v' \equiv v'$, which implies $\bigcirc g(v') \wedge \bigcirc\bigcirc v' \equiv \bigcirc v'$.
- $v = v_1 \wedge v_2$: By induction hypothesis we have $g(v_1) \wedge \bigcirc v_1 \equiv v_1$ and $g(v_2) \wedge \bigcirc v_2 \equiv v_2$. This implies $g(v_1) \wedge g(v_2) \wedge \bigcirc v_1 \wedge \bigcirc v_2 \equiv v_1 \wedge v_2$.
- $v = \Diamond v'$: We have defined $g(\Diamond v') = \text{true}$. As $\Diamond v'$ is an alternating formula it satisfies $\bigcirc\Diamond v' \equiv \Diamond v'$, which proves the claim.

$\square$

Consider Example 5.1 and let $v = \Box a$. The reason for not choosing $v = \Diamond\Box a$ is that the heuristic that we define next is used on formulas in which a v -formula is a direct subformula of an until-operator ($\varphi \mathcal{U} (v \vee \psi)$ is the most general form). In this example, $\neg g(v) \wedge \bigcirc v$ corresponds to $\neg a \wedge \bigcirc\Box a$, which is realized by the transition from state φ to state $\Box a$ in Fig. 11c.

Lemma 5.4 shows the general transformation scheme (applied left to right). It introduces non-determinism, but we show that it is not a cause of ambiguity as the languages of the two disjuncts are disjoint. An important difference to the known rule for $\mathcal{U}$ is that the left-hand side of the $\mathcal{U}$ -formula stays unchanged. This is favorable as it is the left-hand side that may introduce loops in the automaton.

Lemma 5.4 Let v be a disjunction-free and purely-universal formula. Then

$$1. \varphi \mathcal{U} (v \vee \psi) \equiv v \vee \gamma \text{ and } 2. \mathcal{L}(v) \cap \mathcal{L}(\gamma) = \emptyset$$

where $\gamma = \varphi \mathcal{U} ((\varphi \wedge \neg g(v) \wedge \bigcirc v) \vee (\psi \wedge \neg v))$.

Proof 1. “ $\implies$ ”: Take a word w that satisfies $\varphi \mathcal{U}(v \vee \psi)$. If $w \models v$ we are done. We know that either $w \models \varphi \mathcal{U}v$ or $w \models \varphi \mathcal{U}\psi$. In the first case there is a *least* i such that $w[i..] \models v$ and for all $j < i$: $w[j..] \models \varphi$. We get $w[(i-1)..] \models \varphi \wedge \neg g(v) \wedge \bigcirc v$ by Lemma 5.3 and thus $w \models \gamma$. In the second case there is an i such that $w[i..] \models \psi$ and for all $j < i$: $w[j..] \models \varphi$. We can assume that $w[i..] \not\models v$, as the first case applies otherwise, which implies $w[i..] \models \psi \wedge \neg v$ and thus proves $w \models \gamma$.

“ $\Leftarrow$ ”: For this direction it suffices to observe that:

$$\gamma \implies \varphi \mathcal{U}((\varphi \wedge \bigcirc v) \vee \psi) \text{ and } \varphi \mathcal{U}((\varphi \wedge \bigcirc v) \vee \psi) \implies \varphi \mathcal{U}(v \vee \psi)$$

2. Recall that by Lemma 5.3, $v \equiv g(v) \wedge \bigcirc v$ and hence in particular $\neg g(v) \implies \neg v$. It follows that $\gamma \implies \Diamond \neg v$. As v is suffix closed (see Lemma 5.1), we have $\Diamond \neg v \implies \neg v$. We can conclude that $\gamma \implies \neg v$. $\square$

LTL formulas may become larger when applying this transformation. However, they are comparable to the LTL formulas produced by the standard disambiguation transformations in terms of the number of subformulas. Also, the only new subformulas are formed by applying Boolean combinations and $\bigcirc$ to existing subformulas. Such formulas generally contribute less to the complexity of the procedure. In our implementation, we adapt the above transformation to locally disambiguate states in the VWAA with a corresponding transition structure.

5.2 Postprocessing $\mathcal{G}_{\mathcal{A}}$ by splitting states

In the translation from VWAA to t-GBA, states of the t-GBA are sets (representing conjunctions) of VWAA states. It might introduce redundant states: if the t-GBA has the two states $S \cup \{q\}$ and $S \cup \{\tilde{q}\}$, where S is a set of VWAA states and q is a VWAA state, then the state S is redundant, as it can be expressed as the union of the former two. Thus, if we discover three states of the form S , $S \cup \{q\}$, and $S \cup \{\tilde{q}\}$, we remove S and redirect the transitions to both $S \cup \{q\}$ and $S \cup \{\tilde{q}\}$. We first define this formally as a state-splitting operation on the t-GBA-level. It takes a state P and a set of states S in the t-GBA and redirects all incoming transitions of P to all states in S . The redirected transitions keep their acceptance marks. In principle, this transformation works also in the other way: if a state has a nondeterministic choice between states $S \cup \{q\}$ and $S \cup \{\tilde{q}\}$, we could replace these transitions with a single transition to S .

Let $\mathcal{A} = (Q, \Sigma, \delta_{\mathcal{A}}, \iota_{\mathcal{A}}, Q_F)$ be a VWAA and $\mathcal{G} = (2^Q, \Sigma, \delta_{\mathcal{G}}, \iota_{\mathcal{G}}, \bigwedge_{q \in Q_F} \text{Inf}(\mathcal{T}_q))$ be a t-GBA as produced by the translation in Definition 2.2.

