From LTL to Unambiguous Büchi Automata via Disambiguation of Alternating Automata

Simon Jantsch$^{1[0000-0003-1692-2408]}$, David Müller$^{1[0000-0002-5384-9644]}$, Christel Baier$^{1[0000-0002-5321-9343]}$, and Joachim Klein$^{1[0000-0003-4681-6964]}$

Technische Universität Dresden, Germany ⋆

Abstract. This paper proposes a new algorithm for the generation of unambiguous Büchi automata (UBA) from LTL formulas. Unlike existing tableau-based LTL-to-UBA translations, our algorithm deals with very weak alternating automata (VWAA) as an intermediate representation. It relies on a new notion of unambiguity for VWAA and a disambiguation procedure for VWAA. We introduce optimizations on the VWAA level and new LTL simplifications targeted at generating small UBA. We report on an implementation of the construction in our tool `Duggi` and discuss experimental results that compare the automata sizes and computation times of `Duggi` with the tableau-based LTL-to-UBA translation of the `SPOT` tool set. Our experiments also cover the analysis of Markov chains under LTL specifications, which is an important application of UBA.

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 (see, e.g. [37,4,9]). In the worst case, such translations produce automata that are exponentially larger than the input formula. However, a lot of effort has been put into optimizing the general case, which has turned LTL-to-NBA translations feasible in practice. Two classes of algorithms have emerged as being especially well suited: tableau-based decomposition of the LTL formula into an automaton (see, e.g. [18,11]), as represented by the `SPOT` family of tools [15], and translations via very weak alternating automata (VWAA) [17], where `LTL3BA` [3] is the leading tool currently.

A property that has been studied in many areas of automata theory is *unambiguity* [10]. It allows non-deterministic branching but requires that each input word has at most one accepting run. Prominent cases in which unambiguity can be utilized include the universality check for automata ("Is every word accepted?") on finite words, which is PSPACE-complete for arbitrary non-deterministic finite

⋆ 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).

automata (NFA), but in P for unambiguous finite automata (UFA) [34]. Another example is model checking of Markov chains, which is in P if the specification is given as an unambiguous Büchi automaton (UBA) [5], and PSPACE-hard for arbitrary NBA [35]. Thus, using UBA leads to 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.

Every ω-regular language is expressible by UBA [1], but NBA may be exponentially more succinct than UBA [24] and UBA may be exponentially more succinct than any deterministic automaton [7]. Universality and language inclusion are in P for subclasses of UBA [7,22], but the complexity is open for general UBA.

Although producing UBA was not the goal of the early translation from LTL to NBA by Vardi and Wolper [37], their construction is asymptotically optimal and produces *separated* automata, a subclass of UBA where the languages of the states are pairwise disjoint. Separated automata can express all ω-regular languages [8], but UBA may be exponentially more succinct [7]. LTL-to-NBA translations have been studied intensively [17,18,16,13], but the generation of UBA from LTL formulas has not received much attention so far. We are only aware of three approaches targeted explicitly at generating UBA or subclasses. The first approach by Couvreur et al. [12] adapts the algorithm of [37], 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. [6] and Duret-Lutz [14]. These adapt tableau-based LTL-to-NBA algorithms ([18] in the case of [6] and [11] in the case of [14]) 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, thus ensuring unambiguity.

To the best of our knowledge, the only available tool that supports the translation of LTL formulas to UBA is `ltl2tgba`, which is part of the `SPOT` tool set and implements the LTL-to-UBA algorithm of [14].

You can find an extended version of this paper at [23] with further details and proofs.

Contribution. We describe a novel LTL-to-UBA construction. It relies on an intermediate representation of LTL formulas using VWAA and adapts the known translation from VWAA to NBA by Gastin and Oddoux [17]. We introduce a notion of unambiguity for VWAA, show that the subsequent translation steps preserve it and that checking whether a VWAA is unambiguous is PSPACE-complete (Section 3). To the best of our knowledge, unambiguity for alternating automata has not been considered before.

We present a disambiguation procedure for VWAA that relies on intermediate unambiguity checks to identify ambiguous states and local disambiguation transformations for the VWAA (Section 4). It has the property that an already unambiguous VWAA is not changed. Figure 1 gives an overview of our LTL-to-UBA algorithm. Apart from the main construction, we introduce novel LTL rewrite rules and a heuristic, both of which are aimed at producing small UBA and may also benefit existing tools (see Figure 2). The heuristic is targeted at states with a certain structure, defined using the concepts of *purely-universal*

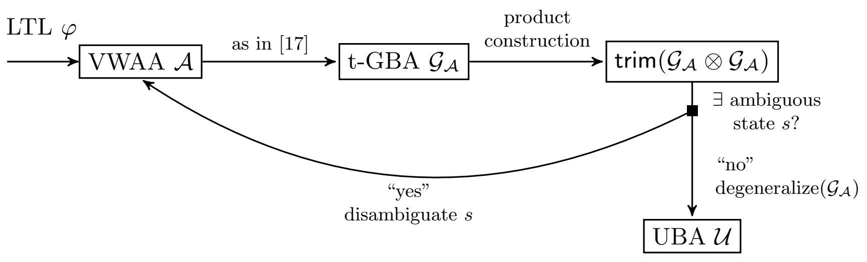

LTL φ
VWAA $\mathcal{A}$
as in [17]
t-GBA $\mathcal{G}_\mathcal{A}$
product construction
$\mathsf{trim}(\mathcal{G}_\mathcal{A} \otimes \mathcal{G}_\mathcal{A})$
∃ ambiguous state s?
"no" degeneralize($\mathcal{G}_\mathcal{A}$)
"yes" disambiguate s
UBA $\mathcal{U}$

Fig. 1: The LTL-to-UBA step. A sequence of unambiguity checks and disambiguation transformations are applied and ultimately a UBA is returned. We use $\mathsf{trim}(\mathcal{G}_\mathcal{A} \otimes \mathcal{G}_\mathcal{A})$ to check whether unambiguity is achieved or more iterations are necessary.

