Proactive Real-Time First-Order Enforcement

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

[1] Department of Computer Science, ETH Zürich, Zurich, Switzerland
[2] Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

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 new 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 yet technically challenging task.

Runtime *enforcement* [49] tackles this problem by observing and controlling a target system, also known as system under scrutiny (SuS), such that its actions, possibly modified, comply with a given policy. Runtime enforcement is performed by a system called *enforcer*, which observes the SuS and influences its behavior as specified by the system model, e.g., by suppressing or causing SuS actions. Enforcement is thus an inherently *online* problem that must be 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 reported. Applications of runtime enforcement are manifold, ranging from safety protocols in industrial automation to regulatory compliance, e.g., enforcing privacy rules in systems processing personal data.

Policies can be decomposed into provisions and obligations [33]. 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 policies: Hublet et al.'s [35] 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 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 both 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 [43] 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 online [1]) is the first proactive first-order policy enforcer (Section 7). All proofs can be found in Appendix B.

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 (e.g., 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 denoted 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 $\mathcal{P}(\{(e, \bar{d}) \mid e \in \mathbb{E},\ \bar{d} \in \mathbb{D}^{\iota(e)}\})$. 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 satisfy *monotonicity* ($\forall i < |\sigma|.\ \tau_i \le \tau_{i+1}$) and *progress* (if $|\sigma| = \infty$, then $\lim_{i\to\infty} \tau_i = \infty$). 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 system uses data 3 with category 1.

MFOTL formulae are defined by the following grammar

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

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 $\bot := \neg\top$, $\varphi \to \psi := \neg\varphi \vee \psi$, $\varphi \leftrightarrow \psi := (\varphi \to \psi) \wedge (\psi \to \varphi)$, $\forall x.\ \varphi := \neg(\exists x.\ \neg\varphi)$, $\Diamond_I \varphi := \top\ \mathsf{U}_I\ \varphi$ (eventually), $\blacklozenge_I \varphi := \top\ \mathsf{S}_I\ \varphi$ (once), $\Box_I \varphi := \neg \Diamond_I \neg\varphi$ (always), and $\blacksquare_I \varphi := \neg \blacklozenge_I \neg\varphi$ (historically). A *polarity* $p \in \{+, -\}$ acts upon a formula φ by $+\varphi := \varphi$ and $-\varphi := \neg\varphi$. We omit intervals of the form $[0, \infty)$ from the subscript of the temporal operators. 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

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

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 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'.

$v, i \vDash e(\bar{t})$ 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

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. [43] 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, e.g., MonPoly [13] and VeriMon [9] which only support formulas in relational algebra normal form [20] and DejaVu [31] 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* [42] of its verdicts (satisfactions or violations) in the form of proofs that can be checked using a proof checker. An example proof that σ_2 does not satisfy φ_{del} is provided in Appendix A. We refer to Lima et al.'s work [43] 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 [35]. Within the model, we define enforcers (Section 3.2).

3.1 System model

Figure 2 shows a system S supervised by an enforcer E described using a communication diagram [30]. The system S interacts with an environment X that E can-

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

not control. The enforcer E must ensure that the sequence of actions executed by S complies with a given policy P. To achieve this, S creates an event (from $\mathbb{E}$) for each of its observable actions, and sends (possibly several) such events to E. The events are received by E in addition to the current time $\tau \in \mathbb{N}$, provided by a global clock that is incremented by steps of 1. E can instruct S to cause and suppress some of its observable actions. Actions that can be suppressed by E correspond to *suppressable* events ($\mathsf{Sup} \subseteq \mathbb{E}$), while actions that can be caused by E correspond to *causable* events ($\mathsf{Cau} \subseteq \mathbb{E}$). The remaining observable actions (neither suppressable nor causable) correspond to *only-observable* events ($\mathsf{Obs} = \mathbb{E} \setminus (\mathsf{Sup} \cup \mathsf{Cau})$).

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 only observe the remaining actions. The corresponding events are then $\mathsf{Cau} = \{\mathsf{delete}\}$, $\mathsf{Sup} = \{\mathsf{use}\}$, and $\mathsf{Obs} = \{\mathsf{consent}, \mathsf{legal\_ground}, \mathsf{deletion\_request}\}$.

To achieve its goal, 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 responds with a *reactive command* which we denote as $\mathsf{ReCom}(D_C, D_S)$, where $D_C \in \mathsf{DB}(\mathsf{Cau})$ is a set of causable events and $D \supseteq D_S \in \mathsf{DB}(\mathsf{Sup})$ is a set of suppressable events. S then performs the actions corresponding to the events in $(D \setminus D_S) \cup D_C$, i.e., all actions corresponding to events in D_C (resp. D_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 a reactive command $\mathsf{ReCom}(D_C, \emptyset)$. As no suppressable actions were 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 send a *proactive command* $\mathsf{PrCom}(D_C)$ with $D_C \in \mathsf{DB}(\mathsf{Cau})$ to S. The system S then performs the actions corresponding to the events in D_C. Note that sending a proactive command before a tick is always possible, but that 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.

1: $\mathsf{run}(s, \sigma, \sigma', \tau) = \textbf{case } \sigma' \textbf{ of}$
2: $\mid \varepsilon \Rightarrow \varepsilon$
3: $\mid (\tau', D) \cdot \sigma'' \textbf{ when } \tau' > \tau \Rightarrow \textbf{let } (o, s') = \mu(\sigma, s, \tau) \textbf{ in}$
4: $\quad \textbf{case } o \textbf{ of } \mid \mathsf{PrCom}(D_C) \Rightarrow (\tau, D_C) \cdot \mathsf{run}(s', \sigma \cdot (\tau, D_C), \sigma', \tau + 1)$
5: $\quad\quad\quad\quad \mid \mathsf{NoCom} \Rightarrow \mathsf{run}(s', \sigma, \sigma', \tau + 1)$
6: $\mid (\tau', D) \cdot \sigma'' \textbf{ when } \tau' = \tau \Rightarrow \textbf{let } (o, s') = \mu(\sigma \cdot (\tau', D), s, \bot); D' = (D \setminus D_S) \cup D_C) \textbf{ in}$
7: $\quad \textbf{case } o \textbf{ of } \mid \mathsf{ReCom}(D_C, D_S) \Rightarrow (\tau', D') \cdot \mathsf{run}(s', \sigma \cdot (\tau', D'), \sigma'', \tau + 1)$
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

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] provide two necessary conditions: (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. 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 consecutive prefixes of a trace of a SuS and returns commands as described by (1)–(3) in Section 3.1:

Definition 1. *A* command *is any element of the form* $\mathsf{ReCom}(D_C, D_S)$ *('reactive command'),* $\mathsf{PrCom}(D_C)$ *('proactive command'), or* NoCom *('no command') where* $D_C \in \mathsf{DB}(\mathsf{Cau})$ *and* $D_S \in \mathsf{DB}(\mathsf{Sup})$*. 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:*

$$\forall \sigma, \tau, D, s.\ \exists D_C, D_S, s'.\ \mu(\sigma \cdot (\tau, D), s, \bot) = (\mathsf{ReCom}(D_C, D_S), s') \wedge D_S \subseteq D$$
$$\forall \sigma, s, \tau \in \mathbb{N}.\ \exists D_C, s'.\ \mu(\sigma, s, \tau) \in \{(\mathsf{PrCom}(D_C), s'), (\mathsf{NoCom}, s')\}.$$

If μ's third argument is $\bot$, then μ returns a reactive command; if it is an integer timestamp, then μ returns either a proactive 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 a reactive command

(lines 6–7), and once before each clock tick to (possibly) generate a proactive command (lines 3–5). The generated commands are executed sequentially to produce the enforced trace $\mathcal{E}(\sigma)$. The enforced trace $\mathcal{E}(\sigma)$ thus reflects the actions performed by the SuS when composed with the enforcer as in Section 3.1.

To be considered *correct* with respect to a given property P, enforcers are typically required to fulfill two properties: *soundness* and *transparency* [41]. 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, 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, $\Gamma : \mathbb{E} \to \{\mathsf{Cau}, \mathsf{Sup}\}$ is a mapping from event names to either of the symbols Cau or Sup, φ is an MFOTL formula, and α is a type in $\{\mathsf{Cau}, \mathsf{Sup}\}$. EMFOTL is defined as the set of all φ for which $\exists \Gamma.\, \Gamma \vdash \varphi : \mathsf{Cau}$. Intuitively, a formula types to Cau under Γ ('φ is causable under Γ') if it can be enforced by causing events $e_c(\bar{t})$ such that $\Gamma(e_c) = \mathsf{Cau}$ and suppressing events $e_s(\bar{t})$ such that $\Gamma(e_s) = \mathsf{Sup}$. It types to Sup under Γ ('φ is suppressable under Γ') if $\neg\varphi$ can be enforced under the same conditions on Γ. The type names Cau and Sup overload the names of the sets of suppressable and causable events in a natural way: any event $e_c(\bar{t})$ with $e_c \in \mathsf{Cau}$ (resp. $e_s \in \mathsf{Sup}$) has type Cau (resp. Sup) under the context $\{e_c \mapsto \mathsf{Cau}\}$ (resp. $\{e_s \mapsto \mathsf{Sup}\}$).

