How Hard is Finding Shortest Counter-Example Lassos in Model Checking?[⋆]

Rüdiger Ehlers

Clausthal University of Technology, Clausthal-Zellerfeld, Germany
`ruediger.ehlers@tu-clausthal.de`

Abstract. Modern model checkers help system engineers to pinpoint the reason for the faulty behavior of a system by providing counter-example traces. For finite-state systems and ω-regular specifications, they come in the form of lassos. Lassos that are unnecessarily long should be avoided, as they make finding the cause for an error in a trace harder. We give the first thorough characterization of the computational complexity of finding the shortest and approximately shortest counter-example lassos in model checking for the full class of ω-regular specifications. We show how to build (potentially exponentially larger) *tight automata* for arbitrary ω-regular specifications, which can be used to reduce finding shortest counter-example lassos for some finite-state system to finding a shortest accepting lasso in a (product) Büchi automaton. We then show that even approximating the size of the shortest counter-example lasso is an NP-hard problem for any polynomial approximation function, which demonstrates the hardness of obtaining short counter-examples in practical model checking. Minimizing only the length of the lasso cycle is however possible in polynomial time for a fixed but arbitrary upper limit on the size of strongly connected components in specification automata.

1 Introduction

With *model checking*, we can exhaustively test if a reactive system (or a model of it) satisfies a given specification. A key feature of most model checking tools is that they provide a *counter-example* whenever this is not the case. Counter-examples are helpful for the system engineer to understand the reason for non-satisfaction and to find out whether the model is erroneous and hence needs to be fixed, or whether the design itself has an error, which necessitates refining the design. For safety properties, such a counter-example can be a finite trace, where the violation of the property becomes apparent with the last state of the trace. For a specification outside of the set of safety properties such as a *liveness property*, a finite trace cannot show the absence of a specification violation in the model. In this case, an infinite trace is needed, and if and only if a finite-state system does not satisfy an ω-regular specification, there is a *lasso-shaped*

⋆ This work was supported by the German Science Foundation (DFG) under Grant No. 322591867. It was inspired by discussions at Dagstuhl seminar 19081.

counter-example that can be presented to the engineer. Such lassos consist of a *handle* that shows how the system initially evolves, followed by a *cycle* that shows repetitive behavior that the system can follow indefinitely long. The trace consisting of following the handle once and cycle infinitely often is then the counter-example.

To fulfill the promise of helping the system engineer with revising the model of the design, a counter-example needs to be *understandable.* While a full formalization of this requirement is difficult, it is commonly agreed on that counter-examples should be *short*, as deriving the core reason for the violation of overly long counter-examples is cumbersome. The length of counter-example lassos can be defined both over the lengths of the handle and the cycle, but the most common definition is the sum of these. Finding a shortest counter-example lasso in a *Büchi automaton* that models the intersection between the complement of the specification and the traces that the system permits is computationally easy as polynomial-time algorithms are known for this task [18, 11]. The intersection Büchi automaton in this context is built from a finite-state machine description of the system and a Büchi automaton representation of the specification. This means that the syntactic structure of the latter influences the length of the counter-example lassos in the product, and hence finding a shortest lasso in it does not mean that the lasso's projection on the system FSM yields a shortest lasso in the FSM alone. Hence, following this approach can lead to unnecessarily long counter-examples.

When the specification of the system is given as a *linear temporal logic* (LTL) property, this problem can however be avoided [17]. The translation from LTL to a Büchi automaton can be made *tight*, i.e., such that it ensures that lassos in the finite-state machine system description along which a specification is violated give rise to lassos in the product automaton of the same size. In this way, shortest counter-example lassos are present in the product automaton. The construction is however bound to LTL and algorithms that post-process specification Büchi automata to reduce their size or improve their amenability for model checking [19] can break tightness. Since such post-processing procedures have been shown to be important for good model checking performance, computing shortest counter-example lassos for LTL specifications remains practically more difficult than computing *any* counter-example. Even more important, novel specification logics such as *linear dynamic logic* (LDL, [4]) and *property specification logic* (PSL,[7]) have recently been proposed to achieve full ω-regular expressivity and to support the industrial adoption of model checking techniques. The results from Schuppan and Biere [17] do not carry over to these logics, hence leaving a gap for the question of how difficult the problem of obtaining shortest counter-example lassos for these logics and ω-regular specifications in general is.

In this paper, we revisit the computational hardness of the problem of computing shortest counter-example lassos for arbitrary ω-regular specifications. We start by showing that by applying two constructions by Calbrix et al. [2] and Farzah et al. [9] in succession, we can translate an arbitrary Büchi automaton into an equivalent tight automaton. The construction leads to an exponential blow-up while already for the safety case, the lower bound identified by Kupferman and Vardi [14] is also exponential. While a tight Büchi automaton can be used to find a shortest counter-example lasso, the automaton blow-up leads to the question if there is a more efficient way to compute shortest counter-example lassos with specification automata that are not tight. To study this question, we define the problem of finding a short *component lasso* in a Büchi automaton that is the product of a specification automaton and a finite-state machine. While it is relatively easy to show that the problem is NP-complete, we mainly examine the approximation hardness of the problem, as approximately shortest counter-example lassos may also suffice in practical applications. Unfortunately, it turns out that the problem is also NP-hard to approximate within any polynomial approximation function. On the positive side, we give a polynomial-time construction for minimizing the lassos cycle length for specification Büchi automata with small strongly connected components (of a fixed maximal size), which are common when model checking against liveness properties.

The hardness results that we present provide an a-posteriori justification for heuristic approaches to finding short counter-example lassos in model checking.

1.1 Related Work

Minimizing the size of counter-example traces or lassos is a classical problem in the model checking literature. Standard depth-first search Büchi automaton language emptiness checking algorithms commonly implemented in explicit-state model checkers such as `spin` [12] are not guaranteed to yield shortest counter-examples. A simple improvement is to minimize the lasso cycle length in the product automaton between the system and the specification automaton, which can be done in time polynomial in the sizes of the automata. Gastin et al. [10] give an approach to find shortest counter-examples in explicit-state model checking without increasing memory usage substantially. Edelkamp et al. [5] give an approach for doing so in explicit-state model checking when using external memory for storing states. In symbolic model checking using binary decision diagrams (BDDs), lasso cycle length minimization comes as a side effect of the typically implemented algorithms. Clarke et al. [3] showed that adding fairness requirements to the lasso to be found (such as in the acceptance condition of *generalized Büchi automata*) makes the problem of finding shortest accepting lassos NP-hard, even in the product automata used in model checking.