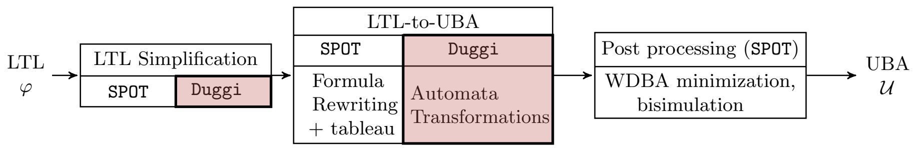


Fig. 2: Overview of the general LTL-to-UBA generation algorithm. The LTL simplification step, the actual LTL-to-UBA translation step, and the automaton post processing step can be combined freely. We propose novel rewriting rules for LTL and a LTL-to-UBA translation, both implemented in our tool `Duggi`.

and *alternating* formulas (Section 5). Finally, we report on an implementation of our construction in our 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 (Section 6).

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 $w[i]w[i{+}1]\ldots$. We write $\mathcal{B}^+(X)$ to denote the set of positive Boolean formulas over a finite set of variables X. A *minimal model* of a formula $f \in \mathcal{B}^+(X)$ is a set $M \subseteq X$ such that $M \models f$, but no $M' \subset M$ satisfies $M' \models f$. LTL is defined using $\mathcal{U}$ ("Until")and $\bigcirc$ ("Next"). Additionally we use syntactical derivations $\Diamond$ ("Finally"), $\Box$ ("Globally"), and $\mathcal{R}$ ("Release") (see [4,19] for details).

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, $\Delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is the transition function, $\iota \in \mathcal{B}^+(Q)$ is the initial condition and Φ is the acceptance condition. Additionally, we define the function $\delta : Q \times \Sigma \to 2^{2^Q}$ which assigns to a pair $(q, a) \in Q \times \Sigma$ the set of minimal models of $\Delta(q,a)$ and the set $I \subseteq 2^Q$ as the set of minimal models of ι. We denote by $\mathcal{A}(\iota')$ the automaton $(Q, \Sigma, \delta, \iota', \Phi)$ and we write $\mathcal{A}(Q_0)$ for $\mathcal{A}(\bigwedge_{q \in Q_0} q)$, if $Q_0 \subseteq Q$. We call the number of the reachable states of an automaton $\mathcal{A}$ its size.

A run of $\mathcal{A}$ for $w \in \Sigma^\omega$ is a directed acyclic graph (DAG) (V, E) [28], where

1. $V \subseteq Q \times \mathbb{N}$, and $E \subseteq \bigcup_{0 \leq l} (Q \times \{l\}) \times (Q \times \{l+1\})$,
2. $\{q : (q, 0) \in V\} \in I$,
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. A run is called accepting if every infinite path in it meets the acceptance condition.

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 $\mathcal{L}(\mathcal{A})$. We distinguish between Büchi, generalized Büchi and co-Büchi acceptance conditions. A Büchi condition is denoted by $\mathrm{Inf}(Q_f)$ for a set $Q_f \subseteq Q$. An infinite path $\pi = q_0\, q_1 \ldots$ meets $\mathrm{Inf}(Q_f)$ if $Q_f \cap \mathrm{inf}(\pi) \neq \varnothing$, where $\mathrm{inf}(\pi)$ denotes the set of infinitely occurring states in π. A co-Büchi condition is denoted by $\mathrm{Fin}(Q_f)$ and π meets $\mathrm{Fin}(Q_f)$ if $Q_f \cap \mathrm{inf}(\pi) = \varnothing$. An infinite path π meets a generalized Büchi condition $\bigwedge_{i \in F} \mathrm{Inf}(Q_i)$ if it meets $\mathrm{Inf}(Q_i)$ for all $i \in F$. 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.

We call a subset $C \subseteq Q$ a configuration and say that C is reachable if it is a layer of some run. A configuration C is reachable from a state q, also written as $q \longrightarrow^* C$, if C is a reachable configuration of $\mathcal{A}(q)$. Analogously, $C' \subseteq Q$ is reachable from $C \subseteq Q$, or $C \longrightarrow^* C'$, if C' is a reachable configuration of $\mathcal{A}(C)$. A configuration C is *reachable via u* if there is a run (V, E) for a word uw, with $u \in \Sigma^*, w \in \Sigma^\omega$, such that $C = V(|u|)$. We extend this notion to reachability from states and configurations via finite words in the expected way and write $q \xrightarrow{u}{}^* C'$ and $C \xrightarrow{u}{}^* C'$. We define $\mathcal{L}(C) = \mathcal{L}(\mathcal{A}(C))$.

