Explainable Online Monitoring of Metric First-Order Temporal Logic

Leonardo Lima, Jonathan Julián Huerta y Munive, and Dmitriy Traytel

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

Abstract. Metric first-order temporal logic (MFOTL) is an expressive formalism for specifying temporal and data-dependent constraints on streams of time-stamped, data-carrying events. It serves as the specification language of several runtime monitors. These monitors input an MFOTL formula and an event stream prefix and output satisfying assignments to the formula's free variables. For complex formulas, it may be unclear why a certain assignment is output. We propose an approach that accompanies assignments with detailed explanations, in the form of proof trees. We develop a new monitor that outputs such explanations. Our tool incorporates a formally verified checker that certifies the explanations and a visualization that allows users to interactively explore and understand the outputs.

1 Introduction

Runtime monitoring is concerned with the analysis of events produced by a system during its execution. An online monitor searches for given complex patterns in event streams, processing the stream incrementally, i.e., one event at a time. If it finds a pattern match, the monitor outputs a verdict to its user. The nature of a verdict depends on both the monitor and its pattern specification language. For propositional specification languages, such as metric temporal logic (MTL) [6, 21], typical verdicts are streams of Booleans [8, 28, 31], where each Boolean signifies the presence or the absence of a pattern match, i.e., the satisfaction or violation of the MTL formula at every position in the input stream.

Users might find Boolean outputs difficult to interpret, especially when complex patterns like nesting temporal operators are involved. In particular, Boolean verdicts give no insight into how monitors produce them—we have to trust their correctness. Even when assuming infallible monitors, verdict justifications can help us to ensure that we expressed correctly our intentions in the specification and, e.g., that it is not vacuously true [23].

Lima et al. [25] propose the use of richer verdicts in an MTL monitor. Specifically, they use proof trees in a dedicated proof system resembling MTL's semantics to explain why a formula is satisfied or violated. They develop the EXPLANATOR2 monitor, which outputs a stream of size-minimal proof trees, and design an interactive graphical user interface for exploring and understanding these informative verdicts. In addition, they formally verify, in the Isabelle/HOL proof assistant, a proof tree checker certifying that their proof system rules were correctly applied. Thus proof tree verdicts serve a two-fold purpose: as machine-checkable certificates and human-readable explanations.

In this work, we significantly widen the scope of the "proof tree verdicts" approach. We provide certifiable and explainable monitoring verdicts for metric first-order temporal logic (MFOTL) [14] with bounded future operators and without equality between variables. MFOTL extends MTL with data parameters and first-order quantification and is an

expressive formalism with many practical applications [7,10–13]. We extend Lima et al.'s MTL proof system to MFOTL with the expected rules for quantifiers (Section 2): e.g., the universally quantified formula $\forall x.\ \alpha$ is satisfied if α with x replaced by d is satisfied *for all* domain values d. The key challenge here is that the domain is typically infinite, which results in the above proof rule for $\forall$ to be infinitely branching. This is problematic because it is unclear how to validate a correct application of the $\forall$ rule in a proof tree checker.

A crucial observation is that without equality between variables, proof trees cannot distinguish values outside of the *active domain*, i.e., the finite set of data values from the monitored event stream prefix and from the formula's constants. Thus, the active domain's size plus one bounds the number of choices for d requiring *different* proof trees, and we can reuse them–with the extra "plus one" representing values outside the active domain. Thus, to represent the $\forall$ rule it suffices to store a finite partition of the domain and one subproof for each part. We obtain finite proof objects, develop a checker for them, and formally verify the checker's correctness in Isabelle/HOL (Section 3).

The proof system explains how to deal with closed MFOTL formulas. A Boolean verdict for a formula with free variables only makes sense relative to a variable assignment. Hence, traditional MFOTL monitors compute sets of satisfying variable assignments [15, 30] instead of Boolean verdicts. In our setting, an explanation for a formula with free variables must provide a proof tree for any variable assignment (satisfying or violating). For infinite domains, there are infinitely many assignments, but the same idea that worked for quantifiers comes to our rescue: it suffices to consider a finite partition of the domain for each variable. Inspired by binary decision diagrams (BDDs) [16], we organize the partitions for different variables hierarchically in *partitioned decision trees* (*PDT*s). PDTs are trees where each leaf stores a generic data item and each node (representing a variable) branches on a finite partition of the domain (Section 4). The partitions may change from one node to the other. PDTs can be compacted (or reduced in BDD terminology).

We thus have arrived at our notion of *explainable verdicts* for MFOTL formulas: PDTs whose leaves are proof objects. We extend our verified checker from proof objects to such verdicts and Lima et al.'s algorithm for MTL [25] to MFOTL (Section 5). Our algorithm extension is modular in the sense that it merely adds a layer of PDTs, but keeps Lima et al.'s algorithms for temporal operators unchanged. We implement the extended algorithm in a new monitor and also extend Lima et al.'s interactive visualization of proof objects. We demonstrate the effectiveness of our new tool on MFOTL policies from the literature (Section 6). In summary, we make the following contributions:

- We develop a proof system for MFOTL satisfaction and violation at a time-point for a given event stream and verify its soundness and completeness in Isabelle/HOL.
- We finitely represent our proof system's proof trees and formally verify a checker for them. The key idea is that finite partitions of infinite domains are sufficient.
- We design partitioned decision trees (PDTs) to represent functions from variable assignments to generic data items in a way that enables sharing and compression.
- We develop an algorithm computing explanations: PDTs with proof objects as leaves. We implement the algorithm in a new monitor, along with an interactive visualization of explanations and integrated with the verified proof tree checker for certification.

Our tool, called WHYMON, is publicly available [2].

$v,i \vDash tt$	$v,i \vDash \exists x.\, \alpha$ iff $v[x \mapsto d], i \vDash \alpha$ for some $d \in \mathbb{D}$
$v,i \nvDash ff$	$v,i \vDash \forall x.\, \alpha$ iff $v[x \mapsto d], i \vDash \alpha$ for all $d \in \mathbb{D}$
$v,i \vDash \mathsf{p}(\bar{t})$ iff $\mathsf{p}(\llbracket \bar{t} \rrbracket_v) \in \Gamma_i$	$v,i \vDash \bullet_I \alpha$ iff $i > 0$, $\tau_i - \tau_{i-1} \in I$, and $v, i-1 \vDash \alpha$
$v,i \vDash x \approx c$ iff $v(x) = c$	$v,i \vDash \bigcirc_I \alpha$ iff $\tau_{i+1} - \tau_i \in I$ and $v, i+1 \vDash \alpha$
$v,i \vDash \neg\alpha$ iff $v,i \nvDash \alpha$	$v,i \vDash \blacklozenge_I \alpha$ iff $v, j \vDash \beta$ for some $j \leq i$ with $\tau_i - \tau_j \in I$
$v,i \vDash \alpha \wedge \beta$ iff $v,i \vDash \alpha$ and $v,i \vDash \beta$	$v,i \vDash \lozenge_I \alpha$ iff $v, j \vDash \beta$ for some $j \geq i$ with $\tau_j - \tau_i \in I$
$v,i \vDash \alpha \vee \beta$ iff $v,i \vDash \alpha$ or $v,i \vDash \beta$	$v,i \vDash \blacksquare_I \alpha$ iff $v, j \vDash \beta$ for all $j \leq i$ with $\tau_i - \tau_j \in I$
$v,i \vDash \alpha \rightarrow \beta$ iff $v,i \nvDash \alpha$ or $v,i \vDash \beta$	$v,i \vDash \square_I \alpha$ iff $v, j \vDash \beta$ for all $j \geq i$ with $\tau_j - \tau_i \in I$
$v,i \vDash \alpha\, \mathcal{S}_I\, \beta$ iff $v, j \vDash \beta$ for some $j \leq i$ with $\tau_i - \tau_j \in I$ and $v,k \vDash \alpha$ for all $j < k \leq i$	
$v,i \vDash \alpha\, \mathcal{U}_I\, \beta$ iff $v, j \vDash \beta$ for some $j \geq i$ with $\tau_j - \tau_i \in I$ and $v,k \vDash \alpha$ for all $i \leq k < j$	

Fig. 1: Semantics of MFOTL for a fixed stream $\sigma = \langle \tau_i, \Gamma_i \rangle_{i \in \mathbb{N}}$.

Further Related Work Lima et al.'s work [25], which we extend, is based on the work by Basin et al. [9] that employed proof trees as explanations in the context of understanding counterexamples of LTL model checkers. We refer to these works for a discussion of related proof systems for propositional temporal logics and regular expressions.

In the first-order monitoring setting, we are on unexplored territory with verdicts that go beyond satisfying assignments. Nonetheless our work incorporates ideas from existing first-order monitors. Most closely related is Havelund et al's DEJAVU monitor [18], which uses BDDs to represent sets of satisfying assignments. Our work generalizes BDDs to branching over partitions of the domain and storing generic data (e.g., proof objects) instead of Booleans in the leaves. In addition, the DEJAVU authors make use of the fact that without equality between variables the formula's satisfaction cannot be influenced by different values outside the active domain. We generalize this observation so that not only the satisfaction but rather entire proof trees can be reused when exchanging values outside the active domain. Finally, DEJAVU only supports past temporal operators and closed formulas, whereas our algorithm supports both past and future operators and free variables.

Havelund et al.'s key observation fails for equalities between variables. For example, the formula $x \approx y \rightarrow \mathsf{p}(x,y)$ is satisfied for any pair of distinct values $c \neq d$ outside of the predicate p's interpretation, but it is violated if we pick the same value c for both x and y. A classic result by Ailamazyan et al. [5,19] shows that for the relational calculus (MFOTL without temporal operators) it suffices to distinguish a finite number of equivalence classes of values outside of the active domain. While it is conceivable that this result generalizes to MFOTL with equality, we leave this generalization as future work.

The MFOTL monitor MonPoly [14, 15] and its formally verified counterpart VeriMon [30] output streams of satisfying assignments for formulas in the so-called monitorable fragment. The fragment ensures that all subformulas always evaluate to finite sets of satisfying assignments. Our monitor does not suffer from this limitation; even more it returns all satisfying and violating assignments (labeled and explained as such).

Outside of first-order monitoring, our visualization takes some inspiration from the stream runtime verification tool TeSSLa [24], which can provide output for all intermediate streams. Similarly, we provide output for all subformulas, but our proof trees allow us to focus on the relevant dependencies between a formula and its subformulas.

Metric first-order temporal logic (MFOTL) We recall MFOTL's syntax and semantics. We fix an infinite domain $\mathbb{D}$ (e.g., containing integers and strings). Terms $t \in \mathbb{T}$ are either variables $x, y, z \in \mathbb{V}$ or constants $c, d \in \mathbb{D}$. Overlines indicate lists (finite sequences), e.g.,

if t is a term, then $\bar{t}$ is a list of terms. The grammar below specifies MFOTL's syntax, where $\mathsf{p} \in \mathbb{E}$ is a predicate name (e.g., a string) and $I \in \mathbb{I} \subseteq 2^{\mathbb{N}}$ is a non-empty interval.

$$\begin{aligned}\alpha ::= {} & tt \mid ff \mid \mathsf{p}(\bar{t}) \mid x \approx c \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \to \alpha \mid \exists x.\ \alpha \mid \forall x.\ \alpha \mid \\ & \bullet_I \alpha \mid \bigcirc_I \alpha \mid \blacklozenge_I \alpha \mid \lozenge_I \alpha \mid \blacksquare_I \alpha \mid \square_I \alpha \mid \alpha\ \mathcal{S}_I\ \alpha \mid \alpha\ \mathcal{U}_I\ \alpha\end{aligned}$$

Besides the first-order logic operators, the syntax includes the past $\bullet$ (previous), $\blacklozenge$ (once), $\blacksquare$ (historically), $\mathcal{S}$ (since) and future $\bigcirc$ (next), $\lozenge$ (eventually), $\square$ (always), $\mathcal{U}$ (until) temporal operators. We use $\bigwedge$ for universal and $\bigvee$ for existential quantification at the meta-alanguage level to avoid confusion with MFOTL formulas. We also use common interval notation $[a,b) = \{n \mid a \le n < b\}$ or $[a,c] = \{n \mid a \le n \le c\}$, for $a, c \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\infty\}$, and omit intervals when $I = [0,\infty) = \mathbb{N}$. Whenever we write $[a,c]$, we exclusively denote the range $[a,\ldots,c]$ (rather than the two element sequence $[a,c]$). Furthermore, we assume that the future operators ($\lozenge$, $\square$, and $\mathcal{U}$) intervals are finite (also called bounded). We write $a + I$ for $\{a + x \mid x \in I\}$ and $a\,\mathcal{R}\,I$ for $\bigwedge x \in I.\ a\,\mathcal{R}\,x$ (where $\mathcal{R} \in \{<, \le, >, \ge\}$). We interpret formulas over streams σ: infinite sequences of time-stamped sets of events $\sigma = \langle \tau_i, \Gamma_i \rangle_{i \in \mathbb{N}}$. We call the indices $i \in \mathbb{N}$ time-points, so that Γ_i is the set of events and $\tau_i \in \mathbb{N}$ is the time-stamp at time-point i. The time-stamps τ_i must be monotone ($\bigwedge i\, j.\ i \le j \longrightarrow \tau_i \le \tau_j$) and eventually increasing ($\bigwedge \tau.\ \bigvee i.\ \tau_i > \tau$). Each event has the form $\mathsf{p}(d_1,\ldots,d_n)$ where p is the event name and $d_i \in \mathbb{D}$. Given a total assignment v mapping variables to values in $\mathbb{D}$, we define $[\![x]\!]_v = v(x)$ and $[\![c]\!]_v = c$. The notation $[\![\bar{t}]\!]_v = \bar{c}$ lifts this operation to lists of terms. We define the satisfaction relation $v, i \vDash_\sigma \alpha$ in the usual way (Figure 1). Finally, the *earliest time-point* $\mathsf{ETP}_\sigma(\tau)$ of $\tau \in \mathbb{N}$ on σ is the smallest time-point i such that $\tau_i \ge \tau$. Analogously, the *latest time-point* $\mathsf{LTP}_\sigma(\tau)$ of $\tau \ge \tau_0$ on σ is the largest i such that $\tau_i \le \tau$. We omit the stream σ $\big($e.g., $\vDash$, $\mathsf{ETP}\,(\tau)$ and $\mathsf{LTP}\,(\tau)\big)$ if it is clear from the context.

