Translating LTL to Lasso Automata

Rüdiger Ehlers, Nils Hölzner, Tobias Kühnel
Clausthal University of Technology, Clausthal-Zellerfeld, Germany
{ruediger.ehlers, nils.hoelzner, tobias.marcel.kuehnel}@tu-clausthal.de

***Abstract*—We provide a novel approach for translating linear temporal logic (LTL) formulas to deterministic lasso automata, which accept those words of the form $u\$v$ (for some finite words u and v) for which uv^ω satisfies the LTL formula. Such lasso automata characterize ω-regular languages, can be minimized in polynomial time, and have found first applications, such as for the computation of *tight Büchi automata*.**

We experimentally compare the new translation approach to an existing indirect approach based on a construction by Calbrix, Nivat, and Podelski.

I. Introduction

In the automata-theoretic approach to verification and synthesis, specifications of systems under design are written in some form of temporal logic and translated to automata as a more technical representation of the specification. Depending on whether the application at hand is reactive synthesis, probabilistic verification, or classical verification of non-deterministic systems, different types of automata are employed whose properties match the ones needed by the respective verification or synthesis algorithm. At the same time, the chosen automata types should allow representing the specification of interest in a concise way. For instance, for reactive systems verification, non-deterministic Büchi automata are often employed, while for some quantitative verification and synthesis tasks, deterministic parity or Rabin automata are used. The determinism of the latter is important for synthesis and probabilistic verification, but determinism comes at an exponential size increase.

Among the available automaton types for languages over infinite words (as used in reactive systems verification and synthesis), so-called *lasso automata* [8], [7], [6] (also named $L_\$$-automata) are a somewhat special case. For some language L over infinite words, they accept finite words of the form $u\$v$ for which uv^ω is in L. Rather than accepting infinite words with infinite runs, they accept or reject representations of *ultimately periodic words*. Since an ω-regular language is uniquely determined by the set of such words in the language, lasso automata are suitable for describing such languages. While a few variations of lasso automata exist (such as non-saturated lasso automata [6] and multiple variations of families of DFAs [2]), representing ω-regular languages by finite word automata over lassos is currently mainly used in the context of language learning, with a few exceptions, such as computing so-called *tight Büchi automata* [11]. Such tight Büchi automata are interesting in model checking of reactive systems, because they guarantee that the shortest counter-example lasso in the product between the system and the specification built in classical model checking also contains the shortest counter-example in the system model alone. Hence, they support obtaining understandable counter-example traces in model checking.

Lasso automata have some useful properties. For instance, in the form of deterministic automata (over finite words), they can be minimized using classical Myhill/Nerode equivalence. Furthermore, unlike for deterministic parity automata [4], language union and intersection operations are easy to perform on them and lead to only a quadratic blow-up.

It can be argued that lasso automata currently being seldomly applied in verification and synthesis approaches is at least in part due to the lack of exploration of how to obtain them from logical specifications (such as in linear temporal logic, LTL). While Calbrix et al. [6] gave a construction for translating a non-deterministic Büchi automaton (NBA) with n states to lasso automata, it involves building non-deterministic automata over finite words (NFA) of size $2n^n$, which appears to be prohibitive, given that subsequent determinization is still needed, and obtaining a non-deterministic Büchi automaton from the LTL specification causes an additional exponential worst-case blow-up. In a closer analysis, Calbrix et al. showed that the subsequent determinization step does not cause an additional exponential blow-up. Yet, the overall complexity still appears to be prohibitive.

Hence, for fostering the application of lasso automata in practice, it makes sense to revisit the translation from logic to lasso automata, analyze the strengths and weaknesses of the existing translation approach and to examine if by a direct translation from LTL to lasso automata, the high cost of the existing approach can be avoided. This situation is analogous to the study of deterministic parity and Rabin automata, for which verification and synthesis tools employing them started to appear after optimized translation tools from LTL to such automata became widely available in the previous two decades [16], [17], [10].

In this paper, we give a novel translation procedure from LTL to lasso automata and provide a thorough experimental comparison to employing the construction by Calbrix, Nivat, and Podelski [6]. Our novel translation (still) has a doubly-exponential worst-case complexity

and performs well on the majority of current (non-random) LTL-to-automata translation benchmark formulas. The approach extends the ideas of a classical LTL-to-generalized-Büchi automaton translation procedure [13] to yield a non-deterministic lasso automaton of size only exponential in the length of the LTL formula. The approach comes in two variants, where in the first variant, the lasso language NFA is first built, then determinized, and finally minimized. In the second variant, active language learning [1], [19] is used in an iterative refinement process to build the minimized deterministic lasso automaton without an intermediate blow-up.

Our work also provides insights into the properties of lasso automata and their amenability for practical verification and synthesis approaches. As a side-result of our experimental evaluation, we give a comparison between the sizes of minimal lasso automata and the sizes of deterministic parity automata for a collection of specifications, which allows estimating the usefulness of lasso automata in future applications. We also provide the sizes of *recurrent families of DFAs* [2] for the respective specifications, which are commonly employed in language learning and to which lasso automata are easy to translate. They can be exponentially smaller than lasso automata [2], are easy to obtain from lasso language DFAs, and our experiments shed light on how much smaller they tend to be in practice for common LTL properties.

A. Further related work

Representing omega-regular languages by lasso languages appears in many variations in the literature. In some of them, for every ultimately periodic word $w = uv^\omega$ in some ω-language, only *normalized* words $u\$v$ need to be contained in the lasso language. These words have the property that there is no alternative representation of $w = (u')(v')^\omega$ with $|u'| + |v'| < |u| + |v|$ [6], [3]. All approaches compared in this paper compute lasso automata that include both the normalized and non-normalized lasso words.

II. Preliminaries

We consider words over the letters $\Sigma = 2^{\mathsf{AP}}$ for some set of atomic propositions AP over which formulas in linear temporal logic (LTL, [18]) are defined. An LTL formula can be brought into negation normal form (NNF), where the $\neg$ operator can only occur in front of atomic propositions. By using the fact that for all subformulas ψ, we have $\mathsf{G}\psi \equiv \mathbf{false}\,\mathcal{R}\,\psi$ and $\mathsf{F}\psi \equiv \mathbf{true}\,\mathcal{U}\,\psi$, we can ignore the G and F operators in all constructions to follow.

An automaton over finite words is a tuple $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ with the set of states S, the alphabet Σ, the transition relation $\delta : S \times \Sigma \to 2^S$, the initial state $s_0 \in S$, and the set of final states F. Such automata can either be non-deterministic (NFA) or deterministic (DFA).

Given an LTL formula φ over some set AP, we call the language $L_\$ = \{u\$v \mid uv^\omega \models \varphi\}$ over $\Sigma = 2^{\mathsf{AP}} \cup \{\$\}$ the *lasso language* of φ. In such a word uv^ω, we call u the lasso stem and v is the period of the word. An automaton for the lasso language of φ is called the lasso automaton of φ.

III. Computing Lasso Language DFAs

We present two variants of a new approach for computing a deterministic finite automaton for the lasso language of an LTL formula. Both start from a direct translation of an LTL formula to an NFA for the lasso language. In the first variant, the NFA is then determinized and subsequently minimized. In the second variant, active automaton learning is used to iteratively learn the DFA, and for the equivalence checking step in active learning, this DFA is checked for language inclusion against the NFA built on-the-fly.

We start by showing how to construct an NFA for the lasso language of a given LTL formula.

A. An NFA for the lasso language of an LTL formula

Let φ be an LTL formula in negation normal form. We provide a construction for translating φ to an NFA $\mathcal{A}$ for the lasso language of φ. Every state in the NFA is labeled by a collection of *obligations* that the remainder of a lasso word needs to fulfill in order for the overall word to be in the lasso language. An obligation is a subformula of the LTL formula. Whenever there are multiple possible obligation sets for which the satisfaction of all subformulas in one of them would suffice to prove that the overall word is in the lasso language, we employ non-determinism in the automaton to select one of them.

Our construction distinguishes the cases of transitions before a \$ character of a finite word has been seen and after the first (and only allowed) such character, hence the NFA has distinct sets of states for before and after a \$ character. The transitions in the first state set are defined like in the construction by Gerth et al. [13] for translating an LTL formula in NNF to a *generalized Büchi automaton*, except that we do not employ a generalized Büchi acceptance condition.

Let φ be an LTL formula in negation normal form over some set of propositions AP and Ψ be the set of subformulas of the LTL formula. We define the part of the NFA before taking a \$ transition as $\mathcal{A}^{pre} = (S^{pre}, \Sigma, \delta^{pre}, s_0, \emptyset)$ with $S^{pre} = 2^\Psi$, $\Sigma = 2^{\mathsf{AP}}$, $s_0 = \{\varphi\}$, and

$$\delta(s, x) = \bigcup_{\substack{c \in (\Psi \to \{0,1\}) \\ \forall \psi \in s : succ(\psi, x, c) \text{ is defined}}} \left\{ \bigcup_{\psi \in s} succ(\psi, x, c) \right\}.$$