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 $\mathcal{A}$ has a co-Büchi acceptance. If $|C_0| = 1$ for every $C_0 \in I$, 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 $\mathcal{L}(q) \neq \varnothing$ holds for every state q in $\mathcal{A}$, and we write $\mathsf{trim}(\mathcal{A})$ 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, $\mathsf{trim}(\mathcal{A})$ can be computed in linear time using standard graph algorithms.

From LTL to NBA. 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 [36,31]. It has been used as a first step in an LTL-to-NBA translation in [17], whose construction we follow. Additionally, we use the optimizations proposed in [3]. We also maintain the following invariant, as proposed in [17]: for all $(q,a) \in Q \times \Sigma$ and successor sets $S_1, S_2 \in \delta(q,a)$, such that $S_1 \neq S_2$, it holds that $S_1 \not\subseteq S_2$.

A VWAA $\mathcal{A}$ can be transformed into a transition-based generalized Büchi automaton (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 1. *Let $\mathcal{A} = (Q, \Sigma, \Delta, \iota, \mathrm{Fin}(Q_f))$ be a VWAA. The t-GBA $\mathcal{G}_{\mathcal{A}}$ is the tuple $(2^Q, \Sigma, \delta', I, \bigwedge_{f \in Q_f} \mathrm{Inf}(\mathcal{T}_f))$, where*

- $\delta'(C, a) = \bigotimes_{q \in C} \delta(q, a)$, *where* $T_1 \otimes T_2 = \{C_1 \cup C_2 \,:\, C_1 \in T_1, C_2 \in T_2\}$
- $\mathcal{T}_f = \{(C, a, C') \,:\, f \notin C'$ *or there exists* $Y \in \delta(f, a)$ *and* $f \notin Y \subseteq C'\}$

Theorem 2 (Theorem 2 of [17]). *Let $\mathcal{A}$ be a VWAA and $\mathcal{G}_{\mathcal{A}}$ be as in Definition 1. 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 [17], which translates $\mathcal{G}_{\mathcal{A}}$ into an NBA $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$ of at most $|Q_f| \cdot 2^{|Q|}$ reachable states.

3 Unambiguous VWAA

In this section we introduce a notion of unambiguity for VWAA and show that unambiguous VWAA are translated to UBA by the translation presented in Section 2. We define unambiguity in terms of configurations of the VWAA, which are strongly related to the states of the resulting NBA. Let $\mathcal{A} = (Q, \Sigma, \Delta, \iota, \mathrm{Fin}(Q_f))$ be a fixed VWAA for the rest of this section.

Definition 3. *$\mathcal{A}$ is unambiguous if it has no distinct configurations C_1, C_2 that are reachable via the same word $u \in \Sigma^*$ and such that $\mathcal{L}(C_1) \cap \mathcal{L}(C_2) \neq \varnothing$.*

The standard definition of unambiguity is that an automaton is unambiguous if it has 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.

Theorem 4. *Let $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$ be the NBA for $\mathcal{A}$, obtained by the translation from Section 2. If $\mathcal{A}$ is unambiguous, then $\mathcal{N}_{\mathcal{G}_{\mathcal{A}}}$ is unambiguous.*

We show that every step in the translation from VWAA to NBA preserves unambiguity. First, we establish the following correspondance:

Lemma 5. *If $\mathcal{A}$ is unambiguous, then for every accepting run $r = Q_0 Q_1 \ldots$ of $\mathcal{G}_{\mathcal{A}}$ 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 $i \geq 0$.*

Table 1: The adapted expansion laws for $\mathcal{U}$ and $\mathcal{R}$ are the result of applying the disjunction rule to the classic expansion laws.

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

Intuitively, the lemma states that if $\mathcal{A}$ is unambiguous, then every accepting run r of $\mathcal{G}_{\mathcal{A}}$ can be matched by an accepting run ρ of $\mathcal{A}$ such that the states of r are the layers of ρ. The proof is not immediate and requires $\mathcal{A}$ to be unambiguous.

A direct consequence of Lemma 5 is that if $\mathcal{A}$ is unambiguous, then so is $\mathcal{G}_{\mathcal{A}}$. The degeneralization construction in [17] makes $|Q_f| + 1$ copies of $\mathcal{G}_{\mathcal{A}}$. As the next copy is uniquely determined by the current state and word label, it preserves unambiguity. In combination with Lemma 7 we obtain Theorem 4.

We now show that deciding whether a VWAA is unambiguous is PSPACE-complete. The idea for proving hardness is to reduce LTL satisfiability, which is known to be PSPACE-hard, to VWAA emptiness (this follows directly by the LTL $\rightarrow$ VWAA translation) and VWAA emptiness to VWAA unambiguity. The second step uses the following trick: a VWAA $\mathcal{A}$ accepts the empty language if and only if the disjoint union of $\mathcal{A}$ with itself is unambiguous.

To check wether VWAA is unambiguous we first show that for every accepting run of $\mathcal{A}$, we find a matching accepting run of $\mathcal{G}_{\mathcal{A}}$, which follows directly from the definition of $\mathcal{G}_{\mathcal{A}}$:

Lemma 6. *For every accepting run $\rho = (V, E)$ of $\mathcal{A}$ for $w \in \Sigma^\omega$ there exists an accepting run $r = Q_0 Q_1 \ldots$ of $\mathcal{G}_{\mathcal{A}}$ for w, such that $Q_i = V(i)$ for all $i \geq 0$.*

Lemma 5 and Lemma 6 give us the following:

Lemma 7. *$\mathcal{A}$ is unambiguous if and only if $\mathcal{G}_{\mathcal{A}}$ is unambiguous.*

However, checking whether $\mathcal{G}_{\mathcal{A}}$ is unambiguous can be done in space polynomial in the size of $\mathcal{A}$, and we conclude:

Theorem 8. *Deciding whether a VWAA is unambiguous 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 (see Table 1). 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 [6].

