Proactive Real-Time First-Order Enforcement

François Hublet[1], Leonardo Lima[2], David Basin[1], Srđan Krstić[1], and Dmitriy Traytel[2]

[1] ETH Zürich, Zurich, Switzerland
{francois.hublet, basin, srdan.krstic}@inf.ethz.ch
[2] University of Copenhagen, Denmark
{leonardo, traytel}@di.ku.dk

Abstract. Modern software systems must comply with increasingly complex regulations in domains ranging from industrial automation to data protection. Runtime enforcement addresses this challenge by empowering systems to not only observe, but also actively control, the behavior of target systems by modifying their actions to ensure policy compliance. We propose a novel approach to the proactive real-time enforcement of policies expressed in metric first-order temporal logic (MFOTL). We introduce a new system model, define an expressive MFOTL fragment that is enforceable in that model, and develop a sound enforcement algorithm for this fragment. We implement this algorithm in a tool called WhyEnf and carry out a case study on enforcing GDPR-related policies. Our tool can enforce all policies from the study in real-time with modest overhead. Our work thus provides the first tool-supported approach that can proactively enforce expressive first-order policies in real time.

Keywords: runtime enforcement · temporal logic · obligations

1 Introduction

As modern software systems become increasingly complex, they are required to comply with a myriad of growingly intricate regulations. The ability to monitor and control such systems is an important, technically challenging task.

Runtime *enforcement* [58] tackles this problem by observing and controlling a target system under scrutiny (SuS), so that its actions, possibly modified, comply with a given policy. Runtime enforcement is performed by a component called *enforcer*, which observes the SuS and influences its behavior as permitted by the system model, e.g., by suppressing or causing SuS actions. Enforcement is thus an inherently *online* problem performed during the SuS's execution. When time constraints are involved, enforcement is called *real-time*. This is a more difficult problem than runtime *monitoring* [8], where the SuS is only observed and policy violations are reported, but not prevented. Applications of runtime enforcement are manifold, ranging from safety protocols in industrial automation to regulatory compliance and it is closely related to the problem of controller synthesis [1, 56].

Policies can be decomposed into provisions and obligations [37]. Compliance with provisions depends on past and present SuS behavior, and it is sufficient for an enforcer to react to the current SuS action. Compliance with obligations,

on the other hand, depends on future SuS behavior, requiring the enforcer to account for this behavior and *proactively act* [11] to prevent violations.

In existing approaches to proactive runtime enforcement [11], policies are typically propositional: they regard every system action as either true or false. In practice, however, actions are often parameterized with data values coming from an infinite domain, like strings or integers, and first-order policies are used to formulate dependencies between such actions' parameters. To the best of our knowledge, no previous work supports proactive enforcement of first-order policies: Hublet et al.'s [39] enforcement is real-time, but not proactive; Aceto et al. [5] similarly support only the *reactive* runtime enforcement of first-order provisions.

In this paper, we propose an approach for proactively enforcing metric first-order temporal logic (MFOTL) [18] policies. Our approach features a realistic system model that supports proactive real-time enforcement *in the nick of time* [11, 12], i.e., the enforcer can act at least once per clock tick. Our model includes causable, suppressable, and only-observable SuS actions. Due to its proactivity, our enforcer supports an expressive MFOTL fragment with past and future operators.

Our enforcer is *sound* (modified SuS behavior complies with a given policy) for an enforceable MFOTL fragment (EMFOTL), and *transparent* (if SuS behavior is already policy-compliant, then it is not modified) for a fragment of EMFOTL. Our enforcer relies on the runtime *monitoring* tool WhyMon [49] as a backend. After reviewing MFOTL and WhyMon (Section 2) we describe our approach and evaluate the associated implementation. Our work makes the following contributions:

- We introduce a new system model for the proactive real-time enforcement of metric first-order policies (Section 3).
- We present an enforceable MFOTL fragment (called EMFOTL) with past and future operators that we characterize using a type system (Section 4).
- We develop an enforcement algorithm for EMFOTL and prove its soundness. We also prove its transparency for a fragment of EMFOTL (Section 5).
- We implement the type system and the algorithm into a new tool, called WhyEnf. We carry out a case study on monitoring core GDPR provisions [7], using WhyEnf to enforce the monitored policies. We find that WhyEnf can seamlessly enforce all monitorable policies from this case study in real time with modest runtime overhead (Section 6).

To our knowledge, WhyEnf (available at [43]) is the first proactive first-order policy enforcer (Section 7). All proofs can be found in our extended report [42].

2 Preliminaries

We introduce traces that model system executions, metric-first order temporal logic (MFOTL), and WhyMon, a monitor for an expressive MFOTL fragment.

Let $x, y, z \in \mathbb{V}$ be variables and $c, d \in \mathbb{D}$ be values from an infinite domain $\mathbb{D}$ of constant symbols, like integers or strings. Terms $t \in \mathbb{V} \cup \mathbb{D}$ are either variables or constants. Finite sequences of terms $t_1, \ldots, t_n$ are written as $\bar{t}$. Let $\mathbb{E}$ denote a finite set of *event names*, and the function $\iota : \mathbb{E} \to \mathbb{N}$ map event names to arities. An *event* is a pair $(e, (d_1, \ldots, d_{\iota(e)})) \in \mathbb{E} \times \mathbb{D}^{\iota(e)}$ of an event name e and $\iota(e)$

arguments. We fix a *signature* $\Sigma = (\mathbb{D}, \mathbb{E}, \iota)$ and define the set $\mathbb{DB}$ of *databases* over Σ as $\mathcal{P}(\{(e, \overline{d}) \mid e \in \mathbb{E},\ \overline{d} \in \mathbb{D}^{\iota(e)}\})$. The subset of all databases with event names in $E \subseteq \mathbb{E}$ is $\mathsf{DB}(E) \coloneqq \{D \in \mathbb{DB} \mid \forall (e, (d_1, \ldots, d_{\iota(e)})) \in D.\ e \in E\}$.

Example 1. Consider a system logging GDPR-relevant events defined with the signature $\Sigma_0 = (\mathbb{N}, \mathbb{E}_0, \iota_0)$, where $\mathbb{E}_0 = \{\mathsf{use}, \mathsf{consent}, \mathsf{delete}, \mathsf{deletion\_request}, \mathsf{legal\_ground}\}$, $\iota_0(\mathsf{use}) = \iota_0(\mathsf{delete}) = \iota_0(\mathsf{deletion\_request}) = 3$, and $\iota_0(\mathsf{consent}) = \iota_0(\mathsf{legal\_ground}) = 2$. The events' denotations are: $\mathsf{use}(c, d, u)$ means 'system uses user u's data d from category c', $\mathsf{delete}(c, d, u)$ means 'user u's data d from category c is deleted', $\mathsf{deletion\_request}(c, d, u)$ means 'user u requests deletion of data d from category c', $\mathsf{consent}(u, c)$ means 'user u provides consent for category c', and $\mathsf{legal\_ground}(u, d)$ means 'legal ground was claimed to process user u's data d'.

A *trace* σ is a sequence $\langle(\tau_i, D_i)\rangle_{0 \le i \le k}, k \in \mathbb{N} \cup \{\infty\}$ of timestamps $\tau_i \in \mathbb{N}$ and finite databases $D_i \in \mathbb{DB}$, where timestamps grow *monotonically* ($\forall i < |\sigma|.\ \tau_i \le \tau_{i+1}$) and *progress* (if $|\sigma| = \infty$, then $\forall \tau.\ \exists i.\ \tau < \tau_i$). An index $0 \le i < |\sigma|$, in a trace σ is called a *time-point*. The empty trace is denoted by ε, the set of all traces by $\mathbb{T}$, and the set of finite (resp. infinite) traces by $\mathbb{T}_f$ (resp. $\mathbb{T}_\omega$). For traces $\sigma \in \mathbb{T}_f$ and $\sigma' \in \mathbb{T}$, $\sigma \cdot \sigma'$ denotes their concatenation. A *property* is a subset $P \subseteq \mathbb{T}_\omega$.

Example 2. Consider two infinite traces of a data management system

$$\sigma_1 = (10, \{\mathsf{consent}(1,1), \mathsf{consent}(1,2)), (50, \{\mathsf{use}(1,3,1), \mathsf{use}(2,1,1)\}), \ldots$$
$$\sigma_2 = (10, \{\mathsf{deletion\_request}(2,1,1)\}), (50, \{\mathsf{use}(1,3,1)\}), \ldots$$

In σ_1, user 1 provides consent for categories 1 and 2 at time-point 0 with timestamp 10; at time-point 1 with timestamp 50, the system uses user 1's data 3 (with category 1) and user 1's data 1 (with category 2). In σ_2, user 1 requests deletion of data 1 with category 2, and then the system uses data 3 with category 1.

MFOTL formulae are defined by the following grammar

$$\varphi ::= \ \top \mid e(\overline{t}) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\ \varphi \mid \bigcirc_I \varphi \mid \bullet_I \varphi \mid \varphi\ \mathsf{U}_I\ \varphi \mid \varphi\ \mathsf{S}_I\ \varphi,$$

where $e \in \mathbb{E}$, $x \in \mathbb{V}$, and $I \in \mathbb{I}$ ranges over non-empty intervals in $\mathbb{N}$. We use the standard abbreviations $\bot \coloneqq \neg\top$, $\varphi \vee \psi \coloneqq \neg(\neg\varphi \wedge \neg\psi)$, $\varphi \to \psi \coloneqq \neg\varphi \vee \psi$, $\varphi \leftrightarrow \psi \coloneqq (\varphi \to \psi) \wedge (\psi \to \varphi)$, $\forall x.\ \varphi \coloneqq \neg(\exists x.\ \neg\varphi)$, $\Diamond_I \varphi \coloneqq \top \mathsf{U}_I \varphi$ (eventually), $\blacklozenge_I \varphi \coloneqq \top\ \mathsf{S}_I\ \varphi$ (once), $\Box_I \varphi \coloneqq \neg \Diamond_I \neg\varphi$ (always), and $\blacksquare_I \varphi \coloneqq \neg \blacklozenge_I \neg\varphi$ (historically). A *polarity* $p \in \{+, -\}$ acts upon a formula φ by $+\varphi \coloneqq \varphi$ and $-\varphi \coloneqq \neg\varphi$. We omit intervals of the form $[0, \infty)$ from the temporal operators' subscript. We write $\varphi[d/x]$ for the formula resulting from substituting the free variable x with the constant d in the formula φ. The notation $\varphi[v]$ generalizes such a unary substitution to applying a full *valuation* $v : \mathbb{V} \to \mathbb{D}$, i.e., a mapping from variables to domain values.

Example 3. Suppose that the time unit is *days*. Consider the formulae

$$\varphi_{\mathsf{law}} \equiv \Box\,(\forall c, d, u.\ \mathsf{use}\,(c, d, u) \to \blacklozenge\,(\mathsf{consent}\,(u, c) \vee \mathsf{legal\_grounds}\,(u, d)))$$
$$\varphi_{\mathsf{del}} \equiv \Box\,\big(\forall c, d, u.\ \mathsf{deletion\_request}\,(c, d, u) \to \Diamond_{[0,30]}\ \mathsf{delete}\,(c, d, u)\big)$$

$v, i \vDash e(\bar{t})$ iff $(e, [\![\bar{t}]\!]_v) \in D_i$ | $v, i \vDash \top$
$v, i \vDash \exists x.\ \varphi$ iff $v[x \mapsto d], i \vDash \varphi$ for some $d \in \mathbb{D}$ | $v, i \vDash \neg\varphi$ iff $v, i \nvDash \varphi$
$v, i \vDash \bigcirc_I \varphi$ iff $v, i+1 \vDash \varphi$ and $\tau_{i+1} - \tau_i \in I$ | $v, i \vDash \varphi \wedge \psi$ iff $v, i \vDash \varphi$ and $v, i \vDash \psi$
$v, i \vDash \bullet_I \varphi$ iff $i > 0$ and $v, i-1 \vDash \varphi$ and $\tau_i - \tau_{i-1} \in I$
$v, i \vDash \varphi\, \mathsf{U}_I\, \psi$ iff $v, j \vDash \psi$ for some $j \geq i$ with $\tau_j - \tau_i \in I$ and $v, k \vDash \varphi$ for all $i \leq k < j$
$v, i \vDash \varphi\, \mathsf{S}_I\, \psi$ iff $v, j \vDash \psi$ for some $j \leq i$ with $\tau_i - \tau_j \in I$ and $v, k \vDash \varphi$ for all $j < k \leq i$

Fig. 1. MFOTL semantics for a fixed, infinite trace σ

The formula φ_{law} formalizes *lawfulness of processing*: 'whenever data d with category c belonging to user u is processed, then either u has consented to her data with category c being used, or the controller has claimed a legal ground to process d.' The formula φ_{del} formalizes the GDPR's *right to erasure*: 'whenever a user u requests the deletion of data d of category c, then d must be deleted within 30 days'.

We write $\mathsf{fv}(\varphi)$ and $\mathsf{cs}(\varphi)$ for the set of free variables and constants of a formula φ, respectively. We define the *active domain* $\mathsf{AD}_i(\varphi)$ of a formula φ at time-point i as $\mathsf{cs}(\varphi) \cup \left(\bigcup_{j \leq i} \{d \mid d \text{ is one of } d_k \text{ in } e(d_1, \ldots, d_{\iota(e)}) \in D_j\}\right)$. The active domain of φ at i contains all constants occurring in φ together with all constants occurring as event arguments in the trace up to time-point i.

