Optimal Proofs for Linear Temporal Logic on Lasso Words

David Basin, Bhargav Nagaraja Bhatt, and Dmitriy Traytel

Institute of Information Security, Department of Computer Science, ETH Zürich, Switzerland

Abstract. Counterexamples produced by model checkers can be hard to grasp. Often it is not even evident why a trace violates a specification. We show how to provide easy-to-check evidence for the violation of a linear temporal logic (LTL) formula on a lasso word, based on a novel sound and complete proof system for LTL on lasso words. Valid proof trees in our proof system follow the syntactic structure of the formula and provide insight on why each Boolean or temporal operator is violated or satisfied. We introduce the notion of optimal proofs with respect to a user-specified preference order and identify sufficient conditions for efficiently computing optimal proofs. We design and evaluate an algorithm that performs this computation, demonstrating that it can produce optimal proofs for complex formulas in under a second.

1 Introduction

Model checking is a successful formal verification technique. Designing an error-free system using a model checker follows the cycle: 1) model the system and formulate the specification it should adhere to, 2) run the model checker, 3) if the tool finds an error, determine whether the error is in the model or the specification, and 4) go back to Step 1 and change the model or the specification accordingly. Our focus in this paper is on Step 3, which can be extremely challenging and time-consuming. To succeed there, the user must understand the interaction between the model, the specification, and the counterexample. Many prior approaches focus on explaining the interaction between the model and the counterexample [3, 12–14], while neglecting the specification.

In many cases, the interaction between the specification and the counterexample is non-trivial to understand. Hence the question "Why does this counterexample violate this specification?" may be a hard one. Our work focuses on this question in the context of model checking properties expressed in linear temporal logic with past and future (LTL), where counterexamples are finite words or infinite but ultimately periodic words (also called *lasso* words). In this paper, we restrict our attention to lasso word counterexamples, although it is possible to transfer most of our results to finite words.

Given our restriction, we refine the question that we investigate to "Why does the counterexample uv^ω violate the LTL formula φ?" This question is independent of the specific model checking technique used to find the counterexample. Ignoring the "Why" in the question, we obtain the *LTL path-checking* problem [21], a core decision problem in for runtime verification. A decision procedure for this problem, namely a path-checking algorithm, computes a Boolean result which provides no insight into why the formula is violated. However, the algorithm itself knows why the formula is violated.

To expose the internal knowledge of an LTL path-checking algorithm, we must fix a suitable representation of this information. Our approach is to devise a *proof system*

for LTL on lasso words, where proof search amounts to solving LTL path checking. Then, *proof trees* (or just *proofs*) in this setting capture the knowledge of the proof search algorithm and are the data structure that we output to explain violations. To be understandable, the proof system's rules must be as simple and as close to the standard semantics of LTL as possible. In particular, they should *not* be tainted with algorithmic details such as LTL's unrolling equations used in many path-checking algorithms.

Typically there are multiple (often infinitely many) different proof trees for a given formula and lasso word. Each proof tree represents a different way to explain the violation. Deciding which proof among the set of all valid proofs helps the user best understand the violation depends on the application scenario and the user. If the user's objective is to identify the most severe violation given an ordering of severity on atomic events, then he or she may be interested in proofs that focus on the particular events. For example, consider the formula $\Box(\mathit{PipeSealed} \wedge \mathit{LightsOn})$, stating that the pipe is always sealed and the light is always turned on. When there are different violations, we might prefer learning about the more severe cases of pipe leakage than about the lights being switched off. In addition, it is useful to learn about the earliest point in time when the pipe started to leak. The proofs in our proof system can represent this information. Moreover, it might be preferable to give the user a concise proof. To flexibly handle different scenarios, we allow the user to specify a *preference order* on proofs. For preference orders that satisfy some monotonicity conditions, we devise a proof search algorithm that computes an optimal proof with respect to the order.

In summary, we make the following contributions. We describe a sound and complete proof system for LTL on lasso words, where proof search amounts to path-checking (§3). We define a notion of optimal proofs in our proof system with respect to a user-specified preference order. Since computing optimal proofs can be intractable for arbitrary orders, we identify sufficient conditions on the preference order for which an optimal proof can be efficiently computed (§4) and we use these conditions to devise an algorithm that computes an optimal proof (§5). Taken together, these contributions provide a new approach to explaining counterexamples generated by LTL model checkers. Finally, we evaluate and demonstrate the effectiveness of a prototype implementation of our algorithm on realistic examples (§6). We postpone the discussion of related work until after the presentation of our technical contributions (§7).

2 Linear Temporal Logic

We briefly recapitulate the syntax and semantics of *linear temporal logic* (*LTL*). The set of LTL formulas over a set of atomic propositions P is defined inductively:

$$\varphi = p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathcal{S}\, \varphi \mid \varphi \,\mathcal{U}\, \varphi,$$

where $p \in P$. Along with the standard Boolean operators, LTL includes the temporal operator $\mathcal{S}$ (*since*) and $\mathcal{U}$ (*until*), which may be nested freely. For simplicity, we omit *next* and *previous*; however, all results in this paper can be easily extended to accommodate them. LTL formulas are evaluated on words over the alphabet $\Sigma = 2^P$. A word is an infinite sequence $\rho = \rho(0), \rho(1), \rho(2), \ldots$, where each $\rho(i) \in 2^P$. Whether an LTL formula is satisfied at time-point i for a fixed word ρ is defined inductively as follows.

$$\frac{a \in \rho(i)}{i \vdash^+ a}\ ap^+ \qquad \frac{i \vdash^- \varphi}{i \vdash^+ \neg\varphi}\ \neg^+ \qquad \frac{a \notin \rho(i)}{i \vdash^- a}\ ap^- \qquad \frac{i \vdash^+ \varphi}{i \vdash^- \neg\varphi}\ \neg^-$$

$$\frac{i \vdash^+ \varphi_1}{i \vdash^+ \varphi_1 \vee \varphi_2}\ \vee_L^+ \qquad \frac{i \vdash^+ \varphi_2}{i \vdash^+ \varphi_1 \vee \varphi_2}\ \vee_R^+ \qquad \frac{i \vdash^- \varphi_1 \quad i \vdash^- \varphi_2}{i \vdash^- \varphi_1 \vee \varphi_2}\ \vee^-$$

$$\frac{i \vdash^+ \varphi_1 \quad i \vdash^+ \varphi_2}{i \vdash^+ \varphi_1 \wedge \varphi_2}\ \wedge^+ \qquad \frac{i \vdash^- \varphi_1}{i \vdash^- \varphi_1 \wedge \varphi_2}\ \wedge_L^- \qquad \frac{i \vdash^- \varphi_2}{i \vdash^- \varphi_1 \wedge \varphi_2}\ \wedge_R^-$$

$$\frac{j \leq i \quad j \vdash^+ \varphi_2 \quad \forall k \in (j, i].\ k \vdash^+ \varphi_1}{i \vdash^+ \varphi_1\, \mathcal{S}\, \varphi_2}\ \mathcal{S}^+ \qquad \frac{j \leq i \quad j \vdash^- \varphi_1 \quad \forall k \in [j, i].\ k \vdash^- \varphi_2}{i \vdash^- \varphi_1\, \mathcal{S}\, \varphi_2}\ \mathcal{S}^-$$

$$\frac{j \geq i \quad j \vdash^+ \varphi_2 \quad \forall k \in [i, j).\ k \vdash^+ \varphi_1}{i \vdash^+ \varphi_1\, \mathcal{U}\, \varphi_2}\ \mathcal{U}^+ \qquad \frac{j \geq i \quad j \vdash^- \varphi_1 \quad \forall k \in [i, j].\ k \vdash^- \varphi_2}{i \vdash^- \varphi_1\, \mathcal{U}\, \varphi_2}\ \mathcal{U}^-$$

$$\frac{\forall k \in [0, i].\ k \vdash^- \varphi_2}{i \vdash^- \varphi_1\, \mathcal{S}\, \varphi_2}\ \mathcal{S}_\infty^- \qquad \frac{\forall k \in [i, max(i, |u| + h_p(\varphi_2) \times |v|) + |v|).\ k \vdash^- \varphi_2}{i \vdash^- \varphi_1\, \mathcal{U}\, \varphi_2}\ \mathcal{U}_\infty^-$$

Fig. 1. Proof system for a fixed lasso word $\rho = uv^\omega$

$$\begin{array}{llll}
i \models p & \text{iff } p \in \rho(i) & i \models \neg\varphi & \text{iff } i \not\models \varphi \\
i \models \varphi_1 \vee \varphi_2 & \text{iff } i \models \varphi_1 \text{ or } i \models \varphi_2 & i \models \varphi_1 \wedge \varphi_2 & \text{iff } i \models \varphi_1 \text{ and } i \models \varphi_2 \\
i \models \varphi_1\, \mathcal{S}\, \varphi_2 & \text{iff } j \models \varphi_2 \text{ for some } j \leq i \text{ and } k \models \varphi_1 \text{ for all } j < k \leq i & & \\
i \models \varphi_1\, \mathcal{U}\, \varphi_2 & \text{iff } j \models \varphi_2 \text{ for some } j \geq i \text{ and } k \models \varphi_1 \text{ for all } i \leq k < j & &
\end{array}$$

Note that here, and in subsequent definitions, the dependence on ρ is left implicit.