In this context, the function *succ* takes an LTL formula ψ, a letter x, and a choice function for resolving non-determinism for each subformula. Given an LTL formula ψ, the function computes the subformulas that

a word $w_0 w_1 \ldots \in \Sigma^\omega$ needs to satisfy in order for the word $x w_0 w_1 w_2 \ldots$ to satisfy ψ. The choice function selects which subformulas are expected to hold on the rest of the word in case there are multiple possibilities (such as for disjunctions), and the union over all choice functions in the formula above ensures that all possible concrete ways to satisfy the LTL formula are considered. Concretely, we define, for subformulas ψ' and ψ'', letters x, propositions p and some $c \in \Psi \to \{0,1\}$:

$$
\begin{aligned}
&succ(\mathbf{false}, x, c) \text{ is undefined}\\
&succ(\mathbf{true}, x, c) = \emptyset\\
&succ(p, x, c) = \begin{cases} \emptyset & \text{if } p \in x \\ \text{undefined} & \text{otherwise} \end{cases}\\
&succ(\neg p, x, c) = \begin{cases} \emptyset & \text{if } p \notin x \\ \text{undefined} & \text{otherwise} \end{cases}\\
&succ(\psi' \wedge \psi'', x, c) = succ(\psi', x, c) \cup succ(\psi'', x, c)\\
&succ(\psi' \vee \psi'', x, c) = \begin{cases} succ(\psi', x, c) & \text{if } c(\psi' \vee \psi'') = 0 \\ succ(\psi'', x, c) & \text{if } c(\psi' \vee \psi'') = 1 \end{cases}\\
&succ(\mathsf{X}\psi', x, c) = \{\psi'\}\\
&succ(\psi' \boldsymbol{\mathcal{U}} \psi'', x, c) = \begin{cases} succ(\psi'', x, c) & \text{if } c(\psi' \boldsymbol{\mathcal{U}} \psi'') = 0 \\ succ(\psi', x, c) & \\ \quad \cup \{\psi' \boldsymbol{\mathcal{U}} \psi''\} & \text{if } c(\psi' \boldsymbol{\mathcal{U}} \psi'') = 1 \end{cases}\\
&succ(\psi' \mathcal{R} \psi'', x, c) = \begin{cases} succ(\psi', x, c) & \\ \cup\, succ(\psi'', x, c) & \text{if } c(\psi' \mathcal{R} \psi'') = 0 \\ succ(\psi'', x, c) & \\ \quad \cup \{\psi' \mathcal{R} \psi''\} & \text{if } c(\psi' \mathcal{R} \psi'') = 1 \end{cases}
\end{aligned}
$$

We employ the LTL formula $\varphi^e = y \boldsymbol{\mathcal{U}} (z \wedge \mathsf{X} z)$ as a running example henceforth. The top part of Figure 1 shows the NFA part $\mathcal{A}^{pre}$ built for this formula after stripping it from the unreachable states. The automaton part below the horizontal line is for the part of the NFA after a \$ character, which we discuss below. Three states are needed for $\mathcal{A}^{pre}$. A run can stay in the (top) left state while y holds, whereas the transition to the middle state is useful at the point in the word from which $z \wedge \mathsf{X}z$ holds. The obligations in the top middle state are $\{z\}$, as then z has to hold in the next step. The top right state is reachable for prefixes of words all of whose infinite extensions satisfy φ^e, and then there are no obligations to be fulfilled left.

The bottom part of Figure 1 contains the rest of the lasso language NFA for φ^e. It is reached after reading a \$ character, and we describe the translation for this part below. To simplify the construction and the running example slightly (by removing some intermediate states), we combine the \$ character with the next character in the word henceforth. In addition to the obligations, the states in this second part of the NFA are labeled by *promises* and *eventualities*, which we motivate next.

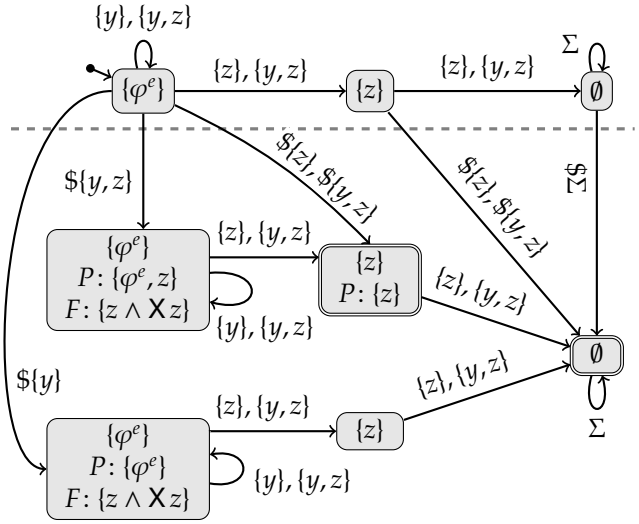

Fig. 1. Example NFA for the lasso language of φ^e

A *promise* (marked with a P: in Figure 1) is a subformula that is declared to hold at the beginning of the lasso cycle. Promises are used at the end of a lasso cycle to determine whether the state reached should be accepting or not. If all obligations remaining at the end of a cycle have been promised to hold from the beginning of the cycle, then when repeating the cycle infinitely often, the corresponding ultimately periodic word satisfies the LTL formula.

As an example, consider the lasso word $w = \$\{y,z\}\{y\}\{z\}$. Its corresponding infinite word $\{y,z\}\{y\}\{z\}\{y,z\}\ldots$ satisfies φ^e as y holds until eventually z holds for two times in a row, namely in the third and fourth letter of the infinite word. The cycle however only has three letters, so this can only be seen when considering multiple cycle copies. This is implemented in the NFA in Figure 1 by the possibility to take a $\$\{y,z\}$ transition from the initial state to the left state that is labeled with the promise z (which has just been satisfied with the first lasso cycle letter). When then reading a $\{y\}$ letter followed by a $\{z\}$ letter, the middle state is reached. It is accepting because while the z obligation is remaining, there is also a z promise, so when the cycle ends there, it closes an ultimately periodic word satisfying the LTL formula as the word part afterwards has been declared to satisfy z.

Promises stay part of the state label along a transition. As an optimization, a promise can be removed when the promise is not a subformula of some obligation any more. We applied this optimization in Figure 1 to avoid having multiple copies of the state with the empty obligation set differing by their promises.

Promises and obligations would together suffice for translating so-called *safety properties* to lasso automata. To support LTL formulas beyond safety properties, we employ *eventualities* as third labeling component (marked

with an F: in Figure 1). They keep track of subformulas that need a specific point in time at which they are satisfied and for which such a point has not been reached along a lasso cycle already. For instance, in the running example, the subformula $\varphi'^e = z \wedge \mathsf{X}z$ occurs on the right-hand side of an $\mathcal{U}$ operator. A word satisfying φ^e needs to satisfy φ'^e at some point along a run. In the NFA, it is resolved non-deterministically at which point φ'^e is expected to be satisfied, and then $succ(\phi'^e, x, c)$ is added to the obligations while the eventuality is removed. This happens along the transitions from the lower two left-most states to their respective right neighbors. States labeled with not-yet-fulfilled eventualities are non-accepting in the NFA.

With these ideas in mind, we can concretize the translation of some LTL formula φ to a lasso language NFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$. We start with a first automaton version $\mathcal{A}^{pre} = (S^{pre}, \Sigma^{pre}, \delta^{pre}, s_0, \emptyset)$ for φ as defined above. Then, let Ψ' be the subformulas of φ that occur as the operand of a right-hand-side operand of a $\mathcal{U}$ operator and for every $P \subseteq \Psi$ and choice function $c : \Psi \to \{0,1\}$, let $ev(P, c) = \{\beta \in \Psi' \mid \exists \alpha : \alpha \,\mathcal{U}\, \beta \in P \text{ and } c(\alpha \,\mathcal{U}\, \beta) = 1\}$. We define the components of $\mathcal{A}$ as follows (where $s, p \in 2^{\Psi}$, $f \in 2^{\Psi'}$, and $x \in \Sigma$):

$$
\begin{aligned}
S &= S^{pre} \cup 2^{\Psi} \times 2^{\Psi} \times 2^{\Psi'} \\
\Sigma &= \Sigma^{pre} \cup \{\$\} \cdot \Sigma^{pre} \\
\delta(s, x) &= \delta^{pre}(s, x) \\
\delta(s, \$x) &= \bigcup_{\substack{c \in (\Psi \to \{0,1\}), p \subseteq \Psi \\ \forall \psi \in s \cup p : succ(\psi, x, c) \text{ is defined}}} \Big\{\Big(\bigcup_{\psi \in s \cup p} succ(\psi, x, c), p, \\
&\qquad ev(s \cup p, c)\Big)\Big\} \\
\delta((s, p, f), x) &= \bigcup_{\substack{c \in (\Psi \to \{0,1\}),\ c' \in (f \to \{0,1\}) \\ \forall \psi \in s \cup \{\psi' \in f \mid c'(\psi') = 1\}: \\ succ(\psi, x, c) \text{ is defined}}} \Big\{\Big(\bigcup_{\psi \in s \cup \{\psi' \in f \mid c'(\psi') = 1\}} succ(\psi, \\
&\qquad x, c), p, \{\psi' \in f \mid c'(\psi') = 0\}\Big)\Big\} \\
F &= \{(s, p, \emptyset) \in S \mid s \subseteq p\}
\end{aligned}
$$

There are no transitions from states not in $\mathcal{A}^{pre}$ for characters starting with \$. As before, this construction defines an exponentially-sized state set for which we only have to consider the states that are reachable from the initial state. In the construction, states of the form (s, p, f) contain the obligation set s, the promises p, and the eventualities f. When transitioning from the states in S^{pre} to the rest of the states, the promises are chosen non-deterministically and for every promise ψ added, $succ(\psi, x, c)$ is added to the set of obligations to fulfill next because they encode what remains to be satisfied in order to fulfill the promise. When transitioning to the second part, those eventualities are added that need to be satisfied in order for the promises to hold and for which c encodes that they are not satisfied right away. Within the second part of the automaton, non-determinism is used to choose the points at which the remaining eventualities are satisfied (using the additional choice function c'). At such a point, the eventuality gets removed from the eventualities set and added as an obligation.

While the construction above describes the general idea, we employ some optimizations in order to avoid adding unnecessary states to the lasso language NFA:

- $\mathcal{A}$ is built in a breath-first-search manner so that unreachable states and transitions are never added to the NFA.
- Only subformulas that can appear at the end of a lasso cycle are used as promises. This excludes, for instance, conjunctions.
- Whenever before a \$ transition, some obligation is part of the state, the corresponding promise is always added to the successor state, as the promise does not cause additional obligations in this case.
- Whenever along a \$ transition, in the destination state, for some obligation, one of its subformulas is reachable in the obligation's formula parse tree through conjunction and F operators, and the subformula starts with a F operator, the subformula is always added as promise. This is because in such a case, the subformula needs to be fulfilled somewhere in the lasso period, and hence it can be added to the promise set without loss of generality.

