LTL to Self-Loop Alternating Automata with Generic Acceptance and Back*

František Blahoudek[a], Juraj Major[b], Jan Strejček[b]

[a] *University of Texas at Austin, USA*
[b] *Masaryk University, Brno, Czech Republic*

Abstract

Self-loop alternating automata (SLAA) with Büchi or co-Büchi acceptance are popular formalisms also known as very weak alternating automata (VWAA). They are often used as an intermediate results in translations of LTL to deterministic or nondeterministic automata. This paper considers SLAA with generic transition-based Emerson-Lei acceptance and presents translations of LTL to these automata and back. Importantly, the translation of LTL to SLAA with generic acceptance produces considerably smaller automata than previous translations of LTL to Büchi or co-Büchi SLAA. Our translation is already implemented in the tool ltl3tela, where it helps to produce small deterministic or nondeterministic transition-based Emerson-Lei automata for given LTL formulae.

Keywords: LTL, Omega-automata, LTL to automata translation, Alternating automata, ltl3tela

1. Introduction

Translation of *linear temporal logic (LTL)* [26] into equivalent automata over infinite words is an important part of many methods for model checking, controller synthesis, monitoring, etc. This paper presents improved translations of LTL to *self-loop alternating automata (SLAA)* [30], which are alternating automata that contain no cycles except self-loops. These automata are studied for more than 20 years under several different names including *very weak* [13, 28], *linear* [19], *linear weak* [15], or *1-weak* [25] *alternating automata*. The first publications showing that any LTL formula can be easily translated to an SLAA with only a linear number of states in the length of the formula are even older [22, 31]. An LTL to SLAA translation forms the first step of many LTL to automata translations. For example, it is used in popular tools ltl2ba [13] and ltl3ba [2] translating LTL to nondeterministic automata, and also in the tool ltl3dra [1] translating a fragment of LTL to deterministic automata.

*This is an extended version of a paper presented at ICTAC 2019 [5].

Email addresses: frantisek.blahoudek@gmail.com (František Blahoudek), major@fi.muni.cz (Juraj Major), strejcek@fi.muni.cz (Jan Strejček)

A nice survey of various instances of LTL to SLAA translations can be found in Tauriainen's doctoral thesis [30], where he also presents another improved LTL to SLAA translation. To our best knowledge, the only new improvement since publication of the thesis has been presented by Babiak et al. [2]. All the LTL to SLAA translations considered so far produce SLAA with (state-based) Büchi or co-Büchi acceptance. The only exception is the translation by Tauriainen producing SLAA with transition-based co-Büchi acceptance.

In this paper, we follow a general trend of recent research and development in the field of automata over infinite words and their applications: consider a more general acceptance condition to construct smaller automata. In theory, this change usually does not decrease the upper bound on the size of constructed automata. Moreover, the complexity of algorithms processing automata with a more involved acceptance condition can be even higher. However, practical experiences show that achieved reduction of automata size often outweighs complications with a more general acceptance condition. This can be documented by observations of nondeterministic as well as deterministic automata.

Nondeterministic automata are usually considered with Büchi acceptance. However, all three most popular LTL to nondeterministic automata translators, namely ltl2ba [13], ltl3ba [2], and Spot [9], translate LTL formulae to *transition-based generalized Büchi automata (TGBA)*, which are further transformed to Büchi automata. When solving emptiness check, which is the central part of many model checking tools, algorithms designed for TGBA perform better than algorithms analyzing the corresponding Büchi automata [7, 27].

Deterministic automata were typically considered with Rabin, Streett, or parity acceptance. Tools of the Rabinizer family [18] and the tool ltl3dra [1] produce also deterministic automata with transition-based generalized Rabin acceptance. The equivalent Rabin automata are often dramatically larger. Direct processing of generalized Rabin automata can be substantially more efficient as shown by Chatterjee et al. [6] for probabilistic model checking and LTL synthesis.

All the previously mentioned acceptance conditions can be expressed by a generic acceptance condition originally introduced by Emerson and Lei [11] and recently reinvented in the *Hanoi omega-automata (HOA) format* [3]. Emerson-Lei acceptance condition is any positive boolean formula over terms of the form Inf● and Fin●, where ● is an *acceptance mark*. A run of a nondeterministic automaton (or an infinite branch of a run of an alternating automaton) satisfies Inf● or Fin● if it visits the acceptance mark ● infinitely often or finitely often, respectively. The acceptance marks placed on states denote traditional state-based acceptance, while marks placed on transitions correspond to transition-based acceptance.

Some tools that work with *transition-based Emerson-Lei automata (TELA)* already exist. For example, Delag [23] produces deterministic TELA, Spot is now able to produce both deterministic and nondeterministic TELA, and we have recently developed ltl3tela [21] that directly specializes in production of deterministic and nondeterministic TELA. The produced TELA are often smaller than the automata produced by the tools mentioned in the previous paragraphs. Since version 2.7, Spot provides also an emptiness check for TELA, and a

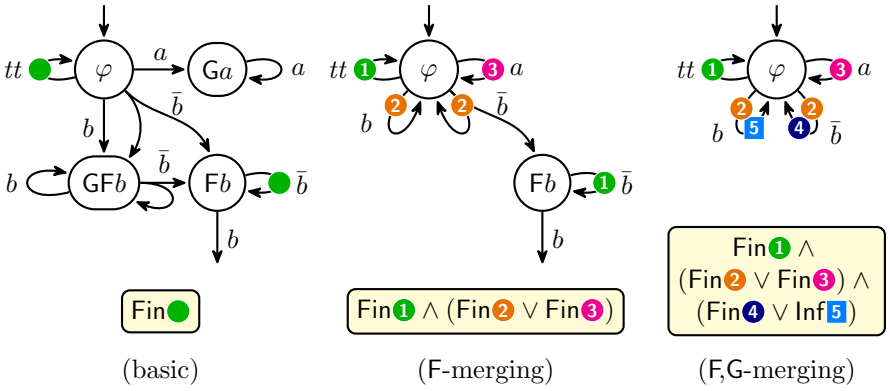

Figure 1: Automata for the formula $\varphi = \mathsf{F}(\mathsf{G}a \vee \mathsf{GF}b)$: the co-Büchi SLAA produced by the basic translation (left), the Inf-less SLAA produced by F-merging (middle), and the SLAA produced by F,G-merging (right). Graphical notation is explained in Section 2.

probabilistic model checking algorithm working with deterministic Emerson-Lei automata has been implemented in PRISM. In both cases, an improved performance over previous solutions has been reported [4].

This paper presents a translation of LTL to SLAA with transition-based Emerson-Lei acceptance. The translation aims to take advantage of the generic acceptance and produce SLAA with less states. We present it in three steps.

1. Section 3 recalls a basic translation producing co-Büchi SLAA. The description uses the same terminology and notation as the following modified translations. In particular, the acceptance marks are on transitions.
2. In Section 4, we modify the translation such that states for subformulae of the form $\mathsf{F}\psi$ are merged with states for ψ. The technique is called F-*merging*. The acceptance condition of constructed SLAA is a positive boolean combination of Fin-terms. We call such automata Inf-*less SLAA*.
3. Finally, we further modify the translation in Section 5, where states for some subformulae of the form $\mathsf{G}\psi$ are merged with states for ψ. The resulting technique is thus called F,G-*merging*. Constructed SLAA use acceptance condition containing both Inf- and Fin-terms.

The difference between these translations is illustrated by Figure 1 showing three SLAA for the formula $\mathsf{F}(\mathsf{G}a \vee \mathsf{GF}b)$. One can observe that the initial state of the middle automaton is merged with the states for $\mathsf{G}a$ and $\mathsf{GF}b$ due to F-merging. In the automaton on the right, the state for $\mathsf{GF}b$ is merged with $\mathsf{F}b$ and the initial state is then merged with $\mathsf{G}a$ and $\mathsf{GF}b$. Hence, the resulting automaton contains only one state and the LTL to SLAA translation in this case produces directly a nondeterministic automaton.

LTL to SLAA translations are traditionally accompanied by automata simplification based on transition dominance [13]. Section 6 extends this idea to SLAA with generic acceptance and introduces additional simplifications that reduce the state space and increase determinism of SLAA.

Section 7 completes the theoretical part of the paper with a backward translation which takes an SLAA with transition-based Emerson-Lei acceptance and produces an equivalent LTL formula. Altogether, we get that SLAA with the generic acceptance have the same expressiveness as LTL.

The three presented LTL to SLAA translations are implemented in the tool ltl3tela [21], which also implements an SLAA dealternation algorithm. Section 8 provides an experimental comparison of the three presented translations and the LTL to SLAA translation implemented in ltl3ba. The effect of SLAA simplifications is shown as well. On randomly generated formulae containing only temporal operators *eventually* (F) and *always* (G), which are favourable to our translation improvements, the F,G-merging can save nearly 50% of states. This is a considerable reduction, especially with respect to the fact that even the simplest LTL to SLAA translations produce automata of linear size and thus the space for reduction is not big. Our experiments also show that SLAA produced by the F,G-merging are often nondeterministic (or even deterministic) as they do not contain any universal branching. We compared such automata with the results of state-of-the art LTL to deterministic or nondeterministic automata translators, namely ltl3tela, Spot, Delag, ltl2dgra (the current replacement of Rabinizer 4 distributed with Owl [17]), and ltl3ba.

This paper extends our ICTAC 2019 paper [5] in several directions. In particular, here we include proofs of theorems claiming that our translations are correct. In Section 6, we present one modified SLAA simplification (the original formulation was incorrect), introduce one new simplification, and illustrate the simplifications with examples. Further, we add a new section about the tool ltl3tela. Finally, the experimental evaluation now presents some additional comparisons and we redone the original experiments using the current versions of tools.

2. Preliminaries

This section recalls the notion of *linear temporal logic* [26] and the definition of *self-loop alternating automata* [30]. We always use automata with transition-based acceptance condition given in the format of Emerson-Lei acceptance. For example, co-Büchi automaton is an automaton with acceptance condition $\mathsf{Fin}(\bullet)$.

2.1. Linear Temporal Logic (LTL)

We define the syntax of LTL formulae directly in the positive normal form as

$$\psi ::= tt \mid ff \mid a \mid \neg a \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathsf{X}\psi \mid \psi \,\mathsf{U}\, \psi \mid \psi \,\mathsf{R}\, \psi,$$

where tt stands for *true*, ff for *false*, a ranges over a set AP of *atomic propositions*, and $\mathsf{X}, \mathsf{U}, \mathsf{R}$ are temporal operators called *next*, *until*, and *release*, respectively. A *word* is an infinite sequence $u = u_0u_1u_2 \ldots \in \Sigma^\omega$, where $\Sigma \subseteq 2^{AP}$. By $u_{i..}$ we denote the suffix $u_{i..} = u_iu_{i+1} \ldots$. We define when a word u *satisfies* ψ, written $u \models \psi$, as follows:

$$
\begin{array}{lll}
u \models tt & & \\
u \not\models ff & & \\
u \models a & \text{iff} & a \in u_0 \\
u \models \neg a & \text{iff} & a \notin u_0 \\
u \models \psi_1 \vee \psi_2 & \text{iff} & u \models \psi_1 \text{ or } u \models \psi_2 \\
u \models \psi_1 \wedge \psi_2 & \text{iff} & u \models \psi_1 \text{ and } u \models \psi_2 \\
u \models \mathsf{X}\psi & \text{iff} & u_{1..} \models \psi \\
u \models \psi_1 \,\mathsf{U}\, \psi_2 & \text{iff} & \exists i \geq 0 \text{ such that } u_{i..} \models \psi_2 \text{ and } \forall\, 0 \leq j < i \,.\, u_{j..} \models \psi_1 \\
u \models \psi_1 \,\mathsf{R}\, \psi_2 & \text{iff} & \exists i \geq 0 \text{ such that } u_{i..} \models \psi_1 \text{ and } \forall\, 0 \leq j \leq i \,.\, u_{j..} \models \psi_2, \\
 & & \text{or } \forall i \geq 0 \,.\, u_{i..} \models \psi_2
\end{array}
$$

For a fixed alphabet $\Sigma \subseteq 2^{AP}$, a formula ψ defines the language $L(\psi) = \{u \in \Sigma^\omega \mid u \models \psi\}$. If Σ is not specified, we consider $\Sigma = 2^{AP(\psi)}$, where $AP(\psi)$ denotes the set of atomic propositions occurring in ψ. Further, we extend the syntax of LTL with derived operators *eventually* (F) and *always* (G) defined by $\mathsf{F}\psi \equiv tt\,\mathsf{U}\,\psi$ and $\mathsf{G}\psi \equiv ff\,\mathsf{R}\,\psi$. A *temporal formula* is a formula where the topmost operator is neither conjunction, nor disjunction. A formula without any temporal operator is called *state formula*. Formulae $tt, ff, a, \neg a$ are both temporal and state formulae.

2.2. Self-Loop Alternating Automata (SLAA)

An *alternating automaton* is a tuple $\mathcal{A} = (S, \Sigma, \mathcal{M}, \Delta, s_I, \Phi)$, where

- S is a finite set of *states*,
- Σ is a finite *alphabet*,
- $\mathcal{M}$ is a finite set of *acceptance marks*,
- $\Delta \subseteq S \times \Sigma \times 2^{\mathcal{M}} \times 2^S$ is an *alternating transition relation*,
- $s_I \in S$ is the *initial state*, and
- Φ is an *acceptance formula*, which is a positive boolean combination of terms Fin● and Inf●, where ● ranges over $\mathcal{M}$.

An alternating automaton is a *self-loop alternating automaton (SLAA)* if there exists a partial order on S such that for every $(s, \alpha, M, C) \in \Delta$, all states in C are lower or equal to s. In other words, SLAA contain no cycles except self-loops.

Subsets $C \subseteq S$ are called *configurations*. A quadruple $t = (s, \alpha, M, C) \in \Delta$ is called a *transition* from s to C under α (or labelled by α or α-transition) marked by elements of M. A transition $t = (s, \alpha, M, C) \in \Delta$ is *looping* (or simply a *loop*) if its *destination configuration* C contains its *source* s.

A *multitransition* T under α is a set of transitions under α such that the source states of the transitions are pairwise different. The *source configuration* $\mathrm{dom}(T)$ of a multitransition T is the set of source states of transitions in T. The *destination configuration* $\mathrm{range}(T)$ is the union of destination configurations of transitions in T. For an alternating automaton $\mathcal{A}$, $\Gamma^{\mathcal{A}}$ denotes the set of all multitransitions of $\mathcal{A}$ and $\Gamma^{\mathcal{A}}_{\alpha}$ denotes the set of all multitransitions of $\mathcal{A}$ under α.