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 True, False, Cau, Sup). The constant $\top$ (resp. $\bot$) is causable (resp. suppressable). Event $e(t_1, \ldots, t_k)$ is causable (resp. suppressable) under Γ if $e \in \mathsf{Cau}$ and $\Gamma(e) = \mathsf{Cau}$ (resp. $e \in \mathsf{Sup}$ and $\Gamma(e) = \mathsf{Sup}$).

Negation (Rules NotCau, NotSup). The negation operator straightforwardly exchanges Cau and Sup: a formula is causable iff its negation is suppressable; it is suppressable iff its negation is causable.

Conjunction (Rules AndCau, AndSupL/R). Causing and suppressing conjunctions is straightforward: a conjunction is causable if both of its conjuncts are causable; it is suppressable if either of its conjuncts is suppressable.

Quantifiers (Rules ExistsCau, ExistsSup). The formula $\varphi' = \exists x.\ \varphi$ is causable if φ is causable, since it is enough to set x to any value v and cause $\varphi[x/v]$ to cause $\exists x.\ \varphi$. On the other hand, 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

Past-guardedness

$$\frac{}{\vdash e(\dots,x,\dots) : \mathsf{PG}(x)^+}\ \textsc{PG}^+ \qquad \frac{\vdash \varphi : \mathsf{PG}(x)^{\neg p}}{\vdash \neg\varphi : \mathsf{PG}(x)^p}\ \textsc{NotPG}$$

$$\frac{x \neq z \quad \vdash \varphi : \mathsf{PG}(z)^p}{\vdash \exists x.\ \varphi : \mathsf{PG}(z)^p}\ \textsc{ExistsPG} \qquad \frac{\vdash \varphi : \mathsf{PG}(x)^+}{\vdash \varphi \wedge \psi : \mathsf{PG}(x)^+}\ \textsc{AndPGL}^+$$

$$\frac{\vdash \psi : \mathsf{PG}(x)^+}{\vdash \varphi \wedge \psi : \mathsf{PG}(x)^+}\ \textsc{AndPGR}^+ \qquad \frac{\vdash \varphi : \mathsf{PG}(x)^+}{\vdash \bullet_I\, \varphi : \mathsf{PG}(x)^+}\ \textsc{PrevPG}^+$$

$$\frac{0 \notin I \quad \vdash \varphi : \mathsf{PG}(x)^+}{\vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{PG}(x)^+}\ \textsc{SincePGL}^+ \qquad \frac{\vdash \psi : \mathsf{PG}(x)^+}{\vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{PG}(x)^+}\ \textsc{SincePGR}^+$$

$$\frac{0 \notin I \quad \vdash \varphi : \mathsf{PG}(x)^+}{\vdash \varphi\, \mathsf{U}_I\, \psi : \mathsf{PG}(x)^+}\ \textsc{UntilPGL}^+ \qquad \frac{\vdash \varphi : \mathsf{PG}(x)^+ \quad \vdash \psi : \mathsf{PG}(x)^+}{\vdash \varphi\, \mathsf{U}_I\, \psi : \mathsf{PG}(x)^+}\ \textsc{UntilPGLR}^+$$

$$\frac{\vdash \varphi : \mathsf{PG}(x)^- \quad \vdash \psi : \mathsf{PG}(x)^-}{\vdash \varphi \wedge \psi : \mathsf{PG}(x)^-}\ \textsc{AndPG}^- \qquad \frac{0 \in I \quad \vdash \psi : \mathsf{PG}(x)^-}{\vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{PG}(x)^-}\ \textsc{SincePG}^-$$

$$\frac{0 \in I \quad \vdash \psi : \mathsf{PG}(x)^-}{\vdash \varphi\, \mathsf{U}_I\, \psi : \mathsf{PG}(x)^-}\ \textsc{UntilPG}^-$$

Typing of formulae as causable/suppressable

$$\frac{}{\Gamma \vdash \top : \mathsf{Cau}}\ \textsc{True} \qquad \frac{}{\Gamma \vdash \bot : \mathsf{Sup}}\ \textsc{False}$$

$$\frac{e \in \mathsf{Cau} \quad \Gamma(e) = \mathsf{Cau}}{\Gamma \vdash e(t_1,\dots,t_k) : \mathsf{Cau}}\ \textsc{Cau} \qquad \frac{e \in \mathsf{Sup} \quad \Gamma(e) = \mathsf{Sup}}{\Gamma \vdash e(t_1,\dots,t_k) : \mathsf{Sup}}\ \textsc{Sup}$$

$$\frac{\Gamma \vdash \varphi : \mathsf{Sup}}{\Gamma \vdash \neg\varphi : \mathsf{Cau}}\ \textsc{NotCau} \qquad \frac{\Gamma \vdash \varphi : \mathsf{Cau}}{\Gamma \vdash \neg\varphi : \mathsf{Sup}}\ \textsc{NotSup}$$

$$\frac{\Gamma \vdash \varphi : \mathsf{Cau}}{\Gamma \vdash \exists x.\ \varphi : \mathsf{Cau}}\ \textsc{ExistsCau} \qquad \frac{\Gamma \vdash \varphi : \mathsf{Sup} \quad \vdash \varphi : \mathsf{PG}(x)^+}{\Gamma \vdash \exists x.\ \varphi : \mathsf{Sup}}\ \textsc{ExistsSup}$$

$$\frac{\Gamma \vdash \varphi : \mathsf{Cau} \quad \Gamma \vdash \psi : \mathsf{Cau}}{\Gamma \vdash \varphi \wedge \psi : \mathsf{Cau}}\ \textsc{AndCau} \qquad \frac{\Gamma \vdash \varphi : \mathsf{Sup}}{\Gamma \vdash \varphi \wedge \psi : \mathsf{Sup}}\ \textsc{AndSupL}$$

$$\frac{\Gamma \vdash \psi : \mathsf{Sup}}{\Gamma \vdash \varphi \wedge \psi : \mathsf{Sup}}\ \textsc{AndSupR} \qquad \frac{\Gamma \vdash \varphi : \mathsf{Cau} \quad b > 0}{\Gamma \vdash \bigcirc_{[0,b)}\, \varphi : \mathsf{Cau}}\ \textsc{NextCau} \qquad \frac{\Gamma \vdash \varphi : \mathsf{Sup}}{\Gamma \vdash \bigcirc_I\, \varphi : \mathsf{Sup}}\ \textsc{NextSup}$$

$$\frac{0 \in I \quad \Gamma \vdash \psi : \mathsf{Cau}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{Cau}}\ \textsc{SinceCau} \qquad \frac{0 \notin I \quad \Gamma \vdash \varphi : \mathsf{Sup}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{Sup}}\ \textsc{SinceSupL}$$

$$\frac{0 \in I \quad \Gamma \vdash \varphi : \mathsf{Sup} \quad \Gamma \vdash \psi : \mathsf{Sup}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{Sup}}\ \textsc{SinceSupLR} \qquad \frac{b \neq \infty \quad \Gamma \vdash \psi : \mathsf{Cau}}{\Gamma \vdash \varphi\, \mathsf{U}_{[0,b]}\, \psi : \mathsf{Cau}}\ \textsc{UntilCauR}$$

$$\frac{b \neq \infty \quad \Gamma \vdash \varphi : \mathsf{Cau} \quad \Gamma \vdash \psi : \mathsf{Cau}}{\Gamma \vdash \varphi\, \mathsf{U}_{[a,b]}\, \psi : \mathsf{Cau}}\ \textsc{UntilCauLR} \qquad \frac{\Gamma \vdash \psi : \mathsf{Sup}}{\Gamma \vdash \varphi\, \mathsf{U}_I\, \psi : \mathsf{Sup}}\ \textsc{UntilSup}$$

Fig. 3. Typing rules for EMFOTL

satisfying φ' may only be discovered *strictly after* i. When this occurs, it may not be possible to decide which $\varphi[x/v]$ to suppress at time-point i. Our fragment rules out this situation by requiring that x be *past-guarded* in φ, i.e., that any value of x that satisfies φ true 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* $\forall v, i.\ v, i \vDash \varphi \wedge x \in \operatorname{dom} v \implies v(x) \in \mathsf{AD}_i(\varphi)$.