B. Using the lasso language NFA in two translation approaches

We define two translation variants from LTL to deterministic lasso automata that both employ the NFA construction from the previous subsection.

Approach 1: The first variant bases on determinizing the NFA that is computed using the construction above. After splitting the transitions at \$ symbols, a classical power-set based determinization construction is used. We employ an *antichain*-based on-the-fly state space reduction [9], [12] to merge some equivalent states along the way. In particular, whenever for some NFA states (s, p, f) and (s', p', f') that are members of some DFA state label, we have $s \subseteq s'$, $p \supseteq p'$, and $f \subseteq f'$, then we know that the set of words accepted from state (s, p, f) is a superset of the words accepted from (s', p', f'), and hence (s', p', f') can be removed from the DFA state label.

The DFA is then subsequently minimized using Hopcroft's DFA minimization algorithm [14].

Approach 2: In the second variant, we employ active automaton learning [1]. The motivation for this approach is that with the first approach, the minimized DFA is frequently smaller than the NFA built according to the construction from the previous subsection. In active learning, we can obtain a minimally-sized DFA for the lasso language without building big deterministic intermediate automata. A learner repeatedly computes conjecture DFAs from example words in and not in the target language, and then the DFA is checked for

TABLE I
SOME EXAMPLE LTL FORMULAS AND RESULTS FOR THEIR TRANSLATIONS

ID	LTL Formula	Time (learning)	Time (non-learning)	Time (Calbrix et al.)	#States lasso DFA	Family of DFA size	#States det. parity aut.
00015	F(p1 ∧ F(p2 ∧ F(p3 ∧ F(p4 ∧ F p5)))) ∧ F(q1 ∧ F(q2 ∧ F(q3 ∧ F(q4 ∧ F q5))))	timeout	memout	memout	memout	memout	36
00020	F(p ∧ X(p ∧ X(p ∧ X(p ∧ X p)))) ∧ F(q ∧ X(q ∧ X(q ∧ X(q ∧ X q))))	49s391ms	339ms	1s723ms	814	73	36
00023	p0 U (p1 ∧ X(p2 ∧ F(p3 ∧ X F(p4 ∧ X F(p5 ∧ X F p6)))))	47s930ms	1m24s175ms	memout	108	19	9
00034	F(p ∧ X p ∧ X X p ∧ X X X p) ∧ F(q ∧ X q ∧ X X q ∧ X X X q)	17s871ms	432ms	286ms	364	51	25
00079	G((p0 ∧ F p1) → (((p2 ∧ X(¬p1 U p3)) → X(¬p1 U (p3 ∧ F p4))) U p1))	827ms	92ms	memout	38	78	20
00101	G((p0 → F p1) ∧ (p2 → F p3) ∧ (p4 → F p5) ∧ (p6 → F p7))	1m50s250ms	3s167ms	memout	99	642	33
00124	G((¬(p1 ↔ X p1) ∨ ¬(p0 ↔ X p0) ∨ ¬(p2 ↔ X p2) ∨ ¬(p3 ↔ X p3)) → (X¬p4 ∧ X(¬(¬(p1 ↔ X p1) ∨ ¬(p0 ↔ X p0) ∨ ¬(p2 ↔ X p2) ∨ ¬(p3 ↔ X p3)) U p4)))	timeout	1s349ms	2m13s934ms	1635	1124	65
00133	G(¬p0 ∨ X(¬p0 ∨ X(¬p0 ∨ X(¬p0 ∨ X(¬p0 ∨ X(¬p0 ∨ X(¬p0 ∨ X(¬p0 ∨ X(¬p0 ∨ X(¬p0 ∨ X¬p0))))))))))	8m18s170ms	207ms	42ms	224	182	13
00180	(p0 ∧ X p1) R X(((p2 U p3) R p0) U (p2 R p0))	23ms	31ms	17ms	17	17	6

correctness. Whenever it is not correct, new example words are reported to the learner.

The correctness check has two parts: we first check if the DFA is syntactically correct, i.e., it only accepts words with exactly one $ symbol. Afterwards, we label, in a breadth-first-search manner, every state of the conjecture DFA by which states of the NFA for the lasso language can be reached at the same time under some joint word. Once a combination of a rejecting DFA state with an accepting NFA state is found, this joint word is used as additionally example for the learner. To also detect when the DFA accepts too many words, we also label the DFA states with the NFA built according to the construction above for the complement of the LTL formula. In this way, whenever a combination of accepting NFA state and accepting DFA state is found, the word leading to this combination witnesses that the DFA accepts a word that is not in the target language, yielding another example word for the learning process. This approach builds the NFA on-the-fly and exploits that most incorrect DFAs often have simple reasons for why they do not encode the correct lasso language, so that example words can be found quickly during the learning process.

IV. EXPERIMENTS

We implemented both variants of the translation procedure described above (using a non-symbolic alphabet) in a tool in C++ . Additionally, we implemented the construction by Calbrix, Nivat, and Podelski [6] and applied it to automata built by *spot*'s [10] LTL-to-Büchi translator. Both implementations are available under an open-source license from https://github.com/TUC-ES/actuator.

For active automaton learning, we employ the extension of Rivest and Schapire [19] to Angluin's L* algorithm [1] as implemented in the `libalf` library [5].

We compare all implemented approaches on two benchmark sets: one with LTL specification patterns curated for *spot* [10] in a selection also used by Jankola and Strejček for benchmarking the computation of tight Büchi automata [15] (which motivates computing lasso automata), and one comprising random formulas. We also consider the sizes of families of DFAs for the LTL specification patterns. Maximum computation times are 1 hour while 4 GB of memory is available.

On the 207 LTL pattern formulas, the learning-based approach fails on four formulas while the non-learning-based variant fails on only one formula, namely formula 15 in Table I. In contrast, the method by Calbrix et al. fails on 8 formulas (such as formulas 15, 23, 79, and 101 in Table I) due to running out of memory, respectively. A closer look reveals that this is due to the NFAs built in the method by Calbrix et al. being extremely big. On formulas 23 and 79, they have 94692 and 1277 states at the time of running out of memory. On these formulas, the NFAs built by the new non-learning-based approach only have 108 and 38 states, respectively, which simplifies the determinization step. All formulas translatable with Calbrix et al.'s method can also be tackled by our new approach. The learning-based approach is faster than the non-learning-based approach in only two cases, namely for formulas 23 and 180 in Table I. In both cases, the speed-ups were moderate.

In 189 out of the 207 benchmarks, both the method by Calbrix et al. and the new non-learning based approach need less than one second of computation time. On

the 10 benchmarks of the remaining 18 ones for which neither timeout, the method by Calbrix et al. has higher computation times in 8 cases. In these 8 cases, the computation times of the method by Calbrix et al. is longer by a factor of at least 5, and Formula 124 in Table I shows such a case. Formulas 34 and 133 in the table show the two cases in which the method by Calbrix et al. was found to be faster. For both of them, `spot`'s `ltl2tgba` tool computed a deterministic Büchi automaton from the LTL specification, which we found to be beneficial for the performance of Calbrix et al.'s method.

We also observed that the sum of the number of states in the families of DFAs computed from the lasso automata is smaller than the corresponding lasso automata in 86 out of 206 cases, and bigger in 104 cases. Among the formulas for which the families of DFAs are bigger, they are bigger by 39.65% on average. When the lasso automata are bigger, they are bigger by 80.94% on average. Formulas 20 and 101 in Table I are the most extreme cases.

We also employed *spot* to compute (not necessarily minimal) deterministic parity automata for the formulas. On the 206 LTL patterns for which we were able to obtain a lasso automaton (so all formulas except for Formula 15 in Table I), deterministic parity automata are always smaller than the corresponding lasso automata. On average, the lasso automata are bigger by a factor of 4.3. Formula 124 in Table I shows the case with the largest relative difference.

As the benchmark set of 207 specification patterns curated for *spot* [10] in the selection also used by Jankola and Strejček [15] contains only a single LTL formula for which our new approach times out (in the non-learning-based variant), we also compare the computation times of the method by Calbrix et al. (including the time to translate LTL to Büchi automata) and the new approach on a set of randomly generated formulas. Figure 2 shows the results. On random formulas, the results are inconclusive and there is no clear winner.

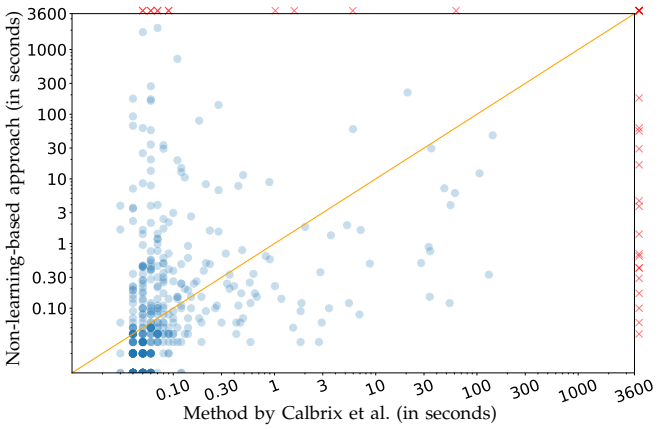

Fig. 2. Computation time comparison between Calbrix et al.'s method and the new approach (without learning) on random formulas. Timeouts/memouts are depicted as ×.

V. Conclusion

We presented a novel translation from LTL to lasso automata that comes in two variants. On an example LTL specification pattern benchmark set taken from the literature, we found the non-learning based variant to be substantially more scalable than the previous method by Calbrix et al. when applied to the output of an LTL-to-Büchi translator. On random LTL formulas, the results are inconclusive, even though the new approach timed out in fewer cases.