2 Proof System

We introduce a local proof system for MFOTL (Figure 2). "Local" means here that the proof system does not talk about satisfiability in general, but rather about the formula's satisfaction or violation for a fixed stream, assignment, and time-point.

Our proof system consists of two mutually dependent judgments, $\vdash^+_\sigma$ and $\vdash^-_\sigma$ (again σ is omitted when clear), that characterize a formula's satisfaction $v, i \vdash^+_\sigma \alpha$ and violation $v, i \vdash^-_\sigma \alpha$ relations for assignment v, stream σ, and time-point i. The rules of our proof system closely follow the MFOTL semantics (Figure 1) and extend the proof system used by Lima et al. [25] with assignments (that are mostly passed around without modification) and the rules for quantifiers (which modify the assignments). The rules for atomic predicates and Boolean constants and operators are self-explanatory: e.g., predicates are satisfied if a matching event is present in the trace; a conjunction is satisfied if both conjuncts are satisfied; a conjunction is violated if either of the conjuncts is violated.

The rule $\exists^+$ states that for v to satisfy $\exists x.\ \alpha$ at i, it suffices to provide a domain value d such that the updated assignment $v[x \mapsto d]$ setting x to d satisfies α at i. Conversely, $\exists^-$ asserts that the violation of $\exists x.\ \alpha$ under v at i requires showing that all domain values make $v[x \mapsto d]$ violate α at i. Since the universal quantifier is dual to the existential one, the rules $\forall^-$ and $\forall^+$ exchange the relations $\vdash^+_\sigma$ and $\vdash^-_\sigma$ compared to $\exists^+$ and $\exists^-$.

$$\frac{}{v,i\vdash^+ tt}\ tt^+ \quad \frac{}{v,i\vdash^- ff}\ ff^- \quad \frac{\mathsf{p}(\llbracket \bar{t}\rrbracket_v)\in\Gamma_i}{v,i\vdash^+ \mathsf{p}(\bar{t})}\ p^+ \quad \frac{\mathsf{p}(\llbracket \bar{t}\rrbracket_v)\notin\Gamma_i}{v,i\vdash^- \mathsf{p}(\bar{t})}\ p^- \quad \frac{v,i\vdash^+ \alpha \quad v,i\vdash^- \beta}{v,i\vdash^- \alpha\to\beta}\ \to^-$$

$$\frac{v,i\vdash^- \alpha}{v,i\vdash^+ \neg\alpha}\ \neg^+ \quad \frac{v,i\vdash^- \alpha}{v,i\vdash^- \alpha\wedge\beta}\ \wedge_L^- \quad \frac{v,i\vdash^- \beta}{v,i\vdash^- \alpha\wedge\beta}\ \wedge_R^- \quad \frac{v,i\vdash^+ \alpha \quad v,i\vdash^+ \beta}{v,i\vdash^+ \alpha\wedge\beta}\ \wedge^+ \quad \frac{v,i\vdash^- \alpha}{v,i\vdash^+ \alpha\to\beta}\ \to_L^+$$

$$\frac{v,i\vdash^+ \alpha}{v,i\vdash^- \neg\alpha}\ \neg^- \quad \frac{v,i\vdash^+ \alpha}{v,i\vdash^+ \alpha\vee\beta}\ \vee_L^+ \quad \frac{v,i\vdash^+ \beta}{v,i\vdash^+ \alpha\vee\beta}\ \vee_R^+ \quad \frac{v,i\vdash^- \alpha \quad v,i\vdash^- \beta}{v,i\vdash^- \alpha\vee\beta}\ \vee^- \quad \frac{v,i\vdash^+ \beta}{v,i\vdash^+ \alpha\to\beta}\ \to_R^+$$

$$\frac{v[x\mapsto d],i\vdash^+ \alpha}{v,i\vdash^+ \exists x.\,\alpha}\ \exists^+ \quad \frac{\bigwedge d.\ v[x\mapsto d],i\vdash^+ \alpha}{v,i\vdash^+ \forall x.\,\alpha}\ \forall^+ \quad \frac{\bigwedge d.\ v[x\mapsto d],i\vdash^- \alpha}{v,i\vdash^- \exists x.\,\alpha}\ \exists^- \quad \frac{v[x\mapsto d],i\vdash^- \alpha}{v,i\vdash^- \forall x.\,\alpha}\ \forall^-$$

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

$$\frac{\tau_{i+1}-\tau_i\in I \quad v,i+1\vdash^+ \alpha}{v,i\vdash^+ \bigcirc_I\alpha}\ \bigcirc^+ \quad \frac{\tau_{i+1}-\tau_i<I}{v,i\vdash^- \bigcirc_I\alpha}\ \bigcirc_{<I}^- \quad \frac{\tau_{i+1}-\tau_i>I}{v,i\vdash^- \bigcirc_I\alpha}\ \bigcirc_{>I}^- \quad \frac{v,i+1\vdash^- \alpha}{v,i\vdash^- \bigcirc_I\alpha}\ \bigcirc^-$$

$$\frac{j\le i \quad \tau_i-\tau_j\in I \quad v,j\vdash^+ \alpha}{v,i\vdash^+ \blacklozenge_I\alpha}\ \blacklozenge^+ \quad \frac{\tau_i<\tau_0+I}{v,i\vdash^- \blacklozenge_I\alpha}\ \blacklozenge_{<l}^- \quad \frac{j\ge i \quad \tau_j-\tau_i\in I \quad v,j\vdash^+ \alpha}{v,i\vdash^+ \lozenge_I\alpha}\ \lozenge^+$$

$$\frac{\tau_i\ge\tau_0+I \quad \bigwedge j\in[\mathsf{E}_i^{\mathsf{p}}(I),\mathsf{L}_i^{\mathsf{p}}(I)].\ v,j\vdash^- \alpha}{v,i\vdash^- \blacklozenge_I\alpha}\ \blacklozenge^- \quad \frac{\bigwedge j\in[\mathsf{E}_i^{\mathsf{f}}(I),\mathsf{L}_i^{\mathsf{f}}(I)].\ v,j\vdash^- \beta}{v,i\vdash^- \lozenge_I\alpha}\ \lozenge^-$$

$$\frac{j\le i \quad \tau_i-\tau_j\in I \quad v,j\vdash^- \alpha}{v,i\vdash^- \blacksquare_I\alpha}\ \blacksquare^- \quad \frac{\tau_i<\tau_0+I}{v,i\vdash^+ \blacksquare_I\alpha}\ \blacksquare_{<l}^+ \quad \frac{j\ge i \quad \tau_j-\tau_i\in I \quad v,j\vdash^- \alpha}{v,i\vdash^+ \square_I\alpha}\ \square^-$$

$$\frac{\tau_i\ge\tau_0+I \quad \bigwedge j\in[\mathsf{E}_i^{\mathsf{p}}(I),\mathsf{L}_i^{\mathsf{p}}(I)].\ v,j\vdash^+ \alpha}{v,i\vdash^+ \blacksquare_I\alpha}\ \blacksquare^+ \quad \frac{\bigwedge j\in[\mathsf{E}_i^{\mathsf{f}}(I),\mathsf{L}_i^{\mathsf{f}}(I)].\ v,j\vdash^+ \beta}{v,i\vdash^+ \square_I\alpha}\ \square^+$$

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

$$\frac{i\le j \quad \tau_j-\tau_i\in I \quad v,j\vdash^+ \beta \quad \forall k\in[i,j).\ v,k\vdash^+ \alpha}{v,i\vdash^+ \alpha\,\mathcal{U}_I\,\beta}\ \mathcal{U}^+ \quad \frac{\tau_i<\tau_0+I}{v,i\vdash^- \alpha\,\mathcal{S}_I\,\beta}\ \mathcal{S}_{<I}^- \quad \frac{v(x)=c}{v,i\vdash^+ x\approx c}\ \approx^+$$

$$\frac{\tau_i\ge\tau_0+I \quad \bigwedge k\in[\mathsf{E}_i^{\mathsf{p}}(I),\mathsf{L}_i^{\mathsf{p}}(I)].\ v,k\vdash^- \beta}{v,i\vdash^- \alpha\,\mathcal{S}_I\,\beta}\ \mathcal{S}_\infty^- \quad \frac{\bigwedge k\in[\mathsf{E}_i^{\mathsf{f}}(I),\mathsf{L}_i^{\mathsf{f}}(I)].\ v,k\vdash^- \beta}{v,i\vdash^- \alpha\,\mathcal{U}_I\,\beta}\ \mathcal{U}_\infty^-$$

$$\frac{\mathsf{E}_i^{\mathsf{p}}(I)\le j \quad j\le i \quad \tau_i\ge\tau_0+I \quad v,j\vdash^- \alpha \quad \bigwedge k\in[j,\mathsf{L}_i^{\mathsf{p}}(I)].\ v,k\vdash^- \beta}{v,i\vdash^- \alpha\,\mathcal{S}_I\,\beta}\ \mathcal{S}^-$$

$$\frac{i\le j \quad j<\mathsf{L}_i^{\mathsf{f}}(I) \quad v,j\vdash^- \alpha \quad \forall k\in[\mathsf{E}_i^{\mathsf{f}}(I),j].\ v,k\vdash^- \beta}{v,i\vdash^- \alpha\,\mathcal{U}_I\,\beta}\ \mathcal{U}^- \quad \frac{v(x)\neq c}{v,i\vdash^- x\approx c}\ \approx^-$$

Fig. 2: Local proof system for MFOTL on a fixed stream $\sigma=\langle\tau_i,\Gamma_i\rangle_{i\in\mathbb{N}}$.

The rules $\blacklozenge^+$ and $\lozenge^+$ are mere restatements of the MFOTL semantics. Since the operators $\blacksquare_I$ and $\square_I$ are respectively dual to $\blacklozenge_I$ and $\lozenge_I$, their violation rules $\blacksquare^-$ and $\square^-$ once again exchange $\vdash_\sigma^+$ and $\vdash_\sigma^-$ compared to $\blacklozenge^+$ and $\lozenge^+$. The rule $\blacksquare_{<l}^+$ accounts for the vacuous truth of the operator $\blacksquare_I$ near the start of the stream (when no time-points fall within the interval I). Dually, the rule $\blacklozenge_{<l}^-$ asserts the violation of $\blacklozenge_I$ near the start of the stream. The remaining rules $\blacklozenge^-$, $\lozenge^-$, $\blacksquare^+$, and $\square^+$ use notation $\mathsf{E}_i^{\mathsf{p}}(I)$, $\mathsf{L}_i^{\mathsf{p}}(I)$, $\mathsf{E}_i^{\mathsf{f}}(I)$, and $\mathsf{L}_i^{\mathsf{f}}(I)$ to refer to time-points of particular interest relative to the current time-point i. Specifically,

for a future formula $\varphi = \mathcal{F}_I\,\alpha$ with $\mathcal{F} \in \{\bigcirc, \Diamond, \Box\}$ and interval $I = [a,b]$ or $I = [a,b)$ such that $b \neq \infty$, the formula's semantics at time-point i may need to refer to any time-point with time-stamp in $[\tau_i + a, \ldots, \tau_i + b]$. The latest such time-point is $\mathsf{L}^{\mathrm{f}}_i(I) = \mathsf{LTP}(\tau_i + b)$ while the earliest one is $\mathsf{E}^{\mathrm{f}}_i(I) = \max(i, \mathsf{ETP}(\tau_i + a))$. For past operators $\mathcal{P} \in \{\bullet, \blacklozenge, \blacksquare\}$, the relevant time-stamp interval is $[\tau_i - b, \ldots, \tau_i - a]$ and the interval's earliest time-point is $\mathsf{E}^{\mathrm{p}}_i(I) = \mathsf{ETP}(\tau_i - b)$ and its latest time-point is $\mathsf{L}^{\mathrm{p}}_i(I) = \min(i, \mathsf{LTP}(\tau_i - a))$.

Proof trees emerging from repeated application of the rules in our proof system contain all the necessary information to explain why a formula is satisfied or violated. In other words, our proof system is sound and complete, i.e., the following result holds.