Example 4. As $\mathsf{cs}(\varphi_{\mathsf{law}}) = \mathsf{cs}(\varphi_{\mathsf{del}}) = \emptyset$, we have $\mathsf{AD}_0(\varphi_{\mathsf{law}}) = \mathsf{AD}_0(\varphi_{\mathsf{del}}) = \{1, 2\}$ and $\mathsf{AD}_1(\varphi_{\mathsf{law}}) = \mathsf{AD}_1(\varphi_{\mathsf{del}}) = \{1, 2, 3\}$ for σ_1.

MFOTL's semantics (Figure 1) is defined over infinite traces. Given a valuation v, we define the interpretation of terms as $[\![x]\!]_v = v(x)$ (for variables) and $[\![c]\!]_v = c$ (for constants). We lift this operation straightforwardly to lists of terms. A valuation update is denoted as $v[d/x]$. Each sequent $v, i \vDash_\sigma \varphi$ denotes that φ is satisfied at time-point i of trace σ under valuation v. We omit σ whenever it is clear from the context. The *language* of a formula φ is $\mathcal{L}(\varphi) = \{\sigma \in \mathbb{T}_\omega \mid \exists v.\ v, 0 \vDash_\sigma \varphi\}$.

Lima et al. [49] present an algorithm and a tool, called WhyMon, that can monitor an expressive safety fragment of MFOTL both online and offline. This fragment contains *all* formulae with future-bounded until operators. Thus, it strictly extends the fragments supported by other tools like MonPoly [13] and VeriMon [9], which only support formulas in relational algebra normal form [20], and DejaVu [35], which is restricted to past temporal operators.

Abstractly, WhyMon implements a function $\textsc{Sat}(v, \varphi, i) = v, i \vDash \varphi$ that checks if a valuation satisfies the formula φ on a (fixed) trace σ at time-point i. Internally, it manipulates objects representing proofs of φ's subformulae. This technique additionally allows WhyMon to output *explanations* [48] of its verdicts (satisfactions or violations) in the form of proofs that can be checked using a proof checker. We refer to Lima et al.'s work [49] for further details.

3 Proactive, Real-Time, First-Order Enforcement

Our system model (Section 3.1) is inspired by Basin et al.'s model for proactive *propositional* enforcement [11, 12] and Hublet et al.'s model for (non-proactive) *first-order* enforcement [39]. Within this model, we define enforcers (Section 3.2).

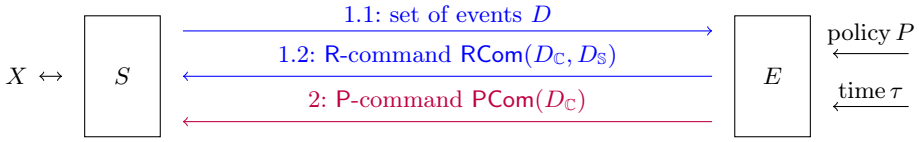

Fig. 2. System model for proactive real-time first-order enforcement

3.1 System model

Figure 2 shows a system S supervised by an enforcer E described using a communication diagram [32]. The system S interacts with an environment X that E cannot control. The enforcer E must ensure that the sequence of actions executed by S complies with a given policy P. To this end, S reports to E sets of events (from $\mathbb{E}$) that capture the system's observable actions. The enforcer E can send commands to S, whereby it instructs S to cause or suppress the actions corresponding to specific events. There are two kinds of such commands, R-commands and P-commands, which will be described below. We assume that the set of events is partitioned into a set of *causable* events $\mathbb{C}$ capturing actions that E can instruct S to cause, a set of *suppressable* events $\mathbb{S}$ capturing actions that E can instruct S to suppress, and a set of *only-observable* events $\mathbb{O} = \mathbb{E} \setminus (\mathbb{S} \cup \mathbb{C})$ capturing actions that can be neither caused nor suppressed.

Example 5. Suppose that the system from Example 1 can be instrumented so that an enforcer can (observe and) prevent data usage and cause data deletion, but can only observe the remaining actions. The corresponding event sets are then $\mathbb{C} = \{\mathsf{delete}\}$, $\mathbb{S} = \{\mathsf{use}\}$, and $\mathbb{O} = \{\mathsf{consent}, \mathsf{legal\_ground}, \mathsf{deletion\_request}\}$.

More specifically, we assume that E interacts with S in three modes: (1) *Before* performing any suppressable actions, S sends the corresponding set of (suppressable) events $D \in \mathbb{DB}$ to E. The enforcer inspects D and *reactively* responds with an R-command $\mathsf{RCom}(D_\mathbb{C}, D_\mathbb{S})$, where $D_\mathbb{C} \in \mathsf{DB}(\mathbb{C})$ is a set of causable events and $D \supseteq D_\mathbb{S} \in \mathsf{DB}(\mathbb{S})$ is a set of suppressable events. S then performs the actions corresponding to the events in $(D \setminus D_\mathbb{S}) \cup D_\mathbb{C}$, i.e., all actions corresponding to events in $D_\mathbb{C}$ (resp. $D_\mathbb{S}$) are caused (resp. suppressed). (2) *After* performing actions that are *not suppressable*, S sends the corresponding set of events $D \in \mathbb{DB}$ to E. The enforcer inspects D and responds with an R-command $\mathsf{RCom}(D_\mathbb{C}, \emptyset)$. As no suppressable actions are to be performed and the events are sent after the actions, the enforcer can only instruct S to cause actions, but not to suppress them. (3) *Before* any clock tick ('in the nick of time' [12]), E can *proactively* send a P-command $\mathsf{PCom}(D_\mathbb{C})$ with $D_\mathbb{C} \in \mathsf{DB}(\mathbb{C})$ to S. The system S then performs the actions corresponding to the events in $D_\mathbb{C}$. Note that sending a P-command before a tick is always possible, but the enforcer may instead choose not to send any command.

These modes of interaction cover different enforcement scenarios. In mode (1), E *reacts* to suppressable events by possibly suppressing or causing events. E.g., the formula φ_{law} from Example 3 can be enforced by suppressing data usage (the use events) if no appropriate event has previously occurred. In mode (2), E reacts to only-observable events (e.g., the $\mathsf{consent}$ events) by possibly causing events corresponding to corrective actions after the executed action. Finally,

mode (3) enforces policies by causing events at times when the SuS does not, on its own, send any observable events. This is the case, e.g., when enforcing φ_{del} on σ_2: data 1 with category 2 must be deleted between timestamps 10 and 40.

Discussion. Assume that the enforcer E can ensure that the sequence of actions it observes complies with P. When does this guarantee that the system actually complies with P? Basin et al. [12] state two conditions for achieving soundness: (a) the system and enforcer must be synchronized and (b) the enforcer must be fast enough to keep up with the real-time system behavior. These conditions also apply in our model. Condition (a) ensures that the order of events observed by E reflect the order of S's actions. Condition (b) ensures that the timestamps of events reflect the time at which the corresponding actions are performed by S. The interval t between two clock ticks must satisfy the *real-time condition* $t > \delta_S + 2\delta_{S \leftrightarrow E} + \delta_E$, where δ_S is the worst-case time needed by S to create events before performing observable actions and process the enforcer's reactions, $\delta_{S \leftrightarrow E}$ is the worst-case communication time between S and E, and δ_E is the worst-case latency of the enforcer. Threats to the model's validity may thus stem from high communication time, or poor SuS or enforcer performance.

3.2 Enforcers

An enforcer reads the consecutive prefixes of an SuS's trace and returns commands:

Definition 1. *A* command *is any element of the form* $\mathsf{RCom}(D_{\mathbb{C}}, D_{\mathbb{S}})$ *('*R*-command'),* $\mathsf{PCom}(D_{\mathbb{C}})$ *('*P*-command'), or* NoCom *('no command'), where* $D_{\mathbb{C}} \in \mathsf{DB}(\mathbb{C})$ *and* $D_{\mathbb{S}} \in \mathsf{DB}(\mathbb{S})$*. The set of commands is denoted by* $\mathcal{C}$*.*

Definition 2. *An* enforcer $\mathcal{E}$ *is a triple* $(\mathcal{S}, s_0, \mu)$*, where* $\mathcal{S}$ *is a set of states,* $s_0 \in \mathcal{S}$ *is an initial state, and* $\mu : \mathbb{T}_f \times \mathcal{S} \times (\mathbb{N} \cup \{\bot\}) \to \mathcal{C} \times \mathcal{S}$ *is a computable* update *function such that the following two conditions hold:*

$$\forall \sigma, \tau, D, s.\ \exists D_{\mathbb{C}}, D_{\mathbb{S}}, s'.\ \mu(\sigma \cdot (\tau, D), s, \bot) = (\mathsf{RCom}(D_{\mathbb{C}}, D_{\mathbb{S}}), s') \wedge D_{\mathbb{S}} \subseteq D$$
$$\forall \sigma, s, \tau \in \mathbb{N}.\ \exists D_{\mathbb{C}}, s'.\ \mu(\sigma, s, \tau) \in \{(\mathsf{PCom}(D_{\mathbb{C}}), s'), (\mathsf{NoCom}, s')\}.$$

If μ's third argument is $\bot$, then μ returns an R-command. The set of events to be suppressed contained in this command is a subset of the last set of events reported by the SuS. On the other hand, if μ's third argument is an integer timestamp, then μ returns either a P-command for the corresponding timestamp, or no command. Any enforcer induces the following trace transduction:

Definition 3. *For any* $\sigma \in \mathbb{T}$ *and enforcer* $\mathcal{E} = (\mathcal{S}, s_0, \mu)$*, the* enforced trace $\mathcal{E}(\sigma)$ *is defined co-recursively in Algorithm 1, where* $\mathsf{fts}(\sigma)$ *is the first timestamp in* σ*.*

Algorithm 1 formalizes the interaction described in Section 3.1: the enforcer is called once at every time-point in the input trace σ to generate an R-command (lines 6–7), and once before each clock tick to (possibly) generate a P-command (lines 3–5). The generated commands are executed sequentially to produce the enforced trace $\mathcal{E}(\sigma)$, which thus reflects the actions performed by the SuS when composed with the enforcer as in Section 3.1.

1: $\mathsf{run}(s, \sigma, \sigma', \tau) = \textbf{case } \sigma' \textbf{ of}$
2: $\mid \varepsilon \Rightarrow \varepsilon$
3: $\mid (\tau', D) \cdot \sigma''$ **when** $\tau' > \tau \Rightarrow$ **let** $(o, s') = \mu(\sigma, s, \tau)$ **in**
4: $\quad$ **case** o **of** $\mid \mathsf{PCom}(D_C) \Rightarrow (\tau, D_C) \cdot \mathsf{run}(s', \sigma \cdot (\tau, D_C), \sigma', \tau + 1)$
5: $\quad\quad\quad\quad \mid \mathsf{NoCom} \Rightarrow \mathsf{run}(s', \sigma, \sigma', \tau + 1)$
6: $\mid (\tau', D) \cdot \sigma''$ **when** $\tau' = \tau \Rightarrow$ **let** $(o, s') = \mu(\sigma \cdot (\tau', D), s, \bot); D' = (D \setminus D_S) \cup D_C)$ **in**
7: $\quad$ **case** o **of** $\mid \mathsf{RCom}(D_C, D_S) \Rightarrow (\tau', D') \cdot \mathsf{run}(s', \sigma \cdot (\tau', D'), \sigma'', \tau + 1)$
8: $\mathcal{E}(\sigma) = \mathsf{run}(s_0, \varepsilon, \sigma, \textbf{if } \sigma = \varepsilon \textbf{ then } 0 \textbf{ else } \mathsf{fts}(\sigma))$

Algorithm 1: Enforced trace

To be considered *correct* with respect to a given property P, enforcers are typically required to fulfill two properties: *soundness* and *transparency* [47]. Soundness states that any trace modified by the enforcer must be compliant with P, while transparency states that the enforcer does not alter a trace that already complies with the policy. A transparent enforcer modifies the system's behavior *only when necessary*. The following definition formalizes these notions.

Definition 4. *An enforcer $\mathcal{E}$ is* sound *with respect to a property P iff for any $\sigma \in \mathbb{T}_\omega$, we have $\mathcal{E}(\sigma) \in P$. An enforcer $\mathcal{E} = (\mathcal{S}, s_0, \mu)$ is* transparent *with respect to a property P iff for all $\sigma \in P$, $\mathcal{E}(\sigma) = \sigma$. A property P (resp. a formula φ) is* enforceable *iff there exists a sound enforcer with respect to P (resp. $\mathcal{L}(\varphi)$).*

4 Enforceable MFOTL Formulae

In this section, we present EMFOTL, an expressive and enforceable fragment of MFOTL. An enforcer for EMFOTL formulae will be presented in Section 5.

EMFOTL is defined using the typing rules in Figure 3. These consist of sequents of the form $\Gamma \vdash \varphi : \alpha$, reading '$\varphi$ types to α under Γ'. Here, context $\Gamma : \mathbb{E} \to \{\mathbb{C}, \mathbb{S}\}$ is a mapping from event names to either of the symbols $\mathbb{C}$ or $\mathbb{S}$, φ is an MFOTL formula, and α is a type in $\{\mathbb{C}, \mathbb{S}\}$. The type names $\mathbb{C}$ and $\mathbb{S}$ overload the names of the sets of suppressable and causable events in a natural way: any event $e_c(\bar{t})$ with $e_c \in \mathbb{C}$ (resp. $e_s \in \mathbb{S}$) has type $\mathbb{C}$ (resp. $\mathbb{S}$) under the context $\{e_c \mapsto \mathbb{C}\}$ (resp. $\{e_s \mapsto \mathbb{S}\}$). EMFOTL is defined as the set of all φ for which $\exists \Gamma.\, \Gamma \vdash \varphi : \mathbb{C}$. Intuitively, a formula types to $\mathbb{C}$ under Γ ('φ is causable under Γ') if it can be enforced by causing events $e_c(\bar{t})$ such that $\Gamma(e_c) = \mathbb{C}$ and suppressing events $e_s(\bar{t})$ such that $\Gamma(e_s) = \mathbb{S}$. It types to $\mathbb{S}$ under Γ ('φ is suppressable under Γ') if $\neg\varphi$ can be enforced under the same conditions on Γ.