All approaches mentioned so far can however still compute unnecessarily long counter-examples as they search for short counter-example lassos in the product automaton. As an alternative, Schuppan and Biere [17] showed how to compute *tight* specification automata for linear time temporal logic (LTL). Such specification automata ensure that shortest counter-example lassos in the finite-state machine modeling the system to be tested induce shortest lassos in the product automaton, enabling the application of an approach to obtain short accepting lassos in Büchi automata [18, 11, 10] to compute shortest counter-example lassos. Post-processing such specification automata by simulation-based automaton minimization [8], as usually done in practical model checking, can break this property, though, requiring the specification automata to remain unaltered after their construction.

For the safety fragment of the ω-regular specifications, Kupferman and Vardi gave a construction to build tight automata [14]. Starting with a non-deterministic Büchi automaton, their construction leads to an exponential blowup in the specification automaton size, which they show to be unavoidable.

For general ω-regular properties and for every system to be checked, the liveness-to-reachability reduction from [16] can be applied to obtain a Büchi automaton that is, in a sense, tight enough not to miss the shortest counter-example in the product. The necessary blow-up depends on the *diameter* of the system to be checked (or alternatively some refinement of this concept), and hence the approach cannot be used to compute one Büchi automaton that can be used for finding shortest counter-examples for *all* finite-state systems. We improve upon this result in this paper by giving a construction to obtain not only "tight-enough" Büchi automata, but completely tight automata, for which the diameter of the system to be checked does not need to be known.

An alternative type of counter-example has been defined by Kupferman and Sheinvald [13], where the goal is to find the shortest lasso-shaped *input/output word* that the system can read and emit during a counter-example lasso. They call such lasso-shaped words *witnesses* (for the violation of a specification by a system). While these can be much more compact than counter-example lassos (for instance when the output of the system is always the same along a non-trivial counter-example lasso), even approximating the size of the shortest such counter-example word within any polynomial approximation function is NP-hard [6] for specifications in all commonly used automata types. Hence, abstracting from the concrete system states adds complexity to the problem, which is a motivation to revisit the simpler counter-example lassos in this work.

2 Preliminaries

Words and graphs: Given an alphabet Σ, we denote the set of finite words over Σ as Σ^*, and the set of infinite words as Σ^ω. A word in Σ^ω is called *ultimately periodic* if it is of the form uv^ω for some $u, v \in \Sigma^*$, where the ω operator denotes infinite repetition of the operand. The empty word is denoted by ϵ. A graph is a two-tuple (V, E) consisting of a set of vertices V and an edge relation $E \subseteq V \times V$.

Automata over finite words: An automaton over finite words is a tuple $\mathcal{A} = (Q, \Sigma, \delta_\mathcal{A}, Q_0, F)$ with the finite set of states Q, the finite alphabet Σ, the transition relation $\delta_\mathcal{A} \subseteq Q \times \Sigma \times Q$, the set of initial states $Q_0 \subseteq Q$, and the set of accepting states $F \subseteq Q$. We say that $\mathcal{A}$ is *deterministic* if for every $q \in Q$ and $x \in \Sigma$, there is only at most one $q' \in Q$ with $(q, x, q') \in \delta_\mathcal{F}$, and furthermore Q_0 contains exactly one element. A *run* for a finite word $w = w_0 \ldots w_{n-1} \in \Sigma^*$ is a sequence $\pi = \pi_0 \ldots \pi_n$ with $\pi_0 \in Q_0$ and for all $1 \leq i \leq n$, we have $\pi_i \in \delta_\mathcal{A}(\pi_{i-1}, x_{i-1})$. We say that $\mathcal{A}$ accepts a finite word $w \in \Sigma^*$ if there exists an accepting *run* $\pi = \pi_0 \ldots \pi_n$ for w, i.e., for which $\pi_n \in F$. The set of words accepted by $\mathcal{A}$ is also called its *language* and denoted by $\mathcal{L}(\mathcal{A})$. The size of an automaton, written as $|\mathcal{A}|$ is defined to be the sum of the number of states and the number of transitions.

Büchi automata: A (non-deterministic) Büchi automaton is a tuple $\mathcal{A} = (Q, \Sigma, \delta_\mathcal{A}, Q_0, F)$ with the same structure as an automaton over finite words, the same definitions of determinism and the size of an automaton, and the same definition of runs, except that they can be infinitely long, as Büchi automata represent languages over Σ^ω. We say that $\mathcal{A} = (Q, \Sigma, \delta_\mathcal{A}, Q_0, F)$ accepts a word $w \in \Sigma^\omega$ if and only if there exists an *accepting run* $\pi = \pi_0\pi_1 \ldots$ of $\mathcal{A}$ for w. A run is accepting if for infinitely many $i \in \mathbb{N}$, we have $\pi_i \in F$. The set of accepted words again forms the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$. We define the reachability relation $R_\mathcal{A} \subseteq Q \times Q$ of $\mathcal{A}$ as the transitive closure of $\{(q, q') \in Q^2 \mid \exists x \in \Sigma, (q, x, q') \in \delta\}$. We say that a set of states $Q' \subseteq Q$ forms a *strongly connected component* (SCC) of $\mathcal{A}$ if for every $q, q' \in Q'$, we have that $(q, q') \in R_\mathcal{A}$. We say that $\mathcal{A}$ encodes a *safety language* if every word that is not in the language has a prefix all of whose extensions are also not in the language.

Regular and ω-regular expressions: Such expressions are a way to represent languages over finite and infinite words. Starting from elements in Σ, they are composed using the *concatenation* ($\cdot$), *union* ($\cup$), *intersection* ($\cap$), and *finite repetition* ($*$) operators. The "$\cdot$" operator is often omitted when clear from the context. In case of ω-regular expressions, the additional $^\omega$ operator denotes infinite repetition. No sub-expression can be concatenated to the right of such an operator application. The set of properties over infinite words representable by ω-regular expressions is called the *ω-regular languages.* It is known that these are exactly the properties representable by non-deterministic Büchi automata.

Finite-state machines: A *finite-state machine* (FSM) is a tuple $\mathcal{F} = (S, \Sigma, \delta_\mathcal{F}, s_0, L)$ with the finite set of states S, the alphabet Σ, the transition relation $\delta_\mathcal{F} \subseteq S \times S$, the initial state s_0, and the labeling function L. A trace of $\mathcal{F}$ is an (infinite) sequence $\rho = \rho_0\rho_1 \ldots \in S^\omega$ such that $\rho_0 = s_0$ and for every $i \in \mathbb{N}$, we have $(\rho_i, \rho_{i+1}) \in \delta_\mathcal{F}$. The trace *induces a word* $w = w_0w_1 \ldots \in \Sigma^\omega$ with $w_i = L(\rho_i)$ for all $i \in \mathbb{N}$.