Theorem 1. *Let α be a formula, v a variable assignment, $i \in \mathbb{N}$ a time-point, and $\sigma = \langle \tau_i, \Gamma_i \rangle_{i \in \mathbb{N}}$ a trace. Then $v, i \vdash^+_\sigma \alpha \longleftrightarrow v, i \vDash_\sigma \alpha$ and $v, i \vdash^-_\sigma \alpha \longleftrightarrow v, i \nvDash_\sigma \alpha$.*

We have formalized and verified this result in Isabelle/HOL.

Example 1. Consider the standard publish–approve example [14] requiring that any file f published by an author a, must first be approved by a manager m of a within the previous seven days. The formalization of this policy as a closed MFOTL formula is:

$$\varphi = \forall a.\, \forall f.\, \mathsf{publish}(a,f) \rightarrow \left(\blacklozenge_{[0,7]} \exists m.\, (\neg \mathsf{mgr}_{\mathsf{F}}(m,a)\; \mathcal{S}\; \mathsf{mgr}_{\mathsf{S}}(m,a)) \wedge \mathsf{approve}(m,f)\right).$$

Here, the events $\mathsf{mgr}_{\mathsf{S}}(m,a)$ and $\mathsf{mgr}_{\mathsf{F}}(m,a)$ mark m starting and finishing being a's manager. Formally, m is currently a manager of a if m started being a's manager in the past and has not finished being a's manager since. Thus, the manager relation changes over time. Consider the stream $\langle \tau_i, \Gamma_i \rangle_{i \in \mathbb{N}}$, where $\tau_0 = \tau_1 = 0$, $\tau_2 = 4$, $\tau_3 = 10$, and

$$\begin{aligned}
\Gamma_0 &= \{\mathsf{mgr}_{\mathsf{S}}(\mathtt{Mallory}, \mathtt{Alice}), \mathsf{mgr}_{\mathsf{S}}(\mathtt{Merlin}, \mathtt{Bob}), \mathsf{mgr}_{\mathsf{S}}(\mathtt{Merlin}, \mathtt{Charlie})\}, \text{ and}\\
\Gamma_1 &= \{\mathsf{approve}(\mathtt{Mallory}, 152)\}, \text{ and}\\
\Gamma_2 &= \{\mathsf{approve}(\mathtt{Merlin}, 163), \mathsf{publish}(\mathtt{Alice}, 160), \mathsf{mgr}_{\mathsf{F}}(\mathtt{Merlin}, \mathtt{Charlie})\}, \text{ and}\\
\Gamma_3 &= \{\mathsf{approve}(\mathtt{Merlin}, 187), \mathsf{publish}(\mathtt{Bob}, 163), \mathsf{publish}(\mathtt{Alice}, 163),\\
&\qquad \mathsf{publish}(\mathtt{Charlie}, 163), \mathsf{publish}(\mathtt{Charlie}, 152)\}.
\end{aligned}$$

In the following we abbreviate the subformulas of φ as follows: $\varphi_L = \mathsf{publish}(a,f)$, $\varphi_1 = \neg \mathsf{mgr}_{\mathsf{F}}(m,a)\; \mathcal{S}\; \mathsf{mgr}_{\mathsf{S}}(m,a)$, $\varphi_2 = \mathsf{approve}(m,f)$, $\varphi_\exists = \exists m.\, \varphi_1 \wedge \varphi_2$, $\varphi_R = \blacklozenge_{[0,7]}\, \varphi_\exists$, and $\varphi' = \varphi_L \rightarrow \varphi_R$. The following proof tree shows that φ is violated at time-point 3 for any v:

$$\dfrac{\dfrac{\dfrac{\mathsf{publish}(\mathtt{Charlie}, 152) \in \Gamma_3}{v[a \mapsto \mathtt{Charlie}, f \mapsto 152], 3 \vdash^+ \varphi_L}\, p^+ \quad \dfrac{\dfrac{\dfrac{\mathsf{approve}(d, 152) \notin \Gamma_i}{v[a \mapsto \mathtt{Charlie}, f \mapsto 152, m \mapsto d], i \vdash^- \varphi_2}\, p^-}{v[a \mapsto \mathtt{Charlie}, f \mapsto 152, m \mapsto d], i \vdash^- \varphi_1 \wedge \varphi_2}\, \wedge^-_R}{v[a \mapsto \mathtt{Charlie}, f \mapsto 152], i \vdash^- \varphi_\exists}\, \exists^- \quad \dfrac{v[a \mapsto \mathtt{Charlie}, f \mapsto 152], i \vdash^- \varphi_\exists}{v[a \mapsto \mathtt{Charlie}, f \mapsto 152], 3 \vdash^- \varphi_R}\, \blacklozenge^-}{v[a \mapsto \mathtt{Charlie}, f \mapsto 152], 3 \vdash^- \varphi_L \rightarrow \varphi_R}\, \rightarrow^-}{v[a \mapsto \mathtt{Charlie}], 3 \vdash^- \forall f.\, \varphi'}\, \forall^-}{v, 3 \vdash^- \forall a.\, \forall f.\, \varphi'}\, \forall^-$$

Given φ_R's temporal constraint, we note that $\tau_3 \geq 0$ and need to check $v, i \vdash^- \varphi_\exists$ for the time-points $i \in \{2,3\}$ (as $[\mathsf{E}^{\mathrm{p}}_3([0,7]), \mathsf{L}^{\mathrm{p}}_3([0,7])] = \{2,3\}$). Both subproofs are identical, so we parameterize them over i. In addition, the $\exists^-$ subproofs are valid for an arbitrary manager $d \in \mathbb{D}$ (abbreviating infinite branching over all possible domain values). $\square$

$$\begin{aligned}
\mathfrak{sp} &= tt^+(\mathbb{N}) \mid p^+(\mathbb{N},\mathbb{E},\vec{t}) \mid \neg^+(\mathfrak{vp}) \mid \wedge^+(\mathfrak{sp},\mathfrak{sp}) \mid \vee^+_L(\mathfrak{sp}) \mid \vee^+_R(\mathfrak{sp}) \mid \rightarrow^+_L(\mathfrak{vp}) \mid \rightarrow^+_R(\mathfrak{sp}) \\
&\mid \forall^+(\mathbb{V},\biguplus_{\mathbb{D}}(\mathfrak{sp})) \mid \exists^+(\mathbb{V},\mathbb{D},\mathfrak{sp}) \mid \bullet^+(\mathfrak{sp}) \mid \bigcirc^+(\mathfrak{sp}) \mid \blacklozenge^+(\mathbb{N},\mathfrak{sp}) \mid \lozenge^+(\mathbb{N},\mathfrak{sp}) \\
&\mid \blacksquare^+_{<I}(\mathbb{N}) \mid \blacksquare^+(\mathbb{N},\overline{\mathfrak{sp}}) \mid \square^+(\mathbb{N},\overline{\mathfrak{sp}}) \mid \mathcal{S}^+(\mathfrak{sp},\overline{\mathfrak{sp}}) \mid \mathcal{U}^+(\mathfrak{sp},\overline{\mathfrak{sp}}) \,, \\
\mathfrak{vp} &= \mathit{ff}^-(\mathbb{N}) \mid p^-(\mathbb{N},\mathbb{E},\vec{t}) \mid \neg^-(\mathfrak{sp}) \mid \wedge^-_R(\mathfrak{vp}) \mid \wedge^-_L(\mathfrak{vp}) \mid \vee^-(\mathfrak{vp},\mathfrak{vp}) \mid \rightarrow^-(\mathfrak{sp},\mathfrak{vp}) \\
&\mid \forall^-(\mathbb{V},\mathbb{D},\mathfrak{vp}) \mid \exists^-(\mathbb{V},\biguplus_{\mathbb{D}}(\mathfrak{vp})) \mid \bullet^-(\mathfrak{vp}) \mid \bullet^-_{<I}(\mathbb{N}) \mid \bullet^-_{>I}(\mathbb{N}) \mid \bullet^-_{0} \mid \bigcirc^-(\mathfrak{vp}) \mid \bigcirc^-_{<I}(\mathbb{N}) \\
&\mid \bigcirc^-_{>I}(\mathbb{N}) \mid \blacklozenge^-(\mathbb{N},\overline{\mathfrak{vp}}) \mid \blacklozenge^-_{<I}(\mathbb{N}) \mid \lozenge^-(\mathbb{N},\overline{\mathfrak{vp}}) \mid \blacksquare^-(\mathbb{N},\mathfrak{vp}) \mid \square^-(\mathbb{N},\mathfrak{vp}) \\
&\mid \mathcal{S}^-_{<I}(\mathbb{N}) \mid \mathcal{S}^-(\mathbb{N},\mathfrak{vp},\overline{\mathfrak{vp}}) \mid \mathcal{S}^-_{\infty}(\mathbb{N},\overline{\mathfrak{vp}}) \mid \mathcal{U}^-(\mathbb{N},\mathfrak{vp},\overline{\mathfrak{vp}}) \mid \mathcal{U}^-_{\infty}(\mathbb{N},\overline{\mathfrak{vp}})
\end{aligned}$$

Fig. 3: Grammar for our proof objects.

3 Proof Object Checker

This section introduces our proof objects and their checker: finite data-representations of our proof system's trees, and an algorithm that certifies if a given proof object faithfully proves the satisfaction or violation of a formula under a given assignment and stream. We discuss the soundness, completeness, and executability of these constructions.

To algorithmically manipulate proof trees, we define an explicit representation of *satisfactions* $\mathfrak{sp}$ and *violations* $\mathfrak{vp}$ via the grammar in Figure 3, where each constructor corresponds to a proof rule of our proof system (Figure 2), and its arguments represent subproofs and parameters that are part of a rule. The disjoint union $\mathfrak{p} = \mathfrak{sp} \uplus \mathfrak{vp}$ is our type of *proof objects*. The proof object $\forall^+$ requires information about satisfactions for all domain elements $d \in \mathbb{D}$ which we finitely represent with our *valued partitions* $P \in \biguplus_{\mathbb{D}}(\mathfrak{sp})$. Recall that a partition P of a set A is a collection of non-empty, pair-wise disjoint subsets of A that cover A. That is, $D_i \cap D_j = \varnothing$ for $D_i, D_j \in P$ with $D_i \neq D_j$ and $\bigcup P = A$. Partitions enable us to finitely represent all elements of the domain using finitely many finite sets and the co-finite complement of their union. In valued partitions $P \in \biguplus_{\mathbb{D}}(\mathfrak{sp})$, each set in the partition is tagged with a satisfaction explaining why its elements satisfy the argument of a universally quantified formula. Formally, our valued partitions $P \in \biguplus_{\mathbb{D}}(Z)$ are lists of pairs of a set D_i and a value $z \in Z$ from a given set Z such that the sets D_i form a partition of $\mathbb{D}$. Similarly, $\exists^-$ stores a valued partition $P \in \biguplus_{\mathbb{D}}(\mathfrak{vp})$ of violations.

Our proof objects $p \in \mathfrak{p}$ represent satisfactions or violations at a certain time-point. We define a function $\mathtt{tp}(p)$ (omitted) to compute this time-point. Either this information can be obtained recursively (e.g., $\mathtt{tp}(\bigcirc^+(p)) = \mathtt{tp}(p) - 1$) or, in cases where it cannot, it is stored directly in the proof objects (e.g., $\mathtt{tp}(tt^+(i)) = i$). We lift $\mathtt{tp}$ to sequences (yielding sequences of time-points) and valued partitions as $\mathtt{tp}(P) = \mathtt{tp}(p_1)$, where (D_1, p_1) is the partition P's first entry. To characterize *valid* proof objects, we define the relation $\vdash_\sigma$ (Figure 4) that checks that proof objects constitute correct applications of our proof system's rules. Here, $\vdash$ is not an executable algorithm yet since the proof objects $\forall^+$ and $\exists^-$ require a recursive call for each element of each set in the partition, and at least one of such sets is infinite for infinite domains. We will improve on this aspect after an example.