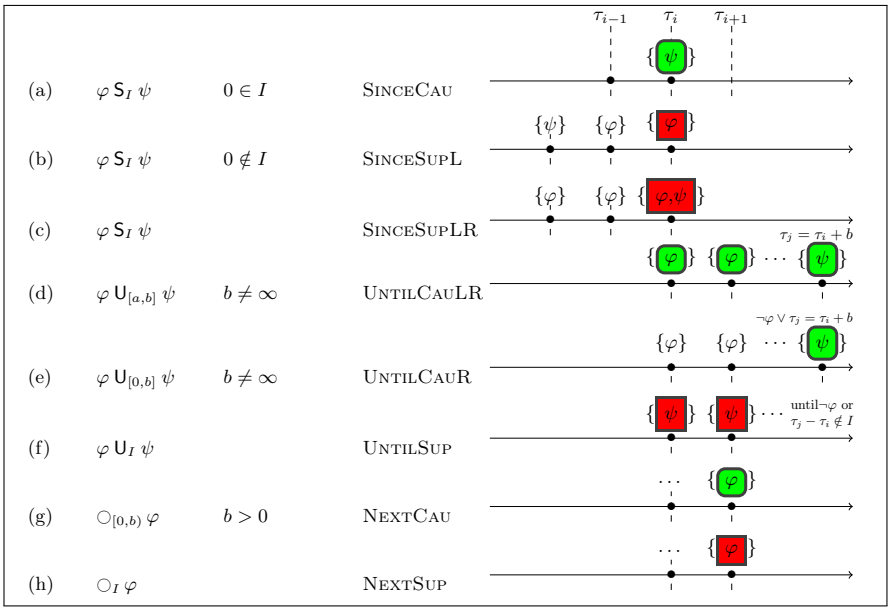

Fig. 4. Enforcement for temporal operators: φ = cause φ and φ = suppress φ

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 Appendix B, we prove

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

Since (Rules SinceCau, SinceSupL, SinceSupLR). 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 depending on whether $0 \in I$. If $0 \notin I$, then to suppress φ', it is enough 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 UntilCauL, UntilCauLR, UntilSup). 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 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). On the other hand, 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 NextCau, NextSup). 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 needs to 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 NextCau and use the previous definition of UntilCau. 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. [35]. 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.

The formulae φ_{law} and φ_{del} are in EMFOTL (see typing in Appendix A).

5 Enforcing EMFOTL

We now describe our enforcement algorithm. First, we present the enforcer's state space, which consists of a set of *obligations* (Section 5.1). We then explain how Lima et al.'s monitoring algorithm [43] 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 show 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, called *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 $\xi : \mathbb{N} \to$ 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

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

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

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

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

Fig. 6. Additional proof rules

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 proactive 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. [43], 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, $\Box\varphi$ evaluates to true at i *assuming* that fo is satisfied at $i+1$. Since the enforcer will suppress all time-points not containing TP, future time-points can be assumed to all contain TP.

Let $\{\mathsf{Cau}\}_+ := \mathsf{Cau}$, $\{\mathsf{Cau}\}_- := \mathsf{Sup}$ and $\overline{\sigma}^{\mathtt{TP}} = \langle(\tau_i, D_i \cup \{\mathtt{TP}\})\rangle_{i\in\mathbb{N}}$ for $\sigma = \langle(\tau_i, D_i)\rangle_{i\in\mathbb{N}}$. Consider $\varphi \in$ EMFOTL, and obtain Γ such that $\Gamma \vdash \varphi : \mathsf{Cau}$. Our satisfiability checker under assumptions is a function $\textsc{Sat} : (\mathbb{V} \to \mathbb{D}) \times \text{MFOTL} \times \mathbb{T}_f \times \mathcal{P}(\mathsf{FO}) \to \{\top, \bot\}$ such that, for any $p \in \{+, -\}$ and φ such that $\Gamma \vdash \varphi : \{\mathsf{Cau}\}_p$ and $X \subseteq \mathsf{FO}$, $\textsc{Sat}(v, \varphi, \sigma', X)$ implies

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

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 [43] with the proof rules in Figure 6. In Appendix B, we show:

Lemma 2. *The proof system of [43] extended with the rules from Figure 6 yields a decision procedure* SAT *that satisfies (⋆).*

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 (⋆) 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 $s_\varphi = \{(\mathsf{fo}_{\mathsf{init},\varphi}, \emptyset, +)\}$.

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 enf is parameterized by the current timestamp ts and a boolean b. The boolean b 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', a reactive command $\mathsf{ReCom}(D_C, D_S)$ 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 proactive command $\mathsf{PrCom}(D_C)$ 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 Cau (resp. Sup). The algorithm implements the approach described in Section 4. For space reasons, we only explain the more complex cases: $\varphi = \varphi_1 \wedge^{\textsc{AndCau}} \varphi_2$, $\varphi = \exists^{\textsc{ExistsSup}} x.\ \varphi_1$, and $\varphi = \varphi_1\ \mathsf{U}_I^{\textsc{UntilCauLR}}\ \varphi_2$.

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 [35]. Consider, e.g., the EMFOTL formula $\varphi = \psi \wedge (\psi \rightarrow \chi)$, where ψ and χ both type to Cau. If neither ψ nor χ is true, then the right conjunct of φ is true; however, to make the left conjunct true, formula ψ must be caused. But after causing ψ, the right conjunct becomes false, 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 Appendix B, we prove that this fixed-point computation terminates. (Theorem 11).

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 Appendix B.

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 $\mathtt{TP}$.

A detailed execution of the enforcement algorithm on the example traces σ_1 and σ_2 and formulae φ_{law} and φ_{del} is described in Appendix A.

5.4 Correctness

In this section, let φ be a closed formula to be enforced. The proofs of all lemmata are given in Appendix B. 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 Cau. In this case, enforcement is performed greedily: if the monitor cannot construct a proof of $\Diamond\, \chi$ (which occurs whenever χ cannot be proven true in the present), then ψ is caused. Thus our algorithm actually enforces a stronger formula, which we denote by $[\psi \vee \Diamond\, \chi]_+ \equiv \neg(\neg\psi \wedge^{R\omega} \neg(\top \,\mathsf{U}\, \chi))$, where $\wedge^{R\omega}$ has the semantics

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

The formula $[\psi \vee \Diamond\, \chi]_+$, unlike $\psi \vee \Diamond\, \chi$, is safety. In Appendix B, we define a similar transformation $[\bullet]_p$, $p \in \{+, -\}$ for all operators and prove

Lemma 4. *For any φ such that $\Gamma \vdash \varphi : \{\mathsf{Cau}\}_p$, 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., formula χ must not depend on the future in $\psi \wedge^{\textsc{SupAndL}} \chi$), one can obtain an EMFOTL fragment for which $[\varphi]_+ = \varphi$ and the enforcer $\mathcal{E}_\varphi$ is transparent (Appendix C).

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**
    $(\mathsf{ReCom}(C \setminus \{\mathtt{TP}\}, S), X')$
  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 $(\mathsf{PrCom}(D_C \setminus \{\mathtt{TP}\}), X')$ else (NoCom, X)
  end if
end function

function $\mathsf{enf}^+_{ts,b}(\varphi, \sigma, X, v)$
  if $\varphi = \top^{\textsc{True}}$ **then**
    $(\emptyset, \emptyset, \emptyset)$
  else if $\varphi = p(\bar{t})$ **then**
    $(\{(p, (\llbracket \bar{t} \rrbracket_v))\}, \emptyset, \emptyset)$
  else if $\varphi = \neg^{\textsc{NotCau}}\varphi_1$ **then**
    $\mathsf{enf}^-_{ts,b}(\varphi_1, \sigma, X, v)$
  else if $\varphi = \varphi_1 \wedge^{\textsc{AndCau}} \varphi_2$ **then**
    $\mathsf{fp}(\sigma, X, \mathsf{enf}^+_{\mathsf{and},\varphi_1,\varphi_2,v,ts})$
  else if $\varphi = \exists^{\textsc{ExistsCau}} x.\ \varphi_1$ **then**
    $\mathsf{enf}^+_{ts,b}(\varphi_1, \sigma, X, v[0/x])$
  else if $\varphi = \bigcirc_I^{\textsc{NextCau}} \varphi_1$ **then**
    $(\emptyset, \emptyset, \{(\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1}, v, +)\})$
  else if $\varphi = \varphi_1\ \mathsf{S}_I^{\textsc{SinceCau}}\ \varphi_2$ **then**
    $\mathsf{enf}^+_{ts,b}(\varphi_2, \sigma, X, v)$
  else if $\varphi = \varphi_1\ \mathsf{U}_I^{\textsc{UntilCauLR}}\ \varphi_2$ **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 $\varphi = \varphi_1\ \mathsf{U}_I^{\textsc{UntilCauR}}\ \varphi_2$ **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 $\varphi = \bot^{\textsc{False}}$ **then**
    $(\emptyset, \emptyset, \emptyset)$
  else if $\varphi = p(\bar{t})$ **then**
    $(\emptyset, \{(p, (\llbracket \bar{t} \rrbracket_v))\}, \emptyset)$
  else if $\varphi = \neg^{\textsc{NotSup}}\varphi_1$ **then**
    $\mathsf{enf}^+_{ts,b}(\varphi_1, \sigma, X, v)$
  else if $\varphi = \varphi_1 \wedge^{\textsc{AndSupL}} \varphi_2$ **then**
    $\mathsf{enf}^-_{ts,b}(\varphi_1, \sigma, X, v)$
  else if $\varphi = \varphi_1 \wedge^{\textsc{AndSupR}} \varphi_2$ **then**
    $\mathsf{enf}^-_{ts,b}(\varphi_2, \sigma, X, v)$
  else if $\varphi = \exists^{\textsc{ExistsSup}} x.\ \varphi_1$ **then**
    $\mathsf{fp}(\sigma, X, \mathsf{enf}^-_{\mathsf{ex},\varphi_1,v,ts,b})$
  else if $\varphi = \bigcirc_I^{\textsc{NextSup}} \varphi_1$ **then**
    $(\emptyset, \emptyset, \{(\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1}, v, -)\})$
  else if $\varphi = \varphi_1 \mathsf{S}_I^{\textsc{SinceSupL}} \varphi_2$ **then**
    $\mathsf{enf}^-_{ts,b}(\varphi_1, \sigma, X, v)$
  else if $\varphi = \varphi_1 \mathsf{S}_I^{\textsc{SinceSupR}} \varphi_2$ **then**
    let $\varphi' =$
    $\neg(\varphi_1 \wedge^{\textsc{AndSupL}} (\varphi_1\ \mathsf{S}_I\ \varphi_2))$ **in**
    $\mathsf{fp}(\sigma, X, \mathsf{enf}^+_{\mathsf{and},\varphi',\neg\varphi_2,v,ts,b})$
  else if $\varphi = \varphi_1 \mathsf{U}_I^{\textsc{UntilSup}} \varphi_2$ **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

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

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

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 [43], 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, 37]?

