




Explainable Online Monitoring of Metric Temporal Logic

Leonardo Lima^{1†} , Andrei Herasimau², Martin Raszyk^{3†} ,
Dmitriy Traytel^{1†} , and Simon Yuan²

¹ Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

² Department of Computer Science, ETH Zürich, Zurich, Switzerland

³ DFINITY Foundation, Zurich, Switzerland

Abstract. Runtime monitors analyze system execution traces for policy compliance. Monitors for propositional specification languages, such as metric temporal logic (MTL), produce Boolean verdicts denoting whether the policy is satisfied or violated at a given point in the trace. Given a sufficiently complex policy, it can be difficult for the monitor’s user to understand how the monitor arrived at its verdict. We develop an MTL monitor that outputs verdicts capturing why the policy was satisfied or violated. Our verdicts are proof trees in a sound and complete proof system that we design. We demonstrate that such verdicts can serve as explanations for end users by augmenting our monitor with a graphical interface for the interactive exploration of proof trees. As a second application, our verdicts serve as certificates in a formally verified checker we develop using the Isabelle proof assistant.

Keywords: metric temporal logic · runtime monitoring · explanations · proof system · formal verification · certification

1 Introduction

In runtime verification, monitoring is the task of analyzing an event stream produced by a running system for violations of specified policies. An online monitor for a propositional policy specification language, such as metric temporal logic (MTL), consumes the stream event-wise and gradually produces a stream of Boolean verdicts denoting the policy’s satisfaction or violation at every point in the event stream. MTL monitors [1, 15, 18, 19, 24] use complex algorithms, whose correctness is not obvious, to efficiently arrive at the verdicts. Yet, users must rely on the algorithms being correct and correctly implemented, as the computed verdicts carry no information as to why the policy is satisfied or violated.

The two main approaches to increase the reliability of complex algorithm implementations are verification and certification. Formal verification using proof assistants or software verifiers is laborious and while it provides an ultimate level of trust, the user of a verified tool still gains no insight into why a specific, surely correct verdict was produced. In contrast, certification can yield both trust (especially when the certificate checker is itself formally verified) and insight, provided that the certificate is not only machine-checkable but also human-understandable.

[†] Lima and Traytel are supported by a Novo Nordisk Fonden start package grant (NNF20OC0063462). Raszyk’s work was carried out during his past employment at ETH Zürich supported by the Swiss National Science Foundation grant Big Data Monitoring (167162). All authors thank David Basin for supporting this work.

In this paper, we develop a certification approach to MTL monitoring: instead of Boolean verdicts, we require the monitor to produce checkable and understandable certificates. To this end, we develop a sound and complete local proof system (§2) for the satisfaction and violation of MTL policies. Following Cini and Francalanza [12], local means that a proof denotes the policy satisfaction on a given stream of events and not general MTL satisfiability (for any stream). Our proof system is an adaptation of the local proof system for LTL satisfiability on lasso words [2] to MTL with past and bounded future temporal operators. A core design choice for our proof system was to remain close to the MTL semantics and thus to be understandable for users who reason about policies in terms of the semantics. Therefore, proof trees in our proof system, or rather their compact representation as proof objects (§3), serve as understandable certificates.

With the certificate format in place, we devise an algorithm that computes minimal (in terms of size) proof objects (§4). We implement the algorithm in OCaml and augment it with an interactive web application¹ to visualize and explore the computed proof objects (§5). Independently, we prove the soundness and completeness of our proof system and formally verify a proof checker using the Isabelle/HOL proof assistant. We extract OCaml code from this formalization and use it to check the correctness of the verdicts produced by our unverified algorithm. To ensure that our correct verdicts are also minimal, we develop a second formally verified but less efficient monitoring algorithm in Isabelle, which we use to compute the minimal proof object size when testing our unverified algorithm.

Finally, we demonstrate how our work provides explainable monitoring output through several examples (§6). In summary, we make the following contributions:

- We develop a sound and complete local proof system for past and bounded future MTL that follows closely the semantics of the MTL operators.
- We develop an efficient algorithm to compute size-minimal proof objects representing proof trees in our proof system.
- We implement our algorithm in a new full fledged monitoring tool EXPLANATOR2 that includes a web front end and a formally verified proof object checker. Our implementation is publicly available [17].

Related Work. We take the earlier work on optimal proofs for LTL on lasso words as our starting point [2] but change the setting from lasso words to streams of time-stamped events and the logic from LTL to MTL. Moreover, the earlier work considered the offline path checking problem, whereas we are tackling online monitoring here. Parts of the work presented here are also described in two B.Sc. theses by Yuan [30] and Herasimau [13]. Yuan developed the MTL proof system we present here as well as a monitoring algorithm for computing optimal proofs based on dynamic programming (similarly to the algorithm for LTL on lasso words [2]). Herasimau formalized Yuan’s development in Isabelle/HOL. We use his work as the basis for our formally verified checker. Here, we present a different algorithm that resembles the algorithms used by state-of-the-art monitors for metric first-order temporal logic [3, 21], which performs much better than dynamic programming algorithms for non-trivial metric interval bounds.

Our algorithm minimizes the computed proof objects’ (weighted) size. Yuan [30] describes a more flexible approach parameterized by a comparison relation on proof

¹ <https://runtime-monitoring.github.io/explanator2/>

$$\begin{aligned}
 i \models p & \quad \text{iff } p \in \pi_i \mid i \models \alpha \vee \beta \text{ iff } i \models \alpha \text{ or } i \models \beta \mid i \models \bullet_I \alpha \text{ iff } i > 0 \text{ and } \tau_i - \tau_{i-1} \in I \text{ and } i-1 \models \alpha \\
 i \models \neg \alpha & \quad \text{iff } i \not\models \alpha \mid i \models \alpha \wedge \beta \text{ iff } i \models \alpha \text{ and } i \models \beta \mid i \models \circ_I \alpha \text{ iff } \tau_{i+1} - \tau_i \in I \text{ and } i+1 \models \alpha \\
 i \models \alpha \mathcal{S}_I \beta & \text{ iff } j \models \beta \text{ for some } j \leq i \text{ with } \tau_i - \tau_j \in I \text{ and } k \models \alpha \text{ for all } j < k \leq i \\
 i \models \alpha \mathcal{U}_I \beta & \text{ iff } j \models \beta \text{ for some } j \geq i \text{ with } \tau_j - \tau_i \in I \text{ and } k \models \alpha \text{ for all } i \leq k < j
 \end{aligned}$$

 Fig. 1: Semantics of MTL for a fixed event stream $\rho = \langle (\pi_i, \tau_i) \rangle_{i \in \mathbb{N}}$

objects that describes what the algorithm should optimize for. Yuan thereby discovers a flaw in the correctness claim for the algorithm for LTL on lasso words [2] and corrects it by further restricting the supported comparisons. Herasimau [13] relaxes Yuan’s requirements while formally establishing the correctness statement. We focus on the proof object size as it both simplifies the presentation and caters for a more efficient algorithm.

Formal verification of monitors is a timely topic. Some verified monitors were developed recently using proof assistants, e.g., VeriMon [21] and Vydra [20] in Isabelle and lattice-mtl [5] in Coq. Others leveraged SMT technology to increase their trustworthiness [9, 11]. To the best of our knowledge, we present the first verified checker for an online monitor’s output, even though verified certifiers are standard practice in other areas such as distributed systems [26], model checking [28, 29], and SAT solving [8, 16].

Our work follows the “proof trees as explanations” paradigm and thereby joins a series of works on LTL [2, 12, 23], CFTL [10], and CTL [6]. Of these only Basin et al. [2] supports past operators and none support metric intervals. Two of the above works [6, 12] use proof systems based on the unrolling equations for temporal operators instead on the operator’s semantics, which we believe is suboptimal for understandability: users think about the operators in term of their semantics and not in terms of unrolling equations.

Outside of the realm of temporal logics one can find the “proof trees as explanations” paradigm in regular expression matching [22] and in the database community [7].

Metric Temporal Logic. We briefly recall MTL’s syntax and point-based semantics [4]. MTL formulas are built from atomic propositions (a, b, c, \dots) via Boolean (\wedge, \vee, \neg) and metric temporal operators (previous \bullet_I , next \circ_I , since \mathcal{S}_I , until \mathcal{U}_I), where $I = [l, r]$ is a non-empty interval of natural numbers with $l \in \mathbb{N}$ and $r \in \mathbb{N} \cup \{\infty\}$. We omit the interval when $l = 0$ and $r = \infty$. For the until operator $\mathcal{U}_{[l, r]}$, we require the interval to be bounded, i.e., $r \neq \infty$. Formulas are interpreted over streams of time-stamped events $\rho = \langle (\pi_i, \tau_i) \rangle_{i \in \mathbb{N}}$, also called traces. An event π_i is a set of atomic propositions that hold at the respective time-point i . Time-stamps τ_i are natural numbers that are required to be monotone (i.e., $i \leq j$ implies $\tau_i \leq \tau_j$) and progressing (i.e., for all τ there exists a time-point i with $\tau_i > \tau$). Note that consecutive time-points can have the same time-stamp. Figure 1 shows MTL’s standard semantics for a formula φ at time-point i for a fixed trace ρ .