Example 2. The following violation proof object p at time-point 3 (i.e., $\mathtt{tp}(p) = 3$) is valid for formula φ on stream σ from Example 1 (i.e., $v, p \vdash_\sigma \varphi$ for any assignment v):

$$\begin{aligned}
p &= \forall^-(a, \mathtt{Charlie}, \forall^-(f, 152, p^-_{\rightarrow})), \text{ where} \\
p^-_{\rightarrow} &= \rightarrow^-(p^+_L, p^-_{\blacklozenge}),\ p^+_L = p^+(3, \mathsf{publish}, [a, f]), \\
p^-_{\blacklozenge} &= \blacklozenge^-(3, [\exists^-(x, [(\mathbb{D}, p^-_2)]), \exists^-(x, [(\mathbb{D}, p^-_3)])]), \text{ and} \\
p^-_i &= \wedge^-_R(p^-(i, \mathsf{approve}, [m, f])) \text{ for } i \in \{2, 3\}.
\end{aligned}$$

$$
\begin{array}{l|l|l|l}
v, tt^{+}(i) \vdash tt & & v, ff^{-}(i) \vdash ff & \\
v, p^{+}(i,\mathsf{p},\bar{t}) \vdash \mathsf{p}(\bar{t}) & \text{iff } \mathsf{p}([\![\bar{t}]\!]_v) \in \Gamma_i & v, p^{-}(i,\mathsf{p},\bar{t}) \vdash \mathsf{p}(\bar{t}) & \text{iff } \mathsf{p}([\![\bar{t}]\!]_v) \notin \Gamma_i \\
v, \approx^{+}(i,x,c) \vdash x \approx c & \text{iff } v(x) = c & v, \approx^{-}(i,x,c) \vdash x \approx c & \text{iff } v(x) \neq c \\
v, \neg^{+}(vp) \vdash \neg\alpha & \text{iff } v, vp \vdash \alpha & v, \neg^{-}(sp) \vdash \neg\alpha & \text{iff } v, sp \vdash \alpha \\
v, \rightarrow^{+}_{L}(vp) \vdash \alpha \rightarrow \beta & \text{iff } v, vp \vdash \alpha & v, \wedge^{-}_{L}(vp) \vdash \alpha \wedge \beta & \text{iff } v, vp \vdash \alpha \\
v, \rightarrow^{+}_{R}(sp) \vdash \alpha \rightarrow \beta & \text{iff } v, sp \vdash \beta & v, \wedge^{-}_{R}(vp) \vdash \alpha \wedge \beta & \text{iff } v, vp \vdash \beta \\
v, \exists^{+}(x,d,sp) \vdash \exists x.\ \alpha & \text{iff } v[x \mapsto d], sp \vdash \alpha & v, \vee^{+}_{L}(sp) \vdash \alpha \vee \beta & \text{iff } v, sp \vdash \alpha \\
v, \forall^{-}(x,d,vp) \vdash \forall x.\ \alpha & \text{iff } v[x \mapsto d], vp \vdash \alpha & v, \vee^{+}_{R}(sp) \vdash \alpha \vee \beta & \text{iff } v, sp \vdash \beta
\end{array}
$$

$v, \wedge^{+}(sp_1, sp_2) \vdash \alpha \wedge \beta$ iff $v, sp_1 \vdash \alpha$ and $v, sp_2 \vdash \beta$ and $\mathsf{tp}(sp_1) = \mathsf{tp}(sp_2)$

$v, \vee^{-}(vp_1, vp_2) \vdash \alpha \vee \beta$ iff $v, vp_1 \vdash \alpha$ and $v, vp_2 \vdash \beta$ and $\mathsf{tp}(vp_1) = \mathsf{tp}(vp_2)$

$v, \rightarrow^{-}(sp_1, vp_2) \vdash \alpha \rightarrow \beta$ iff $v, sp_1 \vdash \alpha$ and $v, vp_2 \vdash \beta$ and $\mathsf{tp}(sp_1) = \mathsf{tp}(vp_2)$

$v, \forall^{+}(x,P) \vdash \forall x.\ \alpha$ iff $\bigwedge(D_k, sp_k) \in P.\ \mathsf{tp}(sp_k) = \mathsf{tp}(P)$ and $\bigwedge d \in D_k.\ v[x \mapsto d], sp_k \vdash \alpha$

$v, \exists^{-}(x,P) \vdash \exists x.\ \alpha$ iff $\bigwedge(D_k, vp_k) \in P.\ \mathsf{tp}(vp_k) = \mathsf{tp}(P)$ and $\bigwedge d \in D_k.\ v[x \mapsto d], vp_k \vdash \alpha$

$v, \bullet^{+}(sp) \vdash \bullet_I \alpha$ iff $v, sp \vdash \alpha$ and $\mathsf{tp}(\bullet^{+}(sp)) = \mathsf{tp}(sp) + 1$ and $\tau_{\mathsf{tp}(\bullet^{+}(sp))} - \tau_{\mathsf{tp}(sp)} \in I$

$v, \bigcirc^{+}(sp) \vdash \bigcirc_I \alpha$ iff $v, sp \vdash \alpha$ and $\mathsf{tp}(\bigcirc^{+}(sp)) + 1 = \mathsf{tp}(sp)$ and $\tau_{\mathsf{tp}(sp)} - \tau_{\mathsf{tp}(\bigcirc^{+}(sp))} \in I$

$v, \blacklozenge^{+}(i,sp) \vdash \blacklozenge_I \alpha$ iff $v, sp \vdash \alpha$ and $i \geq \mathsf{tp}(sp)$ and $\tau_i - \tau_{\mathsf{tp}(sp)} \in I$

$v, \lozenge^{+}(i,sp) \vdash \lozenge_I \alpha$ iff $v, sp \vdash \alpha$ and $i \leq \mathsf{tp}(sp)$ and $\tau_{\mathsf{tp}(sp)} - \tau_i \in I$

$v, \blacksquare^{+}(i,\overline{sp}) \vdash \blacksquare_I \alpha$ iff $(\bigwedge sp \in \overline{sp}.\ v, sp \vdash \alpha)$ and $\mathsf{tp}(\overline{sp}) = \left[\mathsf{E}^{\mathsf{p}}_i(I), \mathsf{L}^{\mathsf{p}}_i(I)\right]$ and $\tau_i \geq \tau_0 + I$

$v, \square^{+}(i,\overline{sp}) \vdash \square_I \alpha$ iff $(\bigwedge sp \in \overline{sp}.\ v, sp \vdash \alpha)$ and $\mathsf{tp}(\overline{sp}) = \left[\mathsf{E}^{\mathsf{f}}_i(I), \mathsf{L}^{\mathsf{f}}_i(I)\right]$

$v, \mathcal{S}^{+}(sp,\overline{sp}) \vdash \alpha\ \mathcal{S}_I\ \beta$ iff $(\bigwedge sp' \in \overline{sp}.\ v, sp' \vdash \alpha)$ and $v, sp \vdash \beta$ and $\mathsf{tp}(\mathcal{S}^{+}(sp,\overline{sp})) \geq \mathsf{tp}(sp)$
and $\mathsf{tp}(\overline{sp}) = \left[\mathsf{tp}(sp) + 1, \mathsf{tp}(\mathcal{S}^{+}(sp,\overline{sp}))\right]$ and $\tau_{\mathsf{tp}(\mathcal{S}^{+}(sp,\overline{sp}))} - \tau_{\mathsf{tp}(sp)} \in I$

$v, \mathcal{U}^{+}(sp,\overline{sp}) \vdash \alpha\ \mathcal{U}_I\ \beta$ iff $(\bigwedge sp' \in \overline{sp}.\ v, sp' \vdash \alpha)$ and $v, sp \vdash \beta$ and $\mathsf{tp}(\mathcal{U}^{+}(sp,\overline{sp})) \leq \mathsf{tp}(sp)$
and $\mathsf{tp}(\overline{sp}) = \left[\mathsf{tp}(\mathcal{U}^{+}(sp,\overline{sp})), \mathsf{tp}(sp)\right)$ and $\tau_{\mathsf{tp}(sp)} - \tau_{\mathsf{tp}(\mathcal{U}^{+}(sp,\overline{sp}))} \in I$

$$
\begin{array}{l|l|l|l}
v, \bullet^{-}_{0} \vdash \bullet_I \alpha & \text{iff } \mathsf{tp}(\bullet^{-}_{0}) = 0 & v, \bullet^{-}_{<I}(i) \vdash \bullet_I \alpha & \text{iff } i > 0 \text{ and } \tau_i - \tau_{i-1} < I \\
v, \bigcirc^{-}_{<I}(i) \vdash \bigcirc_I \alpha & \text{iff } \tau_{i+1} - \tau_i < I & v, \bullet^{-}_{>I}(i) \vdash \bullet_I \alpha & \text{iff } i > 0 \text{ and } \tau_i - \tau_{i-1} > I \\
v, \bigcirc^{-}_{>I}(i) \vdash \bigcirc_I \alpha & \text{iff } \tau_{i+1} - \tau_i > I & v, \blacksquare^{+}_{<I}(i) \vdash \blacksquare_I \alpha & \text{iff } \tau_i < \tau_0 + I \\
v, \blacklozenge^{-}_{<I}(i) \vdash \blacklozenge_I \alpha & \text{iff } \tau_i < \tau_0 + I & v, \mathcal{S}^{-}_{<I}(i) \vdash \alpha\ \mathcal{S}_I\ \beta & \text{iff } \tau_i < \tau_0 + I
\end{array}
$$

$v, \bullet^{-}(vp) \vdash \bullet_I \alpha$ iff $v, vp \vdash \alpha$ and $\mathsf{tp}(\bullet^{-}(vp)) = \mathsf{tp}(vp) + 1$

$v, \bigcirc^{-}(vp) \vdash \bigcirc_I \alpha$ iff $v, vp \vdash \alpha$ and $\mathsf{tp}(\bigcirc^{-}(vp)) + 1 = \mathsf{tp}(vp)$

$v, \blacklozenge^{-}(i,\overline{vp}) \vdash \blacklozenge_I \alpha$ iff $(\bigwedge vp \in \overline{vp}.\ v, vp \vdash \alpha)$ and $\mathsf{tp}(\overline{vp}) = \left[\mathsf{E}^{\mathsf{p}}_i(I), \mathsf{L}^{\mathsf{p}}_i(I)\right]$ and $\tau_i \geq \tau_0 + I$

$v, \lozenge^{-}(i,\overline{vp}) \vdash \lozenge_I \alpha$ iff $(\bigwedge vp \in \overline{vp}.\ v, vp \vdash \alpha)$ and $\mathsf{tp}(\overline{vp}) = \left[\mathsf{E}^{\mathsf{f}}_i(I), \mathsf{L}^{\mathsf{f}}_i(I)\right]$

$v, \blacksquare^{-}(i,vp) \vdash \blacksquare_I \alpha$ iff $v, vp \vdash \alpha$ and $i \geq \mathsf{tp}(vp)$ and $\tau_i - \tau_{\mathsf{tp}(vp)} \in I$

$v, \square^{-}(i,vp) \vdash \square_I \alpha$ iff $v, vp \vdash \alpha$ and $i \leq \mathsf{tp}(vp)$ and $\tau_{\mathsf{tp}(vp)} - \tau_i \in I$

$v, \mathcal{S}^{-}_{\infty}(i,\overline{vp}) \vdash \alpha\ \mathcal{S}_I\ \beta$ iff $(\bigwedge vp \in \overline{vp}.\ v, vp \vdash \beta)$ and $\mathsf{tp}(\overline{vp}) = \left[\mathsf{E}^{\mathsf{p}}_i(I), \mathsf{L}^{\mathsf{p}}_i(I)\right]$ and $\tau_i \geq \tau_0 + I$

$v, \mathcal{S}^{-}(i,vp,\overline{vp}) \vdash \alpha\ \mathcal{S}_I\ \beta$ iff $(\bigwedge vp \in \overline{vp}.\ v, vp \vdash \beta)$ and $v, vp \vdash \alpha$ and $\mathsf{E}^{\mathsf{p}}_i(I) \leq \mathsf{tp}(vp) \leq i$
and $\mathsf{tp}(\overline{vp}) = \left[\mathsf{tp}(vp), \mathsf{L}^{\mathsf{p}}_i(I)\right]$ and $\tau_i \geq \tau_0 + I$

$v, \mathcal{U}^{-}_{\infty}(i,\overline{vp}) \vdash \alpha\ \mathcal{U}_I\ \beta$ iff $(\bigwedge vp \in \overline{vp}.\ v, vp \vdash \beta)$ and $\mathsf{tp}(\overline{vp}) = \left[\mathsf{E}^{\mathsf{f}}_i(I), \mathsf{L}^{\mathsf{f}}_i(I)\right]$

$v, \mathcal{U}^{-}(i,vp,\overline{vp}) \vdash \alpha\ \mathcal{U}_I\ \beta$ iff $(\bigwedge vp \in \overline{vp}.\ v, vp \vdash \beta)$ and $v, vp \vdash \alpha$ and $i \leq \mathsf{tp}(vp) < \mathsf{L}^{\mathsf{f}}_i(I)$
and $\mathsf{tp}(\overline{vp}) = \left[\mathsf{E}^{\mathsf{f}}_i(I), \mathsf{tp}(vp)\right]$

Fig. 4: Proof checker for a fixed stream $\sigma = \langle \tau_i, \Gamma_i \rangle_{i \in \mathbb{N}}$.