RQ2. At what maximum event rate can WhyEnf perform real-time enforcement?

RQ3. Do WhyEnf's performance and capabilities improve the state of the art?

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

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 7 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 [37]: 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, 37], 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 minimiza-*

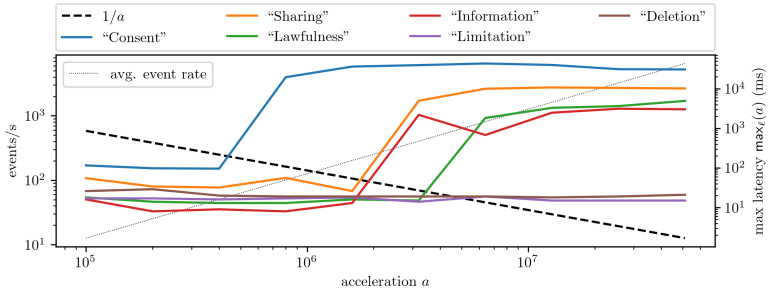

Fig. 8. RQ2: Maximum latency of WhyEnf and event rate for the formulae in Figure 7.

tion, 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 7 (i.e., all but φ_{min}). As we do not have access to the SuS, we simulate online enforcement by reproducing [39] the events from the above trace to WhyEnf at the speed specified by the trace's timestamps. We consider different accelerations of the original trace's real-time behavior to challenge WhyEnf. We 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 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 average processing time (avg_t), and 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) can be 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 computer 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 8. We include similar plots for WhyMon* and EnfPoly and latency profiles for some individual runs in Appendix D.

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 enforcement of the same trace can in fact be performed for much shorter real-time 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

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

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 [35], 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 10, imposing a 10-minute timeout (t.o.).

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 9). 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 10).

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 the computation of verdicts more costly. The same behavior is observed in terms of average processing time (Figure 10).

7 Related Work

Security automata [49, 26] were first used for enforcement by terminating the SuS. Bauer et al. [21] support enforcers that can cause and suppress events, as do Ligatti et al. [41], who used edit automata. 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 [47, 27] have also been proposed.

Most runtime enforcement approaches (and tools [28, 29]) rely on automata as

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

policies. Metric interval temporal logic formulae can be enforced via translation to timed automata [46, 48]. Basin et al. [11, 12] use dynamic condition response graphs [32] to formalize and enforce obligations in real time by suppressing and (proactively) causing events. Finally, controller synthesis tools for LTL [38, 25, 51], Timed CTL [22, 45], or MTL [40, 34] can generate enforcement mechanisms.

To the best of our knowledge, only two approaches enforce *first-order temporal* policies. Hublet et al. [35, 37] provide 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 still do not support time constraints. They also focus on instrumentation scenarios where all events are suppressable.

Many runtime monitoring tools support (different fragments of) MFOTL [23], including MonPoly [13, 17–19], VeriMon [9, 10, 50] and DejaVu [31]. Lima et al. [42] recently introduced the Explanator2, an MTL monitor that outputs explanations. They later extended their work to MFOTL with the WhyMon tool [43], on which our enforcer relies. WhyMon supports a larger fragment of MFOTL than in earlier works, as a consequence of using partitioned decision trees (PDTs) 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. The 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 [44], let bindings [52], and aggregations [16] would further improve our enforcer's expressiveness. Finally, refinements of the type system in which the same event can be both caused and suppressed in different contexts would be a useful addition.

References

1. Anonymized repository of WhyEnf (2024), https://github.com/or9ysbcrjp/whyenf
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 Transactions 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 in system design **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 Transactions 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. Gomaa, H.: Software modeling and design: UML, use cases, patterns, and software architectures. Cambridge University Press (2011)
31. Havelund, K., Peled, D., Ulus, D.: First-order temporal logic monitoring with bdds. Formal Methods in System Design **56**(1-3), 1–21 (2020)
32. 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)
33. 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)
34. 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)
35. 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)
36. Hublet, F., Basin, D., Krstić, S.: Real-time policy enforcement with metric first-order temporal logic (2022), extended version available at https://hblt.eu/static/sci/pdf/2022_esorics_ext.pdf

37. 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)
38. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: International Conference Formal Methods in Computer-Aided Design (FMCAD). pp. 117–124. IEEE (2006)
39. 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)
40. 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)
41. Ligatti, J., Bauer, L., Walker, D.: Edit automata: Enforcement mechanisms for run-time security policies. International Journal of Information Security **4**, 2–16 (2005)
42. 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)
43. Lima, L., Huerta y Munive, J., Traytel, D.: Explainable online monitoring of metric first-order temporal logic. In: TACAS (2024), to appear
44. 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)
45. 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)
46. 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)
47. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A., Nguena Timo, O.: Runtime enforcement of timed properties revisited. Formal Methods in System Design **45**, 381–422 (2014)
48. 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)
49. Schneider, F.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
50. 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, (RV). LNCS, vol. 11757, pp. 310–328. Springer (2019)
51. 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)
52. 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)

A Further examples

A.1 WhyMon (Section 2)

WhyMon's algorithm [43] shows that σ_2 violates φ_{del} (without $\Box$) at time-point 0 by constructing the proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{deletion\_request}(2,1,1) \in D_0}{\{c \mapsto 2, d \mapsto 1, u \mapsto 1\}, 0 \vdash^+ \mathsf{deletion\_request}(c,d,u)}\, p^+ \quad P}{\{c \mapsto 2, d \mapsto 1, u \mapsto 1\}, 0 \vdash^- \mathsf{deletion\_request}(c,d,u) \to \Diamond_{[0,30]}\, \mathsf{delete}(c,d,u)}\, \to^-}{\{c \mapsto 2, d \mapsto 1\}, 0 \vdash^- \forall u.\, \mathsf{deletion\_request}(c,d,u) \to \Diamond_{[0,30]}\, \mathsf{delete}(c,d,u)}\, \forall^-}{\{c \mapsto 2\}, 0 \vdash^- \forall d, u.\, \mathsf{deletion\_request}(c,d,u) \to \Diamond_{[0,30]}\, \mathsf{delete}(c,d,u)}\, \forall^-}{\emptyset, 0 \vdash^- \forall c, d, u.\, \mathsf{deletion\_request}(c,d,u) \to \Diamond_{[0,30]}\, \mathsf{delete}(c,d,u)}\, \forall^-$$

where P corresponds to the subproof

$$\dfrac{\dfrac{\mathsf{delete}(2,1,1) \notin D_0}{\{c \mapsto 2, d \mapsto 1, u \mapsto 1\}, 0 \vdash^- \mathsf{delete}(c,d,u)}\, p^-}{\{c \mapsto 2, d \mapsto 1, u \mapsto 1\}, 0 \vdash^- \Diamond_{[0,30]}\, \mathsf{delete}(c,d,u)}\, \Diamond^-$$

A.2 EMFOTL (Section 4)

For $\forall$, $\blacklozenge$, $\to$, $\Box$, and $\Diamond$, we can straightforwardly derive from the rules in Figure 3 the following simpler rules:

$$\dfrac{x \neq z \quad \vdash \varphi : \mathsf{PG}(z)^-}{\vdash \forall x.\, \varphi : \mathsf{PG}(z)^-}\, \textsc{ForallPG}^- \qquad \dfrac{\vdash \varphi : \mathsf{PG}(x)^+}{\vdash \blacklozenge \varphi : \mathsf{PG}(x)^+}\, \textsc{OncePG}^+ \qquad \dfrac{\vdash \varphi : \mathsf{PG}(x)^+}{\vdash \varphi \to \psi : \mathsf{PG}(x)^-}\, \textsc{ImpPGL}^-$$

$$\dfrac{\vdash \psi : \mathsf{PG}(x)^-}{\vdash \varphi \to \psi : \mathsf{PG}(x)^-}\, \textsc{ImpPGR}^- \qquad \dfrac{\vdash \varphi : \mathsf{PG}(x)^- \quad \Gamma \vdash \varphi : \mathsf{Cau}}{\Gamma \vdash \forall x.\, \varphi : \mathsf{Cau}}\, \textsc{ForallCau} \qquad \dfrac{\vdash \varphi : \mathsf{Sup}}{\Gamma \vdash \varphi \to \psi : \mathsf{Cau}}\, \textsc{ImpCauL}$$

$$\dfrac{\vdash \psi : \mathsf{Cau}}{\Gamma \vdash \varphi \to \psi : \mathsf{Cau}}\, \textsc{ImpCauR} \qquad \dfrac{b \neq \infty \quad \Gamma \vdash \varphi : \mathsf{Cau}}{\Gamma \vdash \Box_{[a,b]}\, \varphi : \mathsf{Cau}}\, \textsc{AlwaysCau} \qquad \dfrac{\Gamma \vdash \varphi : \mathsf{Cau}}{\Gamma \vdash \Diamond_I\, \varphi : \mathsf{Cau}}\, \textsc{EventuallyCau}$$