Fix a trace $\rho = \langle (\pi_i, \tau_i) \rangle_{i \in \mathbb{N}}$. The *earliest time-point* of a time-stamp τ on ρ is the smallest time-point i such that $\tau_i \geq \tau$ and is denoted as $\text{ETP}_\rho(\tau)$. Similarly, the *latest time-point* of a time-stamp $\tau \geq \tau_0$ on ρ is the greatest time-point i such that $\tau_i \leq \tau$ and is denoted as $\text{LTP}_\rho(\tau)$. Whenever the trace ρ is fixed, we will only write $\text{ETP}(\tau)$ and $\text{LTP}(\tau)$.

2 Local Proof System

We introduce a local proof system for monitoring MTL formulas. It contains two mutually dependent judgments: \vdash^+ (for satisfaction proofs) and \vdash^- (for violation proofs). A

$$\begin{array}{c}
\frac{a \in \pi_i}{i \vdash^+ a} ap^+ \quad \frac{a \notin \pi_i}{i \vdash^- a} ap^- \quad \frac{i \vdash^- \alpha}{i \vdash^+ \neg \alpha} \neg^+ \quad \frac{i \vdash^+ \alpha}{i \vdash^- \neg \alpha} \neg^- \\
\\
\frac{i \vdash^+ \alpha}{i \vdash^+ \alpha \vee \beta} \vee_L^+ \quad \frac{i \vdash^+ \beta}{i \vdash^+ \alpha \vee \beta} \vee_R^+ \quad \frac{i \vdash^+ \alpha \quad i \vdash^+ \beta}{i \vdash^+ \alpha \wedge \beta} \wedge^+ \\
\\
\frac{i \vdash^- \alpha \quad i \vdash^- \beta}{i \vdash^- \alpha \vee \beta} \vee^- \quad \frac{i \vdash^- \alpha}{i \vdash^- \alpha \wedge \beta} \wedge_L^- \quad \frac{i \vdash^- \beta}{i \vdash^- \alpha \wedge \beta} \wedge_R^- \\
\\
\frac{j \leq i \quad \tau_i - \tau_j \in I \quad j \vdash^+ \beta \quad \forall k \in (j, i]. k \vdash^+ \alpha}{i \vdash^+ \alpha S_I \beta} S^+ \quad \frac{\tau_i - \tau_0 < l}{i \vdash^- \alpha S_{[l, r]} \beta} S_{<I}^- \\
\\
\frac{E_i^p([l, r]) \leq j \quad j \leq i \quad m = L_i^p([l, r]) \quad \tau_i - \tau_0 \geq l \quad j \vdash^- \alpha \quad \forall k \in [j, m]. k \vdash^- \beta}{i \vdash^- \alpha S_{[l, r]} \beta} S^- \\
\\
\frac{j = E_i^p([l, r]) \quad m = L_i^p([l, r]) \quad \tau_i - \tau_0 \geq l \quad \forall k \in [j, m]. k \vdash^- \beta}{i \vdash^- \alpha S_{[l, r]} \beta} S_\infty^- \\
\\
\frac{i \leq j \quad \tau_j - \tau_i \in I \quad j \vdash^+ \beta \quad \forall k \in [i, j]. k \vdash^+ \alpha}{i \vdash^+ \alpha \mathcal{U}_I \beta} \mathcal{U}^+ \\
\\
\frac{m = E_i^f(I) \quad i \leq j \quad j \leq L_i^f(I) \quad j \vdash^- \alpha \quad \forall k \in [m, j]. k \vdash^- \beta}{i \vdash^- \alpha \mathcal{U}_I \beta} \mathcal{U}^- \\
\\
\frac{m = E_i^f(I) \quad j = L_i^f(I) \quad \forall k \in [m, j]. k \vdash^- \beta}{i \vdash^- \alpha \mathcal{U}_I \beta} \mathcal{U}_\infty^- \\
\\
\frac{i > 0 \quad \tau_i - \tau_{i-1} \in I \quad i-1 \vdash^+ \alpha}{i \vdash^+ \bullet_I \alpha} \bullet^+ \quad \frac{\tau_{i+1} - \tau_i \in I \quad i+1 \vdash^+ \alpha}{i \vdash^+ \circ_I \alpha} \circ^+ \\
\\
\frac{i > 0 \quad \tau_i - \tau_{i-1} < l}{i \vdash^- \bullet_I \alpha} \bullet_{<I}^- \quad \frac{}{0 \vdash^- \bullet_I \alpha} \bullet_0^- \quad \frac{\tau_{i+1} - \tau_i < l}{i \vdash^- \circ_I \alpha} \circ_{<I}^- \quad \frac{i+1 \vdash^- \alpha}{i \vdash^- \circ_I \alpha} \circ^- \\
\\
\frac{i > 0 \quad i-1 \vdash^- \alpha}{i \vdash^- \bullet_I \alpha} \bullet^- \quad \frac{i > 0 \quad \tau_i - \tau_{i-1} > l}{i \vdash^- \bullet_I \alpha} \bullet_{>I}^- \quad \frac{\tau_{i+1} - \tau_i > l}{i \vdash^- \circ_I \alpha} \circ_{>I}^-
\end{array}$$

Fig. 2: Local proof system for MTL for a fixed event stream $\rho = \langle (\pi_i, \tau_i) \rangle_{i \in \mathbb{N}}$

satisfaction proof describes the satisfaction of a formula at a given time-point on a fixed event stream $\rho = \langle (\pi_i, \tau_i) \rangle_{i \in \mathbb{N}}$ and similarly for violation proofs. Each deduction rule is suffixed by $^+$ or $^-$, indicating whether an operator has been satisfied or violated. Furthermore, we define $E_i^p(I) := \text{ETP}(\tau_i - r)$ and $L_i^p(I) := \min(i, \text{LTP}(\tau_i - l))$ for $I = [l, r]$, which correspond to the earliest and latest time-point within the interval I , respectively, when formulas having S_I as their topmost operator are considered. Note that in the definition of $L_i^p(I)$ we must take the minimum to account for multiple time-stamps with the same value. For formulas having \mathcal{U}_I as their topmost operator, both definitions are mirrored, resulting in $E_i^f(I) := \max(i, \text{ETP}(\tau_i + l))$ and $L_i^f(I) := \text{LTP}(\tau_i + r)$. As usual, our proof system is defined as the least relation satisfying the rules shown in Figure 2.

The semantics of the MTL operators directly corresponds to the satisfaction rules ap^+ , \neg^+ , \vee_L^+ , \vee_R^+ , \wedge^+ , S^+ , \mathcal{U}^+ , \bullet^+ , and \circ^+ . For instance, consider two time-points j and i such that $j \leq i$. The rule S^+ is applied whenever the time-stamp difference $\tau_i - \tau_j$ belongs to the interval I , and there is a witness for a violation proof of β in the form of $j \vdash^+ \beta$, and there is a finite sequence of satisfaction proofs of α for all $k \in (j, i]$. The violation

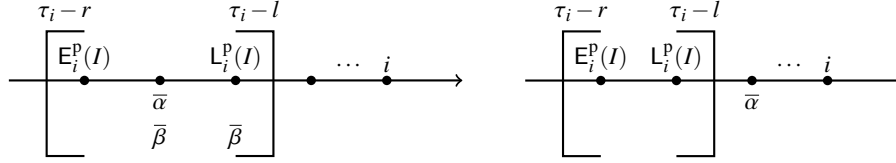
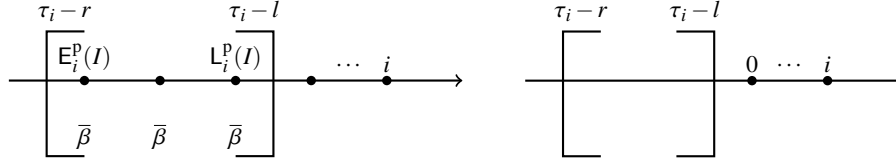