Definition 5.2 Let $P \in 2^Q$ and $S \subseteq 2^Q$. We define the automaton $\mathcal{G}[P \mapsto S] = (2^Q, \Sigma, \delta', \iota', \bigwedge_{q \in Q_F} \text{Inf}(\mathcal{T}'_q))$ as follows:

- If $P \in \iota_{\mathcal{G}} : \iota' = (\iota_{\mathcal{G}} \setminus \{P\}) \cup S$. Otherwise: $\iota' = \iota_{\mathcal{G}}$.
- For all $(R, a) \in 2^Q \times \Sigma$:

$$\delta'(R, a) = \begin{cases} \delta_{\mathcal{G}}(R, a) & \text{if } P \notin \delta_{\mathcal{G}}(R, a) \\ (\delta_{\mathcal{G}}(R, a) \setminus \{P\}) \cup S & \text{otherwise} \end{cases}$$

- For all $q \in Q_F$:

$$\mathcal{T}'_q = \{(R, a, R') : (R' \notin \delta_{\mathcal{G}}(R, a) \text{ and } (R, a, P) \in \mathcal{T}_q) \\ \text{or } (R' \in \delta_{\mathcal{G}}(R, a) \text{ and } (R, a, R') \in \mathcal{T}_q)\}$$

The definition of the acceptance sets of $\mathcal{G}[P \mapsto S]$ is such that the transitions introduced in $\mathcal{G}[P \mapsto S]$ have the same acceptance status as the corresponding transition that was removed. As all incoming transitions to P are redirected, P is not reachable in the resulting automaton $\mathcal{G}[P \mapsto S]$. We first show that the language of all states of the automaton is preserved by this operation.

Lemma 5.5 *Let $P \subseteq Q, S \subseteq 2^Q$ such that*

1. *for all $S \in \mathcal{S} : P \subseteq S$ and $P \neq S$*
2. *$\mathcal{L}_{\mathcal{A}}(P) \subseteq \bigcup_{S \in \mathcal{S}} \mathcal{L}_{\mathcal{A}}(S)$*

Then for all initial state sets $\iota \subseteq 2^Q$ we have $\mathcal{L}(\mathcal{G}(\iota)) = \mathcal{L}(\mathcal{G}[P \mapsto S](\iota))$.

Proof Let $\mathcal{G}' = \mathcal{G}[P \mapsto S]$. We show for every state q_0 of $\mathcal{G}$: $\mathcal{L}(\mathcal{G}(q_0)) = \mathcal{L}(\mathcal{G}'(q_0))$.

“ $\supseteq$ ”: Let $u \in \Sigma^\omega$ such that there exists an accepting run $r = Q_0 Q_1 \dots$ of $\mathcal{G}'(q_0)$ for u . Then, the sequence $(Q_i)_{i \geq 0}$ satisfies for all $i \in \mathbb{N}$:

- (a) $Q_{i+1} \in \bigotimes_{q \in Q_i} \delta_{\mathcal{A}}(q, u[i])$, or (old transitions)
- (b) $Q_{i+1} \in S$ and $P \in \bigotimes_{q \in Q_i} \delta_{\mathcal{A}}(q, u[i])$ (new transitions)

From the fact that all $S \in \mathcal{S}$ satisfy $P \subseteq S$, it follows that all states in Q_i have a successor set in $\mathcal{A}$ for $u[i]$ that is included in Q_{i+1} , for all i . Hence, there is a run of $\mathcal{A}$ for u with layers pointwise included in Q_i . It remains to show that the successor sets can be chosen such that the resulting run is accepting, which means that no infinite run gets trapped in a state in Q_F .

Suppose that this is not the case. Then there exist $N \in \mathbb{N}$ and $q_f \in Q_F$ such that for all $n > N$ and $Q' \in \bigotimes_{q \in Q_n} \delta_{\mathcal{A}}(q, u[n])$ with $Q' \subseteq Q_{n+1}$: $q_f \in Q'$ and no $Y \in \delta_{\mathcal{A}}(q_f, u[n])$ satisfies both $Y \subseteq Q'$ and $q_f \notin Y$. As r is accepting, there is an edge $(R, a, R') \in T'_{q_f}$ such that for infinitely many $n : R = Q_n, u[n] = a, R' = Q_{n+1}$. Suppose first, that $R' \in \delta_{\mathcal{G}}(R, a)$ and $(R, a, R') \in T_{q_f}$. Then, either $q_f \notin R'$ or there exists an $Y \subseteq R'$ such that $Y \in \delta_{\mathcal{A}}(q_f, a)$ and $q_f \notin Y$. This contradicts the existence of N as above. Now suppose that $R' \notin \delta_{\mathcal{G}}(R, a)$ and $(R, a, P) \in T_{q_f}$. In this case we have $R' \in S$ and $P \in \bigotimes_{q \in Q_n} \delta_{\mathcal{A}}(q, u[n])$. As $(R, a, P) \in T_{q_f}$, we have either $q_f \notin P$ or there exists an $Y \subseteq P$ such that $Y \in \delta_{\mathcal{A}}(q_f, a)$ and $q_f \notin Y$. This is again a contradiction to the existence of N as above. This shows that there exists an accepting run of $\mathcal{A}(q_0)$ for u , which implies that u is accepted by $\mathcal{G}(q_0)$ by Lemma 3.3.

“ $\subseteq$ ”: Let r be an accepting run of $\mathcal{G}(q_0)$ for u . Then there exists an accepting run ρ_0 of $\mathcal{A}(q_0)$ for u by Theorem 2.1. We will iteratively construct a DAG γ that extends ρ_0 and whose layers will be an accepting run of $\mathcal{G}'(q_0)$.

To this end, whenever P appears as a layer of ρ_0 , we replace it by some configuration in S , all of which are supersets of P . The fact that this can be done in a way that yields an accepting run follows by condition (2.), which tells us that if an accepting run exists from P , then also some $S \in \mathcal{S}$ has an accepting run for the suffix word.

Formally we construct γ as follows. Up till the first position n such that $\rho_0(n) = P$, γ simply copies ρ_0 . By (2.), there exists an $S \in \mathcal{S}$ with $u[n..] \in \mathcal{L}_{\mathcal{A}}(S)$. It follows that we find an accepting run ρ_1 of $\mathcal{A}(S)$ for $u[n..]$. We assume, w.l.o.g., that the suffix of ρ_0 from position n coincides with the restriction of ρ_1 to initial states P (recall that $P \subseteq S$). We continue to construct γ by copying ρ_1 until a position in which this run has layer P , and so on. Then, each infinite path of γ corresponds to an infinite path of some accepting run of $\mathcal{A}$ by construction, and hence is accepting. Let $\mathcal{J} \subseteq \mathbb{N}$ be the positions where we encountered a P in the above construction. States $q \in (\gamma(j) \setminus P)$ have no predecessors in γ , if $j \in \mathcal{J}$. This is because such states were added to γ when extending a layer P to some $S \in \mathcal{S}$.

