Explainable Online Monitoring of Metric Temporal Logic

Leonardo Lima[1 †], Andrei Herasimau[2], Martin Raszyk[3 †], Dmitriy Traytel[1 †], and Simon Yuan[2]

[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 [3, 19, 24, 27, 33] 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 [15], 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 Basin et al.'s [4] local proof system for LTL satisfiability on lasso words 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 application[1] 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) and empirically evaluate our algorithm's performance in comparison to other monitors (§7). 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 and empirically evaluate an efficient algorithm to compute size-minimal proof objects representing proof trees in our proof system.
- We implement our algorithm in a new, publicly available monitoring tool EXPLANATOR2 [22] that includes a web front end and a formally verified proof object checker.

Related Work. We take the work by Basin et al. [4] on optimal proofs for LTL on lasso words as our starting point but change the setting from lasso words to streams of time-stamped events and the logic from LTL to MTL. Moreover, Basin et al. considered the offline path checking problem, whereas we tackle online monitoring here.

Parts of the work presented here are also described in two B.Sc. theses by Yuan [39] and Herasimau [16]. 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 Basin et al.'s algorithm [4]). 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 [5, 29], which perform much better than dynamic programming algorithms for non-trivial metric interval bounds.

Basin et al.'s approach [4] is parameterized by a comparison relation on proof objects that specifies what the algorithm should optimize for. Yuan [39] discovers a flaw in the correctness claim for Basin et al.'s algorithm and corrects it by further restricting the

[1] https://runtime-monitoring.github.io/explanator2

$$
\begin{array}{ll|l|l}
i \vDash p & \text{iff } p \in \pi_i & i \vDash \alpha \vee \beta \text{ iff } i \vDash \alpha \text{ or } i \vDash \beta & i \vDash \bullet_I \alpha \text{ iff } i > 0 \text{ and } \tau_i - \tau_{i-1} \in I \text{ and } i-1 \vDash \alpha \\
i \vDash \neg\alpha & \text{iff } i \nvDash \alpha & i \vDash \alpha \wedge \beta \text{ iff } i \vDash \alpha \text{ and } i \vDash \beta & i \vDash \bigcirc_I \alpha \text{ iff } \tau_{i+1} - \tau_i \in I \text{ and } i+1 \vDash \alpha
\end{array}
$$

$$
\begin{array}{l}
i \vDash \alpha\, \mathcal{S}_I\, \beta \text{ iff } j \vDash \beta \text{ for some } j \leq i \text{ with } \tau_i - \tau_j \in I \text{ and } k \vDash \alpha \text{ for all } j < k \leq i \\
i \vDash \alpha\, \mathcal{U}_I\, \beta \text{ iff } j \vDash \beta \text{ for some } j \geq i \text{ with } \tau_j - \tau_i \in I \text{ and } k \vDash \alpha \text{ for all } i \leq k < j
\end{array}
$$

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

supported comparisons. Herasimau [16] relaxes Yuan's requirements while formally verifying the correctness statement. Our algorithm minimizes the computed proof objects' size as this 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 [29] and Vydra [28] in Isabelle and lattice-mtl [8] in Coq. Others leveraged SMT technology to increase their trustworthiness [12, 14]. 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 [35], model checking [37,38], and SAT solving [11,21].

Several monitors visualize their output [1, 2, 7, 18, 25, 30]; some of these even present visually separate verdicts for different parts of the policy. Our work takes inspiration from these approaches, but goes deeper: our minimal proof trees characterize precisely how the verdicts for the different parts compose to a verdict for the overall policy.

Our work follows the "proof trees as explanations" paradigm and thereby joins a series of works on LTL [4, 15, 32], CFTL [13], and CTL [9]. Of these only Basin et al. [4] supports past operators and none support metric intervals. Two of the above works [9, 15] use proof systems based on the unrolling equations for temporal operators instead of the operator's semantics, which we believe is suboptimal for understandability: users think about the operators in terms 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 [31] and in the database community [10].

Metric Temporal Logic. We briefly recall MTL's syntax and point-based semantics [6]. MTL formulas are built from atomic propositions (a, b, c, ...) via Boolean ($\wedge$, $\vee$, $\neg$) and metric temporal operators (previous $\bullet_I$, next $\bigcirc_I$, 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 $\mathsf{ETP}_\rho(\tau)$. 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 $\mathsf{LTP}_\rho(\tau)$. Whenever the trace ρ is fixed, we will only write $\mathsf{ETP}(\tau)$ and $\mathsf{LTP}(\tau)$.

2 Local Proof System

We introduce a local proof system for monitoring MTL formulas as the least relation satisfying the rules shown in Figure 2. It contains two mutually dependent judgments: $\vdash^+$

$$\frac{a \in \pi_i}{i \vdash^+ a}\, ap^+ \qquad \frac{i \vdash^- \alpha}{i \vdash^+ \neg\alpha}\, \neg^+ \qquad \frac{i \vdash^+ \alpha}{i \vdash^+ \alpha \vee \beta}\, \vee_L^+ \qquad \frac{i \vdash^+ \beta}{i \vdash^+ \alpha \vee \beta}\, \vee_R^+ \qquad \frac{i \vdash^+ \alpha \quad i \vdash^+ \beta}{i \vdash^+ \alpha \wedge \beta}\, \wedge^+$$

$$\frac{a \notin \pi_i}{i \vdash^- a}\, ap^- \qquad \frac{i \vdash^+ \alpha}{i \vdash^- \neg\alpha}\, \neg^- \qquad \frac{i \vdash^- \alpha}{i \vdash^- \alpha \wedge \beta}\, \wedge_L^- \qquad \frac{i \vdash^- \beta}{i \vdash^- \alpha \wedge \beta}\, \wedge_R^- \qquad \frac{i \vdash^- \alpha \quad i \vdash^- \beta}{i \vdash^- \alpha \vee \beta}\, \vee^-$$

$$\frac{j \le i \quad \tau_i - \tau_j \in I \quad j \vdash^+ \beta \quad \forall k \in (j, i].\, k \vdash^+ \alpha}{i \vdash^+ \alpha\, \mathcal{S}_I\, \beta}\, \mathcal{S}^+ \qquad \frac{i > 0 \quad \tau_i - \tau_{i-1} \in I \quad i-1 \vdash^+ \alpha}{i \vdash^+ \bullet_I \alpha}\, \bullet^+$$

$$\frac{\mathsf{E}_i^{\mathsf{p}}([l,r]) \le j \quad j \le i \quad m = \mathsf{L}_i^{\mathsf{p}}([l,r]) \quad \tau_i - \tau_0 \ge l \quad j \vdash^- \alpha \quad \forall k \in [j, m].\, k \vdash^- \beta}{i \vdash^- \alpha\, \mathcal{S}_{[l,r]}\, \beta}\, \mathcal{S}^-$$

$$\frac{j = \mathsf{E}_i^{\mathsf{p}}([l,r]) \quad m = \mathsf{L}_i^{\mathsf{p}}([l,r]) \quad \tau_i - \tau_0 \ge l \quad \forall k \in [j, m].\, k \vdash^- \beta}{i \vdash^- \alpha\, \mathcal{S}_{[l,r]}\, \beta}\, \mathcal{S}_\infty^- \qquad \frac{\tau_i - \tau_0 < l}{i \vdash^- \alpha\, \mathcal{S}_{[l,r]}\, \beta}\, \mathcal{S}_{<I}^-$$

$$\frac{}{0 \vdash^- \bullet_I \alpha}\, \bullet_0^- \qquad \frac{i > 0 \quad i-1 \vdash^- \alpha}{i \vdash^- \bullet_I \alpha}\, \bullet^- \qquad \frac{i > 0 \quad \tau_i - \tau_{i-1} < I}{i \vdash^- \bullet_I \alpha}\, \bullet_{<I}^- \qquad \frac{i > 0 \quad \tau_i - \tau_{i-1} > I}{i \vdash^- \bullet_I \alpha}\, \bullet_{>I}^-$$

$$\frac{i \le j \quad \tau_j - \tau_i \in I \quad j \vdash^+ \beta \quad \forall k \in [i, j).\, k \vdash^+ \alpha}{i \vdash^+ \alpha\, \mathcal{U}_I\, \beta}\, \mathcal{U}^+ \qquad \frac{\tau_{i+1} - \tau_i \in I \quad i+1 \vdash^+ \alpha}{i \vdash^+ \bigcirc_I \alpha}\, \bigcirc^+$$

$$\frac{m = \mathsf{E}_i^{\mathsf{f}}(I) \quad i \le j \quad j \le \mathsf{L}_i^{\mathsf{f}}(I) \quad j \vdash^- \alpha \quad \forall k \in [m, j].\, k \vdash^- \beta}{i \vdash^- \alpha\, \mathcal{U}_I\, \beta}\, \mathcal{U}^- \qquad \frac{\tau_{i+1} - \tau_i < I}{i \vdash^- \bigcirc_I \alpha}\, \bigcirc_{<I}^-$$

$$\frac{m = \mathsf{E}_i^{\mathsf{f}}(I) \quad j = \mathsf{L}_i^{\mathsf{f}}(I) \quad \forall k \in [m, j].\, k \vdash^- \beta}{i \vdash^- \alpha\, \mathcal{U}_I\, \beta}\, \mathcal{U}_\infty^- \qquad \frac{i+1 \vdash^- \alpha}{i \vdash^- \bigcirc_I \alpha}\, \bigcirc^- \qquad \frac{\tau_{i+1} - \tau_i > I}{i \vdash^- \bigcirc_I \alpha}\, \bigcirc_{>I}^-$$

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

(for satisfaction proofs) and $\vdash^-$ (for violation proofs). A satisfaction (violation) proof describes the satisfaction (violation) of a formula at a given time-point on a fixed trace ρ. Each rule is suffixed by $^+$ or $^-$, indicating whether an operator has been satisfied or violated. Moreover, we define $\mathsf{E}_i^{\mathsf{p}}(I) := \mathsf{ETP}(\tau_i - r)$ and $\mathsf{L}_i^{\mathsf{p}}(I) := \min(i, \mathsf{LTP}(\tau_i - l))$ for $I = [l, r]$, which correspond to the earliest and latest time-point within the interval I, respectively, when formulas having $\mathcal{S}_I$ as their topmost operator are considered. In the definition of $\mathsf{L}_i^{\mathsf{p}}(I)$ we take the minimum to account for consecutive time-stamps with the same value. For formulas having $\mathcal{U}_I$ as their topmost operator, both definitions are mirrored, resulting in $\mathsf{E}_i^{\mathsf{f}}(I) := \max(i, \mathsf{ETP}(\tau_i + l))$ and $\mathsf{L}_i^{\mathsf{f}}(I) := \mathsf{LTP}(\tau_i + r)$.