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.

If no ambiguity is found in step 2, the VWAA is unambiguous. The high-level overview is also depicted in Figure 1. In what follows we fix a VWAA $\mathcal{A} = (Q, \Sigma, \Delta, \iota, \mathrm{Fin}(Q_f))$ and assume that it has a single initial state.

Complement states. The transformations we apply for disambiguation rely on the following precondition: for every state s of $\mathcal{A}$ there should be another state $\tilde{s}$ such that $\mathcal{L}(\tilde{s}) = \overline{\mathcal{L}(s)}$. We compute these complement states in a preprocessing step and add them to $\mathcal{A}$. Complementing alternating automata can be done without any blow up by dualizing both the acceptance condition and transition structure, as shown by Muller and Schupp [32]. 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.

The complement automaton has the same underlying graph and is therefore also very weak. Furthermore, no state s is reachable from its own complement state $\tilde{s}$, which is an invariant that we maintain and which ensures that very weakness is preserved in the construction.

Source configurations and source states. To characterize ambiguous situations we define *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 $\mathcal{L}(C_1) \cap \mathcal{L}(C_2) \neq \varnothing$. By definition, $\mathcal{A}$ is not unambiguous if a source configuration exists.

Let C be a source configuration of $\mathcal{A}$ and let C_1, C_2 be the successor configurations as described above. A source state of C 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 $(S_1 \cup S_2) \setminus (C_1 \cap C_2) \neq \varnothing$. The last condition ensures that either S_1 or S_2 contains a state that is not common to C_1 and C_2. By Definition 1, $C_i = \bigcup_{q \in C} S_q$ with $S_q \in \delta(a, q)$ for all $q \in C$, and thus C must contain a source state.

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 $\mathsf{trim}(\mathcal{G}_{\mathcal{A}} \otimes \mathcal{G}_{\mathcal{A}})$ with $C_1 \neq C_2$.

The pair of configurations (C_1, C_2) is a witness to ambiguity of $\mathcal{A}$. We look for a symbol $a \in \Sigma$ and a configuration C such that $(C, C) \xrightarrow{a} (C_1', C_2') \rightarrow^* (C_1, C_2)$

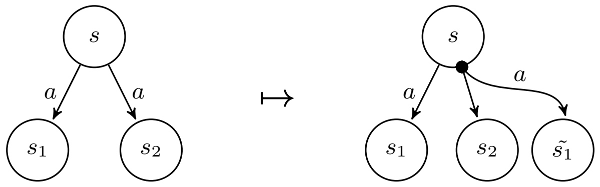


Fig. 3: Disambiguation scheme for a source state s with successors s_1 and s_2 in the VWAA. Transitions with successor set of size ≥ 1 are conjoined by a $\bullet$.

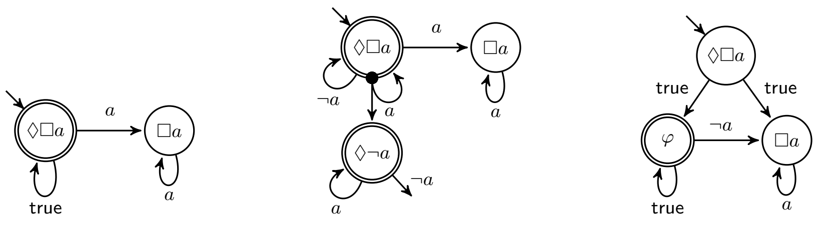
is a path in $\mathsf{trim}(\mathcal{G}_\mathcal{A} \otimes \mathcal{G}_\mathcal{A})$ and $C'_1 \neq C'_2$. Such a configuration must exist as we have assumed that $\mathcal{A}$ has a single initial state q_i, which implies that $\mathsf{trim}(\mathcal{G}_\mathcal{A} \otimes \mathcal{G}_\mathcal{A})$ has a single initial state $(\{q_i\}, \{q_i\})$. C is a source configuration and therefore must contain a source state which we can find by inspecting all pairs of transitions of states in C.

Disambiguating a source state. The general scheme for disambiguating source states is depicted in Figure 3. Assume that we have identified a source state s with successor sets S_1 and S_2 as explained above. The LTL-to-VWAA construction 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 the $|S_1|$ 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 $\delta(s, a)$.

This transformation does not guarantee that s is not a source state anymore. However, it removes the ambiguity that stems from the non-deterministic choice of transitions $S_1, S_2 \in \delta(a, s)$. 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. The transformation preserves very weakness as it only adds transitions from s to complement states of successors of s and by assumption there is no path between a state and its complement state.

Iterative algorithm. Putting things together, our algorithm works as follows: it searches for source configurations of $\mathcal{A}$ (using $\mathcal{G}_\mathcal{A}$), applies the local disambiguation transformations to $\mathcal{A}$ as described and recurses (see Figure 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.

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 $\mathcal{A}$ of size n. In our disambiguation procedure we enlarge $\mathcal{A}$ by adding complement states for every state in the original automaton, yielding a VWAA of size $2n$. Thus, a first



(a) VWAA for $\Diamond\Box a$.