Model checking: Given an FSM $\mathcal{F}$ and an *error specification* in the form a Büchi automaton $\mathcal{A}$ over the same alphabet, the model checking problem is to test if $\mathcal{F}$ has a trace that induces a word in the language of $\mathcal{A}$. Whenever this is the case, we are interested in a *counter-example* of $\mathcal{F}$ that proves that it has a run whose word is in the language of $\mathcal{A}$. Such a counter-example has the form of a *lasso*, i.e., is of the shape $((c_1, \ldots, c_m), (c'_1, \ldots, c'_n))$, where $c_1, \ldots, c_m$ is the *lasso handle* and $c'_1, \ldots, c'_n$ is the *lasso cycle*. They fulfill the following conditions:

– All elements $c_1, \ldots, c_m, c'_1, \ldots, c'_n$ are in S (the lasso elements are states).
– For all $1 < i \leq m$, we have $(c_{i-1}, c_i) \in \delta$, and for all $1 < i \leq n$, we have $(c'_{i-1}, c'_i) \in \delta$ (the lasso parts describe state transitions).
– We have $c_m = c'_1$ (lasso handle and lasso cycle are connected) and $(c'_n, c'_1) \in \delta_\mathcal{F}$ (the lasso cycle is closed).

The lasso $((c_1, \ldots, c_m), (c'_1, \ldots, c'_n))$ represents the trace $c_1, \ldots, c_{m-1}(c'_1, \ldots, c'_n)^\omega$, i.e., on which the lasso cycle is repeated an infinite number of times. We say that this lasso is a counter-example for $\mathcal{A}$ if the word induced by the trace is in the language of $\mathcal{A}$. The constants m and n are also called the *handle length* and *cycle length* of a lasso. The *combined length* of a lasso is defined to be $m + n - 1$, where the subtraction by one comes from counting the state shared by the handle and the cycle only once.

We say that a lasso in $\mathcal{F}$ is a counter-example to some state q in $\mathcal{A}$ if there exists a counter-example lasso for $\mathcal{F}$ and $\mathcal{A}$, where state q is assumed to be the sole initial state in $\mathcal{A}$. We say that a lasso cycle is a counter-example lasso cycle if it can be completed to a counter-example lasso with a single-state handle (which is then part of the cycle itself).

Product Büchi automaton for model checking: A finite-state machine $\mathcal{F} = (S, \Sigma, \delta_\mathcal{F}, s_0, L)$ has a trace (a *counter-example*) that induces a word in the language represented by an (error) specification in the form of a Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta_\mathcal{A}, Q_0, F)$ if and only if there exists a counter-example lasso for the FSM. Testing if such a lasso exists can be done by testing the *product Büchi automaton* of $\mathcal{F}$ and $\mathcal{A}$ for language emptiness. Formally, we define this product automaton as $\mathcal{P} = (Q_\mathcal{P}, \Sigma, \delta_\mathcal{P}, Q_{\mathcal{P},0}, F_\mathcal{P})$ with $Q_\mathcal{P} = Q \times S$, $Q_{\mathcal{P},0} = Q_0 \times \{s_0\}$, $F_\mathcal{P} = F \times S$, and the transition relation is defined as

$$\delta_\mathcal{P} = \{((q, s), x, (q', s')) \in Q_\mathcal{P} \times \Sigma \times Q_\mathcal{P} \mid (s, s') \in \delta_\mathcal{F}, q' \in \delta(q, L(s))\}.$$

A lasso in $\mathcal{P}$ is defined in the same way as in a finite-state machine. A lasso in $\mathcal{P}$ is *accepting* if its cycle contains at least one state from $F_\mathcal{P}$. Since a Büchi automaton has a non-empty language exactly if and only if such a lasso can be found, $\mathcal{P}$ can be used to model check $\mathcal{F}$ against $\mathcal{A}$.

Tight automata: While a product Büchi automaton $\mathcal{P}$ between an FSM $\mathcal{F}$ and a Büchi automaton can be used to check if a counter-example for $\mathcal{F}$ and $\mathcal{A}$ exists, the combined length of the shortest accepting lasso in $\mathcal{P}$ may be higher than the length of the shortest counter-example lasso for $\mathcal{A}$ in $\mathcal{F}$. Intuitively, the reason for this difference is that $\mathcal{P}$ incorporates the structure of $\mathcal{A}$, whereas the definition of a counter-example lasso in $\mathcal{F}$ does not. However, the structure of $\mathcal{A}$ may be suitable to avoid this issue. We say that $\mathcal{A}$ is *tight* if for every counter-example lasso for every FSM $\mathcal{F}$, there is an accepting lasso in $\mathcal{P}$ of the same length. Since $\mathcal{P}$ is the product of $\mathcal{A}$ and $\mathcal{F}$, it is easy to obtain a counter-example lasso from an accepting lasso in $\mathcal{P}$ such that both are of the same length. This allows using $\mathcal{P}$ to find shortest counter-example lassos, and since it is known that finding shortest accepting lassos in $\mathcal{P}$ is solvable in polynomial time [18, 11], so is the former problem.

Satisfiability problem: The *satisfiability* problem is to check if a Boolean formula in conjunctive normal form over some set of Boolean variables $\{v_1, \ldots, v_n\}$ has a satisfying assignment. The conjuncts of the formula are called its *clauses*, and the number of clauses is also denoted by $|\psi|$. The satisfiability problem often serves as the canonical NP-complete problem. We write $c \in \psi$ for some clause c if c is a clause in ψ.

3 Tightening Büchi automata

With a tight specification automata, computing shortest counter-examples is (computationally) easy. We will show in this section that by utilizing two previous constructions from Calbrix et al. [2] and Farzah et al. [9], we can translate an arbitrary Büchi automaton into an equivalent tight automaton. The first construction is captured by the following theorem:

Theorem 1 ([2], Section 5). *Given a non-deterministic Büchi automaton $\mathcal{A}$ over an alphabet Σ with n states, we can build a deterministic finite-state automaton $\mathcal{A}'$ over the alphabet $\Sigma \cup \{\$\}$ of size exponential in n that accepts exactly the words $u\$w$ with $u, w \in \Sigma^*$ for which uv^ω is accepted by $\mathcal{A}$. Building $\mathcal{A}'$ does not take more time than polynomial in the combined input and output sizes.*

Two Büchi automata accept the same language if they accept the same ultimately periodic words [2]. Since these are captured by the automaton $\mathcal{A}'$, that automaton encodes the essence of an ω-regular language. Since $\mathcal{A}'$ is a deterministic automaton over finite words (*DFA*), it can also be minimized in polynomial time.