The semantics of the MTL operators directly corresponds to the satisfaction rules ap^+, $\neg^+$, $\vee_L^+$, $\vee_R^+$, $\wedge^+$, $\mathcal{S}^+$, $\mathcal{U}^+$, $\bullet^+$, and $\bigcirc^+$. For instance, consider two time-points j and i such that $j \le i$. The rule $\mathcal{S}^+$ is applied whenever the time-stamp difference $\tau_i - \tau_j$ belongs to the interval I, and there is a witness for a satisfaction proof of β in the form of $j \vdash^+ \beta$ together with a finite sequence of satisfaction proofs of α for all $k \in (j, i]$. The violation 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 $\bullet_I$, $\bigcirc_I$, $\mathcal{S}_I$, and $\mathcal{U}_I$ are derived by negating and rewriting their semantics. Consider $\mathcal{S}_I$ with $I = [l, r]$:

$$\begin{aligned} i \nvDash \alpha\, \mathcal{S}_I\, \beta \;\leftrightarrow\; & \big(\tau_i - \tau_0 \ge l \wedge \exists j \in (\mathsf{E}_i^{\mathsf{p}}(I), i].\, j \nvDash \alpha \wedge \forall k \in [j, \mathsf{L}_i^{\mathsf{p}}(I)].\, k \nvDash \beta\big) \vee \\ & \big(\tau_i - \tau_0 \ge l \wedge \forall k \in [\mathsf{E}_i^{\mathsf{p}}(I), \mathsf{L}_i^{\mathsf{p}}(I)].\, k \nvDash \beta\big) \vee \tau_i - \tau_0 < l \end{aligned} \qquad (1)$$

The rules $\mathcal{S}^-$, $\mathcal{S}_\infty^-$, and $\mathcal{S}_{<I}^-$ correspond to the three disjuncts in Equation (1). We argue that these three cases intuitively represent different ways of violating a since operator. In the first disjunct, α is violated at some time-point after the interval starts and β is violated

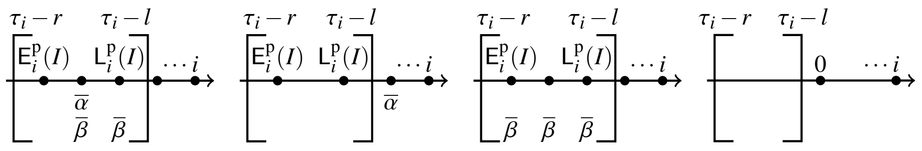


Fig. 3(a): $\mathcal{S}^-$ cases Fig. 3(b): $\mathcal{S}^-_\infty$ case Fig. 3(c): $\mathcal{S}^-_{<I}$ case

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

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. Moreover, if $l \neq 0$, i.e., if the interval does not include the current time-point, then α may be violated between the interval's end and the current time-point. Figure 3(a) shows both cases, where $\overline{\varphi}$ denotes a violation of φ. In the second disjunct, β is violated at every time-point inside the interval (Figure 3(b)). The third disjunct captures the special case at the beginning of the trace when the interval is located before the first time-point (Figure 3(c)). Next, we consider $\mathcal{U}_I$:

$$\begin{aligned} i \nvDash \alpha\,\mathcal{U}_I\,\beta \;\leftrightarrow\; & \big(\exists j \in [i, \mathsf{L}^{\mathsf{f}}_i(I)).\ j \nvDash \alpha \wedge \forall k \in [\mathsf{E}^{\mathsf{f}}_i(I), j].\ k \nvDash \beta\big) \vee \\ & \big(\forall k \in [\mathsf{E}^{\mathsf{f}}_i(I), \mathsf{L}^{\mathsf{f}}_i(I)].\ k \nvDash \beta\big) \end{aligned} \tag{2}$$

The rules $\mathcal{U}^-$ and $\mathcal{U}^-_\infty$ correspond to the two disjuncts in Equation (2). 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. *Fix an arbitrary trace $\rho = \langle(\pi_i,\tau_i)\rangle_{i\in\mathbb{N}}$. For any formula φ and $i \in \mathbb{N}$, we have $i \vdash^+ \varphi$ iff $i \vDash \varphi$ and $i \vdash^- \varphi$ iff $i \nvDash \varphi$, i.e., the proof system is sound and complete.*

In other words, proof trees in our proof system contain all the necessary information to explain why a formula has been satisfied or violated on a given trace. A mechanically checked proof of the above statement can be found in our Isabelle formalization [22].

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 $5 \nvDash \varphi$ has the following form:

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

In ρ, only events with time-stamp 3 satisfy the interval conditions, resulting in $\mathsf{E}^{\mathsf{p}}_5(I) = 1$ and $\mathsf{L}^{\mathsf{p}}_5(I) = 4$, where $I = [1,2]$. (Time-points are zero-based.) Thus, the portion of the trace we are interested in is $\langle(\{a,b\},3),(\{a,b\},3),(\{\cdot\},3),(\{a\},3)\rangle$. Here, a is only violated at time-point 3, so our proof includes the witness $3 \vdash^- a$. From there until time-point $\mathsf{L}^{\mathsf{p}}_5(I) = 4$ the subformula $b \wedge c$ is violated, witnessed by $3 \vdash^- b$ and $4 \vdash^- b$. ■

3 Proof Objects

To make proofs from our proof system explicit, we define an inductive syntax for satisfaction ($\mathfrak{sp}$) and violation ($\mathfrak{vp}$) 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.

$$\begin{aligned}\mathfrak{sp} &= ap^+(\mathbb{N},\Sigma) \mid \neg^+(\mathfrak{vp}) \mid \vee^+_L(\mathfrak{sp}) \mid \vee^+_R(\mathfrak{sp}) \mid \wedge^+(\mathfrak{sp},\mathfrak{sp}) \mid \bullet^+(\mathfrak{sp}) \mid \circ^+(\mathfrak{sp})\\ &\mid \mathcal{S}^+(\mathfrak{sp},\overline{\mathfrak{sp}_\varnothing}) \mid \mathcal{U}^+(\mathfrak{sp},\overline{\mathfrak{sp}_\varnothing})\\ \mathfrak{vp} &= ap^-(\mathbb{N},\Sigma) \mid \neg^-(\mathfrak{sp}) \mid \vee^-(\mathfrak{vp},\mathfrak{vp}) \mid \wedge^-_L(\mathfrak{vp}) \mid \wedge^-_R(\mathfrak{vp}) \mid \bullet^-(\mathfrak{vp}) \mid \bullet^-_{<I}(\mathbb{N})\\ &\mid \bullet^-_{>I}(\mathbb{N}) \mid \bullet^-_0 \mid \circ^-(\mathfrak{vp}) \mid \circ^-_{<I}(\mathbb{N}) \mid \circ^-_{>I}(\mathbb{N}) \mid \mathcal{S}^-_{<I}(\mathbb{N}) \mid \mathcal{S}^-(\mathbb{N},\mathfrak{vp},\overline{\mathfrak{vp}_\varnothing})\\ &\mid \mathcal{S}^-_\infty(\mathbb{N},\overline{\mathfrak{vp}_\varnothing}) \mid \mathcal{U}^-(\mathbb{N},\mathfrak{vp},\overline{\mathfrak{vp}_\varnothing}) \mid \mathcal{U}^-_\infty(\mathbb{N},\overline{\mathfrak{vp}_\varnothing})\end{aligned}$$

Here, $\overline{\mathfrak{sp}}$ and $\overline{\mathfrak{vp}}$ denote finite non-empty sequences of $\mathfrak{sp}$ and $\mathfrak{vp}$ subproofs and $\overline{\mathfrak{sp}_\varnothing}$ and $\overline{\mathfrak{vp}_\varnothing}$ denote finite possibly empty sequences of $\mathfrak{sp}$ and $\mathfrak{vp}$ subproofs. We define $\mathfrak{p} = \mathfrak{sp} \uplus \mathfrak{vp}$ to be the disjoint union of satisfaction and violation proofs. Given a proof $p \in \mathfrak{p}$, we define $\mathbb{V}(p)$ to be $\top$ if $p \in \mathfrak{sp}$ and $\bot$ if $p \in \mathfrak{vp}$. Each constructor corresponds to a rule in our proof system. 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^-, $\bullet^-_{<I}$, $\bullet^-_{>I}$, $\circ^-_{<I}$, $\circ^-_{>I}$, $\mathcal{S}^-_{<I}$, $\mathcal{S}^-$, and $\mathcal{S}^-_\infty$, 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 $\rho = \langle(\pi_i,\tau_i)\rangle_{i\in\mathbb{N}}$ and a formula φ, we call a proof p *valid* at $\mathsf{tp}(p)$, denoted by $p \vdash \varphi$, if p represents a valid proof according to the rules of our local proof system. Note that once again we leave the dependency on ρ implicit in $p \vdash \varphi$. Formally, validity $p \vdash \varphi$ is defined recursively, checking for each constructor that the corresponding rule has been correctly applied. For example, atomic proofs are valid if the mentioned atom is (not) contained in the trace at the specified time-points: $ap^+(i,a) \vdash a \leftrightarrow a \in \pi_i$ ($ap^-(i,a) \vdash a \leftrightarrow a \notin \pi_i$). Moreover, for $r = \mathcal{S}^+(p,[q_1,\dots,q_n])$ we have

$$\begin{aligned} r \vdash \alpha\, \mathcal{S}_I\, \beta \;\leftrightarrow\; & \mathsf{tp}(p) \le \mathsf{tp}(r) \wedge \tau_{\mathsf{tp}(r)} - \tau_{\mathsf{tp}(p)} \in I \,\wedge \\ & [\mathsf{tp}(q_1),\dots,\mathsf{tp}(q_n)] = [\mathsf{tp}(p)+1,\mathsf{tp}(r)] \wedge p \vdash \beta \wedge (\forall k \in [1,n].\ q_k \vdash \alpha).\end{aligned}$$

Multiple valid proofs may exist for a time-point i and formula φ as we demonstrate next.