We note that from the 207 specification patterns considered, only a single case could not be translated by our new approach with a time limit of one hour. This specification is the conjunction of two equally long LTL subformulas that can be translated separately to lasso language DFAs of 40 states each (in less than one second). Subsequently taking the conjunction on the lasso automata level would then yield the lasso automaton for the conjunction of the subformulas.

This observation highlights why lasso automata (and families of DFAs derived from the lasso automata) are of potential interest for future practical applications: they allow building specification automata in a compositional way while making use of exact DFA minimization of the intermediate automata. We provide our translation approach as an open-source tool to enable such future applications of lasso automata.

References

[1] Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6

[2] Angluin, D., Fisman, D.: Learning regular omega languages. Theor. Comput. Sci. **650**, 57–72 (2016). https://doi.org/10.1016/J.TCS.2016.07.031

[3] Bohn, L., Li, Y., Löding, C., Schewe, S.: Saturation problems for families of automata. In: 52nd International Colloquium on Automata, Languages, and Programming (ICALP). pp. 146:1–146:19 (2025). https://doi.org/10.4230/LIPICS.ICALP.2025.146

[4] Boker, U.: Why these automata types? In: Barthe, G., Sutcliffe, G., Veanes, M. (eds.) 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-22). EPiC Series in Computing, vol. 57, pp. 143–163. EasyChair (2018). https://doi.org/10.29007/C3BJ

[5] Bollig, B., Katoen, J., Kern, C., Leucker, M., Neider, D., Piegdon, D.R.: libalf: The automata learning framework. In: 22nd International Conference on Computer Aided Verification (CAV). pp. 360–364 (2010). https://doi.org/10.1007/978-3-642-14295-6_32

[6] Calbrix, H., Nivat, M., Podelski, A.: Ultimately periodic words of rational ω-languages. In: 9th International Conference on Mathematical Foundations of Programming Semantics. pp. 554–566 (1993). https://doi.org/10.1007/3-540-58027-1_27

[7] Chernev, A., Hansen, H.H., Kupke, C.: Dual adjunction between ω-automata and Wilke algebra quotients. In: 21st International Colloquium on Theoretical Aspects of Computing (ICTAC). p. 96–113. Springer-Verlag, Berlin, Heidelberg (2024)

[8] Cruchten, M.: Kleene theorems for lasso languages and ω-languages. Theory of Computing Systems (2026)

[9] Doyen, L., Raskin, J.: Antichain algorithms for finite automata. In: 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 2–22 (2010). https://doi.org/10.1007/978-3-642-12002-2_2
[10] Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From spot 2.0 to spot 2.10: What's new? In: 34th International Conference on Computer Aided Verification (CAV), Proceedings, Part II. pp. 174–187 (2022). https://doi.org/10.1007/978-3-031-13188-2_9
[11] Ehlers, R.: How hard is finding shortest counter-example lassos in model checking? In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) Third World Congress on Formal Methods (FM). pp. 245–261. Lecture Notes in Computer Science, Springer (2019). https://doi.org/10.1007/978-3-030-30942-8_16
[12] Ehlers, R., Adabala, K.: Reactive synthesis of graphical user interface glue code. In: 17th International Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 387–403 (2019). https://doi.org/10.1007/978-3-030-31784-3_23
[13] Gerth, R., Peled, D.A., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: 15th IFIP WG6.1 International Symposium on Protocol Specification Testing and Verification. pp. 3–18 (1995)
[14] Hopcroft, J.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Kohavi, Z., Paz, A. (eds.) Theory of Machines and Computations, pp. 189–196. Academic Press (1971). https://doi.org/10.1016/B978-0-12-417750-5.50022-1
[15] Jankola, M., Strejček, J.: Tighter construction of tight Büchi automata. In: 27th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), Proceedings, Part I. pp. 234–255 (2024). https://doi.org/10.1007/978-3-031-57228-9_12
[16] Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theor. Comput. Sci. **363**(2), 182–195 (2006). https://doi.org/10.1016/J.TCS.2006.07.022
[17] Kretínský, J., Meggendorfer, T., Sickert, S.: Owl: A library for ω-words, automata, and LTL. In: Lahiri, S.K., Wang, C. (eds.) 16th International Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 543–550. Lecture Notes in Computer Science, Springer (2018). https://doi.org/10.1007/978-3-030-01090-4_34
[18] Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (FOCS). pp. 46–57 (1977). https://doi.org/10.1109/SFCS.1977.32
[19] Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Inf. Comput. **103**(2), 299–347 (1993). https://doi.org/10.1006/INCO.1993.1021

Appendix

A. On Families of DFAs

In the main part of the paper, we compare the sizes of lasso automata to the sizes of recurrent families of DFAs [2]. These are uniquely defined for every ω-regular language and there are languages over ω-regular words known for which lasso automata may need to be exponentially bigger than the family of DFAs [2]. Also, on a benchmark set considered in the same paper, Angluin and Fisman found that for ω-regular languages stemming from randomly generated Muller automata, recurrent families of DFAs tend to be smaller than *periodic families of DFAs*. Such periodic families of DFAs resemble the shape of lasso automata, but still split up the lasso language into separate DFAs for the lasso stem and the lasso cycle.

We included a comparison to recurrent families of DFAs in this paper because with the exponential size gap, they appear to be a promising language representation for verification and synthesis approaches starting from a polynomial-time minimizable language representation. At the same time, it is unclear if in practice, smaller automaton sizes will be observed for typical LTL specification patterns, which also motivated their inclusion in the paper.

The comparison is based on a translation of lasso language DFAs to recurrent families of DFAs, which our *actuator* tool provides as an additional functionality.

This translation is comparably simple. Recurrent families of DFAs consist of a *leading DFA* and one *progress DFA* for each state of the leading DFA. The minimized lasso automaton already has the leading DFA of the Family of DFAs as the part of the lasso automaton reachable without using a \$ transition. Then, for each state of the leading DFA, we take a product between the leading DFA and the lasso language DFA, using the combination of the leading automaton state and the \$-successor reachable in the lasso language DFA from it as initial state of the product. Exactly those states in the product are accepting for which the leading DFA component is the one for which the product was built and for which the lasso language DFA is in an accepting state. The product automata are then subsequently minimized to obtain the minimal progress DFAs. The correctness of this construction is a direct consequence of the definition of recurrent families of DFAs [2].

B. More details on the experiments

With Table 2, starting from the page after page 9, we provide the complete experimental results on the LTL specification pattern benchmark set. Apart from comparing automaton sizes and computation times, the table shows the sizes of the lasso languages NFAs for the non-learning-based approach. The table first lists the interesting cases, i.e., timeout or memout cases and cases where not all methods finish within 100 ms.

C. Proof of correctness of the main construction in the paper

We establish the formal correctness of the translation from an LTL formula φ in negation normal form (NNF) to the lasso language NFA $\mathcal{A} = (S, \Sigma, \delta, s_0, F)$ defined in Section II and Section III-A. Specifically, we prove that $\mathcal{A}$ is both sound ($L(\mathcal{A}) \subseteq L_{\$}(\varphi)$) and complete ($L_{\$}(\varphi) \subseteq L(\mathcal{A})$). The following proof was synthesized by a large language model (LLM) and manually checked.

Theorem 1 (Soundness)**.** *Let φ be an LTL formula in NNF and let $\mathcal{A}$ be the lasso NFA constructed for φ. For every lasso word $w = u(\$v_0)v_1 \dots v_{n-1}$ with stem $u = u_0 \dots u_{m-1} \in \Sigma^*$ ($m \geq 0$) and period $v = v_0 \dots v_{n-1} \in \Sigma^+$ ($n \geq 1$), if $w \in L(\mathcal{A})$, then $uv^\omega \models \varphi$.*

Proof. Let $w^\omega = uv^\omega \in \Sigma^\omega$ be the infinite periodic word, with positions $t \in \mathbb{N}$ given by $w^\omega_t = u_t$ for $0 \leq t < m$ and

$w_t^\omega = v_{(t-m) \bmod n}$ for $t \geq m$. Since $w \in L(\mathcal{A})$, there exists an accepting run of $\mathcal{A}$ on w:

- A sequence of pre-\$ states $s_0, s_1, \ldots, s_m \in S^{pre}$ with $s_0 = \{\varphi\}$, and choice functions $c_0, \ldots, c_{m-1}$ such that $s_{t+1} = \bigcup_{\psi \in s_t} succ(\psi, u_t, c_t)$ for $0 \leq t < m$;
- A loop-entry transition on $\$v_0$ choosing promise set $p \subseteq \Psi$ and choice function c_m reaching state $(s'_1, p, f_1) \in \delta(s_m, \$v_0)$, where $s'_1 = \bigcup_{\psi \in s_m \cup p} succ(\psi, v_0, c_m)$ and $f_1 = ev(s_m \cup p, c_m)$, with $succ(\psi, v_0, c_m)$ defined for all $\psi \in s_m \cup p$;
- Cycle states: for $k = 1, \ldots, n-1$, states $(s'_{k+1}, p, f_{k+1}) \in \delta((s'_k, p, f_k), v_k)$ with choice functions c_{m+k} and $c'_{m+k} : f_k \to \{0,1\}$ such that $s'_{k+1} = \bigcup_{\psi \in s'_k \cup \{\psi' \in f_k \mid c'_{m+k}(\psi') = 1\}} succ(\psi, v_k, c_{m+k})$ and $f_{k+1} = \{\psi' \in f_k \mid c'_{m+k}(\psi') = 0\}$;
- Acceptance: $(s'_n, p, f_n) \in F$, which requires $s'_n \subseteq p$ and $f_n = \emptyset$.

We extend the run to an infinite sequence of obligation sets $(O_t)_{t \geq 0}$ and choice functions $(C_t)_{t \geq 0}$ over w^ω:

- For $0 \leq t < m$: $O_t = s_t$ and $C_t = c_t$;
- For $t = m$: $O_m = s_m \cup p$ and $C_m = c_m$;
- For $t > m$, write $t = m + q \cdot n + k$ with $q \geq 0$ and $0 \leq k < n$: if $k = 0$ ($q \geq 1$), set $O_t = p$ and $C_t = c_m$; if $1 \leq k < n$, set $O_t = s'_k$ and $C_t = c_{m+k}$.