A *lasso* (*word*) has the form $\rho = uv^\omega$, where u and v are finite words over the alphabet 2^P and $|v| \neq 0$. We refer to u as the *prefix* and v as the *loop* of the lasso word ρ.

Let $\mathsf{SF}(\varphi)$ denote the set of φ's strict subformulas (i.e., excluding φ) defined as usual. We pick some well-founded total order $\sqsubset$ over $\mathsf{SF}(\varphi)$ that respects the subformula ordering: if $\varphi_1 \in \mathsf{SF}(\varphi_2)$, then $\varphi_1 \sqsubset \varphi_2$. For a formula φ, the *past height* $h_p(\varphi)$ and the *future height* $h_f(\varphi)$ are defined as the number of nested past operators and future operators in φ, respectively. The *temporal height* $h(\varphi)$ is defined as $h_p(\varphi) + h_f(\varphi)$.

3 Proof System for LTL on Lasso Words

We introduce a proof system for LTL *path checking*. Proofs in this system witness the satisfaction or violation of an LTL formula with respect to a given lasso $\rho = uv^\omega$. Although we are primarily interested in violations, in the presence of negation it is convenient to reason about satisfaction as well. Our proof system therefore consists of two mutually dependent judgments: $\vdash^+$ (satisfaction) and $\vdash^-$ (violation), and is defined as the least relation satisfying the deduction rules shown in Figure 1. The names of the satisfaction and violation judgment rules are suffixed by $^+$ and $^-$, respectively.

The satisfaction rules ap^+, $\neg^+$, $\vee_L^+$, $\vee_R^+$, $\wedge^+$, $\mathcal{U}^+$, and $\mathcal{S}^+$ directly follow the semantics of the corresponding LTL operators. For example, in the case of $\mathcal{S}^+$, the premise for $i \vdash^+ \varphi_1\, \mathcal{S}\, \varphi_2$ includes a witness time-point j such that $j \vdash^+ \varphi_2$ and a finite sequence of

satisfaction proofs of φ_1 for all $k \in (j,i]$. The violation rules for the non-temporal operators ap^-, $\neg^-$, $\vee^-$, $\wedge_L^-$, and $\wedge_R^-$ are dual. The violation rules for the temporal operators are more interesting. To arrive at $\mathcal{S}^-$ and $\mathcal{S}_\infty^-$, we negate and rewrite the semantics of $\mathcal{S}$:

$$\begin{aligned} i \not\models \varphi_1 \,\mathcal{S}\, \varphi_2 \text{ iff } & \forall j \leq i.\ j \not\models \varphi_2 \vee (\exists k \in (j,i].\ k \not\models \varphi_1) \\ \text{iff } & (\exists j \leq i.\ j \not\models \varphi_1 \wedge (\forall k \in [j,i].\ k \not\models \varphi_2)) \vee (\forall k \leq i.\ k \not\models \varphi_2) \qquad (\text{Eq.} \equiv_{\mathcal{S}}) \end{aligned}$$

The rules $\mathcal{S}^-$ and $\mathcal{S}_\infty^-$ correspond to the two disjuncts in the last right-hand side. Using this particular format for the rules (as opposed, for example, to the first right hand side, which requires deciding between two choices for all previous time-points) is a deliberate design decision. The violation proof $j \vdash^- \varphi_1$ in $\mathcal{S}^-$ allows us to disregard all the previous time-points at which φ_2 held, since these previous time-points cannot witness the satisfaction of $\varphi_1 \,\mathcal{S}\, \varphi_2$ exactly because of $j \vdash^- \varphi_1$. The second rule $\mathcal{S}_\infty^-$ covers the case when there is no such j with $j \vdash^- \varphi_1$. Then, to violate $\varphi_1 \,\mathcal{S}\, \varphi_2$ at i, the formula φ_2 must have previously always been violated.

For the future operator $\mathcal{U}$, we consider the dual semantic equation:

$$i \not\models \varphi_1 \,\mathcal{U}\, \varphi_2 \text{ iff } (\exists j \geq i.\ j \not\models \varphi_1 \wedge (\forall k \in [i,j].\ k \not\models \varphi_2)) \vee (\forall k \geq i.\ k \not\models \varphi_2) \qquad (\text{Eq.} \equiv_{\mathcal{U}})$$

The rules $\mathcal{U}^-$ and $\mathcal{U}_\infty^-$ model the two disjuncts. Yet the assumption in $\mathcal{U}_\infty^-$ appears to be much weaker than what the right disjunct demands: instead of providing infinitely many violation proofs for φ_2 for every time-point $\geq i$, the rule requires only a finite number (that depends on the past height of φ_2, $|u|$ and $|v|$) of violation proofs for φ_2. This rule's soundness follows by taking the cyclic nature of the fixed lasso word into account. To see this, we recall a theorem from Markey and Schnoebelen [21] that states that the satisfiability of φ is periodic on lasso words after a certain threshold time-point.

Lemma 1. *Fix a lasso $\rho = uv^\omega$. For all φ and $k > |u| + h_p(\varphi) \cdot |v|$, we have $k \models \varphi$ iff $k + |v| \models \varphi$.*

Proof. By structural induction on φ with an arbitrary k. □

Another induction extends this result to cover all time-points modulo $|v|$, starting from the threshold.

Corollary 1. *Fix a lasso $\rho = uv^\omega$. For all φ, $n \in \mathbb{N}$, and $k > |u| + h_p(\varphi) \cdot |v|$, we have $k \models \varphi$ iff $k + n \cdot |v| \models \varphi$.*

Proof. Induction over n with an arbitrary k using Lemma 1 in the induction step. □

Corollary 1 yields that if $k \not\models \varphi$ for $k \in [thr, thr + |v|)$ (where $thr \geq |u| + h_p(\varphi) \times |v|$), then $\forall k \geq thr.\ k \not\models \varphi$. Intuitively, if we can prove often enough that a formula does not hold, then it will never hold. From this, the soundness of $\mathcal{U}_\infty^-$ follows easily. So does the soundness and completeness of the entire proof system.

Theorem 1. *Fix a lasso $\rho = uv^\omega$. For all φ and $i \in \mathbb{N}$, we have $i \vdash^+ \varphi$ iff $i \models \varphi$ and $i \vdash^- \varphi$ iff $i \not\models \varphi$.*

Proof. ($\Longrightarrow$, *Soundness*): Mutual induction over the structure of the derivations $\vdash^+$ and $\vdash^-$ using the Equations $\equiv_{\mathcal{U}}$ and $\equiv_{\mathcal{S}}$ for the temporal operators and Corollary 1.
($\Longleftarrow$, *Completeness*): Induction over the structure of φ for an arbitrary i. □

We conclude this section with an example.

Example 1. Let $\rho = \{a,c\}(\{a,b\}\{c\})^\omega$ and $\varphi = a\,\mathcal{U}\,(b \wedge c)$. Here is a proof of $0 \not\models \varphi$:

$$\dfrac{\dfrac{\dfrac{b \notin \{a,c\}}{0 \vdash^- b}\,ap^-}{0 \vdash^- b \wedge c}\,\wedge_L^- \quad \dfrac{\dfrac{c \notin \{a,b\}}{1 \vdash^- c}\,ap^-}{1 \vdash^- b \wedge c}\,\wedge_R^- \quad \dfrac{\dfrac{b \notin \{c\}}{2 \vdash^- b}\,ap^-}{2 \vdash^- b \wedge c}\,\wedge_L^-}{0 \vdash^- a\,\mathcal{U}\,(b \wedge c)}\,\mathcal{U}_\infty^-$$

Such proofs explain why a formula is satisfied or violated on a lasso word. In this example, we immediately see that the $\mathcal{U}$ operator is violated because its second argument $b \wedge c$ is always violated. The proof provides witnesses for the violation of $b \wedge c$ at the first three time-points.