 Fig. 3(a). \mathcal{S}^- cases

 Fig. 3(b). \mathcal{S}_∞^- and $\mathcal{S}_{<l}^-$ cases

 Fig. 3: Graphical representation of the violation cases for $\alpha \mathcal{S}_I \beta$ with $I = [l, r]$

rules for the non-temporal operators ap^- , \neg^- , \vee^- , \wedge_L^- , \wedge_R^- are dual to their satisfaction counterparts. On the other hand, the violation rules for the temporal operators \mathcal{S}_I and \mathcal{U}_I are derived by negating and rewriting their semantics. Consider \mathcal{S}_I with $I = [l, r]$:

$$i \not\models \alpha \mathcal{S}_I \beta \Leftrightarrow (\tau_i - \tau_0 \geq l \wedge \exists j \in [E_i^p(I), i]. j \not\models \alpha \wedge \forall k \in [j, L_i^p(I)]. k \not\models \beta) \vee (\tau_i - \tau_0 \geq l \wedge \forall k \in [E_i^p(I), L_i^p(I)]. k \not\models \beta) \vee \tau_i - \tau_0 < l$$

The rules \mathcal{S}^- , \mathcal{S}_∞^- and $\mathcal{S}_{<l}^-$ correspond to the three above disjuncts, respectively. The condition $\tau_i - \tau_0 < l$ is a special case at the beginning of the trace where the interval would be positioned before the start of the trace. In the first disjunct, α is violated at some time-point after the interval starts and β is violated from that time-point until the interval ends. Indeed, the violation proof $j \vdash^- \alpha$ is enough to dismiss all previous occurrences of a satisfaction of β . Also, one can easily notice that if $l \neq 0$, i.e., if the interval does not include the current time-point, then α may be violated somewhere between the end of the interval and the current time-point. Both cases are shown in Figure 3(a), where $\bar{\varphi}$ denotes a violation of φ . In the second disjunct, β is violated at every time-point inside the interval, and the third disjunct captures the trivial case when the interval is located before the first time-point. These cases are shown in Figure 3(b). Next, consider \mathcal{U}_I :

$$i \not\models \alpha \mathcal{U}_I \beta \Leftrightarrow (\exists j \in [i, L_i^f(I)]. j \not\models \alpha \wedge \forall k \in [E_i^f(I), j]. k \not\models \beta) \vee (\forall k \in [E_i^f(I), L_i^f(I)]. k \not\models \beta)$$

The rules \mathcal{U}^- and \mathcal{U}_∞^- correspond to the two above disjuncts, respectively. In the first disjunct, β is violated from the interval start until a time-point j at which also α is violated. Symmetrically to \mathcal{S}^- , we can dismiss all satisfactions of β after j because of the violation proof $j \vdash^- \alpha$. In the second disjunct, β is violated at every time-point inside the interval.

Theorem 1. Let $\rho = \langle (\pi_i, \tau_i) \rangle_{i \in \mathbb{N}}$ be an arbitrary trace. For any formula φ and $i \in \mathbb{N}$, we have $i \vdash^+ \varphi$ iff $i \models \varphi$ and $i \vdash^- \varphi$ iff $i \not\models \varphi$, i.e., the proof system is sound and complete.

In other words, proof trees in our proof system contain all the information needed to explain why a formula has been satisfied or violated on a given trace.

Example 1. Let $\rho = \langle (\{a, b, c\}, 1), (\{a, b\}, 3), (\{a, b\}, 3), (\{\cdot\}, 3), (\{a\}, 3), (\{a\}, 4) \rangle$ and $\varphi = a \mathcal{S}_{[1,2]} (b \wedge c)$. A proof of $5 \not\models \varphi$ has the following form:

$$\frac{\frac{a \notin \{\cdot\}}{3 \vdash^- a} ap^- \quad \frac{\frac{b \notin \{\cdot\}}{3 \vdash^- b} ap^- \quad \frac{b \notin \{a\}}{4 \vdash^- b} ap^-}{3 \vdash^- b \wedge c} \wedge_L^- \quad \frac{4 \vdash^- b \wedge c}{4 \vdash^- b \wedge c} \wedge_L^-}{5 \vdash^- a \mathcal{S}_{[1,2]}(b \wedge c)} \mathcal{S}^-$$

Only events with time-stamp 3 in the trace satisfy the interval conditions. Hence, the portion of the trace we are interested in is $\langle (\{a, b\}, 3), (\{a, b\}, 3), (\{\cdot\}, 3), (\{a\}, 3) \rangle$ and $E_i^p(I) = 1$ and $L_i^p(I) = 4$. Note that a is only violated at time-point 3, so our proof includes a witness that $3 \vdash^- a$. Moreover, from time-point 3 until the interval end (time-point 4), we provide witnesses that $b \wedge c$ is violated, which are $3 \vdash^- b$ and $4 \vdash^- b$.

3 Proof Objects

To make proofs from our proof system explicit, we define an inductive syntax for satisfaction (sp) and violation (vp) proofs and call this representation *proof objects*. Proof objects allow us to easily compute with, modify and compare the size of proof trees. From now on, the term proof will be used for both proof tree and proof object.

$$\begin{aligned} \mathsf{sp} &= ap^+(\mathbb{N}, \Sigma) \mid \neg^+(\mathsf{vp}) \mid \vee_L^+(\mathsf{sp}) \mid \vee_R^+(\mathsf{sp}) \mid \wedge^+(\mathsf{sp}, \mathsf{sp}) \mid \bullet^+(\mathsf{sp}) \mid \circ^+(\mathsf{sp}) \\ &\mid \mathcal{S}^+(\mathsf{sp}, \overline{\mathsf{sp}}_\emptyset) \mid \mathcal{U}^+(\mathsf{sp}, \overline{\mathsf{sp}}_\emptyset) \\ \mathsf{vp} &= ap^-(\mathbb{N}, \Sigma) \mid \neg^-(\mathsf{sp}) \mid \vee_L^-(\mathsf{vp}, \mathsf{vp}) \mid \wedge_L^-(\mathsf{vp}) \mid \wedge_R^-(\mathsf{vp}) \mid \bullet^-(\mathsf{vp}) \mid \bullet_{<I}^-(\mathbb{N}) \\ &\mid \bullet_{>I}^-(\mathbb{N}) \mid \bullet_\emptyset^- \mid \circ_{<I}^-(\mathsf{vp}) \mid \circ_{>I}^-(\mathbb{N}) \mid \mathcal{S}_{<I}^-(\mathbb{N}) \mid \mathcal{S}^-(\mathbb{N}, \mathsf{vp}, \overline{\mathsf{vp}}_\emptyset) \\ &\mid \mathcal{S}_\infty^-(\mathbb{N}, \overline{\mathsf{vp}}_\emptyset) \mid \mathcal{U}^-(\mathbb{N}, \mathsf{vp}, \overline{\mathsf{vp}}_\emptyset) \mid \mathcal{U}_\infty^-(\mathbb{N}, \overline{\mathsf{vp}}_\emptyset) \end{aligned}$$

Here, $\overline{\mathsf{sp}}$ and $\overline{\mathsf{vp}}$ denote finite non-empty sequences of sp and vp subproofs and $\overline{\mathsf{sp}}_\emptyset$ and $\overline{\mathsf{vp}}_\emptyset$ denote finite possibly empty sequences of sp and vp subproofs. We define $\mathsf{p} = \mathsf{sp} \uplus \mathsf{vp}$ to be the disjoint union of satisfaction and violation proofs. Given a proof $p \in \mathsf{p}$, we define $\mathbb{V}(p)$ to be \top if $p \in \mathsf{sp}$ and \perp if $p \in \mathsf{vp}$. Each constructor corresponds to a rule in our proof system. For instance, the proof object representing the proof tree from Example 1 is $P_1 = \mathcal{S}^-(5, ap^-(3, a), [\wedge_L^-(ap^-(3, b)), \wedge_L^-(ap^-(4, b))])$ and we have $\mathbb{V}(P_1) = \perp$.

Each proof p has an associated time-point $\mathsf{tp}(p)$ for which it witnesses the satisfaction or violation. In some cases, $\mathsf{tp}(p)$ can be computed recursively from p 's subproofs. For example, $\mathsf{tp}(\mathcal{S}^+(p, [q_1, \dots, q_n]))$ is $\mathsf{tp}(q_n)$ if $n > 0$ and $\mathsf{tp}(p)$ otherwise. Similarly, $\mathsf{tp}(\mathcal{U}^+(p, [q_1, \dots, q_n]))$ is $\mathsf{tp}(q_1)$ if $n > 0$ and $\mathsf{tp}(p)$ otherwise. Other cases, namely ap^+ , ap^- , $\bullet_{<I}^-$, $\bullet_{>I}^-$, \bullet_\emptyset^- , $\circ_{<I}^-$, $\circ_{>I}^-$, $\mathcal{S}_{<I}^-$, \mathcal{S}^- , and \mathcal{S}_∞^- , explicitly store the associated time-points as an argument of type \mathbb{N} because we cannot compute them from the respective subproofs. For example, $\mathsf{tp}(ap^+(j, a)) = j$ and $\mathsf{tp}(\mathcal{S}^-(j, q, [p_1, \dots, p_n])) = j$.