A *run* ρ of an alternating automaton $\mathcal{A}$ over a word $u = u_0u_1 \ldots \in \Sigma^\omega$ is an infinite sequence $\rho = T_0T_1 \ldots$ of multitransitions such that $\mathrm{dom}(T_0) = \{s_I\}$ and, for all $i \geq 0$, T_i is labelled by u_i and $\mathrm{range}(T_i) = \mathrm{dom}(T_{i+1})$. Each run ρ defines a directed acyclic edge-labelled graph $G_\rho = (V, E, \lambda)$, where

$$V = \bigcup_{i=0}^{\infty} V_i, \text{ where } V_i = \mathrm{dom}(T_i) \times \{i\},$$
$$E = \bigcup_{i=0}^{\infty} \left\{ \big((s,i),(s',i+1)\big) \mid (s,\alpha,M,C) \in T_i, s' \in C \right\}, \text{ and}$$

the labeling function $\lambda : E \to 2^{\mathcal{M}}$ assigns to each edge $e = \big((s,i),(s',i+1)\big) \in E$ the acceptance marks from the corresponding transition, i.e., $\lambda(e) = M$ where $(s,\alpha,M,C) \in T_i$. A *branch* of the run ρ is a maximal (finite or infinite) sequence $b = (v_0,v_1)(v_1,v_2)\ldots$ of consecutive edges in G_ρ such that $v_0 \in V_0$. For an infinite branch b, let $M(b)$ denote the set of marks that appear in infinitely many sets of the sequence $\lambda((v_0,v_1))\lambda((v_1,v_2))\ldots$. An infinite branch b satisfies Inf● if ● $\in M(b)$ and it satisfies Fin● if ● $\notin M(b)$. An infinite branch is *accepting* if it satisfies the acceptance formula Φ. We say that a run ρ is *accepting* if all its infinite branches are accepting. The language of $\mathcal{A}$ is the set $L(\mathcal{A}) = \{u \in \Sigma^\omega \mid \mathcal{A} \text{ has an accepting run over } u\}$.

Several examples of SLAA are given in Figure 1. Examples of SLAA with their runs can be found in Figure 5. Note that a transition $(s,\alpha,M,C) \in \Delta$ of an alternating automaton is visualised as a branching edge leading from s to all states in C. In this paper, an automaton alphabet has always the form $\Sigma = 2^{AP'}$, where AP' is a finite set of atomic propositions. To keep the visual representation of automata concise, edges are labelled with boolean formulae over atomic propositions in a condensed notation: $\bar{a}$ denotes $\neg a$ and conjunctions are omitted. Hence, $ab\bar{c}$ would stand for $a \wedge b \wedge \neg c$. Every edge represents all transitions under combinations of atomic propositions satisfying its label.

3. Basic Translation

This section presents a basic translation of LTL to co-Büchi SLAA similar to the one implemented in ltl3ba [2]. To simplify the presentation, in contrast to the translation of ltl3ba we omit the optimization called *suspension*, we describe transitions for each $\alpha \in \Sigma$ separately, and we slightly modify the acceptance condition of the SLAA; in particular, we switch from state-based to transition-based acceptance.

Let φ be an LTL formula, where subformulae of the form $\mathsf{F}\psi$ and $\mathsf{G}\psi$ are seen as abbreviations for $tt\,\mathsf{U}\,\varphi$ and $ff\,\mathsf{R}\,\varphi$, respectively. An equivalent SLAA is constructed as $\mathcal{A}_\varphi = (S, \Sigma, \{●\}, \Delta, \varphi, \mathsf{Fin}●)$, where states in S are subformulae of φ and $\Sigma = 2^{AP(\varphi)}$. The construction of the transition relation Δ treats it equivalently as a function $\Delta : S \times \Sigma \to 2^{\mathcal{P}}$ where $\mathcal{P} = 2^{\mathcal{M}} \times 2^S$. The construction of Δ is defined inductively and it directly corresponds to the semantics of LTL.

The acceptance mark ● is used to ensure that an accepting run cannot stay in a state $\psi_1 \,\mathsf{U}\, \psi_2$ forever. In other words, it ensures that ψ_2 will eventually hold. The translation uses an auxiliary product operator $\otimes$ and a marks eraser me defined for each $P, P' \subseteq \mathcal{P}$ as:

$$P \otimes P' = \{(M \cup M', C \cup C') \mid (M, C) \in P, (M', C') \in P'\}$$
$$\mathsf{me}(P) = \{(\emptyset, C) \mid (M, C) \in P\}$$

The product operator is typically used to handle conjunction: to get successors of $\psi_1 \wedge \psi_2$, we compute the successors of ψ_1 and the successors of ψ_2 and combine them using the product operator $\otimes$. The marks eraser has two applications. First, it is used to remove unwanted acceptance marks on transitions looping on states of the form $\psi_1 \,\mathsf{R}\, \psi_2$. Second, it is used to remove irrelevant accepting marks from the automaton, which are all marks not lying on loops. Indeed, only looping transition can appear infinitely often on some branch of an SLAA run and thus only marks on loops are relevant for acceptance.

$$\begin{aligned}
\Delta(tt, \alpha) &= \{(\emptyset, \emptyset)\} \\
\Delta(ff, \alpha) &= \emptyset \\
\Delta(a, \alpha) &= \{(\emptyset, \emptyset)\} \text{ if } a \in \alpha,\ \emptyset \text{ otherwise} \\
\Delta(\neg a, \alpha) &= \{(\emptyset, \emptyset)\} \text{ if } a \notin \alpha,\ \emptyset \text{ otherwise} \\
\Delta(\psi_1 \wedge \psi_2, \alpha) &= \mathsf{me}\big(\Delta(\psi_1, \alpha) \otimes \Delta(\psi_2, \alpha)\big) \\
\Delta(\psi_1 \vee \psi_2, \alpha) &= \mathsf{me}\big(\Delta(\psi_1, \alpha) \cup \Delta(\psi_2, \alpha)\big) \\
\Delta(\mathsf{X}\psi, \alpha) &= \big\{(\emptyset, \{\psi\})\big\} \\
\Delta(\psi_1 \mathsf{U} \psi_2, \alpha) &= \mathsf{me}\big(\Delta(\psi_2, \alpha)\big) \cup \Big(\big\{(\{●\}, \{\psi_1 \mathsf{U} \psi_2\})\big\} \otimes \mathsf{me}\big(\Delta(\psi_1, \alpha)\big)\Big) \\
\Delta(\psi_1 \mathsf{R} \psi_2, \alpha) &= \mathsf{me}\big(\Delta(\psi_1, \alpha) \otimes \Delta(\psi_2, \alpha)\big) \cup \mathsf{me}\Big(\big\{(\emptyset, \{\psi_1 \mathsf{R} \psi_2\})\big\} \otimes \Delta(\psi_2, \alpha)\Big)
\end{aligned}$$

The automaton $\mathcal{A}_\varphi$ has at most $|\varphi|$ states as the states are subformulae of φ. To prove that the constructed automaton is a self-loop alternating automaton, it is enough to consider the partial order *'being a subformula of'* on states.

As the basic translation handles formulae $\mathsf{F}\psi$ and $\mathsf{G}\psi$ as abbreviations, we do not have to explicitely define $\Delta(\mathsf{F}\psi, \alpha)$ and $\Delta(\mathsf{G}\psi, \alpha)$. Nevertheless, we provide their values to facilitate the comparison of the basic translation and the following translations, which differ particularly in processing of $\mathsf{F}\psi$ and $\mathsf{G}\psi$.

$$\Delta(\mathsf{F}\psi, \alpha) = \Delta(tt \,\mathsf{U}\, \psi, \alpha) = \big\{(\{●\}, \{\mathsf{F}\psi\})\big\} \cup \mathsf{me}\big(\Delta(\psi, \alpha)\big)$$
$$\Delta(\mathsf{G}\psi, \alpha) = \Delta(ff \,\mathsf{R}\, \psi, \alpha) = \big\{(\emptyset, \{\mathsf{G}\psi\})\big\} \otimes \mathsf{me}\big(\Delta(\psi, \alpha)\big)$$

4. F-Merging Translation

Now we modify the basic translation on subformulae of the form $\mathsf{F}\psi$. The modified translation produces Inf-*less SLAA*, which are SLAA without Inf■ terms in acceptance formulae.

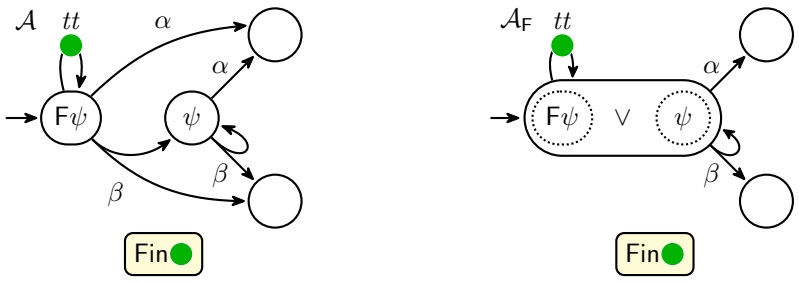

Figure 2: Automata for $\mathsf{F}\psi$: the SLAA $\mathcal{A}$ built by the basic translation (left) and the SLAA $\mathcal{A}_\mathsf{F}$ built by the F-merging translation (right).

Before giving the formal translation, we discuss three examples to explain the ideas behind F-merging. We start with a formula $\mathsf{F}\psi$ where ψ is a temporal formula. Further, assume that the state ψ of the SLAA constructed by the basic translation has two types of transitions: non-looping labelled by α and loops labelled by β. The SLAA $\mathcal{A}$ for $\mathsf{F}\psi$ can be found in Figure 2 (left). States $\mathsf{F}\psi$ and ψ can be merged into a single state that represents their disjunction (which is equivalent to $\mathsf{F}\psi$) as shown by the SLAA $\mathcal{A}_\mathsf{F}$ of Figure 2 (right). The construction is still correct: (i) Clearly, $\mathcal{A}_\mathsf{F}$ can precisely mimic each sequence of transitions that can be taken in $\mathcal{A}$. (ii) The sequences of transitions of $\mathcal{A}_\mathsf{F}$ that cannot be precisely mimicked by $\mathcal{A}$ are those where the tt-loop is taken after some β-loop. However, every accepting run of $\mathcal{A}_\mathsf{F}$ uses the tt-loop only finitely many times and thus we can find a corresponding accepting run of $\mathcal{A}$: instead of each β-loop that occurs before the last tt-loop we can use the tt-loop since β implies tt.

The second example deals with the formula $\mathsf{F}\psi$ where $\psi = (a\,\mathsf{R}\,b) \wedge \mathsf{G}c$. Figure 3 (left) depicts the SLAA $\mathcal{A}$ produced by the basic translation. The state ψ is dotted as it is unreachable. Hence, merging $\mathsf{F}\psi$ with ψ would not save any state. However, we can modify the translation rules to make ψ reachable and $a\,\mathsf{R}\,b$ unreachable at the same time. The modification is based on the following

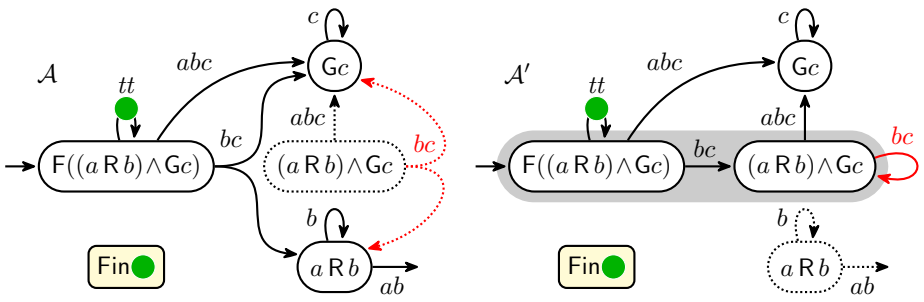

Figure 3: Automata for $\mathsf{F}((a\,\mathsf{R}\,b) \wedge \mathsf{G}c)$: the SLAA $\mathcal{A}$ built by the basic translation (left) and the modified SLAA $\mathcal{A}'$ where states in the grey area can be merged (right).

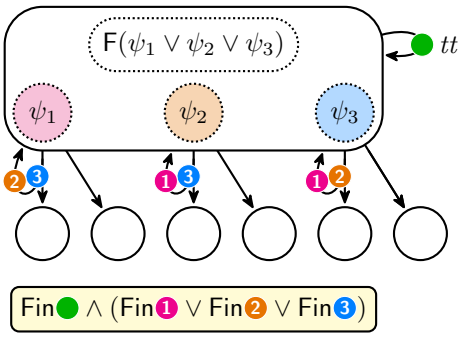

Figure 4: Transitions of the state $\mathsf{F}(\psi_1 \vee \psi_2 \vee \psi_3)$ merged with states ψ_1, ψ_2, and ψ_3.

observation. Taking the red bc-edge in $\mathcal{A}$ would mean that both $a \,\mathsf{R}\, b$ and $\mathsf{G}c$ have to hold in the next step, which is equivalent to $(a \,\mathsf{R}\, b) \wedge \mathsf{G}c$. Thus we can replace the red edge by the red bc-loop as shown in the automaton $\mathcal{A}'$ of Figure 3 (right). Because transitions leaving the state $\mathsf{F}\psi$ are computed from transitions of ψ, this replacement makes the state $(a \,\mathsf{R}\, b) \wedge \mathsf{G}c$ reachable and the state $a \,\mathsf{R}\, b$ becomes unreachable. The states $\mathsf{F}\psi$ and ψ of $\mathcal{A}'$ can be now merged for the same reason as in Figure 2.