Indeed, we use the definition in Figure 4 to certify that $v, p \vdash_\sigma \varphi$:

$$
\begin{aligned}
v, p \vdash \varphi \text{ iff } & v[a \mapsto \texttt{Charlie}], \forall^{-}(f, 152, p^{-}_{\rightarrow}) \vdash \forall f.\ \varphi_L \rightarrow \varphi_R \\
\text{iff } & v[a \mapsto \texttt{Charlie}, f \mapsto 152], p^{-}_{\rightarrow} \vdash \varphi_L \rightarrow \varphi_R \\
\text{iff } & v[a \mapsto \texttt{Charlie}, f \mapsto 152], p^{+}_{L} \vdash \varphi_L \text{ and } \mathsf{tp}(p^{+}_{L}) = 3 = \mathsf{tp}(p^{-}_{\blacklozenge}) \text{ and} \\
& \quad v[a \mapsto \texttt{Charlie}, f \mapsto 152], p^{-}_{\blacklozenge} \vdash \varphi_R \\
\text{iff } & v[a \mapsto \texttt{Charlie}, f \mapsto 152], \exists^{-}(x, [(\mathbb{D}, p^{-}_{i})]) \vdash \varphi_{\exists} \texttt{ for } i \in \{2,3\} \\
\text{iff } & v[a \mapsto \texttt{Charlie}, f \mapsto 152, x \mapsto d], p^{-}_{i} \vdash \varphi_1 \wedge \varphi_2 \text{ for all } d \in \mathbb{D}, i \in \{2,3\} \\
\text{iff } & \texttt{approve}(\texttt{d}, 152) \notin \Gamma_i \text{ for all } d \in \mathbb{D}, i \in \{2,3\}, \text{ which is true.}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{LRTP}\ i\ tt &&= \mathsf{LRTP}\ i\ ff = \mathsf{LRTP}\ i\ (\mathsf{p}(\bar{t})) = \mathsf{LRTP}\ i\ (x \approx c) = i,\\
&\mathsf{LRTP}\ i\ (Q\,x.\ \alpha) &&= \mathsf{LRTP}\ i\ (\neg\alpha) = \mathsf{LRTP}\ i\ \alpha \text{ for } Q \in \{\forall, \exists\},\\
&\mathsf{LRTP}\ i\ (\alpha \oplus \beta) &&= \mathsf{max}\ (\mathsf{LRTP}\ i\ \alpha)\ (\mathsf{LRTP}\ i\ \beta) \text{ for } \oplus \in \{\vee, \wedge, \rightarrow\},\\
&\mathsf{LRTP}\ i\ (\bullet_I \alpha) &&= \mathsf{LRTP}\ (i-1)\ \alpha, \qquad \mathsf{LRTP}\ i\ (\bigcirc_I \alpha) = \mathsf{LRTP}\ (i+1)\ \alpha,\\
&\mathsf{LRTP}\ i\ (\Diamond_I \alpha) &&= \mathsf{LRTP}\ i\ (\Box_I \alpha) = \mathsf{LRTP}\ (\mathsf{L}^{\mathrm{f}}_i(I))\ \alpha,\\
&\mathsf{LRTP}\ i\ (\blacklozenge_I \alpha) &&= \mathsf{LRTP}\ i\ (\blacksquare_I \alpha) = \mathsf{LRTP}\ (\mathsf{LTP}^{\mathrm{past}}_i\ I)\ \alpha,\\
&\mathsf{LRTP}\ i\ (\alpha\ \mathcal{S}_I\ \beta) &&= \mathsf{max}\ (\mathsf{LRTP}\ i\ \alpha)\ (\mathsf{LRTP}\ (\mathsf{LTP}^{\mathrm{past}}_i\ I)\ \beta),\\
&\mathsf{LRTP}\ i\ (\alpha\ \mathcal{U}_I\ \beta) &&= \mathsf{max}\ (\mathsf{LRTP}\ (\mathsf{L}^{\mathrm{f}}_i(I) - 1)\ \alpha)\ (\mathsf{LRTP}\ (\mathsf{L}^{\mathrm{f}}_i(I))\ \beta), \text{ where}\\
&\mathsf{LTP}^{\mathrm{past}}_i\ I &&= \big(\text{if } \tau_i \geq \tau_0 + I \text{ then } \mathsf{L}^{\mathrm{p}}_i(I) \text{ else } 0\big).
\end{aligned}
$$

Fig. 5: The formula's latest relevant time-point at i for a fixed stream $\sigma = \langle \tau_i, \Gamma_i \rangle_{i \in \mathbb{N}}$.

We implicitly use in the above the true statements $\mathsf{publish}(\texttt{Charlie}, 152) \in \Gamma_3$, $0 \leq \tau_3$, and $\mathsf{tp}([\exists^-(x, [(\mathbb{D}, p_2^-)]), \exists^-(x, [(\mathbb{D}, p_3^-)])]) = [2,3] = [\mathsf{E}^{\mathrm{p}}_i(I), \mathsf{L}^{\mathrm{p}}_i(I)]$. □

Theorem 2. *Fix a stream σ. The relation $\vdash$ is sound and complete in the sense that $v, i \vDash \alpha$ iff there is a satisfaction sp such that $v, sp \vdash \alpha$ and $\mathsf{tp}(sp) = i$. Similarly $v, i \nvDash \alpha$ iff there is a violation vp such that $v, vp \vdash \alpha$ and $\mathsf{tp}(vp) = i$.*

We have established the above result in Isabelle. Below we sketch our overall approach and highlight the main challenge. We show both soundness and completeness by relating proof object validity ($\vdash$) to the proof system ($\vdash^+$ and $\vdash^-$), which we already know to be sound and complete, i.e., related to the semantics $\vDash$. Soundness is easy as the proof object directly provides the recipe for correctly applying the proof system rules. Formally, if $v, sp \vdash \alpha$ then $v, \mathsf{tp}(sp) \vdash^+ \alpha$, and if $v, vp \vdash \alpha$, then $v, \mathsf{tp}(vp) \vdash^- \alpha$. The proof follows immediately by mutual induction on the proof object structure.

Completeness of $\vdash$ requires us to provide a valid proof object just from knowing $v, i \vdash^+ \alpha$ or $v, i \vdash^- \alpha$. We proceed by mutual induction on the derivations of $\vdash^+$ and $\vdash^-$. Only two of the quantifier cases are challenging. For the satisfaction of the universal quantifier (and similarly for the violation of $\exists$), we must construct a valued partition with finitely many subproofs. However, the induction hypothesis yields a separate proof object for every element of the domain $\mathbb{D}$, and all these proof objects may a priori be different. The crucial observation is that for all values that do not occur in the stream (or at least are not in reach of α with respect to a time-point i) we can reuse the same proof object. To formalize this observation, we first define a formula's *active domain at i*, written $\mathsf{AD}_i(\alpha)$, which formalizes the in "reach" intuition. To this end, we first define the *latest relevant time point* ($\mathsf{LRTP}\ i\ \alpha$) of α at i (Figure 5). Intuitively, $\mathsf{LRTP}\ i\ \alpha$ marks the largest time-point that may influence α's satisfiability at i. It exists, because we assume that future temporal operators have bounded intervals. Based on this, we define:

$$\mathsf{AD}_i(\alpha) = \mathbb{D}(\alpha) \cup \bigcup\nolimits_{k \leq \mathsf{LRTP}\ i\ \alpha} \{d \mid d \text{ appears in some } \mathsf{p}(d_1, \ldots, d_n) \in \Gamma_k\}.$$

Here we write $\mathbb{D}(\alpha)$ for the set of constants $d \in \mathbb{D}$ occurring in subformulas of the form $x \approx d$ in α. (In contrast to constants occurring in atomic predicates, constants occurring in equalities may appear in α's satisfying assignments even if they are not part of the trace.) Note that $\mathsf{AD}_i(\alpha)$ is finite. The active domain lets us formalize the key observation:

Lemma 1. *Fix a stream σ, a formula α, a proof p, and two assignments v and v'. Let $i = \mathsf{tp}(p)$, $\mathsf{AD} = \mathsf{AD}_i(\alpha)$, and V be the set of α's free variables. Assume that v and v'*

may only disagree on V for values outside of the active domain at i, i.e.,

$$\forall x \in V.\ v(x) = v'(x) \vee (v(x) \notin \mathsf{AD} \wedge v'(x) \notin \mathsf{AD}).$$

Then, p's validity status is the same for both assignments, i.e., $v, p \vdash \alpha$ iff $v', p \vdash \alpha$.

We now can finish the $\forall^+$ case of the completeness proof. By the induction hypothesis, there is a satisfaction $p(d) \in \mathfrak{sp}$ for each domain element $d \in \mathbb{D}$. Moreover, $\{\{d\} \mid d \in \mathsf{AD}_i(\alpha)\} \cup \{\mathbb{D} \setminus \mathsf{AD}_i(\alpha)\}$ is a finite partition of $\mathbb{D}$. Hence, the list of pairings $(\{d\}, p(d))$ for each $d \in \mathsf{AD}_i(\alpha)$ and $(\mathbb{D} \setminus \mathsf{AD}_i(\alpha), p(z))$ for some $z \in \mathbb{D} \setminus \mathsf{AD}_i(\alpha)$ (which exists as $\mathbb{D}$ is infinite) is a valued partition. Moreover, all subproofs are valid for all the values contained in the partition sets by combining the induction hypothesis with the above congruence Lemma 1 (for $p = p(z)$), and thus so is the overall $\forall^+$ proof object.

Lastly, we address the executability issue. The validity relation $\vdash$ works with assignments v of values to variables. To avoid performing infinitely many recursive calls for the $\forall^+$ and $\exists^-$ proof objects we now will work with *set assignments* V of sets of values to variables. We define a validity relation $V, p \vdash \alpha$ based on set assignments. The definition is the same as the one of $v, p \vdash \alpha$ except for the predicate and the quantifier cases:

$$\begin{array}{ll}
V, p^+(i, \mathsf{p}, \bar{t}) \vdash \mathsf{p}(\bar{t}) & \text{iff } \{\mathsf{p}\} \times [\![\bar{t}]\!]_V \subseteq \Gamma_i \\
V, p^-(i, \mathsf{p}, \bar{t}) \vdash \mathsf{p}(\bar{t}) & \text{iff } \{\mathsf{p}\} \times [\![\bar{t}]\!]_V \subseteq \mathbb{D} \setminus \Gamma_i \\
V, \forall^+(x, P) \vdash \forall x.\ \alpha & \text{iff } \bigwedge(D_k, sp_k) \in P.\ \mathsf{tp}(sp_k) {=} \mathsf{tp}(P) \text{ and } V[x \mapsto D_k], sp_k \vdash \alpha \\
V, \exists^+(x, d, sp) \vdash \exists x.\ \alpha & \text{iff } V[x \mapsto \{d\}], sp \vdash \alpha
\end{array}$$

and dually for $\exists^-$ and $\forall^-$. Here, $[\![\bar{t}]\!]_V$ represents a transformation of the list of values $[\![\bar{t}]\!]_v$ to the set of all possible lists of values generated by V. Set assignments allow us to delay deciding values for quantifier subproofs to the predicate base case. Note that $\{\mathsf{p}\} \times [\![\bar{t}]\!]_V \subseteq \Gamma_i$ and $\{\mathsf{p}\} \times [\![\bar{t}]\!]_V \subseteq \mathbb{D} \setminus \Gamma_i$ are decidable because due to our partitions, we only encounter finite and co-finite sets. The set-assignment-based validity check is thus executable and thus provides the algorithm that we use as our formally verified proof object checker: $v, p \vdash \alpha = (\lambda x.\{v(x)\}), p \vdash \alpha$ (proved by induction on α using Lemma 1).

4 Partitioned Decision Trees

Our proof system is parameterized with an assignment, but in our monitoring approach we are interested in computing a proof object for every assignment. In this section, we introduce *partitioned decision trees* (PDTs), a specialized data structure for representing and efficiently manipulating variable assignments, inspired by the use of BDDs in runtime verification [17]. We want to represent functions of the form $f : \mathbb{D} \times \ldots \times \mathbb{D} \to \mathfrak{p}$, i.e., mappings from tuples of domain elements to proof trees, where each tuple corresponds to a variable assignment to the formula's free variables. As argued in the previous section, we are only interested in such functions with a finite range. Thus, we organize the domain into a finite number of subsets $\mathbb{D} \times \ldots \times \mathbb{D}$ such that each tuple element is partitioned separately (using valued partitions over the domain). As before, we work with finite and co-finite sets in the partition. PDTs $\mathbb{P}(A)$ are defined inductively as follows:

$$\mathbb{P}(A) = \mathsf{Leaf}\ A \mid \mathsf{Node}\ (\mathbb{V}, \biguplus_{\mathbb{D}}(\mathbb{P}(A)))$$

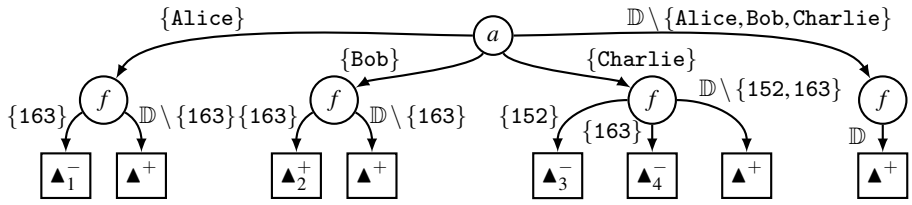

Fig. 6: Resulting PDT for our running example at time-point 3.

PDTs have leaves and nodes. Leaves store objects from the set A, while nodes store pairs of the form (x,P), where x is a variable and P, a valued partition of the domain storing PDTs. PDTs generalize binary decision trees along two dimensions. First, the branching of their nodes is not binary but follows a given partition of the infinite domain $\mathbb{D}$. Second, their leaves do not store Boolean values. Instead, they store arbitrary objects, even though we will mostly use them with proof objects $A = \mathbb{p}$. PDTs provide a way to organize the infinitely many possible variable assignments in a structured manner, storing only finitely many different proof objects. In monitoring, partitions will arise naturally, guided by the values occurring in the stream and assembled via operations that combine them.