We now review the typing rules presented in Figure 3. Our approach for enforcing temporal operators is illustrated in Figure 4.

Constants and predicates (Rules $\top^{\mathbb{C}}$, $\bot^{\mathbb{S}}$, $\mathbb{E}^{\mathbb{C}}$, $\mathbb{E}^{\mathbb{S}}$). The constant $\top$ (resp. $\bot$) is causable (resp. suppressable). Event $e(t_1, \ldots, t_k)$ is causable (resp. suppressable) under Γ if $e \in \mathbb{C}$ and $\Gamma(e) = \mathbb{C}$ (resp. $e \in \mathbb{S}$ and $\Gamma(e) = \mathbb{S}$).

Negation (Rules $\neg^{\mathbb{C}}$, $\neg^{\mathbb{S}}$). Negation exchanges $\mathbb{C}$ and $\mathbb{S}$: a formula is causable iff its negation is suppressable; it is suppressable iff its negation is causable.

Conjunction (Rules $\wedge^{\mathbb{C}}$, $\wedge^{\mathbb{S}\mathrm{L}}$, $\wedge^{\mathbb{S}\mathrm{R}}$). A conjunction is causable if both of its conjuncts are causable; it is suppressable if either of its conjuncts is suppressable.

$$\frac{}{\vdash e(\dots,x,\dots) : \mathsf{PG}(x)^+}\ \mathbb{E}^+_{\mathrm{PG}} \qquad \frac{\vdash \varphi : \mathsf{PG}(x)^{\neg p}}{\vdash \neg\varphi : \mathsf{PG}(x)^p}\ \neg_{\mathrm{PG}} \qquad \frac{x \neq z \quad \vdash \varphi : \mathsf{PG}(z)^p}{\vdash \exists x.\ \varphi : \mathsf{PG}(z)^p}\ \exists_{\mathrm{PG}}$$

$$\frac{\vdash \varphi : \mathsf{PG}(x)^+}{\vdash \varphi \wedge \psi : \mathsf{PG}(x)^+}\ \wedge^{\mathrm{L}+}_{\mathrm{PG}} \qquad \frac{\vdash \psi : \mathsf{PG}(x)^+}{\vdash \varphi \wedge \psi : \mathsf{PG}(x)^+}\ \wedge^{\mathrm{R}+}_{\mathrm{PG}} \qquad \frac{\vdash \varphi : \mathsf{PG}(x)^- \quad \vdash \psi : \mathsf{PG}(x)^-}{\vdash \varphi \wedge \psi : \mathsf{PG}(x)^-}\ \wedge^{-}_{\mathrm{PG}}$$

$$\frac{0 \notin I \quad \vdash \varphi : \mathsf{PG}(x)^+}{\vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{PG}(x)^+}\ \mathsf{S}^{\mathrm{L}+}_{\mathrm{PG}} \qquad \frac{\vdash \psi : \mathsf{PG}(x)^+}{\vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{PG}(x)^+}\ \mathsf{S}^{\mathrm{R}+}_{\mathrm{PG}} \qquad \frac{0 \in I \quad \vdash \psi : \mathsf{PG}(x)^-}{\vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{PG}(x)^-}\ \mathsf{S}^{-}_{\mathrm{PG}}$$

$$\frac{0 \notin I \quad \vdash \varphi : \mathsf{PG}(x)^+}{\vdash \varphi\, \mathsf{U}_I\, \psi : \mathsf{PG}(x)^+}\ \mathsf{U}^{\mathrm{L}+}_{\mathrm{PG}} \qquad \frac{\vdash \varphi : \mathsf{PG}(x)^+ \quad \vdash \psi : \mathsf{PG}(x)^+}{\vdash \varphi\, \mathsf{U}_I\, \psi : \mathsf{PG}(x)^+}\ \mathsf{U}^{\mathrm{LR}+}_{\mathrm{PG}}$$

Past-guardedness

$$\frac{0 \in I \quad \vdash \psi : \mathsf{PG}(x)^-}{\vdash \varphi\, \mathsf{U}_I\, \psi : \mathsf{PG}(x)^-}\ \mathsf{U}^{-}_{\mathrm{PG}} \qquad \frac{\vdash \varphi : \mathsf{PG}(x)^+}{\vdash \bullet_I\, \varphi : \mathsf{PG}(x)^+}\ \bullet^{+}_{\mathrm{PG}}$$

$$\frac{}{\Gamma \vdash \top : \mathbb{C}}\ \top^{\mathbb{C}} \qquad \frac{}{\Gamma \vdash \bot : \mathbb{S}}\ \bot^{\mathbb{S}} \qquad \frac{e \in \mathbb{C} \quad \Gamma(e) = \mathbb{C}}{\Gamma \vdash e(t_1,\dots,t_k) : \mathbb{C}}\ \mathbb{E}^{\mathbb{C}} \qquad \frac{e \in \mathbb{S} \quad \Gamma(e) = \mathbb{S}}{\Gamma \vdash e(t_1,\dots,t_k) : \mathbb{S}}\ \mathbb{E}^{\mathbb{S}}$$

$$\frac{\Gamma \vdash \varphi : \mathbb{S}}{\Gamma \vdash \neg\varphi : \mathbb{C}}\ \neg^{\mathbb{C}} \qquad \frac{\Gamma \vdash \varphi : \mathbb{C}}{\Gamma \vdash \neg\varphi : \mathbb{S}}\ \neg^{\mathbb{S}} \qquad \frac{\Gamma \vdash \varphi : \mathbb{C}}{\Gamma \vdash \exists x.\ \varphi : \mathbb{C}}\ \exists^{\mathbb{C}} \qquad \frac{\Gamma \vdash \varphi : \mathbb{S} \quad \vdash \varphi : \mathsf{PG}(x)^+}{\Gamma \vdash \exists x.\ \varphi : \mathbb{S}}\ \exists^{\mathbb{S}}$$

$$\frac{\Gamma \vdash \varphi : \mathbb{C} \quad \Gamma \vdash \psi : \mathbb{C}}{\Gamma \vdash \varphi \wedge \psi : \mathbb{C}}\ \wedge^{\mathbb{C}} \qquad \frac{\Gamma \vdash \varphi : \mathbb{S}}{\Gamma \vdash \varphi \wedge \psi : \mathbb{S}}\ \wedge^{\mathbb{S}\mathrm{L}} \qquad \frac{\Gamma \vdash \psi : \mathbb{S}}{\Gamma \vdash \varphi \wedge \psi : \mathbb{S}}\ \wedge^{\mathbb{S}\mathrm{R}}$$

$$\frac{0 \in I \quad \Gamma \vdash \psi : \mathbb{C}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathbb{C}}\ \mathsf{S}^{\mathbb{C}} \qquad \frac{0 \notin I \quad \Gamma \vdash \varphi : \mathbb{S}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathbb{S}}\ \mathsf{S}^{\mathbb{S}\mathrm{L}} \qquad \frac{0 \in I \quad \Gamma \vdash \varphi : \mathbb{S} \quad \Gamma \vdash \psi : \mathbb{S}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathbb{S}}\ \mathsf{S}^{\mathbb{S}\mathrm{LR}}$$

$$\frac{\Gamma \vdash \psi : \mathbb{S}}{\Gamma \vdash \varphi\, \mathsf{U}_I\, \psi : \mathbb{S}}\ \mathsf{U}^{\mathbb{S}} \qquad \frac{b \neq \infty \quad \Gamma \vdash \psi : \mathbb{C}}{\Gamma \vdash \varphi\, \mathsf{U}_{[0,b]}\, \psi : \mathbb{C}}\ \mathsf{U}^{\mathbb{C}\mathrm{R}} \qquad \frac{b \neq \infty \quad \Gamma \vdash \varphi : \mathbb{C} \quad \Gamma \vdash \psi : \mathbb{C}}{\Gamma \vdash \varphi\, \mathsf{U}_{[a,b]}\, \psi : \mathbb{C}}\ \mathsf{U}^{\mathbb{C}\mathrm{LR}}$$

Typing of formulae as causable/suppressable

$$\frac{\Gamma \vdash \varphi : \mathbb{C} \quad b > 0}{\Gamma \vdash \bigcirc_{[0,b)}\, \varphi : \mathbb{C}}\ \bigcirc^{\mathbb{C}} \qquad \frac{\Gamma \vdash \varphi : \mathbb{S}}{\Gamma \vdash \bigcirc_I\, \varphi : \mathbb{S}}\ \bigcirc^{\mathbb{S}}$$

Fig. 3. Typing rules for EMFOTL

Quantifiers (Rules $\exists^{\mathbb{C}}$, $\exists^{\mathbb{S}}$). The formula $\varphi' = \exists x.\ \varphi$ is causable if φ is causable: it is enough to set x to some value v and cause $\varphi[x/v]$ to cause φ'.In contrast, to suppress φ' at i, we must ensure that *no value of* $v \in \mathbb{D}$ can satisfy φ. If φ depends on the future, then values of v satisfying φ' may only be discovered *strictly after* i. Then, it may not be possible to decide which $\varphi[x/v]$ to suppress at i. Our fragment rules this case out by requiring that x be *past-guarded* in φ, i.e., that any value of x that satisfies φ is a constant or present in the trace up until i. Formally:

Definition 5 (Past-guardedness). *A variable x is past-guarded in φ iff* $\forall v, i.\ v, i \vDash \varphi \wedge x \in \operatorname{dom} v \implies v(x) \in \mathsf{AD}_i(\varphi)$.

Past-guardedness can be soundly overapproximated using the type system in the upper half of Figure 3. The PG typing rules define sequents of the form $\vdash \varphi : \mathsf{PG}(x)^p$, where $p \in \{+,-\}$. In our extended report [42], we prove

Lemma 1. *For $p \in \{+,-\}$, if $\vdash \varphi : \mathsf{PG}(x)^p$, then x is past-guarded in $p\varphi$.*

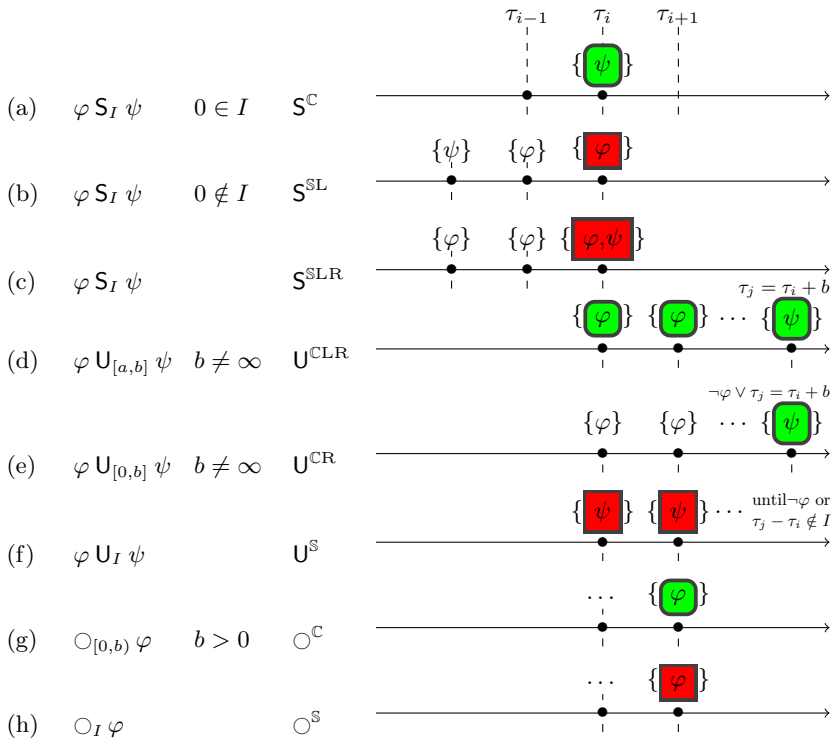

Fig. 4. Enforcement for temporal operators: φ = cause φ and φ = suppress φ

Since (Rules $\mathsf{S}^{\mathbb{C}}$, $\mathsf{S}^{\mathbb{S}\mathrm{L}}$, $\mathsf{S}^{\mathbb{S}\mathrm{LR}}$). As enforcers cannot affect the past, causation of $\varphi' = \varphi\,\mathsf{S}_I\,\psi$ is only possible when $0 \in I$ and ψ is enforceable. In this case, φ' is caused by causing ψ in the present (Figure 4, a). To suppress φ', we consider two scenarios. If $0 \notin I$, then to suppress φ', it suffices to suppress φ in the present (Figure 4, b). If $0 \in I$, both φ and ψ may need to be suppressed (Figure 4, c).