While the previous paragraph studied a formula $\mathsf{F}\psi$ where ψ is a conjunction, the third example focuses on disjunctions. Let us consider a formula $\mathsf{F}\psi$ where $\psi = \psi_1 \vee \psi_2 \vee \psi_3$ and each ψ_i is a temporal formula. As in the previous example, the state ψ is unreachable in the SLAA produced by the basic translation and thus merging $\mathsf{F}\psi$ with ψ does not make any sense. However, we can merge the state $\mathsf{F}\psi$ with states ψ_1, ψ_2, ψ_3 as indicated in Figure 4. In contrast to the original SLAA, a single run of the merged SLAA can use a loop corresponding to a state ψ_i and subsequently a transition corresponding to a different ψ_j. Instead of every such a loop, the original SLAA can simply use the tt-loop of $\mathsf{F}\psi$. However, as the tt-loop is marked by $\bullet$, we can use it only finitely many times. In fact, the runs of the merged automaton that contain infinitely many loops corresponding to two or more different states ψ_i should be nonaccepting. Therefore we adjust the acceptance formula to $\mathsf{Fin}(\bullet) \wedge (\mathsf{Fin}(1) \vee \mathsf{Fin}(2) \vee \mathsf{Fin}(3))$ and place the new acceptance marks as shown in Figure 4. Clearly, $\mathsf{Fin}(1)$ says that transitions of ψ_2 and ψ_3 are taken only finitely many times, $\mathsf{Fin}(2)$ does the same for transitions of ψ_1 and ψ_3, and $\mathsf{Fin}(3)$ for transitions of ψ_1 and ψ_2.

The F-merging translation combines the ideas presented above when processing any F-subformula, while other subformulae are handled as in the basic translation. Hence, we no longer treat $\mathsf{F}\psi$ as an abbreviation for $tt \,\mathsf{U}\, \psi$. Further, we think about formulae $\mathsf{F}\psi$ as formulae of the form $\mathsf{F} \bigvee_i \bigwedge_j \psi_{i,j}$, where $\psi_{i,j}$ are temporal formulae. Formally, we define formula decomposition into disjunctive normal form $\overline{\psi}$ as follows:

$$\overline{\psi} = \{\{\psi\}\} \text{ if } \psi \text{ is a temporal formula}$$
$$\overline{\psi_1 \vee \psi_2} = \overline{\psi_1} \cup \overline{\psi_2}$$

$$\overline{\psi_1 \wedge \psi_2} = \{C_1 \cup C_2 \mid C_1 \in \overline{\psi_1} \text{ and } C_2 \in \overline{\psi_2}\}.$$

Let $K \in \overline{\psi}$ be a set of temporal formulae. We use ψ_K to denote $\psi_K = \bigwedge_{\psi' \in K} \psi'$. Clearly, ψ is equivalent to $\bigvee_{K \in \overline{\psi}} \psi_K$. We define two auxiliary transition functions $\Delta_{\mathsf{L}}, \Delta_{\mathsf{NL}}$ to implement the trick illustrated with red edges in Figure 3. Intuitively, $\Delta_{\mathsf{L}}(\psi_K, \alpha)$ is the set of α-transitions of ψ_K such that their destination configuration subsumes K (the transitions are looping in this sense, hence the subscript L), while $\Delta_{\mathsf{NL}}(\psi_K, \alpha)$ represents the remaining α-transitions of ψ_K (non-looping, hence the subscript NL). To mimic the trick of Figure 3, we should replace in the destination configuration of each looping transition all elements of K by the state corresponding to ψ_K. To simplify this step, we define the destination configurations of looping transitions in $\Delta_{\mathsf{L}}(\psi_K, \alpha)$ directly without elements of K.

$$\Delta_{\mathsf{L}}(\psi_K, \alpha) = \big\{(M, C \smallsetminus K) \mid (M, C) \in \Delta(\psi_K, \alpha),\ K \subseteq C\big\}$$
$$\Delta_{\mathsf{NL}}(\psi_K, \alpha) = \big\{(M, C) \mid (M, C) \in \Delta(\psi_K, \alpha),\ K \not\subseteq C\big\}$$

Simultaneously with the trick of Figure 3, we apply the idea indicated in Figure 4 and merge the state $\mathsf{F}\psi$ with all states ψ_K for $K \in \overline{\psi}$. Hence, instead of extending the looping transitions with states ψ_K, we extend it with the merged state called simply $\mathsf{F}\psi$. Altogether, we get

$$\Delta(\mathsf{F}\psi, \alpha) = \big\{(\{\bullet\}, \{\mathsf{F}\psi\})\big\} \cup \bigcup_{K \in \overline{\psi}} \Big(\mathsf{me}\big(\Delta_{\mathsf{NL}}(\psi_K, \alpha)\big) \ \cup\ \big\{(M_K, \{\mathsf{F}\psi\})\big\} \otimes \Delta_{\mathsf{L}}(\psi_K, \alpha)\Big)$$

where

$$M_K = \{\bullet^{K'} \mid K' \in \overline{\psi} \text{ and } K' \neq K\}.$$

In other words, the merged state $\mathsf{F}\psi$ has three kinds of transitions: the *tt*-loop marked by $\bullet$, non-looping transitions of states ψ_K for each disjunct $K \in \overline{\psi}$, and the looping transitions of states ψ_K which are marked as shown in Figure 4. Finally, we redefine the set of acceptance marks $\mathcal{M}$ and the acceptance formula Φ of the constructed SLAA as follows, where $\mathcal{F}_\varphi$ denotes the set of all subformulae of φ of the form $\mathsf{F}\psi$:

$$\mathcal{M} = \{\bullet\} \cup \{\bullet^K \mid \mathsf{F}\psi \in \mathcal{F}_\varphi \text{ and } K \in \overline{\psi}\}$$
$$\Phi = \mathsf{Fin}(\bullet) \wedge \bigwedge_{\mathsf{F}\psi \in \mathcal{F}_\varphi} \bigvee_{K \in \overline{\psi}} \mathsf{Fin}(\bullet^K)$$

In fact, we can reduce the number of orange marks (marks with an upper index) produced by F-merging translation. Let $n_\varphi = \max\{|\overline{\psi}| \mid \mathsf{F}\psi \in \mathcal{F}_\varphi\}$ be the maximal number of such marks corresponding to any subformula $\mathsf{F}\psi$ of φ. We can limit the total number of orange marks to n_φ by reusing them. We redefine $\mathcal{M}$ and Φ as

$$\mathcal{M} = \{\bullet\} \cup \{\bullet^i \mid 1 \leq i \leq n_\varphi\} \qquad \text{and} \qquad \Phi = \mathsf{Fin}(\bullet) \wedge \bigvee_{i=1}^{n_\varphi} \mathsf{Fin}(\bullet^i)$$

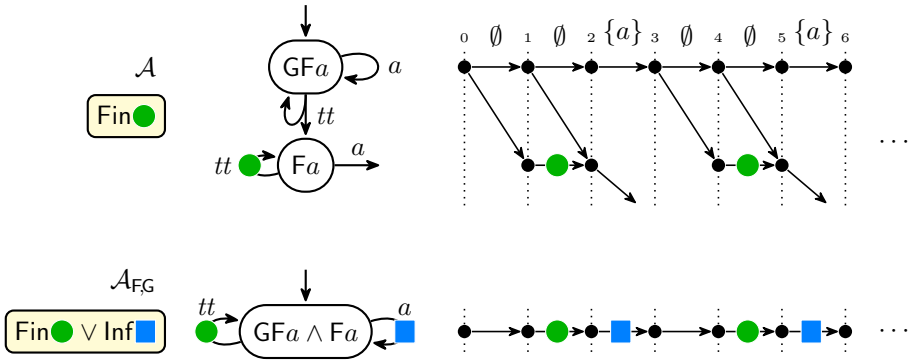

Figure 5: An SLAA $\mathcal{A}$ for the formula $\mathsf{GF}a$ built by the basic translation (top) and an equivalent SLAA $\mathcal{A}_{\mathsf{FG}}$ built by the F,G-merging translation (bottom) and their runs over the word $(\emptyset\emptyset\{a\})^\omega$.

and alter the above definition of M_K. For every $\mathsf{F}\psi \in \mathcal{F}_\varphi$, we assign a unique index i_K, $1 \leq i_K \leq n_\varphi$, to each $K \in \overline{\psi}$. Sets M_K in transitions of $\mathsf{F}\psi$ are then defined to contain all orange marks except ^{i_K}, formally $M_K = \mathcal{M} \setminus \{, ^{i_K}\}$. This optimization is correct as any branch of an SLAA run cannot cycle between two states of the form $\mathsf{F}\psi$.

5. F,G-Merging Translation

We further improve the F-merging translation by adding a special rule also for subformulae of the form $\mathsf{G}\psi$. The resulting F,G-merging translation produces SLAA with an acceptance formula that is not Inf-less.

We start again with a simple example. Consider the formula $\mathsf{GF}a$. The basic translation produces the SLAA $\mathcal{A}$ from Figure 5 (top). In general, each transition of the state $\mathsf{G}\psi$ is a transition of ψ extended with a loop back to $\mathsf{G}\psi$. The one-to-one correspondence between transitions of $\mathsf{G}\psi$ and ψ leads to the idea to merge the states into one that corresponds to their conjunction $(\mathsf{G}\psi) \wedge \psi$, which is equivalent to $\mathsf{G}\psi$. However, merging these states needs a special care. Figure 5 (bottom) shows an SLAA where the states $\mathsf{GF}a$ and $\mathsf{F}a$ are merged. Consider now the word $u = (\emptyset\emptyset\{a\})^\omega$ and the runs of the two automata over u. The branches of the top run collapse into a single branch in the bottom run. While each branch of the top run has at most one occurrence of , the single branch in the bottom run contains infinitely many of these marks. The SLAA $\mathcal{A}_{\mathsf{FG}}$ accepts u only because of the added marks. The intuition for their placement is explained using the concept of *escaping multitransitions*.

A multitransition T of an SLAA $\mathcal{A}'$ is *s-escaping* for a state s if it contains a non-looping transition $(s, \alpha, M, C) \in T$. For an acceptance mark , we define its *owners* $\mathcal{O}() = \{s \in S \mid (s, \alpha, M, C) \in \Delta \text{ and } \in M\}$ as the set of all states with outgoing transitions marked by . The following observation holds for every mark with a single owner s. *A run of $\mathcal{A}'$ satisfies* Fin *(i.e., all*

its infinite branches satisfy $\mathsf{Fin}(\bullet)$*) if and only if the run contains only a finite number of multitransitions* T *marked by* $\bullet$ *or if it contains infinitely many* s*-escaping multitransitions.*

Transitions of $\mathcal{A}_{\mathsf{FG}}$ correspond to multitransitions of $\mathcal{A}$ with source configuration $\{\mathsf{GF}a, \mathsf{F}a\}$. The observation implies that $\mathcal{A}_{\mathsf{FG}}$ would be equivalent to $\mathcal{A}$ if we mark all transitions corresponding to $\mathsf{F}a$-escaping multitransitions with a new mark $\blacksquare$ and change the acceptance formula to $\mathsf{Fin}(\bullet) \vee \mathsf{Inf}(\blacksquare)$ as shown in Figure 5 (bottom).

This approach naturally extends to the case of $\mathsf{G}\psi$ where $\psi = \bigwedge_i \psi_i$ is a conjunction of temporal formulae. In this case, $\mathsf{G}\psi$ can be merged with all states ψ_i into a single state representing the conjunction $(\mathsf{G}\psi) \wedge \bigwedge_i \psi_i$. However, we have to pay a special attention to acceptance marks as the observation formulated above holds only for marks with a single owner. For every ψ_i with a $\bullet$-marked transition, we need to track ψ_i-escaping multitransitions separately. To implement this, we create a copy of $\bullet$ for each ψ_i and we use a specific $\blacksquare_{\psi_i}$ mark to label transitions corresponding to ψ_i-escaping multitransitions.

Unfortunately, due to the duality of the F and G operators and the fact that the transition relation of SLAA is naturally in disjunctive normal form (which is suitable for F), we did not find any way to improve the translation of $\mathsf{G}\psi$ if ψ is a disjunction. On the bright side, we can generalize the merging to $\mathsf{G}\bigwedge_i \psi_i$ where each ψ_i is a temporal or state formula (which can contain disjunctions). This is due to the fact that a state formula ψ_i affects only the labels of transitions with origin in the state $\mathsf{G}\bigwedge_i \psi_i$ and thus it does not create any reachable state or acceptance mark.

Formally, we first modify the translation rules introducing acceptance marks to work with marks of the form $\bullet_\psi$ as discussed above. More precisely, we change the rule for $\psi_1 \mathsf{U} \psi_2$ presented in the basic translation and the rule for $\mathsf{F}\psi$ of the F-merging translation to the following (note that the optimization reusing orange marks make no longer sense as we need to create their copies for each $\mathsf{F}\psi$ anyway).

$$\Delta(\psi_1 \mathsf{U} \psi_2, \alpha) = \mathsf{me}\big(\Delta(\psi_2, \alpha)\big) \cup \Big(\big\{(\{\bullet_{\psi_1 \mathsf{U} \psi_2}\}, \{\psi_1 \mathsf{U} \psi_2\})\big\} \otimes \mathsf{me}\big(\Delta(\psi_1, \alpha)\big)\Big)$$

$$\Delta(\mathsf{F}\psi, \alpha) = \big\{(\{\bullet_{\mathsf{F}\psi}\}, \{\mathsf{F}\psi\})\big\} \cup \cup \bigcup_{K \in \overline{\psi}} \Big(\mathsf{me}\big(\Delta_{\mathsf{NL}}(\psi_K, \alpha)\big) \cup \big\{(M_K, \{\mathsf{F}\psi\})\big\} \otimes \Delta_{\mathsf{L}}(\psi_K, \alpha)\Big),$$

where $M_K = \{\bullet^{K'}_{\mathsf{F}\psi} \mid K' \in \overline{\psi} \text{ and } K' \neq K\}$

Further, we add a specific rule for formulae $\mathsf{G}\psi$ where $\psi = \bigwedge_{\psi' \in K} \psi'$ for some set K of temporal and state formulae. Formulae $\mathsf{G}\psi$ of other forms are handled as $\mathit{ff}\,\mathsf{R}\,\psi$. On the top level, the rule simply defines transitions of $\mathsf{G}\psi$ as transitions of $\mathsf{G}\psi \ \wedge \bigwedge_{\psi' \in K} \psi'$:

$$\Delta(\mathsf{G}\psi, \alpha) = \big\{(\emptyset, \{\mathsf{G}\psi\})\big\} \otimes \bigotimes_{\psi' \in K} \Delta'(\psi', \alpha)$$