(b) Standard disambiguation.

(c) Modified transformation. Here $\varphi = \Diamond(\neg a \wedge \bigcirc\Box a)$.

Fig. 4: 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.

size estimate of $\mathcal{G}_{\mathcal{A}}$ in our construction is 4^n. However, no state in $\mathsf{trim}(\mathcal{G}_{\mathcal{A}})$ 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 $\mathcal{L}(s) \cap \mathcal{L}(\tilde{s}) = \varnothing$. Thus, $\mathsf{trim}(\mathcal{G}_{\mathcal{A}})$ 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. Thus, our procedure computes a UBA in time exponential in the length of the formula.

5 Heuristics for purely-universal formulas

In this section we introduce alternative disambiguation transformations for special source states representing formulas $\varphi\mathcal{U}\nu$, where ν 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 ν holds whenever moving to a state representing $\varphi\mathcal{U}\nu$ and, if not, finding the *last* position where it does not hold.

Example 9. Consider the formula $\Diamond\Box a$. A VWAA for it is shown in Figure 4a. It is ambiguous, as a word satisfying $\Box a$ may loop in the initial state for an arbitrary amount of steps before moving to the next state.

In the standard disambiguation transformation the state $\Diamond\neg a$ is added to the self loop of the initial state (Figure 4b). The automaton in Figure 4c, 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 $\bigcirc\Box a$. 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 *purely-universal* (ν), *purely-eventual*

(μ) and *alternating* (ξ) formulas ([16,3]). In the following definition φ ranges over arbitrary LTL formulas:

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

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

Lemma 10 ([16,3]). *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 $\mathfrak{g}(\nu)$, called the *goal* of ν, that is simpler than ν and satisfies $\mathfrak{g}(\nu) \wedge \bigcirc\nu \equiv \nu$. 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 $\neg\mathfrak{g}(\nu) \wedge \bigcirc\nu$.

It is not clear how to define $\mathfrak{g}(\nu)$ 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 occurences of $\mathcal{R}$ are also in the scope of some $\Box$ for purely-universal formulas.

Lemma 11. *Every purely-universal formula ν can be rewritten into a formula $\nu_1 \vee \ldots \vee \nu_n$, where ν_i is disjunction-free for all $1 \leq i \leq n$.*

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

Definition 12. *Let ν be a disjunction-free and purely-universal formula. We define $\mathfrak{g}(\nu)$ inductively as follows:*

$$\begin{aligned}
\mathfrak{g}(\Box\varphi) &= \varphi & \mathfrak{g}(\bigcirc\nu) &= \bigcirc\mathfrak{g}(\nu) \\
\mathfrak{g}(\nu_1 \wedge \nu_2) &= \mathfrak{g}(\nu_1) \wedge \mathfrak{g}(\nu_2) & \mathfrak{g}(\Diamond\nu) &= \mathsf{true}
\end{aligned}$$

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

Lemma 13. *Let ν be a disjunction-free and purely-universal formula. Then $\mathfrak{g}(\nu) \wedge \bigcirc\nu \equiv \nu$.*

In Example 9 $\neg\mathfrak{g}(\nu) \wedge \bigcirc\nu$ corresponds to $\neg a \wedge \bigcirc\Box a$, which is realized by the transition from state φ to state $\Box a$ in Figure 4c.

Lemma 14 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 14. *Let ν be a disjunction-free and purely-universal formula. Then*

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

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

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. If all occurrences of $\bigcirc$ in ν are in the scope of some $\Box$, then no subformulas are added. Otherwise, $\mathfrak{g}(\nu)$ and $\bigcirc\nu$ may introduce new $\bigcirc$-subformulas.

6 Implementation and Experiments

The tool `Duggi` is an LTL-to-UBA translator based on the construction introduced in the foregoing sections.[1] 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 $\tilde{s}$ 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 [33,16,3,30]. We have included the following rules that are not used by `SPOT`:

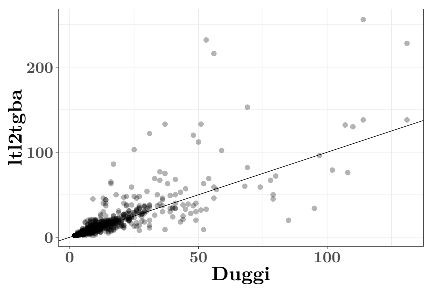
  I $(\Box\Diamond\varphi) \wedge (\Diamond\Box\psi) \mapsto \Box\Diamond(\varphi \wedge \Box\psi)$ II $(\Diamond\Box\varphi) \vee (\Box\Diamond\psi) \mapsto \Diamond\Box(\varphi \vee \Diamond\psi)$

  These rewrite rules are more likely to produce formulas of the form $\Diamond\Box\varphi$, to which the heuristic of Section 5 can be applied. They stem from [30], where the reversed rules have been used to achieve a normal form.

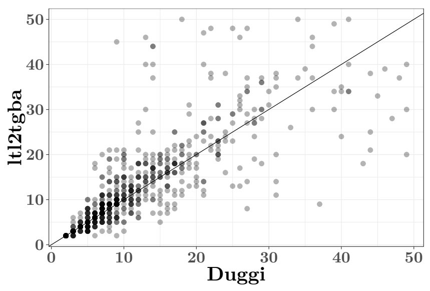
LTL benchmarks from the literature. We now compare the UBA sizes for LTL formulas of the benchmark set LTLSTORE [25]. It collects formulas from various case studies and tool evaluation papers in different contexts. We include the negations of all formulas and filter out duplicates, leaving 1419 formulas.