Until (Rules $\mathsf{U}^{\mathbb{S}}$, $\mathsf{U}^{\mathbb{C}\mathrm{R}}$, $\mathsf{U}^{\mathbb{C}\mathrm{LR}}$). The formula $\varphi' = \varphi\,\mathsf{U}_I\,\psi$ is causable if both φ and ψ are causable: one can cause φ until the interval I has elapsed, and then cause ψ 'in the nick of time' (Figure 4, d). This requires a finite upper bound for I; otherwise, the enforcer may wait indefinitely to cause ψ, producing a non-compliant trace. (For $I = [a, \infty)$, we could enforce φ' non-transparently by causing ψ after an arbitrary, finite interval $[a, b)$. In this case, the user could have as well specified $\varphi\,\mathsf{U}_{[a,b)}\,\psi$. Hence, our type system requires a finite I.) Alternatively, if $0 \in I$, then φ' can be caused when ψ is causable, with the enforcer causing ψ as soon as φ ceases to hold or the interval has elapsed (Figure 4, e). In contrast, φ' can be suppressed whenever ψ is suppressable (Figure 4, f). This also applies when I is unbounded: if necessary, the formula ψ can be suppressed indefinitely. Enforcement can thus be performed for formulae that are generally not supported by existing monitors [18]. Namely, monitors exclude non-future-bounded formulae, for which compliance cannot be guaranteed by observing a finite prefix of the trace and hence verdicts cannot be given in finite time. However, an enforcer can ensure compliance at every time-point.

Previous The formula $\varphi' = \bullet_I \varphi$ can neither be caused nor suppressed without editing databases of events that happened strictly in the past. This goes beyond the enforcer's capabilities in our model.

Next (Rules $\bigcirc^{\mathbb{C}}$, $\bigcirc^{\mathbb{S}}$). If φ is suppressable, the formula $\varphi' = \bigcirc_I \varphi$ is also suppressable: φ' is suppressed by suppressing φ at the next time-point (Figure 4, g). In contrast, causing φ' is not possible for arbitrary I. If $I = [a, b)$ with $a > 0$, then, to cause φ' at i, one must ensure $\tau_{i+1} \geq \tau_i + a$. But the next time-point in the input trace might be $\tau_{i+1} < \tau_i + a$ (e.g., $\tau_{i+1} = \tau_i$), and this timestamp cannot be suppressed. If $I = [0, 0]$, then enforcing $\varphi'' = \Box \varphi'$ is not possible, since no trace satisfies φ'' (a trace must satisfy progress): one cannot both support $I = [0, 0]$ in rule $\bigcirc^{\mathbb{C}}$ and use the previous definition of $\mathsf{U}^{\mathbb{S}}$. Therefore, our fragment only supports causation of $\bigcirc_I \varphi$ for intervals I of the form $[0, b)$, $b > 0$ (Figure 4, h). Our use of the context Γ is inspired by Hublet et al. [39]. By ensuring that all events with the same name are only caused or only suppressed, we exclude non-enforceable formulae such as $e \wedge \neg e$, where e is both causable and suppressable.

Example 6. We show that φ_{del} presented in Example 3 is in EMFOTL. We work with the "desugared" variant of φ_{del} (instead of using abbreviations like $\Diamond$):

$$\varphi'_{\mathsf{del}} \equiv \neg \left(\top \,\mathsf{U}\, \left(\exists c, d, u.\, \mathsf{deletion\_request}\,(c, d, u) \wedge \left(\neg \left(\top \,\mathsf{U}_{[0,30]}\, \mathsf{delete}\,(c, d, u) \right) \right) \right) \right)$$

Furthermore, we shorten $\varphi'_{\mathsf{del}} \equiv \neg(\top \,\mathsf{U}\, \varphi_{\exists_1})$, where:

$$\varphi_{\exists_1} \equiv \exists c.\, \varphi_{\exists_2} \qquad \varphi_{\exists_2} \equiv \exists d.\, \varphi_{\exists_3} \qquad \varphi_{\exists_3} \equiv \exists u.\, \varphi_{\wedge} \qquad \varphi_{\wedge} \equiv \varphi_{\wedge_1} \wedge \varphi_{\wedge_2}$$
$$\varphi_{\wedge_1} \equiv \mathsf{deletion\_request}\,(c, d, u) \qquad \varphi_{\wedge_2} \equiv \neg \varphi_{\mathsf{U}} \qquad \varphi_{\mathsf{U}} \equiv \top \,\mathsf{U}_{[0,30]}\, \mathsf{delete}\,(c, d, u)$$

Lastly, we use the typing rules presented in Figure 3 to show that φ'_{del} types to $\mathbb{C}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{delete} \in \mathbb{C}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \mathsf{delete}(c, d, u) : \mathbb{C}}\,\mathbb{E}^{\mathbb{C}}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \varphi_{\mathsf{U}} \equiv \top \,\mathsf{U}_{[0,30]}\, \mathsf{delete}\,(c, d, u) : \mathbb{C}}\,\mathsf{U}^{\mathbb{C}\mathrm{R}}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \varphi_{\wedge_2} \equiv \neg\varphi_{\mathsf{U}} : \mathbb{S}}\,\neg^{\mathbb{S}}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \varphi_{\wedge} \equiv \varphi_{\wedge_1} \wedge \varphi_{\wedge_2} : \mathbb{S}}\,\wedge^{\mathbb{S}\mathrm{R}}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \varphi_{\exists_3} \equiv \exists u.\, \varphi_{\wedge} : \mathbb{S}}\,\exists^{\mathbb{S}}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \varphi_{\exists_2} \equiv \exists d.\, \varphi_{\exists_3} : \mathbb{S}}\,\exists^{\mathbb{S}}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \varphi_{\exists_1} \equiv \exists c.\, \varphi_{\exists_2} : \mathbb{S}}\,\exists^{\mathbb{S}}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \top \,\mathsf{U}\, \varphi_{\exists_1} : \mathbb{S}}\,\mathsf{U}^{\mathbb{S}}}{\{\mathsf{delete} \mapsto \mathbb{C}\} \vdash \varphi'_{\mathsf{del}} \equiv \neg(\top \,\mathsf{U}\, \varphi_{\exists_1}) : \mathbb{C}}\,\neg^{\mathbb{C}}$$

where $P_{1,2,3}$ respectively stand for:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \varphi_{\wedge_1} : \mathsf{PG}(c)^+}\,\mathbb{E}^+_{\mathrm{PG}}}{\vdash \varphi_{\wedge} : \mathsf{PG}(c)^+}\,\wedge^{\mathrm{L}+}_{\mathrm{PG}} \quad u \neq c}{\vdash \varphi_{\exists_3} : \mathsf{PG}(c)^+}\,\exists_{\mathrm{PG}} \quad d \neq c}{\vdash \varphi_{\exists_2} : \mathsf{PG}(c)^+}\,\exists_{\mathrm{PG}} \qquad \dfrac{\dfrac{\dfrac{}{\vdash \varphi_{\wedge_1} : \mathsf{PG}(d)^+}\,\mathbb{E}^+_{\mathrm{PG}}}{\vdash \varphi_{\wedge} : \mathsf{PG}(d)^+}\,\wedge^{\mathrm{L}+}_{\mathrm{PG}} \quad u \neq d}{\vdash \varphi_{\exists_3} : \mathsf{PG}(d)^+}\,\exists_{\mathrm{PG}}$$

$$\dfrac{\dfrac{}{\vdash \varphi_{\wedge_1} : \mathsf{PG}(u)^+}\,\mathbb{E}^+_{\mathrm{PG}}}{\vdash \varphi_{\wedge} : \mathsf{PG}(u)^+}\,\wedge^{\mathrm{L}+}_{\mathrm{PG}}$$

The formula φ_{law} is also in EMFOTL (see our extended report [42]).

$$\mathsf{fo}_{\mathsf{init},\varphi_1} = \lambda\_.\, \varphi_1$$
$$\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1} = \lambda\tau'.\, \text{if } \tau' - \tau \leq \sup I \text{ then } (\neg \mathtt{TP})\ \mathsf{U}_{I-(\tau'-\tau)}\ (\mathtt{TP} \wedge \varphi_1) \text{ else } \bot$$
$$\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2} = \lambda\tau'.\, \text{if } \tau' - \tau \leq \sup I \text{ then } (\mathtt{TP} \rightarrow \varphi_1)\ \mathsf{U}_{I-(\tau'-\tau)}\ (\mathtt{TP} \wedge \varphi_2) \text{ else } \bot$$

Fig. 5. Mappings in the first component of future obligations

5 Enforcing EMFOTL

We now describe our enforcement algorithm. First, we present the enforcer's state, which consists of a set of *obligations* (Section 5.1). We then explain how Lima et al.'s monitoring algorithm [49] can be extended to check the satisfaction of a formula φ *under assumptions* about the future (Section 5.2). Finally, we present our algorithm (Section 5.3) and prove its soundness and transparency (Section 5.4).

5.1 Obligations

Our algorithm manipulates sets of *obligations* that encode the formulae to be caused or suppressed in the future. There are two types of obligations, *present* and *future obligations*. A *present obligation* is a triple (φ, v, p) of an MFOTL formula φ, a valuation v, and a polarity $p \in \{+, -\}$ such that $p\varphi \in$ EMFOTL. After reading a new time-point, our enforcer's state will contain a finite set of such present obligations. Some of these obligations will be immediately discharged via causation or suppression. Others will be processed to generate simpler present obligations and new *future obligations* that will then be propagated to the next time-point. Future obligations are triples (ξ, v, p) where $\xi : \mathbb{N} \rightarrow$ MFOTL maps timestamps to EMFOTL formulae and v and p are as before. The set of future obligations is denoted by FO. The mapping ξ is evaluated with the next timestamp to generate present obligations at the next time-point in the trace.

In some cases (e.g., φ_{del}), the enforcer must insert a time-point. In other cases (e.g., φ_{law}), the enforcer can modify the events at existing time-points. To insert a time-point *only when necessary*, we use a special, causable `TP` event encoding the existence of a time-point. When processing a time-point already present in the trace (l. 6 in Algorithm 1), the enforcer receives the additional present obligation $(\mathtt{TP}, \emptyset, +)$, as the time-point cannot be suppressed. When computing P-commands (l. 3 in Algorithm 1), this obligation is *not* given to the enforcer, but `TP` may be generated from other obligations, in which case a time-point is inserted.

Figure 5 shows the mappings used in the first component of future obligations. There are three types of mappings, corresponding to the obligations passed to the enforcer in the initial state and those generated from unrolling $\bigcirc$ and U.

5.2 Checking satisfaction of MFOTL formulae under assumptions

Our enforcer uses WhyMon's monitoring algorithm to check the satisfaction of formulae. Unlike Lima et al. [49], we must however compute satisfactions under assumptions encoding future obligations. To guarantee, e.g., that causing φ in the present and satisfying $fo = (\lambda\tau'.\, \top\ \mathsf{U}\ (\mathtt{TP} \wedge \neg\varphi), \emptyset, -)$ guarantees $\Box\,\varphi$, one must be able to check that after causing φ, $\Box\,\varphi$ is satisfied at i *assuming* that *fo*

$$\frac{(\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1}, v, +) \in X}{v, i, X \vdash^+ \bigcirc_I \varphi_1}\ \bigcirc^+_{\mathsf{assm}} \qquad \frac{v, i, X \vdash^+ \varphi_1 \quad (\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}, v, +) \in X}{v, i, X \vdash^+ \varphi_1\, \mathsf{U}_I\, \varphi_2}\ \mathsf{U}^+_{\mathsf{assm}}$$

$$\frac{(\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1}, v, -) \in X}{v, i, X \vdash^- \bigcirc_I \varphi_1}\ \bigcirc^-_{\mathsf{assm}} \qquad \frac{0 \in I \implies v, i, X \vdash^- \varphi_2 \quad (\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}, v, -) \in X}{v, i, X \vdash^- \varphi_1\, \mathsf{U}_I\, \varphi_2}\ \mathsf{U}^-_{\mathsf{assm}}$$

Fig. 6. Additional proof rules

is satisfied at $i+1$. Since the enforcer will suppress all time-points not containing $\mathtt{TP}$, future time-points can be assumed to all contain $\mathtt{TP}$.

Let $\{\mathbb{C}\}_+ := \mathbb{C}$, $\{\mathbb{C}\}_- := \mathbb{S}$, and $\overline{\sigma}^{\mathtt{TP}} = \langle(\tau_i, D_i \cup \{\mathtt{TP}\})\rangle_{i\in\mathbb{N}}$ for the trace $\sigma = \langle(\tau_i, D_i)\rangle_{i\in\mathbb{N}}$. Consider $\varphi \in$ EMFOTL, and obtain Γ such that $\Gamma \vdash \varphi : \mathbb{C}$. Our satisfiability checker under assumptions is a function

$$\textsc{Sat} : (\mathbb{V} \to \mathbb{D}) \times \text{MFOTL} \times \mathbb{T}_f \times \mathcal{P}(\mathsf{FO}) \to \{\top, \bot\}$$

. The implementation of the checker must ensure that, for any $p \in \{+,-\}$, φ such that $\Gamma \vdash \varphi : \{\mathbb{C}\}_p$, and $X \subseteq \mathsf{FO}$, $\textsc{Sat}(v, \varphi, \sigma', X)$ implies

$$\forall ts \in \mathbb{N}, D \in \mathbb{DB}, \sigma'' \in \mathbb{T}_\omega.\ (\forall (\xi, v', p') \in X.\ v', |\sigma'| \vDash_{\sigma' \cdot \overline{(ts,D)\cdot\sigma''}^{\mathtt{TP}}} p'\xi(ts)) \implies v, |\sigma'| - 1 \vDash_{\sigma' \cdot \overline{(ts,D)\cdot\sigma''}^{\mathtt{TP}}} \varphi. \qquad (\star)$$

Intuitively, this condition expresses that whenever $\textsc{Sat}(v, \varphi, \sigma', X)$ is true and the (infinite) trace $\sigma = \sigma' \cdot \overline{(ts, D) \cdot \sigma''}^{\mathtt{TP}}$ satisfies all the future obligations in X at time-point $|\sigma'|$, then φ holds over σ at time-point $|\sigma'| - 1$.