The automaton $\mathcal{A}'$ can now be translated back to an automaton $\mathcal{A}''$ with the same language as $\mathcal{A}$ with only polynomial blow-up. A construction for this step has been given by Farzah et al. [9], whose properties needed in this section we distill into the following proposition:

Proposition 1 ([9]). *Let $\mathcal{A}' = (Q', \Sigma \cup \{\$\}, \delta', Q'_0, F')$ be a DFA that accepts exactly the words $u\$v$ with $u, v \in \Sigma^*$ for which uv^ω is a word in some ω-regular language L. We can compute, in polynomial time, a set of pairs P of regular languages such that*

$$L = \bigcup_{(A,B) \in P} AB^\omega. \tag{1}$$

Furthermore, (1) P is of cardinality quadratic in $|Q'|$, (2) for every $(A, B) \in P$, the languages A and B are representable by DFAs with at most $|Q'| + 1$ states, and (3) for every word $u\$v$ in the language of $\mathcal{A}'$, there exists some $(A, B) \in P$ with $u \in \mathcal{L}(A)$ and $v \in \mathcal{L}(B)$.

This characterization enables the efficient construction of a non-deterministic Büchi automaton for an ω-regular language from a DFA accepting its ultimately periodic words. While the core idea of the following construction was already suggested in a footnote in [9], we changed it substantially to make the resulting automaton tight.

Lemma 1. *Let $\mathcal{A}' = (Q', \Sigma \cup \{\$\}, \delta', Q'_0, F')$ be a DFA that accepts exactly the words $u\$v$ with $u, v \in \Sigma^*$ for which uv^ω is a word in some ω-regular language L. Let $P = \{P_1, \ldots, P_n\}$ be the set of pairs of regular languages described in Proposition 1. Let furthermore $\mathcal{A}^A_i = (Q^A_i, \Sigma, \delta^A_i, Q^A_{0,i}, F^A_i)$ and $\mathcal{A}^B_i = (Q^B_i, \Sigma, \delta^B_i, Q^B_{0,i}, F^B_i)$ be the DFAs for the elements $P_i = (A_i, B_i)$ for $1 \leq i \leq n$, where w.l.o.g, all states in the these automata have distinct names.*

We can build a non-deterministic Büchi automaton $\hat{\mathcal{A}} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{Q}_0, \hat{F})$ capturing L with the following components:

$$\hat{Q} = \bigcup_{1 \leq i \leq n} Q_i^A \cup (Q_i^B \times \mathbb{B})$$

$$\hat{Q}_0 = \left(\bigcup_{1 \leq i \leq n} Q_{0,i}^A \right) \cup \bigcup_{1 \leq i \leq n, Q_{0,i}^A \cap F_i^A \neq \emptyset} Q_{0,i}^B \times \mathbb{B}$$

$$\begin{aligned}
\hat{\delta} = & \bigcup_{1 \leq i \leq n} \delta_i^A \cup \{((q,b),x,(q',b') \mid (q,x,q') \in \delta_i^B, b' = (q' \in F_i^B)\} \\
& \cup \bigcup_{1 \leq i \leq n} \{(q,x,(q',b)) \mid q \in Q_i^A, x \in \Sigma, \exists q'' \in F_i^A, (q,x,q'') \in \delta_i^A, \\
& \qquad q' \in Q_{0,i}^B, b \in \mathbb{B}\} \\
& \cup \bigcup_{1 \leq i \leq n} \{((q,b),x,(q',\mathbf{true})) \mid q \in Q_i^B, x \in \Sigma, \exists q'' \in F_i^B, (q,x,q'') \in \delta_i^B, \\
& \qquad q' \in Q_{0,i}^B, b \in \mathbb{B}\}
\end{aligned}$$

$$\hat{F} = \bigcup_{1 \leq i \leq n} Q_{0,i}^B \times \{\mathbf{true}\}$$

The construction in the lemma essentially defines a Büchi automaton implementing Equation 1, with the modification that states in the automata $\mathcal{A}_i^B$ have been duplicated by attaching a Boolean flag. Due to the changes (which are necessary to derive Corollary 1 later), a proof of correctness is in order:

Proof. Since a Büchi automaton $\hat{\mathcal{A}}$ captures an ω-regular language L if the language of $\hat{\mathcal{A}}$ has exactly the same ultimately periodic words as the ones in L, we can restrict our attention to those.

First proof direction: Let uv^ω be an ultimately periodic word in L. Then, by Theorem 1, we have that $u\$v \in \mathcal{L}(\mathcal{A}')$. By Proposition 1, we have that $u \in A_i$ and $v \in \mathcal{L}(B_i)$ for some $(A_i, B_i) \in P$. We can now construct an accepting lasso for uv^ω in $\hat{\mathcal{A}}$. Since $u = u_1 \ldots u_k \in \mathcal{L}(A_i)$, there exits a accepting run $\pi_A = \pi_0 \ldots \pi_k$ for u in A_i.

By the construction of $\hat{\mathcal{A}}$, there exists a prefix run $\pi_0 \ldots \pi_{k-1}$ for the same word in $\hat{\mathcal{A}}$. Furthermore, if $k > 0$, then there exist a transition $(\pi_{k-1}, u_k, (q, \mathbf{true}))$ for some $q \in Q_{0,i}^B$. If u is the empty word, then $(q, \mathbf{true})$ is an initial state. Hence, in both cases every state in $Q_{0,i}^B \times \{\mathbf{true}\}$ is reached by some run in $\hat{\mathcal{A}}$ after reading u.

Since $v \in \mathcal{L}(B_i)$, there exists a run $\pi'_0 \ldots \pi'_r$ with $\pi'_0 \in Q_{0,i}^B$ and $\pi'_r \in F_i^B$ for $v = v_1 \ldots v_r$. By the construction of $\hat{\mathcal{A}}$, the (prefix) run $\pi'_0 \ldots \pi'_{r-1}$ exists in $\hat{\mathcal{A}}$ as well, except that every state element is labeled by whether the last visited state is accepting, where the label for π'_0 can also be $\mathbf{true}$. From the last such state (π'_{r-1}, b) for some $b \in \mathbb{B}$, the construction of $\hat{\mathcal{A}}$ furthermore ensures that $((\pi'_{r-1}, b), v_r, (\pi'_0, \mathbf{true})) \in \hat{\delta}$. This closes a cycle in $\hat{\mathcal{A}}$. Since the cycle can be repeated indefinitely long when reading v^ω and it contains at least one accepting state, namely $(\pi'_0, \mathbf{true})$, we constructed an accepting infinite run in $\hat{\mathcal{A}}$ for uv^ω, proving its acceptance.