We next prove a couple of lemmata that will then be used to complete the proof of Theorem 1.

Lemma 2 (Obligation Propagation)**.** *For all $t \geq 0$ and all $\psi \in O_t$, $succ(\psi, w_t^\omega, C_t)$ is defined and $succ(\psi, w_t^\omega, C_t) \subseteq O_{t+1}$.*

Proof. For $t < m-1$, $O_{t+1} = s_{t+1} = \bigcup_{\psi \in s_t} succ(\psi, u_t, c_t)$. For $t = m-1$, $succ(s_{m-1}, u_{m-1}, c_{m-1}) = s_m \subseteq s_m \cup p = O_m$ (if $m = 0$, this step is vacuous). For $t = m$, $succ(s_m \cup p, v_0, c_m) = s'_1$; if $n = 1$, $s'_1 = s'_n \subseteq p = O_{m+1}$ by acceptance; if $n > 1$, $s'_1 = O_{m+1}$. For $t = m + qn + k$ with $1 \leq k < n$, $s'_{k+1} \supseteq \bigcup_{\psi \in s'_k} succ(\psi, v_k, c_{m+k})$. If $k = n-1$, $O_{t+1} = p \supseteq s'_n$. For $t = m + qn$ ($q \geq 1$), $O_t = p \subseteq s_m \cup p$, so definedness is inherited from loop entry and $succ(p, v_0, c_m) \subseteq s'_1 \subseteq O_{t+1}$. □

Lemma 3 (One-Step Semantic Connection of *succ*)**.** *Let $t \in \mathbb{N}$ and $\psi \in \Psi$. If $succ(\psi, w_t^\omega, C_t)$ is defined and $w^\omega, t+1 \models \theta$ for all $\theta \in succ(\psi, w_t^\omega, C_t)$, then: (i) for $p \in \mathsf{AP}$, $w^\omega, t \models p$; (ii) for $\neg p$, $w^\omega, t \models \neg p$; (iii) for $\psi_1 \wedge \psi_2$ or $\psi_1 \vee \psi_2$, $w^\omega, t \models \psi$; (iv) for $\mathsf{X}\psi_1$, $w^\omega, t \models \mathsf{X}\psi_1$; (v) for $\alpha \mathcal{U} \beta$ with $C_t(\psi) = 0$, $w^\omega, t \models \beta$; with $C_t(\psi) = 1$, $w^\omega, t \models \alpha$, and if $w^\omega, t+1 \models \alpha \mathcal{U} \beta$, then $w^\omega, t \models \alpha \mathcal{U} \beta$; (vi) for $\alpha \mathcal{R} \beta$ with $C_t(\psi) = 0$, $w^\omega, t \models \alpha \wedge \beta$; with $C_t(\psi) = 1$, $w^\omega, t \models \beta$, and if $w^\omega, t+1 \models \alpha \mathcal{R} \beta$, then $w^\omega, t \models \alpha \mathcal{R} \beta$.*

Proof. Follows directly by inspecting each case in the definition of *succ* against the standard inductive semantics of LTL. □

Lemma 4 (Semantic Satisfaction)**.** *For all $t \geq 0$ and all $\psi \in O_t$, $w^\omega, t \models \psi$.*

Proof. By well-founded structural induction on formula complexity $|\psi|$. The base cases (**true**, $p, \neg p$) and connectives ($\wedge, \vee, \mathsf{X}$) follow immediately from the induction hypothesis on strictly smaller subformulas via Lemma 3. For release formulas $\psi = \alpha \mathcal{R} \beta \in O_{t_0}$: if there exists a smallest $t_1 \geq t_0$ with $C_{t_1}(\psi) = 0$, then $w^\omega, j \models \beta$ for all $j \in [t_0, t_1]$ and $w^\omega, t_1 \models \alpha$, so $w^\omega, t_0 \models \psi$; if $C_t(\psi) = 1$ for all $t \geq t_0$, then $w^\omega, t \models \beta$ for all $t \geq t_0$, which also satisfies $\alpha \mathcal{R} \beta$.

For until formulas $\psi = \alpha \mathcal{U} \beta \in O_{t_0}$, suppose for contradiction that $w^\omega, t \not\models \beta$ for all $t \geq t_0$. Then $C_t(\psi) = 1$ for all $t \geq t_0$ (as $C_t(\psi) = 0$ requires $w^\omega, t \models \beta$). By Lemma 2, $\psi \in O_t$ and $w^\omega, t \models \alpha$ for all $t \geq t_0$. By periodicity of w^ω, $w^\omega, m+k \not\models \beta$ for all $0 \leq k < n$. Pick $q \geq 1$ such that $T = m + qn > t_0$. Then $O_T = p$, so $\psi \in p \subseteq s_m \cup p$. At loop entry on $\$v_0$, $c_m(\psi)$ must be 1 (as $c_m(\psi) = 0$ implies $w^\omega, m \models \beta$), so $\beta \in ev(s_m \cup p, c_m) = f_1$. If $n = 1$, $f_n = f_1 \ni \beta$, contradicting $f_n = \emptyset$. If $n > 1$, discharging β at any cycle step $k \in \{1, \ldots, n-1\}$ by $c'_{m+k}(\beta) = 1$ would require $succ(\beta, v_k, c_{m+k})$ to be defined, which by induction hypothesis implies $w^\omega, m+k \models \beta$, which is impossible. Hence $c'_{m+k}(\beta) = 0$ for all k, so β remains in f throughout the cycle, yielding $\beta \in f_n \neq \emptyset$, which contradicts $f_n = \emptyset$. Thus there exists a minimal $k^* \geq t_0$ with $w^\omega, k^* \models \beta$, and α holds on $[t_0, k^* - 1]$, proving $w^\omega, t_0 \models \alpha \mathcal{U} \beta$. □

Completing the proof of Theorem 1: Since $\varphi \in s_0 = O_0$, Lemma 4 implies $w^\omega, 0 \models \varphi$, i.e., $uv^\omega \models \varphi$. □

Theorem 5 (Completeness)**.** *Let φ be an LTL formula in NNF and let $\mathcal{A}$ be the lasso NFA constructed for φ. For every lasso word $w = u(\$v_0)v_1 \ldots v_{n-1}$ with stem $u \in \Sigma^*$ ($|u| = m \geq 0$) and period $v \in \Sigma^+$ ($|v| = n \geq 1$), if $uv^\omega \models \varphi$, then $w \in L(\mathcal{A})$.*

Proof. Let $w^\omega = uv^\omega \in \Sigma^\omega$. For each $t \geq 0$, define the semantic satisfaction set $\Phi_t = \{\psi \in \Psi \mid w^\omega, t \models \psi\}$. Since w^ω is n-periodic for $t \geq m$, $\Phi_{m+qn+k} = \Phi_{m+k}$ for all $q \geq 0$ and $0 \leq k < n$. Define canonical choice functions $c_t : \Psi \to \{0,1\}$ for each $t \geq 0$ by: $c_t(\psi_1 \vee \psi_2) = 0$ if $w^\omega, t \models \psi_1$, else 1; $c_t(\alpha \mathcal{U} \beta) = 0$ if $w^\omega, t \models \beta$, else 1; $c_t(\alpha \mathcal{R} \beta) = 0$ if $w^\omega, t \models \alpha$, else 1; and 0 otherwise. The functions c_t are n-periodic on the cycle: $c_{m+qn+k} = c_{m+k}$.

Lemma 6 (Semantic Consistency)**.** *For all $t \geq 0$ and $\psi \in \Phi_t$, $succ(\psi, w_t^\omega, c_t)$ is defined and $succ(\psi, w_t^\omega, c_t) \subseteq \Phi_{t+1}$.*

Proof. Straightforward case verification on formula structure using the definition of c_t and LTL semantics. □

We construct an accepting run of $\mathcal{A}$ on w:

- Pre-\$ run: Set $s_0 = \{\varphi\} \subseteq \Phi_0$. For $0 \leq t < m$, set $s_{t+1} = \bigcup_{\psi \in s_t} succ(\psi, u_t, c_t)$. By Lemma 6, $s_t \subseteq \Phi_t$ for all $t \leq m$, so $s_m \subseteq \Phi_m$.
- Loop entry: Choose promise set $p = \Phi_m \subseteq \Psi$. Then $s_m \subseteq p$, so $s_m \cup p = \Phi_m$. At $\$v_0$ with choice c_m, $s'_1 = \bigcup_{\psi \in \Phi_m} succ(\psi, v_0, c_m) \subseteq \Phi_{m+1}$, and $f_1 = ev(\Phi_m, c_m) = \{\beta \in \Psi' \mid \exists \alpha : \alpha \mathcal{U} \beta \in \Phi_m \wedge c_m(\alpha \mathcal{U} \beta) = 1\}$.

If $n = 1$, $w_t^\omega = (v_0)^\omega = w_m^\omega$ for all $t \geq m$, so every $\alpha \,\mathcal{U}\, \beta \in \Phi_m$ already has $w^\omega, m \models \beta$, forcing $c_m(\alpha \,\mathcal{U}\, \beta) = 0$. Hence $f_1 = \emptyset = f_n$, and $s'_1 = s'_n \subseteq \Phi_{m+1} = \Phi_m = p$.