For our algorithm to eventually recognize satisfaction and terminate, one must ensure that for large enough X, the implication $(\star)$ is an equivalence. This guarantees that after generating a finite set of reactions and future obligations, the algorithm can use Sat to assess that no more immediate actions are needed.

To support assumptions about the future, we extend Lima et al's algorithm [49] with the proof rules in Figure 6. In our extended report [42], we show

Lemma 2. *The proof system of [49] extended with the rules from Figure 6 yields a decision procedure* Sat *that satisfies* $(\star)$.

Lemma 3. *There exists a set* $\mathsf{FO}^+_{i,ts}(\varphi)$ *such that whenever* $X \supseteq \mathsf{FO}^+_{|\sigma|,\tau_{|\sigma|}}(\varphi)$, *the converse of* $(\star)$ *also holds for* Sat *constructed as in Lemma 2.*

5.3 The enforcement algorithm

Our enforcer's update function enf is shown in Algorithm 2. It is used to define an enforcer $\mathcal{E}_\varphi = (\mathcal{S}, s_\varphi, \mathsf{enf})$, where $\mathcal{S} = \mathcal{P}(\mathsf{FO})$ and $s_\varphi = \{(\mathsf{fo}_{\mathsf{init},\varphi}, \emptyset, +)\}$. In the algorithm and its description below, we annotate operators that fulfill the typing conditions in Figure 3 with the respective typing rule names. For example, we write $\varphi \mathsf{U}^{\mathbb{C}\mathrm{LR}}_{[a,b]} \psi$ to denote $\varphi \mathsf{U}_{[a,b]} \psi$ where $b \neq \infty$, $\Gamma \vdash \varphi : \mathbb{C}$, and $\Gamma \vdash \psi : \mathbb{C}$ under some Γ.

As required by Definition 2, the function enf takes a trace σ, a set of future obligations X, and a timestamp ts as input. If $ts = \bot$, i.e., the enforcer processes

a time-point already present in the trace, then ts is set to the latest timestamp $\tau_{|\tau|}$ (line 4). The enforcer computes a (closed) formula Φ that summarizes all obligations at the present time-point (line 5). Then Φ, σ, an empty set of future obligations, and an empty valuation are passed to $\mathsf{enf}^+_{ts,\bot}$ (line 6). The function $\mathsf{enf}^+_{ts,b}$ takes a formula φ, a trace σ, a set of (new) future obligations X, and a valuation v as input, and returns a triple (D_C, D_S, X') such that D_C is a set of events to cause, D_S is a set of events to suppress, and X' is an updated version of X. The function is parameterized by the current timestamp ts and a Boolean b that is true iff the current time-point is the last one with the current timestamp. The definition of enf^+ (resp. enf^-) guarantees that if we update D_i according to D_S and D_C and assume that all obligations in X' are satisfied at time-point $i+1$, then φ is always (resp. never) satisfied under v at i on the new trace.

After computing D_S, D_C, and X', an R-command $\mathsf{RCom}(D_C, D_S)$ is returned (line 7) and the state is updated to X'. If $ts \neq \bot$, a similar approach is followed, but now $\mathtt{TP}$ is not conjoined with Φ (line 9) and the boolean b is set to $\top$ as enforcement happens 'in the nick of time.' If $\mathtt{TP}$ is part of the set D_C returned by enf^+, then a P-command $\mathsf{PCom}(D_C)$ and a new state X' are returned. Otherwise, NoCom is returned and the state is not updated.

The functions enf^+ and enf^- recurse over the structure of φ. The traversal of φ is guided by the typing: the function enf^+ (resp. enf^-) is only called on subformulae of type $\mathbb{C}$ (resp. $\mathbb{S}$). The algorithm implements the approach described in Section 4. For space reasons, we only explain the more complex cases: $\varphi = \varphi_1 \wedge^{\mathbb{C}} \varphi_2$, $\varphi = \exists^{\mathbb{S}} x.\ \varphi_1$, and $\varphi = \varphi_1 \,\mathsf{U}_I^{\mathrm{CLR}}\, \varphi_2$.

Causing $\varphi_1 \wedge \varphi_2$ (Algorithm 2, enf^+ l. 9). Causing $\varphi_1 \wedge \varphi_2$ where both φ_1 and φ_2 are causable requires a fixed-point computation [39]. Consider, e.g., the EMFOTL formula $\varphi = \psi \wedge (\psi \to \chi)$, where ψ and χ both type to $\mathbb{C}$. If neither ψ nor χ are satisfied, then the right conjunct of φ is satisfied; however, to satisfy the left conjunct, ψ must be caused. But after causing ψ, the right conjunct is not satisfied, and χ must be caused too. In general, the two conjuncts are repeatedly enforced until both are satisfied. This is achieved by combining the function fp (performing a fixed-point computation) and $\mathsf{enf}^+_{\mathsf{and},\varphi_1,\varphi_2,v,ts}$ that calls the function enf^+ on both φ_1 and φ_2 if none of these formulae is satisfied. In our extended report [42], we prove the termination of this fixed-point computation.

Suppressing $\exists x.\ \varphi_1$ (Algorithm 2, enf^- l. 13). The suppression of $\exists$ follows a similar pattern, but this time there are $\mathsf{AD}_{|\sigma|}(\varphi_1)$ rather than just 2 cases to consider, corresponding to all potential values of the (past-guarded) variable x. Similar to the previous case, we prove termination in our extended report [42].

Causing $\varphi_1 \,\mathsf{U}_{[a,b]}\, \varphi_2$**,** $b \neq \infty$ (Algorithm 2, enf^+ l. 17–22). There are two cases for causing $\varphi_1 \,\mathsf{U}_I\, \varphi_2$: we cause φ_1 and generate the future obligation $\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}$ if $I \neq [0,0]$ or $b = \bot$; otherwise, we cause φ_2 and $\mathtt{TP}$.

Example 7. Let us enforce φ_{del} on σ_2. Consider the following abbreviations:

$$\varphi_{\mathsf{del}} \equiv \Box\, \varphi_\forall \qquad \varphi_\forall \equiv \forall c, d, u.\, \mathsf{deletion\_request}\,(c,d,u) \to \Diamond_{[0,30]}\, \mathsf{delete}\,(c,d,u)$$
$$\varphi_{\mathsf{U}} \equiv (\mathtt{TP} \to \top)\ \mathsf{U}\ (\mathtt{TP} \wedge \neg \varphi_\forall) \quad E_1 \equiv \mathsf{deletion\_request}(2,1,1) \quad E_1' \equiv \mathsf{delete}(2,1,1)$$
$$fo_2^{x,y} \equiv (\lambda \tau'.\ \Diamond_{[0,x]-(y-\tau')}\,(\mathtt{TP} \wedge \mathsf{delete}\,(c,d,u))\,, \{c \mapsto 2, d \mapsto 1, u \mapsto 1\}, +)$$

function $\mathsf{enf}(\sigma, X, ts)$
  let $\langle \tau \rangle, \langle D \rangle = \mathsf{unzip}(\sigma)$ **in**
  if $ts = \bot$ **then**
    let $ts = \tau_{|\tau|}$ **in**
    let $\Phi = \mathtt{TP} \wedge \bigwedge_{(\xi,v,\top)\in X} \xi(ts)[v] \wedge \bigwedge_{(\xi,v,\bot)\in X} \neg\xi(ts)[v]$ **in**
    let $(D_C, D_S, X') = \mathsf{enf}^+_{ts,\bot}(\Phi, \sigma, \emptyset, \emptyset)$ **in**
    $(\mathsf{RCom}(C \setminus \{\mathtt{TP}\}, S), X')$
  else
    let $\Phi = \bigwedge_{(\xi,v,\top)\in X} \xi(ts)[v] \wedge \bigwedge_{(\xi,v,\bot)\in X} \neg\xi(ts)[v]$ **in**
    let $(D_C, D_S, X') = \mathsf{enf}^+_{ts,\top}(\Phi, \sigma \cdot (ts, \emptyset), \emptyset, \emptyset)$ **in**
    if $\mathtt{TP} \in D_C$ then $(\mathsf{PCom}(D_C \setminus \{\mathtt{TP}\}), X')$ else (NoCom, X)
  end if
end function

function $\mathsf{enf}^+_{ts,b}(\varphi, \sigma, X, v)$
  if $\varphi = \top^{\mathbb{C}}$ **then**
    $(\emptyset, \emptyset, \emptyset)$
  else if $\varphi = p(\bar{t})$ **then**
    $(\{(p, (\![\bar{t}]\!]_v))\}, \emptyset, \emptyset)$
  else if $\varphi = \neg^{\mathbb{C}}\varphi_1$ **then**
    $\mathsf{enf}^-_{ts,b}(\varphi_1, \sigma, X, v)$
  else if $\varphi = \varphi_1 \wedge^{\mathbb{C}} \varphi_2$ **then**
    $\mathsf{fp}(\sigma, X, \mathsf{enf}^+_{\mathsf{and},\varphi_1,\varphi_2,v,ts})$
  else if $\varphi = \exists^{\mathbb{C}} x.\ \varphi_1$ **then**
    $\mathsf{enf}^+_{ts,b}(\varphi_1, \sigma, X, v[0/x])$
  else if $\varphi = \bigcirc^{\mathbb{C}}_I \varphi_1$ **then**
    $(\emptyset, \emptyset, \{(\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1}, v, +)\})$
  else if $\varphi = \varphi_1\, \mathsf{S}^{\mathbb{C}}_I\, \varphi_2$ **then**
    $\mathsf{enf}^+_{ts,b}(\varphi_2, \sigma, X, v)$
  else if $\varphi = \varphi_1\, \mathsf{U}^{\mathbb{CLR}}_I\, \varphi_2$ **then**
    if $I = [0,0] \wedge b$ **then**
      $\mathsf{enf}^+_{ts,b}(\varphi_2, \sigma, X, v) \Cup (\{\mathtt{TP}\}, \emptyset, \emptyset)$
    else
      $\mathsf{enf}^+_{ts,b}(\varphi_1, \sigma, X, v) \Cup$
      $(\emptyset, \emptyset, \{(\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}, v, +)\})$
    end if
  else if $\varphi = \varphi_1\, \mathsf{U}^{\mathbb{CR}}_I\, \varphi_2$ **then**
    if $I = [0,0] \wedge b$ **then**
      $\mathsf{enf}^+_{ts,b}(\varphi_2, \sigma, X, v) \Cup (\{\mathtt{TP}\}, \emptyset, \emptyset)$
    else if $\neg\textsc{Sat}(v, \varphi_1, \sigma, X)$ **then**
      $\mathsf{enf}^+_{ts,b}(\varphi_2, \sigma, X, v)$
    else
      $(\emptyset, \emptyset, \{(\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}, v, +)\})$
    end if
  end if
end function

function $\mathsf{fp}(\sigma \cdot \langle(\tau, D)\rangle, X, f)$
  $(D_C, D_S) \leftarrow (\emptyset, \emptyset)$
  $r \leftarrow \mathsf{None}$
  while $(D_C, D_S, X) \neq r$ **do**
    $r \leftarrow (D_S, D_C, X)$
    let $D' = (D \setminus D_S) \cup D_C$ **in**
    $(D_C, D_S, X) \leftarrow r \Cup f(\sigma \cdot \langle(\tau, D')\rangle, X)$
  end while
  (D_C, D_S, X)
end function

function $\mathsf{enf}^-_{\mathsf{ex},\varphi_1,v,ts,b}(\sigma, X)$
  $r \leftarrow (\emptyset, \emptyset, \emptyset)$
  for $d \in \mathsf{AD}_{|\sigma|}(\varphi_1)$ **do**
    if $\neg\textsc{Sat}(v[d/x], \neg\varphi_1, \sigma, X)$ **then**
      $r \leftarrow r \Cup \mathsf{enf}^-_{ts,b}(\varphi_1, \sigma, X, v[d/x])$
    end if
  end for
  r
end function

function $\mathsf{enf}^-_{ts,b}(\varphi, \sigma, X, v)$
  if $\varphi = \bot^{\mathbb{S}}$ **then**
    $(\emptyset, \emptyset, \emptyset)$
  else if $\varphi = p(\bar{t})$ **then**
    $(\emptyset, \{(p, (\![\bar{t}]\!]_v))\}, \emptyset)$
  else if $\varphi = \neg^{\mathbb{S}}\varphi_1$ **then**
    $\mathsf{enf}^+_{ts,b}(\varphi_1, \sigma, X, v)$
  else if $\varphi = \varphi_1 \wedge^{\mathbb{SL}} \varphi_2$ **then**
    $\mathsf{enf}^-_{ts,b}(\varphi_1, \sigma, X, v)$
  else if $\varphi = \varphi_1 \wedge^{\mathbb{SR}} \varphi_2$ **then**
    $\mathsf{enf}^-_{ts,b}(\varphi_2, \sigma, X, v)$
  else if $\varphi = \exists^{\mathbb{S}} x.\ \varphi_1$ **then**
    $\mathsf{fp}(\sigma, X, \mathsf{enf}^-_{\mathsf{ex},\varphi_1,v,ts,b})$
  else if $\varphi = \bigcirc^{\mathbb{S}}_I \varphi_1$ **then**
    $(\emptyset, \emptyset, \{(\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1}, v, -)\})$
  else if $\varphi = \varphi_1 \mathsf{S}^{\mathbb{SL}}_I \varphi_2$ **then**
    $\mathsf{enf}^-_{ts,b}(\varphi_1, \sigma, X, v)$
  else if $\varphi = \varphi_1 \mathsf{S}^{\mathbb{SR}}_I \varphi_2$ **then**
    let $\varphi' =$
     $\neg(\varphi_1 \wedge^{\mathbb{SL}} (\varphi_1\, \mathsf{S}_I\, \varphi_2))$ **in**
    $\mathsf{fp}(\sigma, X, \mathsf{enf}^+_{\mathsf{and},\varphi',\neg\varphi_2,v,ts,b})$
  else if $\varphi = \varphi_1 \mathsf{U}^{\mathbb{S}}_I \varphi_2$ **then**
    $\mathsf{fp}(\sigma, X, \mathsf{enf}^-_{\mathsf{until},I,\varphi_1,\varphi_2,v,ts,b})$
  end if
end function