The definition of $\Delta'(\psi', \alpha)$ differs from $\Delta(\psi', \alpha)$ in two aspects. First, it removes ψ' from destination configurations because ψ' is merged with $\mathsf{G}\psi$ and the state $\mathsf{G}\psi$ is added to each destination configuration by the product on the top level. Second, it identifies all non-looping transitions of ψ' and marks them with $\blacksquare_{\psi'}$ as ψ'-escaping. We distinguish between looping and non-looping transitions only when ψ' has the form $\psi_1 \,\mathsf{U}\, \psi_2$ or $\mathsf{F}\psi_1$. All other ψ' have only looping transitions (e.g., G-formulae) or no marked transitions (e.g., state formulae or R- or X-formulae) and thus there are no ψ'-escaping transitions or we do not need to watch them. Similarly to $\mathcal{F}_\varphi$, we use $\mathcal{U}_\varphi$ for the set of all subformulae of φ of the form $\psi_1 \,\mathsf{U}\, \psi_2$. The function $\Delta'(\psi', \alpha)$ is defined as follows:

$$\Delta'(\psi', \alpha) = \begin{cases} \Delta'_{\mathsf{L}}(\psi', \alpha) \cup \Delta'_{\mathsf{NL}}(\psi', \alpha) & \text{if } \psi' \in \mathcal{F}_\varphi \text{ or } \psi' \in \mathcal{U}_\varphi \\ \big\{(M, C \smallsetminus \{\psi'\}) \mid (M, C) \in \Delta(\psi', \alpha)\big\} & \text{otherwise} \end{cases}$$

$$\Delta'_{\mathsf{L}}(\psi', \alpha) = \big\{(M, C \smallsetminus \{\psi'\}) \mid (M, C) \in \Delta(\psi', \alpha), \psi' \in C\big\}$$
$$\Delta'_{\mathsf{NL}}(\psi', \alpha) = \big\{(\{\blacksquare_{\psi'}\}, C) \mid (M, C) \in \Delta(\psi', \alpha), \psi' \notin C\big\}$$

Finally, we redefine the set of marks $\mathcal{M}$ and the acceptance formula Φ. Now each subformula from $\mathcal{U}_\varphi$ and $\mathcal{F}_\varphi$ has its own set of marks, and the $\blacksquare$ marks are used to implement the intuition given using the Figure 5.

$$\mathcal{M} = \{\bullet_\psi, \blacksquare_\psi \mid \psi \in \mathcal{U}_\varphi\} \cup \Big\{\bullet_{\mathsf{F}\psi}, \blacksquare_{\mathsf{F}\psi}, \bullet^K_{\mathsf{F}\psi} \mid \mathsf{F}\psi \in \mathcal{F}_\varphi \text{ and } K \in \overline{\psi}\Big\}$$

$$\Phi = \bigwedge_{\psi \in \mathcal{U}_\varphi} \big(\mathsf{Fin}\bullet_\psi \vee \mathsf{Inf}\blacksquare_\psi\big) \wedge \bigwedge_{\mathsf{F}\psi \in \mathcal{F}_\varphi} \left(\Big(\mathsf{Fin}\bullet_{\mathsf{F}\psi} \wedge \bigvee_{K \in \overline{\psi}} \mathsf{Fin}\bullet^K_{\mathsf{F}\psi}\Big) \vee \mathsf{Inf}\blacksquare_{\mathsf{F}\psi}\right)$$

5.1. Correctness and Complexity

Here we prove that the F,G-merging translation produces correct automata and we analyse the size of these automata. For the rest of this section, let φ be an arbitrary but fixed LTL formula and let $\mathcal{A}_\varphi$ be the corresponding SLAA over alphabet $\Sigma = 2^{AP(\varphi)}$ built by the F,G-merging translation. For any subformula ψ of φ, by $\mathcal{A}_\psi$ we mean the automaton identical to $\mathcal{A}_\varphi$ except for the initial state, which is ψ instead of φ.

In the proof of correctness, we often compose runs of automata for some subformulae of φ over suffixes of w. The following auxiliary lemma says that one can select runs that are so-called *synchronized*, which means that whenever the runs read the same suffix from the same state, they use the same transitions.

Lemma 1. *Let $w_{i..}$ and $w_{j..}$ be two suffixes of some $w \in \Sigma^\omega$. Further, let ψ_1, ψ_2 be subformulae of φ such that $w_{i..} \in L(\mathcal{A}_{\psi_1})$ and $w_{j..} \in L(\mathcal{A}_{\psi_2})$. Then there always exist an accepting run $R_0 R_1 \dots$ of $\mathcal{A}_{\psi_1}$ over $w_{i..}$ and an accepting run $S_0 S_1 \dots$ of $\mathcal{A}_{\psi_2}$ over $w_{j..}$ such that the two runs are* synchronized*, i.e., for each $k \geq j$ we have that $T_k = R_{k-i} \cup S_{k-j}$ is a multitransition of $\mathcal{A}_\varphi$.*

PROOF. Consider an accepting run $R_0R_1\dots$ of $\mathcal{A}_{\psi_1}$ over $w_{i..}$ and an accepting run $S_0S_1\dots$ of $\mathcal{A}_{\psi_2}$ over $w_{j..}$. The runs are not synchronized if and only if there exists some $k \geq j$ such that both runs reach the same state $q \in \mathrm{dom}(R_{k-i}) \cup \mathrm{dom}(S_{k-j})$ before reading $w_{k..}$ and each run uses a different transition leading from q. If we modify one of these runs to use the same transition as the other run whenever such situation arises, we get two synchronized runs. Note that the modified run is still accepting as the other run was accepting. □

The lemma can be easily extended to any finite number of suffixes. Now we are ready to formulate and prove the main theorem.

Theorem 1. *The* F,G*-merging translation is correct, i.e.,* $L(\mathcal{A}_\varphi) = L(\varphi)$*. Further, the number of states of* $\mathcal{A}_\varphi$ *is linear to the size of* φ *and the number of acceptance marks is at most exponential to the size of* φ*.*

PROOF. We prove the correctness part of the theorem by structural induction. We suppose that for each strict subformula ψ of φ, the SLAA $\mathcal{A}_\psi$ represents the language $L(\psi)$ over alphabet Σ.

We use the following notation. Let T be a multitransition and S be a set of states. Then $T[S] = \{t = (s, \alpha, M, C) \mid t \in T \text{ and } s \in S\}$ is the set of transitions from T with a source from S, and $T[\bar{S}] = T \setminus T[S]$. For $K \in \overline{\psi}$, we again use the notation $\psi_K = \bigwedge_{\psi' \in K} \psi'$.

To prove $L(\varphi) \subseteq L(\mathcal{A}_\varphi)$, let $w \in \Sigma^\omega$ be a word such that $w \models \varphi$. We find an accepting run $\rho = T_0T_1\dots$ of $\mathcal{A}_\varphi$ over w for each possible form of φ. In the following, when we choose multiple accepting runs over suffixes of some word, we always assume they are synchronized. This assumption is valid due to Lemma 1.

If φ is one of tt, a, or $\neg a$ where $a \in AP$, we have $T_0 = \{(\varphi, w_0, \emptyset, \emptyset)\}$ and $T_i = \emptyset$ for $i \geq 1$. The transition in T_0 exists in $\mathcal{A}_\varphi$ by definition and the run is accepting since it has no infinite branch. If $\varphi \equiv ff$, then no word satisfies φ and there is nothing to prove.

Let $\varphi \equiv \psi_1 \vee \psi_2$. We can assume that $w \models \psi_1$ (the proof for the case $w \models \psi_2$ is analogous). Therefore, there is an accepting run σ of $\mathcal{A}_{\psi_1}$ over w, and we define ρ as $\rho = \{(\varphi, w_0, \emptyset, C)\}T_1T_2\dots$ where $\sigma = \{(\psi_1, w_0, M, C)\}T_1T_2$. The run ρ is accepting as it has branches isomorphic to branches of σ.

Let $\varphi \equiv \psi_1 \wedge \psi_2$. Then we can assume two synchronized runs $R_0R_1\dots$ of $\mathcal{A}_{\psi_1}$ and $S_0S_1\dots$ of $\mathcal{A}_{\psi_2}$ over w with $R_0 = \{(\psi_1, w_0, M_1, C_1)\}$ and $S_0 = \{(\psi_2, w_0, M_2, C_2)\}$. We define $T_0 = \{(\varphi, w_0, \emptyset, C_1 \cup C_2)\}$ and $T_i = R_i \cup S_i$ for $i > 0$. The run ρ is a union of branches of the two runs and thus is accepting.

If $\varphi \equiv \mathsf{X}\psi$, then we create ρ as $\{(\varphi, w_0, \emptyset, \{\psi\})\}T_1T_2\dots$ where $T_1T_2\dots$ is an accepting run of $\mathcal{A}_\psi$ over $w_{1..}$.

Let $\varphi \equiv \psi_1 \,\mathsf{U}\, \psi_2$ where $\psi_1 \not\equiv tt$ and let $k \geq 0$ be the smallest index such that $w_{k..} \models \psi_2$. We know that $w_{j..} \models \psi_1$ for all $j < k$ as $w \models \varphi$, and thus for each such j there exist an accepting run $\sigma^j = S_0^jS_1^j\dots$ of $\mathcal{A}_{\psi_1}$ over $w_{j..}$. Further, there is an accepting run $\sigma^k = S_0^kS_1^k\dots$ of $\mathcal{A}_{\psi_2}$ over $w_{k..}$. We define

the multitransitions of ρ as follows.

$$T_i = \bigcup_{\substack{0 \le j < i \\ j < k}} S^j_{i-j} \cup \begin{cases} \{(\varphi, w_i, \{\bullet_\varphi\}, C \cup \{\varphi\}) \mid (\psi_1, w_i, M, C) \in S^i_0\} & i < k \\ \{(\varphi, w_i, \emptyset, C) \mid (\psi_2, w_i, M, C) \in S^k_0\} & i = k \\ S^k_{i-k} & i > k \end{cases}$$

The run ρ contains the branches of all σ^j for $j \le k$, some of them prefixed with finitely many $\bullet_\varphi$-labelled edges between nodes of the form (φ, i). Since $\bullet_\varphi$ only appears in the first k multitransitions, $\mathsf{Fin}\bullet_\varphi$ is satisfied, the rest of the acceptance formula is satisfied as we reuse the branches of accepting runs.

If $\varphi \equiv \mathsf{F}\psi$, we again denote by k the smallest index such that $w_{k..} \models \psi_K$ for some $K \in \overline{\psi}$. Let $M_K = \{\bullet^{K'}_\varphi \mid K \neq K' \in \overline{\psi}\}$. By the induction hypothesis, there exists a run $\sigma = S_0 S_1 \dots$ of $\mathcal{A}_{\psi_K}$ over $w_{k..}$. Let l be the minimal index $l \ge k$ such that $K \not\subseteq \mathrm{range}(S_l[K])$ if exists and $l = \infty$ otherwise. In the following definition, we exceptionally redefine $S_0[K] = S_0$ and $S_0[\bar{K}] = \emptyset$ in the special case of $k = 0$. To mitigate the implementation of the trick with looping transitions of ψ_K, for each $k \le i < l$ we define $C_i = \mathrm{range}(S_i[K]) \setminus K$. We can finally define the multitransitions of ρ as follows.

$$T_i = \begin{cases} \{(\varphi, w_i, \{\bullet_\varphi\}, \{\varphi\})\} & i < k \\ \{(\varphi, w_i, M_K, C_i \cup \{\varphi\})\} \cup S_i[\bar{K}] & k \le i < l \\ \{(\varphi, w_i, \emptyset, \mathrm{range}(S_i[K]))\} \cup S_i[\bar{K}] & i = l \\ S_i & i > l \end{cases}$$

The run ρ is accepting because $\bullet_\varphi$ only appears in first k multitransitions, $\bullet^K_\varphi$ does not appear anywhere in the run, and the rest of the condition inherited from automata for subformulae is trivially satisfied by the branch that stays in φ forever and is satisfied on the other branches as they only contain marks of $\mathcal{A}_{\psi_K}$ and they mimic (using $S_i[\bar{K}]$) the accepting run σ.

Let $\varphi \equiv \psi_1 \,\mathsf{R}\, \psi_2$ such that $\psi_1 \not\equiv \mathit{ff}$ or ψ_2 is not a conjunction of temporal and state formulae. Let k be the smallest number such that $w_{k..} \models \psi_1 \wedge \psi_2$ and that for all $i < k$ we have $w_{i..} \models \psi_2$. If no such k exists, we set $k = \infty$. Similarly as for the U-case, from the induction principle we have accepting runs $\sigma^i = S^i_0 S^i_1 \dots$ of $\mathcal{A}_{\psi_2}$ over $w_{i..}$ for $0 \le i \le k$ and an accepting run $\sigma = S_0 S_1 \dots$ of $\mathcal{A}_{\psi_1}$ over $w_{k..}$. The multitransitions of ρ are defined as follows.

$$T_i = \bigcup_{j=0}^{i-1} S^j_{i-j} \cup \begin{cases} \{(\varphi, w_i, \emptyset, C \cup \{\varphi\}) \mid (\psi_2, w_i, M, C) \in S^i_0\} & i < k \\ \{(\varphi, w_k, \emptyset, \mathrm{range}(S_0) \cup \mathrm{range}(S^k_0))\} & i = k \\ S_{i-k} & i > k \end{cases}$$

Finally, let $\varphi \equiv \mathsf{G} \bigwedge_{\psi \in K} \psi$ where each $\psi \in K$ is a temporal or state formula. For every $\psi \in K$ and every $i \ge 0$ we know that $w_{i..} \models \psi$ and thus by induction hypothesis we have accepting runs $\sigma^{\psi,i} = S^{\psi,i}_0 S^{\psi,i}_1 \dots$ of $\mathcal{A}_\psi$ over $w_{i..}$. Further, we may assume that these runs are synchronized due to Lemma 1, and thus $S^{\psi,i}_0 \subseteq S^{\psi,i-j}_j$ for all $j \le i$. Similarly to the $\mathsf{F}\psi'$ case, we define $C^\psi_i = \mathrm{range}(S^{\psi,i}_0) \setminus \{\psi\}$.