Further, let

$$\varphi_{11} = \forall c.\, \varphi_{12} \qquad \varphi_{12} = \forall d.\, \varphi_{13} \qquad \varphi_{13} = \forall u.\, \varphi_{14}$$
$$\varphi_{14} = \mathsf{use}(c,d,u) \to \blacklozenge(\mathsf{consent}(u,c) \vee \mathsf{legal\_ground}(u,u)).$$

With these, we prove that φ_{law} types to Cau as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{use} \in \mathsf{Sup}}{\{\mathsf{use} \mapsto \mathsf{Sup}\} \vdash \mathsf{use}(c,d,u) : \mathsf{Sup}}\, \textsc{Sup} \quad P_1}{\{\mathsf{use} \mapsto \mathsf{Sup}\} \vdash \varphi_{14} : \mathsf{Cau}}\, \textsc{ImpCauL}}{\{\mathsf{use} \mapsto \mathsf{Sup}\} \vdash \varphi_{13} \equiv \forall d.\, \varphi_{14} : \mathsf{Cau}}\, \textsc{ForallCau} \quad P_2}{\{\mathsf{use} \mapsto \mathsf{Sup}\} \vdash \varphi_{12} \equiv \forall d.\, \varphi_{13} : \mathsf{Cau}}\, \textsc{ForallCau} \quad P_3}{\{\mathsf{use} \mapsto \mathsf{Sup}\} \vdash \varphi_{11} \equiv \forall c.\, \varphi_{12} : \mathsf{Cau}}\, \textsc{ForallCau}}{\{\mathsf{use} \mapsto \mathsf{Sup}\} \vdash \varphi_{\mathsf{law}} \equiv \Box\, \varphi_{11} : \mathsf{Cau}}\, \textsc{AlwaysCau}$$

where $P_{1,2,3}$ stand for the subproofs

$$\dfrac{\dfrac{}{\vdash \mathsf{use}(c,d,u) : \mathsf{PG}(c)^+}\ \mathrm{PG}^+}{\vdash \varphi_{14} : \mathsf{PG}(c)^-}\ \textsc{ImpPGL}^-, \quad \dfrac{d \neq u \qquad \dfrac{\dfrac{}{\vdash \mathsf{use}(c,d,u) : \mathsf{PG}(d)^+}\ \mathrm{PG}^+}{\vdash \varphi_{14} : \mathsf{PG}(d)^-}\ \textsc{ImpPGL}^-}{\vdash \varphi_{13} \equiv \forall u.\, \varphi_{14} : \mathsf{PG}(d)^-}\ \textsc{ForallPG}^-,$$

$$\text{and} \quad \dfrac{c \neq d \quad \dfrac{c \neq u \quad \dfrac{\dfrac{}{\vdash \mathsf{use}(c,d,u) : \mathsf{PG}(c)^+}\ \mathrm{PG}^+}{\vdash \varphi_{14} : \mathsf{PG}(c)^-}\ \textsc{ImpPGL}^-}{\vdash \varphi_{13} \equiv \forall u.\, \varphi_{14} : \mathsf{PG}(c)^-}\ \textsc{ForallPG}^-}{\vdash \varphi_{12} \equiv \forall d.\, \varphi_{13} : \mathsf{PG}(c)^-}\ \textsc{ForallPG}^-,$$

respectively. Similarly, with

$$\varphi_{21} = \forall c.\, \varphi_{22} \qquad \varphi_{22} = \forall d.\, \varphi_{23} \qquad \varphi_{23} = \forall u.\, \varphi_{24}$$
$$\varphi_{24} = \mathsf{deletion\_request}(c,d,u) \rightarrow \Diamond_{[0,30]}\, \mathsf{delete}(c,d,u),$$

we prove that φ_{del} types to Cau as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{delete} \in \mathsf{Cau}}\ \textsc{Cau}}{\{\mathsf{delete} \mapsto \mathsf{Cau}\} \vdash \mathsf{delete}(c,d,u) : \mathsf{Cau}}\ \textsc{EventuallyCau}}{\{\mathsf{delete} \mapsto \mathsf{Cau}\} \vdash \Diamond_{[0,30]}\, \mathsf{delete}(c,d,u) : \mathsf{Cau}}\ \textsc{ImpCauR} \qquad P_4}{\{\mathsf{delete} \mapsto \mathsf{Cau}\} \vdash \varphi_{24} : \mathsf{Cau}}\ \textsc{ForallCau} \qquad P_5}{\{\mathsf{delete} \mapsto \mathsf{Cau}\} \vdash \varphi_{23} \equiv \forall d.\, \varphi_{24} : \mathsf{Cau}}\ \textsc{ForallCau} \qquad P_6}{\{\mathsf{delete} \mapsto \mathsf{Cau}\} \vdash \varphi_{22} \equiv \forall d.\, \varphi_{23} : \mathsf{Cau}}\ \textsc{ForallCau}$$
$$\dfrac{\{\mathsf{delete} \mapsto \mathsf{Cau}\} \vdash \varphi_{21} \equiv \forall c.\, \varphi_{22} : \mathsf{Cau}}{\{\mathsf{delete} \mapsto \mathsf{Cau}\} \vdash \varphi_{\mathsf{del}} \equiv \Box\, \varphi_{21} : \mathsf{Cau}}\ \textsc{AlwaysCau}$$

where $P_{4,5,6}$ stand for the subproofs

$$\dfrac{\dfrac{}{\vdash \mathsf{delete}(c,d,u) : \mathsf{PG}(c)^+}\ \mathrm{PG}^+}{\vdash \varphi_{24} : \mathsf{PG}(c)^-}\ \textsc{ImpPGL}^-, \quad \dfrac{d \neq u \qquad \dfrac{\dfrac{}{\vdash \mathsf{delete}(c,d,u) : \mathsf{PG}(d)^+}\ \mathrm{PG}^+}{\vdash \varphi_{24} : \mathsf{PG}(d)^-}\ \textsc{ImpPGL}^-}{\vdash \varphi_{23} \equiv \forall u.\, \varphi_{24} : \mathsf{PG}(d)^-}\ \textsc{ForallPG}^-,$$

$$\text{and} \quad \dfrac{c \neq d \quad \dfrac{c \neq u \quad \dfrac{\dfrac{}{\vdash \mathsf{delete}(c,d,u) : \mathsf{PG}(c)^+}\ \mathrm{PG}^+}{\vdash \varphi_{24} : \mathsf{PG}(c)^-}\ \textsc{ImpPGL}^-}{\vdash \varphi_{23} \equiv \forall u.\, \varphi_{24} : \mathsf{PG}(c)^-}\ \textsc{ForallPG}^-}{\vdash \varphi_{22} \equiv \forall d.\, \varphi_{23} : \mathsf{PG}(c)^-}\ \textsc{ForallPG}^-,$$

respectively.

A.3 Enforcement of φ_{law} and φ_{del} over σ_1 and σ_2 (Section 5.3)

Recall the following formulae:

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

– *Enforcing* φ_{law} *over* σ_1 (the trace is already compliant).
Let $fo_1 = (\lambda\_.\, \top\ \mathsf{U}\ \neg\varphi_{11}, \emptyset, -)$. The enforcement is as follows:

tp	ts	D_{tp}	b	X	Φ	C	S	X'	o	σ
0	10	$\{\mathsf{consent}(1,1),\mathsf{consent}(1,2)\}$	$\bot$	$\{(\lambda\_.\,\varphi_{\mathsf{law}},\emptyset,+)\}$	$\mathtt{TP}\wedge\varphi_{\mathsf{law}}$	$\{\mathtt{TP}\}$	$\emptyset$	$\{fo_1\}$	$\mathsf{ReCom}(\emptyset,\emptyset)$	$\langle(10,D_0)\rangle$
–	10	–	$\top$	$\{fo_1\}$	φ_{law}	$\emptyset$	$\emptyset$	$\{fo_1\}$	NoCom	$\langle(10,D_0)\rangle$
–	11	–	$\top$	$\{fo_1\}$	φ_{law}	$\emptyset$	$\emptyset$	$\{fo_1\}$	NoCom	$\langle(10,D_0)\rangle$
–	…	–	…	…	…	…	…	…	…	…
–	49	–	$\top$	$\{fo_1\}$	φ_{law}	$\emptyset$	$\emptyset$	$\{fo_1\}$	NoCom	$\langle(10,D_0)\rangle$
1	50	$\{\mathsf{use}(1,3,1),\mathsf{use}(2,1,1)\}$	$\bot$	$\{fo_1\}$	$\mathtt{TP}\wedge\varphi_{\mathsf{law}}$	$\{\mathtt{TP}\}$	$\emptyset$	$\{fo_1\}$	$\mathsf{ReCom}(\emptyset,\emptyset)$	$\langle(10,D_0),(50,D_1)\rangle$