4 Proof Trees

To manipulate and compare proofs in the above system, we make them explicit. Namely, we define an inductive syntax for satisfying ($\mathfrak{sp}$) and violating ($\mathfrak{vp}$) proofs built using the rules in our proof system.

$$\begin{aligned}\mathfrak{sp} &= ap^+(\mathbb{N},\Sigma) \mid \neg^+(\mathfrak{vp}) \mid \wedge^+(\mathfrak{sp},\mathfrak{sp}) \mid \vee_L^+(\mathfrak{sp}) \mid \vee_R^+(\mathfrak{sp}) \mid \mathcal{S}^+(\mathfrak{sp},\overline{\mathfrak{sp}}) \mid \mathcal{U}^+(\mathfrak{sp},\overline{\mathfrak{sp}}) \\ \mathfrak{vp} &= ap^-(\mathbb{N},\Sigma) \mid \neg^-(\mathfrak{sp}) \mid \wedge_L^-(\mathfrak{vp}) \mid \wedge_R^-(\mathfrak{vp}) \mid \vee^-(\mathfrak{vp},\mathfrak{vp}) \mid \mathcal{S}^-(\mathfrak{vp},\overline{\mathfrak{vp}}) \mid \mathcal{U}^-(\mathfrak{vp},\overline{\mathfrak{vp}}) \\ &\quad \mid \mathcal{S}_\infty^-(\overline{\mathfrak{vp}}) \mid \mathcal{U}_\infty^-(\overline{\mathfrak{vp}})\end{aligned}$$

Here, $\overline{\mathfrak{sp}}$ and $\overline{\mathfrak{vp}}$ abbreviate finite sequences $[\mathfrak{sp},\ldots,\mathfrak{sp}]$ and $[\mathfrak{vp},\ldots,\mathfrak{vp}]$ of subproofs. The proof from Example 1 can be written as $P_1 = \mathcal{U}_\infty^-([\wedge_L^-(ap^-(0,b)), \wedge_R^-(ap^-(1,c)), \wedge_L^-(ap^-(2,b))])$ using this syntax. In the following, we treat satisfaction and violation proofs uniformly, operating on the disjoint union $\mathfrak{p} = \mathfrak{sp} \uplus \mathfrak{vp}$. For a proof $p \in \mathfrak{p}$, we define $\mathbb{V}(p)$ to be $\top$ if $p \in \mathfrak{sp}$ and $\bot$ otherwise.

Observe that the time-points are only stored in the proofs of the atomic propositions. This is sufficient to associate each proof tree p with a time-point $i(p)$. For example, $i(\mathcal{S}^+(p,[q_1,\ldots,q_n]))$ is $i(q_n)$ if $n > 0$ and $i(p)$ otherwise. If the proof additionally passes a few syntactic checks with respect to a given formula φ (such as the constructors in the proof match the constructors in the formula) and a lasso word ρ (for atomic propositions), we call it valid for φ at $i(p)$ in ρ, written $p \vdash \varphi$ (again leaving the dependence on ρ implicit). We omit the straightforward formal definitions of $i(p)$ and $p \vdash \varphi$. It is easy to see that when $p \vdash \varphi$, we have that $\mathbb{V}(p)$ implies $i(p) \vdash^+ \varphi$ and $\neg\mathbb{V}(p)$ implies $i(p) \vdash^- \varphi$.

For a time-point i and a formula φ, multiple (in fact, potentially infinitely many) valid proofs may exist. For example, two additional valid proofs for the formula and the lasso word from Example 1 at time-point 0 are $P_2 = \mathcal{U}^-(ap^-(2,a),[\wedge_L^-(ap^-(0,b)), \wedge_R^-(ap^-(1,c)), \wedge_L^-(ap^-(2,b))])$ and $P_3 = \mathcal{U}^-(ap^-(4,a),[\wedge_L^-(ap^-(0,b)), \wedge_R^-(ap^-(1,c)), \wedge_L^-(ap^-(2,b)), \wedge_R^-(ap^-(3,c)), \wedge_L^-(ap^-(4,b))])$. Let us compare the three proofs. The proof using $\mathcal{U}_\infty^-$ is smaller in size. In contrast, the proofs using $\mathcal{U}^-$ might be preferable as the $\mathcal{U}^-$ rule is very close to $\mathcal{U}$'s semantics and thus easy to understand (whereas understanding $\mathcal{U}_\infty^-$ requires understanding Lemma 1). Of the two $\mathcal{U}^-$ proofs, the shorter one is easier to digest. Still, different proofs might be preferable in different situations.

With no reasonable way to decide which proof to present to a user, we offer users a way to specify their preference using a *well-quasi-order* (*wqo*) $\preceq \subseteq \mathfrak{p} \times \mathfrak{p}$. A wqo is a

well-founded preorder: a transitive and reflexive relation $\preceq$ for which the induced strict relation $\prec$ (defined as $p \prec q \iff p \preceq q \wedge q \not\preceq p$) is well-founded.

Example 2. Let $w :: \Sigma \to \mathbb{N}$ be a *weight function* on the set Σ of atomic predicates. We define, the weighted size $|-|_w :: \mathfrak{p} \to \mathbb{N}$ of a proof tree recursively as follows:

$$
\begin{array}{ll}
|ap^+(i,a)|_w = w(a) & |ap^-(i,a)|_w = w(a) \\
|\neg^+(p)|_w = 1 + |p|_w & |\neg^-(p)|_w = 1 + |p|_w \\
|\vee^+_L(p)|_w = 1 + |p|_w & |\wedge^-_L(p)|_w = 1 + |p|_w \\
|\vee^+_R(p)|_w = 1 + |p|_w & |\wedge^-_R(p)|_w = 1 + |p|_w \\
|\wedge^+(p_1,p_2)|_w = 1 + |p_1|_w + |p_2|_w & |\vee^-(p_1,p_2)|_w = 1 + |p_1|_w + |p_2|_w \\
|\mathcal{S}^+(p,\overline{q})|_w = 1 + |p|_w + \sum_{i=1}^n |q_i|_w & |\mathcal{S}^-(p,\overline{q})|_w = 1 + |p|_w + \sum_{i=1}^n |q_i|_w \\
|\mathcal{U}^+(p,\overline{q})|_w = 1 + |p|_w + \sum_{i=1}^n |q_i|_w & |\mathcal{U}^-(p,\overline{q})|_w = 1 + |p|_w + \sum_{i=1}^n |q_i|_w \\
|\mathcal{S}^-_\infty(\overline{q})|_w = 1 + \sum_{i=1}^n |q_i|_w & |\mathcal{U}^-_\infty(\overline{q})|_w = 1 + \sum_{i=1}^n |q_i|_w
\end{array}
$$

Here, $\overline{q}$ abbreviates $[q_1,\ldots,q_n]$. The weighted size induces a total wqo on proofs by $p \preceq^w_{size} q \iff |p|_w \leq |q|_w$. If the weight function is the constant function $w(a) = 1$ for all $a \in \Sigma$, then $|p|_w$ is the number of constructors occurring in p, written $|p|$. We write $\preceq_{size}$ for the corresponding wqo. For the above proofs for the formula and lasso word from Example 1, we have $|P_1| = 7$, $|P_2| = 8$, and $|P_3| = 12$. Hence, $P_1 \preceq_{size} P_2 \preceq_{size} P_3$.

Our goal is to compute *optimal* proofs with respect to a user-supplied relation $\preceq$. We call p optimal for φ at $i(p)$ if for all valid proofs q for φ at $i(p)$, we have $q \not\prec p$. Note that at least one valid satisfaction or violation proof always exists by the completeness of our proof system and (non-unique) minimal elements of non-empty sets of proofs exist by the well-foundedness of $\prec$.

The existence of optimal proofs does not provide us with an algorithm to compute them. In general, it is impossible to compute the minimal elements of an infinite set of proofs with respect to a wqo $\preceq$. As a first step towards an algorithm, we restrict our attention to relations on which the proof constructors behave in a monotone fashion.

Definition 1. *A relation $\preceq$ is* constructor-monotone *if it satisfies:*

1. *If $i \leq j$ then $ap^+(i,a) \preceq ap^+(j,a)$ and $ap^-(i,a) \preceq ap^-(j,a)$ for any atom a.*
2. *If $p \preceq p'$ then $\neg^+(p) \preceq \neg^+(p')$, $\vee^+_R(p) \preceq \vee^+_R(p')$, $\vee^+_L(p) \preceq \vee^+_L(p')$, $\wedge^-_R(p) \preceq \wedge^-_R(p')$, and $\wedge^-_L(p) \preceq \wedge^-_L(p')$.*
3. *If $p_1 \preceq p'_1$ and $p_2 \preceq p'_2$ then $\wedge^+(p_1,p_2) \preceq \wedge^+(p'_1,p'_2)$ and $\vee^-(p_1,p_2) \preceq \vee^-(p'_1,p'_2)$.*
4. *If $m \leq n$, $p \preceq p'$, and $q_i \preceq q'_i$ for each $i \in \{1,\ldots,m\}$ then $\mathcal{S}^+(p,[q_1,\ldots,q_m]) \preceq \mathcal{S}^+(p',[q'_1,\ldots,q'_n])$, $\mathcal{S}^-(p,[q_1,\ldots,q_m]) \preceq \mathcal{S}^-(p',[q'_1,\ldots,q'_n])$, and $\mathcal{S}^-_\infty([q_1,\ldots,q_m]) \preceq \mathcal{S}^-_\infty([q'_1,\ldots,q'_n])$.*
5. *If $m \leq n$, $p \preceq p'$, and $q_i \preceq q'_i$ for each $i \in \{m,\ldots,n\}$ then $\mathcal{U}^+(p,[q_m,\ldots,q_n]) \preceq \mathcal{U}^+(p',[q'_1,\ldots,q'_n])$, $\mathcal{U}^-(p,[q_m,\ldots,q_n]) \preceq \mathcal{U}^-(p',[q'_1,\ldots,q'_n])$, and $\mathcal{U}^-_\infty([q_m,\ldots,q_n]) \preceq \mathcal{U}^-_\infty([q'_1,\ldots,q'_n])$.*

Lemma 1 only guarantees the equisatisfiability of a formula at time-points k and $k+|v|$, for k suitably large. Constructor-monotonicity can be used to significantly generalize this result to a proof comparison with respect to $\preceq$ at those time-points as follows.