Example 2. The proof object representing the proof tree from Example 1 is $P_1 = \mathcal{S}^-(5, ap^-(3,a), [\wedge^-_L(ap^-(3,b)), \wedge^-_L(ap^-(4,b))])$. However, we could have argued differently, using the fact that c is violated at all time-points inside the interval. Then, $\mathcal{S}^-_\infty$ would be used instead to 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 $\mathsf{tp}(P_2) = 5$. 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 $\mathsf{tp}(P_3) = 5$. It is structurally identical to 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, both b and c are violated at time-points 3 and 4, so we can use either to justify the violations of $b \wedge c$.

We now compare P_1, P_2, and P_3. The proof P_2 uses $\mathcal{S}^-_\infty$, so we must store a witness of the violation of $b \wedge c$ for each one of the 4 time-points inside the interval. The proofs P_1 and P_3 use $\mathcal{S}^-$, taking advantage of the violation proof $3 \vdash^- a$ that allows us to dismiss both $1 \vdash^+ a$ and $2 \vdash^+ a$. Formally, we define the size $|p|$ of a proof p to be the number of proof object constructors occurring in p. Then, $|P_1| = |P_3| = 6$, and $|P_2| = 9$. ■

We are particularly interested in small proofs as they tend to be easier to understand. Given a trace ρ and a formula φ, a proof p is *minimal* at time-point i if and only if it is valid at i ($p \vdash \varphi$ and $\mathsf{tp}(p) = i$), and all other valid proofs q (at i) have greater or equal size ($q \vdash \varphi$ and $\mathsf{tp}(q) = i$ implies $|p| \le |q|$). In our example, P_1 and P_3 are minimal.

$$\textbf{type } \mathit{buf} = \mathfrak{p}\ \mathit{list} \times \mathfrak{p}\ \mathit{list} \qquad \textbf{type } \mathit{buft} = \mathfrak{p}\ \mathit{list} \times \mathfrak{p}\ \mathit{list} \times ((\mathit{ts} \times \mathit{tp})\ \mathit{list})$$

$$\begin{array}{llll}
\textbf{type } \mathit{saux} = \{\ \mathsf{ts}_{\mathit{zero}} : \mathit{ts\ option},\ \mathsf{ts\_tp}_{\mathit{in}} : (\mathit{ts} \times \mathit{tp})\ \mathit{list},\ \mathsf{ts\_tp}_{\mathit{out}} : (\mathit{ts} \times \mathit{tp})\ \mathit{list}, & & & \\
\quad \mathsf{s\_beta\_alphas}_{\mathit{in}} & : (\mathit{ts} \times \mathfrak{sp})\ \mathit{slist}, & \mathsf{s\_beta\_alphas}_{\mathit{out}} & : (\mathit{ts} \times \mathfrak{sp})\ \mathit{list}, \\
\quad \mathsf{v\_alpha\_betas}_{\mathit{in}} & : (\mathit{ts} \times \mathfrak{vp})\ \mathit{slist}, & \mathsf{v\_alphas}_{\mathit{out}} & : (\mathit{ts} \times \mathfrak{vp})\ \mathit{slist}, \\
\quad \mathsf{v\_betas}_{\mathit{in}} & : (\mathit{ts} \times \mathfrak{vp})\ \mathit{list}, & \mathsf{v\_alphas\_betas}_{\mathit{out}} & : (\mathit{ts} \times \mathfrak{vp}\ \mathit{option} \times \mathfrak{vp}\ \mathit{option})\ \mathit{list}\ \}
\end{array}$$

$$\begin{array}{l}
\textbf{type } \mathit{state} = \mathsf{Pred}_{\mathsf{S}}\ \mathit{string} \mid \mathsf{Neg}_{\mathsf{S}}\ \mathit{state} \mid \mathsf{And}_{\mathsf{S}}\ \mathit{state\ state\ buf} \mid \mathsf{Or}_{\mathsf{S}}\ \mathit{state\ state\ buf} \\
\mid\ \mathsf{Prev}_{\mathsf{S}}\ \mathcal{I}\ \mathit{state\ bool}\ \mathfrak{p}\ (\mathit{ts\ list}) \mid \mathsf{Next}_{\mathsf{S}}\ \mathcal{I}\ \mathit{state\ bool}\ (\mathit{ts\ list}) \\
\mid\ \mathsf{Since}_{\mathsf{S}}\ \mathcal{I}\ \mathit{state\ state\ buft\ saux} \mid \mathsf{Until}_{\mathsf{S}}\ \mathcal{I}\ \mathit{state\ state\ buft\ uaux}
\end{array}$$

$$\textbf{function } \mathsf{init} :: \mathit{formula} \Rightarrow \mathit{state} \qquad \textbf{function } \mathsf{eval} :: \mathit{ts} \times \mathit{tp} \Rightarrow \mathit{atom\ set} \Rightarrow \mathit{state} \Rightarrow \mathfrak{p}\ \mathit{list} \times \mathit{state}$$

Fig. 4: Types of the monitor's state and evaluation functions

4 Computing Minimal Proofs

Given an MTL formula φ, our (online) 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, such that a single event may trigger the output of multiple proofs at once. In this section, we describe our algorithm in detail and explain its correctness.

4.1 Monitor's State

Figure 4 shows the types of our algorithm's main functions init, which computes the monitor'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 (type *state* in Figure 4) 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, in the state of $\alpha\ \mathcal{S}_I\ \beta$, 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*. Our monitor's overall structure is modeled after VERIMON [29], 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 structures, which store sufficient information to compute minimal proofs for formulas with topmost operator $\mathcal{S}$ and $\mathcal{U}$. Here, we only describe *saux* in detail.

The data structure *saux* for a formula $\varphi = \alpha\ \mathcal{S}_I\ \beta$ is a record consisting of nine fields. We will describe it next assuming that φ is being evaluated at the current time-point *cur*. Furthermore, some fields have the type *option*, which means they are of the form $\perp$ (if no value is available) or $\lfloor v \rfloor$ (storing the value v). The function THE retrieves the optional value from $\lfloor v \rfloor$, i.e., THE $(\lfloor v \rfloor) = v$. The field $\mathsf{ts}_{\mathit{zero}}$ stores $\perp$ in the initial state, and after the first event arrives, it stores the first time-stamp $\lfloor \tau_0 \rfloor$. Fields $\mathsf{ts\_tp}_{\mathit{in}}$ and $\mathsf{ts\_tp}_{\mathit{out}}$ store lists of time-stamp-time-point pairs inside the interval (between $\mathsf{E}^{\mathsf{p}}_{\mathit{cur}}(I)$ and $\mathsf{L}^{\mathsf{p}}_{\mathit{cur}}(I)$) and after the interval (between $\mathsf{L}^{\mathsf{p}}_{\mathit{cur}}(I) + 1$ and *cur*), respectively. The other fields store

```
1:  procedure UPDATE_SAUX ($[l,r], \tau_{cur}, cur, p_1, p_2, saux$)
2:      $saux.\mathsf{ts}_{zero} \leftarrow$ if $saux.\mathsf{ts}_{zero} = \bot$ then $\lfloor \tau_{cur} \rfloor$ else $saux.\mathsf{ts}_{zero}$
3:      $saux \leftarrow$ ADD_SUBPS ($\tau_{cur}, p_1, p_2, saux$)
            ▷ update s_betas_alphas$_{in}$, s_betas_alphas$_{out}$, v_alphas_betas$_{out}$, and v_alphas$_{out}$
4:      if $\tau_{cur} <$ THE ($saux.\mathsf{ts}_{zero}$) $+ l$ then
5:          $saux.\mathsf{ts\_tp}_{out} \leftarrow$ APPEND ($saux.\mathsf{ts\_tp}_{out}, [(\tau_{cur}, cur)]$)
6:          return ($\mathcal{S}^{-}_{<I}(cur), saux$)
7:      else
8:          $lr \leftarrow$ (if $r = \infty$ then THE ($saux.\mathsf{ts\_zero}$) else MAX ($0, \tau_{cur} - r$), $\tau_{cur} - l$)
9:          $saux \leftarrow$ SHIFT_SAUX($lr, l, \tau_{cur}, cur, saux$)
10:         $minimal\_proof \leftarrow$ EVAL_SAUX($cur, saux$)   ▷ extract proofs; pick one of minimal size
11:         return ($minimal\_proof, saux$)
12:
13: procedure SHIFT_SAUX ($lr, l, \tau_{cur}, cur, saux$)
14:     $saux \leftarrow$ SHIFT_TS_TPS ($lr, l, \tau_{cur}, cur, saux$)   ▷ update ts_tp$_{out}$ and ts_tp$_{in}$
15:     $saux \leftarrow$ SHIFT_SAT ($lr, saux$)   ▷ update s_beta_alphas$_{out}$ and s_beta_alphas$_{in}$
16:     $saux \leftarrow$ SHIFT_VIO ($lr, saux$)   ▷ update v_alphas_betas$_{out}$, v_alpha_betas$_{in}$, and v_betas$_{in}$
17:     $saux \leftarrow$ REMOVE_SAUX ($lr, saux$)   ▷ remove too old proofs (that fell out of the interval)
18:     return $saux$
```

Algorithm 1: State update algorithm for Since

satisfaction (prefix s_) or violation (v_) proofs. Specifically, s_beta_alphas$_{in}$ stores $\mathcal{S}^{+}$ proofs inside and s_beta_alphas$_{out}$ stores $\mathcal{S}^{+}$ proofs after the interval. Crucially, while s_beta_alphas$_{out}$ is an ordinary list, s_beta_alphas$_{in}$ has type *slist*, which is a variant of the list type that indicates that the stored proofs are sorted in ascending order (with respect to size). We maintain this invariant to optimize the number of proofs we must store, i.e., if a proof enters the interval, we can delete all larger proofs that entered the interval prior to it. In addition, 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 analogue of s_beta_alphas$_{in}$ for $\mathcal{S}^{-}$ proofs with a violation of α inside the interval, and a sequence of violations of β until the end of the interval. Note that $\mathcal{S}^{-}$ proofs can also be constructed using a single violation proof of α that occurs after the interval, and these are instead stored in the also sorted list v_alphas$_{out}$. Moreover, $\mathcal{S}^{-}_{\infty}$ proofs require that β is violated at all time-points inside the interval, so v_betas$_{in}$ stores a 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.