Example 3. We continue the publish–approve example from Example 1. We consider the same stream but drop the top-level quantifiers from the formula φ: we only consider φ' with its free variables a and f. Figure 6 shows the PDT representing all assignments for φ' at time-point 3. The root node represents variable a, and the edges partition the values that a can take into the following domain subsets: $\{\texttt{Alice}\}, \{\texttt{Bob}\}, \{\texttt{Charlie}\}$, and $\mathbb{D} \setminus \{\texttt{Alice}, \texttt{Bob}, \texttt{Charlie}\}$. The second level is analogous for variable f. At every level of the PDT, the union of all choices cover the entire domain $\mathbb{D}$ (by definition of partitions) and the partitions may differ at every node. The leaves of the PDT are different proof trees (formally, proof objects) which we represent by small black triangles. For example, $\blacktriangle_3^-$ is the proof tree of φ''s violation shown in Example 1. In contrast, $\blacktriangle^+$ (occurring in multiple leaves) is the proof tree shown in Figure 7 of φ''s vacuous satisfaction: the left hand side of the implication ($\mathsf{publish}(a,f)$) is violated for any assignment v updated by following the path from the PDT's root to the respective leaf (e.g., taking $a = \texttt{Alice}$ and $f = 42 \in \mathbb{D} \setminus \{\texttt{163}\}$). □

$$\dfrac{\dfrac{\mathsf{publish}(a,f) \notin \Gamma_3}{v,3 \vdash^- \mathsf{publish}(a,f)}\ p^-}{v,3 \vdash^+ \varphi'}\ \rightarrow_L^+$$

Fig. 7: Proof tree $\blacktriangle^+$.

Since PDTs are a generalization of BDDs, we use similar functions to manipulate them. We list the most important ones, for partitions and PDTs in Figure 8, but we only show and discuss the implementation of apply2, merge2, and hide. Most PDT-functions are parameterized by a variable list $vs :: \overline{\mathbb{V}}$ fixing the variable order. The functions map_part and apply1 lift unary functions on objects to partitions and PDTs respectively.

The functions merge2 and apply2 do the same for binary functions; apply2 generalizes the well-known apply function on BDDs [16]. On leaves, apply2 maps f to the objects. When operating on a leaf and a node, apply2 pushes f partially applied to the leaf to the node's leaves using apply1. Finally, on pairs of nodes, it proceeds recursively depending which of x, y, and z are equal. The most interesting case, $x = y = z$ occurs when both PDTs partition the domain values for z in different ways. Thus, we must

$\mathsf{map\_part} :: (A \Rightarrow B) \Rightarrow \biguplus_{\mathbb{D}}(A) \Rightarrow \biguplus_{\mathbb{D}}(B)$
$\mathsf{merge2} :: (A \Rightarrow B \Rightarrow C) \Rightarrow \biguplus_{\mathbb{D}}(A) \Rightarrow \biguplus_{\mathbb{D}}(B) \Rightarrow \biguplus_{\mathbb{D}}(C)$
$\mathsf{merge2}\, f\, []\, P_2 = []$
$\mathsf{merge2}\, f\, ((D_1,\, v_1) \#\, P_1)\, P_2 =$
  $\underline{\mathsf{let}}\ P_3 = \mathsf{map\_filter}\ (\lambda(D_2,\, v_2).\ \underline{\mathsf{if}}\ D_1 \cap D_2 \neq \varnothing\ \underline{\mathsf{then}}\ \mathsf{Some}\ (D_1 \cap D_2,\, f\ v_1\ v_2)\ \underline{\mathsf{else}}\ \mathsf{None})\ P_2;$
    $P_4 = \mathsf{map\_filter}\ (\lambda(D_2,\, v_2).\ \underline{\mathsf{if}}\ D_2 \setminus D_1 \neq \varnothing\ \underline{\mathsf{then}}\ \mathsf{Some}\ (D_2 \setminus D_1,\, v_2)\ \underline{\mathsf{else}}\ \mathsf{None})\ P_2$
  $\underline{\mathsf{in}}\ P_3\ @\ \mathsf{merge2}\, f\, P_1\, P_4$
$\mathsf{pdt\_of} :: \overline{\mathbb{V}} \Rightarrow A \Rightarrow A \Rightarrow 2^{(\mathbb{V} \rightharpoonup \mathbb{D})} \Rightarrow \mathbb{P}(A)$ $\mathsf{split\_prod} :: \mathbb{P}(A \times B) \Rightarrow \mathbb{P}(A) \times \mathbb{P}(B)$
$\mathsf{apply1} :: \overline{\mathbb{V}} \Rightarrow (A \Rightarrow B) \Rightarrow \mathbb{P}(A) \Rightarrow \mathbb{P}(B)$ $\mathsf{split\_list} :: \mathbb{P}(\overline{A}) \Rightarrow \overline{\mathbb{P}(A)}$
$\mathsf{apply2} :: \overline{\mathbb{V}} \Rightarrow (A \Rightarrow B \Rightarrow C) \Rightarrow \mathbb{P}(A) \Rightarrow \mathbb{P}(B) \Rightarrow \mathbb{P}(C)$
$\mathsf{apply2}\ vs\, f\ (\mathsf{Leaf}\ l_1)\ (\mathsf{Leaf}\ l_2) = \mathsf{Leaf}\ (f\ l_1\ l_2)$
$\mathsf{apply2}\ vs\, f\ (\mathsf{Leaf}\ l_1)\ (\mathsf{Node}\ (x,\, P_2)) = \mathsf{Node}\ (x, \mathsf{map\_part}\ (\mathsf{apply1}\ vs\ (\lambda l_2.\, f\ l_1\ l_2))\ P_2)$
$\mathsf{apply2}\ vs\, f\ (\mathsf{Node}\ (x,\, P_1))\ (\mathsf{Leaf}\ l_2) = \mathsf{Node}\ (x, \mathsf{map\_part}\ (\mathsf{apply1}\ vs\ (\lambda l_1.\, f\ l_1\ l_2)\ P_1)$
$\mathsf{apply2}\ (z \#\, vs)\, f\ (\mathsf{Node}\ (x,\, P_1))\ (\mathsf{Node}\ (y,\, P_2)) =$
  $\underline{\mathsf{if}}\ x = z\ \underline{\mathsf{and}}\ y = z\ \underline{\mathsf{then}}\ \mathsf{Node}\ (z,\ \mathsf{merge2}\ (\mathsf{apply2}\ vs\, f)\ P_1\ P_2)$
  $\underline{\mathsf{else}}\ \underline{\mathsf{if}}\ x = z\ \underline{\mathsf{then}}\ \mathsf{Node}\ (x, \mathsf{map\_part}\ (\lambda l.\ \mathsf{apply2}\ vs\, f\ l\ (\mathsf{Node}\ (y,\, P_2)))\ P_1)$
  $\underline{\mathsf{else}}\ \underline{\mathsf{if}}\ y = z\ \underline{\mathsf{then}}\ \mathsf{Node}\ (y, \mathsf{map\_part}\ (\lambda r.\ \mathsf{apply2}\ vs\, f\ (\mathsf{Node}\ (x,\, P_1))\ r)\ P_2)$
  $\underline{\mathsf{else}}\ \mathsf{apply2}\ vs\, f\ (\mathsf{Node}\ (x,\, P_1))\ (\mathsf{Node}\ (y,\, P_2))$
$\mathsf{apply3} :: \overline{\mathbb{V}} \Rightarrow (A \Rightarrow B \Rightarrow C \Rightarrow D) \Rightarrow \mathbb{P}(A) \Rightarrow \mathbb{P}(B) \Rightarrow \mathbb{P}(C) \Rightarrow \mathbb{P}(D)$
$\mathsf{hide} :: \overline{\mathbb{V}} \Rightarrow (A \Rightarrow A) \Rightarrow (\biguplus_{\mathbb{D}}(A) \Rightarrow A) \Rightarrow \mathbb{P}(A) \Rightarrow \mathbb{P}(A)$
$\mathsf{hide}\ vs\ leaf\ node\ (\mathsf{Leaf}\ l) = \mathsf{Leaf}\ (leaf\ l)$
$\mathsf{hide}\ [z]\ leaf\ node\ (\mathsf{Node}\ (x,\, P)) = \mathsf{Leaf}\ (node\ (\mathsf{map\_part}\ \mathsf{unleaf}\ P))$
$\mathsf{hide}\ (z \#\, vs)\ leaf\ node\ (\mathsf{Node}\ (x,\, P)) =$
  $\underline{\mathsf{if}}\ x = z\ \underline{\mathsf{then}}\ \mathsf{Node}\ (z,\ \mathsf{map\_part}\ (\mathsf{hide}\ vs\ leaf\ node)\ P)\ \underline{\mathsf{else}}\ \mathsf{hide}\ vs\ leaf\ node\ (\mathsf{Node}\ (x,\, P))$

Fig. 8: Selected functions on partitions and PDTs.

combine both partitions. For this, we use merge2 that takes two valued partitions P_1 and P_2, and iteratively "erodes" P_2 by intersecting its elements with the sets in P_1 while applying f. Since both P_1 and P_2 cover $\mathbb{D}$, the resulting set of intersections is a valued partition. The function apply3 analogously combines three PDTs into one.

The function hide traverses the PDT similarly to apply1, while eliminating the last variable in the given variable list. It uses two higher-order arguments, in case the last layer is present (*node*) or absent (*leaf*). The function pdt_of *vs A B V* constructs a PDT from a finite set of partial assignments ($V :: 2^{(\mathbb{V} \rightharpoonup \mathbb{D})}$) using A for leaves reached by paths from the set, and B for the other leaves. Finally, the split_$*$ functions transpose a PDT storing pairs (lists of equal length) into a pair (list) of PDTs.

5 Monitoring Algorithm

We follow the typical online monitoring algorithm structure consisting of an initialization and a step (evaluation) function [25, 30]. The initializer init (omitted as standard) computes our monitor's initial state $s \in \mathbb{S}$ from an MFOTL specification α. Figure 9 shows an excerpt of our monitor's state, which recursively follows the formula structure and augments some operators with additional information, such as buffers storing verdicts from subformulas ($\mathbb{B}_2$ for $\wedge$ and $\mathbb{B}_3$ for $\mathcal{S}$) or an operator-specific state ($\mathbb{S}_{saux}$ for $\mathcal{S}$).

$\mathbb{B}_2 = \overline{\mathbb{P}(\mathfrak{p})} \times \overline{\mathbb{P}(\mathfrak{p})} \qquad \mathbb{B}_3 = \overline{\mathbb{P}(\mathfrak{p})} \times \overline{\mathbb{P}(\mathfrak{p})} \times \overline{\mathbb{N} \times \mathbb{N}} \qquad \mathbb{S}_{saux} = \ldots$
$\mathbb{S} = \mathsf{MPred}\ \mathbb{E}\ \overline{\mathbb{T}} \mid \mathsf{MAnd}\ \mathbb{S}\ \mathbb{S}\ \mathbb{B}_2 \mid \mathsf{MExists}\ \mathbb{E}\ \mathbb{S} \mid \mathsf{MSince}\ \mathbb{I}\ \mathbb{S}\ \mathbb{S}\ \mathbb{B}_3\ (\mathbb{P}(\mathbb{S}_{saux})) \mid \cdots$

$\mathsf{eval} :: \overline{\mathbb{V}} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{E} \times \overline{\mathbb{D}} \Rightarrow \mathbb{S} \Rightarrow \overline{\mathbb{P}(\mathfrak{p})} \times \mathbb{S}$
$\mathsf{eval}\ vs\ \tau\ i\ \Gamma\ (\mathsf{MPred}\ \mathsf{p}\ ts) =$
  $\underline{\mathsf{let}}\ e = \mathsf{pdt\_of}\ (\mathsf{filter}\ (\lambda v.\ v \in \mathsf{fv}(ts))\ vs)\ (p^+(i,\mathsf{p},ts))\ (p^-(i,\mathsf{p},ts))$
    $\{\sigma \mid \exists ds.\ \mathsf{p}(ds) \in \Gamma \wedge \mathsf{match}\ ts\ ds = \mathsf{Some}\ \sigma\}\ \underline{\mathsf{in}}\ ([e], \mathsf{MPred}\ \mathsf{p}\ ts)$
$\mathsf{eval}\ vs\ \tau\ i\ \Gamma\ (\mathsf{MAnd}\ s_1\ s_2\ buf) = \underline{\mathsf{let}}\ (es_1, s_1') = \mathsf{eval}\ vs\ \tau\ i\ \Gamma\ s_1; \quad (es_2, s_2') = \mathsf{eval}\ vs\ \tau\ i\ \Gamma\ s_2;$
  $(es, buf') = \mathsf{buf2\_take}\ (\mathsf{apply2}\ vs\ \mathsf{do\_and})\ (\mathsf{buf2\_add}\ buf\ es_1\ es_2)\ \underline{\mathsf{in}}\ (es, \mathsf{MAnd}\ s_1'\ s_2'\ buf')$
$\mathsf{eval}\ vs\ \tau\ i\ \Gamma\ (\mathsf{MExists}\ x\ s) = \underline{\mathsf{let}}\ (es, s') = \mathsf{eval}\ (vs@[x])\ \tau\ i\ \Gamma\ s$
  $\underline{\mathsf{in}}\ (\mathsf{map}\ (\mathsf{hide}\ (vs@[x])\ (\mathsf{do\_exists\_leaf}\ x)\ (\mathsf{do\_exists\_node}\ x))\ es, \mathsf{MExists}\ x\ s')$
$\mathsf{eval}\ vs\ \tau\ i\ \Gamma\ (\mathsf{MSince}\ I\ s_1\ s_2\ buf\ saux) =$
  $\underline{\mathsf{let}}\ (es_1, s_1') = \mathsf{eval}\ vs\ \tau\ i\ \Gamma\ s_1; \quad (es_2, s_2') = \mathsf{eval}\ vs\ \tau\ i\ \Gamma\ s_2;$
    $(es, buf', saux') = \mathsf{buf2t\_take}\ (\lambda e_1\ e_2\ (\tau,\ i)\ saux.$
      $\underline{\mathsf{let}}\ (saux', es') = \mathsf{split\_prod}\ (\mathsf{apply3}\ vs\ (\mathsf{update\_since}\ I\ \tau\ i)\ e_1\ e_2\ saux')$
      $\underline{\mathsf{in}}\ (saux', \mathsf{split\_list}\ es'))\ (\mathsf{buf2t\_add}\ buf\ es_1\ es_2\ [(\tau, i)])\ saux$
  $\underline{\mathsf{in}}\ (es, \mathsf{MSince}\ I\ s_1'\ s_2'\ buf'\ saux')$