Theorem 2. *Given a lasso $\rho = uv^\omega$, an LTL formula φ, and a constructor-monotone relation $\preceq$, if $k > |u| + h_p(\varphi) \cdot |v|$ then for all valid proofs q of φ with $i(q) = k + |v|$ there exists a valid proof p of φ with $i(p) = k$ and $p \preceq q$.*

Proof. Proof by induction on structure of φ with an arbitrary k.

- Case $\varphi = a$ directly follows from the constructor-monotonicity condition 1.
- Cases $\varphi = \neg\psi$, $\varphi = \varphi_1 \wedge \varphi_2$, and $\varphi = \varphi_1 \vee \varphi_2$ directly follow from the constructor-monotonicity conditions 2 and 3 and the induction hypotheses.
- Case $\varphi = \varphi_1 \,\mathcal{S}\, \varphi_2$. Let $thr(\varphi) = |u| + h_p(\varphi) \cdot |v|$. Suppose $k + |v| \models \varphi$. Let $P = \mathcal{S}^+(p, [q_{j+1}, \ldots, q_{k+|v|}])$ be some valid proof of φ at $k + |v|$, i.e., there exists a $j \leq k + |v|$ such that p as a valid proof of φ_2 at j and the q_i are valid proofs for φ_1 at i for $i \in j+1, \ldots, k+|v|$. If $j \leq k$ then $P' = \mathcal{S}^+(p, [q_{j+1}, \ldots, q_k])$ is a valid proof for φ at k. Moreover, $P' \preceq P$ by the constructor-monotonicity condition 4. (Here, we need the condition to apply to finite sequences of different length.) Otherwise if $k < j$, then we have $j > k > thr(\varphi) >= thr(\varphi_2) + |v|$ and thus $j - |v| > thr(\varphi_2)$. Thus we can use the induction hypothesis on φ_2 (and p at $j - |v|$) and on φ_1 (and each of the q_i for $i \in \{j+1, \ldots, k+|v|\}$ at $i - |v|$) to obtain valid proofs p' of φ_2 at $j - |v|$ and q'_i for φ_1 at $i - |v|$ such that $p' \preceq p$ and $q'_i \preceq q_i$ for each $i \in \{j+1, \ldots, k+|v|\}$. Then $P' = \mathcal{S}^+(p', [q'_{j+1}, \ldots, q'_k])$ is a valid proof of φ at k and moreover by the constructor-monotonicity condition 4 we have $P' \preceq P$.
  Similarly for the case $k + |v| \not\models \varphi$, any proof P for φ at $k + |v|$ must either have the form $\mathcal{S}^-(p, [q_j, \ldots, q_{k+|v|}])$ or $\mathcal{S}^-_\infty([q_0, \ldots, q_{k+|v|}])$. In the former case, the reasoning to obtain a proof $P' \preceq P$ of φ at k is very similar to the case where $k + |v| \models \varphi$. In the latter case, it suffices to take $P' = \mathcal{S}^-_\infty([q_0, \ldots, q_k])$ to obtain $P' \preceq P$ by the constructor-monotonicity condition 4.
- Case $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$ is similar to the $\varphi_1 \,\mathcal{S}\, \varphi_2$ case. For a valid proof P of φ at $k + |v|$, all immediate subproofs of P are proofs at future time-points, i.e., at least $k + |v|$. Thus the induction hypothesis is immediately applicable (unlike in the $\mathcal{S}$ case, where a case distinction was required). Thus, we obtain a valid proof P' of φ at k that has exactly the same structure as P, in particular regarding the lengths of the finite sequences of immediate subproofs in P and P'. Using the constructor-monotonicity condition 5, we can conclude $P' \preceq P$. □

Theorem 2 allows us to stop the search for an optimal proof after a finite number of time-points. Thus, we could in principle compute a finite set of candidate proofs that is guaranteed to contain an optimal one and select a minimal element from this set. Such an algorithm would not be very efficient, as the set of candidate proofs might become extremely large. Instead, we give an algorithm that selects minimal elements eagerly and lifts optimal proofs of temporal connectives from time-points $i-1$ (for $\mathcal{S}$) and $i+1$ (for $\mathcal{U}$) to a proof at i. We first define the operator $+\!\!+$ that performs this lifting by combining subproofs of temporal formulas. We thereby abbreviate $[q_1, \ldots, q_n]$ by $\overline{q}$.

$$\begin{array}{cc} \mathcal{S}^+(p, \overline{q}) +\!\!+ r = \mathcal{S}^+(p, [q_1, \ldots, q_n, r]) & \mathcal{U}^+(p, \overline{q}) +\!\!+ r = \mathcal{U}^+(p, [r, q_1, \ldots, q_n]) \\ \mathcal{S}^-(p, \overline{q}) +\!\!+ r = \mathcal{S}^-(p, [q_1, \ldots, q_n, r]) & \mathcal{U}^-(p, \overline{q}) +\!\!+ r = \mathcal{U}^-(p, [r, q_1, \ldots, q_n]) \\ \mathcal{S}^-_\infty(\overline{q}) +\!\!+ r = \mathcal{S}^-_\infty([q_1, \ldots, q_n, r]) & \mathcal{U}^-_\infty(\overline{q}) +\!\!+ r = \mathcal{U}^-_\infty([r, q_1, \ldots, q_n]) \end{array}$$

For a valid satisfaction proof p of $\varphi_1 \,\mathcal{S}\, \varphi_2$ at $i-1$ (or $\varphi_1 \,\mathcal{U}\, \varphi_2$ at $i+1$) and a valid satisfaction proof r of φ_1 at i, $p +\!\!+ r$ is a valid satisfaction proof of $\varphi_1 \,\mathcal{S}\, \varphi_2$ (or $\varphi_1 \,\mathcal{U}\, \varphi_2$) at i. Similarly, for a valid violation proof p of $\varphi_1 \,\mathcal{S}\, \varphi_2$ at $i-1$ (or $\varphi_1 \,\mathcal{U}\, \varphi_2$ at $i+1$) and a valid violation proof r of φ_2 at i, $p +\!\!+ r$ is a valid violation proof of $\varphi_1 \,\mathcal{S}\, \varphi_2$ (or $\varphi_1 \,\mathcal{U}\, \varphi_2$) at i.

Constructor-monotonicity does not ensure that composing optimal proofs p and r will yield an optimal proof $p +\!\!+ r$. We therefore further extend our requirements on $\preceq$.

Definition 2. *A constructor-monotone $\preceq$ is* monotone *if it additionally satisfies:*

6. *If $p \preceq p'$ and $r \preceq r'$ then $p +\!\!+ r \preceq p' +\!\!+ r'$.*

We conclude this section by providing some (counter)examples of monotone wqos. The relation induced by the (weighted) size $\preceq_{size}$ (Example 2) is a monotone (total) wqo. Moreover, the relation $\preceq_{reach}$ defined as $p \preceq_{reach} q = reach(p) \leq reach(q)$ where $reach(p)$ is the largest time-point occurring in the proof p is a monotone (total) wqo. The dual relation that maximizes the smallest occurring time-point is neither monotone nor well-founded. Given two monotone wqos $\preceq_1$ and $\preceq_2$, the Cartesian product $p \preceq_\times q \iff (p \preceq_1 q \wedge p \preceq_2 q)$ is a monotone wqo too. A more exotic example of a monotone wqo is the multiset extension of a total order on atomic propositions. Finally, the relation that compares the sets of occurring atomic propositions by inclusion is not monotone.

5 Computing Optimal Proofs

We now describe a *proof search* algorithm O that takes as input the prefix u and the loop v of a lasso word $\rho = uv^\omega$, an LTL formula φ, a time-point i, and a monotone wqo $\preceq$ and computes an optimal (with respect to $\preceq$) valid proof for the violation or satisfaction of φ at i. The algorithm determines whether the formula is satisfied or violated during proof search; it thereby solves the LTL path-checking problem along the way. Because our proof system is complete, O always returns a proof of either satisfaction or violation. This is in contrast to a previously proposed proof search algorithm for LTL [8] that computes (non-optimal) proofs in an incomplete proof system.