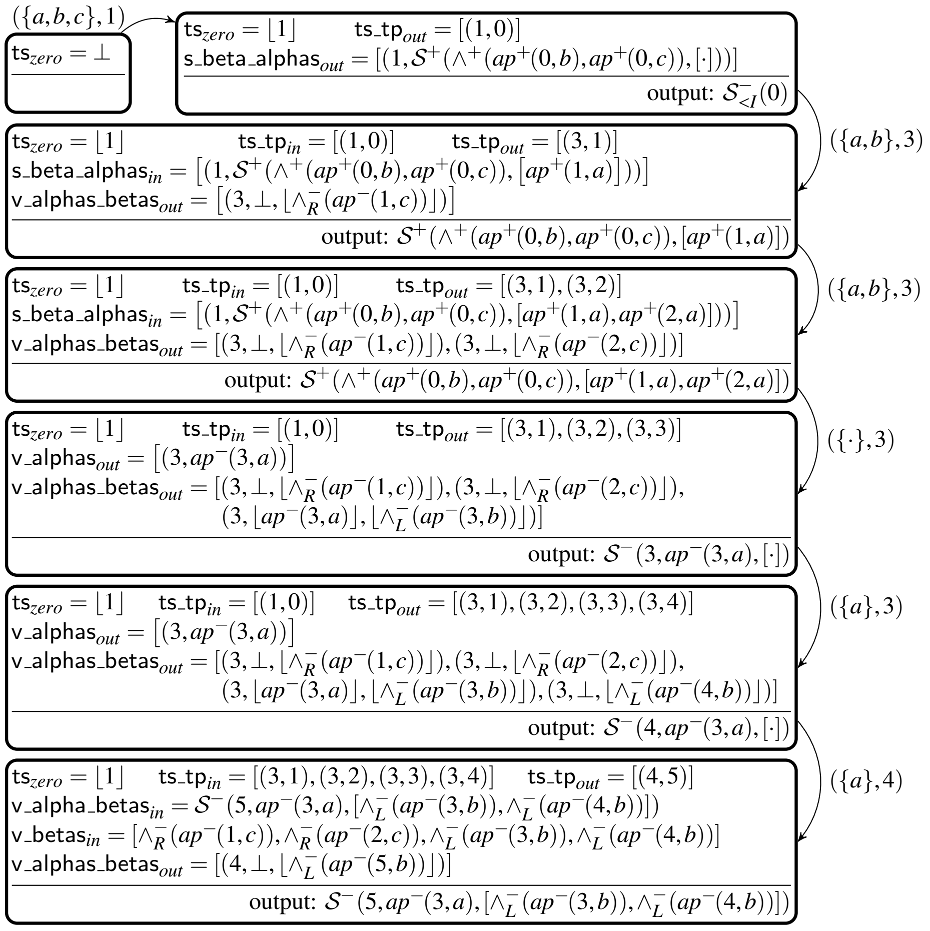
4.2 State Update

Algorithm 1 shows the skeleton of our procedure for updating (and simultaneously evaluating) the state of a since operator. The state update for $\varphi = \alpha\ \mathcal{S}_I\ \beta$ is parametrized by the interval $I = [l,r]$, the current time-point *cur* and its time-stamp τ_{cur}, minimal proofs p_1 and p_2 (obtained recursively) for the subformulas α and β, respectively, and the current state *saux*. The procedure first checks if *cur* is the first time-point to arrive and initializes ts_{zero} accordingly (line 2). Next, we add the new subproofs to their destinations (ADD_SUBPS). For example, if $p_1 \in \mathfrak{sp}$ then all proofs from $\mathsf{s\_betas\_alphas}_{in}$ and $\mathsf{s\_betas\_alphas}_{out}$ are extended with this additional satisfaction proof for α. In contrast, if $p_1 \in \mathfrak{vp}$ then both $\mathsf{s\_betas\_alphas}$ lists are emptied and the violation of α is stored in $\mathsf{v\_alphas}_{out}$ and $\mathsf{v\_alphas\_betas}_{out}$ instead. A similar case distinction happens for p_2. After storing the proofs, we handle the case where *cur* is a time-point at the beginning of the trace for which the past interval has not started yet (lines 4–6), which corresponds to the $\mathcal{S}^-_{<I}$ case depicted in Figure 3(b) on the right. Here, we add a new time-stamp-time-point pair to $\mathsf{ts\_tp}_{out}$ (line 5), and return the proof $\mathcal{S}^-_{<I}(cur)$ and the updated *saux*.

In the general case (when the interval has started), we compute the absolute time-stamp pair *lr* that constitute the boundaries of the past interval *I* relative to τ_{cur} (line 8). We use the absolute boundaries to identify a potential interval shift and move proofs in *saux* from the *out* lists to the *in* lists accordingly (line 9). Lines 13–18 provide additional details in which order the various components are shifted. Lastly, we compute a minimal proof (line 10), performing a case distinction. If $\mathsf{s\_beta\_alphas}_{in}$ is non-empty, then its head must be a minimal satisfaction proof. Otherwise, the formula is violated and a minimal violation proof is either the head of $\mathsf{v\_alpha\_betas}_{in}$ or the head of $\mathsf{v\_alphas}_{out}$ (after adding a $\mathcal{S}^-$ constructor) or the application of $\mathcal{S}^-_\infty$ to $\mathsf{v\_betas}_{in}$ (provided that this suffix spans the entire interval which can be deduced by comparing the lengths of $\mathsf{v\_betas}_{in}$ and $\mathsf{ts\_tp}_{in}$). We extract these (at most three) candidates, compute their sizes, and pick one of minimal size. This minimal proof and the updated *saux* are then returned (line 11).

Example 3. To illustrate how the state is updated, we once again consider the formula and trace introduced in Example 1. Figure 5 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 $\bot$. When the first event $(\{a,b,c\},1)$ arrives, the list $\mathsf{ts\_tp}_{out}$ 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 $\mathsf{s\_beta\_alphas}_{out}$. This proof is clearly not valid for the current time-point 0, considering that the interval $[1,2]$ has not yet started, so the monitor outputs 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, and both $\mathsf{ts\_tp}_{out}$ and $\mathsf{ts\_tp}_{in}$ are updated accordingly. Furthermore, the algorithm extends the $\mathcal{S}^+$ proof previously stored in $\mathsf{s\_beta\_alphas}_{out}$ by adding $ap^+(1,a)$ to the sequence of a satisfactions, after which the resulting proof is moved to $\mathsf{s\_beta\_alphas}_{in}$. The algorithm also appends the proof $ap^-(1,c)$ to $\mathsf{v\_alphas\_betas}_{out}$. Because $\mathsf{s\_beta\_alphas}_{in}$ 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 $\mathsf{s\_beta\_alphas}_{in}$. Aside from outputting the extended satisfaction proof, the algorithm also adds the proof $ap^-(2,c)$ to $\mathsf{v\_alphas\_betas}_{out}$. When event $(\{\cdot\},3)$ arrives, the sequence of a satisfactions comes to an end, which indicates that the proofs in $\mathsf{s\_beta\_alphas}_{in}$ and $\mathsf{s\_beta\_alphas}_{out}$ are no longer valid nor useful. Hence, we clear both lists. In addition, the proof $ap^-(3,a)$ is stored in $\mathsf{v\_alphas}_{out}$, since the a violation happens after the interval. This subproof is also appended to $\mathsf{v\_alphas\_betas}_{out}$ 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 $\mathsf{v\_alphas}_{out}$ and outputs it. When $(\{a\},3)$ arrives, the algorithm appends the proof $\wedge^-_L$ to


$(\{a,b,c\},1)$

$\mathsf{ts}_{zero} = \bot$

$\mathsf{ts}_{zero} = \lfloor 1 \rfloor \quad \mathsf{ts\_tp}_{out} = [(1,0)]$
$\mathsf{s\_beta\_alphas}_{out} = [(1, \mathcal{S}^+(\wedge^+(ap^+(0,b), ap^+(0,c)), [\cdot]))]$
output: $\mathcal{S}^-_{<I}(0)$

$(\{a,b\},3)$

$\mathsf{ts}_{zero} = \lfloor 1 \rfloor \quad \mathsf{ts\_tp}_{in} = [(1,0)] \quad \mathsf{ts\_tp}_{out} = [(3,1)]$
$\mathsf{s\_beta\_alphas}_{in} = \big[(1, \mathcal{S}^+(\wedge^+(ap^+(0,b), ap^+(0,c)), \big[ap^+(1,a)\big]))\big]$
$\mathsf{v\_alphas\_betas}_{out} = \big[(3, \bot, \lfloor \wedge^-_R(ap^-(1,c)) \rfloor)\big]$
output: $\mathcal{S}^+(\wedge^+(ap^+(0,b), ap^+(0,c)), [ap^+(1,a)])$