The sequence of layers $\gamma(0)\gamma(1)\dots$ is a run of $\mathcal{G}'(q_0)$. This is because for all i we have either: $\gamma(i+1) \in \bigotimes_{q \in \gamma(i)} \delta_{\mathcal{A}}(q, u[i])$ or $P \in \bigotimes_{q \in \gamma(i)} \delta_{\mathcal{A}}(q, u[i])$ and $\gamma(i+1) \in \mathcal{S}$. The positions $j+1 \in \mathcal{J}$ are exactly the ones where $\gamma(j+1) \notin \delta_{\mathcal{G}}(\gamma(j), u[j])$.

Assume, for contradiction, that it is not an accepting run. Then there exists some $q \in Q_F$ such that no transition in $\mathcal{T}'_q$ is seen infinitely often. In other words, there is an $N \in \mathbb{N}$ such that for all $n \geq N$: $(\gamma(n), u[n], \gamma(n+1)) \notin \mathcal{T}'_q$. By definition of $\mathcal{T}'_q$ this implies that for all $n > N$: $q \in \gamma(n)$ and

- if $n+1 \notin \mathcal{J}$, then there is no $Y \subseteq \gamma(n+1)$ s.t. $Y \in \delta_{\mathcal{A}}(q, u[n])$ and $q \notin Y$.
- if $n+1 \in \mathcal{J}$, then there is no $Y \subseteq P$ s.t. $Y \in \delta_{\mathcal{A}}(q, u[n])$ and $q \notin Y$.

From this it follows that γ has an infinite rejecting path $(q, n) \rightarrow (q, n+1) \rightarrow (q, n+2) \rightarrow \dots$, as shown next. Recall that the set of successors of (q, n) in γ corresponds to a successor set $Y \in \delta_{\mathcal{A}}(q, u[n])$ for all n . For $n > N$ there is no such $Y \subseteq \gamma(n+1)$ with $q \notin Y$, except if $n+1 \in \mathcal{J}$ and $Y \not\subseteq P$. But we have already observed that states $q \in (\gamma(j) \setminus P)$ have no predecessors in γ if $j \in \mathcal{J}$, and hence Y cannot be of this form. It follows that for all $n > N$ we have $q \in \delta_{\mathcal{A}}(q, u[n])$ and hence that there exists an infinite rejecting path in γ . But this yields a contradiction to the fact that all infinite paths of γ correspond to some accepting run of $\mathcal{A}$ by construction. We can conclude that $\gamma(0)\gamma(1)\dots$ is an accepting run of $\mathcal{G}'(q_0)$ for u . $\square$

Based on the previous lemma, we argue that the following strategy to split states is correct: whenever states X , $X \cup \{p\}$ and $X \cup \{\tilde{p}\}$ exist in the t-GBA $\mathcal{G}_{\mathcal{A}}$, apply the split operation

$$\mathcal{G}'_{\mathcal{A}} = \mathcal{G}_{\mathcal{A}}[X \mapsto \{X \cup \{p\}, X \cup \{\tilde{p}\}\}]$$

Then, iterate the procedure until no triple of states as above is found. The result of such a sequence of splitting operations is equivalent to the result of:

$$\mathcal{G}' = (\dots((\mathcal{G}_{\mathcal{A}}[P_1 \mapsto S_1])[P_2 \mapsto S_2])\dots[P_n \mapsto S_n]) \quad (6)$$

where the P_i are the states X that are removed by the split operations, and S_i is the final set of states that P_i is split into after the iterative procedure finishes. That is, if $X = P_i$ is first transformed into $X \cup \{p\}$ and $X \cup \{\tilde{p}\}$, and later $X \cup \{p\}$ is transformed into $X \cup \{p, q\}$ and $X \cup \{p, \tilde{q}\}$, then $S_i = \{X \cup \{p, q\}, X \cup \{p, \tilde{q}\}, X \cup \{\tilde{p}\}\}$. In particular, for all i, j we have $P_i \notin S_j$.

Hence, each split in Eq. (6) satisfies the preconditions of Lemma 5.5. The proof of Lemma 5.5 can be adapted to show that splitting multiple states at once, as in Eq. (6), also preserves the language under the given preconditions. Finally, we show that splitting states preserves unambiguity.

Lemma 5.6 *Let $P \subseteq Q, S \subseteq 2^Q$, such that all states $S \in \mathcal{S}$ are reachable in $\mathcal{G}$ and*

1. *for all $S \in \mathcal{S}$: $P \subseteq S$ and $P \neq S$,*
2. *for all $S_1, S_2 \in \mathcal{S}$: $S_1 \neq S_2$ implies $\mathcal{L}_{\mathcal{G}}(S_1) \cap \mathcal{L}_{\mathcal{G}}(S_2) = \emptyset$.*

Then, if $\mathcal{G}$ is unambiguous, then so is $\mathcal{G}[P \mapsto S]$.

Proof A non-deterministic automaton is unambiguous if and only if for all reachable states q , all symbols a and all $q_1, q_2 \in \delta(q, a)$: $q_1 \neq q_2 \implies \mathcal{L}(q_1) \cap \mathcal{L}(q_2) = \emptyset$.

We show that this property is preserved by the transformation $\mathcal{G}' = \mathcal{G}[P \mapsto S]$. Take any reachable state R of $\mathcal{G}'$ together with a pair of different successors $R_1, R_2 \in \delta'(R, a)$ for some $a \in \Sigma$. We show that $\mathcal{L}_{\mathcal{G}'}(R_1) \cap \mathcal{L}_{\mathcal{G}'}(R_2) = \emptyset$. By Lemma 5.5 it is enough to show that $\mathcal{L}_{\mathcal{G}}(R_1) \cap \mathcal{L}_{\mathcal{G}}(R_2) = \emptyset$. One of the following must hold:

- (a) $R_1, R_2 \in \delta_{\mathcal{G}}(R, a)$ (b) $R_1 \notin \delta_{\mathcal{G}}(R, a)$ and $R_2 \in \delta_{\mathcal{G}}(R, a)$ (c) $R_1, R_2 \notin \delta_{\mathcal{G}}(R, a)$