At time-point 0, the only future obligation is $(\mathsf{fo}_{\mathsf{init},\varphi_{\mathsf{law}}},\emptyset,+)=(\lambda\_.\,\varphi_{\mathsf{law}},\emptyset,+)$, which is conjoined with $\mathtt{TP}$ as a time-point already exists. The goal formula $\Phi=\mathtt{TP}\wedge\varphi_{\mathsf{law}}$ is computed and passed to enf^+ with $ts=10$ and $b=\bot$. The function enf^+ first decomposes the goal into the present obligations $(\mathtt{TP},\emptyset,+)$ and $(\varphi_{\mathsf{law}},\emptyset,+)=(\Box\,\varphi_{11},\emptyset,+)$. It discharges the former by causing $\mathtt{TP}$, and unrolls the latter into the present obligation $(\varphi_{11},\emptyset,+)$ and the future obligation $fo_1=(\mathsf{fo}_{\mathsf{U},[0,\infty),\top,\neg\varphi_{11}},\emptyset,-)=(\lambda\_.\,\top\,\mathsf{U}\,\neg\varphi_{11},\emptyset,-)$ (recall that $\Box\,\varphi_{11}$ is syntactic sugar for $\neg(\top\,\mathsf{U}\,\neg\varphi_{11})$). The present obligation $(\varphi_{11},\emptyset,+)$ is already satisfied since no use event takes place in the present. Hence the enforcer returns $C=\{\mathtt{TP}\}$ and $X'=\{fo_1\}$, which emits the command $\mathsf{ReCom}(\emptyset,\emptyset)$. Next, as the next time-point has timestamp $50>0$, the enforcer processes the timestamps 10 to 49 'in the nick of time.' For each of these timestamps, the function computes $\Phi=\mathsf{fo}_{\mathsf{U},[0,\infty),\top,\neg\varphi_{11}}(ts)=\Box\,\varphi_{11}=\varphi_{\mathsf{law}}$ and calls enf^+ on Φ using ts and $b=\top$. It decomposes Φ into the present obligation $(\varphi_{11},\emptyset,+)$, which is immediately satisfied since no use takes place, and the future obligation fo_1, which is propagated to the next iteration with the command NoCom. At time-point 1 with timestamp 50, we have the same goal $\mathtt{TP}\wedge\varphi_{\mathsf{law}}$ as in iteration 0. Again the present obligation $(\varphi_{11},\emptyset,+)$ derived from φ_{law} is already satisfied, since every $\mathsf{use}(c,d,u)$ is matched by some $\mathsf{consent}(u,c)$ in the past ($\mathsf{use}(1,3,1)$ by $\mathsf{consent}(1,1)$; $\mathsf{use}(2,1,1)$ by $\mathsf{consent}(1,2)$). Hence $\mathsf{ReCom}(\emptyset,\emptyset)$ is again emitted, and the set of obligations $X'=\{fo_1\}$ is propagated. The trace, which was already compliant with φ_{law}, has not been modified.

– *Enforcing* φ_{del} *over* σ_1 (the trace is already compliant).

Let $fo_2=(\lambda\_.\,\top\,\mathsf{U}\,\neg\varphi_{21},\emptyset,-)$. The enforcement is as follows:

tp	ts	D_{tp}	b	X	Φ	C	S	X'	o	σ
0	10	$\{\mathsf{consent}(1,1),\mathsf{consent}(1,2)\}$	$\bot$	$\{(\lambda\_.\,\varphi_{\mathsf{del}},\emptyset,+)\}$	$\mathtt{TP}\wedge\varphi_{\mathsf{del}}$	$\{\mathtt{TP}\}$	$\emptyset$	$\{fo_2\}$	$\mathsf{ReCom}(\emptyset,\emptyset)$	$\langle(10,D_0)\rangle$
–	10	–	$\top$	$\{fo_2\}$	φ_{del}	$\emptyset$	$\emptyset$	$\{fo_2\}$	NoCom	$\langle(10,D_0)\rangle$
–	11	–	$\top$	$\{fo_2\}$	φ_{del}	$\emptyset$	$\emptyset$	$\{fo_2\}$	NoCom	$\langle(10,D_0)\rangle$
–	…	–	…	…	…	…	…	…	…	…
–	49	–	$\top$	$\{fo_2\}$	φ_{del}	$\emptyset$	$\emptyset$	$\{fo_2\}$	NoCom	$\langle(10,D_0)\rangle$
1	50	$\{\mathsf{use}(1,3,1),\mathsf{use}(2,1,1)\}$	$\bot$	$\{fo_2\}$	$\mathtt{TP}\wedge\varphi_{\mathsf{del}}$	$\{\mathtt{TP}\}$	$\emptyset$	$\{fo_2\}$	$\mathsf{ReCom}(\emptyset,\emptyset)$	$\langle(10,D_0),(50,D_1)\rangle$

Again, there are no violations. The execution is similar to the previous case, but this time satisfaction of φ_{11} is obtained at all time-points because no $\mathsf{deletion\_request}$ event is ever present in the trace.

– *Enforcing* φ_{law} *over* σ_2 (violation at time-point 1 : no prior consent given).

tp	ts	D_{tp}	b	X	Φ	C	S	X'	o	σ
0	10	$\{\mathsf{deletion\_request}(2,1,1)\}$	$\bot$	$\{(\lambda\_.\,\varphi_{\mathsf{law}},\emptyset,+)\}$	$\mathtt{TP}\wedge\varphi_{\mathsf{law}}$	$\{\mathtt{TP}\}$	$\emptyset$	$\{fo_1\}$	$\mathsf{ReCom}(\emptyset,\emptyset)$	$\langle(10,D_0)\rangle$
–	10	–	$\top$	$\{fo_1\}$	φ_{law}	$\emptyset$	$\emptyset$	$\{fo_1\}$	NoCom	$\langle(10,D_0)\rangle$
–	11	–	$\top$	$\{fo_1\}$	φ_{law}	$\emptyset$	$\emptyset$	$\{fo_1\}$	NoCom	$\langle(10,D_0)\rangle$
–	…	–	…	…	…	…	…	…	…	…
–	49	–	$\top$	$\{fo_1\}$	φ_{law}	$\emptyset$	$\emptyset$	$\{fo_1\}$	NoCom	$\langle(10,D_0)\rangle$
1	50	$\{\mathsf{use}(1,3,1)\}$	$\bot$	$\{fo_1\}$	$\mathtt{TP}\wedge\varphi_{\mathsf{law}}$	$\{\mathtt{TP}\}$	$\{\mathsf{use}(1,3,1)\}$	$\{fo_1\}$	$\mathsf{ReCom}(\emptyset,\{\mathsf{use}(1,3,1)\})$	$\langle(10,D_0),(50,\emptyset)\rangle$

The execution is similar to the enforcement of φ_{law} on σ_2 until time-point 1. There, the goal $\mathtt{TP}\wedge\varphi_{\mathsf{law}}$ is decomposed into the present obligations $(\mathtt{TP},\emptyset,+)$ and $(\varphi_{11},\emptyset,+)$ and the future obligation fo_1. But now φ_{11} is violated, since $\mathsf{use}(1,3,1)$ is not matched by any past $\mathsf{consent}(1,1)$ event. To recover satisfaction,

the use event needs to be suppressed. Note that the corresponding $\rightarrow$ operator is labeled with ImpSupL, thus guiding enf^p towards suppression of $\mathsf{use}(c,d,u)$ whenever the implication is false. Hence enf^+ returns $S = \{\mathsf{use}(1,3,1)\}$ together with $X' = \{fo_1\}$. This results in the command $\mathsf{ReCom}(\emptyset, \{\mathsf{use}(1,3,1)\})$ being emitted, and the second time-point in the enforced trace is $(50, \emptyset)$.

– *Enforcing* φ_{del} *over* σ_2 (violation at timestamp 40 : deletion missing).
Let $fo_3^{x,y} = (\lambda\tau'.\ \Diamond_{[0,x]-(\tau'-y)}\, \mathsf{TP} \wedge \mathsf{delete}(c,d,u), \{c \mapsto 2, d \mapsto 1, u \mapsto 1\}, +)$, $\sigma_1 = \langle (10, D_0), (30, \{\mathsf{delete}(2,1,1)\}) \rangle$, $\sigma_2 = \langle (10, D_0), (30, \{\mathsf{delete}(2,1,1)\}), (50, D_1) \rangle$