$(\{a,b\},3)$

$\mathsf{ts}_{zero} = \lfloor 1 \rfloor \quad \mathsf{ts\_tp}_{in} = [(1,0)] \quad \mathsf{ts\_tp}_{out} = [(3,1),(3,2)]$
$\mathsf{s\_beta\_alphas}_{in} = \big[(1, \mathcal{S}^+(\wedge^+(ap^+(0,b), ap^+(0,c)), [ap^+(1,a), ap^+(2,a)]))\big]$
$\mathsf{v\_alphas\_betas}_{out} = [(3, \bot, \lfloor \wedge^-_R(ap^-(1,c)) \rfloor), (3, \bot, \lfloor \wedge^-_R(ap^-(2,c)) \rfloor)]$
output: $\mathcal{S}^+(\wedge^+(ap^+(0,b), ap^+(0,c)), [ap^+(1,a), ap^+(2,a)])$

$(\{\cdot\},3)$

$\mathsf{ts}_{zero} = \lfloor 1 \rfloor \quad \mathsf{ts\_tp}_{in} = [(1,0)] \quad \mathsf{ts\_tp}_{out} = [(3,1),(3,2),(3,3)]$
$\mathsf{v\_alphas}_{out} = \big[(3, ap^-(3,a))\big]$
$\mathsf{v\_alphas\_betas}_{out} = [(3, \bot, \lfloor \wedge^-_R(ap^-(1,c)) \rfloor), (3, \bot, \lfloor \wedge^-_R(ap^-(2,c)) \rfloor),$
$(3, \lfloor ap^-(3,a) \rfloor, \lfloor \wedge^-_L(ap^-(3,b)) \rfloor)]$
output: $\mathcal{S}^-(3, ap^-(3,a), [\cdot])$

$(\{a\},3)$

$\mathsf{ts}_{zero} = \lfloor 1 \rfloor \quad \mathsf{ts\_tp}_{in} = [(1,0)] \quad \mathsf{ts\_tp}_{out} = [(3,1),(3,2),(3,3),(3,4)]$
$\mathsf{v\_alphas}_{out} = \big[(3, ap^-(3,a))\big]$
$\mathsf{v\_alphas\_betas}_{out} = [(3, \bot, \lfloor \wedge^-_R(ap^-(1,c)) \rfloor), (3, \bot, \lfloor \wedge^-_R(ap^-(2,c)) \rfloor),$
$(3, \lfloor ap^-(3,a) \rfloor, \lfloor \wedge^-_L(ap^-(3,b)) \rfloor), (3, \bot, \lfloor \wedge^-_L(ap^-(4,b)) \rfloor)]$
output: $\mathcal{S}^-(4, ap^-(3,a), [\cdot])$

$(\{a\},4)$

$\mathsf{ts}_{zero} = \lfloor 1 \rfloor \quad \mathsf{ts\_tp}_{in} = [(3,1),(3,2),(3,3),(3,4)] \quad \mathsf{ts\_tp}_{out} = [(4,5)]$
$\mathsf{v\_alpha\_betas}_{in} = \mathcal{S}^-(5, ap^-(3,a), [\wedge^-_L(ap^-(3,b)), \wedge^-_L(ap^-(4,b))])$
$\mathsf{v\_betas}_{in} = [\wedge^-_R(ap^-(1,c)), \wedge^-_R(ap^-(2,c)), \wedge^-_L(ap^-(3,b)), \wedge^-_L(ap^-(4,b))]$
$\mathsf{v\_alphas\_betas}_{out} = [(4, \bot, \lfloor \wedge^-_L(ap^-(5,b)) \rfloor)]$
output: $\mathcal{S}^-(5, ap^-(3,a), [\wedge^-_L(ap^-(3,b)), \wedge^-_L(ap^-(4,b))])$

Fig. 5: The monitor's *saux* states when executing Example 1

$\mathsf{v\_alphas\_betas}_{out}$ and again uses the same subproof stored in $\mathsf{v\_alphas}_{out}$ to construct $\mathcal{S}^-(4, ap^-(3,a), [\cdot])$. Note that 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 $\mathsf{ts\_tp}_{in}$ and $\mathsf{ts\_tp}_{out}$ change accordingly. At this stage, the algorithm populates $\mathsf{v\_alpha\_betas}_{in}$ and $\mathsf{v\_betas}_{in}$ with the subproofs stored in $\mathsf{v\_alphas\_betas}_{out}$. In particular, it constructs and stores the proof $\mathcal{S}^-(5, ap^-(3,a), [\wedge^-_L(ap^-(3,b)), \wedge^-_L(ap^-(4,b))])$ in $\mathsf{v\_alpha\_betas}_{in}$. Moreover, a sequence of violations of the conjunction inside the interval is stored in $\mathsf{v\_betas}_{in}$. This sequence of violations fills the entire interval, so it is then used to construct the proof $\mathcal{S}^-_\infty(5, [\wedge^-_R(ap^-(1,c)), \wedge^-_R(ap^-(2,c)), \wedge^-_R(ap^-(3,c)), \wedge^-_R(ap^-(4,c))])$. The $\mathcal{S}^-$ proof corresponds precisely to the proof tree presented in Example 1, and the proof object P_1 in Example 2, whereas the $\mathcal{S}^-_\infty$ proof corresponds to the proof object P_2. Lastly, the size of these two proofs is computed, and the algorithm selects the $\mathcal{S}^-$ proof, since it is smaller (i.e., it includes fewer constructors). ■

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{sorted}(\mathsf{s\_beta\_alphas}_{in}) \wedge \mathsf{sorted}(\mathsf{v\_alpha\_betas}_{in}) \wedge \mathsf{sorted}(\mathsf{v\_alphas}_{out}) \wedge} \\
(1)\ \forall (\tau, u) \in \mathsf{s\_beta\_alphas}_{in}. & \exists p\ \bar{q}.\ u = \mathcal{S}^{+}(p, \overline{q}) \wedge u \vdash \alpha\ \mathcal{S}_I\ \beta \wedge \mathsf{tp}(u) = cur \wedge \tau = \mathsf{ts}(p) \\
(2)\ \forall (\tau, u) \in \mathsf{s\_beta\_alphas}_{out}. & \exists p\ \bar{q}.\ u = \mathcal{S}^{+}(p, \overline{q}) \wedge u \vdash \alpha\ \mathcal{S}\ \beta \wedge \mathsf{tp}(u) = cur \wedge \tau = \mathsf{ts}(p) \\
(3)\ \forall (\tau, u) \in \mathsf{v\_alpha\_betas}_{in}. & \exists p\ \bar{q}.\ u = \mathcal{S}^{-}(cur, p, \overline{q}) \wedge u \vdash \alpha\ \mathcal{S}_I\ \beta \wedge \tau = \mathsf{ts}(p) \\
(4)\ \forall (\tau, p) \in \mathsf{v\_alphas}_{out}. & \mathcal{S}^{-}(cur, p, []) \vdash \alpha\ \mathcal{S}_I\ \beta \wedge \tau = \mathsf{ts}(p) \\
(5)\ \forall (\tau, p) \in \mathsf{v\_betas\_suffix}_{in}. & \mathsf{E}^{\mathsf{p}}_{cur}(I) \le \mathsf{tp}(p) \le \mathsf{L}^{\mathsf{p}}_{cur}(I) \wedge p \vdash \beta \wedge \neg \mathbb{V}(p) \wedge \tau = \mathsf{ts}(p) \\
\multicolumn{2}{l}{(6)\ \forall (\tau, p^*, q^*) \in \mathsf{v\_alphas\_betas}_{out}.\ \exists i \in \left]\mathsf{L}^{\mathsf{p}}_{cur}(I), cur\right].\ \tau = \tau_i \wedge} \\
\multicolumn{2}{l}{\qquad (p^* = \bot \vee (\exists p.\ \neg \mathbb{V}(p) \wedge p^* = \lfloor p \rfloor \wedge p \vdash \alpha)) \wedge (q^* = \bot \vee (\exists q.\ \neg \mathbb{V}(q) \wedge q^* = \lfloor q \rfloor \wedge q \vdash \beta))}
\end{array}$$

Fig. 6: The algorithm's invariant (soundness)

4.3 Correctness

We now formally describe the invariant we maintain for *saux*. We write $\mathsf{ts}(p)$ for the time-stamp associated with a proof, i.e., the time-stamp $\tau_{\mathsf{tp}(p)}$ of the associated time-point $\mathsf{tp}(p)$. We also use functional programming notations like λ-abstractions and the list map function. We define the predicate $\mathsf{sorted}(seq) := \big(\forall (\tau_i, p_i), (\tau_j, p_j) \in seq.\ (i < j) \wedge (j < \mathsf{length}(seq)) \rightarrow \tau_i \le \tau_j \wedge |p_i| \le |p_j|\big)$ 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 $\tau_i \le \tau_j$). The fields ts_{zero}, $\mathsf{ts\_tp}_{in}$ and $\mathsf{ts\_tp}_{out}$ are characterized as follows:

$$\mathsf{ts}_{zero} = \begin{cases} \bot & \text{iff } cur = -1 \\ \lfloor \tau_0 \rfloor & \text{iff } cur \ge 0 \end{cases} \qquad \begin{array}{l} \mathsf{ts\_tp}_{in} \ = \mathsf{map}\ (\lambda i.\ (\tau_i, i))\ \left[\mathsf{E}^{\mathsf{p}}_{cur}(I), \mathsf{L}^{\mathsf{p}}_{cur}(I)\right] \\ \mathsf{ts\_tp}_{out} = \mathsf{map}\ (\lambda i.\ (\tau_i, i))\ \left]\mathsf{L}^{\mathsf{p}}_{cur}(I), cur\right] \end{array}$$

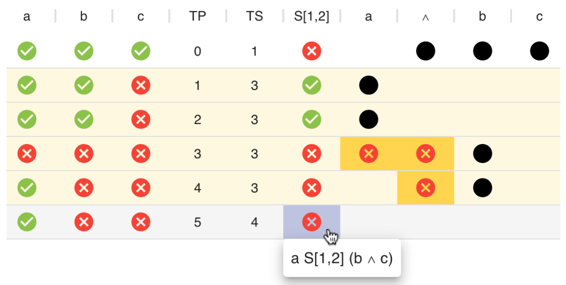
The desired properties of the objects stored in other fields are given in Figure 6.