Languages that are recognizable by *weak deterministic Büchi automata* (WDBA) can be efficiently minimized [27] and `ltl2tgba` applies this algorithm as follows: it computes the minimal deterministic Büchi automaton and the UBA

[1] `Duggi` and the `PRISM` implementation, together with all experimental data, are available at `https://wwwtcs.inf.tu-dresden.de/ALGI/TR/FM19-UBA/`.



(a) Entire set



(b) Instances where both tools produced automata with at most 50 states

Fig. 5: Non-WDBA-recognizable fragment of LTLSTORE (948 formulas). Every point stands for a formula where the according automaton size for `Duggi` is the abcissa, the automaton size of `ltl2tgba` the ordinate. Points above the line stand for formulas where `Duggi` performed better.

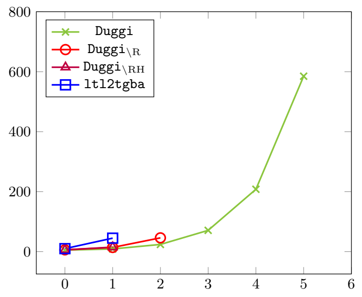
and returns the one with fewer states. Our formula set contains 472 formulas that are WDBA-recognizable and for which we could compute the minimal WDBA within the bounds of 30 minutes and 10 GB of memory using `ltl2tgba`. Of these 472 formulas we found 11 for which the UBA generated by either `Duggi` or `ltl2tgba` 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 `ltl2tgba` (`Duggi`) for 164 (203) formulas. This supports the approach by `ltl2tgba` to apply WDBA minimization when possible and in what follows we focus on the fragment of the LTLSTORE that does not fall into this class. In [14] it was noted that WDBA minimization often leads to smaller automata than the LTL-to-NBA translation of `ltl2tgba`.

We consider the following configurations: `Duggi` is the standard configuration, $\texttt{Duggi}_{\setminus(\mathrm{R},\mathrm{H})}$ is `Duggi` without the new rewrite rules I and II (R) and/or without the heuristic introduced in Section 5 (H). For `SPOT`, `ltl2tgba` is the standard configuration that produces UBA without WDBA-minimization, which is switched on in $\texttt{ltl2tgba}_{\mathrm{WDBA}}$. We use simulation-based postprocessing as provided by `SPOT` in all `Duggi`-configurations (they are enabled by default in `ltl2tgba`). We use `SPOT` with version 2.7.2. 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 10 GB.

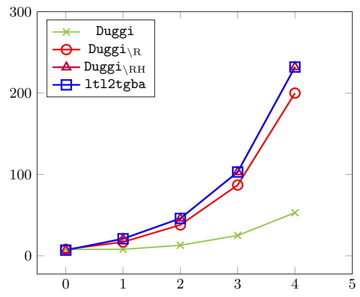
Scatter plots comparing the number of states of UBA produced by `ltl2tgba` and `Duggi` are shown in Figure 5. Table 2 gives cumulative results of different configurations on these formulas. All configurations of `Duggi` use more time than `ltl2tgba`, but produce smaller automata on average. One reason 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 per formula of `Duggi` on the entire LTLSTORE was 9.5.

Table 2: Cumulative results on the LTLSTORE benchmark set.

	non-WDBA-recognizable				WDBA-recognizable			
	states	∅ states	time in s	timeouts	states	∅ states	time in s	timeouts
Duggi	16,169	20.702	38,932	167	6,866	16.308	5,958	51
Duggi$_{\setminus\mathrm{R}}$	15,450	20.196	37,803	183	6,857	16.287	5,978	51
Duggi$_{\setminus\mathrm{RH}}$	14,415	19.323	39,772	202	6,882	16.346	5,758	51
ltl2tgba	19,547	24.618	6,089	154	9,250	20.240	3,965	15
ltl2tgba$_{\mathrm{WDBA}}$	19,411	24.539	7,309	157	7,632	16.700	3,814	15



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



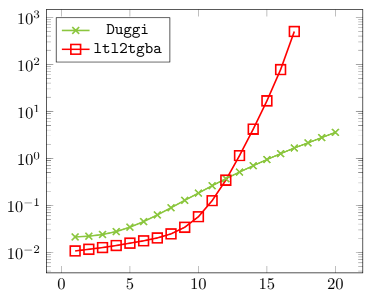
(b) $\theta_n = (\bigwedge_{i \leq n} \Box\Diamond p_i) \rightarrow \Box(req \rightarrow \Diamond res)$

Fig. 6: UBA sizes for two sets of parametrized formulas.