Moreover, assuming $S_0^{\psi,i} = \{(\psi, w_i, M, C)\}$, we define $M_i^\psi = \{\blacksquare_\psi\}$ if $S_0^{\psi,i}$ is ψ-escaping and $\psi \in \mathcal{F}_\varphi \cup \mathcal{U}_\varphi$, and $M_i^\psi = M$ otherwise.

$$T_i = \{(\varphi, w_i, M_i, C_i)\} \cup \bigcup_{\substack{\psi \in K \\ 0 \le j < i}} S_{i-j}^{\psi,j}[\overline{\{\psi\}}], \text{ where}$$

$$M_i = \bigcup_{\psi \in K} M_i^\psi \qquad C_i = \{\varphi\} \cup \bigcup_{\psi \in K} C_i^\psi$$

The acceptance formula is satisfied by ρ as, for each $\psi \in K \cap (\mathcal{U} \cup \mathcal{F})$ the marks $\bullet_\psi$ (and some $\bullet_\psi^S$) appear only in finitely many multitransitions of ρ, or there are infinitely many ψ-escaping multitransitions and we have $\mathsf{Inf}\blacksquare_\psi$ satisfied by the branch staying in φ. The rest of the formula is satisfied by ρ because it was satisfied by all the branches of runs σ_i^ψ.

To prove $L(\mathcal{A}_\varphi) \subseteq L(\varphi)$, we show that if there exists an accepting run $\rho = T_0 T_1 \dots$ of $\mathcal{A}_\varphi$ over w, then $w \models \varphi$. For a run $\sigma = T_0 T_1 \dots$ of $\mathcal{A}_\varphi$, a set of states C, and an index $i \ge 0$ we use $T_i[C] \dots$ to describe the run $\sigma' = T_i[C] T_{i+1}[\mathrm{range}(T_i[C])] \dots$ of $\mathcal{A}_\psi$ over $w_{i..}$ where $\psi = \bigwedge_{\psi' \in C \cap \mathrm{dom}(T_i)} \psi'$. If σ is accepting, σ' is also accepting as it contains only suffixes of branches of σ.

The statement is trivially true for $\varphi \equiv tt$ (every word satisfies tt) and $\varphi \equiv ff$ (there cannot exist an accepting run over $\mathcal{A}_{ff}$).

If $\varphi \equiv a$, then surely T_0 is an w_0-labelled multitransition such that $a \in w_0$, hence $w \models \varphi$. Similar argument applies to $\varphi \equiv \neg a$.

Let $\varphi \equiv \psi_1 \vee \psi_2$. Then ρ is (up to $\mathrm{dom}(T_0)$) equivalent to (also accepting) run of $\mathcal{A}_{\psi_1}$ or $\mathcal{A}_{\psi_2}$ over w. Therefore $w \models \psi_1$ or $w \models \psi_2$, so $w \models \varphi$.

If $\varphi \equiv \psi_1 \wedge \psi_2$, then, by definition of Δ, ρ consists exactly of branches of runs of $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$, again with the only difference in $\mathrm{dom}(T_0)$, all of which have to be accepting. Therefore, by the induction hypothesis, $w \models \psi_1$ and $w \models \psi_2$, so $w \models \varphi$.

Let $\varphi \equiv \mathsf{X}\psi$. By the definition of Δ, $T_1 T_2 \dots$ is an accepting run of $\mathcal{A}_\psi$ over $w_{1..}$, thus $w_{1..} \models \psi$ and so $w \models \varphi$.

Let $\varphi \equiv \psi_1 \,\mathsf{U}\, \psi_2$. Since ρ is accepting but, by the definition of Δ, no $\blacksquare_\varphi$ or $\bullet_\varphi^K$ appear in the run. Therefore $\mathsf{Fin}\bullet_\varphi$ has to hold. We can see that every outgoing transition of φ is looping if and only if its acceptance label contains $\bullet_\varphi$, so there exists some $k \in \mathbb{N}$ such that, after k steps, the run escapes from φ and, by the definition of Δ, the run $T_k T_{k+1} \dots$ is an accepting run of $\mathcal{A}_{\psi_2}$ over $w_{k..}$ and thus $w_{k..} \models \psi_2$ by induction hypothesis. Moreover, for each $i < k$ the multitransition T_i contains (by definition of Δ) a transition $(\varphi, w_i, \{\bullet_\varphi\}, \{\varphi\} \cup C)$ such that $\{(\psi_1, w_i, M, C)\} T_{i+1}[C] \dots$ for some M is an accepting run of $\mathcal{A}_{\psi_1}$ over $w_{i..}$, so $w_{i..} \models \psi_1$.

If $\varphi \equiv \mathsf{F}\psi$, then again no $\blacksquare_\varphi$ appears in the run, so both $\mathsf{Fin}\bullet_\varphi$ and $\mathsf{Fin}\bullet_\varphi^K$ for some $K \in \overline{\psi}$ holds. Therefore there exists $k \in \mathbb{N}$ such that neither $\bullet_\varphi$ nor $\bullet_\varphi^K$ appears in $T_k T_{k+1} \dots$. This happens because of two reasons: either the run eventually leaves the state φ using some transition from $\psi_{K'}$ (where $K' \in \overline{\psi}$),

or we only use transitions from ψ_K (with ψ_K replaced by φ). In both cases, $w_k \models \psi$ so $w \models \varphi$.

If $\varphi \equiv \psi_1 \,\mathsf{R}\, \psi_2$, each T_i that contains φ contains a transition $(\varphi, w_i, \emptyset, C_1 \cup C_2)$ such that $\{(\psi_2, w_i, M, C_2)\}T_{i+1}[C_2]\ldots$ for some M is an accepting run of $\mathcal{A}_{\psi_2}$ over $w_{i..}$, hence $w_{i..} \models \psi_2$. If $C_1 = \{\varphi\}$ for all i, we have that $w \models \varphi$. Otherwise, there is a minimal $k \geq 0$ such that $\{(\psi_1, w_k, M, C_1)\}T_{k+1}[C_1]\ldots$ for some M is an accepting run of $\mathcal{A}_{\psi_1}$ over $w_{k..}$, hence $w_{k..} \models \psi_1$ and again, $w \models \varphi$.

Finally, let $\varphi \equiv \mathsf{G} \bigwedge_{\psi \in K} \psi$ where $\psi \in K$ are all temporal or state formulae. For each $\psi \in K$ and each i we construct an accepting run σ_i^ψ of $\mathcal{A}_\psi$ over $w_{i..}$. By definition of Δ we know that in each T_i there is a transition of the form (φ, w_i, M_i, C_i). Let $C_i^\psi = \{\psi\}$ if $\blacksquare_\psi \notin M_i$, and $C_i^\psi = \emptyset$ otherwise. By definition of Δ, there is some $C_i' \subseteq C_i$ such that $t = (\psi, w_i, M_i', C_i' \cup C_i^\psi)$ is a transition of $\mathcal{A}_\psi$ for some M_i' which is arbitrary if $\blacksquare_\psi \in M_i$ and $M_i' \subseteq M_i$ otherwise. If $C_i^\psi = \emptyset$ then $\sigma_i^\psi = \{t\}T_{i+1}[C_i']\ldots$, and otherwise $\sigma_i^\psi = \{t\}(T_{i+1}[C_i']\ldots \sqcup \sigma_{i+1}^\psi)$ where $\sqcup$ makes a union of corresponding multitransitions (multitransitions on the same positions) of two runs.

Each σ_i^ψ consists of a subset of branches of ρ (or their suffixes) and some branches that loop in ψ for a while. If the branch stays forever in ψ then its marks are exactly those marks of ρ relevant to ψ (have the ψ subscript) and such branch must satisfy the accepting formula of $\mathcal{A}_\psi$. Branches that eventually leave ψ have suffix that is already in ρ and thus must satisfy the acceptance formula too. Hence, σ_i^ψ is an accepting run of $\mathcal{A}_\psi$ over $w_{i..}$ and we have that $w_{i..} \models \varphi$ for each i and ψ, hence $w \models \varphi$.

The linear number of states of $\mathcal{A}_\varphi$ with respect to the size of φ comes from the fact that states of $\mathcal{A}_\varphi$ are subformulae of φ. Let n be the number of states of $\mathcal{A}_\varphi$. Then we have at most n marks of the form $\bullet_\psi$, at most n marks of the form $\blacksquare_\psi$, and finally at most 2^n marks of the form $\bullet_\psi^K$ (the exponential blowup may be caused only by conversion to DNF, if necessary). Overall, the number of marks is at most exponential with respect to size of φ. □

6. SLAA Simplifications

We start with SLAA simplifications based on *transition dominance*, which is a standard technique of improving various automata constructions [13]. In SLAA, a transition $t_1 = (q, \alpha, M_1, C_1)$ *dominates* a different transition $t_2 = (q, \alpha, M_2, C_2)$ if $C_1 \subseteq C_2$ and M_1 is "at least as helpful and at most as harmful for acceptance" as M_2. Any dominated transition can be removed without changing the language of the produced automaton. This automata simplification is usually applied *on-the-fly*, i.e., during the automata construction. In other words, the construction does not add any transition that is dominated by a transition already present in the automaton. Moreover, when it adds a transition, all transitions dominated by the new transition are removed.

This section presents two versions of transition dominance that differ in the check of the condition "at least as helpful and at most as harmful for acceptance".

Basic dominance is a direct extension of dominance for Büchi and co-Büchi automata and *acceptance-aware dominance* improves the basic dominance by utilizing the current acceptance formula.[1] Although we formulate the dominance relations for SLAA, they can be used for arbitrary (alternating or nonalternating) TELA.

We further present a simplification called *destination configuration trimming*, which is tailored to our LTL to SLAA translations and which often amplifies the effect of dominance-based simplifications. This simplification employs the semantic labels of the created states, but it can be generalized to any SLAA construction that allows to easily decide language inclusion of states.

6.1. Basic Transition Dominance

In the classic case of co-Büchi SLAA with acceptance formula $\mathsf{Fin}(\textcolor{green}{\bullet})$, the condition M_1 is "at least as helpful and at most as harmful for acceptance" as M_2 translates into $\textcolor{green}{\bullet} \in M_1 \implies \textcolor{green}{\bullet} \in M_2$. For Büchi SLAA with acceptance formula $\mathsf{Inf}(\textcolor{blue}{\blacksquare})$, the condition has the form $\textcolor{blue}{\blacksquare} \in M_2 \implies \textcolor{blue}{\blacksquare} \in M_1$.

Now consider an SLAA with an arbitrary acceptance formula Φ. Let $\mathsf{Fin}(\Phi)$ and $\mathsf{Inf}(\Phi)$ be the sets of all acceptance marks appearing in Φ in subformulae of the form $\mathsf{Fin}(\textcolor{green}{\bullet})$ and $\mathsf{Inf}(\textcolor{blue}{\blacksquare})$, respectively. We say that a transition (q, α, M_1, C_1) *dominates* another transition (q, α, M_2, C_2) iff $C_1 \subseteq C_2$ and

$$M_1 \cap \mathsf{Fin}(\Phi) \subseteq M_2 \quad \text{and} \quad M_2 \cap \mathsf{Inf}(\Phi) \subseteq M_1.$$

With this formulation, we can simplify the part of an SLAA in Figure 6 by removing the dotted transition t_2 that is dominated by the solid transition t_1. Indeed, each accepting run ρ that uses t_2 has an analogous run ρ' that uses t_1 instead. Clearly, ρ' is accepting since all its branches are also branches of ρ, only some occurrences of $\textcolor{green}{1}$ on these branches are replaced by $\textcolor{blue}{2}$.

6.2. Acceptance-Aware Transition Dominance

While the basic transition dominance is correct (in the sense that it does not change the language of the produced automaton), it does not work well with more complicated acceptance formulae. Consider the example in Figure 7 with the acceptance formula $\Phi = (\mathsf{Fin}(\textcolor{green}{1}) \wedge \mathsf{Inf}(\textcolor{blue}{2})) \vee \mathsf{Fin}(\textcolor{orange}{3})$. The solid transition t_1 does not dominate t_2 by the definition of basic dominance as $\{\textcolor{blue}{2}\} \cap \mathsf{Inf}(\Phi) = \{\textcolor{blue}{2}\} \not\subseteq \emptyset$. However, if there is some accepting branch that uses t_2 infinitely often, it does not satisfy the disjunct $(\mathsf{Fin}(\textcolor{green}{1}) \wedge \mathsf{Inf}(\textcolor{blue}{2}))$ due to $\textcolor{green}{1}$, and thus must satisfy $\mathsf{Fin}(\textcolor{orange}{3})$. Therefore, the mark $\textcolor{blue}{2}$ missing in t_1 does not matter for dominance and we only care about marks that appear in the subformula $\mathsf{Fin}(\textcolor{orange}{3})$. To formalize this observation, we introduce *transition dominance with respect to acceptance formula*.

A *minimal model* O of an acceptance formula Φ is a subset of its terms satisfying Φ and such that no proper subset of O is a model of Φ. For example,

[1] Note that the first version of formula-aware dominance presented at ICTAC 2019 [5] is incorrect as it can modify the language of a simplified automaton.

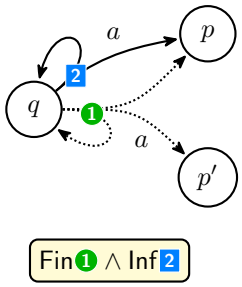

Figure 6: The effect of the basic transition dominance: the dotted transition t_2 is removed as it is dominated by the solid transition t_1.

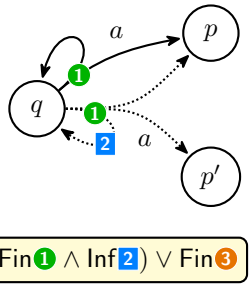

Figure 7: The effect of the acceptance-aware transition dominance: the dotted transition t_2 is removed as it is dominated by the solid transition t_1.

the formula $(\mathsf{Fin}\textcircled{1} \wedge \mathsf{Inf}\textcolor{blue}{\blacksquare}2) \vee \mathsf{Fin}\textcircled{3}$ has two minimal models: $\{\mathsf{Fin}\textcircled{1}, \mathsf{Inf}\textcolor{blue}{\blacksquare}2\}$ and $\{\mathsf{Fin}\textcircled{3}\}$. If we switch $\vee$ and $\wedge$, the formula $(\mathsf{Fin}\textcircled{1} \vee \mathsf{Inf}\textcolor{blue}{\blacksquare}2) \wedge \mathsf{Fin}\textcircled{3}$ has again two minimal models, but now they are $\{\mathsf{Fin}\textcircled{1}, \mathsf{Fin}\textcircled{3}\}$ and $\{\mathsf{Inf}\textcolor{blue}{\blacksquare}2, \mathsf{Fin}\textcircled{3}\}$. For each minimal model O, by $\mathsf{Fin}(O)$ and $\mathsf{Inf}(O)$ we denote the sets of all acceptance marks appearing in O in terms of the form $\mathsf{Fin}\textcolor{green}{\bullet}$ and $\mathsf{Inf}\textcolor{blue}{\blacksquare}$, respectively. Intuitively, the formula-aware dominance checks only those minimal models of Φ that are not broken by some Fin-mark on t_2. Formally, a transition (q, α, M_1, C_1) *dominates* another transition (q, α, M_2, C_2) *with respect to* Φ iff $C_1 \subseteq C_2$ and for each minimal model O of Φ it holds

$$M_2 \cap \mathsf{Fin}(O) = \emptyset \implies \big(M_1 \cap \mathsf{Fin}(O) = \emptyset \text{ and } M_2 \cap \mathsf{Inf}(O) \subseteq M_1\big).$$

The solid transition in Figure 7 dominates with respect to the given acceptance formula the dotted transition.