We describe each of the invariant's statements. In (1) a proof in $\mathsf{s\_beta\_alphas}_{in}$ (which must be sorted) must have form $\mathcal{S}^{+}(p, \overline{q})$ and be a valid proof of $\alpha\ \mathcal{S}_I\ \beta$ at the current time-point, with time-stamp $\mathsf{ts}(p)$. Next, (2) requires 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, namely once $\mathsf{ts}(p)$ moves inside the interval. The statement (3) is precisely the same as (1), but for $\mathcal{S}^{-}$ proofs. In (4), each proof p in $\mathsf{v\_alphas}_{out}$ (which must too be sorted) must be a valid subproof of a $\mathcal{S}^{-}$ proof at the current time-point with time-stamp $\mathsf{ts}(p)$. In (5), each subproof corresponding to the violation of β must be inside the interval with time-stamp $\mathsf{ts}(p)$. 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 statements formalize what must hold for the things stored in *saux*, which yields soundness. We briefly consider completeness, by answering the question of what must be stored, on the example of $\mathsf{s\_beta\_alphas}_{in}$:

$$\begin{aligned}
&\forall p\ \bar{q}\ \tau.\ \mathcal{S}^{+}(p, \overline{q}) \vdash \alpha \mathcal{S}_I \beta \wedge \mathsf{tp}\left(\mathcal{S}^{+}(p, \overline{q})\right) = cur \wedge \tau = \mathsf{ts}(p) \rightarrow \\
&\qquad \big(\exists p'\ \bar{q}'\ \tau'.\ |\mathcal{S}^{+}(p', \overline{q}')| \le |\mathcal{S}^{+}(p, \overline{q})| \wedge \mathcal{S}^{+}(p', \overline{q}') \vdash \alpha\ \mathcal{S}_I\ \beta \wedge \tau' = \mathsf{ts}\left(p'\right) \wedge \\
&\qquad \tau' \ge \tau \wedge \mathsf{tp}\left(\mathcal{S}^{+}(p', \overline{q}')\right) = \mathsf{tp}\left(\mathcal{S}^{+}(p, \overline{q})\right) \wedge \left(\tau', \mathcal{S}^{+}(p', \overline{q}')\right) \in \mathsf{s\_beta\_alphas}_{in}\big)
\end{aligned}$$

In words: for any valid $\mathcal{S}^{+}$ proof for $\varphi = \alpha\ \mathcal{S}_I\ \beta$ at time-point cur, we must store in $\mathsf{s\_beta\_alphas}_{in}$ another proof at most as large and old, that is also valid for φ at cur. Other fields of *saux* have similar completeness statements and so have other state components.

Together, soundness and completeness ensure that given a formula, a trace, and a time-point i, our online monitoring algorithm will eventually output a valid minimal proof at i.



a	b	c	TP	TS	S[1,2]	a	∧	b	c
✓	✓	✓	0	1	✗		●	●	●
✓	✓	✗	1	3	✓	●			
✓	✓	✗	2	3	✓	●			
✗	✗	✗	3	3	✗	✗	✗	●	
✓	✗	✗	4	3	✗		✗	●	
✓	✗	✗	5	4	✗				

a S[1,2] (b ∧ c)

Fig. 7: Visualization of Example 1

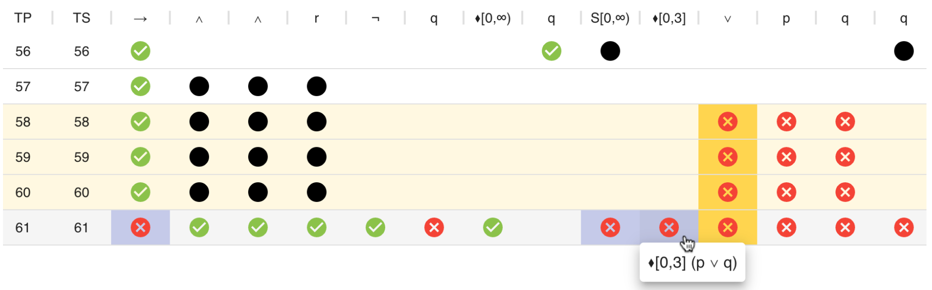
5 Implementation

We implement our algorithm in a new tool called EXPLANATOR2 [22]. The implementation amounts to around 4 000 lines of OCaml. In addition, a 6 900 lines long OCaml program is extracted from our Isabelle formalization consisting of 19 000 lines of definitions and proofs. The extracted program contains the proof object validity checker in the form of a function is_valid : *trace* $\rightarrow$ *formula* $\rightarrow$ *proof* $\rightarrow$ *bool*, which effectively implements what we denote by $p \vdash \varphi$. Moreover, it also contains the minimality checker is_minimal : *trace* $\rightarrow$ *formula* $\rightarrow$ *proof* $\rightarrow$ *bool* that given a trace ρ, a formula φ, and a proof p computes a proof q for φ on ρ at time-point $\mathtt{tp}(p)$ with a minimal size using a verified dynamic programming algorithm and then checks that $|p| \leq |q|$. Note that q may differ from p because minimal proof objects are not unique. Herasimau [16] provides more details on the formalization and the dynamic programming algorithm. We used the verified validity and minimality checkers to thoroughly test our unverified algorithm. Our tool includes a command line option to enable the verified certification of its output, which slows down computation as the verified algorithm is rather inefficient but increases trustworthiness.

EXPLANATOR2 also includes a JavaScript web front end. To this end, we transpile the compiled OCaml bytecode to JavaScript using `Js_of_ocaml` [36]. The resulting JavaScript library runs in any web browser. We augment the library with an interactive visualization using React [17]. Figure 7 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), the time-stamps (column TS) and associated time-points (column TP). The following columns show either the topmost operator of the different subformulas or the atomic propositions of our monitored MTL formula $\varphi = a\,\mathcal{S}_{[1,2]}\,(b \wedge c)$. In particular, the column labeled with φ's topmost operator, namely $\mathcal{S}_{[1,2]}$, shows the Boolean verdicts that a traditional monitor would output. Users of EXPLANATOR2 can further inspect the Boolean verdicts by clicking on them. Figure 7 shows the visualization's state after clicking on φ's violation at time-point 5. The visualization highlights the time interval and the Boolean verdicts for subformulas that justify the verdict 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-point that are relevant for at least one of φ's verdicts. The Boolean value for these relevant subformula verdicts is only revealed upon exploration.

$$\dfrac{\dfrac{\dfrac{\vdots}{61 \vdash^+ r \wedge \neg q}\,\wedge^+ \qquad \dfrac{\dfrac{q \in \{q\}}{56 \vdash^+ q}\,ap^+}{61 \vdash^+ \blacklozenge q}\,\blacklozenge^+}{61 \vdash^+ (r \wedge \neg q) \wedge \blacklozenge q}\,\wedge^+ \qquad \dfrac{\dfrac{\dfrac{\vdots}{58,\ldots,61 \vdash^- p \vee q}\,\vee^-}{61 \vdash^- \blacklozenge_{[0,3]}\,(p \vee q)}\,\blacklozenge^- \qquad \dfrac{q \notin \{r\}}{61 \vdash^- q}\,ap^-}{61 \vdash^- \left(\blacklozenge_{[0,3]}\,(p \vee q)\right) \mathcal{S}\, q}\,\mathcal{S}^-}{61 \vdash^- \Big((r \wedge \neg q) \wedge \blacklozenge q\Big) \to \Big(\left(\blacklozenge_{[0,3]}\,(p \vee q)\right) \mathcal{S}\, q\Big)}\,\to^-$$

Fig. 8: Proof of φ_1's violation at time-point 61



TP	TS	→	∧	∧	r	¬	q	⧫[0,∞)	q	S[0,∞)	⧫[0,3]	∨	p	q	q
56	56	✓							✓	●					●
57	57	✓	●	●	●										
58	58	✓	●	●	●							✗	✗	✗	
59	59	✓	●	●	●							✗	✗	✗	
60	60	✓	●	●	●							✗	✗	✗	
61	61	✗	✓	✓	✓	✓	✗	✓		✗	✗	✗	✗	✗	✗

Fig. 9: Visualization of φ_1's violation at time-point 61

6 Examples

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. To this end, we consider Timescales [34], a benchmark generator for MTL monitors. Timescales uses predefined MTL formulas that represent temporal patterns that commonly occur in real system designs [20]. It generates traces, in which the time-stamps are equal to their corresponding time-points. 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 our approach to explain these violations. In addition to the operators presented in Figure 2, we extended our proof system and algorithm with the following operators: $\top$ (truth), $\bot$ (falsity), $\to$ (implies), $\leftrightarrow$ (iff), $\blacksquare_I$ (historically), $\square_I$ (always), $\blacklozenge_I$ (once), and $\lozenge_I$ (eventually).

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 $\varphi_1 = (r \wedge \neg q \wedge \blacklozenge q) \to \big((\blacklozenge_{[0,u]}\,(p \vee q))\ \mathcal{S}\ q\big)$. We set the bound $u = 3$, and we consider the trace $\langle \ldots, (\{q\}, 56), (\{\cdot\}, 57), (\{\cdot\}, 58), (\{\cdot\}, 59), (\{\cdot\}, 60), (\{r\}, 61)\rangle$, which is the portion pertinent to the proof. The formula φ_1 is violated at time-point 61 and the proof is shown in Figure 8.

To prove the violation of the implication (the formula's topmost operator) the subformula on the left (assumption) must be satisfied and the subformula on the right (conclusion) must be violated. For this reason, two subproofs are constructed. In the left subproof, we can see that the subformula on the left is violated because both conjuncts $r \wedge \neg q$ and $\blacklozenge q$ are satisfied at time-point 61. This part of the formula enforces that: (i) r is satisfied (and q is not satisfied) at the current time-point; and (ii) q is satisfied at some point in the past. Note that (ii) corresponds exactly to $\blacklozenge q$. In the left subproof, we have $61 \vdash^+ r \wedge \neg q$ because r is satisfied and q is violated at time-point 61. Moreover, the proof $61 \vdash^+ \blacklozenge q$

uses the fact that q is satisfied at time-point 56, which is when the last q had arrived. Moving to the subproof $61 \vdash^{-} (\blacklozenge_{[0,3]}\,(p \vee q))\,\mathcal{S}\,q$, the violation occurs because both subformulas are violated at time-point 61. The subproof $61 \vdash^{-} \blacklozenge_{[0,3]}\,(p \vee q)$ uses the violations of p and q in the last 3 time units $(58,\dots,61)$, whereas the proof $61 \vdash^{-} q$ indicates that q is not satisfied at the current time-point. This is sufficient to show that since the last q has arrived (at time-point 56), it is neither the case that a new sequence started (with a new occurrence of q) or that a sequence finished (with an occurrence of p) within 3 time units in the past.