function $\mathsf{enf}^-_{\mathsf{until},I,\varphi_1,\varphi_2,v,ts,b}(\sigma, X)$
  $r \leftarrow (\emptyset, \emptyset, \emptyset)$
  if $0 \in I \wedge \neg\textsc{Sat}(v, \neg\varphi_2, \sigma, X)$ **then**
    $r \leftarrow \mathsf{enf}^-_{ts,b}(\varphi_2, \sigma, X, v)$
  end if
  if $\neg\textsc{Sat}(v, \neg\varphi_1, \sigma, X)$ **then**
    $r \leftarrow r \Cup (\emptyset, \emptyset, \{(\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}, v, -)\}$
  end if
  r
end function

function $\mathsf{enf}^+_{\mathsf{and},\varphi_1,\varphi_2,v,ts,b}(\sigma, X)$
  $r \leftarrow (\emptyset, \emptyset, \emptyset)$
  if $\neg\textsc{Sat}(v, \varphi_1, \sigma, X)$ **then**
    $r \leftarrow r \Cup \mathsf{enf}^+_{ts,b}(\varphi_1, \sigma, X, v)$
  end if
  if $\neg\textsc{Sat}(v, \varphi_2, \sigma, X)$ **then**
    $r \leftarrow r \Cup \mathsf{enf}^+_{ts,b}(\varphi_2, \sigma, X, v)$
  end if
  r
end function

Algorithm 2: Proactive real-time first-order enforcement algorithm

tp	ts	b	X	Φ	D_C	D_S	X'	*Response*
0	10	$\bot$	$\{(\lambda\_.\, \varphi_{\mathsf{del}}, \emptyset, +)\}$	$\mathsf{TP} \wedge \varphi_{\mathsf{del}}$	$\{\mathsf{TP}\}$	$\emptyset$	$\{fo_1, fo_2^{30,10}\}$	$\mathsf{RCom}(\emptyset, \emptyset)$
–	10	$\top$	$\{fo_1, fo_2^{30,10}\}$	$\varphi_{\mathsf{U}} \wedge \Diamond_{[0,30]}(\mathsf{TP} \wedge E_1')$	$\emptyset$	$\emptyset$	$\{fo_1, fo_2^{30,10}\}$	NoCom
–	11	$\top$	$\{fo_1, fo_2^{30,10}\}$	$\varphi_{\mathsf{U}} \wedge \Diamond_{[0,29]}(\mathsf{TP} \wedge E_1')$	$\emptyset$	$\emptyset$	$\{fo_1, fo_2^{29,11}\}$	NoCom
...	...	...	...	...	...	...	...	...
–	39	$\top$	$\{fo_1, fo_2^{2,38}\}$	$\varphi_{\mathsf{U}} \wedge \Diamond_{[0,1]}(\mathsf{TP} \wedge E_1')$	$\emptyset$	$\emptyset$	$\{fo_1, fo_2^{1,39}\}$	NoCom
–	40	$\top$	$\{fo_1, fo_2^{1,39}\}$	$\varphi_{\mathsf{U}} \wedge \Diamond_{[0,0]}(\mathsf{TP} \wedge E_1')$	$\{\mathsf{TP}, E_1'\}$	$\emptyset$	$\{fo_1\}$	$\mathsf{PCom}(\{E_1'\})$
–	41	$\top$	$\{fo_1\}$	φ_{del}	$\emptyset$	$\emptyset$	$\{fo_1\}$	NoCom
...	...	...	...	...	...	...	...	...
–	49	$\top$	$\{fo_1\}$	φ_{del}	$\emptyset$	$\emptyset$	$\{fo_1\}$	NoCom
1	50	$\bot$	$\{fo_1\}$	$\mathsf{TP} \wedge \varphi_{\mathsf{del}}$	$\{\mathsf{TP}\}$	$\emptyset$	$\{fo_1\}$	$\mathsf{RCom}(\emptyset, \emptyset)$

Fig. 7. Enforcement of the formula φ_{del} on trace σ_2

Figure 7 shows our algorithm's execution. Initially, enf decomposes its goal $\Phi = \mathsf{TP} \wedge \varphi_{\mathsf{del}}$ into the present obligations $(\mathsf{TP}, \emptyset, +)$ and $(\varphi_{\mathsf{del}}, \emptyset, +)$. The former is discharged by causing TP; the latter is unrolled into the present obligation $(\varphi_\forall, \emptyset, +)$ and the future obligation $fo_1 = (\mathsf{fo}_{10,\mathsf{U},[0,\infty),\top,\neg\varphi_\forall}, \emptyset, -) = (\lambda\_.\, \varphi_{\mathsf{U}}, \emptyset, -)$. The present obligation $(\varphi_\forall, \emptyset, +)$ is violated, since deletion_request$(2,1,1)$ is satisfied but at this point there is no corresponding delete. In this case, $\mathsf{enf}^+_{10,\bot}$ generates the future obligation $fo_2^{30,10}$. Satisfying this future obligation guarantees the satisfaction of Φ, hence the algorithm proceeds. Next, the algorithm processes the timestamp 10 'in the nick of time'. The function enf computes $\Phi = fo_1(10) \wedge fo_2^{30,10}(10) = \varphi_{\mathsf{U}} \wedge \Diamond_{[0,30]}(\mathsf{TP} \wedge E_1')$ and calls $\mathsf{enf}^+_{10,\top}$ on Φ. First, it decomposes Φ into the present obligations $po_1 = (\varphi_\forall, \emptyset, +)$ and $po_2 = (\Diamond_{[0,30]}(\mathsf{TP} \wedge E_1'), \emptyset, +)$ and the future obligation fo_1. The present obligation po_1 is vacuously satisfied, since no deletion_request takes place. In contrast, the satisfaction of po_2 can rely on the satisfaction of the future obligation $(fo_2^{30,10}, \emptyset, +)$ at the next time-point. Hence, the enforcer emits NoCom and propagates the future obligations $X' = \{fo_1, fo_2^{30,10}\}$ to the next time-point. The timestamp 11 is also processed 'in the nick of time'. The goal $\Phi = fo_1(11) \wedge fo_2^{30,10}(11) = \varphi_{\mathsf{U}} \wedge \Diamond_{[0,29]}(\mathsf{TP} \wedge E_1')$ is computed, and reduced to the future obligations $X' = \{fo_1, fo_2^{29,11}\}$. Similar iterations occur until timestamp 40, when the goal becomes $\Phi = fo_1(40) \wedge fo_2^{1,39}(40) = \varphi_{\mathsf{U}} \wedge \Diamond_{[0,0]}(\mathsf{TP} \wedge E_1')$. Here, $\mathsf{enf}^+_{40,\top}$ produces the present obligations $(\mathsf{TP}, \emptyset, +)$ and $(E_1', \emptyset, +)$, which are discharged by causing TP and E_1', respectively. Thus, $D_C = \{\mathsf{TP}, E_1'\}$ and the command $\mathsf{PCom}(\{E_1'\})$ is emitted, resulting in $(40, \{E_1'\})$ being inserted into the trace. The future obligations $X' = \{fo_1\}$ are propagated to the next timestamp. Similar iterations occur until timestamp 50. At this point, $b = \bot$ and the trace is already compliant, so the enforcer responds with $\mathsf{RCom}(\emptyset, \emptyset)$.

5.4 Correctness

Let φ be a closed formula to be enforced. The proofs of all lemmata are given in our extended report [42]. First, recall the following standard definition of safety [6]:

Definition 6. *P is a* safety property *iff for any $\sigma \in \mathbb{T}_\omega \setminus P$, there exists a finite prefix $\sigma' \in \mathbb{T}_f$ of σ such that for all $\sigma'' \in \mathbb{T}_\omega$, we have $\sigma \cdot \sigma'' \notin P$. A formula φ is a* safety formula *when $\mathcal{L}(\varphi)$ is a safety property.*

Our algorithm can enforce formulae that are *not* safety formulae. This is the case, e.g., for any $\psi \vee \Diamond \chi \equiv \neg(\neg\psi \wedge \neg(\top \mathsf{U} \chi))$, where ψ types to $\mathbb{C}$. In this case, enforcement is performed greedily: if the monitor cannot construct a proof of $\Diamond \chi$ (which occurs whenever χ cannot be satisfied in the present), then ψ is caused. Thus our algorithm actually enforces a stronger formula, which we denote by $[\psi \vee \Diamond \chi]_+ \equiv \neg(\neg\psi \wedge^{\mathrm{R}\omega} \neg(\top \mathsf{U} \chi))$, where $\wedge^{\mathrm{R}\omega}$ has the semantics

$$v, i \vDash_\sigma \varphi \wedge^{\mathrm{R}\omega} \psi \qquad \text{iff } v, i \vDash_\sigma \varphi \text{ and } \exists \sigma'.\, v, i \vDash_{\sigma|_{..i} \cdot \sigma'} \psi.$$

This semantics states that $\varphi \wedge^{\mathrm{R}\omega} \psi$ holds whenever φ holds on σ at time-point i and there exists at least one extension of the prefix $\sigma|_{..i}$ on which ψ holds. The formula $[\psi \vee \Diamond \chi]_+$ thus requires than ψ holds on σ at time-point i and $\Diamond \psi$ holds on σ at time-point i for any extension of $\sigma\,|_{..i}$. The formula $[\psi \vee \Diamond \chi]_+$, unlike $\psi \vee \Diamond \chi$, is safety. In our extended report [42], we define a similar transformation $[\bullet]_p$, $p \in \{+, -\}$ for all operators and prove

Lemma 4. *For any φ such that $\Gamma \vdash \varphi : \{\mathbb{C}\}_p$, we have $v, i \models_\sigma p[\varphi]_p \implies v, i \models_\sigma p\varphi$. In particular, $\mathcal{L}([\varphi]_+) \subseteq \mathcal{L}(\varphi)$.*

We prove that $\mathcal{E}_\varphi$ soundly enforces $[\varphi]_+$, and hence φ:

Theorem 1 (Soundness). *If $\varphi \in$ EMFOTL, the enforcer $\mathcal{E}_\varphi$ is sound with respect to $\mathcal{L}([\varphi]_+) \subseteq \mathcal{L}(\varphi)$. As a consequence, φ is enforceable.*

In our model, *transparent* enforcement of non-safety formulae such as $\psi \vee \Diamond \chi$ is generally not possible, since the necessity to cause ψ depends on future events:

Lemma 5. *If a property admits a transparent enforcer, it is a safety formula.*

Thus, when enforcing a non-safety formula φ, one can at best achieve transparency with respect to some sound safety approximation φ' of φ. We prove:

Theorem 2 (Transparency). *If $\varphi \in$ EMFOTL, the enforcer $\mathcal{E}_\varphi$ is transparent with respect to $\mathcal{L}([\varphi]_+)$.*

By imposing more constraints on the formulae (e.g., the formula χ must not depend on the future in $\psi \wedge^{\mathbb{SL}} \chi$), one can obtain an EMFOTL fragment for which $[\varphi]_+ = \varphi$ and the enforcer $\mathcal{E}_\varphi$ is transparent (see our extended report [42]).

6 Evaluation

We implemented our type system and enforcement algorithm in a tool, called WhyEnf, consisting of 2 800 lines of OCaml code. WhyEnf uses a modified version of WhyMon [49], which we call WhyMon*. It ignores the explanations' structures (not required by our algorithm) and returns only Boolean verdicts.

Our evaluation aims to answer the following research questions:

RQ1. Is EMFOTL expressive enough to formalize real-world policies? Is manual formula rewriting necessary, as in previous works [14, 40]?
RQ2. At what maximum event rate can WhyEnf perform real-time enforcement?
RQ3. Do WhyEnf's performance and capabilities improve upon the state-of-the-art?

The notion of 'real-world policies' in RQ1 is domain-dependent. In the following, we demonstrate our approach's effectiveness in the case of privacy regulations.