tp	ts	D_{tp}	b	X	Φ	C	S	X'	o	σ
0	10	{deletion_request(2, 1, 1)}	$\bot$	$\{(\lambda\_.\ \varphi_{\mathsf{del}}, \emptyset, +)\}$	$\mathsf{TP} \wedge \varphi_{\mathsf{del}}$	{TP}	$\emptyset$	$\{fo_2, fo_3^{30,10}\}$	ReCom($\emptyset, \emptyset$)	$\langle (10, D_0) \rangle$
–	10	–	$\top$	$\{fo_2, fo_3^{30,10}\}$	$\varphi_{\mathsf{del}} \wedge \Diamond_{[0,30]}\, \mathsf{delete}(2,1,1)$	$\emptyset$	$\emptyset$	$\{fo_2, fo_3^{30,10}\}$	NoCom	$\langle (10, D_0) \rangle$
–	11	–	$\top$	$\{fo_2, fo_3^{30,10}\}$	$\varphi_{\mathsf{del}} \wedge \Diamond_{[0,29]}\, \mathsf{delete}(2,1,1)$	$\emptyset$	$\emptyset$	$\{fo_2, fo_3^{29,11}\}$	NoCom	$\langle (10, D_0) \rangle$
–	…	–	…	…	…	…	…	…	…	…
–	39	–	$\top$	$\{fo_2, fo_3^{2,38}\}$	$\varphi_{\mathsf{del}} \wedge \Diamond_{[0,1]}\, \mathsf{delete}(2,1,1)$	$\emptyset$	$\emptyset$	$\{fo_2, fo_3^{1,39}\}$	NoCom	$\langle (10, D_0) \rangle$
–	40	–	$\top$	$\{fo_2, fo_3^{1,39}\}$	$\varphi_{\mathsf{del}} \wedge \Diamond_{[0,0]}\, \mathsf{delete}(2,1,1)$	{TP, delete(2, 1, 1)}	$\emptyset$	$\{fo_2\}$	PrCom({delete(2, 1, 1)})	σ_1
–	41	–	$\top$	$\{fo_2\}$	φ_{del}	$\emptyset$	$\emptyset$	$\{fo_2\}$	NoCom	σ_1
–	…	–	…	…	…	…	…	…	…	…
–	49	–	$\top$	$\{fo_2\}$	φ_{del}	$\emptyset$	$\emptyset$	$\{fo_2\}$	NoCom	σ_1
1	50	{use(1, 3, 1)}	$\bot$	$\{fo_2\}$	$\mathsf{TP} \wedge \varphi_{\mathsf{del}}$	{TP}	$\emptyset$	$\{fo_2\}$	ReCom($\emptyset, \emptyset$)	σ_2

At time-point 0, the function enf^+ first decomposes its goal $\mathsf{TP} \wedge \varphi_{\mathsf{del}}$ into the present obligations $(\mathsf{TP}, \emptyset, +)$ and $(\varphi_{\mathsf{del}}, \emptyset, +) = (\Box\, \varphi_{21}, \emptyset, +)$. It discharges the former by causing TP, and unrolls the latter into the present obligation $(\varphi_{21}, \emptyset, +)$ and the future obligation $fo_2 = (\mathsf{fo}_{\mathsf{U},[0,\infty),\top,\neg\varphi_{21}}, \emptyset, -) = (\lambda\_.\ \top\ \mathsf{U}\ \neg\varphi_{11}, \emptyset, -)$. The present obligation $(\varphi_{21}, \emptyset, +)$ is violated, since $\mathsf{deletion\_request}(2,1,1)$ is true but there is no corresponding delete in the future yet. The corresponding $\rightarrow$ operator is labeled with ImpCauR, leading enf^+ to generate the future obligation $fo_3^{30,10} = (\lambda\tau'.\ \Diamond_{[0,30]-(\tau'-10)}\, \mathsf{TP} \wedge \mathsf{delete}(c,d,u), \{c \mapsto 2, d \mapsto 1, u \mapsto 1\}, +)$. Satisfying this future obligation guarantees the satisfaction of Φ, hence the algorithm proceeds to the next iteration. Next, it processes timestamp 10 'in the nick of time.' The function enf^+ computes $\Phi = fo_2(10) \wedge fo_3^{30,10}(10) = \varphi_{\mathsf{del}} \wedge \Diamond_{[0,30]}(\mathsf{TP} \wedge \mathsf{delete}(2,1,1))$ and calls enf^+ on Φ using ts and $b = \top$. It first decomposes Φ into the present obligations $(\varphi_{21}, \emptyset, +)$ and $(\Diamond_{[0,30]}(\mathsf{TP} \wedge \mathsf{delete}(2,1,1)), \emptyset, +)$ and the future obligation fo_2. The first present obligation is immediately satisfied since no deletion_request takes place. The second present obligation features a top-level $\Diamond$ labeled EventuallyCau with a non-$[0,0]$ interval, and can thus be satisfied by satisfying the future obligation $(fo_3^{30,10}, \emptyset, +)$ at the next time-point. Hence the enforcer emits the order NoCom and propagates the future obligations $X' = \{fo_2, fo_3^{30,10}\}$ to the next time-point. In the next iteration, the timestamp 11 is processed 'in the nick of time.' The goal $\Phi = fo_2(30) \wedge fo_3^{30,10}(11) = \varphi_{\mathsf{del}} \wedge \Diamond_{[0,29]}(\mathsf{TP} \wedge \mathsf{delete}(2,1,1))$ is computed, and similarly reduced to the future obligations $X' = \{fo_2, fo_3^{29,11}\}$. Similar iterations repeat until timestamp 40, when the goal becomes $\Phi = fo_2(40) \wedge fo_3^{1,39}(40) = \varphi_{\mathsf{del}} \wedge \Diamond_{[0,0]}(\mathsf{TP} \wedge \mathsf{delete}(2,1,1))$, which contains a $[0,0]$ interval. Being called with $b = \top$ (in the nick of time), the enf^+ function enters the $\Diamond$, generating the present obligations $(\mathsf{TP}, \emptyset, +)$ and $(\mathsf{delete}(2,1,1), \emptyset, +)$ which it discharges by causing TP and $\mathsf{delete}(2,1,1)$, respectively. Hence $C = \{\mathsf{TP}, \mathsf{delete}(2,1,1)\}$ and the command $\mathsf{PrCom}(\{\mathsf{delete}(2,1,1)\})$ is emitted, leading to $(30, \{\mathsf{delete}(2,1,1)\})$ being inserted into the trace. The future obligations $X' = \{fo_2\}$ are propagated to the next timestamp. The rest of the execution is as with σ_1.

B Proofs of lemmata and theorems

B.1 Past-guarded fragment (Section 4)

We first observe that

Lemma 6. *Let $\varphi, \varphi' \in$ MFOTL and $i, i' \in \mathbb{N}$. Assume that $i' \leq i$ and φ' is a subformula of φ. Then $\mathsf{AD}_{i'}(\varphi') \subseteq \mathsf{AD}_i(\varphi)$.*

Proof. Since φ' is a subformula of φ, we have $\mathsf{cs}(\varphi') \subseteq \mathsf{cs}(\varphi)$. Hence

$$\begin{aligned}\mathsf{AD}_{i'}(\varphi') &\overset{\text{def.}}{=} \mathsf{cs}(\varphi') \cup \left(\bigcup\nolimits_{j \leq i'} \{d \mid d \text{ is one of } d_k \text{ in } e(d_1, \ldots, d_{a(e)}) \in D_j\}\right) \\ &\subseteq \mathsf{cs}(\varphi) \cup \left(\bigcup\nolimits_{j \leq i} \{d \mid d \text{ is one of } d_k \text{ in } e(d_1, \ldots, d_{a(e)}) \in D_j\}\right) \\ &\overset{\text{def.}}{=} \mathsf{AD}_i(\varphi).\end{aligned}$$

Lemma 1. *For $p \in \{+, -\}$, if $\vdash \varphi : \mathsf{PG}(x)^p$, then x is past-guarded in $p\varphi$, i.e., for any v, i such that if $v, i \vDash p\varphi$ and $x \in \operatorname{dom} v$, we have $v(x) \in \mathsf{AD}_i(\varphi)$.*

Proof. Fix x and σ. We prove

$$P(\varphi) \equiv \forall p, v, i.\ \vdash \varphi : \mathsf{PG}(x)^p \implies v, i \vDash p\varphi \implies x \in \operatorname{dom} v \implies v(x) \in \mathsf{AD}_i(\varphi)$$

by structural induction on φ.