Fig. 9: Involved types and selected cases of the monitor's eval function

$\mathsf{do\_exists\_leaf}\ x\ p = \underline{\mathsf{if}}\ p \in \mathfrak{sp}\ \underline{\mathsf{then}}\ \exists^+(x, d \leftarrow \mathbb{D}, p)\ \underline{\mathsf{else}}\ \exists^-(x, [(\mathbb{D}, p)])$
$\mathsf{do\_exists\_node}\ x\ P = \underline{\mathsf{if}}\ \bigvee(D_i,\ p) \in P.\ p \in \mathfrak{sp}$
  $\underline{\mathsf{then}}\ \mathsf{min}\ (\mathsf{map\_filter}\ (\lambda(D_i,\ p).\ \underline{\mathsf{if}}\ p \in \mathfrak{sp}\ \underline{\mathsf{then}}\ \mathsf{Some}\ (\exists^+(x, d \leftarrow D_i, p)))\ \underline{\mathsf{else}}\ \mathsf{None})\ P)$
  $\underline{\mathsf{else}}\ \exists^-(x, P)$

Fig. 10: Functions do_exists_leaf and do_exists_node.

Our function eval, partly shown in Figure 9, takes as inputs a new time-point i (along with its time-stamp τ and database Γ) and a monitor state s and outputs the next state s' and a list of PDTs of proof objects as verdicts. (In addition, eval keeps track of the variable ordering used in PDTs via the parameter vs.) Lists in the output are necessary because delays may occur for (bounded) future operators and a single time-point might trigger multiple outputs. Our algorithm extends Lima et al.'s algorithm [25] computing proof trees for MTL. We highlight our key additions to eval and the state Figure 9 in gray.

We focus on the predicate, conjunction, existential quantifier, and since cases. In the predicate case, we find all partial assignments σ mapping the predicate's variables to the values ds, so that $\mathsf{p}(ds) \in \Gamma$. We reuse VeriMon's match function [30] to compute such partial assignments. We convert this set of assignments to a PDT using pdt_of. In the resulting PDT, matching assignments lead to leaves using the satisfaction proof $p^+(i, \mathsf{p}, ts)$, whereas the others lead to the corresponding violation proof $p^-(i, \mathsf{p}, ts)$.

The conjunction case is taken almost without changes from Lima et al.'s [25] MTL algorithm. We reuse the buffering functions buf2_add and buf2_take. The first adds partial results to the buffer, while the second combines these results and dequeues them once both subformulas have produced results for a time-point. The only difference is that our buffers store PDTs of proof objects, whereas the MTL algorithm works with propositional proof objects. Accordingly, we reuse the Lima et al.'s function $\mathsf{do\_and} :: \mathfrak{p} \Rightarrow \mathfrak{p} \Rightarrow \mathfrak{p}$ to combine two proof objects conjunctively, but lift it to PDTs using apply2.

The quantifier cases are a new addition of our work. As both cases proceed dually, we focus on $\exists x.\, \alpha$ formulas. Considering that α may have one more free variable than $\exists x.\, \alpha$, the recursive call appends x to the variable list ordering. The recursive call's output

is processed using our function hide to eliminate the quantified variable. The interesting cases occur near the leaves of α's PDTs. If x is not present, hide will encounter a leaf, i.e., a proof object, and use the function do_exists_leaf (Figure 10) to perform a case distinction: satisfactions result in a satisfaction ($\mathfrak{sp}$) of $\exists x.\, \alpha$ with an arbitrary element d of the domain as the witness (we write $x \leftarrow X$ to denote an arbitrarily chosen element x of a non-empty set X); violations result in a violation ($\mathfrak{vp}$) with the trivial partition. If x is present as the last decision node, then hide will use do_exists_node (Figure 10) to construct the proof object for $\exists x.\, \alpha$. It performs a case distinction whether a satisfaction proof is contained in the partition of this last node. If it is, $\exists x.\, \alpha$ is satisfied and we compute the smallest (in proof size) such satisfaction proof, taking as our witness an arbitrary element of the respective partition set. Otherwise, all leaves are violations and we obtain a violation proof of $\exists x.\, \alpha$.

To reuse Lima et al.'s [25] temporal operator evaluation, our state stores a PDT whose leaves are the auxiliary state of these algorithms (instead of proof objects). This allows us to keep the complex auxiliary state and its update unchanged. For example, we use apply3 to lift Lima et al.'s [25] update_since function to two PDTs storing proof objects for subformulas and a third one storing the auxiliary state. The resulting PDT has type $\mathbb{P}(\mathbb{S}_{saux} \times \overline{\mathfrak{p}})$, which we transpose into the desired $\mathbb{P}(\mathbb{S}_{saux}) \times \overline{\mathbb{P}(\mathfrak{p})}$ using split_prod and split_list.

6 Implementation and Case Study

We implement our algorithm in a new monitoring tool, called WHYMON [2]. Our implementation consists of 4 500 lines of OCaml code and incorporates an optimization of collapsing partition sets with the same stored values both in proof objects and in PDTs. Our formally verified checker contributes additional 1 700 lines of OCaml code generated from our Isabelle formalization, which itself comprises 6 400 lines of definitions and proofs. The checker's main function lifts the validity check of proof objects ($\vdash$) to PDTs, i.e., check : *trace* $\rightarrow$ *formula* $\rightarrow$ *pdt* $\rightarrow$ *bool*, and is used to certify WHYMON's output. WHYMON includes a visualization [3] implemented in React [20] that consists of 2 400 lines of JavaScript and invokes a JavaScript version of our monitor, generated by `Js_of_ocaml` [32]. Here, we consider the data race policy [18] that captures possible concurrency issues in multithreaded programs on a stream prefix generated by Raszyk [27, Section 4.3]. Furthermore, we consider Nokia's Data-collection Campaign [4], which comes with a stream prefix of around 5 million time-points [1], for which we focus on the *del-2-3* policy [12] controlling data propagation between databases. We describe a violation for each scenario highlighting the advantages of our approach.

Example 4. We first return to Example 3 in our visualization tool, depicted in Figure 11. The table includes TP (time-points), TS (time-stamps), and Values columns. The following columns show the topmost operator of φ''s subformulas or its predicate names (and their variables). In the Values column, for each of the already evaluated time-points, there is an associated button enclosing a ✓ (for satisfactions), or a ✗ (for violations) or both. After clicking on this button, we are presented with a dropdown menu (as in Figure 12) that corresponds to a partition. The listed values are the (potentially multiple) variable assignments of the resulting PDT for that specific time-point. The formula φ' contains two free variables, a and f, and to single out a verdict we must select one value for

each. In particular, at time-point 3 we select $a = \texttt{Charlie}$ and $f = 152$. Note that in the visualization we focus on readability and omit set parentheses. Moreover, **Other** denotes the complement of the listed values. After choosing the assignments, a Boolean verdict appears in the next column matching the topmost operator of φ', namely $\rightarrow$. Clicking on this Boolean verdict reveals and highlights the Boolean verdicts associated with its justification. The subformulas' columns of the current inspection are also highlighted. In this case, the implication is violated because the left side is satisfied, while the $\blacklozenge_{[0,7]}$ subformula is violated. We can explore this verdict further: the violation is justified by those of its subformula at time-points 2 and 3 (the time-points inside the interval are also highlighted). For each time-point, there is another dropdown menu where we can select an assignment for m. Here, the only listed value is **Any**, which corresponds to $\mathbb{D}$. Thus, the existential quantifier is violated because the subformula $\mathsf{approve}(m, 152)$ is violated for all values that m can be assigned to ($\mathbb{D}$), and all justifications are identical.

Data Race Detection Multithreaded programs are pervasive and hard to debug. In particular, they are prone to data races, which occur when two threads access (read or write to) a shared address concurrently and at least one of these accesses is a write. Locking mechanisms that synchronize access to variables shared between threads are a plausible solution. We consider the following policy to detect data race potentials [18]:

$$\begin{aligned}
\varphi_{dr} &= \mathsf{datarace}(t_1, t_2, x) \rightarrow \exists l.(\mathsf{acqnrel}(t_1, x, l) \wedge \mathsf{acqnrel}(t_2, x, l)), \text{ with} \\
\mathsf{datarace}(t_1, t_2, x) &= \blacklozenge\,(\mathsf{read}\,(t_1, x) \vee \mathsf{write}\,(t_1 x)) \wedge \blacklozenge \mathsf{write}\,(t_2, x)\,, \text{ and} \\
\mathsf{acqnrel}(t, x, l) &= \blacksquare\,((\mathsf{read}\,(t, x) \vee \mathsf{write}\,(t, x)) \rightarrow (\neg \mathsf{rel}\,(t, l)\; \mathcal{S}\; \mathsf{acq}\,(t, l)))
\end{aligned}$$

where the predicates $\mathsf{read}(t, x)$ and $\mathsf{write}(t, x)$ specify read and write operations performed by thread t to shared address x, and $\mathsf{acq}(t, l)$ and $\mathsf{rel}(t, l)$ specify the acquisition and the release of lock l by thread t. Havelund et al. [18] consider a closed formula variant of this policy as their tool, DejaVu, only supports closed formulas. In contrast, WhyMon supports open formulas. We consider the stream prefix:

$$\begin{aligned}
&\langle (0, \{\mathsf{acq}(9, 9)\}), (1, \{\mathsf{read}(9, 3)\}), (2, \{\mathsf{acq}(13, 19)\}), (3, \{\mathsf{acq}(15, 3)\}), \\
&\;(4, \{\mathsf{acq}(18, 15)\}), (5, \{\mathsf{read}(13, 5)\}), (6, \{\mathsf{write}(15, 4)\}), (7, \{\mathsf{write}(15, 3)\}), \ldots\rangle
\end{aligned}$$

At time-point 7, WhyMon outputs a PDT with non-trivial assignments. We focus on the single violation in this PDT, which corresponds to the assignment $(\{9\}, \{15\}, \{3\})$ for (t_1, t_2, x). This violation is shown in Figure 13. The topmost operator of φ_{dr} is an implication, and it is violated because the left side is satisfied (there was a data race), while the right side (the lock requirement) is violated. Specifically, the data race occurred because thread $t_1 = 9$ read address 3 at time-point 1, satisfying the $\blacklozenge_{[0,\infty)}$ subformula in the left conjunct, and thread $t_2 = 15$ wrote to address 3 at the current time-point 7, satisfying the $\blacklozenge_{[0,\infty)}$ subformula in the right conjunct. Moving to the right side of the implication, the violation of the existential indicates that its subformula is violated for every value of $\mathbb{D}$. In particular, the subsets of the domain $\{9\}$ and $\{9\}^{\mathsf{C}}$ are each associated with a different violation. Here, we focus on the violation where $l = 9$. The subformula is a conjunction, and to be violated it suffices that one of the conjuncts is violated. This violation stems from the violation of the right conjunct $\blacksquare_{[0,\infty)}$ (note that $t_2 = 15$ is listed as the variable in the predicate columns). We omit the columns referring to the left conjunct, since all

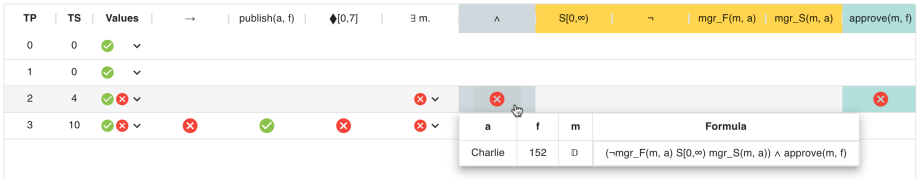

Fig. 11: Visualization of φ''s violation at time-point 3 for $(\{\texttt{Charlie}\}, \{152\})$.

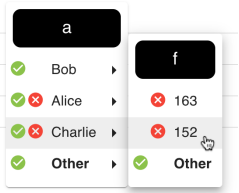

Fig. 12: Assignment selection for φ' at time-point 3.