6.3. Destination Configuration Trimming

The last simplification we present *trims* the destination configurations of transitions. It is based on the LTL semantics that the states carry. In a transition (s, α, M, C) with $\{\psi, \mathsf{F}\psi\} \subseteq C$, we can replace C with $C \setminus \{\mathsf{F}\psi\}$ as the satisfaction of $\mathsf{F}\psi$ is ensured by visiting ψ anyway. This approach naturally extends to transitions with $\{\psi, \mathsf{F}(\psi \vee \rho)\} \subseteq C$, where we remove $\mathsf{F}(\psi \vee \rho)$ from C. Symmetrically, if $\{\psi, \mathsf{G}(\psi \wedge \rho)\} \subseteq C$ then we replace C with $C \setminus \{\psi\}$ as the satisfaction of ψ is enforced by the state $\mathsf{G}(\psi \wedge \rho)$.

This simplification is not only beneficial on its own, it also supports (any kind of) transition dominance by increasing the potential of transitions for dominating other transitions. Consider, for example, the three automata for the formula $\varphi = \mathsf{G}(a \wedge (b \,\mathsf{U}\, \mathsf{X}a))$ in Figure 8. The automaton with three states (on the left) is produced by the basic translation and can be reduced to two states using the F,G-merging technique (in the middle). Trimming the destination configuration $\{\varphi, a\}$ of the transitions represented by the edge labeled by a in the middle automaton to $\{\varphi\}$, has two effects: (i) the state a becomes unreachable, and

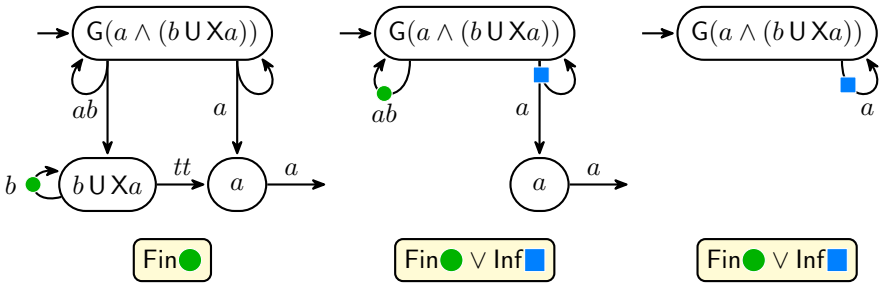

Figure 8: SLAA for the formula $\mathsf{G}(a \wedge (b \,\mathsf{U}\, \mathsf{X}a))$ produced by the basic translation (left), F,G-merging (middle), and F,G-merging with destination configuration trimming and basic transition dominance (right).

(ii) it triggers transition dominance removing the edge labeled by ab and the automaton even becomes deterministic. The resulting automaton (on the right) is the minimal automaton for the formula $\mathsf{G}a$, which is indeed equivalent to the original formula.

7. Translation of SLAA to LTL

We have already shown that every LTL formula can be translated into an equivalent SLAA. This section shows a translation of SLAA to equivalent LTL formulae. Before we start, we need to solve one technical issue connected to alphabets used by the two formalisms. While SLAA can work with any finite alphabet Σ, LTL works with subsets of 2^{AP}. To bridge this gap, in the following we assume that for every $a \in \Sigma$ there exists a state formula φ_a satisfied exactly by the words of Σ^ω starting with a. For example, if $\Sigma = 2^{AP'}$ for some finite set of atomic propositions $AP' \subseteq AP$, then φ_α can be defined as

$$\varphi_\alpha = \bigwedge_{a \in \alpha} a \quad \wedge \bigwedge_{a \in AP' \setminus \{\alpha\}} \neg a.$$

Theorem 2. *For every SLAA $\mathcal{A}$ there exists an LTL formula φ such that $L(\mathcal{A}) = L(\varphi)$.*

Proof. Let $\mathcal{A} = (S, \Sigma, \mathcal{M}, \Delta, s_I, \Phi)$ be an SLAA. For each state $s \in S$ we construct an LTL formula $\varphi(s)$ such that $u \in \Sigma^\omega$ satisfies $\varphi(s)$ if and only if u is accepted by $\mathcal{A}$ with its initial state replaced by s. Hence, $\mathcal{A}$ is equivalent to the formula $\varphi(s_I)$.

The formula construction is inductive. Let s be a state such that we have already constructed LTL formulae for all its successors. Further, let $\mathit{Mod}(\Phi)$ denote the set of all minimal models of Φ. Further, given a set of states C, by

$\varphi(C)$ we denote the conjunction $\varphi(C) = \bigwedge_{s \in C} \varphi(s)$. We define $\varphi(s)$ as follows:

$$\varphi(s) = \varphi_1(s) \vee \big(\varphi_2(s) \wedge \varphi_3(s)\big)$$

$$\varphi_1(s) = \bigvee_{\substack{(s,\alpha,M,C)\in\Delta \\ s\in C}} \varphi_\alpha \wedge \mathsf{X}\varphi(C \smallsetminus \{s\}) \quad \mathsf{U} \bigvee_{\substack{(s,\alpha,M,C)\in\Delta \\ s\notin C}} \varphi_\alpha \wedge \mathsf{X}\varphi(C)$$

$$\varphi_2(s) = \mathsf{G} \bigvee_{\substack{(s,\alpha,M,C)\in\Delta \\ s\in C}} \varphi_\alpha \wedge \mathsf{X}\varphi(C \smallsetminus \{s\})$$

$$\varphi_3(s) = \bigvee_{O\in Mod(\Phi)} \Bigg(\bigwedge_{\blacksquare\in \mathsf{Inf}(O)} \Big(\mathsf{GF} \bigvee_{\substack{(s,\alpha,M,C)\in\Delta \\ s\in C,\ \blacksquare\in M}} \varphi_\alpha \wedge \mathsf{X}\varphi(C \smallsetminus \{s\}) \Big) \wedge \bigwedge_{\bullet\in \mathsf{Fin}(O)} \Big(\mathsf{FG} \bigvee_{\substack{(s,\alpha,M,C)\in\Delta \\ s\in C,\ \bullet\notin M}} \varphi_\alpha \wedge \mathsf{X}\varphi(C \smallsetminus \{s\}) \Big) \Bigg)$$

Intuitively, $\varphi_1(s)$ covers the case when a run leaves s after a finite number of looping transitions. Indeed, the *until* operator ensures that looping transitions (corresponding to the left subformula) are taken only finitely many times and then the state is left by some non-looping transition (corresponding to the right subformula). Note that $\varphi_1(s)$ completely ignores acceptance marks on transitions of s as they play no role in acceptance of runs leaving s.

Further, $\varphi_2(s)$ describes the case when a run never leaves s. In this case, $\varphi_3(s)$ ensures that the infinite branch of the run staying in s forever is accepting. More precisely, $\varphi_3(s)$ says that there exists some minimal model O of the acceptance condition Φ such that each mark in $\mathsf{Inf}(O)$ appears infinitely often (the GF-subterm) on the branch and each mark in $\mathsf{Fin}(O)$ does not appear on the branch since some moment (the FG-subterm). □

Theorems 1 and 2 imply that LTL and the class of SLAA with alphabets of the form $2^{AP'}$ for a finite set $AP' \subseteq AP$ have the same expressive power.

8. Experimental Evaluation

We have implemented the presented translations in the tool *ltl3tela* [21]. The tool is built on top of the Spot library [9] and is released under the GNU GPL 3.0 license at `https://github.com/jurajmajor/ltl3tela`. In addition to the algorithms presented in this paper, the tool also offers translations of LTL to nondeterministic or deterministic TELA. More information about the tool and its usage for LTL to SLAA translation can be found in Appendix A.

This section provides basically three experimental comparisons. The first one focuses on LTL to SLAA translations and compares the three translations presented in this paper and the ltl3ba tool. One of the outcomes of this comparison is that the F,G-translation combined with the simplifications of Section 6

often produces SLAA without universal branching. These automata can be nondeterministic or even deterministic, altogether, we call them *nonalternating*. The non-trivial number of nonalternating automata produced by our algorithm motivates the second comparison — we let Spot further reduce these automata and then we compare them with the automata produced by state-of-the-art translators of LTL to deterministic or nondeterministic automata. Finally, the third comparison returns back to translations of LTL to SLAA, but this time we consider parametric formulae. This comparison aims to investigate the behavior of our algorithms on large formulae. All scripts and formulae used for the evaluations presented below are available in a Jupyter notebook.[2]

8.1. Settings of Comparisons on Nonparametric Formulae

The translation improvements presented in Sections 4 and 5 can have some impact only on formulae that contain at least one $\mathsf{F}\psi$ subformula where ψ contains some temporal subformula, or at least one $\mathsf{G}\bigwedge_i \psi_i$ subformula where some ψ_i is temporal. We call such formulae *mergeable*. We first evaluate how likely we can obtain a formula that is mergeable in Section 8.2, and then we present the impact of our merging technique on mergeable formulae in Sections 8.3 (SLAA) and 8.4 (nonalternating automata). For that, we consider formulae that come from two sources.

1. We use formulae collected from literature [10, 12, 16, 24, 29] that can be obtained using the tool `genltl` from Spot [8]. For each such a formula, we consider also its negation. We simplified all the formulae and removed duplicates and formulae equivalent to tt or ff. The resulting benchmark set contains 221 formulae.
2. We run the tool `randltl` from Spot to generate random formulae. We generate formulae with up to five atomic propositions and with tree size equal to 15 (the default settings of `randltl`) before simplifications. The generator allows the user to specify *priority* for each LTL operator which determines how likely the given operator appears in the generated formulae. By default, all operators (boolean and temporal) have priority 1 in `randltl`. We consider 4 different sets of random formulae. For the sets *rnd1*, *rnd2*, and *rnd4*, the number indicates the priority that was used for F and G and sets 0 to all other temporal operators. We generate 1000 formulae with each priority setting to study mergeability in Section 8.2. For Sections 8.3 and 8.4, we generate for each priority setting 1000 formulae that are mergeable (and throw away the rest).

The experimental evaluation employs five different LTL to automata translators, namely ltl3ba 1.1.3, ltl3tela 2.2.1, Spot 2.9 (more precisely, the translator `ltl2tgba` of the Spot library), and two translators Delag and ltl2dgra (formerly known as Rabinizer 4) from the library Owl 19.06.3.

[2] `https://github.com/jurajmajor/ltl3tela/blob/master/Experiments/TCS.ipynb`

Table 1: Comparison of LTL to SLAA translations on mergeable formulae.

		ltl3ba	basic	F-merg.	F,G-merg.	F,G+simpl.
states	lit (24)	140	140	110	65	65
	rnd1 (1000)	6253	6234	5417	4582	4573
	rnd2 (1000)	6313	6317	5317	4291	4276
	rnd4 (1000)	6412	6389	5213	3988	3961
	rndfg (1000)	5051	5068	3936	2656	2656
edges	lit (24)	330	358	285	214	210
	rnd1 (1000)	13103	14164	12089	10364	10191
	rnd2 (1000)	13195	14755	11911	10092	9821
	rnd4 (1000)	13395	15314	11925	9795	9417
	rndfg (1000)	9546	11005	8378	6757	6461
acc. marks	lit (24)	24	24	46	98	98
	rnd1 (1000)	1000	997	1160	2973	2973
	rnd2 (1000)	1000	1000	1248	3334	3334
	rnd4 (1000)	1000	1000	1366	3693	3693
	rndfg (1000)	1000	1000	1363	3009	3009

8.2. Mergeability

The set of formulae from literature contains mainly quite simple formulae. As a result, only 24 out of the 221 formulae are mergeable. We refer to the set of these 24 formulae as *lit*. For *rnd1*, *rnd2*, *rnd4*, and *rndfg* we have 302, 488, 697, and 802 mergeable formulae out of 1000, respectively. Consistently with intuition, the ratio of mergeable formulae increases considerably with F and G being more frequent. Recall that *rnd1*, *rnd2*, *rnd4*, and *rndfg* refer to sets of 1000 mergeable formulae since now on.

8.3. Comparison of LTL to SLAA Translations

We compare the SLAA produced by the basic translation of Section 3, the F-merging translation of Section 4, the F,G-merging translation of Section 5, and the F,G-merging translation combined with simplifications of Sections 6.2 and 6.3 as implemented in ltl3tela, and the SLAA produced by ltl3ba [2]. The exact options used in the execution of the translators can be found in Table B.5 in Appendix B. In particular, ltl3tela is always set to use formula preprocessing and the basic transition dominance (except the instance for the F,G-merging translation with simplifications, which applies the acceptance-aware transition dominance and the destination configuration trimming). Due to these settings, ltl3tela's behavior is more consistent with ltl3ba which also applies formula rewriting rules and basic transition dominance.

Table 1 shows the cumulative numbers of states, edges, and acceptance marks of SLAA produced by the individual translations on the considered sets of mergeable formulae. One can observe that the basic translation produces

SLAA of similar size as ltl3ba: small differences are caused by some additional techniques implemented in ltl3ba [2] and by LTL rewriting rules implemented in Spot (and thus applied by ltl3tela during formula preprocessing) but not in ltl3ba. For example, the formula $\mathsf{FG}(a \wedge \mathsf{F}b)$ would be translated to a 4-state automaton by both ltl3ba and the basic translation, but Spot's formula simplification procedure rewrites the formula to $\mathsf{G}(\mathsf{F}b \wedge \mathsf{FG}a)$ and the basic translation subsequently produces an automaton with 5 states. Further, the F-merging, F,G-merging, and F,G-merging with simplifications bring gradual reduction of the automata size measured by the number of states. The ratio of states that can be saved compared to ltl3ba or the basic translation grows with the increasing occurrence of F and G operators up to 48% for the formulae of *rndfg*. The scatter plots in Figure 9 offer more data about the improvement (in the number of states) of F,G-merging with simplifications over the basic translation. The scatter plots reveal that F,G-merging with simplifications saves up to 3 states in most cases. However, there are also cases where the F,G-merging with simplifications reduces the resulting SLAA to 1 state only while the basic translation needs 8 or even more states. The trimming simplification described in Section 6.3 helped to get rid of the very rare cases where the basic translation produces smaller automaton than unoptimized F-merging and F,G-merging (one such a case can be seen in the ICTAC 2019 version of this paper [5]).