In (b) we assume w.l.o.g. that R_1 is the set that is not in $\delta_{\mathcal{G}}(R, a)$. In case (a), $\mathcal{L}_{\mathcal{G}}(R_1) \cap \mathcal{L}_{\mathcal{G}}(R_2) = \emptyset$ follows from unambiguity of $\mathcal{G}$. In case (b), we know that $P \in \delta_{\mathcal{G}}(R, a)$ and $R_1 \in \mathcal{S}$, which implies $P \subseteq R_1$. Observe that $C_1 \subseteq C_2$ implies $\mathcal{L}_{\mathcal{A}}(C_2) \subseteq \mathcal{L}_{\mathcal{A}}(C_1)$, because a larger configuration means that a word needs to be accepted from more states. Together with the fact that $\mathcal{L}_{\mathcal{G}}(C) = \mathcal{L}_{\mathcal{A}}(C)$ for all C (by Theorem 2.1) we may conclude that $\mathcal{L}_{\mathcal{G}}(R_1) \subseteq \mathcal{L}_{\mathcal{G}}(P)$. With $\mathcal{L}_{\mathcal{G}}(P) \cap \mathcal{L}_{\mathcal{G}}(R_2) = \emptyset$ we get: $\mathcal{L}_{\mathcal{G}}(R_1) \cap \mathcal{L}_{\mathcal{G}}(R_2) = \emptyset$. In case (c) we get $R_1, R_2 \in \mathcal{S}$ and hence $\mathcal{L}_{\mathcal{G}}(R_1) \cap \mathcal{L}_{\mathcal{G}}(R_2) = \emptyset$ follows by 2. $\square$

The heuristic runs in cubic time in the number of states of the t-GBA and never increases its size. As the information on complement states is readily available in our procedure, this heuristic is included in all configurations of our tool *Duggi*. As our disambiguation procedure often introduces states of the form $S \cup \{q\}$ and $S \cup \{\bar{q}\}$ this heuristic can often be applied.

6 Implementation and experiments

The tool *Duggi* is an LTL-to-UBA translator based on the construction introduced in the foregoing sections.² It reads LTL formulas in a prefix syntax and produces (unambiguous) automata in the HOA format [2]. In the implementation we deviate from or extend the procedure described above in the following ways:

- We make use of the knowledge given by the VWAA-complement states in the translation steps to t-GBA $\mathcal{G}_{\mathcal{A}}$ and the product $\mathcal{G}_{\mathcal{A}} \otimes \mathcal{G}_{\mathcal{A}}$. It allows an easy emptiness check: if s and $\bar{s}$ are present in some $\mathcal{G}_{\mathcal{A}}$ or $\mathcal{G}_{\mathcal{A}} \otimes \mathcal{G}_{\mathcal{A}}$ state, then it accepts the empty language and does not have to be further expanded.
- We have included the following optimization of the LTL-to-VWAA procedure: when translating a formula $\Box\mu$, where μ is purely-eventual, we instead translate $\Box \bigcirc \mu$. This results in an equivalent state with fewer transitions. It is close to the idea of suspension as introduced in [3], but is not covered by it.
- Additionally, *Duggi* features an LTL rewriting procedure that uses many of the LTL simplification rules in the literature [3,17,33,41]. We have included the following rules that are not used by SPOT:

$$\text{I } (\Box\Diamond\varphi) \wedge (\Diamond\Box\psi) \mapsto \Box\Diamond(\varphi \wedge \Box\psi) \quad \text{II } (\Diamond\Box\varphi) \vee (\Box\Diamond\psi) \mapsto \Diamond\Box(\varphi \vee \Diamond\psi)$$

These rewrite rules are more likely to produce formulas of the form $\Diamond\Box\varphi$, to which the heuristic of Sect. 5.1 can be applied. They stem from [33], where the reversed rules have been used to achieve a normal form.

Now we report on the experiments. At first, we demonstrate the behavior of *Duggi* for the LTL benchmark set LTLSTORE [26] comparing automata sizes, time consumption and number of necessary `local_disamb` transformations. We evaluate next the LTL rewrite rules and the purely-universal heuristic in the setting of certain LTL families designated to this heuristics. At third, we discuss the transformation `local_disamb`. We left the choice open, which state is complemented in the local disambiguation step. We measure the effects

² *Duggi* and the PRISM implementation, together with all experimental data, are available at <https://www.tcs.inf.tu-dresden.de/ALGI/TR/FMSD20/>.

Fig. 12 Non-WDBA-recognizable fragment of LTLSTORE (955 formulas). Every point stands for a formula where the according automaton size for Duggi is the abscissa, the automaton size of 1t12tgba the ordinate. Points above the line stand for formulas where Duggi performed better

of different choices and set here a standard variant used for the experiments in this section. At last we address the performance of our disambiguation algorithm in the overall context of Markov chain analysis.

LTL benchmarks from the literature Our first benchmark set includes the formulas of the LTLSTORE [26] and their negation³, which yields 1432 formulas in total (after removing syntactical duplicates). The LTLSTORE collects formulas from various case studies and tool evaluation papers in different contexts.

Languages that are recognizable by *weak deterministic Büchi automata* (WDBA) can be efficiently minimized [30] and 1t12tgba applies this algorithm as follows: it computes the minimal WDBA and the UBA and returns the one with fewer states. Our formula set contains 477 formulas that are WDBA-recognizable and for which we could compute the minimal WDBA within the bounds of 30 minutes and 10GB of memory using 1t12tgba. Of these 477 formulas we found 12 for which the UBA generated by either Duggi or 1t12tgba was smaller than the minimal WDBA, and only two where the difference was bigger than 3 states. On the other hand, the minimal WDBA were smaller than the UBA produced by 1t12tgba (Duggi) for 167 (180) formulas. This supports the approach by 1t12tgba to apply WDBA minimization when possible and in what follows we focus on the 955 formulas of the LTLSTORE that do not fall into this class. In [15] it was noted that WDBA minimization often leads to smaller automata than the LTL-to-NBA translation of 1t12tgba.