$\mathsf{collect}(c,d,u)^{699}$	$\mathsf{use}(c,d,u)\text{-}^{2316}$	$\mathsf{consent}(u,c)^{699}$	$\mathsf{legal\_grounds}(u,d)^{397}$	$\mathsf{revoke}(u,c)\text{+}^{8}$
$\mathsf{inform}(u)\text{+}^{0}$	$\mathsf{deletion\_request}(c,d,u)^{8}$	$\mathsf{delete}(c,d,u)\text{+}^{521}$	$\mathsf{share}(p,d)^{982}$	$\mathsf{notify}(p,d)\text{+}^{0}$

"Minimization"	$\varphi_{\mathsf{min}} = \square(\forall c,d,u.\, \mathsf{collect}(c,d,u) \rightarrow \Diamond\, \mathsf{use}(c,d,u))$
"Limitation"	$\varphi_{\mathsf{lim}} = \square(\forall c,d,u.\, \mathsf{collect}(c,d,u) \rightarrow \Diamond\, \mathsf{delete}(c,d,u))$
"Lawfulness"	$\varphi_{\mathsf{law}} = \square(\forall c,d,u.\, \mathsf{use}(c,d,u) \rightarrow \blacklozenge(\mathsf{consent}(u,c) \vee \mathsf{legal\_grounds}(u,d)))$
"Consent"	$\varphi_{\mathsf{con}} = \square(\forall c,d,u.\, \mathsf{use}(c,d,u) \rightarrow (\blacklozenge\, \mathsf{legal\_grounds}(u,d)) \vee (\neg \mathsf{revoke}(u,c)\ \mathsf{S}\ \mathsf{consent}(u,c)))$
"Information"	$\varphi_{\mathsf{inf}} = \square(\forall c,d,u.\, \mathsf{collect}(c,d,u) \rightarrow ((\bigcirc\, \mathsf{inform}(u)) \vee (\blacklozenge\, \mathsf{inform}(u))))$
"Deletion"	$\varphi_{\mathsf{del}} = \square(\forall c,d,u.\, \mathsf{deletion\_request}(c,d,u) \rightarrow \Diamond_{[0,30]}\, \mathsf{delete}(c,d,u))$
"Sharing"	$\varphi_{\mathsf{sha}} = \square(\forall c,d,u,p.\, \mathsf{deletion\_request}(c,d,u) \wedge (\blacklozenge\, \mathsf{share}(p,d)) \rightarrow \Diamond_{[0,30]}\, \mathsf{notify}(p,d))$

c: data category; d: data ID; u: user ID; p: processor ID; -: suppressable; +: causable

Fig. 8. Selected events and policies from Arfelt et al. [7]

Case study. Arfelt et al. [7] define events and MFOTL formulae formalizing core GDPR provisions that they monitor on a trace produced by a real-world system [24]. Relevant events (superscripted by their number of occurrences in the trace) and formulae are shown in Figure 8 and Examples 1 and 3. We pre-process the trace to obtain 3 846 time-points containing 5 630 system events distributed over 515 days. We interpret the 'Lawyer review' and 'Architect review' events as both use and share (sharing with third-parties) events, and the 'Abort' events as both revoke (revoking consent) and deletion_request. Otherwise, we follow Arfelt et al.'s pre-processing. We make the following assumptions [40]: use events are suppressable, while delete, inform (informing the user), and notify (notifying a third-party) events are causable. All metric constraints are specified in days.

RQ1: Expressiveness. Except for φ_{min}, all formulae are in EMFOTL. Unlike in previous works [15, 18, 40], no further policy engineering (e.g., manual rewriting to equivalent formulae in supported fragments) is needed. For all enforceable formulae except φ_{lim}, our algorithm guarantees *transparent* enforcement. For φ_{lim}, which contains an unbounded $\Diamond$ operator, non-transparent enforcement is possible by enforcing the stronger formula $\varphi^{b}_{\mathsf{lim}} = \square(\forall c,d,u.\, \mathsf{collect}(c,d,u) \rightarrow \Diamond_{[0,b]}\, \mathsf{delete}(c,d,u))$ for any $b \in \mathbb{N}$. The formula φ_{min}, capturing *data minimization*, is intrinsically non-enforceable, as a sound $\mathcal{E}_{\varphi_{\mathsf{min}}}$ must either always suppress collect, or eventually cause use, which is only suppressable.

WhyEnf's type system helps determine appropriate suppressible and causable events. For instance, if use was marked as only-observable, the type checker would state that φ_{law} is not enforceable and suggest to make use suppressible, or otherwise make either consent or legal_ground causable. Since use actually is suppressable, the type checker concludes that φ_{law} is transparently enforceable.

RQ2: Maximum event rate. We enforce the enforceable formulae from Figure 8, i.e., all but φ_{min}. As we do not have access to the SuS, we simulate online enforcement by reproducing [45] the events from the above trace to WhyEnf at the speed specified by the trace's timestamps. We also consider different accelerations of the original trace's real-time behavior to challenge WhyEnf. We

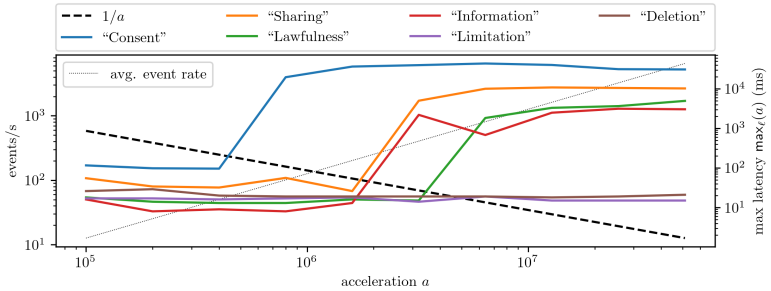

Fig. 9. RQ2: Maximum latency of WhyEnf and event rate for the formulae in Figure 8.

measure WhyEnf's latency ℓ and processing time t for each time-point. Latency is the time delay between the emission of a time-point to WhyEnf and the reception of the corresponding command, whereas processing time is the time WhyEnf effectively takes to process the time-point. We report the average latency ($\mathsf{avg}_\ell(a)$) and maximum latency ($\mathsf{max}_\ell(a)$) given an acceleration a, as well as the average processing time (avg_t), and the maximum processing time (max_t) all computed over the entire trace. If $\mathsf{max}_\ell(a)$ is smaller than the interval $\frac{1}{a}$ between two timestamps in the accelerated trace, then the real-time condition (Section 3.1) is met assuming that the SuS's and communication latency are small enough.

All measurements were performed on a 2.4 GHz Intel i5-1135G7 CPU with 32 GB RAM. For each formula and acceleration $a \in \{10^5 \cdot 2^0, \ldots, 10^5 \cdot 2^9\}$, we plot $\mathsf{max}_\ell(a)$, the function $\frac{1}{a}$ (right y-axis), and the corresponding average event rate $\mathsf{avg}_{er}(a)$ (left y-axis) in Figure 9. We include similar plots for WhyMon* and EnfPoly and latency profiles for individual runs in our extended report [42].

As presented in Figure 9, for all formulae, WhyEnf meets the real-time condition for all accelerations up to $4 \cdot 10^5$, which corresponds to a maximum latency of 96 ms and an average event rate of 51 events/s. Hence, even though the analyzed trace specifies time intervals in days, the real-time enforcement of the same trace can in fact be performed for sub-second intervals. Note that the average latency is much lower (20 ms for the most challenging policy), with the maximum latency occurring when many events occur within a short time span. The two formulae that only define future obligations, φ_{lim} and φ_{del}, have much lower maximum latency, of 14 and 19 ms, respectively, corresponding to an average event rate of about 600 events/s. Due to proactivity, the enforcer does not need to keep the history of past events for these formulae. Overall, our experiments show that WhyEnf can efficiently enforce a real-world SuS.

RQ3: Comparison with the state of the art. We compare WhyEnf's performance to its two most closely related tools: WhyMon*, which provides similar expressiveness as WhyEnf but no enforcement, and EnfPoly [39], the only tool supporting non-proactive enforcement of an MFOTL fragment. In addition to the real-world log [24], we generate synthetic traces with $n \in \{100 \cdot 2^0, \ldots, 100 \cdot 2^8\}$ time-points each containing $k \in \{2^0, \ldots, 2^8\}$ random events. We report avg_t for the three tools and six formulae in Figure 11, imposing a 10-minute timeout (t.o.).

	WhyEnf						WhyMon*						EnfPoly					
Policy	a	avg_ℓ	max_ℓ	avg_t	max_t	avg_{er}	a	avg_ℓ	max_ℓ	avg_t	max_t	avg_{er}	a	avg_ℓ	max_ℓ	avg_t	max_t	avg_{er}
φ_{lim}	3.2e6	0.19	14	0.22	1.0	632	has unbounded future						requires proactivity					
φ_{law}	3.2e6	2.6	15	2.6	15	405	3.2e6	2.5	12	2.5	12	405	5.1e7	0.10	1.0	0.14	1.0	6479
φ_{con}	4e5	20	96	20	96	51	8e5	9.3	51	9.3	52	101	5.1e7	0.10	1.0	0.14	1.0	6479
φ_{inf}	1.6e6	2.9	13	3.0	13	202	3.2e6	0.16	16	0.19	1.0	405	requires proactivity					
φ_{del}	3.2e6	0.19	19	0.22	1.0	632	1e5	42	434	42	434	13	requires proactivity					
φ_{sha}	1.6e6	4.6	26	4.7	26	202	1e5	69	289	69	299	13	requires proactivity					

Fig. 10. RQ2–3: Latency and processing time for the largest a such that $\mathsf{max}_\ell(a) \leq 1/a$.

	WhyEnf						WhyMon*						EnfPoly					
$k = 10$	n :	100	400	1.6e3	6.4e3	2.6e4	n :	100	400	1.6e3	6.4e3	2.6e4	n :	100	400	1.6e3	6.4e3	2.6e4
φ_{lim}		.29	.28	.28	.30	.30	has unbounded future						requires proactivity					
φ_{law}		.73	1.3	2.0	2.2	2.7		.26	.57	1.4	3.5	15		.16	.16	.16	.16	.16
φ_{con}		1.8	4.9	9.1	11	12		.53	1.7	7.4	11	t.o.		.19	.16	.18	.17	.17
φ_{inf}		.78	1.0	1.2	1.1	1.2		.22	.31	.51	1.0	2.2	requires proactivity					
φ_{del}		.17	.24	.26	.28	.56		.40	1.2	2.9	4.4	4.9	requires proactivity					
φ_{sha}		.86	2.3	5.3	7.6	7.0		.54	2.3	13	56	t.o.	requires proactivity					
$n = 1000$	k :	1	4	16	64	256	k :	1	4	16	64	256	k :	1	4	16	64	256
φ_{lim}		.24	.24	.35	.83	4.7	has unbounded future						requires proactivity					
φ_{law}		.61	1.2	2.2	2.9	6.1		.38	.71	1.3	1.6	2.3		.14	.19	.18	.22	.38
φ_{con}		1.4	4.1	9.5	11.5	13.3		1.2	4.3	5.3	4.2	4.9		.14	.16	.16	.20	.32
φ_{inf}		.48	.79	1.4	4.8	24		.21	.28	.44	.78	1.1	requires proactivity					
φ_{del}		.23	.24	.32	.40	1.0		.44	1.1	3.0	4.8	6.3	requires proactivity					
φ_{sha}		.78	3.2	7.4	7.1	12		1.2	4.3	9.7	14	16	requires proactivity					

Fig. 11. RQ3: Average processing time (ms) for different trace and time-point sizes.

WhyMon* cannot monitor φ_{lim}, as the formula has an unbounded $\Diamond$ operator. For all other formulae, WhyMon* satisfies the real-time condition for accelerations $a \leq 10^5$. WhyEnf's latency is at most twice WhyMon*'s for φ_{law} and φ_{con} as the enforcer calls the monitor at least once per iteration and also performs fixed-point computations (Figure 10). In contrast, WhyEnf can enforce φ_{lim} and has significantly (up to 22 times) lower latency for φ_{inf}, φ_{sha}, and φ_{del}. Unlike WhyMon*, WhyEnf is able to lazily evaluate implications involving future obligations, which improves its runtime performance. WhyEnf's processing time also scales better than WhyMon*'s for large values of n and k (Figure 11).

Only φ_{law} and φ_{con} are transparently enforceable without proactivity. We enforce them using EnfPoly after manually rewriting them into equivalent formulae in EnfPoly's fragment. WhyEnf's average and maximum latencies are higher than EnfPoly's, but WhyEnf's algorithm covers a much larger fragment of MFOTL than EnfPoly, which makes computating verdicts more costly. The same behavior is observed in terms of average processing time (Figure 11).

7 Related Work

Security automata [26, 58] were first used for enforcement by terminating the SuS. Fredrikson et al. [31] also terminate the SuS upon violation detection, but use symbolic automata which allow policies to refer to the SuS's state. Bauer et al. [21] investigate enforcers that can cause and suppress events, as do Ligatti et al. [47], who use edit automata with the ability to buffer events. Ngo et al. [51] study policy enforcement for reactive systems for which they disallow the enforcer to buffer events or inspect SuS code. Basin et al. [15] distinguish

between suppressable and only-observable events, without considering causation. More complex bidirectional enforcement [3, 4] and enforcement through delaying events [27, 54] have also been proposed. Pinisetty et al. [55] further allow the enforcer to inspect the SuS's code to perform *predictive* enforcement.

Most runtime enforcement approaches (and tools [28,29]) rely on automata as policies. Metric interval temporal logic formulae can be enforced via translation to timed automata [53, 57]. Basin et al. [11, 12] use dynamic condition response graphs [36] to formalize and enforce obligations in real time by suppressing and (proactively) causing events. Finally, controller synthesis tools for LTL [25, 44, 60], Timed CTL [22, 52], or MTL [38, 46] can generate enforcement mechanisms.