Second proof direction: Let $w_1 w_2 \ldots = uv^\omega$ be a word accepted by $\hat{\mathcal{A}}$. Since $\hat{\mathcal{A}}$ has a finite number of states, there exists an accepting run of the shape $\pi = \pi_0 \ldots \pi_{k-1}(\pi_k \ldots \pi_{r-1})^\omega$ for it such that at least one state in $\hat{F}$ occurs in $\pi_k \ldots \pi_{r-1}$. Without loss of generality, we can also assume that $\pi_k \in \hat{F}$, as the prefix of the lasso can always be extended slightly to rotate the cycle.

Due to the construction of $\hat{\mathcal{A}}$, the complete run π takes place in a part of $\hat{\mathcal{A}}$ generated by one element (A_i, B_i) in P (as there are no transitions between these parts). Let s be the index in π at which π reaches the states in $Q_i^B \times \mathbb{B}$ for the first time. We have that $w_1 \ldots w_s$ is a word in the language of A_i by the fact that such a switch is only possible in $\hat{\mathcal{A}}$ after reading a word in $\mathcal{L}(A_i)$. By the fact that π_k is an accepting state and the construction of $\hat{\mathcal{A}}$, we have that $w_{s+1} \ldots w_k$ is a word in $\mathcal{L}(B_i)$ (if non-empty), as $\pi_s \ldots \pi_k$ simulates the Q_i^B component of a run from an initial state in B_i to an accepting state, except that π_k is replaced by a state in $(Q_{0,i}^B, \mathbf{true})$. As such states can only be reached when reaching an accepting state for $w_{s+1} \ldots w_k$ in B_i, it follows that $w_{s+1} \ldots w_k \in \mathcal{L}(B_i)$.

For the same reason and since π is an accepting lasso, we have that the word $w_{k+1} \ldots w_r$ is accepted by B_i as well. Since π is a lasso for the word w, this actually means that $w_{r+j\cdot(r-k)+1} \ldots w_{r+(j+1)\cdot(r-k)}$ is the same word for every $j \in \mathbb{N}$. This observation allows us to overall decompose w as follows:

$$w = \underbrace{w_0 w_1 \ldots w_s}_{\in \mathcal{L}(A_i)} \; \underbrace{w_{s+1} \ldots w_k}_{\in \mathcal{L}(B_i) \text{ if not } \epsilon} \; \underbrace{w_{k+1} \ldots w_r}_{\in \mathcal{L}(B_i)} \underbrace{w_{r+1} \ldots w_{2r-k}}_{\in \mathcal{L}(B_i)} \cdots$$

This proves that $w \in L$ by Proposition 1. □

Corollary 1. *Let $\hat{\mathcal{A}}$ be a Büchi automaton built from some DFA $\mathcal{A}' = (Q', \Sigma \cup \{\$\}, \delta', Q'_0, F')$ that accepts exactly the words $u\$v$ with $u, v \in \Sigma^*$ for which uv^ω is a word in some ω-regular language L using the construction from Lemma 1.*

We have that $\hat{\mathcal{A}}$ is a tight automaton.

Proof. The first direction of the proof of the preceding lemma proves the existence of an accepting lasso of length $|u| + |v| - 1$ in $\hat{\mathcal{A}}$ for every word uv^ω in L. □

By applying the constructions from Theorem 1 and Lemma 1 in succession, we can thus obtain a tight Büchi automaton from an arbitrary Büchi automaton. Since the construction from Lemma 1 has only a polynomial blow-up in the automaton size, we obtain an overall size exponential in the size of the original Büchi automaton. Since Kupferman and Vardi [14] gave an exponential lower bound for the safety case, this construction is complexity-theoretically optimal on a large scale. On a finer scale, our approach yields automata of size $2^{O(n^2)}$, while the lower bound by Kupferman and Vardi's for the safety case is only $O(2^n)$, leaving a small gap to be filled in future work.

4 Component Lassos – Negative Result

While we have seen above that in general, tight Büchi automata need to be exponentially larger than (the smallest) equivalent arbitrary Büchi automata, this does not automatically mean that finding shortest counter-example lassos for arbitrary specification Büchi automata is not possible in polynomial time.

In previous work [6], we showed that the smallest value of $|u| + |v|$ for some ultimately periodic word uv^ω in the language of a Büchi automata is NP-hard to approximate within any polynomial approximation function p. Under the assumption that NP≠P, this means that no polynomial-time algorithm exists that given a non-deterministic Büchi automaton always outputs a value between $|u|+|v|$ and $p(|u|+|v|)$ for some ultimately periodic word uv^ω in the language of the automaton with minimal $|u|+|v|$. The result does not apply to the problem of finding shortest counter-example lassos as it adds the requirement that some state is reached twice after $|u| + |v|$ steps. Even if that problem is still NP-complete, its approximation hardness could be lower, which would be useful for the practical application of model checking. We show that, unfortunately, this is not the case. The following proof transfers the main ideas from [6] to the *component lasso* setting.

Proposition 2. *Approximating the length of the shortest lasso of a finite-state machine $\mathcal{F}$ that is accepted by a non-deterministic Büchi automaton $\mathcal{A}$ within any polynomial approximation function p is NP-hard.*

Proof. We reduce the NP-hard *satisfiability* problem to the (approximation) problem at hand. Let p be a polynomial function and ψ be a satisfiability problem in conjunctive normal form over the set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$, where we assume that every solution has $v_1 =$ **false**. The satisfiability is still NP-hard under this restriction, as it is easy to extend a SAT instance by one variable and to add a clause that requires the additional variable to have a **false** value.

We build a Büchi automaton over the language $\Sigma = \{\mathbf{false}, \mathbf{true}\}$ that implements the following language:

$$L = \bigodot_{c \in \psi} \underbrace{\Bigg(\bigcup_{x_1,\ldots,x_n \in \mathbb{B}^n, (v_1 = x_1, \ldots, v_n = x_n) \models c} (x_1 \ldots x_n) \Bigg)}_{enc_c}{}^{p(n)+1} \cdot \Sigma^\omega$$

In this equation, the $\bigodot$ operator refers to taking the concatenation of the elements in its scope. Here, the operator ranges over all clauses in the SAT instance ψ, where the order does not matter for the scope of this proof. Note that while the union operator in the equation ranges over a set of size exponential in n, a Büchi automaton part for one element enc_c is of size at most $2n$, as we demonstrate in Figure 1. We furthermore consider a finite-state machine $\mathcal{F}$ with $2n - 1$ states of the shape given in Figure 2. To prove the approximation hardness result, we show that:

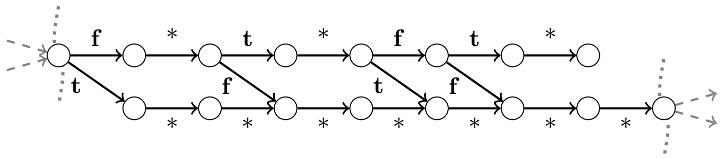

Fig. 1. Example automaton part for $n = 8$ and a clause $c_1 \vee \neg c_3 \vee c_5 \vee \neg c_6$, where **f** is an abbreviation for **false**, **t** is an abbreviation for **true**, and $*$ captures both characters.