Our algorithm exploits the monotonicity of the $\preceq$ relation to both bound the number of proofs that must be considered and eagerly combine optimal proofs for subformulas to obtain an optimal proof for the entire formula. In other words, we compute proofs in a bottom-up manner (in terms of formula structure), such that proofs of φ are constructed using only the optimal proofs of φ's immediate subformulas. For a fixed monotone wqo $\preceq$ and $\rho = uv^\omega$, our algorithm consists of two mutually recursive functions $\mathsf{O}(i,\varphi)$ and $\mathsf{C}(i,\varphi)$, shown in Figure 2. The *optimal proof* function $\mathsf{O}(i,\varphi)$ merely selects an optimal proof from a small set of *candidate proofs* (computed by C) at time-point i. The function $\mathsf{C}(i,\varphi)$ composes optimal proofs (computed by O) for φ's subformulas at the current time-point i and optimal proofs for φ at the previous $(i-1)$ and next $(i+1)$ time-points, exploiting the standard unrolling equations for $\mathcal{S}$ and $\mathcal{U}$.

The function $\mathsf{C}(i,\varphi)$ is defined by pattern matching on the formula φ's structure. The cases for atomic propositions and Boolean connectives are simple; for conjunction and disjunction we use the auxiliary functions *doDisj* and *doConj* (Figure 3) to compose optimal proofs of the subformulas. The precise outcome depends on whether the subformulas are satisfied or violated. For example, for $\varphi_1 \wedge \varphi_2$, we obtain a $\wedge^+$ proof only if both subformulas are satisfied, i.e., $\mathbb{V}(p_1)$ and $\mathbb{V}(p_2)$ are $\top$.

Temporal operators are more challenging to deal with. Setting the special case of $i = 0$ aside, C computes for the formula $\varphi_1 \,\mathcal{S}\, \varphi_2$ optimal proofs of φ_1 and φ_2 at i and

$\mathsf{C}(i,a) =$
  if $a \in \rho(i)$ then $\{ap^+(i,a)\}$ else $\{ap^-(i,a)\}$
$\mathsf{C}(i,\neg\varphi) =$
  let $p = \mathsf{O}(i,\varphi)$ in
  if $\mathbb{V}(p)$ then $\{\neg^-(p)\}$ else $\{\neg^+(p)\}$
$\mathsf{C}(i,\varphi_1 \vee \varphi_2) =$
  $doDisj(\mathsf{O}(i,\varphi_1), \mathsf{O}(i,\varphi_2))$
$\mathsf{C}(i,\varphi_1 \wedge \varphi_2) =$
  $doConj(\mathsf{O}(i,\varphi_1), \mathsf{O}(i,\varphi_2))$
$\mathsf{C}(i,\varphi_1 \,\mathcal{S}\, \varphi_2) =$
  if $i = 0$ then
    let $p_2 = \mathsf{O}(i,\varphi_2)$ in
    if $\mathbb{V}(p_2)$ then $\{\mathcal{S}^+(p_2,[])\}$ else $\{\mathcal{S}^-_\infty([p_2])\}$
  else
    $doSince(\mathsf{O}(i,\varphi_1), \mathsf{O}(i,\varphi_2), \mathsf{O}(i-1,\varphi_1 \,\mathcal{S}\, \varphi_2))$
$\mathsf{C}(i,\varphi_1 \,\mathcal{U}\, \varphi_2) =$
  $doUntil^\omega(i,\varphi_2) \cup doUntil^n(i,\varphi_1,\varphi_2)$

$\mathsf{O}(i,\varphi) = \min_\preceq(\mathsf{C}(i,\varphi))$

$doUntil^\omega(i,\varphi_2) =$
  if $i < |u| + h_p(\varphi_2) \cdot |v|$ then $\{\}$
  else
    let $ps_2 = [\mathsf{O}(i,\varphi_2), \ldots, \mathsf{O}(i+|v|-1,\varphi_2)]$ in
    if $\forall x \in ps_2.\neg\mathbb{V}(x)$ then $\{\mathcal{U}^-_\infty(ps_2)\}$ else $\{\}$

$doUntil^n(i,\varphi_1,\varphi_2) =$
  if $i < |u| + h_p(\varphi_1 \,\mathcal{U}\, \varphi_2) \cdot |v|$
  then
    $doUntil(\mathsf{O}(i,\varphi_1), \mathsf{O}(i,\varphi_2), \mathsf{O}(i+1,\varphi_1 \,\mathcal{U}\, \varphi_2))$
  else
    let $ps_1 = [\mathsf{O}(i,\varphi_1), \ldots, \mathsf{O}(i+|v|-1,\varphi_1)]$ in
    let $ps_2 = [\mathsf{O}(i,\varphi_2), \ldots, \mathsf{O}(i+|v|-1,\varphi_2)]$ in
    $\{\mathcal{U}^+(ps_2[j],[ps_1[0],\ldots,ps_1[j-1]]) \mid j < |v| \wedge$
      $\mathbb{V}(ps_2[j]) \wedge \forall k < j.\ \mathbb{V}(ps_1[k])\} \cup$
    $\{\mathcal{U}^-(ps_1[j],[ps_2[0],\ldots,ps_2[j]]) \mid j < |v| \wedge$
      $\neg\mathbb{V}(ps_1[j]) \wedge \forall k \leq j.\ \neg\mathbb{V}(ps_2[k])\}$

Fig. 2. Optimal proof algorithm: functions C,O, $doUntil^\omega$ and $doUntil^n$

an optimal proof of φ at $i-1$. These proofs are given to the auxiliary *doSince* function (Figure 4) that performs a case distinction on their truth values $\mathbb{V}(-)$ and accordingly constructs a set of proofs unrolling $\mathcal{S}$: $i \models \varphi_1 \,\mathcal{S}\, \varphi_2$ iff $i \models \varphi_2 \vee i \models \varphi_1 \wedge i-1 \models \varphi_1 \,\mathcal{S}\, \varphi_2$. Note that the set $\mathsf{C}(i,\varphi_1 \,\mathcal{S}\, \varphi_2)$ contains at least one and at most two elements. For example, if $i \models \varphi_2$, $i \models \varphi_2$, and $i-1 \models \varphi_1 \,\mathcal{S}\, \varphi_2$ all hold (and we have computed the optimal proofs for them), then there are two candidate proofs for $\varphi_1 \,\mathcal{S}\, \varphi_2$ corresponding to the two disjuncts in the unrolling equation.

Performing a dual unrolling for $\mathcal{U}$ would immediately lead to non-termination. However, since ρ is a lasso word, we can bound the proof search using Theorem 2. The case $\mathsf{C}(i,\varphi_1 \,\mathcal{U}\, \varphi_2)$ splits the proof search in two parts: $doUntil^\omega$ which corresponds to applying the rule $\mathcal{U}^-_\infty$ and $doUntil^n$ which corresponds to either extending any $\mathcal{U}$ proof at time-point $i+1$ or applying $\mathcal{U}^+$ or $\mathcal{U}^-$ with a fixed bound on the time-point j occurring in the assumptions of those rules. The function $doUntil^\omega$ checks whether the premises of $\mathcal{U}^-_\infty$ holds (i.e., $i > |u| + h_p(\varphi_2) \cdot |v|$ and $\forall k \in [i, i+|v|).\ k \not\models \varphi_2$) and either returns a single $\mathcal{U}^-_\infty$ proof or an empty set (if some premise is violated). The function $doUntil^n$ checks if the time-point i is larger than $|u| + h_p(\varphi_1 \,\mathcal{U}\, \varphi_2) \cdot |v|$, which is the threshold after which the measure of valid proofs of $\varphi_1 \,\mathcal{U}\, \varphi_2$ cannot decrease anymore by Theorem 2. For time-points before the threshold, the unrolling characterization of $\mathcal{U}$ is used to construct the proofs using *doUntil* (similar to *doSince*). For time-points after the threshold, it is sufficient to only search for proofs that use as subproofs only those proofs of φ_1 and φ_2 at time-points less than $i+|v|$, which ensures termination of the entire algorithm. Because of the soundness and completeness of the proof system, exactly one of the sets is necessarily non-empty in the union returned in the else branch of $doUntil^n$.

We prove that our algorithm always terminates.