To the best of our knowledge, only a few approaches enforce *first-order temporal* policies. Hallé and Villemaire [33, 34] develop a monitor for LTL-FO$^+$, a first-order variant of future-only linear temporal logic. They use the monitor to block the system in case of detected policy violations, in the spirit of the work on security automata [26,58]. Hublet et al. [39–41] developed the EnfPoly tool that enforces policies from a fragment of MFOTL that can contain future operators, but only nested with past ones such that the formula overall does not refer to the future. Independently, Aceto et al. [2–5] consider the safety fragment of Hennessy-Milner Logic (HML) with recursion as their policy language. They generalize HML to allow quantification over event parameters, but do not support time constraints. They also focus on instrumentation scenarios where all events are suppressable.

A satisfiability checking tool [30] and many runtime monitoring tools support (different fragments of) MFOTL [23], including MonPoly [13, 17–19], VeriMon [9, 10, 59] and DejaVu [35]. Lima et al. [48] recently introduced Explanator2, an MTL monitor that outputs explanations. They later extended their work to MFOTL with the WhyMon tool [49], upon which our enforcer relies. WhyMon supports a large fragment of MFOTL as it uses partitioned decision trees to represent variable assignments. To the best of our knowledge, all existing monitoring tools only support safety formulae of the form $\Box\,\varphi$. Our work additionally supports (non-transparent) enforcement of some non-safety formulae.

8 Conclusion

We have presented the first proactive real-time enforcement algorithm and an efficient tool, WhyEnf, for metric first-order temporal logic. Our approach lends itself to a number of extensions. For instance, WhyMon's runtime performance can be optimized for large formulae. Features like complex data types [50], let bindings [61], and aggregations [16] would further improve our enforcer's expressiveness. Finally, refinements of the type system when the same event can be both caused and suppressed in different contexts would be a useful addition.

Acknowledgement. Hublet is supported by the Swiss National Science Foundation grant "Model-driven Security & Privacy" (204796). Lima and Traytel are supported by a Novo Nordisk Fonden start package grant (NNF20OC0063462). We thank the anonymous reviewers for their insightful feedback.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Ausiello, G., Dezani-Ciancaglini, M., Rocca, S.R.D. (eds.) 16th International Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 372, pp. 1–17. Springer (1989). https://doi.org/10.1007/BFB0035748
2. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: On runtime enforcement via suppressions. In: 29th International Conference on Concurrency Theory (2018)
3. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: On bidirectional runtime enforcement. In: International Conference on Formal Techniques for Distributed Objects, Components, and Systems. pp. 3–21. Springer (2021)
4. Aceto, L., Cassar, I., Francalanza, A., Ingolfsdottir, A.: Bidirectional runtime enforcement of first-order branching-time properties. Logical Methods in Computer Science **19** (2023)
5. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: On first-order runtime enforcement of branching-time properties. Acta Informatica pp. 1–67 (2023)
6. Alpern, B., Schneider, F.B.: Defining liveness. Information processing letters **21**(4), 181–185 (1985)
7. Arfelt, E., Basin, D., Debois, S.: Monitoring the GDPR. In: European Symposium on Research in Computer Security. pp. 681–699. Springer (2019)
8. Bartocci, E., Falcone, Y.: Lectures on runtime verification. Springer (2018)
9. Basin, D., Dardinier, T., Hauser, N., Heimes, L., Huerta y Munive, J., Kaletsch, N., Krstić, S., Marsicano, E., Raszyk, M., Schneider, J., Tirore, D.L., Traytel, D., Zingg, S.: VeriMon: A formally verified monitoring tool. In: Seidl, H., Liu, Z., Pasareanu, C.S. (eds.) 19th International Colloquium on Theoretical Aspects of Computing (ICTAC). LNCS, vol. 13572, pp. 1–6. Springer (2022)
10. Basin, D., Dardinier, T., Heimes, L., Krstić, S., Raszyk, M., Schneider, J., Traytel, D.: A formally verified, optimized monitor for metric first-order dynamic logic. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) 10th International Joint Conference on Automated Reasoning, (IJCAR). LNCS, vol. 12166, pp. 432–453. Springer (2020)
11. Basin, D., Debois, S., Hildebrandt, T.T.: In the nick of time: Proactive prevention of obligation violations. In: 29th Computer Security Foundations Symposium (CSF). pp. 120–134. IEEE (2016)
12. Basin, D., Debois, S., Hildebrandt, T.: Proactive enforcement of provisions and obligations. Journal of Computer Security To appear
13. Basin, D., Harvan, M., Klaedtke, F., Zălinescu, E.: MonPoly: Monitoring usage-control policies. In: 2nd International Conference on Runtime Verification, (RV). pp. 360–364. Springer (2012)
14. Basin, D., Harvan, M., Klaedtke, F., Zalinescu, E.: Monitoring data usage in distributed systems. IEEE Trans. on Software Engineering **39**(10), 1403–1426 (2013)
15. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. **16**(1), 1–26 (2013)
16. Basin, D., Klaedtke, F., Marinovic, S., Zălinescu, E.: Monitoring of temporal first-order properties with aggregations. Formal Methods Syst. Des. **46**, 262–285 (2015)
17. Basin, D., Klaedtke, F., Müller, S., Pfitzmann, B.: Runtime monitoring of metric first-order temporal properties. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2008)
18. Basin, D., Klaedtke, F., Müller, S., Zălinescu, E.: Monitoring metric first-order temporal properties. Journal of the ACM (JACM) **62**(2), 1–45 (2015)

19. Basin, D., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. RV-CuBES **3**, 19–28 (2017)
20. Basin, D., Krstić, S., Schneider, J., Traytel, D.: Correct and efficient policy monitoring, a retrospective. In: 21st International Symposium on Automated Technology for Verification and Analysis (ATVA). pp. 3–30. Springer (2023)
21. Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies. In: Workshop on Foundations of Computer Security (FCS). Citeseer (2002)
22. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) International Conference Computer Aided Verification (CAV). LNCS, vol. 4590, pp. 121–125. Springer (2007)
23. Chomicki, J.: Efficient checking of temporal integrity constraints using bounded history encoding. ACM Trans. on Database Systems (TODS) **20**(2), 149–186 (1995)
24. Debois, S., Slaats, T.: The analysis of a real life declarative process. In: 2015 IEEE Symposium Series on Computational Intelligence. pp. 1374–1382. IEEE (2015)
25. Ehlers, R.: Unbeast: Symbolic bounded synthesis. In: Abdulla, P.A., Leino, K.R.M. (eds.) International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 6605, pp. 272–275. Springer (2011)
26. Erlingsson, Ú., Schneider, F.: SASI enforcement of security policies: a retrospective. In: Kienzle, D., Zurko, M.E., Greenwald, S., Serbau, C. (eds.) Workshop on New Security Paradigms. pp. 87–95. ACM (1999)
27. Falcone, Y., Jéron, T., Marchand, H., Pinisetty, S.: Runtime enforcement of regular timed properties by suppressing and delaying events. Science of Computer Programming **123**, 2–41 (2016)
28. Falcone, Y., Krstić, S., Reger, G., Traytel, D.: A taxonomy for classifying runtime verification tools. Int. J. Softw. Tools Technol. Transf. **23**(2), 255–284 (2021)
29. Falcone, Y., Pinisetty, S.: On the runtime enforcement of timed properties. In: Finkbeiner, B., Mariani, L. (eds.) 19th International Conference on Runtime Verification, (RV). LNCS, vol. 11757, pp. 48–69. Springer (2019)
30. Feng, N., Marsso, L., Sabetzadeh, M., Chechik, M.: Early verification of legal compliance via bounded satisfiability checking. In: Enea, C., Lal, A. (eds.) CAV 2023. LNCS, vol. 13966, pp. 374–396. Springer (2023)
31. Fredrikson, M., Joiner, R., Jha, S., Reps, T.W., Porras, P.A., Saïdi, H., Yegneswaran, V.: Efficient runtime policy enforcement using counterexample-guided abstraction refinement. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 548–563. Springer (2012)
32. Gomaa, H.: Software modeling and design: UML, use cases, patterns, and software architectures. Cambridge University Press (2011)
33. Hallé, S., Villemaire, R.: Browser-based enforcement of interface contracts in web applications with beepbeep. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 648–653. Springer (2009)
34. Hallé, S., Villemaire, R.: Runtime enforcement of web service message contracts with data. IEEE Trans. Serv. Comput. **5**(2), 192–206 (2012)
35. Havelund, K., Peled, D., Ulus, D.: First-order temporal logic monitoring with bdds. Formal Methods Syst. Des. **56**(1-3), 1–21 (2020)
36. Hildebrandt, T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. The Journal of Logic and Algebraic Programming **82**(5-7), 164–185 (2013)
37. Hilty, M., Basin, D., Pretschner, A.: On obligations. In: Computer Security–ESORICS 2005: 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005. Proceedings 10. pp. 98–117. Springer (2005)

38. Hofmann, T., Schupp, S.: TACoS: A tool for MTL controller synthesis. In: Calinescu, R., Pasareanu, C.S. (eds.) International Conference on Software Engineering and Formal Methods (SEFM). LNCS, vol. 13085, pp. 372–379. Springer (2021)
39. Hublet, F., Basin, D., Krstić, S.: Real-time policy enforcement with metric first-order temporal logic. In: European Symposium on Research in Computer Security. pp. 211–232. Springer (2022)
40. Hublet, F., Basin, D., Krstić, S.: Enforcing the GDPR. In: Tsudik, G., Conti, M., Liang, K., Smaragdakis, G. (eds.) Computer Security – ESORICS 2023. LNCS, vol. 14344. Springer (2023)
41. Hublet, F., Basin, D., Krstić, S.: User-controlled privacy: Taint, track, and control. Proc. Priv. Enhancing Technol. **2024**(1), 597–616 (2024)
42. Hublet, F., Lima, L., Basin, D., Krstić, S., Traytel, D.: Proactive real-time first-order enforcement (extended report) (2024), `https://github.com/runtime-enforcement/whyenf/blob/main/docs/cav24-extended.pdf`
43. Hublet, F., Lima, L., Basin, D., Krstić, S., Traytel, D.: WhyEnf (2024), `https://github.com/runtime-enforcement/whyenf`
44. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: International Conference Formal Methods in Computer-Aided Design (FMCAD). pp. 117–124. IEEE (2006)
45. Krstić, S., Schneider, J.: A benchmark generator for online first-order monitoring. In: Runtime Verification: 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6–9, 2020, Proceedings 20. pp. 482–494. Springer (2020)
46. Li, G., Jensen, P., Larsen, K., Legay, A., Poulsen, D.: Practical controller synthesis for $\mathrm{MTL}_{0,\infty}$. In: Erdogmus, H., Havelund, K. (eds.) ACM SIGSOFT International SPIN Symposium on Model Checking of Software. pp. 102–111. ACM (2017)
47. Ligatti, J., Bauer, L., Walker, D.: Edit automata: Enforcement mechanisms for run-time security policies. International Journal of Information Security **4**, 2–16 (2005)
48. Lima, L., Herasimau, A., Raszyk, M., Traytel, D., Yuan, S.: Explainable online monitoring of metric temporal logic. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 473–491. Springer (2023)
49. Lima, L., Huerta y Munive, J.J., Traytel, D.: Explainable online monitoring of metric first-order temporal logic. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 288–307. Springer (2024)
50. Lima Graf, J., Krstić, S., Schneider, J.: Metric first-order temporal logic with complex data types. In: International Conference on Runtime Verification. pp. 126–147. Springer (2023)
51. Ngo, M., Massacci, F., Milushev, D., Piessens, F.: Runtime enforcement of security policies on black box reactive programs. In: Rajamani, S.K., Walker, D. (eds.) 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL). pp. 43–54. ACM (2015)
52. Peter, H., Ehlers, R., Mattmüller, R.: Synthia: Verification and synthesis for timed automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) International Conference on Computer Aided Verification (CAV). LNCS, vol. 6806, pp. 649–655. Springer (2011)
53. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H.: TiPEX: A tool chain for timed property enforcement during execution. In: International Conference on Runtime Verification (RV). pp. 306–320. Springer (2015)
54. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A., Nguena Timo, O.: Runtime enforcement of timed properties revisited. Formal Methods Syst. Des. **45**, 381–422 (2014)

55. Pinisetty, S., Preoteasa, V., Tripakis, S., Jéron, T., Falcone, Y., Marchand, H.: Predictive runtime enforcement. Formal Methods Syst. Des. **51**(1), 154–199 (2017)
56. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: 16th ACM Symposium on Principles of Programming Languages (POPL). pp. 179–190. ACM Press (1989)
57. Renard, M., Rollet, A., Falcone, Y.: GREP: games for the runtime enforcement of properties. In: Yevtushenko, N., Cavalli, A., Yenigün, H. (eds.) International Conference on Testing Software and Systems (ICTSS). LNCS, vol. 10533, pp. 259–275. Springer (2017)
58. Schneider, F.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
59. Schneider, J., Basin, D., Krstić, S., Traytel, D.: A formally verified monitor for metric first-order temporal logic. In: Finkbeiner, B., Mariani, L. (eds.) 19th International Conference on Runtime Verification. LNCS, vol. 11757, pp. 310–328. Springer (2019)
60. Zhu, S., Tabajara, L., Li, J., Pu, G., Vardi, M.: A symbolic approach to safety LTL synthesis. In: Strichman, O., Tzoref-Brill, R. (eds.) International Haifa Verification Conference (HVC). LNCS, vol. 10629, pp. 147–162. Springer (2017)
61. Zingg, S., Krstić, S., Raszyk, M., Schneider, J., Traytel, D.: Verified first-order monitoring with recursive rules. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 236–253. Springer (2022)