1. the Büchi automaton for L can be built in time polynomial in the number of clauses in ψ, n, and $p(n)$ and is of size polynomial in ψ, n, and $p(n)$;
2. if ψ has a solution, then there exists a counter-example lasso in $\mathcal{F}$ of size n;
3. if there exists a counter-example lasso of size at most $p(n)$, then we can obtain a solution to ψ from the lasso.

1) Note that L can be represented by concatenating $|\psi| \cdot (p(n) + 1)$ many automaton parts with $2n$ states each (as shown in Fig. 1). The final Σ^ω component needs a single accepting state. Overall, the number of states of the resulting automaton is $(p(n) + 1) \cdot 2n \cdot |\psi| + 1$.

2) Let $x_1, \ldots, x_n$ be a solution to ψ (where $x_1 =$ **false** by the assumption above). The word $(x_1, \ldots, x_n)^\omega$ induces a counter-example lasso starting in the initial state of the FSM, proceeding to the states labeled by $x_2, \ldots, x_n$, and looping back to the initial state afterwards. Since we have $x_1, \ldots, x_n \models \psi$, we know that $x_1, \ldots, x_n$ is a word in enc_c for every $c \in \psi$. Hence, $(x_1, \ldots, x_n)^\omega$ is accepted by $\bigodot_{c \in \psi}(enc_c)^m$ for any $m \in \mathbb{N}$ and by the subsequent Σ^ω component of L.

3) Let $w = uv^\omega$ be a counter-example lasso of size at most $p(n)$. Note that $|v|$ needs to be a multiple of n for the lasso to be correct. We rewrite w slightly to $u'v'^\omega$ by unrolling the lasso until the length of u' is also a multiple of n (with $|v'| = |v|$). Let $x_1 \ldots x_n$ be the first n characters of v'. Since u' and v' are of sizes that are multiples of n, for every c, we have that v' needs to be accepted by enc_c. This is because as every enc_c is repeated $p(n)+1$ times, v' is not long enough to have $x_1 \ldots x_n$ miss all $p(n) + 1$ repetitions. Due to the construction of enc_c, we have that $x_1 \ldots x_n$ is a model of the clause c. Since this line of reasoning holds for all clauses c, we know that $x_1, \ldots, x_n$ is a model of the whole formula ψ. $\square$

Note that the automaton built in Proposition 2 is actually deterministic, hence showing the hardness of the problem even for deterministic Büchi automata.

5 Component Lassos – Positive Result

Now that we know that finding shortest counter-example lassos for general ω-regular properties is computationally difficult (even for any reasonable approximation version of the problem), the question arises whether there are at least some easy classes of properties and/or finite-state machines. While we have seen in the previous section that the determinism of a specification automaton does not change the computational complexity, other properties of the specification

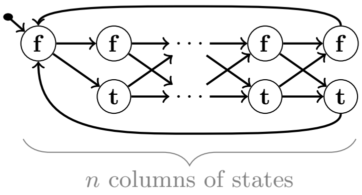

Fig. 2. Finite-state machine shape for the proof of Proposition 2, where **f** is an abbreviation for **false** and **t** is an abbreviation for **true**.

automaton can be used to derive a more detailed characterization of the complexity of finding shortest counter-example lassos.

An interesting structural property of specification automata is the *maximal size of the strongly connected components* (SCCs) in the automata. For instance, if all SCCs of a specification automaton have a size of at most 1, we call such an automaton *very-weak* or *one-weak*. Intuitively, this means that all loops in the automaton are self-loops. This subset of the set of non-deterministic Büchi automata has been identified to characterize the set of properties of reactive systems whose complements are representable by both LTL and computation tree logic with only universal path quantifiers (ACTL), where in the LTL case the formula is checked along all executions of the system [15]. Very-weak automata have been used to derive heuristically shorter counter-example lassos [1] with the model checker `spin`. We extend this previous positive result on a more fundamental level by giving a polynomial-time algorithm to find guaranteed shortest *counter-example lasso cycles* for specification automata with a fixed upper bound on the size of the SCCs. We hence no longer require the SCCs to have a size of at most 1 and give an algorithm that is guaranteed to find shortest counter-examples.

The main idea of the following construction is that if an SCC is small enough, we can keep track of multiple runs of the specification automaton from *all* SCC states in parallel while traversing a lasso cycle in the FSM. When the lasso cycle in the FSM is closed, we then check if transitions of the specification automaton can be stitched together to form an accepting cycle for the lasso cycle of the FSM. Doing so requires time exponential in the number states in an SCC. Switches between SCCs do not have to be taken into consideration here due to the fact that we are only interested in minimizing the length of lasso cycles. We start by defining a graph that is suitable for searching for shortest component lassos.

Definition 1. *Let $\mathcal{A} = (Q, \Sigma, \delta_\mathcal{A}, Q_0, F)$ be a Büchi automaton and $\mathcal{F} = (S, \Sigma, \delta_\mathcal{F}, s_0, L)$ be a finite-state machine. Let $\mathcal{S}_1, \ldots, \mathcal{S}_m$ be the state sets of the (maximal) strongly connected components of $\mathcal{A}$ that contain at least one accepting state each.*

For each $1 \leq k \leq m$, we define the lasso-searching graph *of $\mathcal{S}_k$ and $\mathcal{F}$ as a tuple $(V, E_0 \cup E_1)$ with the set of vertices V, the* normal *edges E_0, and the* closing edges *E_1. These are defined as:*

$$V = S \times \mathcal{S}_k \times \{\mathcal{S}_k \to (\mathcal{S}_k \times \mathbb{B} \cup \bot)\}$$

$$\begin{aligned}E_0 = \{((s,\bar{q},f),(s',\bar{q},f') \in V \times V \mid{}& \exists x \in \Sigma : (s,x,s') \in \delta_{\mathcal{F}}\\ &\wedge \forall q \in \mathcal{S}_k : f'(q) = \bot \vee \exists q', q'' \in \mathcal{S}_k, b \in \mathbb{B} : f(q) = (q', b)\\ &\wedge (q', x, q'') \in \delta_{\mathcal{A}} \wedge f'(q) = (q'', b \vee (q'' \in F))\}\\ E_1 = \{(s,\bar{q},f),(s,\bar{q},f) \in V \times V :{}& \exists q_1, \ldots, q_l \in \mathcal{S}_k : q_1 = \bar{q},\\ &\forall 2 \leq i \leq l, f(q_{i-1})|_Q = q_i, f(q_n)|_Q = q_1,\\ &\exists 1 \leq j \leq l, f(q_j)|_{\mathbb{B}} = \mathbf{true}\}\end{aligned}$$

Note that the SCC decomposition of an automaton can be computed in time linear in its number of states and transitions [20]. Hence, $\mathcal{S}_1, \ldots, \mathcal{S}_m$ can be easily obtained and for every strongly connected component $\mathcal{S}_k$, the graph $(V, E_0 \cup E_1)$ can be built in time polynomial in the sizes of $\mathcal{A}$ and $\mathcal{F}$ and exponential in $|\mathcal{S}_k|$. The first component of a state $(s, \bar{q}, f)$ tracks the state in the FSM, and the second one denotes an *anchor* state of the specification automaton, which never changes along graph edges. The third component keeps track of from which SCC state which other state can be reached for the labels along the cycle part in $\mathcal{F}$ traversed so far.