Lemma 2. $\mathsf{C}(i,\varphi)$ *(and thus also* $\mathsf{O}(i,\varphi)$*) terminates for any i,* φ*,* ρ*, and* $\preceq$*.*

$\mathbb{V}(p_1)$	$\mathbb{V}(p_2)$	$doDisj(p_1,p_2)$	$doConj(p_1,p_2)$
$\top$	$\top$	$\{\vee_L^+(p_1), \vee_R^+(p_2)\}$	$\{\wedge^+(p_1,p_2)\}$
$\top$	$\bot$	$\{\vee_L^+(p_1)\}$	$\{\wedge_R^-(p_2)\}$
$\bot$	$\top$	$\{\vee_R^+(p_2)\}$	$\{\wedge_L^-(p_1)\}$
$\bot$	$\bot$	$\{\vee^-(p_1,p_2)\}$	$\{\wedge_L^-(p_1), \wedge_R^-(p_2)\}$

Fig. 3. Functions *doDisj* and *doConj*

$\mathbb{V}(p_1)$	$\mathbb{V}(p_2)$	$\mathbb{V}(p')$	$doSince(p_1,p_2,p')$	$doUntil(p_1,p_2,p')$
$\top$	$\top$	$\top$	$\{\mathcal{S}^+(p_2,[]), p' +\!\!+ p_1\}$	$\{\mathcal{U}^+(p_2,[]), p' +\!\!+ p_1\}$
$\top$	$\bot$	$\top$	$\{p' +\!\!+ p_1\}$	$\{p' +\!\!+ p_1\}$
$\bot$	$\bot$	$\top$	$\{\mathcal{S}^-(p_1,[p_2])\}$	$\{\mathcal{U}^-(p_1,[p_2])\}$
$\top$	$\bot$	$\bot$	$\{p' +\!\!+ p_2\}$	$\{p' +\!\!+ p_2\}$
$\bot$	$\bot$	$\bot$	$\{\mathcal{S}^-(p_1,[p_2]), p' +\!\!+ p_2\}$	$\{\mathcal{U}^-(p_1,[p_2]), p' +\!\!+ p_2\}$
otherwise			$\{\mathcal{S}^+(p_2,[])\}$	$\{\mathcal{U}^+(p_2,[])\}$

Fig. 4. Functions *doSince* and *doUntil*

Proof. We define the function $\mathsf{dist}(i,\varphi)$ as $|u| + h_p(\varphi)\cdot|v| - i$ if $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$ and as i otherwise. Consider the well-founded relation $\ll$ defined as the lexicographic product of the subformula relation $\sqsubset$ and a dist-comparison, i.e., $(i,\varphi) \ll (j,\psi) \iff \varphi \sqsubset \psi \vee (\varphi = \psi \wedge \mathsf{dist}(i,\varphi) < \mathsf{dist}(j,\psi))$. The relation $\ll$ is decreasing along the call graph of C (after unfolding the definition of O). □

We now prove our algorithm's correctness. We fix a lasso word $\rho = uv^\omega$ and a monotone wqo $\preceq$. We first establish properties of the auxiliary functions *doConj*, *doDisj*, and *doSince*. The case of $\mathcal{U}$ is a bit subtle as the proof search is split between the functions $doUntil^\omega$ and *doUntil*; hence we reason about them together in the following lemma.

Lemma 3. *For a time-point i, let p_1 and p_2 be optimal proofs of φ_1 at i and φ_2 at i, respectively.*

- *Let $\varphi = \varphi_1 \,\mathcal{S}\, \varphi_2$ and p' be an optimal proof of φ at $i-1$. If $i \neq 0$, then all proofs in $doSince(p_1,p_2,p')$ are valid proofs of φ at i and an optimal proof is contained among them.*
- *The same holds for $doDisj(p_1,p_2)$ for $\varphi = \varphi_1 \vee \varphi_2$ and $doConj(p_1,p_2)$ for $\varphi = \varphi_1 \wedge \varphi_2$.*
- *Let $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$ and p' be an optimal proof of φ at $i+1$. Then all proofs in $doUntil^\omega(i,\varphi_2) \cup doUntil(p_1,p_2,p')$ are valid proofs of φ at i and an optimal proof is contained among them.*

Proof (sketch for $\mathcal{S}$). We make a case distinction on the truth values $\mathbb{V}(p_1)$, $\mathbb{V}(p_2)$, and $\mathbb{V}(p')$. Here, we only consider the case where $\mathbb{V}(p_1) = \mathbb{V}(p_2) = \mathbb{V}(p') = \top$. Then $doSince(p_1,p_2,p') = \{\mathcal{S}^+(p_2,[]), p' +\!\!+ p_1\}$. Any valid proof of φ at i is either of the form $\mathcal{S}^+(q_2,[])$ or $q' +\!\!+ q_1$. Using the constructor-monotonicity condition 4, the monotonicity condition 6, and the optimality of p_1, p_2, p', we conclude that either $\mathcal{S}^+(p_2,[])$ or $p' +\!\!+ p_1$ is optimal. The other cases follow by similar arguments using the appropriate (constructor-)monotonicity conditions. □

We next prove the soundness and optimality of C, from which the same properties of O follow (also by induction hypotheses for the recursive calls to O in this proof itself).

Theorem 3. *Let $C = \mathsf{C}(i,\varphi)$ for a fixed lasso $\rho = uv^\omega$ and a monotone wqo $\preceq$. We have:*

Soundness *All proofs in C are valid proofs of φ at i.*
Optimality *An optimal proof of φ at i is contained in C.*

Proof (sketch). Proof by well-founded induction on $\ll$, the relation used to establish the termination of C in the proof of Lemma 2. As before, we write $thr(\varphi)$ to abbreviate the threshold $|u| + h_p(\varphi) \cdot |v|$. We perform a case distinction on φ. The cases for atomic propositions, Boolean operators, and $\mathcal{S}$ are either simple or follow directly from Lemma 3. We focus on optimality in the $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$ case.

Suppose $i < thr(\varphi)$. Then $\mathsf{C}(i,\varphi) = doUntil^\omega(i,\varphi_2) \cup doUntil(p_1, p_2, p'_\varphi)$ with $p_1 = \mathsf{O}(i,\varphi_1)$, $p_2 = \mathsf{O}(i,\varphi_2)$, and $p' = \mathsf{O}(i+1,\varphi)$. (Note that $doUntil^\omega(i,\varphi_2) = \{\}$ for $i < thr(\varphi_2)$.) Using the induction hypothesis, we have the optimality of p_1, p_2, and p'. The case follows using Lemma 3.

Suppose $i \geq thr(\varphi)$. Then the set of candidate proofs computed by $doUntil^n$ consists of only valid $\mathcal{U}^+$ and $\mathcal{U}^-$ proofs of φ whose immediate subproofs are proofs at time-points up to $i + |v|$. As before, $doUntil^\omega(i,\varphi)$ accounts for a potential proof of φ obtained using the $\mathcal{U}^-_\infty$ rule at i. We know that the proofs in ps_1 and ps_2 are optimal by the induction hypothesis. Suppose there exists an optimal proof at i that goes beyond $i + |v|$. Assume it is a satisfaction proof P of the form $\mathcal{U}^+(p, [q_i, \ldots, q_k])$ for some $k > i + |v|$ (the case for violation proofs follows analogously). Then there exist $k_1 > 0$ and $k_2 < |v|$ such that $k = i + k_1 \cdot |v| + k_2$. By the constructor-monotonicity condition 5, for the proof $P' = \mathcal{U}^+(p, [q_{i+k_1\cdot|v|}, \ldots, q_k])$ of φ at $i + k_1 \cdot |v|$ we have $P' \preceq P$. Using Theorem 2 k_1 times, we obtain another proof $P'' = \mathcal{U}^+(p', [q'_i, \ldots, q'_{k_2}])$ of φ at i with $P'' \preceq P'$. Because of the transitivity of $\preceq$ and the optimality of P, P'' is another optimal proof of φ at i. But P'' belongs to the set of proofs computed by $doUntil^n$. □

Corollary 2. *For a fixed lasso word ρ and a monotone wqo $\preceq$, the function $\mathsf{O}(i,\varphi)$ outputs an optimal (with respect to $\preceq$) valid proof p of φ at i.*

Finally, we discuss the time complexity of O.

Theorem 4. *For a fixed lasso $\rho = uv^\omega$ and a monotone wqo $\preceq$, an upper bound for the time complexity of $\mathsf{O}(0,\varphi)$ is $\mathcal{O}((|u| + h(\varphi) \cdot |v|) \cdot |\mathsf{SF}(\varphi)| \cdot f(\preceq) \cdot w(\preceq) \cdot |v|)$, where $f(\preceq)$ is the complexity of comparing proofs with respect to $\preceq$ and $w(\preceq)$ is the* width *of $\preceq$, i.e., the maximum size of an antichain in $\preceq$.*

Proof (sketch). To compute $\mathsf{O}(0,\varphi)$, the largest k at which $\mathsf{O}(k,\varphi)$ can be recursively called is $thr(\varphi) = |u| + h_p(\varphi) \cdot |v|$. Furthermore, to compute $\mathsf{O}(i,\psi)$ at $i \geq thr(\varphi)$ for some subformula ψ of φ, the largest time-point at which O is recursively called on immediate subformulas is $i + |v|$ if $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$ and it is zero otherwise. Therefore, the largest time-point at which O is recursively called is $thr(\varphi) + h_f(\varphi) \cdot |v| = |u| + h(\varphi) \cdot |v|$. In the worst case, for each time-point i, O can be called for every subformula. Hence, the parametrized time complexity is $\mathcal{O}((|u| + h(\varphi) \cdot |v|) \cdot |\mathsf{SF}(\varphi)| \cdot f(\preceq) \cdot w(\preceq) \cdot M)$, where M is the largest cardinality of a set returned by C. Note that the $\mathcal{O}(f(\preceq) \cdot w(\preceq) \cdot M)$ is an upper bound on the complexity of computing a minimal element with respect to $\preceq$ in a set of size M [9]. If $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$ and $i \geq thr(\varphi)$, then M is bounded by $|v|$ and

otherwise it is at most 2. Hence, the calls $\mathsf{O}(i,\varphi)$ that trigger an expensive minimum computation are very rare compared to the overall number of calls. Also note that in case $\preceq$ is total, we have $w(\preceq) = 1$.