Given a trace ρ , a formula φ , and a proof p , if the constructors in the proof match the constructors in the formula and all atomic subproofs ap^+ (and ap^-) are (not) contained in the trace at the specified time-points, we say that a proof p for φ is valid at $\mathsf{tp}(p)$. We use the notation $p \vdash \varphi$, once again leaving the dependency on ρ implicit. It is important to highlight that p is not necessarily unique, i.e., multiple valid proofs may exist for a particular time-point i and formula φ . In Example 1, we could have argued differently, using the fact that c is violated at all time-points inside the interval. Then, we can use \mathcal{S}_∞^- and construct the proof $P_2 = \mathcal{S}_\infty^-(5, [\wedge_R^-(ap^-(1, c)), \wedge_R^-(ap^-(2, c)), \wedge_R^-(ap^-(3, c)), \wedge_R^-(ap^-(4, c))])$, which is also a valid proof at $\mathsf{tp}(P_2) = 5$.

$ ap^+(j, a) _w = w(a)$	$ ap^-(j, 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$
$ \bullet^+(p) _w = 1 + p _w$	$ \bullet^-(p) _w = 1 + p _w$
$ \circ^+(p) _w = 1 + p _w$	$ \circ^-(p) _w = 1 + p _w$
$ \bullet_{<I}^-(j) _w = 1$	$ \circ_{<I}^-(j) _w = 1$
$ \bullet_{>I}^-(j) _w = 1$	$ \circ_{>I}^-(j) _w = 1$
$ \bullet_{\emptyset}^-(j) _w = 1$	$ \mathcal{S}_{<I}^-(j) _w = 1$
$ \mathcal{S}^+(p, \bar{q}) _w = 1 + p _w + \sum_{i=1}^n q_i _w$	$ \mathcal{S}^-(p, \bar{q}) _w = 1 + p _w + \sum_{i=1}^n q_i _w$
$ \mathcal{U}^+(p, \bar{q}) _w = 1 + p _w + \sum_{i=1}^n q_i _w$	$ \mathcal{U}^-(p, \bar{q}) _w = 1 + p _w + \sum_{i=1}^n q_i _w$
$ \mathcal{S}_{\infty}^-(j, \bar{q}) _w = 1 + \sum_{i=1}^n q_i _w$	$ \mathcal{U}_{\infty}^-(j, \bar{q}) _w = 1 + \sum_{i=1}^n q_i _w$

Fig. 4: Weighted size measure

In addition, $P_3 = \mathcal{S}^-(5, ap^-(3, a), [\wedge_L^-(ap^-(3, c)), \wedge_L^-(ap^-(4, c))])$ is another valid proof at $\text{tp}(P_3) = 5$. It has the same structure as P_1 , but instead of using the violations of b as witnesses for time-points 3 and 4, it uses the violations of c . In fact, since b and c are both violated at time-points 3 and 4, we can use either violation to justify the violations of $b \wedge c$.

We now compare P_1 , P_2 and P_3 , and discuss their differences. Starting with P_2 , this proof uses \mathcal{S}_{∞}^- , so we must store a witness of the violation of $b \wedge c$ for each one of the 4 time-points inside the interval. This is clearly the longest proof amongst the three. Alternatively, P_1 and P_3 use \mathcal{S}^- , taking advantage of the fact that the violation proof $3 \vdash^- a$ allows us to dismiss both $1 \vdash^+ a$ and $2 \vdash^+ a$. To compare them, we informally define the size of a proof p to be the number of proof object constructors occurring in p . Hence, $|P_1| = |P_3| = 6$ and $|P_2| = 9$. We are often interested in smaller proofs as they tend to be easier to understand, so we can discard P_2 . However, since P_1 and P_3 share the same structure and thus have the same size, we can not yet determine which one is preferable over the other. To distinguish the two, we extend our notion of *size* by assigning a weight to each one of the atomic propositions. Formally, let w be a function such that each atomic proposition a has a corresponding natural number $w(a)$ representing its weight. The *weighted size* $|-|_w :: \mathbf{p} \rightarrow \mathbb{N}$ of a proof is then defined in Figure 4.

We continue our example setting $w(a) = 1$, $w(b) = 2$ and $w(c) = 3$. Since the number of non- ap constructors in both P_1 and P_3 is 3, and the number of occurrences of a and b (correspondingly c in P_3) are 1 and 2, respectively, one can easily calculate $|P_1|_w = 8$ and $|P_3|_w = 10$. Moreover, $|P_2|_w = 17$. Assuming that proofs with smaller weighted size are preferable, P_1 wins. Thus, the importance of an atomic proposition is inversely proportional to its weight. In our example, we assign a lower value to b , so proofs using witnesses of b are preferable over proofs using witnesses of c . We also observe that coming up with reasonable weights is not straightforward if we still want to take the number of constructors into account. For instance, if we had $w(b) = 10$ and $w(c) = 50$, the number of constructors is nearly irrelevant in the resulting weighted sizes $|P_1|_w = 24$ and $|P_3|_w = 104$. A conservative approach is to initially assign all weights to 1 and after analyzing the structure of the proofs, one can tune the weights accordingly. To simplify our exposition, from now on we will consider all weights being assigned to 1, even though our algorithm supports arbitrary weights.

```

type proof = sproof | vproof
type buf = proof list  $\times$  proof list    type buft = proof list  $\times$  proof list  $\times$  ((ts  $\times$  tp) list)
type saux = { ts_zero : ts option, ts_tp_in : (ts  $\times$  tp) list, ts_tp_out : (ts  $\times$  tp) list,
              s_beta_alphas_in : (ts  $\times$  proof) slist, s_beta_alphas_out : (ts  $\times$  proof) list,
              v_alpha_betas_in : (ts  $\times$  proof) slist, v_alphas_out : (ts  $\times$  vproof) slist,
              v_betas_in : (ts  $\times$  vproof) list,
              v_alphas_betas_out : (ts  $\times$  vproof option  $\times$  vproof option) list }
type state = PredS string | NegS state | AndS state state buf | OrS state state buf
            | PrevS  $\mathcal{I}$  state bool proof (ts list) | NextS  $\mathcal{I}$  state bool (ts list)
            | SinceS  $\mathcal{I}$  state state buft saux | UntilS  $\mathcal{I}$  state state buft uaux
function init :: formula  $\Rightarrow$  state
function eval :: ts  $\times$  tp  $\Rightarrow$  atom set  $\Rightarrow$  state  $\Rightarrow$  proof list  $\times$  state

```

Fig. 5: Types of the monitor's state and evaluation functions

4 Computing Minimal Proofs

In this section, we describe our online monitoring algorithm. Given an MTL formula φ , our monitor incrementally processes a trace and for each time-point i it outputs a minimal proof of the satisfaction or violation of φ at i . The algorithm constructs this minimal proof of φ by combining minimal proofs of φ 's immediate subformulas. To do this efficiently, the monitor maintains just enough information about the trace in its state so that it can guarantee to output minimal proofs. In case the monitored formula includes (bounded) future operators, the monitor's output may be delayed. Moreover, a single event may trigger the output of multiple such delayed verdicts (i.e., proofs) at once.

Figure 5 shows the types of our algorithm's main functions `init`, which computes the algorithm's initial state, and `eval`, which processes a time-stamped event while updating the monitor's state and producing a list of minimal proofs (satisfactions or violations) for an in-order (potentially empty) sequence of time-points. Our monitor's state (also shown in Figure 5) has the same tree-like structure as the monitored MTL formula. Additionally, it stores operator-specific information for each Boolean and temporal operator. For example for $\alpha S_I \beta$, we store the interval I , the states of the subformulas α and β , a buffer `buft` for proofs (and associated time-stamps) coming from the recursive evaluation of subformulas and the operator-specific data structures `saux`. The overall structure of our monitor is modeled after VeriMon [21], which has a similar interface (`init` and `eval`) and `state` type including the used buffers `buf` and `buft`. The main novelty is our design of the `saux` and `uaux` data structure, which store sufficient information to compute minimal proofs for \mathcal{S} and \mathcal{U} . For simplicity and space limitations, we only describe `saux` in detail.