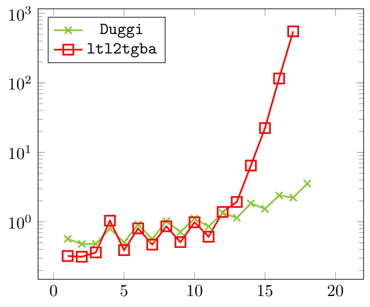
LTL rewrites and the purely-universal heuristic. A formula that benefits from using the rewrite rules I and II is $\Phi_n = \bigwedge_{i \leq n} \Diamond\Box p_{2i} \vee \Box\Diamond p_{2i+1}$, which describes a *strong fairness* condition. Here ltl2tgba applies the rule $\Diamond\varphi \vee \Box\Diamond\psi \mapsto \Diamond(\varphi \vee \Box\Diamond\psi)$ which yields $\bigwedge_{i \leq n} \Diamond(\Box p_{2i} \vee \Box\Diamond p_{2i+1})$. Applying rule II yields the formula $\Psi_n = \Diamond\Box(\bigwedge_{i \leq n} p_{2i} \vee \Diamond p_{2i+1})$. Figure 6a shows that Duggi produces smaller automata for Φ_n. Figure 6b 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 $\varphi_k^{\text{steps}} = \neg a \,\mathcal{U}\, \big(b \wedge \neg a \wedge \bigcirc\neg a \wedge \ldots \wedge \bigcirc^{k-1}\neg a \wedge \bigcirc^k a\big)$. Both Duggi and ltl2tgba produce the minimal UBA, but ltl2tgba 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 Figure 7a).

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 ltl2tgba as automata generation backends. We take two models of the PRISM benchmark suite [26], the bounded retransmission protocol, and the cluster working protocol [21].

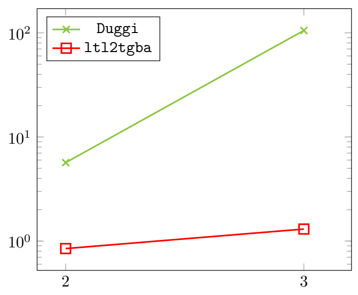


(a) Time in seconds needed for the translation of φ_k^{steps} into a UBA.

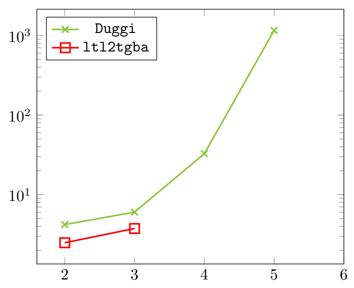


(b) Time in seconds needed for model checking the BRP model with φ_k^{steps}.

Fig. 7: Time consumption for translating and model checking φ_k^{steps} (which includes building the automaton).



(a) Time consumption for φ_k.



(b) Time consumption for ψ_k.

Fig. 8: Model checking times for the cluster protocol with φ_k and ψ_k.

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` (Figure 7b). The reason for the spikes in Figure 7b 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 [20]. It consists of a workstation cluster with two sub-clusters that are connected by a backbone and have $n = 16$ participants each. Let $\texttt{fct}_i$ denote the number of functional working stations in sub-cluster i. We define $\varphi_{\Box\Diamond} = \Box\Diamond(\texttt{fct}_1 = n)$, which expresses that the first cluster stays functional on the long run and $\varphi_{\Diamond\Box} = \bigvee_{i\in\{0,\ldots,k\}} \Diamond\Box(\texttt{fct}_2 = n - i)$, which expresses the property that from some point, the second cluster contains at least $n - k$ functional working stations. We check the two formula patterns $\varphi_k = \varphi_{\Box\Diamond} \wedge \varphi_{\Diamond\Box}$ and $\psi_k = \varphi_{\Box\Diamond} \vee \varphi_{\Diamond\Box}$. We leave out a third property described in [20], which is WDBA-recognizable (see the full version [23] for further details).

The results for φ_k are depicted in Figure 8a. Both tools have a time-out at $k = 4$, although, for smaller k, the time consumption of `Duggi` was bigger than `ltl2tgba`. Comparing the automata size, `Duggi` produces smaller automata for both $k = 2$ and $k = 3$, e.g., 32 (`Duggi`) vs. 137 (`ltl2tgba`) states for $k = 3$. The results for ψ_k can be seen in Figure 8b. `Duggi` performed better than `ltl2tgba`, as `Duggi` reached the time-out at $k = 6$ (vs. $k = 4$ for `ltl2tgba`). However, if no time-out was reached, `ltl2tgba` consumed less time. Nevertheless, for $k \leqslant 3$, model checking time of both tools was below 7 s. Still, `Duggi` produced smaller automata, e.g., 25 (`Duggi`) vs. 59 (`ltl2tgba`) states for $k = 3$.

7 Conclusion

In this paper we have presented a novel LTL-to-UBA translation. In contrast to other LTL-to-UBA translations [12,6,14] we use alternating automata as an intermediate representation. To adapt the VWAA-to-NBA construction of [17] for the unambiguity setting, we introduced a notion of unambiguity for VWAA and a corresponding disambiguation procedure. This may be of independent interest when considering unambiguity for different types of alternating automata. We devise heuristics that exploit structural properties of purely-universal and alternating formulas for disambiguation. Furthermore, we identify LTL rewriting rules that benefit the construction of UBA.

Experimental analysis on a big LTL benchmark set shows that our tool `Duggi` produces smaller automata on average than the existing tools. In particular, formulas containing nested $\Diamond$ and $\Box$ 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.

Our approach opens up many possibilities for optimization, for example by processing multiple source states at once, or in a certain order. This would let us decrease the number of disambiguation steps, and thus the run time. It would be interesting to investigate intermediate strategies in our framework that allow for a trade-off between automata sizes and computation times. Another promising direction is to identify more patterns on LTL or VWAA that allow special disambiguation transformations. As many interesting properties stem from the safety-/cosafety-class, a combination of our approach with the ideas of the UFA generation described in [29] seems to be beneficial. The application of simulation-based automata reductions to UBA is also an open question. Whereas bisimulation preserves unambiguity, it is unclear whether there exist simulation relations targeted specifically at shrinking unambiguous automata.