Lemma 2. *If and only if some state $(s, \bar{q}, f)$ is reachable from itself in $n+1 \in \mathbb{N}$ steps using first only edges from E_0 and then closing the cycle with an edge in E_1, there exists an accepting lasso cycle from FSM state s for the specification automaton $\mathcal{A}$ of length n, using $\bar{q}$ as the first state of the lasso cycle.*

Proof. **⇐:** Let $s_1 \ldots s_n$ be some counter-example lasso cycle for some state $q \in \mathcal{A}$ with the labels $x_1, \ldots, x_n$ along the cycle. In this case, there exists an accepting cycle $q_1 q_2 \ldots \in Q^\omega$ of the specification automaton for the same (suffix) trace of the system. Without loss of generality, we can assume that this cycle is ultimately periodic and that when the same state occurs for the second time at a position that is a multiple of n, the cycle is closed. Let us consider the pairs $(q_{nk+1}, q_{n(k+1)})$. By the assumption, all such pairs have distinct left elements, and let their number be $l \in \mathbb{N}$. Let $q^j_1, \ldots, q^j_{n+1}$ be the states of the cycle in $\mathcal{A}$ (for $1 \leq j \leq l$) between the state pairs, where for all $1 \leq j \leq l$, we have $q^j_1 = q^j_{n+1}$.

We show that there is a loop in $(V, E_0 \cup E_1)$ from state $v = (s_1, q_1, f)$ with $f(q^j_1) = (q^j_1, b)$ for some $b \in \mathbb{N}$ for all states q^j_1 for $1 \leq j \leq l$, and $f(q') = \bot$ for all other states q'. We can obtain this loop by successively transitioning, for each step $1 \leq i \leq n+1$, to state $v_i = (s_j, q, f_i)$ for $f_i(q^j_i) = (q^j_i, b)$ for some $b \in \mathbb{N}$ for all states q^j_i for $1 \leq j \leq l$, and $f_i(q') = \bot$ for all other states $q' \in \mathcal{S}_k$. By the assumption that $q^j_1, \ldots, q^j_{n+1}$ is a valid transition sequence in $\mathcal{A}$ for $x_1, \ldots, x_n$, the construction of $(V, E_0 \cup E_1)$ includes these edges. Furthermore, by the assumption that the lasso is accepted by $\mathcal{A}$, along one of these l sequences, an accepting state is visited. By the construction of E_0, this means that one of the Boolean values encoded by f_n has a **true** value. Since the parts $q^j_1, \ldots, q^j_n$ (for $1 \leq j \leq l$) can be stitched together to form an accepting cycle, the definition of E_1 ensures that a suitable closing edge exists.

⇒: Let $v_1, \ldots, v_{n+1}$ be a path in $(V, E_0 \cup E_1)$ ending with an edge in E_1. We know from the construction of the graph that (1) the FSM loops under the label sequence $x_1, \ldots, x_{n-1}$ used for deriving the cycle, and (2) for every $q \in \mathcal{S}_k$ and $v_i = (s_i, \bar{q}, f_i)$ (for $1 \leq i \leq n+1$) with $f_i(q) \neq \bot$, we have that there exists a transition sequence between $f_1(q)|_Q$ and $f_n(q)|_Q$ for $x_1, \ldots, x_n$. Furthermore, $f_n(q)|_\mathbb{B}$ is **true** if and only if along the way, an accepting state is visited. This allows us to construct an accepting lasso cycle for $x_1, \ldots, x_{n-1}$ by taking $f_1(\bar{q})|_Q, f_2(\bar{q})|_Q, \ldots, f_n(\bar{q})|_Q, f_1(f_n(\bar{q})|_Q)|_Q, f_2(f_n(\bar{q})|_Q)|_Q, \ldots, f_n(f_n(\bar{q})|_Q)|_Q, \ldots$ until the cycle is closed after a multiple of n states. Since the closing edge from E_1 can only be taken if the lasso cycle is closed and one Boolean flag has a **true** value, this part of the proof follows. $\square$

Since finding shortest paths in a graph is computationally easy by performing a breadth-first search, and adapting breadth-first search to use E_1 as final transitions (back to the initial state) during the search is also simple, we can iterate over all states in $(V, E_0 \cup E_1)$ and search for shortest loops back the same state in time polynomial in $|E_0 \cup E_1|$. This allows us to derive the following corollary:

Corollary 2. *For every fixed $c \in \mathbb{N}$, we can label every state in $S \times Q$ by the shortest lasso cycle length of a counter-example lasso in time polynomial in the sizes of $\mathcal{A}$ and $\mathcal{F}$ if all SCCs of $\mathcal{A}$ have sizes of at most c.*

Proof. For every SCC $\mathcal{S}_k$, we build the graph defined above, label every state by the shortest lasso, and then take $\min_{f \in Q \to Q \times \mathbb{B} \cup \bot}(s, \bar{q}, f)$ as the length of the shortest counter-example lasso cycle from $(s, \bar{q})$. $\square$

As a final step of our construction, we need to compute which lasso cycles are actually reachable from (q_0, s_0) for some $q_0 \in Q_0$. By performing a depth-first search in the classical product automaton, we can identify those states (q, s) that are reachable and then select one with a shortest cycle. The actual counter-example lasso can be obtained by taking the path in the product automaton up to the selected state (q, s) as the lasso handle, and taking the FSM state component of a cycle in the graph $(V, E_0 \cup E_1)$ that witnesses the length of the shortest counter-example lasso cycle as lasso cycle. Taking all parts of the construction together, we obtain:

Corollary 3. *Given a FSM $\mathcal{F}$ and a specification Büchi automaton $\mathcal{A}$ in which every strongly connected component has at most $c \in \mathbb{N}$ states, we can compute a counter-example lasso for $\mathcal{F}$ and $\mathcal{A}$ that minimizes the cycle length in time polynomial in $|\mathcal{F}| \cdot |\mathcal{A}|$ and exponential in c.*

Note that the overall construction can be computationally streamlined. For instance, a search for a shortest lasso cycle can also keep the anchor state $\bar{q}$ implicit, reducing the size of the graph. For the simplicity of presenting the main idea of our approach, we did not apply such improvements here.