The data structure `saux` for a since formula $\varphi = \alpha S_I \beta$ is a record consisting of nine fields, which we describe next relative to a current timepoint cur at which we wish to evaluate φ . Field `ts_zero` is \perp in the initial state and stores the first time-stamp $\lfloor \tau_0 \rfloor$ for all following steps. Fields `ts_tp_in` and `ts_tp_out` store lists of time-stamp-time-point pairs inside the interval (between $E_{cur}^p(I)$ and $L_{cur}^p(I)$) and after the interval (between $L_{cur}^p(I) + 1$ and cur), respectively. The other fields store satisfaction (prefix `s_`) or violation (`v_`) proofs. Specifically, `s_beta_alphas_in` stores \mathcal{S}^+ proofs inside and `s_beta_alphas_out` stores \mathcal{S}^+ after the interval, respectively. This is not the only distinction these two fields have:

$s_beta_alphas_{in}$ has type *slist*, which are plain lists but indicates a sortedness (with respect to size) invariant on stored proofs that we maintain to optimize the number of proofs we must store: if a proof enters the interval, we can delete all larger proofs that entered the interval prior to it. Moreover, we can quickly access the first proof of this list which necessarily has minimal size. On the other hand, $s_beta_alphas_{out}$ must store all proofs because it is not possible to predict when and which of these proofs will enter the interval.

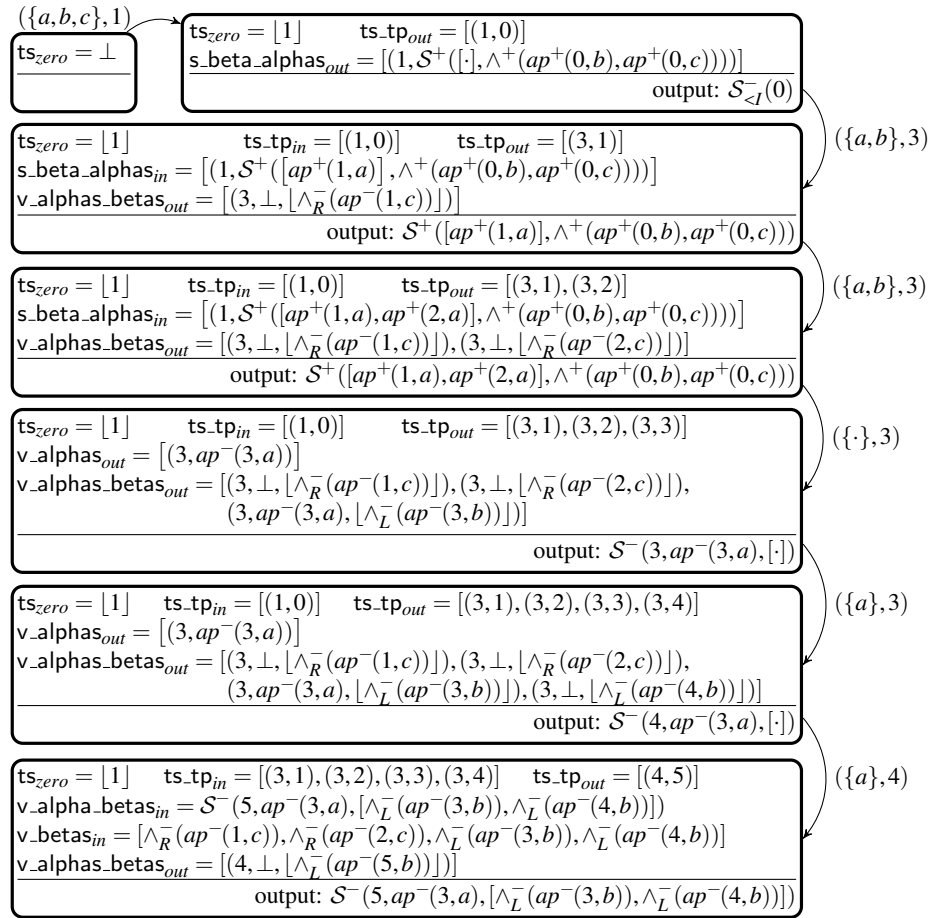
Furthermore, $v_alpha_betas_{in}$ is the (sorted) analogue of $s_beta_alphas_{in}$ for \mathcal{S}^- proofs with the violation of α inside the interval. Recall that \mathcal{S}^- also allow a single violation proof of α to occur after the interval. Moreover, \mathcal{S}_∞^- proofs require that β is violated at all time-points inside the interval. Representing these violations, v_alphas_{out} stores a sorted list of optimal proofs for violations of α after the interval and v_betas_{in} stores a (length-maximal) suffix of β violations inside the interval. Finally, v_alphas_betas stores all α and β violations outside the interval, so all other components that store violation proofs inside the interval can be efficiently updated when the interval shifts.

To output a minimal proof from this state, we perform a case distinction. If list $s_beta_alphas_{in}$ is non-empty then its head must be a minimal satisfaction proof. Otherwise, the formula is violated and the minimal violation proof is either the head of $v_alpha_betas_{in}$ or the head of v_alphas_{out} (after adding a \mathcal{S}^- constructor) or the application of \mathcal{S}_∞^- to v_betas_{in} (provided that this suffix spans the entire interval which can be deduced by comparing the length of v_betas_{in} to the length of ts_tp_{in}).

To illustrate how the state is updated, we once again consider the formula and trace introduced in Example 1. Figure 6 shows the *saux* states of our algorithm and the produced minimal proof after processing every event. In every state, we only show the non-empty components. Initially, all components of the state are empty except for ts_zero , which is \perp . When the first event $(\{a, b, c\}, 1)$ arrives, the list ts_tp_{out} is updated accordingly and a pair with time-stamp 1 and a \mathcal{S}^+ proof using the satisfactions of b and c is added to $s_beta_alphas_{out}$. Obviously, this proof is not valid for the current time-point 0, considering that the interval $[1, 2]$ has not yet started, so the output of the monitor is the trivial proof $\mathcal{S}_{<I}^-(0)$. The time-stamp of the first event moves inside the interval when the second event $(\{a, b\}, 3)$ arrives. Both ts_tp_{out} and ts_tp_{in} are updated accordingly. Furthermore, the algorithm extends the \mathcal{S}^+ proof previously stored in $s_beta_alphas_{out}$ by adding $ap^+(1, a)$ to the list of α satisfactions, and then the resulting proof is moved to $s_beta_alphas_{in}$. The algorithm also appends the proof $ap^-(1, c)$ to $v_alphas_betas_{out}$. Because $s_beta_alphas_{in}$ is not empty, the monitor outputs the first proof of this list.

In the next step, event $(\{a, b\}, 3)$ arrives and the monitor proceeds similarly, adding the proof $ap^+(2, a)$ to the \mathcal{S}^+ proof in $s_beta_alphas_{in}$. Aside from outputting the extended satisfaction proof, the algorithm also adds the proof $ap^-(2, c)$ to $v_alphas_betas_{out}$. When event $(\{\cdot\}, 3)$ arrives, the sequence of a satisfactions comes to an end, which indicate that the proofs in $s_beta_alphas_{in}$ and $s_beta_alphas_{out}$ are no longer valid nor useful. Hence, we clear both lists. Moreover, the proof $ap^-(3, a)$ is stored in v_alphas_{out} , since the a violation is after the interval. This subproof is appended to $v_alphas_betas_{out}$ along with the violation of the conjunction \wedge_L^- . The algorithm then proceeds to construct a violation proof $\mathcal{S}^-(3, ap^-(3, a), [\cdot])$ using the subproof stored in v_alphas_{out} and outputs it.

When $(\{a\}, 3)$ arrives, the algorithm appends the proof \wedge_L^- to $v_alphas_betas_{out}$ and again uses the same subproof stored in v_alphas_{out} to construct $\mathcal{S}^-(4, ap^-(3, a), [\cdot])$. Evi-

Fig. 6: The monitor's *saux* states when executing Example 1

dently, this proof has an associated time-point of 4, which is the only distinction from the last proof that the monitor output. Finally, when the last event $(\{a\}, 4)$ arrives, the interval shifts and ts_tp_in and ts_tp_out change accordingly. At this stage, the algorithm populates $v_alpha_betas_in$ and v_betas_in with the subproofs stored in $v_alphas_betas_out$. In particular, it constructs and stores the proof $S^-(5, ap^-(3,a), [\wedge_L^-(ap^-(3,b)), \wedge_L^-(ap^-(4,b))])$ in $v_alpha_betas_in$. Moreover, the algorithm stores a sequence of violations of the conjunction inside the interval in v_betas_in . This sequence of violations fills the entire interval, so it is then used to construct the proof $S^-_\infty(5, [\wedge_R^-(ap^-(1,c)), \wedge_R^-(ap^-(2,c)), \wedge_R^-(ap^-(3,c)), \wedge_R^-(ap^-(4,c))])$. The S^- proof corresponds precisely to the proof tree presented in Example 1, and the proof object P_1 in Section 3, whereas the S^-_∞ proof corresponds to the proof object P_2 . Lastly, the size of these two proofs is computed by the algorithm and the S^- proof is selected because it is smaller.