References

1. Arnold, A.: Deterministic and non ambiguous rational ω-languages. In: Automata on Infinite Words, Ecole de Printemps d'Informatique Théorique, Le Mont Dore, May 1984. Lecture Notes in Computer Science, vol. 192, pp. 18–27. Springer (1985)
2. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J., Müller, D., Parker, D., Strejček, J.: The Hanoi omega-automata format. In: Proceedings of the 27th International Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 9206, pp. 479–486. Springer (2015)
3. Babiak, T., Křetínský, M., Řehák, V., Strejček, J.: 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, pp. 95–109. Springer (2012)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
5. Baier, C., Kiefer, S., Klein, J., Klüppelholz, S., Müller, D., Worrell, J.: 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, pp. 23–42. Springer (2016)
6. Benedikt, M., Lenhardt, R., Worrell, J.: 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, pp. 32–46. Springer (2013)
7. Bousquet, N., Löding, C.: 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, pp. 118–129. Springer (2010)
8. Carton, O., Michel, M.: Unambiguous Büchi automata. Theor. Comput. Sci. **297**(1-3), 37–81 (2003)
9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (2001)
10. Colcombet, T.: Unambiguity in automata theory. In: Proceedings of the 17th International Workshop on Descriptional Complexity of Formal Systems (DCFS). Lecture Notes in Computer Science, vol. 9118, pp. 3–18. Springer (2015)
11. Couvreur, J.: 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, pp. 253–271. Springer (1999)
12. Couvreur, J., Saheb, N., Sutre, G.: 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, pp. 361–375. Springer (2003)
13. Duret-Lutz, A.: 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, pp. 442–445. Springer (2013)
14. Duret-Lutz, A.: Contributions to LTL and ω-Automata for Model Checking. Habilitation thesis, Université Pierre et Marie Curie (Paris 6) (Feb 2017)
15. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA). Lecture Notes in Computer Science, vol. 9938, pp. 122–129. Springer (Oct 2016)

16. Etessami, K., Holzmann, G.: Optimizing Büchi automata. In: Proceedings of the 11th International Conference on Concurrency Theory (CONCUR). Lecture Notes in Computer Science, vol. 1877, pp. 153–167. Springer (2000)
17. Gastin, P., Oddoux, D.: 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, pp. 53–65. Springer (2001)
18. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: 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, pp. 3–18. Chapman & Hall (1995)
19. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research, Lecture Notes in Computer Science, vol. 2500. Springer (2002)
20. Hahn, E.M., Li, G., Schewe, S., Turrini, A., Zhang, L.: 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 (2015)
21. Haverkort, B.R., Hermanns, H., Katoen, J.P.: On the use of model checking techniques for dependability evaluation. In: 19th IEEE Symposium on Reliable Distributed Systems (SRDS). pp. 228–237. IEEE Computer Society (2000)
22. Isaak, D., Löding, C.: Efficient inclusion testing for simple classes of unambiguous ω-automata. Information Processing Letters **112**(14-15), 578–582 (2012)
23. Jantsch, S., Müller, D., Baier, C., Klein, J.: From LTL to Unambiguous Büchi Automata via Disambiguation of Alternating Automata. Tech. rep., Technische Universität Dresden (2019), `https://arxiv.org/abs/1907.02887/`
24. Karmarkar, H., Joglekar, M., Chakraborty, S.: Improved upper and lower bounds for Büchi disambiguation. In: Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 40–54 (2013)
25. Kretínský, J., Meggendorfer, T., Sickert, S.: LTL store: Repository of LTL formulae from literature and case studies. CoRR **abs/1807.03296** (2018), `http://arxiv.org/abs/1807.03296`
26. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: Proceedings of the 9th International Conference on Quantitative Evaluation of SysTems (QEST). pp. 203–204. IEEE Computer Society (2012)
27. Löding, C.: Efficient minimization of deterministic weak ω-automata. Information Processing Letters **79**(3), 105–109 (2001)
28. Löding, C., Thomas, W.: Alternating automata and logics over infinite words. In: Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics IFIP TCS. pp. 521–535 (2000)
29. Mohri, M.: On the disambiguation of finite automata and functional transducers. International Journal of Foundations of Computer Science **24**(6), 847–862 (2013)
30. Müller, D., Sickert, S.: 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, pp. 180–194. Open Publishing Association (2017)
31. Muller, D.E., Saoudi, A., Schupp, P.E.: 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 (1988)

32. Muller, D.E., Schupp, P.E.: Alternating automata on infinite trees. Theoretical Computer Science **54**, 267–276 (1987)
33. Somenzi, F., Bloem, R.: 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, pp. 248–263. Springer (2000)
34. Stearns, R.E., Hunt, H.B.: On the equivalence and containment problem for unambiguous regular expressions, grammars, and automata. SIAM Journal on Computing pp. 598–611 (1985)
35. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS). pp. 327–338. IEEE Computer Society (1985)
36. Vardi, M.Y.: Nontraditional applications of automata theory. In: Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS). pp. 575–597 (1994)
37. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: Proceedings of the 1st Symposium on Logic in Computer Science (LICS). pp. 332–344. IEEE Computer Society Press (1986)