We consider the following configurations: Duggi is the standard configuration, $\text{Duggi}_{\setminus(R,H)}$ is Duggi without the new rewrite rules I and II (R) and/or without the heuristic introduced in Sect. 5.1 (H). For SPOT, 1t12tgba is the standard configuration that produces UBA without WDBA-minimization, which is switched on in $\text{1t12tgba}_{\text{WDBA}}$. We use simulation-based postprocessing as provided by SPOT in all Duggi-configurations (they are enabled by default in 1t12tgba). We use SPOT with version 2.8.5. All computations, including the PMC experiments, were performed on a computer with two Intel E5-2680 8 cores at 2.70 GHz running Linux, with a time bound of 30 minutes and a memory bound of 10GB.

Scatter plots comparing the number of states of UBA produced by 1t12tgba and Duggi are shown in Fig. 12. Table 1 gives cumulative and average results of different configurations

³ Retrieved on June 29, 2018, commit AD8B5CD7C9C30D1E65DBDA676FDF41821C3A8ADB.

Table 1 Cumulative and average results on the LTLSTORE benchmark set

	Non-WDBA-recognizable		WDBA-recognizable	
	States (avg.)	Time in s (avg.)	states (avg.)	Time in s (avg.)
Duggi	14,447 (18.15)	23,238 (29.19)	6332 (14.86)	3786 (8.89)
Duggi ⁱ _R	14,350 (18.21)	24,520 (31.12)	6332 (14.86)	3792 (8.90)
Duggi ⁱ _(R.H)	14,654 (18.84)	35,191 (45.23)	6359 (14.93)	5709 (13.40)
1t12tgba	18,058 (22.08)	4168 (5.10)	9161 (19.83)	3166 (6.85)
1t12tgbaWDBA	17,922 (21.99)	6590 (8.09)	7527 (16.29)	3121 (6.76)

(a) Average automata size for a given number of executed iterations.

(b) Average time consumption for a given number of iterations.

(c) Number of formulas needing a given number of iterations.

(d) Average number of iterations needed for VWAA $\mathcal{A}$ with $\text{nd}(\mathcal{A}) \leq 20$. In the case of $\text{nd}(\mathcal{A}) = 0$, no iteration is needed.

Fig. 13 Comparison of the number of iterations with automata sizes of the UBA, time consumption for the construction and non-determinism of the initial VWAA

on these formulas. All configurations of Duggi use more time than `ltl2tgba`, but produce smaller automata on average. One reason why Duggi uses more time is the on-demand nature of algorithm, which rebuilds the intermediate t-GBA several times while disambiguating. The average number of disambiguation iterations, that is, the number of applied `local_disamb` transformations, per formula of Duggi on the non-WDBA-recognizable fragment was 12.34.

Figure 13 shows the behavior of Duggi concerning iterations. We only consider automata with at most 14 iterations in Fig. 13a–c, as for more iterations the number of automata is small, which leads to a high variation. Even for a smaller number of iterations, the time consumption varies highly, e.g., for 13 iterations the time consumption varies from less than one second up to 706 s. But still, as a general trend, a higher number of iterations leads to bigger automata and a higher time consumption. Figure 13c shows that the majority of formulas (405) need at most 4 iterations.

The amount of non-determinism present in the VWAA influences the number of iterations necessary for the disambiguation process, since more non-determinism can cause more ambiguity, which has to be resolved by the iterations. Thus, an increase of nd increases also the number of necessary iterations (see Fig. 13d).

$$(a) \Phi_n = \bigwedge_{i \leq n} (\Diamond \Box p_{2i} \vee \Box \Diamond p_{2i+1})$$

$$(b) \theta_n = (\bigwedge_{i \leq n} \Box \Diamond p_i) \rightarrow \Box (req \rightarrow \Diamond res)$$

Fig. 14 UBA sizes for two sequences of parametrized formulas. If there is no entry for some tool and parameter it means that a timeout occurred

(a) Time in seconds needed for the translation of φ_k^{steps} into a UBA.

(b) Time in seconds needed for model checking the BRP model with φ_k^{steps} .

Fig. 15 Time consumption for translating and model checking φ_k^{steps} (which includes building the automaton)

LTL rewrites and the purely-universal heuristic A formula that benefits from using the rewrite rules I and II is $\Phi_n = \bigwedge_{i \leq n} \Diamond \Box p_{2i} \vee \Box \Diamond p_{2i+1}$, which is a *strong fairness* condition. Here `lt12tgba` applies the rule $\Diamond \varphi \vee \Box \Diamond \psi \mapsto \Diamond (\varphi \vee \Box \Diamond \psi)$ which yields $\bigwedge_{i \leq n} \Diamond (\Box p_{2i} \vee \Box \Diamond p_{2i+1})$. Applying rule II yields the formula $\Diamond \Box (\bigwedge_{i \leq n} p_{2i} \vee \Diamond p_{2i+1})$.

Figure 14a shows that `Duggi` produces smaller automata for Φ_n . Figure 14b shows the corresponding results for the parametrized formula $\theta_n = (\bigwedge_{i \leq n} \Box \Diamond p_i) \rightarrow \Box (req \rightarrow \Diamond res)$ which is a request/response pattern under fairness conditions.

A property that profits from the “on-demand” disambiguation is: “*b* occurs *k* steps before *a*”. We express it with the formula $\varphi_k^{\text{steps}} = \neg a \mathcal{U} (b \wedge \neg a \wedge \bigcirc \neg a \wedge \dots \wedge \bigcirc^{k-1} \neg a \wedge \bigcirc^k a)$. Both `Duggi` and `lt12tgba` produce the minimal UBA, but `lt12tgba` produces an exponential-sized automaton in an intermediate step, because it does not realize that the original structure is already unambiguous. This leads to high run times for large *k* (see Fig. 15a).

Complementation state choice The VWAA disambiguation procedure consists of finding source states and successor configuration causing ambiguity (see Sect. 4). We then modify one of the two successor configurations by adding complement states of the other successor

Table 2 Evaluation of the successor configuration choice in the local disambiguation transformations on the non-WDBA recognizable fragment of the LTLSTORE

Succ. configuration heuristic	Non-WDBA-recognizable		Timeouts	Avg. iterations
	States (avg.)	Time in s (avg.)		
nd unreach univ reach loop	14,447 (18.15)	23,238 (29.19)	159	12.34
nd univ unreach reach loop	16,175 (20.37)	31,377 (39.52)	161	12.98
unreach univ nd reach loop	15,675 (19.92)	38,053 (48.35)	168	13.86
unreach nd univ reach loop	15,725 (19.88)	35,947 (45.45)	164	13.90
univ nd unreach reach loop	17,893 (22.74)	37,558 (47.72)	168	13.78

As the first permutation performs best, it is used as standard heuristic for Duggi in all other benchmark runs

configuration. As the algorithm does not specify which of the successor configurations is modified (unless exactly one of them is looping), and this choice may have a considerable effect on the size of the resulting automaton, we now give heuristics on which configuration to choose.