If $n > 1$, for each $\beta \in f_1$, we have $w^\omega, m \not\models \beta$. Since $\alpha \,\mathcal{U}\, \beta \in \Phi_m$, there exists a smallest $j^* > m$ such that $w^\omega, j^* \models \beta$. By periodicity, $j^* < m + n$ (if $j^* \geq m + n$, the position $r = m + ((j^* - m) \bmod n) < j^*$ has $\Phi_r = \Phi_{j^*}$, contradicting the minimality of j^*). Thus $j^* = m + k^*$ for a unique $k^* \in \{1, \ldots, n-1\}$. For each cycle step $k \in \{1, \ldots, n-1\}$ and $\beta \in f_k$, set $c'_{m+k}(\beta) = 1$ if $k = k^*$ and 0 otherwise. When $c'_{m+k}(\beta) = 1$, $\beta \in \Phi_{m+k}$, so $succ(\beta, v_k, c_{m+k}) \subseteq \Phi_{m+k+1}$ is defined, and β is removed from f. Thus $f_n = \emptyset$. Induction gives $s'_{k+1} \subseteq \Phi_{m+k+1}$, so $s'_n \subseteq \Phi_{m+n} = \Phi_m = p$.

In both cases, $(s'_n, p, f_n) \in F$. Hence we have $w \in L(\mathcal{A})$. □

Table 2: Full results for the complete benchmark set considered in the paper. Potentially more interesting benchmarks with longer computation times are given first.

ID	LTL Formula	Time (learning)	Time (non-learning)	Time (Calbrix et al.)	#States lasso NFA	#States lasso DFA	Family of DFA Size	#States det. parity aut.
00005	F p1 & F p2 & F p3 & F p4 & F p5	1s559ms	19ms	186ms	66	65	65	32
00009	G F p1 & G F p2 & G F p3 & G F p4	172ms	10ms	47ms	18	98	17	4
00010	G F p1 & G F p2 & G F p3 & G F p4 & G F p5	2s884ms	90ms	327ms	34	276	33	5
00013	F(p1 & F(p2 & F p3)) & F(q1 & F(q2 & F q3))	7s643ms	535ms	285ms	82	186	33	16
00014	F(p1 & F(p2 & F(p3 & F p4))) & F(q1 & F(q2 & F(q3 & F q4)))	timeout or memout	4m55s923ms	18s206ms	283	1182	51	25
00015	F(p1 & F(p2 & F(p3 & F(p4 & F p5)))) & F(q1 & F(q2 & F(q3 & F(q4 & F q5))))	timeout or memout	timeout or memout	timeout or memout	timeout or memout	timeout or memout	timeout or memout	36
00018	F(p & X(p & X p)) & F(q & X(q & X q))	324ms	10ms	55ms	137	122	33	16
00019	F(p & X(p & X(p & X p))) & F(q & X(q & X(q & X q)))	2s797ms	58ms	287ms	364	558	51	25
00020	F(p & X(p & X(p & X(p & X p)))) & F(q & X(q & X(q & X(q & X q))))	49s391ms	339ms	1s723ms	814	2901	73	36
00023	p0 U (p1 & X(p2 & F(p3 & X F(p4 & X F(p5 & X F p6)))))	47s930ms	1m24s175ms	timeout or memout	108	1191	19	9
00030	G F p0 & G F p1 & G F p2 & G F p3 & G F p4	2s899ms	89ms	320ms	34	276	33	5
00033	F(p & X p & X X p) & F(q & X q & X X q)	441ms	21ms	58ms	137	177	33	16
00034	F(p & X p & X X p & X X X p) & F(q & X q & X X q & X X X q)	17s871ms	432ms	286ms	364	3667	51	25
00035	F(p & X p & X X p & X X X p & X X X X p) & F(q & X q & X X q & X X X q & X X X X q)	timeout or memout	19m21s206ms	1s723ms	814	305843	73	36
00048	F p0 → (∼p0 U (p0 & (∼p1 W (p1 W (∼p1 W (p1 W G∼p1))))))	237ms	11ms	12ms	15	237	26	7
00049	G((p0 & F p1) → ((∼p1 & ∼p2) U (p1 | ((∼p1 & p2) U (p1 | ((∼p1 & ∼p2) U (p1 | ((∼p1 & p2) U (p1 | (∼p2 U p1))))))))))	606ms	11ms	26ms	90	70	60	8
00050	G(p0 → ((∼p1 & ∼p2) U (p2 | ((p1 & ∼p2) U (p2 | ((∼p1 & ∼p2) U (p2 | ((p1 & ∼p2) U (p2 | (∼p1 W p2) | G p1)))))))))	563ms	11ms	27ms	72	66	56	7
00065	G((p0 & ∼p1) → ((p2 → (∼p1 U (∼p1 & p3))) W p1))	170ms	10ms	16ms	23	101	20	4
00069	G((p0 & F p1) → (∼p2 U (p1 | (∼p2 & p3 & X(∼p2 U p4)))))	146ms	11ms	25ms	33	50	24	5
00074	G((p0 & F p1) → (∼(∼p1 & p2 & X(∼p1 U (∼p1 & p3))) U (p1 | p4)))	187ms	11ms	21ms	34	41	24	5
00075	G(p0 → ((∼(∼p1 & p2 & X(∼p1 U (∼p1 & p3))) U (p1 | p4)) | G∼(p2 & X F p3)))	180ms	20ms	24ms	24	97	20	4
00079	G((p0 & F p1) → (((p2 & X(∼p1 U p3)) → X(∼p1 U (p3 & F p4))) U p1))	827ms	92ms	timeout or memout	38	323	78	20
00080	G(p0 → (((p1 & X(∼p2 U p3)) → X(∼p2 U (p3 & F p4))) U (p2 | G((p1 & X(∼p2 U p3)) → X(∼p2 U (p3 & F p4))))))	6s363ms	1s987ms	timeout or memout	28	5520	54	17
00081	G(p0 → F(p1 & X F p2))	6ms	0ms	132ms	13	34	18	6
00084	G((p0 & F p1) → ((p2 → (∼p1 U (∼p1 & p3 & X(∼p1 U p4)))) U p1))	470ms	40ms	22ms	39	132	30	5

ID	LTL Formula	Time (learning)	Time (non-learning)	Time (Calbrix et al.)	#States lasso DFA	#States lasso NFA	Family of DFA Size	#States det. parity aut.
00085	G(p0 → ((p1 → (∼p2 U (∼p2 & p3 & X(∼p2 U p4)))) U (p2 | G(p1 → (p3 & X F p4)))))	2s905ms	180ms	1s383ms	78	417	63	11
00086	G(p0 → F(p1 & ∼p2 & X(∼p2 U p3)))	20ms	1ms	654ms	17	30	23	7
00089	G((p0 & F p1) → ((p2 → (∼p1 U (∼p1 & p3 & ∼p4 & X((∼p1 & ∼p4) U p5)))) U p1))	1s323ms	83ms	39ms	41	132	30	5
00090	G(p0 → ((p1 → (∼p2 U (∼p2 & p3 & ∼p4 & X((∼p2 & ∼p4) U p5)))) U (p2 | G(p1 → (p3 & ∼p4 & X(∼p4 U p5))))))	14s207ms	368ms	31s656ms	121	397	63	13
00094	G F(∼(p1 ↔ X p1) | ∼(p0 ↔ X p0) | ∼(p2 ↔ X p2) | ∼(p3 ↔ X p3))	159ms	52ms	timeout or memout	20	170	19	16
00101	G((p0 → F p1) & (p2 → F p3) & (p4 → F p5) & (p6 → F p7))	1m50s250ms	3s167ms	timeout or memout	99	1312	642	33
00109	G((∼p0 & p1 & ∼p2 & ∼p3 & ∼p4) → F(∼p5 & ∼p6 & ∼p7 & ∼p8))	333ms	20ms	22ms	7	8	8	2
00110	G((p0 & p1 & ∼p2 & ∼p3 & ∼p4) → F(p5 & ∼p6 & ∼p7 & ∼p8))	322ms	21ms	27ms	7	8	8	2
00118	G((p0 & p1 & ∼p2 & X p2) → X(X∼p1 | (p2 U (∼p2 U (p2 U (∼p1 | p3))))))	210ms	10ms	25ms	47	148	42	6
00124	G((∼(p1 ↔ X p1) | ∼(p0 ↔ X p0) | ∼(p2 ↔ X p2) | ∼(p3 ↔ X p3)) → (X∼p4 & X(∼(∼(p1 ↔ X p1) | ∼(p0 ↔ X p0) | ∼(p2 ↔ X p2) | ∼(p3 ↔ X p3)) U p4)))	timeout or memout	1s349ms	2m13s934ms	1635	2434	1124	65
00129	G((∼p0 & X p0) → X(p0 U (p0 & ∼p1 & X(p0 & p1 & (p0 U (p0 & ∼p1 & X(p0 & p1)))))))	68ms	3ms	7s558ms	61	93	56	8
00130	G((p0 & X∼p0) → X(∼p0 U (∼p0 & ∼p1 & X(∼p0 & p1 & (∼p0 U (∼p0 & ∼p1 & X(∼p0 & p1)))))))	89ms	0ms	7s444ms	61	93	56	8
00131	G((p0 & X∼p0) → X(∼p0 U (∼p0 & ∼p1 & X(∼p0 & p1 & (∼p0 U (∼p0 & ∼p1 & X(∼p0 & p1 & (∼p0 U (∼p0 & ∼p1 & X(∼p0 & p1)))))))))))	480ms	12ms	timeout or memout	104	455	90	10
00132	G((∼p0 & X p0) → X(∼(∼p0 & X p0) U (∼p1 & X p1)))	72ms	3ms	1s216ms	68	69	56	9
00133	G(∼p0 | X(∼p0 | X(∼p0 | X(∼p0 | X(∼p0 | X(∼p0 | X(∼p0 | X(∼p0 | X(∼p0 | X(∼p0 | X(∼p0 | X∼p0)))))))))))	8m18s170ms	207ms	42ms	224	14293	182	13
00160	G(p0 → (∼p1 U (p1 U (∼p1 U (p1 U (p1 & p2))))))	121ms	1ms	27ms	51	34	42	6
00181	G p2 | G p0 | (G(p0 | G F p1) & G(p2 | G F∼p1))	203ms	20ms	20ms	18	425	17	5
00182	G p0 | G p1 | (G(p1 | F G p2) & G(p0 | F G∼p2))	163ms	10ms	22ms	10	162	14	4
00183	∼(G p0 | G p1 | (G(p1 | G F p2) & G(p0 | G F∼p2)))	685ms	10ms	49ms	17	136	11	5
00184	∼(G p0 | G p2 | (G(p0 | F G p1) & G(p2 | F G∼p1)))	606ms	31ms	18ms	10	405	9	4
00190	∼((G F p1 & G F p2 & G F p3) → G(q → F r))	381ms	30ms	241ms	19	125	12	6
00191	∼((G F p1 & G F p2 & G F p3 & G F p4) → G(q → F r))	6s96ms	213ms	32s473ms	35	357	20	8
00192	∼((G F p1 & G F p2 & G F p3 & G F p4 & G F p5) → G(q → F r))	2m51s826ms	1s426ms	timeout or memout	67	1037	36	10
00197	G(p → (q | X q | X X q | X X X q | X X X X q))	547ms	323ms	17ms	48	15840	42	6
00001	F p1	0ms	0ms	11ms	6	4	5	2
00002	F p1 & F p2	0ms	0ms	10ms	10	9	9	4