Table 1 also shows that the basic translation produces automata with more edges than ltl3ba. This is mainly due to the suspension technique [2] introduced in ltl3ba and not implemented in our translations, and different rules for the formula preprocessing. All improved translations produce less edges than ltl3ba. As expected, the lowest number of edges is produced by F,G-merging with simplifications, which saves about one third of edges compared to ltl3ba on the *lit* and *rndfg* sets.

The situation with the numbers of acceptance marks is completely different. The basic translation sometimes produces less acceptance marks than ltl3ba as it produces 0 acceptance marks for input formulae with no F or U operator. The improved translations produce SLAA with more acceptance marks than the basic translation or ltl3ba. This is the price for small automata, and it is natural to ask whether it pays off. We believe that there are situations when it is better to have an automaton with less states and more acceptance marks than the SLAA produced by ltl3ba or the basic translation. For example, this can be the case of applications that construct a product of the automaton with another, potentially very large system, like explicit model checking algorithms do. In such applications, every saved state of the automaton can substantially reduce the product. There are also algorithms exponential in the number of states and less sensitive to the number of acceptance marks, like algorithms transforming SLAA to nonalternating automata, see Major's thesis for more details [20]. The increase of acceptance marks can be probably suppressed by some acceptance formula optimization, which is a topic for our future research.

Translation improvements and advanced simplifications have a positive impact also on the type of the produced automata as they remove both some universal and nondeterministic branching from the automaton. Table 2 shows for each

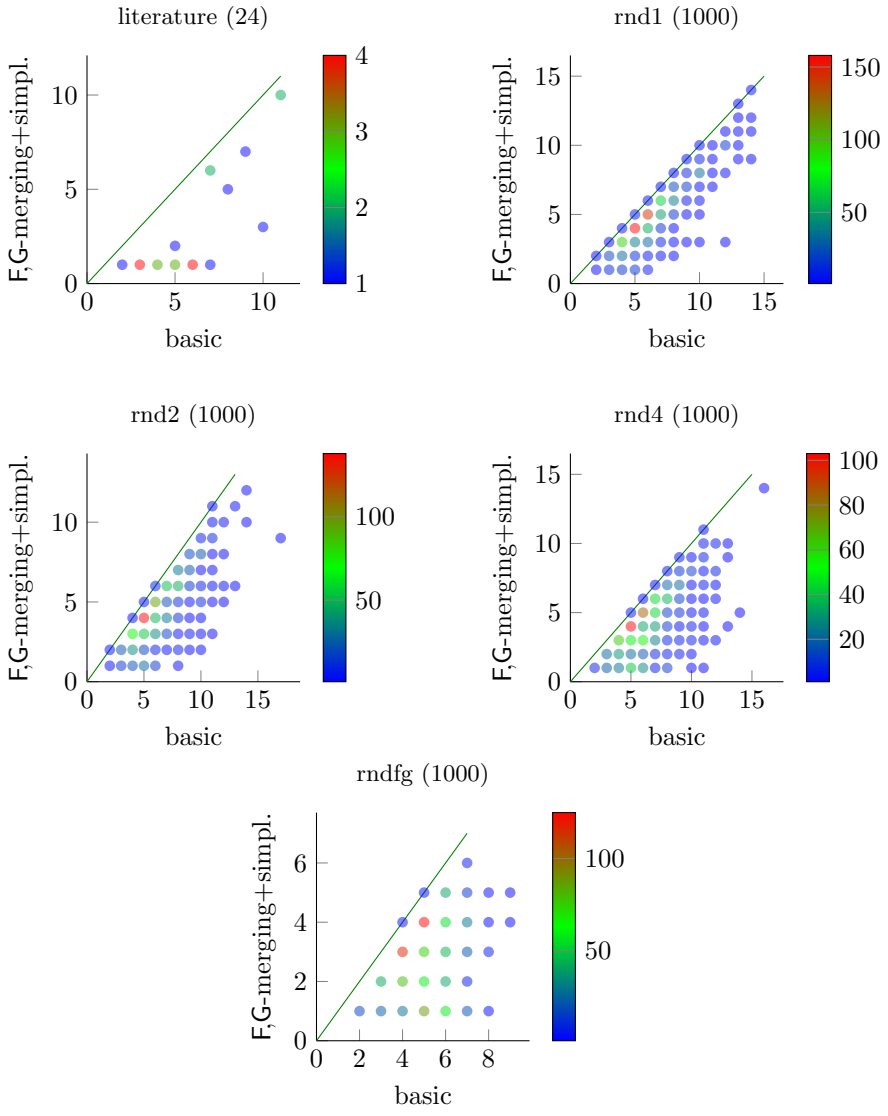

Figure 9: Effect of F,G-merging on SLAA size for mergeable formulae. A dot represents the number of states of the SLAA produced by F,G-merging with simplifications (y-axis) and by the basic translation (x-axis) for the same formula. The color of the dot reflects the number of dots at the position.

Table 2: Numbers of produced SLAA that are in fact deterministic or at least nonalternating.

		ltl3ba	basic	F-merg.	F,G-merg.	F,G+simpl.
determ.	lit (24)	0	0	1	9	10
	rnd1 (1000)	5	5	61	123	137
	rnd2 (1000)	2	2	49	111	130
	rnd4 (1000)	0	0	39	111	134
	rndfg (1000)	0	0	80	223	245
nonalt.	lit (24)	6	6	6	18	18
	rnd1 (1000)	148	148	171	358	362
	rnd2 (1000)	114	116	148	366	377
	rnd4 (1000)	89	97	127	391	403
	rndfg (1000)	144	161	208	645	645

set of formulae and each translation, the number of nonalternating automata (bottom part) and the number of automata that are even deterministic (top part).

8.4. Comparison of Nonalternating Automata

Now we evaluate the competitiveness of the nonalternating and deterministic automata produced by the F,G-merging translation with simplifications. Hence, we have selected the formulae for which the F,G-merging translation with simplifications produces a nonalternating or even deterministic automaton. We compare these automata with the results of state-of-the-art tools ltl3tela, Spot, Delag, and ltl2dgra translating LTL to deterministic automata, and with tools ltl3tela, ltl3ba, and Spot set to translate LTL to nonalternating automata. Note that ltl3tela now refers to the entire translation algorithm of the tool including the combination of different translation approaches, as explained in Appendix A. The precise settings of all these tools can be found in Tables B.6 and B.7 in Appendix B. Note that we apply the tool `autfilt` from Spot to further reduce the size of the automata produced by F,G-merging with simplifications as the same or similar optimizations are included in all other tools as well.

Tables 3 and 4 compare the results on the formulae for which the F,G-merging translation with simplifications produced deterministic or nonalternating automata, respectively. The results show that the full ltl3tela translation produces the smallest (on average) automata measured by the number of states. This is not surprising as it combines the power of the presented F,G-merging translation with simplifications, the translation of Spot, and other techniques and heuristics. In particular, it cannot produce any automaton with more states than F,G-merging with simplifications. However, Table 4 implies that the F,G-merging with simplifications can sometimes produce a nonalternating automaton with less edges that ltl3tela. This is due to the fact that ltl3tela prefers to produce an automaton with less acceptance marks to an automaton with less edges. We further observe that both deterministic and nonalternating automata

Table 3: Comparison of LTL to deterministic automata translators on the formulae where the F,G-merging translation with simplifications produces deterministic automata.

		F,G+simpl.+Spot	ltl3tela	Spot	Delag	ltl2dgra
states	lit (24)	10	10	10	10	11
	rnd1 (137)	353	353	384	366	356
	rnd2 (130)	266	266	288	269	270
	rnd4 (134)	241	241	263	247	249
	rndfg (245)	381	381	383	386	393
edges	lit (24)	68	67	67	72	70
	rnd1 (137)	661	655	753	2172	660
	rnd2 (130)	506	498	576	1295	509
	rnd4 (134)	507	490	572	1177	512
	rndfg (245)	858	828	883	1789	862
acc. marks	lit (24)	23	23	23	26	30
	rnd1 (137)	205	183	188	357	245
	rnd2 (130)	193	168	180	313	241
	rnd4 (134)	217	194	200	331	260
	rndfg (245)	384	351	369	606	480

produced by the F,G-merging with simplifications have on average less states and edges than automata produced by all other considered tools except ltl3tela. Moreover, the numbers of states of the deterministic automata produced by the F,G-merging with simplifications are equal to those produced by ltl3tela. It indicates that if the F,G-merging translation with simplifications produces a deterministic automaton, then the size of the automaton is competitive.

8.5. Comparison of LTL to SLAA Translations on Parametric Formulae

Finally, we compare the same LTL to SLAA translations as in Section 8.3, but this time on parametric formulae from literature [8, 13, 14, 23]. The mentioned papers contain 15 different mergeable parametric formulae. These formulae are also encoded in the tool `genltl`, which can generate their instances for given parameter values. We identify the parametric formulae with the corresponding `genltl` options.

We have generated instances of each parametric formula for all parameter values between 1 and 64. For each parametric formula and each of the five considered LTL to SLAA translations, we ran the translation on each formula instance and observed the maximal parameter value such that the corresponding instance is translated within 30 seconds on a computer with Intel® Core™ i7-8550U CPU and 16 GB of RAM. The results are presented in the topmost part of Figure 10 marked as 'max. n'. Roughly speaking, this measurement compares speed of the translations: the higher 'max. n', the faster the translation. However, there are also other aspects. In particular, the Spot library supports automata with up to 32 acceptance marks. Hence, the translations producing

Table 4: Comparison of LTL to nonalternating automata translators on the formulae where the F,G-merging translation with simplifications produces nonalternating automata.

		F,G+simpl.+Spot	ltl3tela	Spot	ltl3ba
states	lit (84)	21	19	19	51
	rnd1 (362)	1152	1075	1243	1534
	rnd2 (377)	1076	1006	1155	1547
	rnd4 (403)	990	904	1063	1615
	rndfg (645)	1347	1261	1349	2191
edges	lit (84)	108	136	136	154
	rnd1 (362)	2266	2237	2559	2982
	rnd2 (377)	2187	2148	2495	3232
	rnd4 (403)	2225	2150	2530	3447
	rndfg (645)	3092	3061	3260	4498
acc. marks	lit (84)	60	45	45	37
	rnd1 (362)	654	535	416	612
	rnd2 (377)	768	570	479	697
	rnd4 (403)	1021	711	612	819
	rndfg (645)	1501	1009	1006	1156

multiple acceptance marks (i.e., F-merging, F,G-merging, and F,G-merging with simplifications) fail to translate formulae if the corresponding SLAA should have more than 32 acceptance marks. Anyway, for each parametric formula except `gh-r`, `ms-phi-r`, and `ms-phi-s`, the translations can be ordered as

$$\text{ltl3ba} \succeq_n \text{basic} \succeq_n \text{F-merging} \succeq_n \text{F,G-merging} \succeq_n \text{F,G-merging+simpl.},$$

where $\succeq_n$ means that the translation on the left can handle at least as many instances as the translation on the right. For the three remaining formulae, the basic and F-merging translations are swapped in the ordering.

For each parametric formula, the horizontal line and the adjacent number in the topmost part of Figure 10 show the maximal parameter such that the corresponding formula instance was translated by all translations. Such formula instance is called *maximal common instance*. For these maximal common instances, Figure 10 indicates the numbers of states, edges, and acceptance marks produced by each translation.

We can observe that the translations are ordered as

$$\text{ltl3ba} \succeq_s \text{basic} \succeq_s \text{F-merging} \succeq_s \text{F,G-merging} \succeq_s \text{F,G-merging+simpl.},$$

where $\succeq_s$ means that for each maximal common instance, the translation on the left produces an automaton with at least as many states as the translation on the right. The decrease in the number of states usually comes with a higher number of acceptance marks. Further, the parametric formulae can be divided into three sets.

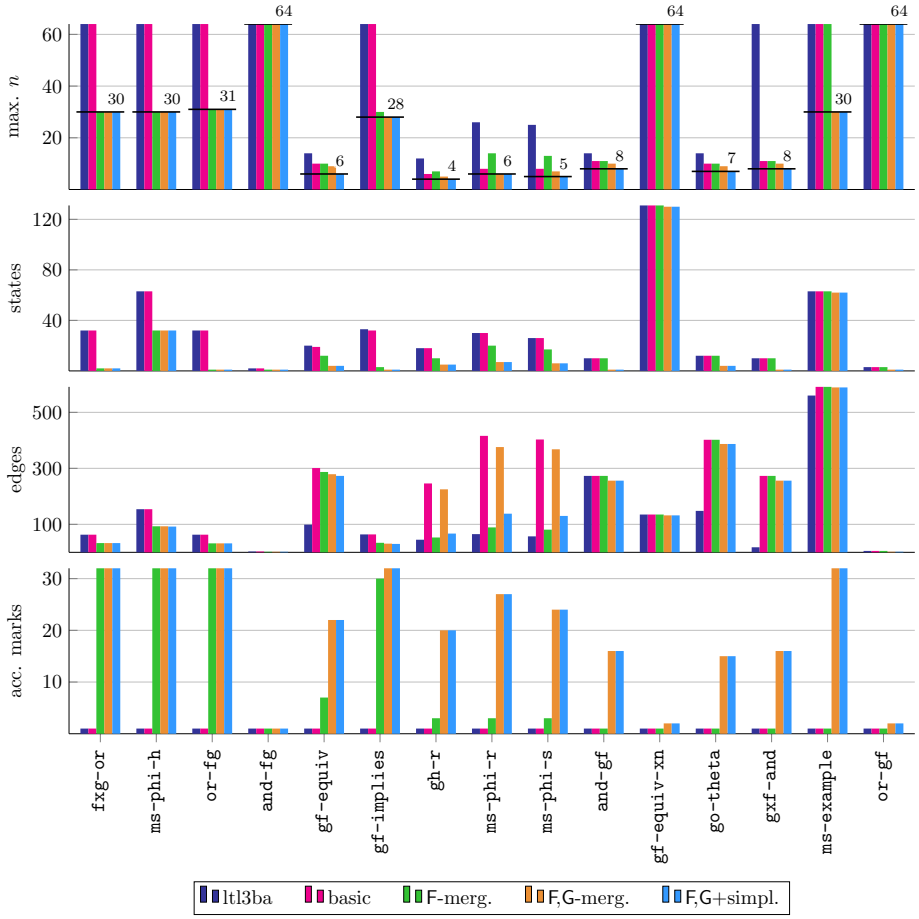

Figure 10: Comparison of LTL to SLAA translations on mergeable parametric formulae, which are on x-axis.