So assume that we have identified a source state q with successor configurations S_1 and S_2 . If both S_1 and S_2 are non-looping, we may decide to replace the transition to S_1 by $\{(S_1 \cup \{\tilde{s}_2\}) : s_2 \in S_2\}$, or to S_2 by $\{(S_2 \cup \{\tilde{s}_1\}) : s_1 \in S_1\}$. We define the following measures, which we apply to the subautomata induced by the states $\tilde{S}_1 = \{\tilde{s}_1 : s_1 \in S_1\}$, $\tilde{S}_2 = \{\tilde{s}_2 : s_2 \in S_2\}$:

- **univ** : $2^Q \rightarrow \mathbb{R}$ measures the average size of successor sets in the subautomaton. Larger successor sets generally restrict the language more and may lead earlier to a state recognizing the empty language in the resulting t-GBA.
- **loop** : $2^Q \rightarrow \mathbb{N}$ measures the number of states in the subautomaton having a self-loop. States with self-loops often produce many more states in the t-GBA than states without self-loops.
- **nd** : $2^Q \rightarrow \mathbb{N}$ measures the nondeterminism of a configuration S in the following sense: for every state q reachable from S the number of outgoing transitions sharing the same label but different goals is counted. The nondeterminism for S is then just the sum of all state values q .
The reasoning behind nd is that the disambiguation restricts the nondeterminism. Therefore we prefer configurations with less nondeterminism.
- **reach** : $2^Q \rightarrow \mathbb{N}$ measures the number of reachable states in the subautomaton induced by the argument.
- **unreach** : $2^Q \rightarrow \mathbb{N}$ measures the number of reachable states that are not already states of the current VWAA.

Hence, we measure how many states are added to the VWAA after the transformation.

For every heuristic we see the subautomaton with a smaller number as the preferable one, except for univ where a bigger number is preferable. We can combine several heuristics by a lexicographic comparison, i.e., define a sequence of heuristics, and if the values for both subautomata agree for one heuristic, move to the next one. Naturally, this happens often in the case of, e.g., the loop heuristic.

Table 2 compares 5 possible orderings of these measures on the non-WDBA recognizable fragment of the LTLSTORE. It shows that the disambiguation procedure is indeed sensitive to changes of this heuristic. In all other computations with Duggi we chose the configuration nd unreach univ reach loop, as it performed best in this comparison.

Fig. 16 Model checking times for the cluster protocol with φ_k and ψ_k

Use case: probabilistic model checking Now we look at an important application of UBA, the analysis of Markov chains. We compare run times of an implementation of [5] for Markov chain model checking with UBA, using PRISM (version 4.4) and either Duggi or lt12tgba as automata generation backends. We take two models of the PRISM benchmark suite [29], the bounded retransmission protocol, and the cluster working protocol [22].

The bounded retransmission protocol (BRP) is a message transmission protocol, where a sender sends a message and receives an acknowledgment if the transmission was successful. We set the parameter N (the number of the message parts) to 16, and MAX (the number of maximal retries) to 128. We reuse φ_k^{steps} , which now means: “ k steps before an acknowledgment there was a retransmit”, where we replace a by `ack_received` and b by `retransmit`. As expected, the faster automaton generation leads to lower model checking times when using Duggi (Fig. 15b). The reason for the spikes in Fig. 15b is that the probability of the property is zero in the BRP model for odd k . This makes the model checking (which uses the numeric procedure of [5]) easier. For bigger k the automaton generation uses a bigger share of the time, making this effect less pronounced.

As second model we analyse the cluster working model with the LTL properties presented in [21]. It consists of a workstation cluster with two sub-clusters that are connected by a backbone and have $n = 16$ participants each.

Let fct_i denote the number of functional working stations in sub-cluster i . We define $\varphi_{\square\Diamond} = \square\Diamond(\text{fct}_1 = n)$, which expresses that the first cluster stays functional on the long run and $\varphi_{\Diamond\square}^k = \bigvee_{i \in \{0, \dots, k\}} \Diamond\square(\text{fct}_2 = n - i)$, which expresses the property that from some point onwards, the second cluster contains at least $n - k$ functional working stations. We check the three formula patterns

$$\begin{aligned}\varphi_k &:= \varphi_{\square\Diamond} \wedge \varphi_{\Diamond\square}^k \text{ and} \\ \psi_k &:= \varphi_{\square\Diamond} \vee \varphi_{\Diamond\square}^k\end{aligned}$$

and additionally the WDBA-realizable formula

$$\varphi_k^{\mathcal{U}} := (\text{fct}_1 = n) \mathcal{U} \left((\text{fct}_1 = n - 1) \mathcal{U} \left(\dots \mathcal{U} ((\text{fct}_1 = n - k) \mathcal{U} (\text{fct}_2 \neq n)) \right) \right)$$

which describes: “There are at most k failures in the first cluster before the first failure in the second cluster.”

The results for φ_k are depicted in Fig. 16a. Both tools have a time-out at $k = 4$, although, for smaller k , the time consumption of Duggi was bigger than lt12tgba. Comparing

Fig. 17 Model checking times for the cluster workstation protocol for formulas φ_k^U

the automata size, Duggi produces smaller automata for both $k = 2$ and $k = 3$, e.g., 33 (Duggi) vs. 137 (1t12tgba) states for $k = 3$. The results for ψ_k can be seen in Fig. 16b. Duggi performed better than 1t12tgba, as Duggi reached the time-out at $k = 6$ (vs. $k = 4$ for 1t12tgba). However, if no time-out was reached, 1t12tgba consumed less time. Nevertheless, for $k \leq 3$, model checking time of both tools was below 4 s. Still, Duggi produced smaller automata, e.g., 25 (Duggi) vs. 59 (1t12tgba) states for $k = 3$.