ID	LTL Formula	Time (learning)	Time (non-learning)	Time (Calbrix et al.)	#States lasso DFA	#States lasso NFA	Family of DFA Size	#States det. parity aut.
00003	F p1 & F p2 & F p3	11ms	0ms	16ms	18	17	17	8
00004	F p1 & F p2 & F p3 & F p4	78ms	0ms	36ms	34	33	33	16
00006	G F p1	0ms	0ms	11ms	4	5	3	1
00007	G F p1 & G F p2	0ms	0ms	13ms	6	14	5	2
00008	G F p1 & G F p2 & G F p3	12ms	0ms	21ms	10	36	9	3
00011	F p1 & F q1	0ms	0ms	11ms	10	9	9	4
00012	F(p1 & F p2) & F(q1 & F q2)	57ms	7ms	24ms	27	35	19	9
00016	F p & F q	0ms	0ms	11ms	10	9	9	4
00017	F(p & X p) & F(q & X q)	30ms	0ms	19ms	41	35	19	9
00021	p0 U (p1 & G p2)	0ms	0ms	12ms	7	6	12	4
00022	p0 U (p1 & X(p2 U p3))	11ms	0ms	25ms	14	10	11	5
00024	F(p0 & X G p1)	0ms	0ms	10ms	6	6	6	2
00025	F(p0 & X(p1 & X F p2))	10ms	0ms	15ms	17	17	9	4
00026	F(p0 & X(p1 U p2))	0ms	0ms	15ms	11	10	7	3
00027	F G p0 | G F p1	0ms	0ms	12ms	6	19	5	1
00028	G(p0 → (p1 U p2))	0ms	0ms	14ms	10	7	12	3
00029	G(p0 & X F(p1 & X F(p2 & X F p3)))	11ms	0ms	18ms	10	15	12	4
00031	(p0 U (p1 U p2)) | (p1 U (p2 U p0)) | (p2 U (p0 U p1))	0ms	0ms	13ms	6	15	7	3
00032	G(p0 → (p1 U (G p2 | G p3)))	20ms	0ms	36ms	17	14	24	8
00036	G∼p0	0ms	0ms	10ms	4	2	6	2
00037	F p0 → (∼p1 U p0)	0ms	0ms	11ms	10	9	13	4
00038	G(p0 → G∼p1)	0ms	0ms	12ms	8	5	10	3
00039	G((p0 & ∼p1 & F p1) → (∼p2 U p1))	9ms	0ms	12ms	16	12	16	4
00040	G((p0 & ∼p1) → (∼p2 W p1))	0ms	0ms	11ms	10	13	12	3
00041	F p0	0ms	0ms	10ms	6	4	5	2
00042	∼p0 W (∼p0 & p1)	0ms	0ms	12ms	7	9	9	3
00043	G∼p0 | F(p0 & F p1)	0ms	0ms	10ms	9	16	9	3
00044	G((p0 & ∼p1) → (∼p1 W (∼p1 & p2)))	1ms	0ms	10ms	10	13	12	3
00045	G((p0 & ∼p1) → (∼p1 U (∼p1 & p2)))	0ms	0ms	14ms	10	7	12	3
00046	∼p0 W (p0 W (∼p0 W (p0 W G∼p0)))	10ms	0ms	11ms	10	47	22	6
00047	F p0 → ((∼p0 & ∼p1) U (p0 | ((∼p0 & p1) U (p0 | ((∼p0 & ∼p1) U (p0 | ((∼p0 & p1) U (p0 | (∼p1 U p0))))))))))	12ms	0ms	12ms	22	29	29	8
00051	G p0	0ms	0ms	10ms	4	2	6	2
00052	F p0 → (p1 U p0)	0ms	0ms	10ms	10	9	13	4
00053	G(p0 → G p1)	0ms	0ms	10ms	8	5	10	3
00054	G((p0 & ∼p1 & F p1) → (p2 U p1))	10ms	0ms	13ms	16	12	16	4
00055	G((p0 & ∼p1) → (p2 W p1))	2ms	0ms	14ms	10	13	12	3
00056	∼p0 W p1	0ms	0ms	12ms	7	9	9	3
00057	F p0 → (∼p1 U (p0 | p2))	0ms	0ms	11ms	10	9	13	4
00058	G∼p0 | F(p0 & (∼p1 W p2))	10ms	0ms	19ms	18	29	13	4
00059	G((p0 & ∼p1 & F p1) → (∼p2 U (p1 | p3)))	19ms	0ms	15ms	17	13	16	4
00060	G((p0 & ∼p1) → (∼p2 W (p1 | p3)))	10ms	0ms	12ms	10	13	12	3
00061	G(p0 → F p1)	0ms	0ms	14ms	7	8	8	2
00062	F p0 → ((p1 → (∼p0 U (∼p0 & p2))) U p0)	0ms	0ms	11ms	10	23	15	4
00063	G(p0 → G(p1 → F p2))	0ms	0ms	14ms	11	18	12	3
00064	G((p0 & ∼p1 & F p1) → ((p2 → (∼p1 U (∼p1 & p3))) U p1))	55ms	1ms	20ms	21	36	20	4

ID	LTL Formula	Time (learning)	Time (non-learning)	Time (Calbrix et al.)	#States lasso DFA	#States lasso NFA	Family of DFA Size	#States det. parity aut.
00066	F p0 → (∼p0 U (∼p0 & p1 & X(∼p0 U p2)))	0ms	0ms	14ms	10	22	13	4
00067	F p0 → (∼p1 U (p0 | (∼p1 & p2 & X(∼p1 U p3))))	11ms	0ms	11ms	13	22	17	5
00068	G∼p0 | (∼p0 U ((p0 & F p1) → (∼p1 U (∼p1 & p2 & X(∼p1 U p3)))))	11ms	0ms	14ms	12	36	15	5
00070	G(p0 → (F p1 → (∼p1 U (p2 | (∼p1 & p3 & X(∼p1 U p4))))))	70ms	10ms	21ms	21	50	20	4
00071	F(p0 & X F p1) → (∼p0 U p2)	0ms	0ms	11ms	11	16	13	4
00072	F p0 → (∼(∼p0 & p1 & X(∼p0 U (∼p0 & p2))) U (p0 | p3))	10ms	0ms	15ms	13	21	17	5
00073	G∼p0 | (∼p0 U (p0 & (F(p1 & X F p2) → (∼p1 U p3))))	41ms	10ms	17ms	21	74	17	5
00076	G((p0 & X F p1) → X F(p1 & F p2))	11ms	1ms	12ms	12	62	24	4
00077	F p0 → (((p1 & X(∼p0 U p2)) → X(∼p0 U (p2 & F p3))) U p0)	47ms	13ms	20ms	16	155	31	6
00078	G(p0 → G((p1 & X F p2) → X(∼p2 U (p2 & F p3))))	76ms	12ms	19ms	18	111	28	5
00082	F p0 → ((p1 → (∼p0 U (∼p0 & p2 & X(∼p0 U p3)))) U p0)	12ms	11ms	18ms	13	84	22	5
00083	G(p0 → G(p1 → (p2 & X F p3)))	12ms	0ms	11ms	13	18	16	4
00087	F p0 → ((p1 → (∼p0 U (∼p0 & p2 & ∼p3 & X((∼p0 & ∼p3) U p4)))) U p0)	32ms	20ms	20ms	13	84	23	5
00088	G(p0 → G(p1 → (p2 & ∼p3 & X(∼p3 U p4))))	49ms	0ms	22ms	19	16	16	4
00091	G(F p0 & F∼p0)	0ms	0ms	14ms	6	7	5	2
00092	G F p0 & G F∼p0	0ms	0ms	11ms	6	7	5	2
00093	G F(∼(p1 ↔ X p1) | ∼(p0 ↔ X p0))	1ms	0ms	18ms	8	30	7	4
00095	G((p0 → F∼p0) & (∼p0 → F p0))	0ms	0ms	10ms	6	11	5	2
00096	G(p0 → F(p0 & p1))	0ms	0ms	10ms	7	8	8	2
00097	G(p0 → F((∼p0 & p1 & p2 & p3) → F p4))	0ms	0ms	10ms	4	11	3	1
00098	G(p0 → F∼p1)	0ms	0ms	11ms	7	8	8	2
00099	G(p0 → F(p1 → F p2))	0ms	0ms	11ms	7	25	8	2
00100	G(p0 → F((p1 & p2) → F p3))	1ms	0ms	10ms	7	25	8	2
00102	G(∼p0 & ∼p1)	0ms	0ms	11ms	4	2	6	2
00103	G∼(p0 & p1)	0ms	0ms	12ms	4	2	6	2
00104	G(p0 → p1)	0ms	0ms	10ms	4	2	6	2
00105	G((p0 → ∼p1) & (p1 → ∼p0))	0ms	0ms	12ms	4	2	6	2
00106	G(∼p0 → (p1 ↔ ∼p2))	0ms	0ms	11ms	4	2	6	2
00107	G((∼p0 & (p1 | p2 | p3)) → p4)	0ms	0ms	15ms	4	2	6	2
00108	G((p0 & p1) → (p2 | ∼(p3 & p4)))	0ms	0ms	10ms	4	2	6	2
00111	G(∼p0 → ∼(p1 & p2 & p3 & p4 & p5))	1ms	0ms	12ms	4	2	6	2
00112	G(∼p0 → ((p1 & p2 & p3 & p4) → ∼p5))	1ms	0ms	11ms	4	2	6	2
00113	G((p0 & p1) → (p2 | p3 | ∼(p4 & p5)))	1ms	0ms	11ms	4	2	6	2
00114	G((∼p0 & (p1 | p2 | p3 | p4)) → (∼p5 ↔ p6))	10ms	0ms	11ms	4	2	6	2
00115	G((p0 & p1) → (p2 | p3 | p4 | ∼(p5 & p6)))	9ms	0ms	14ms	4	2	6	2
00116	G((p0 & p1) → (p2 | p3 | p4 | p5 | ∼(p6 & p7)))	20ms	0ms	12ms	4	2	6	2
00117	G((p0 & p1 & ∼p2 & X p2) → X(p3 | X(∼p1 | p3)))	21ms	0ms	21ms	21	54	20	4
00119	G(p0 & p1 & ∼p2 & X p2) → X(X∼p1 | (p2 U (∼p2 U (p2 U (∼p1 | p3)))))	0ms	0ms	11ms	4	31	3	1