We now provide a formal description of the invariant we maintain for *saux*. While we do not show the implementation of our state update, we contend that the invariant

$$\begin{aligned}
 & \text{sorted}(\text{s_beta_alphas}_{in}) \wedge \text{sorted}(\text{v_alpha_betas}_{in}) \wedge \text{sorted}(\text{v_alphas}_{out}) \wedge \\
 (1) \quad & \forall (\tau, u) \in \text{s_beta_alphas}_{in}. \quad \exists p \bar{q}. u = \mathcal{S}^+(p, \bar{q}) \wedge u \vdash \alpha \mathcal{S}_I \beta \wedge \text{tp}(u) = \text{cur} \wedge \tau = \text{ts}(p) \\
 (2) \quad & \forall (\tau, u) \in \text{s_beta_alphas}_{out}. \quad \exists p \bar{q}. u = \mathcal{S}^+(p, \bar{q}) \wedge u \vdash \alpha \mathcal{S} \beta \wedge \text{tp}(u) = \text{cur} \wedge \tau = \text{ts}(p) \\
 (3) \quad & \forall (\tau, u) \in \text{v_alpha_betas}_{in}. \quad \exists p \bar{q}. u = \mathcal{S}^-(\text{cur}, p, \bar{q}) \wedge u \vdash \alpha \mathcal{S}_I \beta \wedge \tau = \text{ts}(p) \\
 (4) \quad & \forall (\tau, p) \in \text{v_alphas}_{out}. \quad \mathcal{S}^-(\text{cur}, p, []) \vdash \alpha \mathcal{S}_I \beta \wedge \tau = \text{ts}(p) \\
 (5) \quad & \forall (\tau, p) \in \text{v_betas_suffix}_{in}. \quad E_{cur}^p(I) \leq \text{tp}(p) \leq L_{cur}^p(I) \wedge p \vdash \beta \wedge \neg \mathbb{V}(p) \wedge \tau = \text{ts}(p) \\
 (6) \quad & \forall (\tau, p^*, q^*) \in \text{v_alphas_betas}_{out}. \exists i \in [L_{cur}^p(I), \text{cur}]. \tau = \tau_i \wedge \\
 & (p^* = \perp \vee (\exists p. \neg \mathbb{V}(p) \wedge p^* = [p] \wedge p \vdash \alpha)) \wedge (q^* = \perp \vee (\exists q. \neg \mathbb{V}(q) \wedge q^* = [q] \wedge q \vdash \beta))
 \end{aligned}$$

Fig. 7: The algorithm's invariant (soundness)

is sufficient to reconstruct it. We write $\text{ts}(p)$ for the time-stamp associated with a proof, i.e., the time-stamp $\tau_{\text{tp}(p)}$ of the associated time-point $\text{tp}(p)$. We also use the functional programming notations like λ -abstractions and the list map function. Moreover, we define the predicate $\text{sorted}(seq) := (\forall (\tau_i, p_i), (\tau_j, p_j) \in seq. (i < j) \wedge (j < \text{length}(seq)) \implies \tau_i \leq \tau_j \wedge |p_i| \leq |p_j|)$ over a sequence of pairs of time-stamps and proofs and assume that every sequence below is monotone with respect to time-stamps ($i < j$ implies $\tau_i \leq \tau_j$).

The fields ts_{zero} , ts_tp_{in} and ts_tp_{out} are characterized as follows:

$$\text{ts}_{zero} = \begin{cases} \perp & \text{iff } \text{cur} = -1 \\ [\tau_0] & \text{iff } \text{cur} \geq 0 \end{cases} \quad \begin{aligned} \text{ts_tp}_{in} &= \text{map}(\lambda i. (\tau_i, i)) [E_{cur}^p(I), L_{cur}^p(I)] \\ \text{ts_tp}_{out} &= \text{map}(\lambda i. (\tau_i, i)) [L_{cur}^p(I), \text{cur}] \end{aligned}$$

The desired properties of the objects stored in other fields are given in Figure 7.

We go through each one of the invariant's statements. In (1) a proof in $\text{s_beta_alphas}_{in}$ (which must be sorted) must have form $\mathcal{S}^+(p, \bar{q})$ and be a valid proof of $\alpha \mathcal{S}_I \beta$ at the current time-point, with time-stamp $\text{ts}(p)$. Next, in (2) we require proofs to have the same form but instead be valid for a modified formula without the interval I . In this case we can relax the timing constraint because these proofs will only be valid at a later time-point, specifically once $\text{ts}(p)$ moves inside the interval. The statement (3) is precisely the same as (1), but for \mathcal{S}^- proofs. Moreover, in statement (4) each proof p in v_alphas_{out} (which must too be sorted) must be a valid subproof of a \mathcal{S}^- proof at the current time-point with time-stamp $\text{ts}(p)$. In (5), each subproof corresponding to the violation of β must be inside the interval with time-stamp $\text{ts}(p)$. The statement (6) specifies that outside the interval there is either a subproof of a violation of α or β or there are no such proofs. These requirements formalize the soundness of *saux*: what must hold for the things stored in the data structure? We only briefly consider completeness on the example of $\text{s_beta_alphas}_{in}$, which answers the question of what must be stored:

$$\begin{aligned}
 & \forall p \bar{q} \tau. \mathcal{S}^+(p, \bar{q}) \vdash \alpha \mathcal{S}_I \beta \wedge \text{tp}(\mathcal{S}^+(p, \bar{q})) = \text{cur} \wedge \tau = \text{ts}(p) \implies \\
 & (\exists p' \bar{q}' \tau'. |\mathcal{S}^+(p', \bar{q}')| \leq |\mathcal{S}^+(p, \bar{q})| \wedge \mathcal{S}^+(p', \bar{q}') \vdash \alpha \mathcal{S}_I \beta \wedge \tau' = \text{ts}(p') \wedge \\
 & \tau' \geq \tau \wedge \text{tp}(\mathcal{S}^+(p', \bar{q}')) = \text{tp}(\mathcal{S}^+(p, \bar{q})) \wedge (\tau', \mathcal{S}^+(p', \bar{q}')) \in \text{s_beta_alphas}_{in})
 \end{aligned}$$

In other words, considering all \mathcal{S}^+ proofs valid for $\varphi = \alpha \mathcal{S}_I \beta$ at the current time-point cur , we must store in $\text{s_beta_alphas}_{in}$ another proof at most as large and old, that is also valid for φ at cur . The other fields have similar completeness statements. Taken together soundness and completeness guarantee that given an arbitrary formula and trace, our online monitoring algorithm will always output a minimal proof at every time-point.

a	b	c	TP	TS	$a \mathcal{S}_{[1,2]} (b \wedge c)$	a	$b \wedge c$	b	c
✓	✓	✓	0	1	✗		●	●	●
✓	✓	✗	1	3	✓	●			
✓	✓	✗	2	3	✓	●			
✗	✗	✗	3	3	✗	✗	✗	●	
✓	✗	✗	4	3	✗		✗	●	
✓	✗	✗	5	5	✗				

Fig. 8: Visualization of Example 1

5 Implementation

We implement our algorithm in a new tool called EXPLANATOR2 [17]. The implementation amounts to around 3 000 lines of OCaml. In addition, a 3 500 lines long OCaml program is extracted from our Isabelle formalization consisting of 10 000 lines of definitions and proofs. The extracted program contains the proof object validity checker in form of a function $\text{is_valid} : \text{trace} \rightarrow \text{formula} \rightarrow \text{proof} \rightarrow \text{bool}$, which effectively implements what we denote by $p \vdash \varphi$. Moreover, it also contains the minimality checker $\text{is_optimal} : (\text{atom} \rightarrow \text{nat}) \rightarrow \text{trace} \rightarrow \text{formula} \rightarrow \text{proof} \rightarrow \text{bool}$ that given the weights $w : \text{atom} \rightarrow \text{nat}$ for the atomic propositions, a trace ρ , a formula φ , and a proof p computes a proof q for φ on ρ at time-point $i(p)$ with a minimal the weighted size using a verified algorithm based on dynamic programming and then checks that $|p|_w \leq |q|_w$. Note that q may differ from p because minimal proof objects are not unique. We refer to Herasimau’s B.Sc. thesis [13] for more details on the formalization and the dynamic programming algorithm. We employed the verified validity and minimality checkers to thoroughly test our unverified algorithm. Moreover our tool includes a command line option to enable the verified certification of its output, which slows down the computation but increases the trustworthiness.