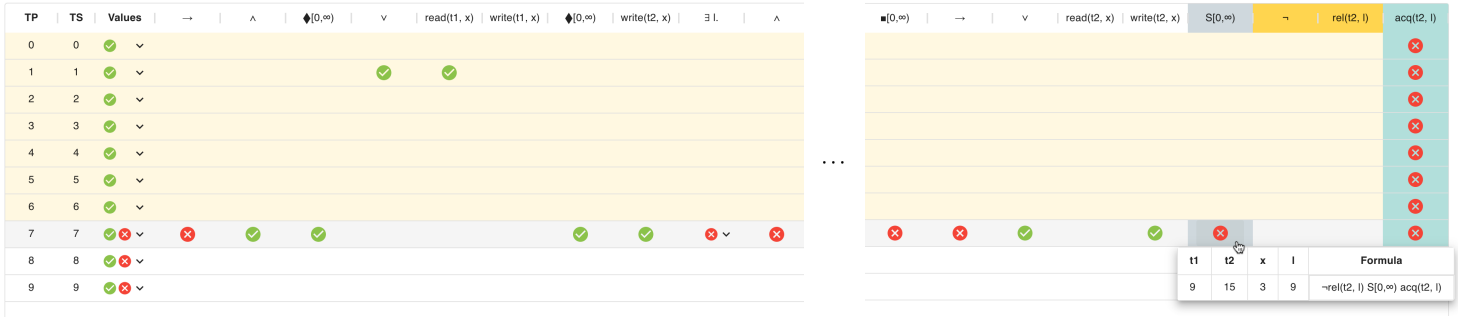

Fig. 13: Visualization of φ_{dr}'s violation at time-point 7 for $(\{9\}, \{15\}, \{3\})$.

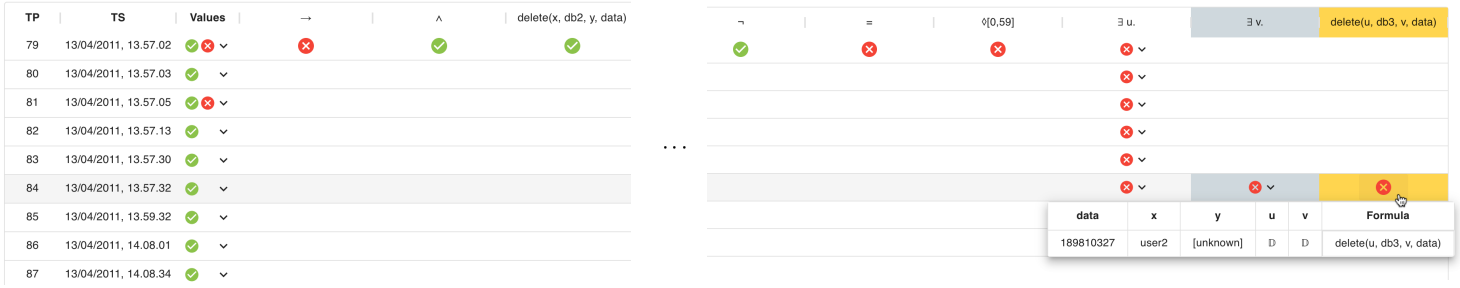

Fig. 14: Visualization of φ_{del}'s violation at time-point 79 for $(\{\texttt{189810327}\}, \{\texttt{user2}\}, \{\texttt{[unknown]}\})$.

entries are empty. Once again, the implication is violated because the left side is satisfied, i.e., thread $t_2 = 15$ wrote to address 3 at time-point 7, satisfying the disjunction, but $\mathcal{S}_{[0,\infty)}$ on the right side is violated, because thread $t_2 = 15$ never acquired the lock $l = 9$.

Data Propagation Nokia's Data-collection Campaign [4] used three databases $\texttt{db}_1$, $\texttt{db}_2$ and $\texttt{db}_3$ in the collection of sensitive information from mobile phones of participants. We focus on the policy φ_{del} [12], which controls the data propagation between databases $\texttt{db}_2$ and $\texttt{db}_3$: if *data* is deleted from $\texttt{db}_2$, then it must be deleted from $\texttt{db}_3$ within 1 minute.

$$\varphi_{del} = \mathsf{delete}(x, \texttt{db}_2, y, data) \wedge data \not\approx \texttt{[unknown]} \rightarrow \Diamond_{[0,60]} \exists u, v.\ \mathsf{delete}(u, \texttt{db}_3, v, data)$$

where $\texttt{db}_2$, $\texttt{db}_3$, and $\texttt{[unknown]}$ are constants and $\mathsf{delete}(db_{user}, db, p_{id}, data)$ specifies the deletion of *data* from participant p_{id} from database *db* using database user db_{user}. We used the REPLAYER tool [22] to convert the stream prefix to WHYMON's format. We executed WHYMON's command line interface with the entire prefix and found two violations. The following experiments were conducted on a computer with an Apple M1 Chip (8 cores) and 16GB of RAM. WHYMON took 17m51s to process the entire prefix. We also executed MONPOLY with a slightly modified yet equivalent policy (due to monitorability restrictions), and its running time amounted to 1m10s. MONPOLY outperforms WHYMON, but we must acknowledge the different outputs both monitors produce. MONPOLY only outputs variable assignments, whereas WHYMON outputs entire PDTs containing all assignments and a justification of the verdict in the form of a proof tree for each. We extract 100 time-points containing both violations and focus on the violation at time-point 79 for the assignment $(\{\texttt{189810327}\}, \{\texttt{user2}\}, \{\texttt{[unknown]}\})$ for (*data*,*x*,*y*), depicted in Figure 14. Time-stamps are converted to actual dates (by enabling the option) and we omit time-points that do not contain relevant events for the violation. Let

$\Gamma_{79} = \{\mathsf{delete}(\texttt{user2}, \texttt{db2}, \texttt{[unknown]}, \texttt{189810327}),$
$\Gamma_{80} = \{\mathsf{delete}(\texttt{triggers}, \texttt{db3}, \texttt{[unknown]}, \texttt{[unknown]})\},$
$\Gamma_{81} = \{\mathsf{delete}(\texttt{user2}, \texttt{db2}, \texttt{[unknown]}, \texttt{189810328})\},$ and $\Gamma_{82} = \Gamma_{83} = \Gamma_{84} = \varnothing$.

The implication is violated because the left side is satisfied (there was a deletion at the current time-point 79), but $\Diamond_{[0,59]}$ is violated. Note that $[0,60)$ was replaced with the equivalent interval $[0,59]$. For each time-point of $[\mathsf{E}^{\mathrm{f}}_{79}([0,59]), \mathsf{L}^{\mathrm{f}}_{79}([0,59])] = \{79,\ldots,84\}$, the subformula is violated. Regardless of the values we assign to u and v (all violations are identical), the subformula $\mathsf{delete}(u, \texttt{db}_3, v, \texttt{189810327})$ is violated.

7 Conclusion

We describe an approach for MFOTL monitoring with verdicts in the form of proof objects for every free variable assignment. Such verdicts are useful for understandability and certification, which increases the monitor's trustworthiness. We implement our approach in the tool WHYMON along with an interactive visualization for these verdicts, which we invite the reader to explore [3]. As future work, we plan to provide support for equality between variables and to improve our monitor's performance by, e.g., stream slicing [29].

Data Availability Statement Our artifact [26] includes WHYMON's source code at the artifact submission time together with instructions on how to set up WHYMON locally, extract our PDT checker, execute our examples, and replicate our case study.

Acknowledgements This research is supported by a Novo Nordisk Fonden start package grant (NNF20OC0063462). We thank David Basin, François Hublet, Srđan Krstić, Matthias Lott, Joshua Schneider for their suggestions on WHYMON's and EXPLANATOR2's user interfaces. We are also grateful to anonymous TACAS 2024 reviewers, who helped us improve the presentation of this paper with their valuable comments.

References

1. The Nokia case study log file (2014), https://sourceforge.net/projects/monpoly/files/ldcc.tar/download
2. WHYMON repository (2023), https://github.com/runtime-monitoring/whymon
3. WHYMON web interface (2023), https://runtime-monitoring.github.io/whymon
4. Aad, I., Niemi, V.: NRC data collection campaign and the privacy by design principles. In: Proceedings of the International Workshop on Sensing for App Phones (PhoneSense) (2010)
5. Ailamazyan, A.K., Gilula, M.M., Stolboushkin, A.P., Schwartz, G.F.: Reduction of a relational model with infinite domains to the case of finite domains. Doklady Akademii Nauk SSSR **286**(2), 308–311 (1986), http://mi.mathnet.ru/dan47310
6. Alur, R., Henzinger, T.A.: Real-time logics: Complexity and expressiveness. Inf. Comput. **104**(1), 35–77 (1993). https://doi.org/10.1006/inco.1993.1025
7. Arfelt, E., Basin, D.A., Debois, S.: Monitoring the GDPR. In: Sako, K., Schneider, S.A., Ryan, P.Y.A. (eds.) ESORICS 2019. LNCS, vol. 11735, pp. 681–699. Springer (2019). https://doi.org/10.1007/978-3-030-29959-0_33
8. Basin, D.A., Bhatt, B.N., Krstić, 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
9. Basin, D.A., 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
10. Basin, D.A., Caronni, G., Ereth, S., Harvan, M., Klaedtke, F., Mantel, H.: Scalable offline monitoring of temporal specifications. Formal Methods Syst. Des. **49**(1-2), 75–108 (2016). https://doi.org/10.1007/s10703-016-0242-y
11. Basin, D.A., Dietiker, D.S., Krstić, S., Pignolet, Y., Raszyk, M., Schneider, J., Ter-Gabrielyan, A.: Monitoring the internet computer. In: Chechik, M., Katoen, J., Leucker, M. (eds.) FM 2023. LNCS, vol. 14000, pp. 383–402. Springer (2023). https://doi.org/10.1007/978-3-031-27481-7_22
12. Basin, D.A., Harvan, M., Klaedtke, F., Zalinescu, E.: Monitoring data usage in distributed systems. IEEE Trans. Software Eng. **39**(10), 1403–1426 (2013). https://doi.org/10.1109/TSE.2013.18
13. Basin, D.A., Klaedtke, F., Müller, S.: Monitoring security policies with metric first-order temporal logic. In: Joshi, J.B.D., Carminati, B. (eds.) SACMAT 2010. pp. 23–34. ACM (2010). https://doi.org/10.1145/1809842.1809849
14. Basin, D.A., 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
15. Basin, D.A., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 19–28. EasyChair (2017). https://doi.org/10.29007/89hs
16. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8), 677–691 (1986). https://doi.org/10.1109/TC.1986.1676819

17. Havelund, K., Peled, D.: BDDs for representing data in runtime verification. In: Deshmukh, J., Nickovic, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 107–128. Springer (2020). https://doi.org/10.1007/978-3-030-60508-7_6
18. Havelund, K., Peled, D., Ulus, D.: First-order temporal logic monitoring with BDDs. Formal Methods Syst. Des. **56**(1), 1–21 (2020). https://doi.org/10.1007/s10703-018-00327-4
19. Hull, R., Su, J.: Domain independence and the relational calculus. Acta Informatica **31**(6), 513–524 (1994). https://doi.org/10.1007/BF01213204
20. 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
21. Koymans, R.: Specifying real-time properties with metric temporal logic. Real Time Syst. **2**(4), 255–299 (1990). https://doi.org/10.1007/BF01995674
22. Krstić, S., Schneider, J.: A benchmark generator for online first-order monitoring. In: Deshmukh, J., Nickovic, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 482–494. Springer (2020). https://doi.org/10.1007/978-3-030-60508-7_27
23. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. Int. J. Softw. Tools Technol. Transf. **4**(2), 224–233 (2003). https://doi.org/10.1007/s100090100062
24. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Schramm, A.: TeSSLa: runtime verification of non-synchronized real-time streams. In: Haddad, H.M., Wainwright, R.L., Chbeir, R. (eds.) SAC 2018. pp. 1925–1933. ACM (2018). https://doi.org/10.1145/3167132.3167338
25. Lima, L., Herasimau, A., Raszyk, M., Traytel, D., Yuan, S.: Explainable online monitoring of metric temporal logic. In: TACAS 2023. LNCS, vol. 13994, pp. 473–491. Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_28
26. Lima, L., Huerta y Munive, J.J., Traytel, D.: Artifact for "Explainable online monitoring of metric first-order temporal logic" (2024). https://doi.org/10.5281/zenodo.10439544
27. Raszyk, M.: Efficient, Expressive, and Verified Temporal Query Evaluation. Ph.D. thesis, ETH Zürich (2022). https://doi.org/10.3929/ethz-b-000553221
28. Raszyk, M., Basin, D.A., Krstić, 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
29. Schneider, J., Basin, D.A., Brix, F., Krstić, S., Traytel, D.: Scalable online first-order monitoring. Int. J. Softw. Tools Technol. Transf. **23**(2), 185–208 (2021). https://doi.org/10.1007/s10009-021-00607-1
30. Schneider, J., Basin, D.A., Krstić, 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
31. Ulus, D.: Online monitoring of metric temporal logic using sequential networks. CoRR **abs/1901.00175** (2019). https://doi.org/10.48550/arxiv.1901.00175
32. 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