For φ_k^U , the results (depicted in Fig. 17) show that in this case 1t12tgba performed significantly better. Duggi was not able to finish the model checking procedure for any k as the automata generation took too long. However, the language described by φ_k^U is WDBA recognizable, which for this particular formula can be recognized already syntactically. Also, the WDBA generated by 1t12tgba is smaller than the UBA generated (without WDBA minimization), e.g., for $k = 9$ the (complete) WDBA has 12 states, whereas the (complete) UBA has 57 states.

7 Conclusion

The main contribution of this paper is a novel LTL-to-UBA translation that uses alternating automata as an intermediate representation (in contrast to the approaches in [7,13,15]). To this end, we introduce a notion of unambiguity for alternating automata, and show that the VWAA-to-NBA translation of [18] preserves it (that is, it produces a UBA if the VWAA is unambiguous). We determine the complexity of checking unambiguity for VWAA to be PSPACE-complete, both for the standard symbolic and explicit encodings of alternating automata. To show hardness for the explicit case we show that the emptiness problem for explicitly encoded VWAA is PSPACE-hard.

The core of the LTL-to-UBA translation is an iterative disambiguation procedure for VWAA which identifies “ambiguous” states in the VWAA and applies local disambiguation transformations (that use alternation) to them. We introduce a heuristic that exploits structural properties of VWAA states for disambiguation, and another one that is aimed at choosing good local transformations. Furthermore, we identify LTL rewriting rules that are beneficial for our construction, and introduce a post-disambiguation heuristic on the level of NBA that allows us to remove certain states from the automaton.

Experimental analysis on a big LTL benchmark set shows that our tool *Duggi* is competitive compared to existing tools in terms of the sizes of produced automata. Formulas containing nested $\Diamond$ and $\Box$ in particular benefit from our heuristics and rewrite rules. Such formulas occur often, for example when modelling fairness properties. Experiments on Markov chain model checking indicate that the positive properties of our approach carry over to this domain.

Further possibilities of optimizing our approach are, for example, to process multiple “ambiguous” states of the VWAA at once, or in a certain order. This could lead to fewer iterations of our procedure, and thus decrease running times. It would be interesting to investigate intermediate strategies in our framework that allow for a trade-off between automata sizes and computation times. Another direction for further study is to identify other kinds of local disambiguation transformations for VWAA and to find more patterns that allow specialised constructions. Also, as many interesting properties are safety or cosafety languages, a combination of our approach with the disambiguation techniques for automata on finite words in [32] could be explored. The application of simulation-based automata reductions to UBA is also an open question. Whereas bisimulation preserves unambiguity, simulation relations targeted specifically at shrinking unambiguous automata have not been studied so far.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Artale A, Kontchakov R, Ryzhikov V, Zakharyashev M (2013) The complexity of clausal fragments of LTL. In: McMillan K, Middeldorp A, Voronkov A (eds) *Logic for programming, artificial intelligence, and reasoning*, pp 35–52. Lecture notes in computer science. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-45221-5_3
2. Babiak T, Blahoudek F, Duret-Lutz A, Klein J, Křetínský J, Müller D, Parker D, Strejček J (2015) The Hanoi omega-automata format. In: *Proceedings of the 27th international conference on computer aided verification (CAV)*. Lecture notes in computer science, vol 9206. Springer, pp 479–486
3. Babiak T, Křetínský M, Řehák V, Strejček J (2012) LTL to Büchi automata translation: fast and more deterministic. In: *Proceedings of the 18th international conference on tools and algorithms for the construction and analysis of systems (TACAS)*. Lecture notes in computer science, vol 7214. Springer, pp 95–109
4. Baier C, Katoen JP (2008) *Principles of model checking*. MIT Press, London
5. Baier C, Kiefer S, Klein J, Klüppelholz S, Müller D, Worrell J (2016) Markov chains and unambiguous Büchi automata. In: *Proceedings of the 28th international conference on computer aided verification (CAV)—part I*. Lecture notes in computer science, vol 9779. Springer, pp 23–42
6. Bauland M, Schneider T, Schnoor H, Schnoor I, Vollmer H (Jan 2009) The complexity of generalized satisfiability for linear temporal logic. *Log Methods Comput Sci* 5(1):1 [https://doi.org/10.2168/LMCS-5\(1:1\)2009](https://doi.org/10.2168/LMCS-5(1:1)2009)
7. Benedikt M, Lenhardt R, Worrell J (2013) LTL model checking of interval Markov chains. In: *Proceedings of the 19th international conference on tools and algorithms for the construction and analysis of systems (TACAS)*. Lecture notes in computer science, vol 7795. Springer, pp 32–46