EXPLANATOR2 also includes a JavaScript web front end. To this end, we transpile the compiled OCaml bytecode of EXPLANATOR2 to JavaScript using `Js_of_ocaml` [27]. The resulting JavaScript library allows us to run EXPLANATOR2 in any web browser. We augment the library with an interactive visualization using React [14]. Figure 8 shows the visualization of our Example 1. On the left, the visualization shows the trace (from top to bottom) consisting of the atomic propositions (columns a, b, and c) and the time-stamps (column TS). The following columns show the different subformulas of our monitored MTL formula $\varphi = a \mathcal{S}_{[1,2]} (b \wedge c)$. The column labeled with φ itself shows the Boolean verdicts that a traditional monitor would output. EXPLANATOR2 allows its users to further inspect the Boolean verdicts by clicking on them. For example, Figure 8 shows the state of the visualization after clicking on the φ ’s violation at time-point 5. The visualization highlights the time interval associated with the inspected formula and time-point. Furthermore, it shows the relevant violations of φ ’s subformulas a and $b \wedge c$: the subformula a is violated at time-point 3 and $b \wedge c$ is violated at time-points 3 and 4, which corresponds to a valid \mathcal{S}^- proof. The user could continue the exploration by

further clicking on the two $b \wedge c$ violations to find out that the tool used b violations to justify both. The visualization uses black circles to denote combinations of subformula and time-stamp that are relevant for at least one of φ 's verdicts. The Boolean value for these relevant subexplanations is only revealed upon exploration.

6 Examples

Here we demonstrate how the minimal proofs produced by our monitor can be useful when trying to comprehend a satisfaction or violation of an MTL formula. We consider Timescales [25], a benchmark generator for MTL monitoring tools. It produces temporal patterns that match preset MTL formulas. These patterns are suitable because they commonly occur in real system designs. In the generated traces, the time-stamps are equal to their corresponding time-points, and they start at 0. Here, we selected the two most complex properties and generated their corresponding traces. At the end of both traces there is a violation of the pattern, and we use the proofs to explain these violations. Even though we explicitly mention the traces in each pattern, all proof trees produced by our proof system contain all necessary information about the trace. We mention the traces here for completeness because some non-temporal subproofs were omitted. Furthermore, our algorithm only supports the operators presented in Figure 2, so we use logical equivalences to extend our syntax. Thus, our monitor also accepts formulas with the following operators: \top (truth), \perp (falsity), \rightarrow (implies), \leftrightarrow (iff), \mathcal{R}_I (release), \mathcal{T}_I (trigger), \blacksquare_I (historically), \square_I (always), \blacklozenge_I (once) and \blacklozenge_I (eventually). Note that if the formula has one of these operators, the structure of the proofs will correspond to the equivalent formula, not the original one. In the long run, we want to support all these operators as first class citizens in our proof system and thus also in our algorithm.

Bounded Recurrence Between q and r . The bounded recurrence property specifies the following pattern: between events q and r there is at least one occurrence of event p every u time units. In MTL, this pattern is captured by the formula $\varphi_1 = (r \wedge \neg q \wedge \blacklozenge q) \rightarrow (\blacklozenge_{[0,u]} (p \vee q) \mathcal{S} q)$. We set the bound $u = 3$. Moreover, we consider the trace $\langle (\dots, \{q\}, 56), (\{\cdot\}, 57), (\{\cdot\}, 58), (\{\cdot\}, 59), (\{\cdot\}, 60), (\{r\}, 61) \rangle$. This is the portion of the trace pertinent to the proof. Note that this formula has operators that are not directly supported by our algorithm, so $\alpha \rightarrow \beta$ is rewritten to $\neg \alpha \vee \beta$ and $\blacklozenge_I \beta$ is rewritten to $\top \mathcal{S}_I \beta$. Strictly speaking, also \top should be rewritten to $a \vee \neg a$, but simplifying notation we pretend that \top is natively supported with the expected unconditional satisfaction rule. The formula φ_1 is violated at $\text{tp}(\varphi_1) = 61$ and the proof is shown in Figure 9.

To prove the violation of the disjunction, the topmost operator of the formula, both disjuncts must be violated. For this reason, two subproofs are constructed, one for the left disjunct and one for the right disjunct. In the left subproof, we can see that the left disjunct is violated because, after getting rid of the negation, both conjuncts $r \wedge \neg q$ and $\top \mathcal{S} q$ are satisfied at time-point 61. This part of the formula enforces that: (i) at the current time-point q is satisfied or r is satisfied (and q is not satisfied); (ii) q was satisfied at some point in the past. Note that (ii) corresponds exactly to $\blacklozenge q$. In the subproof, we have $61 \vdash^+ r \wedge \neg q$ because r is satisfied and q is violated at time-point 61. Moreover, the proof $61 \vdash^+ \top \mathcal{S} q$ uses the fact that q was satisfied at time-point 56, which is when the

$$\frac{\frac{\frac{\vdots}{61 \vdash^+ r \wedge \neg q} \wedge^+ \quad \frac{\frac{\frac{\vdots}{56 \vdash^+ q} \quad \frac{57, \dots, 61 \vdash^+ \top}{61 \vdash^+ \top \mathcal{S} q}}{\top^+} \mathcal{S}^+}{61 \vdash^+ (r \wedge \neg q) \wedge (\top \mathcal{S} q)} \wedge^+ \quad \frac{\frac{\frac{\vdots}{58, \dots, 61 \vdash^- p \vee q} \vee^- \quad \frac{q \notin \{r\}}{61 \vdash^- q} ap^-}{61 \vdash^- \top \mathcal{S}_{[0,3]} (p \vee q)} \mathcal{S}_\infty^-}{61 \vdash^- (\top \mathcal{S}_{[0,3]} (p \vee q)) \mathcal{S} q} \mathcal{S}^-}{61 \vdash^- \neg((r \wedge \neg q) \wedge (\top \mathcal{S} q)) \vee ((\top \mathcal{S}_{[0,3]} (p \vee q)) \mathcal{S} q)} \vee^-$$

Fig. 9: Proof of the violation of φ_1 at $\text{tp}(\varphi_1) = 61$

$$\frac{\frac{\frac{\frac{\vdots}{p \in \{p\}}}{59 \vdash^+ p} ap^+ \quad \frac{\frac{\frac{\vdots}{60, \dots, 62 \vdash^- s}}{60, \dots, 62 \vdash^+ \neg s} \neg^+}{62 \vdash^+ \neg s \mathcal{S}_{[3, \infty)} p} \mathcal{S}^+}{62 \vdash^- \neg(\neg s \mathcal{S}_{[3, \infty)} p)} \neg^-}{\frac{62 \vdash^- (\neg s \vee (\top \mathcal{S}_{[0,3]} p)) \wedge \neg(\neg s \mathcal{S}_{[3, \infty)} p)}{\wedge_R} \quad \frac{\vdots}{62, \dots, 64 \vdash^- q} \mathcal{S}^-}{\frac{\phi}{\vdots} \quad \frac{64 \vdash^- ((\neg s \vee (\top \mathcal{S}_{[0,3]} p)) \wedge \neg(\neg s \mathcal{S}_{[3, \infty)} p)) \mathcal{S} q}{\mathcal{S} q}} \vee^-$$

Fig. 10: Proof of the violation of φ_2 at $\text{tp}(\varphi_2) = 64$

last q had arrived. Moving to the subproof of the right disjunct $(\top \mathcal{S}_{[0,3]} (p \vee q)) \mathcal{S} q$, the violation occurs because both subformulas are violated at time-point 61. The subproof $61 \vdash^- \top \mathcal{S}_{[0,3]} (p \vee q)$ uses the violations of p and q in the last 3 time units (58, ..., 61). Again, note that this corresponds precisely to the formula $\Diamond_{[0,3]}(p \vee q)$. Furthermore, the proof $61 \vdash^- q$ indicates that q is not satisfied at the current time-point.