ID	LTL Formula	Time (learning)	Time (non-learning)	Time (Calbrix et al.)	#States lasso DFA	#States lasso NFA	Family of DFA Size	#States det. parity aut.
00120	G(p0 → (p1 U (∼p1 U (∼p2 | p3))))	20ms	0ms	13ms	20	14	20	4
00121	G(p0 → (p1 U (∼p1 U (p2 | p3))))	21ms	0ms	19ms	20	14	20	4
00122	G((∼p0 & p1) → X p2)	0ms	0ms	14ms	9	6	12	3
00123	G(p0 → X(p0 | p1))	0ms	0ms	11ms	9	6	12	3
00125	G((p0 & ∼p1 & X p1 & X p0) → (p2 → X p3))	1ms	0ms	12ms	9	36	12	3
00126	G(p0 → X(∼p0 U p1))	0ms	0ms	11ms	10	7	12	3
00127	G((∼p0 & X p0) → X((p0 U p1) | G p0))	8ms	0ms	16ms	17	34	20	4
00128	G((∼p0 & X p0) → X(p0 U (p0 & ∼p1 & X(p0 & p1))))	13ms	0ms	35ms	30	26	30	6
00134	G((X p0 → p0) → (p1 ↔ X p1))	10ms	0ms	14ms	27	16	32	6
00135	G((X p0 → p0) → ((p1 → X p1) & (∼p1 → X∼p1)))	10ms	0ms	18ms	27	16	32	6
00136	∼p0 U G∼((p1 & p2) | (p3 & p4) | (p2 & p3) | (p2 & p4) | (p1 & p4) | (p1 & p3))	0ms	0ms	15ms	5	5	10	3
00137	∼p0 U p1	0ms	0ms	11ms	6	4	7	3
00138	(p0 U p1) | G p0	0ms	0ms	10ms	7	9	9	3
00139	p0 & X G∼p0	0ms	0ms	12ms	5	4	8	3
00140	X G(p0 → (G∼p1 | (∼X p1 U p2)))	12ms	0ms	19ms	13	27	20	5
00141	X G((p0 & ∼p1) → (G∼p1 | (∼p1 U p2)))	2ms	0ms	14ms	11	17	14	4
00142	X G((p0 & p1) → ((p1 U p2) | G p1))	2ms	0ms	13ms	11	17	14	4
00143	X p0 & G((∼p0 & X p0) → X X p0)	0ms	0ms	16ms	22	40	24	6
00144	(G F p1 & G F p0) → G F p2	21ms	0ms	39ms	8	36	7	2
00145	G(p0 → (p1 & (p2 U p3)))	0ms	0ms	14ms	10	7	12	3
00146	F(p0 | p1)	0ms	0ms	11ms	6	4	5	2
00147	G F(p0 | p1)	0ms	0ms	10ms	4	5	3	1
00148	(p0 U p1) → ((p2 U p3) | G p2)	11ms	0ms	12ms	13	14	17	5
00149	G(p0 → (∼p1 U (p1 U (∼p1 & (p2 R ∼p1)))))	48ms	1ms	25ms	34	22	30	5
00150	G(p0 → (p1 R ∼p2))	0ms	0ms	12ms	10	6	12	3
00151	G(∼p0 → F p0)	0ms	0ms	12ms	4	7	3	1
00152	G(p0 → F(p1 | p2))	0ms	0ms	13ms	7	8	8	2
00153	∼(∼(p0 | p1) U p2) & G(p3 → ∼(∼(p0 | p1) U p2))	10ms	0ms	14ms	10	12	12	3
00154	G∼p0 → G∼p1	0ms	0ms	11ms	9	8	9	3
00155	G(p0 → (G∼p1 | (∼p2 U p1)))	10ms	0ms	12ms	16	12	16	4
00156	G(p0 → (p1 R (p1 | ∼p2)))	0ms	0ms	10ms	10	6	12	3
00157	G((p0 & p1) → (∼p1 R (p0 | ∼p1)))	0ms	0ms	12ms	9	6	12	3
00158	G(p0 → F(p1 & p2))	0ms	0ms	13ms	7	8	8	2
00159	G(p0 → (∼p1 U (p1 U (p1 & p2))))	13ms	0ms	13ms	20	14	20	4
00161	G F p0 → G F p1	0ms	0ms	16ms	6	19	5	1
00162	G F(p0 | p1) & G F(p1 | p2)	1ms	0ms	18ms	6	14	5	2
00163	p0 U p1	0ms	0ms	10ms	6	4	7	3
00164	p0 U (p1 U p2)	0ms	0ms	10ms	8	7	9	4
00165	∼(p0 U (p1 U p2))	0ms	0ms	11ms	10	8	13	4
00166	F p0 U G p1	0ms	0ms	17ms	6	15	6	2
00167	G p0 U p1	0ms	0ms	13ms	11	7	13	5
00168	∼(F p0 ↔ F p1)	0ms	0ms	14ms	9	21	12	4
00169	∼(G F p0 → G F p1)	0ms	0ms	11ms	4	14	4	1
00170	∼(G F p0 ↔ G F p1)	23ms	10ms	20ms	5	243	5	2
00171	p0 R (p0 | p1)	0ms	0ms	10ms	7	4	9	3
00172	(X p0 U X p1) | ∼X(p0 U p1)	0ms	0ms	10ms	4	20	3	1
00173	(X p0 U p1) | ∼X(p0 U (p0 & p1))	0ms	0ms	10ms	4	11	3	1
00174	((X p0 U p1) | ∼X(p0 U (p0 & p1))) & G(p0 → F p1)	1ms	0ms	12ms	7	31	8	2

ID	LTL Formula	Time (learning)	Time (non-learning)	Time (Calbrix et al.)	#States lasso DFA	#States lasso NFA	Family of DFA Size	#States det. parity aut.
00175	G(p0 → F p1) & ((X p0 U X p1) | ∼X(p0 U p1))	2ms	0ms	11ms	7	63	8	2
00176	∼G(p0 → X(p1 R p2))	0ms	0ms	13ms	11	10	7	3
00177	∼(F G p0 | F G p1)	0ms	0ms	12ms	6	14	5	2
00178	G(F p0 & F p1)	0ms	0ms	12ms	6	13	5	2
00179	F p0 & F∼p0	0ms	0ms	12ms	10	7	9	4
00180	(p0 & X p1) R X(((p2 U p3) R p0) U (p2 R p0))	23ms	31ms	17ms	17	235	17	6
00185	G(p0 | X G p1) & G(p2 | X G∼p1)	10ms	0ms	12ms	12	13	14	4
00186	G(p0 | (X p1 & X∼p1))	0ms	0ms	14ms	4	6	6	2
00187	p0 | (p1 U p0)	0ms	0ms	12ms	6	6	7	3
00188	∼(G F p1 → G(q → F r))	0ms	0ms	11ms	7	17	6	2
00189	∼((G F p1 & G F p2) → G(q → F r))	31ms	0ms	25ms	11	45	8	4
00193	G(p → q)	0ms	0ms	12ms	4	2	6	2
00194	G(p → (q | X q))	0ms	0ms	10ms	9	6	12	3
00195	G(p → (q | X q | X X q))	0ms	0ms	16ms	18	30	20	4
00196	G(p → (q | X q | X X q | X X X q))	31ms	10ms	13ms	31	418	30	5
00198	G(p → (q | X(q | X q)))	0ms	0ms	10ms	18	16	20	4
00199	G(p → (q | X(q | X(q | X q))))	10ms	0ms	18ms	31	48	30	5
00200	G(p → (q | X(q | X(q | X(q | X q)))))	51ms	0ms	16ms	48	160	42	6
00201	G(p → (q & X q))	0ms	0ms	13ms	9	6	12	3
00202	G(p → (q & X q & X X q))	3ms	0ms	11ms	17	20	20	4
00203	G(p → (q & X q & X X q & X X X q))	13ms	1ms	11ms	28	148	30	5
00204	G(p → (q & X q & X X q & X X X q & X X X X q))	83ms	61ms	22ms	42	2424	42	6
00205	G(p → (q & X(q & X q)))	2ms	0ms	10ms	17	16	20	4
00206	G(p → (q & X(q & X(q & X q))))	11ms	0ms	13ms	28	48	30	5
00207	G(p → (q & X(q & X(q & X(q & X q)))))	21ms	1ms	18ms	42	160	42	6