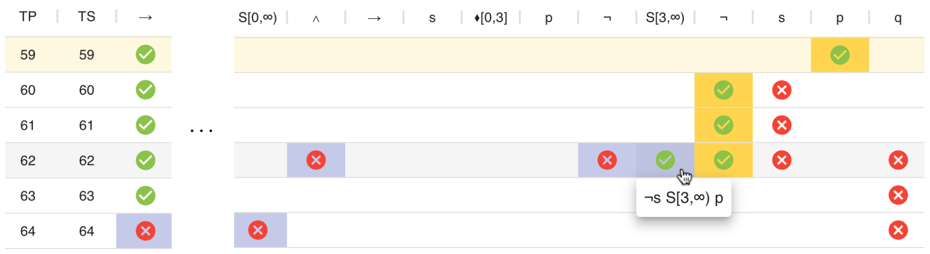
Figure 9 shows our visualization of the above proof. Starting from $\rightarrow$, the columns show the topmost operators of φ_1's subformulas (including atomic propositions). For example, φ_1 is violated because the left subformula is satisfied (the first $\wedge$ column) and the right subformula is violated (column $\mathcal{S}_{[0,\infty)}$). All subformulas have a corresponding column and the order of the columns is such that immediate subformulas of a subformula appear further to the right. The same atomic proposition may occur in different subformulas, in which case there will be multiple columns showing the same proposition (but potentially different time-points of interest). Continuing our example, the right subproof from Figure 8 starts in column $\mathcal{S}_{[0,\infty)}$ in Figure 9. The formula $(\blacklozenge_{[0,3]}\,(p \vee q))\,\mathcal{S}\,q$ is violated at time-point 61 because both subformulas are violated. In the visualization, we focus (by clicking) on the subformula $\blacklozenge_{[0,3]}\,(p \vee q)$ (displayed when hovering over the corresponding cell) and observe that it is violated because $p \vee q$ is violated at time-points $58,\dots,61$ (highlighted cells in the $\vee$ column). Also, the context of this subproof, i.e., all parent nodes in the proof tree, is highlighted. In this case, these are $\rightarrow$ and $\mathcal{S}_{[0,\infty)}$ at time-point 61. Even though it presents the exact same information as the proof tree, our interactive visualization makes the proofs easier to navigate, explore, and digest.

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 $\varphi_2 = ((r \wedge \neg q) \wedge \blacklozenge q) \rightarrow \big(\big(\big(s \rightarrow \blacklozenge_{[l,u]} p\big) \wedge \neg \big(\neg s\,\mathcal{S}_{[u,\infty)}\,p\big)\big)\,\mathcal{S}\,q\big)$. We consider the trace $\langle \dots, (\{q\},58), (\{p\},59), (\{\cdot\},60), (\{\cdot\},61), (\{\cdot\},62), (\{\cdot\},63), (\{r\},64)\rangle$ and set $l = 0$ and $u = 3$. Figure 10 shows a violation proof for φ_2 at time-point 64.

The implication's assumption in φ_2 is the same as the assumption in φ_1 (the *bounded recurrence* formula). We omit the corresponding subproof P from Figure 11 as it has the same structure as the subproof of the *bounded recurrence* example. (Yet, there are differences in the time-points.) The conclusion of φ_2 has the form $\alpha\,\mathcal{S}\,q$. It is violated at time-point 64 because α is violated at time-point 62, and from this point onward until the current time-point 64, q is always violated. According to our proof system, we only need to consider violations of q starting at time-point 62, because α is violated at that point. The formula $\alpha = \big(s \rightarrow \blacklozenge_{[0,3]} p\big) \wedge \neg \big(\neg s\,\mathcal{S}_{[3,\infty)}\,p\big)$ captures two properties: (i) if there is a response s then there must be a recent challenge p (i.e., p must be satisfied within the last 3 time units); (ii) there are no challenges p more than 3 time units in the past without a response s. In our proof, the violation of α is constructed using the violation of (ii). After applying the negation rule, 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 challenge p within the required time constraint. Figure 11 shows the visualization of this subproof. While the static image already helps with the intuition, we invite the reader to explore this and the previous example in our interactive visualization.

$$
\dfrac{\boxed{\begin{matrix}P\\ \vdots\end{matrix}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p \in \{p\}}{59 \vdash^{+} p}\,ap^{+} \qquad \dfrac{\dfrac{\vdots}{60,\ldots,62 \vdash^{-} s}}{60,\ldots,62 \vdash^{+} \neg s}\,\neg^{+}}{62 \vdash^{+} \neg s\, \mathcal{S}_{[3,\infty)}\, p}\,\mathcal{S}^{+}}{62 \vdash^{-} \neg\left(\neg s\, \mathcal{S}_{[3,\infty)}\, p\right)}\,\neg^{-}}{62 \vdash^{-} \left(s \rightarrow \blacklozenge_{[0,3]} p\right) \wedge \neg\left(\neg s\, \mathcal{S}_{[3,\infty)}\, p\right)}\,\wedge^{-}_{R} \qquad \dfrac{\vdots}{62,\ldots,64 \vdash^{-} q}}{64 \vdash^{-} \left(\left(s \rightarrow \blacklozenge_{[0,3]} p\right) \wedge \neg\left(\neg s\, \mathcal{S}_{[3,\infty)}\, p\right)\right) \mathcal{S}\, q}\,\mathcal{S}^{-}}{64 \vdash^{-} \left((r \wedge \neg q) \wedge \blacklozenge q\right) \rightarrow \left(\left(\left(s \rightarrow \blacklozenge_{[0,3]} p\right) \wedge \neg\left(\neg s\, \mathcal{S}_{[3,\infty)}\, p\right)\right) \mathcal{S}\, q\right)}\,\rightarrow^{-}
$$

Fig. 10: Proof of φ_2's violation at time-point 64



TP	TS	→		S[0,∞)	∧	→	s	♦[0,3]	p	¬	S[3,∞)	¬	s	p	q
59	59	✓												✓	
60	60	✓										✓	✗		
61	61	✓	. . .									✓	✗		
62	62	✓			✗					✗	✓	✓	✗		✗
63	63	✓													✗
64	64	✗		✗											✗

¬s S[3,∞) p

Fig. 11: Visualization of φ_2's violation at time-point 64

7 Performance

We empirically evaluate our tool by answering the following research question: How does EXPLANATOR2 scale with respect to the formula size when compared to other state-of-the-art monitoring tools? To this end, we reuse the evaluation setup of the MTL monitor HYDRA [26]. We consider two different settings: (i) past-only MTL formulas; and (ii) MTL formulas (mixing past and future operators). For each setting we pseudo-randomly generate a trace with 100 000 events and collections of five different formulas for each size $s \in \{6, 17, \ldots, 50\}$. We measure the time and space usage of the EXPLANATOR2, HYDRA and VYDRA [27], AERIAL [3] MONPOLY [5], and VERIMON [29]. Our verified dynamic programming algorithm is not included because it times out (with a time-out of 200 seconds) even for the smallest formulas of size 6. The experiments were conducted on a computer with an AMD Ryzen 5 5600X CPU and 16GB of RAM. The results are presented in Figure 12. Each filled shape is an average of the measurements for the corresponding formula size. (Unfilled shapes show the individual runs, but are sometimes invisible.) The axes showing time and space usage measurements are of logarithmic scale.

Time-wise, EXPLANATOR2 outperforms MONPOLY and VERIMON (first-order monitors), and is on par with most of its competitors in the past-only setting. When we include future operators, EXPLANATOR2 performs worse than its competitors, although only by a narrow margin. However, we must consider that in contrast to the others our tool has a clear disadvantage: it produces checkable and understandable output instead of Boolean verdicts. Thus, these results reassure us that we do not compromise too much by

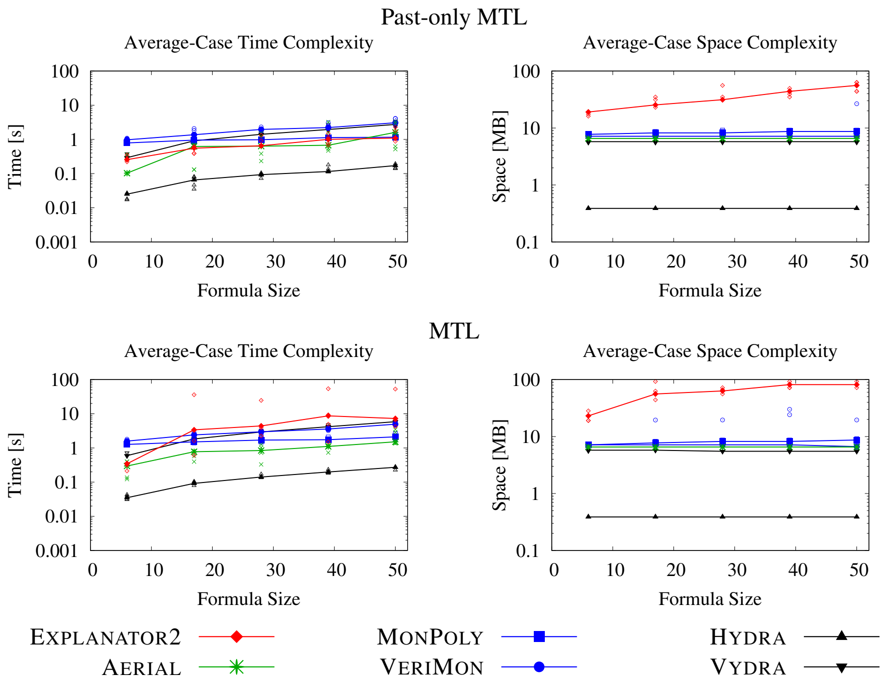


Fig. 12: Evaluation results

providing this feature, and that our algorithm is indeed efficient. In terms of space usage, EXPLANATOR2 performs worse than other monitoring tools in both settings. This is hardly surprising, given that proofs can be huge (e.g., they may contain the entire trace).