We ignored above that O may be called several times with the same arguments. Memoizing O provides a countermeasure against this potential inefficiency. □

6 Implementation and Evaluation

We implemented the presented algorithm in a publicly available prototype [1]. The implementation is concise: altogether about 1500 lines of OCaml code. Our tool supports a much richer LTL syntax than the one shown in this paper: users can write formulas involving *true*, *false*, *(bi)implications* $\rightarrow$ and $\leftrightarrow$, *next* $\bigcirc$, *previous* $\bullet$, *eventually* $\Diamond$, *always* $\Box$, *once* $\blacklozenge$, and *historically* $\blacksquare$. All of these operators are treated as first class citizens: optimal proofs are computed in an extended proof system containing inference rules for each of the new operators. In principle, we could have defined operators like $\Diamond$ in terms of $\mathcal{U}$. However, we want to provide proofs precisely for a formula the user wrote, rather than an equivalent version of it, as these will be easier for the user to understand.

We observed earlier that repeated calls to C (and O) with the same arguments may occur. We therefore memoize the function C to avoid expensive recomputations. The memoization is performed using a hash-table that stores the already computed results of C for its arguments. To hash formulas efficiently, we use hash-consing [10]. Hash-consing of proofs would also improve the space efficiency of our algorithm by sharing subproofs, but we have not implemented this optimization yet. In our experiments, space consumption was not a bottleneck.

Our tool parses the output of the NuSMV model checker [2]. In case of violations, this output contains the LTL formula and the lasso word counterexample—the inputs for our algorithm. The user can choose between a few predefined monotone wqos (or Cartesian products thereof) for optimization. The textual representation of the optimal proof computed is pretty printed separating the different levels in the proof by indentation.

We evaluate our algorithm on counterexamples generated by the NuSMV model checker on realistic models and specifications [18, 25]. The LTL formulas we consider include freely nested past and future operators, with temporal heights ranging from 3 to 8. We used NuSMV release 2.1.2 to regenerate the counterexamples, as some of the models were not compatible with the latest release of NuSMV (2.6.1). We ran our experiments on an Intel Core i7 2.5 GHz processor with 16GB RAM. Figure 5 shows the $|-|$ and *reach* of optimal proofs found by our tool with respect to three monotone wqos: $\preceq_{size}$ and $\preceq_{reach}$ induced by $|-|$ and *reach* (as described in §4) and their Cartesian product $\preceq_{\times}$ (defined as $p \preceq_{\times} q = p \preceq_{size} q \wedge p \preceq_{reach} q$). Columns $|u|$ and $|v|$ show the lengths of the prefix and the loop of the lasso word, whereas h_p and h_f show the past and future height of the specification. In all but one example, optimal proofs with respect to the partial order Cartesian product $\preceq_{\times}$ result in minimal $|-|$ and *reach* values.

The generated proofs helped explain the violations. Figure 6 (left) shows two proofs P (optimal with respect to $\preceq_{reach}$) and Q (optimal with respect to $\preceq_{size}$) output by our tool for the formula φ_0 on the counterexample lasso word uv^{ω} generated by NuSMV for the model *srg5* (a 5-bit counter). The textual representation includes the aforementioned

Model	Spec	$\|u\|$	$\|v\|$	h_p	h_f	$\preceq_{size}$		$\preceq_{reach}$		$\preceq_{\times}$	
						$\|p\|$	$reach(p)$	$\|p\|$	$reach(p)$	$\|p\|$	$reach(p)$
srg5	φ_0	15	2	4	4	7	16	8	6	7	16
srg5	φ_1	0	16	4	4	621	70	621	33	621	33
dme2	φ_2	0	111	2	1	11	242	14	20	11	20
dme3	φ_2	0	216	2	1	11	494	14	62	11	62
dme4	φ_2	0	280	2	1	11	642	14	82	11	82
abp	φ_3	18	20	2	2	7	59	7	3	7	3
1394-3-2	φ_4	15	2	1	2	7	18	7	18	7	18

$\psi = ((\Diamond\Box(\neg p) \wedge \Box\Diamond q) \wedge \Box\Diamond x_0) \rightarrow \Diamond(x_0 \,\mathcal{S}\, (x_1 \,\mathcal{S}\, (x_2 \,\mathcal{S}\, (x_3 \,\mathcal{S}\, x_4))))$

$\varphi_0 = \neg\psi \qquad \varphi_1 = \neg(\psi \wedge ((\Diamond\Box(\neg p) \wedge \Box\Diamond q) \wedge \Box\Diamond x_0))$

$\varphi_2 = \Box(p \rightarrow \neg(\neg p \,\mathcal{S}\, (p \,\mathcal{S}\, q))) \quad \varphi_3 = \Box(p \rightarrow \bullet\blacksquare\neg p) \quad \varphi_4 = \neg\Diamond\Box(p \rightarrow \neg(\neg q \,\mathcal{S}\, r))$

Fig. 5. Evaluation results and LTL properties

additional constructors for $\rightarrow$, $\Box$, and $\Diamond$. For example, P demonstrates the satisfaction of the implication by providing evidence for the conclusion's satisfaction ($\rightarrow_R^+(\ldots)$). In contrast, Q shows that the implication can also be vacuously satisfied by providing evidence for the assumption's violation ($\rightarrow_L^+(\ldots)$). The implication's vacuous satisfaction is not desirable and indicates a problem with the specification (which is amended in φ_1). Yet the vacuity is far from obvious when just given the trace.

Figure 6 (right) shows the atomic propositions occurring in each of the proofs. In the visualization, which our tool can also output, each column corresponds to a single alphabet letter. A gray box in a row labeled by an atomic predicate denotes that the predicate is true (and white denotes that the predicate is false) at that letter. The marked boxes are solely *responsible* for φ_0's violation in P or Q respectively: flipping non-responsible atomic propositions in the trace will not affect P's or Q's validity. The visualization is helpful even though it discards most of the information present in the proof. Support for interactively selecting subformulas and visualizing the responsible atomic propositions in the corresponding subproofs would further improve the usability.

As another example, our tool computed an optimal proof of φ_2's violation with respect to $\preceq_{reach}$ for the lasso word counterexample generated from the model *dme4*. This proof uses the time-point (82) as the earliest possible witness of a violation of $\Box$ in the counterexample of length 280. At that time-point, p is true in the lasso word and $\neg p \,\mathcal{S}\, (p \,\mathcal{S}\, q)$ is violated (witnessed by a recursive subproof, which does not look beyond the time-point 82). For the *1394-3-2* counterexample, the computed proofs made it evident that the implication $p \rightarrow \neg(\neg q \,\mathcal{S}\, r)$ was vacuously satisfied, resulting in φ_4's violation.

Our tool has good performance with memoization being a key optimization. Prior to its use, computation on some of the examples took several minutes. With memoization in place, optimal proofs were generated within one second for each of the examples.

7 Related work

Markey and Schnoebelen [21] provide a comprehensive overview of the *path-checking* problem for various fragments of LTL. They reduce LTL path checking on lasso words

$$P = \neg^-(\rightarrow_R^+(\Diamond^+ \\ (\mathcal{S}^+(\mathcal{S}^+(\mathcal{S}^+(\mathcal{S}^+(ap^+(x_4,6),[]),[]),[]),[]))))$$

$$Q = \neg^-(\rightarrow_L^+(\wedge_R^-(\Box^-(\Diamond^- \\ ([ap^-(x_0,15),ap^-(x_0,16)])))))$$

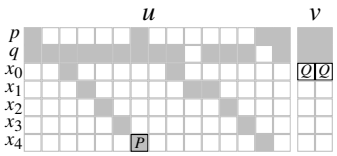

Fig. 6. Two example proofs

to LTL path checking on finite words. We build upon the core argument behind this result, reproduced in our Lemma 1. We further generalize their result to reasoning about optimal proofs in our Theorem 2. The best known upper bound for LTL path checking is provided by Kuhtz and Finkbeiner [15], showing that it can be efficiently parallelized.