Note that the approach cannot be generalized to minimize the combined length of a counter-example lasso. The automata built from satisfiability problem instances in the hardness proof of Section 4 only have a single strongly connected component each, and these components only have a single state each. Hence, finding counter-example lassos with a minimal combined length is hard even the in the case of very-weak Büchi automata.

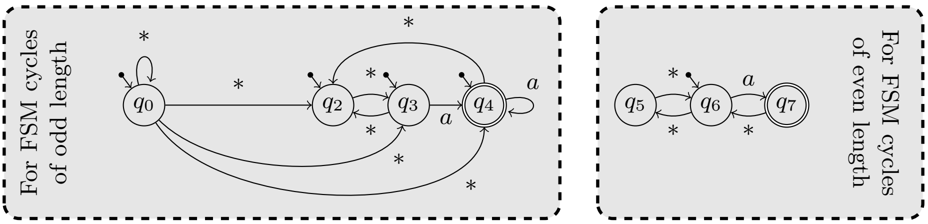

Fig. 3. A tight automaton for the alphabet $\Sigma = \{a, b\}$ and the specification that at infinitely many even positions, the letter in a word is a.

6 Conclusion

In this paper, we revisited the problem of obtaining shortest counter-example lassos for ω-regular specifications. Interestingly, it was open before this paper whether finding (approximately) shortest counter-example lassos is NP-hard or not for specifications given as non-deterministic Büchi automata. Our main result is negative: even approximating the (combined) length of the shortest counter-example lasso is NP-hard within any reasonable approximation function, let alone approximation factor. This is unfortunate, as approximate shortest counter-example lassos could be interesting for the model checking practitioner.

On the positive side, we showed how by using two existing automaton translations, we can make an arbitrary non-deterministic Büchi automaton tight, which enables the use of approaches for finding shortest accepting lassos in product automata to also find shortest counter-example lassos. Furthermore, we looked at the *parameterized complexity* of finding shortest counter-example lasso cycles and showed that for specification automata with small strongly connected components (SCCs), finding counter-example lassos with shortest cycles is possible in polynomial time for arbitrary (but fixed) SCC size limits. This result is interesting for the case of tracking down *starvation bugs* in models – in such situations, the focus is often on how the system can stall without making progress rather than how the system to be checked can reach such a situation. If the corresponding liveness property has a Büchi automaton with small SCCs, our construction is applicable.

While making Büchi automata tight leads to an exponential blow-up in their sizes, the problem could be mitigated by *minimizing* the resulting specification automaton before using it in a model checker. This problem cannot be tackled with previous simulation-based minimization approaches as we have to take care that loops with an accepting state are only removed if there are other loops of the same length that together accept the lasso cycles captured by the removed loop. We visualize this observation in Figure 3, which shows a tight automaton that is only tight because it has different SCCs for lasso cycles with even and odd lengths. Suitable minimization algorithms for tight automata that retain such redundancies in the specification automata still have to be developed, and we leave this challenge to future work.

References

1. Adabala, K., Ehlers, R.: A fragment of linear temporal logic for universal very weak automata. In: Automated Technology for Verification and Analysis - 16th International Symposium (ATVA). Lecture Notes in Computer Science, vol. 11138, pp. 335–351. Springer (2018)
2. Calbrix, H., Nivat, M., Podelski, A.: Ultimately periodic words of rational w-languages. In: 9th International Conference on Mathematical Foundations of Programming Semantics (MFPS). Lecture Notes in Computer Science, vol. 802, pp. 554–566. Springer (1993)
3. Clarke, E.M., Grumberg, O., McMillan, K.L., Zhao, X.: Efficient generation of counterexamples and witnesses in symbolic model checking. In: 32nd Conference on Design Automation (DAC). pp. 427–432. ACM Press (1995)
4. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: 23rd International Joint Conference on Artificial Intelligence (IJCAI). pp. 854–860. IJCAI/AAAI (2013)
5. Edelkamp, S., Sulewski, D., Barnat, J., Brim, L., Simecek, P.: Flash memory efficient LTL model checking. Sci. Comput. Program. **76**(2), 136–157 (2011)
6. Ehlers, R.: Short witnesses and accepting lassos in ω-automata. In: 4th International Conference on Language and Automata Theory and Applications (LATA). Lecture Notes in Computer Science, vol. 6031, pp. 261–272. Springer (2010)
7. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Series on Integrated Circuits and Systems, Springer (2006)
8. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM J. Comput. **34**(5), 1159–1175 (2005)
9. Farzan, A., Chen, Y., Clarke, E.M., Tsay, Y., Wang, B.: Extending automated compositional verification to the full class of omega-regular languages. In: 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 4963, pp. 2–17. Springer (2008)
10. Gastin, P., Moro, P., Zeitoun, M.: Minimization of counterexamples in SPIN. In: Model Checking Software, 11th International SPIN Workshop. Lecture Notes in Computer Science, vol. 2989, pp. 92–108. Springer (2004)
11. Groce, A., Visser, W.: What went wrong: Explaining counterexamples. In: Model Checking Software, 10th International SPIN Workshop. Lecture Notes in Computer Science, vol. 2648, pp. 121–135. Springer (2003)
12. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. **23**(5), 279–295 (May 1997)
13. Kupferman, O., Sheinvald-Faragy, S.: Finding shortest witnesses to the nonemptiness of automata on infinite words. In: 17th International Conference on Concurrency Theory (CONCUR). Lecture Notes in Computer Science, vol. 4137, pp. 492–508. Springer (2006)
14. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. In: 11th International Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 1633, pp. 172–183. Springer (1999)
15. Maidl, M.: The common fragment of CTL and LTL. In: 41st Annual Symposium on Foundations of Computer Science (FOCS). pp. 643–652 (2000)
16. Schuppan, V., Biere, A.: Efficient reduction of finite state model checking to reachability analysis. STTT **5**(2-3), 185–204 (2004)

17. Schuppan, V., Biere, A.: Shortest counterexamples for symbolic model checking of LTL with past. In: 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 3440, pp. 493–509. Springer (2005)
18. Schwoon, S., Esparza, J.: A note on on-the-fly verification algorithms. In: 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 3440, pp. 174–190. Springer (2005)
19. Sebastiani, R., Tonetta, S.: "more deterministic" vs. "smaller" Büchi automata for efficient LTL model checking. In: Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference (CHARME). pp. 126–140 (2003)
20. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)