8 Conclusion

We have developed an online MTL monitor that outputs detailed verdicts in the form of proof trees, which serve as both understandable explanations and checkable certificates. Our monitor incorporates a formally verified checker and an interactive visualization. Our empirical evaluation demonstrates the reasonable performance of our monitor, even though it provides a strictly more informative output than its competitors. 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 lift our approach to the more expressive metric first-order temporal logic. The main challenge here is to incorporate parametric events and quantification. Moreover, we are interested in optimizing other aspects of the proofs than their size.

Data Availability Statement EXPLANATOR2 is available under the GNU Lesser General Public License v3.0 [22] and its interactive visualization is hosted on GitHub. Our artifact [23] contains the snapshot of the tool's source code at paper submission time along with instructions on how to run our test suite and to reproduce our evaluation.

References

1. Artho, C., Havelund, K., Honiden, S.: Visualization of concurrent program executions. In: COMPSAC 2007. pp. 541–546. IEEE Computer Society (2007). https://doi.org/10.1109/COMPSAC.2007.236
2. Bartocci, E., Ferrère, T., Manjunath, N., Nickovic, D.: Localizing faults in Simulink/Stateflow models with STL. In: Prandini, M., Deshmukh, J.V. (eds.) HSCC 2018. pp. 197–206. ACM (2018). https://doi.org/10.1145/3178126.3178131
3. Basin, D., Bhatt, B.N., Krstic, S., Traytel, D.: Almost event-rate independent monitoring. Formal Methods Syst. Des. **54**(3), 449–478 (2019). https://doi.org/10.1007/s10703-018-00328-3
4. Basin, D., Bhatt, B.N., Traytel, D.: Optimal proofs for linear temporal logic on lasso words. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 37–55. Springer (2018). https://doi.org/10.1007/978-3-030-01090-4_3
5. Basin, D., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. J. ACM **62**(2), 15:1–15:45 (2015). https://doi.org/10.1145/2699444
6. Basin, D., Klaedtke, F., Zalinescu, E.: Algorithms for monitoring real-time properties. Acta Informatica **55**(4), 309–338 (2018). https://doi.org/10.1007/s00236-017-0295-4
7. Baumeister, J., Finkbeiner, B., Gumhold, S., Schledjewski, M.: Real-time visualization of stream-based monitoring data. In: Dang, T., Stolz, V. (eds.) RV 2022. LNCS, vol. 13498, pp. 325–335. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_21
8. Chattopadhyay, A., Mamouras, K.: A verified online monitor for metric temporal logic with quantitative semantics. In: Deshmukh, J., Nickovic, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 383–403. Springer (2020). https://doi.org/10.1007/978-3-030-60508-7_21
9. Chechik, M., Gurfinkel, A.: A framework for counterexample generation and exploration. Int. J. Softw. Tools Technol. Transf. **9**(5-6), 429–445 (2007). https://doi.org/10.1007/s10009-007-0047-9
10. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Found. Trends Databases **1**(4), 379–474 (2009). https://doi.org/10.1561/1900000006
11. Cruz-Filipe, L., Heule, M.J.H., Jr., W.A.H., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: de Moura, L. (ed.) CADE 26. vol. 10395, pp. 220–236. Springer (2017). https://doi.org/10.1007/978-3-319-63046-5_14
12. Dauer, J.C., Finkbeiner, B., Schirmer, S.: Monitoring with verified guarantees. In: Feng, L., Fisman, D. (eds.) RV 2021. LNCS, vol. 12974, pp. 62–80. Springer (2021). https://doi.org/10.1007/978-3-030-88494-9_4
13. Dawes, J.H., Reger, G.: Explaining violations of properties in control-flow temporal logic. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 202–220. Springer (2019). https://doi.org/10.1007/978-3-030-32079-9_12
14. Finkbeiner, B., Oswald, S., Passing, N., Schwenger, M.: Verified Rust monitors for Lola specifications. In: Deshmukh, J., Nickovic, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 431–450. Springer (2020). https://doi.org/10.1007/978-3-030-60508-7_24
15. 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). https://doi.org/10.1016/j.jlamp.2020.100636
16. Herasimau, A.: Formalizing Explanations for Metric Temporal Logic. B.Sc. thesis, ETH Zürich (2020)
17. Hunt, P., O'Shannessy, P., Smith, D., Coatta, T.: React: Facebook's functional turn on writing JavaScript. ACM Queue **14**(4), 40 (2016). https://doi.org/10.1145/2984629.2994373
18. Kallwies, H., Leucker, M., Schmitz, M., Schulz, A., Thoma, D., Weiss, A.: TeSSLa – an ecosystem for runtime verification. In: Dang, T., Stolz, V. (eds.) RV 2022. LNCS, vol. 13498, pp. 314–324. Springer (2022). https://doi.org/10.1007/978-3-031-17196-3_20

19. Kane, A., Chowdhury, O., Datta, A., Koopman, P.: A case study on runtime monitoring of an autonomous research vehicle (ARV) system. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 102–117. Springer (2015). https://doi.org/10.1007/978-3-319-23820-3_7
20. Konrad, S., Cheng, B.H.C.: Real-time specification patterns. In: Roman, G., Griswold, W.G., Nuseibeh, B. (eds.) ICSE 2005. pp. 372–381. ACM (2005). https://doi.org/10.1145/1062455.1062526
21. Lammich, P.: Efficient verified (UN)SAT certificate checking. J. Autom. Reason. **64**(3), 513–532 (2020). https://doi.org/10.1007/s10817-019-09525-z
22. Lima, L., Herasimau, A., Raszyk, M., Traytel, D., Yuan, S.: The development repository of EXPLANATOR2. https://github.com/runtime-monitoring/explanator2 (2022)
23. Lima, L., Herasimau, A., Raszyk, M., Traytel, D., Yuan, S.: Artifact for "Explainable online monitoring of metric temporal logic" (2023). https://doi.org/10.5281/zenodo.7509199
24. 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). https://doi.org/10.1007/s10703-017-0275-x
25. Nickovic, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: AMT 2.0: qualitative and quantitative trace analysis with extended signal temporal logic. Int. J. Softw. Tools Technol. Transf. **22**(6), 741–758 (2020). https://doi.org/10.1007/s10009-020-00582-z
26. Raszyk, M.: Efficient, Expressive, and Verified Temporal Query Evaluation. Ph.D. thesis, ETH Zürich (2022). https://doi.org/10.3929/ethz-b-000553221
27. Raszyk, M., Basin, D., Krstic, S., Traytel, D.: Multi-head monitoring of metric temporal logic. In: Chen, Y., Cheng, C., Esparza, J. (eds.) ATVA 2019. LNCS, vol. 11781, pp. 151–170. Springer (2019). https://doi.org/10.1007/978-3-030-31784-3_9
28. Raszyk, M., Basin, D., Traytel, D.: Multi-head monitoring of metric dynamic logic. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. LNCS, vol. 12302, pp. 233–250. Springer (2020). https://doi.org/10.1007/978-3-030-59152-6_13
29. Schneider, J., Basin, D., Krstic, S., Traytel, D.: A formally verified monitor for metric first-order temporal logic. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 310–328. Springer (2019). https://doi.org/10.1007/978-3-030-32079-9_18
30. Schumann, J., Moosbrugger, P., Rozier, K.Y.: Runtime analysis with R2U2: A tool exhibition report. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 504–509. Springer (2016). https://doi.org/10.1007/978-3-319-46982-9_35
31. Sulzmann, M., Lu, K.Z.M.: POSIX regular expression parsing with derivatives. In: Codish, M., Sumii, E. (eds.) FLOPS 2014. LNCS, vol. 8475, pp. 203–220. Springer (2014). https://doi.org/10.1007/978-3-319-07151-0_13
32. Sulzmann, M., Zechner, A.: Constructive finite trace analysis with linear temporal logic. In: Brucker, A.D., Julliand, J. (eds.) TAP 2012. LNCS, vol. 7305, pp. 132–148. Springer (2012). https://doi.org/10.1007/978-3-642-30473-6_11
33. Ulus, D.: Online monitoring of metric temporal logic using sequential networks. CoRR **abs/1901.00175** (2019). https://doi.org/10.48550/arxiv.1901.00175
34. Ulus, D.: Timescales: A benchmark generator for MTL monitoring tools. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 402–412. Springer (2019). https://doi.org/10.1007/978-3-030-32079-9_25
35. Völlinger, K.: Verifying the output of a distributed algorithm using certification. In: Lahiri, S.K., Reger, G. (eds.) RV 2017. LNCS, vol. 10548, pp. 424–430. Springer (2017). https://doi.org/10.1007/978-3-319-67531-2_29
36. Vouillon, J., Balat, V.: From bytecode to JavaScript: the Js_of_ocaml compiler. Softw. Pract. Exp. **44**(8), 951–972 (2014). https://doi.org/10.1002/spe.2187
37. Wimmer, S., Herbreteau, F., van de Pol, J.: Certifying emptiness of timed Büchi automata. In: Bertrand, N., Jansen, N. (eds.) FORMATS 2020. LNCS, vol. 12288, pp. 58–75. Springer (2020). https://doi.org/10.1007/978-3-030-57628-8_4

38. Wimmer, S., von Mutius, J.: Verified certification of reachability checking for timed automata. In: Biere, A., Parker, D. (eds.) TACAS 2020. LNCS, vol. 12078, pp. 425–443. Springer (2020). https://doi.org/10.1007/978-3-030-45190-5_24
39. Yuan, S.: Explaining Monitoring Verdicts for Metric Dynamic Logic. B.Sc. thesis, ETH Zürich (2019)