1. For `fgx-or`, `ms-phi-h`, `or-fg`, and `and-fg`, only F-merging reduces the automaton size (states and edges), F,G-merging and simplifications do not bring any further reductions.
2. For `gf-equiv`, `gf-implies`, `gh-r`, `ms-phi-r`, and `ms-phi-s`, both F-merging and F,G-merging gradually reduce the number of states. Simplifications do not reduce the number of states any further, but reduce the number of edges.
3. For `and-gf`, `gf-equiv-xn`, `go-theta`, `gxf-and`, `ms-example`, and `or-gf`, F-merging has no effect and only F,G-merging reduces the size of automata. In this group, simplifications have no measurable effect.

9. Conclusion

We have presented a novel translation of LTL to self-loop alternating automata (SLAA) with Emerson-Lei acceptance condition. To our best knowledge, it is the first translation of LTL producing SLAA with other than Büchi or co-Büchi acceptance. The translation is implemented in a tool ltl3tela. Our experimental results demonstrated that the expressive acceptance condition allows to produce substantially smaller SLAA comparing to these produced by ltl3ba when F or G operators appear in the translated formula.

This work opens doors for research of algorithms processing SLAA with Emerson-Lei acceptance, in particular the algorithms transforming these SLAA to nonalternating automata with various degrees of determinism: nondeterministic, deterministic, or semi-deterministic (also known as limit-deterministic) automata. Our implementation can serve as a natural source of such alternating automata and it already helped to build the tool ltl3tela that can produce small deterministic and nondeterministic TELA.

Acknowledgments

We would like to thank the two anonymous reviewers for their valuable feedback and suggestions.

F. Blahoudek has been supported by the DARPA grant D19AP00004 and the AFRL grant FA9550-19-1-0169. J. Major and J. Strejček have been supported by the Czech Science Foundation grant GA19-24397S.

References

[1] T. Babiak, F. Blahoudek, M. Křetínský, and J. Strejček. "Effective Translation of LTL to Deterministic Rabin Automata: Beyond the (F, G)-Fragment". In: *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*. Lecture Notes in Computer Science 8172, pp. 24–39. Springer, 2013.

[2] T. Babiak, M. Křetínský, V. Řehák, and J. Strejček. "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'12)*. Lecture Notes in Computer Science 7214, pp. 95–109. Springer, 2012.

[3] T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, D. Müller, D. Parker, and J. Strejček. "The Hanoi Omega-Automata Format". In: *Proceedings of the 27th International Conference on Computer Aided Verification (CAV'15)*. Lecture Notes in Computer Science 9206.I, pp. 479–486. Springer, 2015.

[4] C. Baier, F. Blahoudek, A. Duret-Lutz, J. Klein, D. Müller, and J. Strejček. "Generic Emptiness Check for Fun and Profit". In: *Proceedings of the 17th International Symposium on Automated Technology for Verification and Analysis (ATVA'19)*. Lecture Notes in Computer Science 11781, pp. 445–461. Springer, 2019.

[5] F. Blahoudek, J. Major, and J. Strejček. "LTL to Smaller Self-Loop Alternating Automata and Back". In: *Proceedings of the 16th International Colloquium on Theoretical Aspects of Computing (ICTAC'19)*. Lecture Notes in Computer Science 11884, pp. 152–171. Springer, 2019.

[6] K. Chatterjee, A. Gaiser, and J. Křetínský. "Automata with Generalized Rabin Pairs for Probabilistic Model Checking and LTL Synthesis". In: *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*. Lecture Notes in Computer Science 8044, pp. 559–575. Springer, 2013.

[7] J.-M. Couvreur, A. Duret-Lutz, and D. Poitrenaud. "On-the-Fly Emptiness Checks for Generalized Büchi Automata". In: *Proceedings of the 12th International Spin Workshop on Model Checking of Software (SPIN'05)*. Lecture Notes in Computer Science 3639, pp. 169–184. Springer, 2005.

[8] A. Duret-Lutz. "Manipulating LTL Formulas Using Spot 1.0". In: *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*. Lecture Notes in Computer Science 8172, pp. 442–445. Springer, 2013.

[9] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. "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'16)*. Lecture Notes in Computer Science 9938, pp. 122–129. 2016.

[10] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. "Property Specification Patterns for Finite-State Verification". In: *Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP'98)*, pp. 7–15. ACM, 1998.

[11] E. A. Emerson and C.-L. Lei. "Modalities for Model Checking: Branching Time Logic Strikes Back". In: *Science of Computer Programming* 8.3 (1987), pp. 275–306.

[12] K. Etessami and G. J. Holzmann. "Optimizing Büchi Automata". In: *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*. Lecture Notes in Computer Science 1877, pp. 153–167. Springer, 2000.

[13] P. Gastin and D. Oddoux. "Fast LTL to Büchi Automata Translation". In: *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*. Lecture Notes in Computer Science 2102, pp. 53–65. Springer, 2001.

[14] J. Geldenhuys and H. Hansen. "Larger Automata and Less Work for LTL Model Checking". In: *Proceedings of the 13th International SPIN Symposium on Model Checking of Software (SPIN'06)*. Lecture Notes in Computer Science 3925, pp. 53–70. Springer, 2006.

[15] M. Hammer, A. Knapp, and S. Merz. "Truly On-the-Fly LTL Model Checking". In: *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*. Lecture Notes in Computer Science 3440, pp. 191–205. Springer, 2005.

[16] J. Holeček, T. Kratochvíla, V. Řehák, D. Šafránek, and P. Šimeček. *Verification Results in Liberouter Project*. Tech. rep. 03, 32pp. CESNET, Sept. 2004.

[17] J. Křetínský, T. Meggendorfer, and S. Sickert. "Owl: A Library for ω-Words, Automata, and LTL". In: *Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA'18)*. Lecture Notes in Computer Science 11138, pp. 543–550. Springer, 2018.

[18] J. Křetínský, T. Meggendorfer, S. Sickert, and C. Ziegler. "Rabinizer 4: From LTL to Your Favourite Deterministic Automaton". In: *Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18)*. Lecture Notes in Computer Science 10981, pp. 567–577. Springer, 2018.

[19] C. Löding and W. Thomas. "Alternating Automata and Logics over Infinite Words". In: *Proceedings of International Conference Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics (IFIP TCS'00)*. Lecture Notes in Computer Science 1872, pp. 521–535. Springer, 2000.

[20] J. Major. "Translation of LTL into nondeterministic automata with generic acceptance condition". MA thesis. Masaryk University, Brno, 2017. URL: `https://is.muni.cz/th/o0rn5/dp.pdf`.

[21] J. Major, F. Blahoudek, J. Strejček, M. Sasaráková, and T. Zbončáková. "`ltl3tela`: LTL to Small Deterministic or Nondeterministic Emerson-Lei Automata". In: *Proceedings of the 17th International Symposium on Automated Technology for Verification and Analysis (ATVA'19)*. Lecture Notes in Computer Science 11781, pp. 357–365. Springer, 2019.

[22] D. E. Muller, A. Saoudi, and P. E. Schupp. "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 '88)*, pp. 422–427. IEEE Computer Society, 1988.

[23] D. Müller and S. Sickert. "LTL to Deterministic Emerson-Lei Automata". In: *Proceedings of the 8th International Symposium on Games, Automata, Logics and Formal Verification (GandALF'17)*. EPTCS 256, pp. 180–194. 2017.

[24] R. Pelánek. "BEEM: Benchmarks for Explicit Model Checkers". In: *Proceedings of the 14th international SPIN conference on Model checking software (SPIN'07)*. Lecture Notes in Computer Science 4595, pp. 263–267. Springer, 2007.

[25] R. Pelánek and J. Strejček. "Deeper Connections Between LTL and Alternating Automata". In: *Proceedings of the 10th International Conference on Implementation and Application of Automata (CIAA'05)*. Lecture Notes in Computer Science 3845, pp. 238–249. Springer, 2005.

[26] A. Pnueli. "The Temporal Logic of Programs". In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pp. 46–57. IEEE Computer Society, 1977.

[27] E. Renault, A. Duret-Lutz, F. Kordon, and D. Poitrenaud. "Parallel Explicit Model Checking for Generalized Büchi Automata". In: *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15)*. Lecture Notes in Computer Science 9035, pp. 613–627. Springer, 2015.

[28] G. S. Rohde. "Alternating Automata and the Temporal Logic of Ordinals". PhD thesis. University of Illinois at Urbana-Champaign, 1997. ISBN: 0-591-63604-2.

[29] F. Somenzi and R. Bloem. "Efficient Büchi Automata from LTL Formulae". In: *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*. Lecture Notes in Computer Science 1855, pp. 248–263. Springer, 2000.

[30] H. Tauriainen. "Automata and Linear Temporal Logic: Translations with Transition-Based Acceptance". PhD thesis. Helsinki University of Technology, Laboratory for Theoretical Computer Science, 2006. ISBN: 951-22-8343-3.

[31] M. Y. Vardi. "Nontraditional Applications of Automata Theory". In: *Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'94)*. Lecture Notes in Computer Science 789, pp. 575–597. Springer, 1994.

Appendix A. ltl3tela

Appendix A.1. LTL to SLAA Translation

The LTL to SLAA translations presented in Sections 3–5 can be executed using the command `ltl3tela -p1 [OPTIONS] -f` φ, where `-p1` says that the tool should produce an SLAA and φ is an input LTL formula in any syntax supported by Spot [8]. Options can modify the actual translation process in the following ways:

`-s[0|1]` controls formula preprocessing: `-s1` enables the formula simplification procedure of Spot, while `-s0` disables it.

`-i[0|1]` specifies the construction of initial states: `-i1` enforces that the constructed SLAA will have just one initial state (as described in our translations), while `-i0` can produce SLAA with multiple initial configurations of states as ltl2ba [13] does.

`-X[0|1]` determines the translation of X-subformulae: `-X1` produces automata where every state $\mathsf{X}\psi$ has one transition leading to the state ψ (as described in our translations), while `-X0` constructs automata where transitions from $\mathsf{X}\psi$ lead to configurations of maximal temporal subformulae of ψ as ltl2ba [13] does.

`-F[0|1]` turns on (`-F1`) and off (`-F0`) use of F-merging.

`-G[0|1]` controls the use of G-merging: it is enabled with `-G1` and disabled with `-G0`. Note that G-merging can be enabled even if F-merging is not.

`-d[0|1|2]` specifies the kind of applied transition dominance simplification: no such simplification is used with `-d0`, only the basic dominance is applied with `-d1`, and `-d2` turns on the acceptance-aware dominance.

`-c[0|1]` switches on (`-c1`) and off (`-c0`) the destination configuration trimming.

The default setting is `-s1 -i0 -X0 -F1 -G1 -d2 -c1`. When the SLAA is constructed, its unreachable states are removed and the automaton is sent to standard output in the HOA format [3].

Appendix A.2. LTL to TELA Translation

The tool ltl3tela actually offers much more than just LTL to SLAA translations: its main application is translation of LTL to small nondeterministic or deterministic TELA. Therefore it also implements an algorithm that translates SLAA to nondeterministic TELA [20]. To achieve the best results, ltl3tela combines this LTL to TELA translation via SLAA with other translations and techniques. Roughly speaking, the LTL formula is first decomposed to its maximal temporal subformulae. Each subformula is then translated by our LTL to TELA translation via SLAA and also by the translation of Spot, and the smaller automaton is selected. Then we use various product constructions to combine these automata into the automaton for the whole formula. The process is precisely described in our recent tool paper about ltl3tela [21]. The paper also present experimental comparison with state-of-the-art LTL to automata translators including ltl3ba, Spot, Delag, and ltl2dgra (formerly called Rabinizer 4) showing that ltl3tela currently produces the smallest deterministic and nondeterministic automata (on average).

Appendix B. Exact Settings of Translators in Experiments

Table B.5 provides the exact options used in the comparison of LTL to SLAA translations of Table 1. ltl3ba is run with the option `-H1` to enable the output of

the alternating automaton. The options `-i1 -X1` of ltl3tela were used to stick with the formal description of translations and because ltl3ba treats initial states and X-subformulae in the same way.

Table B.5: Overview of commands executed to measure the differences between various LTL to SLAA translation approaches.

translation	command
ltl3ba	`ltl3ba -H1`
basic	`ltl3tela -p1 -i1 -X1 -F0 -G0 -d1 -c0`
F-merging	`ltl3tela -p1 -i1 -X1 -G0 -d1 -c0`
F,G-merging	`ltl3tela -p1 -i1 -X1 -d1 -c0`
F,G-merg.+simpl.	`ltl3tela -p1 -i1 -X1`

Tables B.6 provides the exact options used for the comparison in Table 3 of the deterministic automata produced by the F,G-merging translation with simplifications and by state-of-the-art LTL to deterministic automata translators.

Table B.6: List of tools used to compare the translation of LTL to deterministic automata.

tool	command
F,G+simpl.+Spot	`ltl3tela -p1 | autfilt --high`
ltl3tela	`ltl3tela -D1`
Spot	`ltl2tgba -DG`
Delag	`delag`
ltl2dgra	`ltl2dgra`

Similarly, Table B.7 provides the exact options used for the comparison of nonalternating automata presented in Table 4. The use of `-H2` option for ltl3ba ensures that the output is a TGBA instead of an (often larger) Büchi automaton.

Table B.7: List of tools used to compare the translation of LTL to nonalternating automata.

tool	command
F,G+simpl.+Spot	`ltl3tela -p1 | autfilt --high`
ltl3tela	`ltl3tela`
ltl3ba	`ltl3ba -H2`
Spot	`ltl2tgba -G`