Bounded Response Between q and r . Closely related to the bounded recurrence, the bounded response property specifies the following pattern: between events q and r , event s must respond to event p within the interval $[l, u]$. In MTL, this pattern is specified by the formula $\varphi_2 = (r \wedge \neg q \wedge \Diamond q) \rightarrow ((s \rightarrow \Diamond_{[l, u]} p) \wedge \neg(\neg s \mathcal{S}_{[u, \infty)} p))$. We consider the trace $\langle (\dots, \{q\}, 58), (\{p\}, 59), (\{\cdot\}, 60), (\{\cdot\}, 61), (\{\cdot\}, 62), (\{\cdot\}, 63), (\{r\}, 64) \rangle$. The formula φ_2 is violated at $\text{tp}(\varphi_1) = 64$ and the proof is presented in Figure 10.

The left disjunct of φ_2 is exactly the same as the left disjunct of φ_1 (the formula of the *bounded recurrence* example). Hence, we omit the corresponding subproof because Φ has the exact same structure of the subproof of the *bounded recurrence* example. However, we still describe it. In Φ , after applying the negation rule, the conjunction $r \wedge \neg q$ and $\top \mathcal{S} q$ are satisfied at time-point 64 instead. Also, the subproof $64 \vdash^+ \top \mathcal{S} q$ uses the satisfaction of q at time-point 58 instead. Let $\alpha = ((\neg s \vee (\top \mathcal{S}_{[0,3]} p)) \wedge \neg(\neg s \mathcal{S}_{[3, \infty)} p)) \mathcal{S} q$.

Here, the right disjunct has the form $\alpha \mathcal{S} q$, and it is violated because α is violated at time-point 62, and from this point onward (until the current time-point 64) q is always violated. In particular, proofs in our proof system only need to consider violations of q starting at time-point 62, due to the fact that α is violated at that point. The formula α captures two properties: (i) if there is a response (i.e. s is satisfied) then there must be a corresponding event p (i.e. p must be satisfied in the last 3 time units); (ii) there are no events p without a response s earlier than 3 time units in the past. In this proof, the violation of α is constructed using the violation of the right conjunct, or the violation of (ii). After getting rid of the negation, the proof $62 \vdash^+ \neg s \mathcal{S}_{[3,\infty)} p$ uses the fact that p is satisfied at time-point 59 and that s is violated at time-points 60, 61 and 62. In other words, there was no response s to the event p within the corresponding timing constraint.

7 Conclusion

We have developed an online MTL monitor that outputs detailed verdicts in form of proof trees, which serve as both understandable explanations and checkable certificates. Our monitor incorporates a formally verified checker and an interactive visualization. Overall, we believe that our approach significantly improves the user experience when using an MTL monitor. In particular, the generated explanations provide insight into root causes of violations and can help with specification debugging. Another plausible application of explanations is teaching temporal logics to students and engineers.

As future work, we will focus on more expressive logics: metric dynamic logic for which we have already taken initial steps [30] and metric first-order temporal logic. Moreover, we plan to thoroughly evaluate EXPLANATOR2's performance. We have already observed that EXPLANATOR2 outperforms the verified proof-producing dynamic programming algorithm (which already struggles with logs of a few hundred events) but cannot compete with optimized monitors producing Boolean verdicts like Hydra [19]. Naturally, computing proofs is more costly than computing Booleans: a proof might need to contain the entire trace. Yet, it would be good to understand the worst-case time and space complexity as well as the practical behavior of monitors producing minimal proofs precisely. Finally, we are interested in optimizing other aspects of the proofs than their size.

References

1. Basin, D., Bhatt, B.N., Krstic, S., Traytel, D.: Almost event-rate independent monitoring. *Formal Methods Syst. Des.* **54**(3), 449–478 (2019)
2. Basin, D., Bhatt, B.N., Traytel, D.: Optimal proofs for linear temporal logic on lasso words. In: ATVA 2018. LNCS, vol. 11138, pp. 37–55. Springer (2018)
3. Basin, D., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. *J. ACM* **62**(2), 15:1–15:45 (2015)
4. Basin, D., Klaedtke, F., Zalinescu, E.: Algorithms for monitoring real-time properties. *Acta Informatica* **55**(4), 309–338 (2018)
5. Chattopadhyay, A., Mamouras, K.: A verified online monitor for metric temporal logic with quantitative semantics. In: RV 2020. LNCS, vol. 12399, pp. 383–403. Springer (2020)
6. Chechik, M., Gurfinkel, A.: A framework for counterexample generation and exploration. *Int. J. Softw. Tools Technol. Transf.* **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. Cruz-Filipe, L., Heule, M.J.H., Jr., W.A.H., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: CADE 26. vol. 10395, pp. 220–236. Springer (2017)
9. Dauer, J.C., Finkbeiner, B., Schirmer, S.: Monitoring with verified guarantees. In: RV 2021. LNCS, vol. 12974, pp. 62–80. Springer (2021)
10. Dawes, J.H., Reger, G.: Explaining violations of properties in control-flow temporal logic. In: RV 2019. LNCS, vol. 11757, pp. 202–220. Springer (2019)
11. Finkbeiner, B., Oswald, S., Passing, N., Schwenger, M.: Verified Rust monitors for Lola specifications. In: RV 2020. LNCS, vol. 12399, pp. 431–450. Springer (2020)
12. Francalanza, A., Cini, C.: Computer says no: Verdict explainability for runtime monitors using a local proof system. *J. Log. Algebraic Methods Program.* **119**, 100636 (2021)
13. Herasimau, A.: Formalizing Explanations for Metric Temporal Logic. B.Sc. thesis, ETH Zürich (2020)
14. Hunt, P., O’Shannessy, P., Smith, D., Coatta, T.: React: Facebook’s functional turn on writing javascript. *ACM Queue* **14**(4), 40 (2016)
15. Kane, A., Chowdhury, O., Datta, A., Koopman, P.: A case study on runtime monitoring of an autonomous research vehicle (ARV) system. In: RV 2015. LNCS, vol. 9333, pp. 102–117. Springer (2015)
16. Lammich, P.: Efficient verified (UN)SAT certificate checking. *J. Autom. Reason.* **64**(3), 513–532 (2020)
17. Lima, L., Herasimau, A., Raszyk, M., Traytel, D., Yuan, S.: The development repository of EXPLANATOR2. <https://github.com/runtime-monitoring/explanator2> (2022)
18. Moosbrugger, P., Rozier, K.Y., Schumann, J.: R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. *Formal Methods Syst. Des.* **51**(1), 31–61 (2017)
19. Raszyk, M., Basin, D., Krstic, S., Traytel, D.: Multi-head monitoring of metric temporal logic. In: ATVA 2019. LNCS, vol. 11781, pp. 151–170. Springer (2019)
20. Raszyk, M., Basin, D., Traytel, D.: Multi-head monitoring of metric dynamic logic. In: ATVA 2020. LNCS, vol. 12302, pp. 233–250. Springer (2020)
21. Schneider, J., Basin, D., Krstic, S., Traytel, D.: A formally verified monitor for metric first-order temporal logic. In: RV 2019. LNCS, vol. 11757, pp. 310–328. Springer (2019)
22. Sulzmann, M., Lu, K.Z.M.: POSIX regular expression parsing with derivatives. In: FLOPS 2014. LNCS, vol. 8475, pp. 203–220. Springer (2014)
23. Sulzmann, M., Zechner, A.: Constructive finite trace analysis with linear temporal logic. In: TAP 2012. LNCS, vol. 7305, pp. 132–148. Springer (2012)
24. Ulus, D.: Online monitoring of metric temporal logic using sequential networks. *CoRR abs/1901.00175* (2019)
25. Ulus, D.: Timescales: A benchmark generator for MTL monitoring tools. In: RV 2019. LNCS, vol. 11757, pp. 402–412. Springer (2019)
26. Völlinger, K.: Verifying the output of a distributed algorithm using certification. In: RV 2017. LNCS, vol. 10548, pp. 424–430. Springer (2017)
27. Vouillon, J., Balat, V.: From bytecode to JavaScript: the Js_of_ocaml compiler. *Softw. Pract. Exp.* **44**(8), 951–972 (2014)
28. Wimmer, S., Herbretau, F., van de Pol, J.: Certifying emptiness of timed Büchi automata. In: FORMATS 2020. LNCS, vol. 12288, pp. 58–75. Springer (2020)
29. Wimmer, S., von Mutius, J.: Verified certification of reachability checking for timed automata. In: TACAS 2020. LNCS, vol. 12078, pp. 425–443. Springer (2020)
30. Yuan, S.: Explaining Monitoring Verdicts for Metric Dynamic Logic. B.Sc. thesis, ETH Zürich (2019)