8. Bousquet N, Löding C (2010) Equivalence and inclusion problem for strongly unambiguous Büchi automata. In: Proceedings of the 4th international conference on language and automata theory and applications (LATA). Lecture notes in computer science, vol 6031. Springer, pp 118–129
9. Carton O, Michel M (2003) Unambiguous Büchi automata. *Theor Comput Sci* 297(1–3):37–81
10. Clarke EM, Grumberg O, Peled DA (2001) Model checking. MIT Press, London
11. Colcombet T (2015) Unambiguity in automata theory. In: Proceedings of the 17th international workshop on desriptional complexity of formal systems (DCFS). Lecture notes in computer science, vol 9118. Springer, pp 3–18
12. Couvreur J (1999) On-the-fly verification of linear temporal logic. In: Proceedings of the world congress on formal methods in the development of computing systems (FM). Lecture notes in computer science, vol 1708. Springer, pp 253–271
13. Couvreur J, Saheb N, Sutre G (2003) An optimal automata approach to LTL model checking of probabilistic systems. In: Proceedings of the 10th international conference on logic for programming artificial intelligence and reasoning (LPAR). Lecture notes in computer science, vol 2850. Springer, pp 361–375
14. Duret-Lutz A (2013) Manipulating LTL formulas using Spot 1.0. In: Proceedings of the 11th international symposium on automated technology for verification and analysis (ATVA). Lecture notes in computer science, vol 8172. Springer, pp 442–445
15. Duret-Lutz A (2017) Contributions to LTL and ω -automata for model checking. Habilitation thesis, Université Pierre et Marie Curie (Paris 6)
16. Duret-Lutz A, Lewkowicz A, Fauchille A, Michaud T, Renault E, Xu L (Oct 2016) Spot 2.0—a framework for LTL and ω -automata manipulation. In: Proceedings of the 14th international symposium on automated technology for verification and analysis (ATVA). Lecture notes in computer science, vol 9938. Springer, pp 122–129
17. Etessami K, Holzmann G (2000) Optimizing Büchi automata. In: Proceedings of the 11th international conference on concurrency theory (CONCUR). Lecture notes in computer science, vol 1877. Springer, pp 153–167
18. Gastin P, Oddoux D (2001) Fast LTL to Büchi automata translation. In: Proceedings of the 13th international conference on computer aided verification (CAV). Lecture notes in computer science, vol 2102. Springer, pp 53–65
19. Gerth R, Peled D, Vardi MY, Wolper P (1995) Simple on-the-fly automatic verification of linear temporal logic. In: Proceedings of the 15th IFIP WG6.1 international symposium on protocol specification (PSTV). IFIP conference proceedings, vol 38. Chapman & Hall, pp 3–18
20. Grädel E, Thomas W, Wilke T (eds) (2002) Automata, logics, and infinite games: a guide to current research. Lecture notes in computer science, vol 2500. Springer
21. Hahn EM, Li G, Schewe S, Turrini A, Zhang L (2015) Lazy probabilistic model checking without determinisation. In: 26th international conference on concurrency theory (CONCUR 2015). Leibniz international proceedings in informatics (LIPIcs), vol 42, pp 354–367. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany
22. Haverkort BR, Hermanns H, Katoen JP (2000) On the use of model checking techniques for dependability evaluation. In: 19th IEEE symposium on reliable distributed systems (SRDS). IEEE Computer Society, pp 228–237
23. Isaak D, Löding C (2012) Efficient inclusion testing for simple classes of unambiguous ω -automata. *Inf Process Lett* 112(14–15):578–582
24. Jantsch S, Müller D, Baier C, Klein J (2019) From LTL to unambiguous Büchi automata via disambiguation of alternating automata. In: ter Beek MH, McIver A, Oliveira JN (eds) Formal methods—the next 30 years. Lecture notes in computer science. Springer, Cham, pp. 262–279. https://doi.org/10.1007/978-3-030-30942-8_17.
25. Kleine Büning H, Subramani K, Zhao X (2007) Boolean functions as models for quantified Boolean formulas. *J Autom Reason* 39(1):49–75. <https://doi.org/10.1007/s10817-007-9067-0>
26. Kretínský J, Meggendorfer T, Sickert S (2018) LTL store: Repository of LTL formulae from literature and case studies. CoRR abs/1807.03296. [arXiv:1807.03296](https://arxiv.org/abs/1807.03296)
27. Kupferman O, Rosenberg A (2011) The blow-up in translating LTL to deterministic automata. In: Proceedings of the 6th international workshop on model checking and artificial intelligence (MoChArt). Lecture notes in computer science, vol 6572. Springer, pp 85–94
28. Kupferman O, Vardi MY (2005) From linear time to branching time. *Trans Comput Logic* 6(2):273–294
29. Kwiatkowska MZ, Norman G, Parker D (2012) The PRISM benchmark suite. In: Proceedings of the 9th international conference on quantitative evaluation of systems (QEST). IEEE Computer Society, pp 203–204
30. Löding C (2001) Efficient minimization of deterministic weak ω -automata. *Inf Process Lett* 79(3):105–109

31. Löding C, Thomas W (2000) Alternating automata and logics over infinite words. In: Theoretical computer science: exploring new frontiers of theoretical informatics IFIP TCS, pp 521–535
32. Mohri M (2013) On the disambiguation of finite automata and functional transducers. *Int J Found Comput Sci* 24(6):847–862
33. Müller D, Sickert S (2017) LTL to deterministic Emerson-Lei automata. In: Proceedings of the 8th international symposium on games, automata, logics and formal verification (GandALF). Electronic proceedings in theoretical computer science, vol. 256. Open Publishing Association, pp 180–194
34. Muller DE, Saoudi A, Schupp PE (1988) Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In: Proceedings of the third annual symposium on logic in computer science (LICS), pp 422–427
35. Muller DE, Schupp PE (1987) Alternating automata on infinite trees. *Theoret Comput Sci* 54:267–276
36. Muller DE, Schupp PE (1995) Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, Mcnaughton and Safra. *Theor Comput Sci* 141(1 & 2):69–107
37. Muscholl A, Walukiewicz I (2005) An NP-complete fragment of LTL. In: Calude CS, Calude E, Dinneen MJ (eds) Developments in language theory. Lecture notes in computer science. Springer, Heidelberg, pp 334–344. https://doi.org/10.1007/978-3-540-30550-7_28
38. Safra S (1988) On the complexity of ω -automata. In: Proceedings of the 29th symposium on foundations of computer science (FOCS). IEEE Computer Society, pp 319–327
39. Schobbens PY, Raskin JF (1999) The logic of “initially” and “next”: complete axiomatization and complexity. *Inform Process Lett* 69(5):221–225 [https://doi.org/10.1016/S0020-0190\(99\)00022-8](https://doi.org/10.1016/S0020-0190(99)00022-8)
40. Sistla AP, Clarke EM (1985) The complexity of propositional linear temporal logics. *J ACM (JACM)* 32(3):733–749. <https://doi.org/10.1145/3828.3837>
41. Somenzi F, Bloem R (2000) Efficient Büchi automata from LTL formulae. In: Proceedings of the 12th international conference on computer aided verification (CAV). Lecture notes in computer science, vol. 1855. Springer, pp 248–263
42. Stearns RE, Hunt HB (1985) On the equivalence and containment problem for unambiguous regular expressions, grammars, and automata. *SIAM J Comput*, pp 598–611
43. Vardi MY (1985) Automatic verification of probabilistic concurrent finite-state programs. In: Proceedings of the 26th IEEE symposium on foundations of computer science (FOCS). IEEE Computer Society, pp 327–338
44. Vardi MY (1994) Nontraditional applications of automata theory. In: Proceedings of the international conference on theoretical aspects of computer software (TACS), pp 575–597
45. Vardi MY, Wolper P (1986) An automata-theoretic approach to automatic program verification (preliminary report). In: Proceedings of the 1st symposium on logic in computer science (LICS). IEEE Computer Society Press, pp 332–344

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.