Several proof systems [5, 19] for LTL have been previously proposed to check a formula's *validity*. However, these proof systems are significantly different from ours, which checks the satisfaction (or violation) of a formula on a particular lasso word. An LTL proof system for checking satisfaction (or violation) closely related to ours was proposed by Cini and Francalanza [8]. They focus on *runtime verification* and provide an online proof search algorithm that processes letter-wise a word's finite prefix. Their proof system is sound but not complete: it cannot prove the violation of liveness properties (or the satisfaction of safety properties), which is natural in a runtime verification application. They are mainly concerned with solving the path-checking problem by proof search and do not focus on leveraging the found proofs as explanations.

Chechik and Gurfinkel [6] give a sound and complete proof system for CTL to explain violations of model-checkers and to debug specifications, in line with our goals. They also develop an interactive user interface for exploring different counterexamples for a model and the corresponding proofs. Their proof system is arguably more complex and thus harder to understand than ours: partly as it handles a branching-time logic and partly because they rely on the unrolling equations of the temporal operators and state summaries to finitely represent the infinite proof trees that arise when considering negations of temporal operators, instead of employing a simple meta-argument (Lemma 1) as we do. They do not consider past operators nor the optimality of the proof trees. Similar unrolling-based proof systems were developed for model-checking games [17]. Winning strategies in such games are certain notions of proof. In general, we argue that unrolling may be a good approach to solving the path-checking problem. However, it is not ideal for explaining a violation to a user who wrote the specification having LTL's standard semantics in mind rather than the recursive equations that underly the unrolling technique.

Sulzmann and Zechner's [26] proof system for LTL on finite words is also motivated by using proofs as explanations. However, they neither support past operators nor lasso words. Moreover, they only consider formulas in negation normal form, but neglect dual future operators, which significantly limits their language's expressiveness. Even if the dual operators were supported, imposing a normal form is problematic when a proof should serve as an explanation. Users typically do not think in term of normal forms but prefer to freely use the syntax of LTL; hence the explanation given should be as close as possible to the users' mindset. Sulzmann and Zechner compute optimal proofs with respect to a particular monotone order (lexicographic combination of the proof size and the relation that prefers $\vee_L^+$ over $\vee_R^+$), which is an instance in our generalized technique.

There has been significant work in the model checking community to address the problem of understanding counterexamples [3, 12–14]. Most of these works focus on the interaction of the system model being verified and the counterexample. Our approach explores the interaction of the LTL property and the counterexample, without knowing the system model. An ideal explanation should combine all three components. We believe that our work is an important step towards achieving this goal.

The notion of *causality* [4, 27] has been used to explain model checking counterexamples. Causality can be encoded as a relation in our framework, but it is not monotone. This is not surprising since the problem of computing the minimal causal set is NP hard and the existing solutions therefore settle for approximations. Our algorithm is more tractable, but can only optimize for simpler relations.

The size of the counterexample input to our algorithm affects the resulting proof tree's size: smaller counterexamples typically result in smaller proof trees. Our work can thereby directly benefit from past work on computing short counterexamples [11, 25]. Other lines of work aim at identifying the vacuous satisfaction of properties [16,20] or justifying why the system satisfies a property when no counterexample is found [22–24]. We provide such a justification for a single trace, but not for an entire system model.

Finally, we refer to a survey on *provenance in databases* [7], which aims at identifying the root cause of violations, yet without taking the temporal dimension into account.

8 Conclusion

We have proposed a sound and complete proof system for LTL on lasso words. A proof tree in this system carries all the information necessary to witness and explain a formula's satisfaction or violation. We have devised and implemented an algorithm for computing a proof tree that is minimal with respect to a given monotone well-quasi-order $\preceq$. The parametrization by $\preceq$ allows the algorithm's users to optimize for different statistics (such as $|-|$ or *reach*) of the proof tree or even their combinations.

Our work lays the foundation for explaining the counterexamples generated by model checking tools. There are several natural continuations. In real world examples, even optimal proof trees can be too large to examine in practice. Devising user-friendly ways to explore them is therefore a practically relevant information visualization challenge. On the theoretical side, an open problem is to develop analogous techniques for other specification languages used by model checkers. Finally it would be interesting to adapt our proof search algorithm to work in an online fashion. This would enable its use to explain online, the verdicts produced by runtime verification algorithms.

Acknowledgment We thank Srđan Krstić, Felix Klaedtke, and Joshua Schneider for discussions on using proof trees as explanations. Srđan Krstić, Karel Kubíček, and anonymous reviewers provided useful comments on early drafts of this paper. This work is supported by the Swiss National Science Foundation grant Big Data Monitoring (167162).

References

1. Explanator: Send in the Explanator—it explains satisfaction/violation of LTL formulas on lasso words. `https://bitbucket.org/traytel/explanator` (2018)

2. NuSMV: A new symbolic model checker. http://nusmv.fbk.eu/ (2018)
3. Ball, T., Naik, M., Rajamani, S.K.: From symptom to cause: Localizing errors in counterexample traces. In: POPL 2003. pp. 97–105. ACM (2003)
4. Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R.J.: Explaining counterexamples using causality. Form. Methods in Syst. Des. **40**(1), 20–40 (2012)
5. Brünnler, K., Lange, M.: Cut-free sequent systems for temporal logic. J. Log. Algebr. Program. **76**(2), 216–225 (2008)
6. Chechik, M., Gurfinkel, A.: A framework for counterexample generation and exploration. STTT **9**(5-6), 429–445 (2007)
7. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Found. Trends Databases **1**(4), 379–474 (2009)
8. Cini, C., Francalanza, A.: An LTL proof system for runtime verification. In: TACAS 2015. LNCS, vol. 9035, pp. 581–595. Springer (2015)
9. Daskalakis, C., Karp, R.M., Mossel, E., Riesenfeld, S., Verbin, E.: Sorting and selection in posets. SIAM J. Comput. **40**(3), 597–622 (2011)
10. Filliâtre, J., Conchon, S.: Type-safe modular hash-consing. In: ACM Workshop on ML. pp. 12–19. ACM (2006)
11. Gastin, P., Moro, P.: Minimal counterexample generation for SPIN. In: SPIN 2007. LNCS, vol. 4595, pp. 24–38. Springer (2007)
12. Groce, A., Chaki, S., Kroening, D., Strichman, O.: Error explanation with distance metrics. STTT **8**(3), 229–247 (2006)
13. Groce, A., Kroening, D.: Making the most of BMC counterexamples. Electr. Notes Theor. Comput. Sci. **119**(2), 67–81 (2005)
14. Groce, A., Visser, W.: What went wrong: Explaining counterexamples. In: SPIN 2003. LNCS, vol. 2648, pp. 121–135. Springer (2003)
15. Kuhtz, L., Finkbeiner, B.: LTL path checking is efficiently parallelizable. In: ICALP 2009. LNCS, vol. 5556, pp. 235–246. Springer (2009)
16. Kupferman, O.: Sanity checks in formal verification. In: CONCUR 2006. LNCS, vol. 4137, pp. 37–51. Springer (2006)
17. Lange, M., Stirling, C.: Model checking games for branching time logics. J. Log. Comput. **12**(4), 623–639 (2002)
18. Latvala, T., Biere, A., Heljanko, K., Junttila, T.A.: Simple is better: Efficient bounded model checking for past LTL. In: VMCAI 2005. LNCS, vol. 3385, pp. 380–395. Springer (2005)
19. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems - specification. Springer (1992)
20. Maretic, G.P., Dasthi, M.T., Basin, D.A.: Semantic vacuity. In: TIME 2015. pp. 111–120. IEEE Computer Society (2015)
21. Markey, N., Schnoebelen, P.: Model checking a path. In: CONCUR 2003. LNCS, vol. 2761, pp. 248–262. Springer (2003)
22. Namjoshi, K.S.: Certifying model checkers. In: CAV 2001. LNCS, vol. 2102, pp. 2–13. Springer (2001)
23. Peled, D.A., Pnueli, A., Zuck, L.D.: From falsification to verification. In: FSTTCS 2001. LNCS, vol. 2245, pp. 292–304. Springer (2001)
24. Peled, D.A., Zuck, L.D.: From model checking to a temporal proof. In: SPIN 2001. LNCS, vol. 2057, pp. 1–14. Springer (2001)
25. Schuppan, V., Biere, A.: Shortest counterexamples for symbolic model checking of LTL with past. In: TACAS 2005. LNCS, vol. 3440, pp. 493–509. Springer (2005)
26. Sulzmann, M., Zechner, A.: Constructive finite trace analysis with linear temporal logic. In: TAP 2012. LNCS, vol. 7305, pp. 132–148. Springer (2012)
27. Wang, C., Yang, Z., Ivancic, F., Gupta, A.: Whodunit? Causal analysis for counterexamples. In: ATVA 2006. LNCS, vol. 4218, pp. 82–95. Springer (2006)