- If $\varphi = e(t_1, \ldots, t_n)$, then $\vdash e(t_1, \ldots, t_n) : \mathsf{PG}(x)^p$ implies $p = +$. We obtain $1 \leq j \leq n$ such that $t_j = x$. Then $v, i \vDash \varphi$ implies $(r, (v(t_1), \ldots, v(t_k))) \in D_i$, and $v(x) = v(t_j) \in \mathsf{AD}_i(\varphi)$.
- If $\varphi = \neg\varphi'$ and $P(\varphi')$ holds, then $\vdash \varphi : \mathsf{PG}(x)^p$ yields $\vdash \varphi' : \mathsf{PG}(x)^{-p}$. Since $v, i \vDash p\neg\varphi' \iff v, i \vDash (-p)\varphi'$, the assumption $P(\varphi')$ instantiated with $-p$, v, and i immediately provides $v(x) \in \mathsf{AD}_i(\varphi') \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.
- If $\varphi = \exists z.\, \varphi'$ and $P(\varphi')$ holds, then $C \vdash \varphi : \mathsf{PG}(x)^p$ yields $\vdash \varphi' : \mathsf{PG}(x)^p$ and $x \neq z$. Since $v, i \vDash p(\exists z.\, \varphi') \iff \exists d.\, v[z \mapsto d], i \vDash p\varphi'$, we obtain d be such that $v[z \mapsto d], i \vDash p\varphi'$ holds. The assumption $P(\varphi')$ instantiated with p, $v[z \mapsto d]$, and i provides $v[z \mapsto d](x) \in \mathsf{AD}_i(\varphi')$. As $x \neq z$, we get $v(x) = v[z \mapsto d](x) \in \mathsf{AD}_i(\varphi') \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.
- If $\varphi = \varphi_1 \wedge \varphi_2$ and both $P(\varphi_1)$ and $P(\varphi_2)$ hold, there are three cases depending on which typing rule is used to derive $\vdash \varphi_1 \wedge \varphi_2 : \mathsf{PG}(x)^p$.
  - With AndPGL$^+$, we get $p = +$ and $\vdash \varphi_1 : \mathsf{PG}(x)^+$. The fact that $v, i \vDash \varphi_1 \wedge \varphi_2$ implies $v, i \vDash \varphi_1$ allows us to instantiate $P(\varphi_1)$ with p, v, and i to get $v(x) \in \mathsf{AD}_i(\varphi_1) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.
  - With AndPGR$^+$, the proof is similar using φ_2 instead of φ_1.
  - With AndPG$^-$, we get $p = -$, $\vdash \varphi_1 : \mathsf{PG}(x)^-$, and $\vdash \varphi_1 : \mathsf{PG}(x)^-$. Now, observe that $v, i \vDash \neg(\varphi_1 \wedge \varphi_2) \iff (v, i \vDash \neg\varphi_1$ or $v, i \vDash \neg\varphi_2)$. If $v, i \vDash \neg\varphi_1$ holds, we instantiate $P(\varphi_1)$ with $-$, v, and i to get $v(x) \in \mathsf{AD}_i(\varphi_1) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6. If $v, i \vDash \neg\varphi_2$ holds, we similarly instantiate $P(\varphi_2)$ to get $v(x) \in \mathsf{AD}_i(\varphi_2) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.

- If $\varphi = \varphi_1 \,\mathsf{S}_I\, \varphi_2$ and both $P(\varphi_1)$ and $P(\varphi_2)$ hold, there are again three cases depending on which typing rule is used to derive $\vdash \varphi_1 \,\mathsf{S}_I\, \varphi_2 : \mathsf{PG}(x)^p$.
  - With SincePGL$^+$, we get $p = +$, $0 \notin I$, and $\vdash \varphi_1 : \mathsf{PG}(x)^+$. Since $v, i \vDash \varphi_1 \,\mathsf{S}_I\, \varphi_2$, we obtain $i' \leq i$ such that $v, j \vDash \varphi_1$ for all $i' < j \leq i$ and $\tau_i - \tau'_i \in I$. Since $0 \notin I$, we have $i' < i$, hence $v, i \vDash \varphi_1$ holds. We can now instantiate $P(\varphi_1)$ with $+$, v, and i to get $v(x) \in \mathsf{AD}_i(\varphi_1) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.
  - With SincePGR$^+$, we get $p = +$ and $\vdash \varphi_2 : \mathsf{PG}(x)^+$. Since $v, i \vDash \varphi_1 \mathsf{S}_I \varphi_2$, we obtain $i' \leq i$ such that $v, i' \vDash \varphi_2$. We instantiate $P(\varphi_2)$ with $+$, v, and i' to get $v(x) \in \mathsf{AD}_{i'}(\varphi_1) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.
  - With SincePG$^-$, we get $p = -$, $0 \in I$, $\vdash \varphi_2 : \mathsf{PG}(x)^-$. As $0 \in I$, $v, i \vDash \neg(\varphi_1 \,\mathsf{S}_I\, \varphi_2)$ implies $v, i \vDash \neg\varphi_2$. We instantiate $P(\varphi_2)$ with $-$, v, and i to get $v(x) \in \mathsf{AD}_i(\varphi_2) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.
- If $\varphi = \varphi_1 \,\mathsf{U}_I\, \varphi_2$ and both $P(\varphi_1)$ and $P(\varphi_2)$ hold, there are again three cases depending on which typing rule is used to derive $\vdash \varphi_1 \,\mathsf{U}_I\, \varphi_2 : \mathsf{PG}(x)^p$.
  - With UntilPGL$^+$, we get $p = +$, $0 \notin I$, and $\vdash \varphi_1 : \mathsf{PG}(x)^+$. Since $v, i \vDash \varphi_1 \mathsf{U}_I \varphi_2$, we obtain $i' \geq i$ such that $v, j \vDash \varphi_1$ for all $i \leq j \leq i'$ and $\tau'_i - \tau_i \in I$. Since $0 \notin I$, we have $i' > i$, hence $v, i \vDash \varphi_1$ holds, and we can instantiate $P(\varphi_1)$ with $+$, v, and i to get $v(x) \in \mathsf{AD}_i(\varphi_1) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.
  - With UntilPGLR$^+$, we get $p = +$, $\vdash \varphi_1 : \mathsf{PG}(x)^+$, and $\vdash \varphi_2 : \mathsf{PG}(x)^+$. Since $v, i \vDash \varphi_1 \,\mathsf{U}_I\, \varphi_2$, we obtain $i' \geq i$ such that $v, i' \vDash \varphi_2$ and, for all, $i \leq j < i'$, $v, j \vDash \varphi_1$. If $i' > i$, we have $v, i \vDash \varphi_1$ and we conclude as in the previous case. Otherwise, $i' = i$ and $v, i \vDash \varphi_2$ holds. We can then instantiate $P(\varphi_2)$ with $+$, v, and i to get $v(x) \in \mathsf{AD}_i(\varphi_2) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.
  - With UntilPG$^-$, we get $p = -$, $0 \in I$, $\vdash \varphi_2 : \mathsf{PG}(x)^-$. As $0 \in I$, $v, i \vDash \neg(\varphi_1 \,\mathsf{U}_I\, \varphi_2)$ implies $v, i \vDash \neg\varphi_2$. We instantiate $P(\varphi_2)$ with $-$, v, and i to get $v(x) \in \mathsf{AD}_i(\varphi_2) \subseteq \mathsf{AD}_i(\varphi)$ using Lemma 6.

B.2 Helper operators and approximated EMFOTL formulae

We use the following helper operators:

$$
\begin{array}{ll}
v, i \vDash_\sigma \varphi \wedge^{L\omega} \psi & \text{iff } \exists \sigma'.\, v, i \vDash_{\sigma|_{..i}\cdot\sigma'} \varphi \text{ and } v, i \vDash_\sigma \psi \\
v, i \vDash_\sigma \varphi \wedge^{R\omega} \psi & \text{iff } v, i \vDash_\sigma \varphi \text{ and } \exists \sigma'.\, v, i \vDash_{\sigma|_{..i}\cdot\sigma'} \psi \\
v, i \vDash_\sigma \varphi \,\mathsf{S}_I^{L\omega-}\, \psi & \text{iff } \exists j \leq i.\ \exists \sigma'.\ v, j \vDash_{\sigma|_{..i}\cdot\sigma'} \psi \text{ with } \tau_i - \tau_j \in I \text{ and } \forall k \in [j+1 \ldots i].\ \exists \sigma'.\ v, k \vDash_{\sigma|_{..i}\cdot\sigma'} \varphi \\
v, i \vDash_\sigma \varphi \,\mathsf{S}_I^{L\omega+}\, \psi & \text{iff } \exists j \leq i.\ \forall \sigma'.\ v, j \vDash_{\sigma|_{..i}\cdot\sigma'} \psi \text{ with } \tau_i - \tau_j \in I \text{ and } \forall k \in [j+1 \ldots i], \sigma'.\ v, k \vDash_{\sigma|_{..i}\cdot\sigma'} \varphi \\
v, i \vDash_\sigma \varphi \,\mathsf{S}_I^{R\omega}\, \psi & \text{iff } \exists j \leq i.\ \exists \sigma'.\ v, j \vDash_{\sigma|_{..i}\cdot\sigma'} \psi \text{ with } \tau_i - \tau_j \in I \text{ and } \forall k \in [j+1 \ldots i].\ \exists \sigma'.\ v, k \vDash_{\sigma|_{..i}\cdot\sigma'} \varphi \\
v, i \vDash_\sigma \varphi \,\mathsf{U}_I^{L\omega-}\, \psi & \text{iff } \exists j \geq i.\ v, j \vDash \psi \text{ with } \tau_j - \tau_i \in I \text{ and } \forall k \in [i \ldots j-1].\ \exists \sigma'.\ v, k \vDash_{\sigma|_{..i}\cdot\sigma'} \varphi \\
v, i \vDash_\sigma \varphi \,\mathsf{U}_I^{L\omega+}\, \psi & \text{iff } \exists j \geq i.\ v, j \vDash \psi \text{ with } \tau_j - \tau_i \in I \text{ and } \forall k \in [i \ldots j-1], \sigma'.\ v, k \vDash_{\sigma|_{..i}\cdot\sigma'} \varphi
\end{array}
$$

Fig. 11. Semantics of helper operators

Consider the following transformation on (typed) MFOTL formulae:

$$
\begin{aligned}
[r(t_1,\ldots,t_n)]_p &= r(t_1,\ldots,t_n) & [\neg\varphi]_p &= \neg[\varphi]_{-p}\\
[\exists x.\ \varphi]_- &= \exists x.\ [\varphi]_- & [\exists x.\ \varphi]_+ &= [\varphi]_+[x/0]\\
[\varphi \wedge \psi]_+ &= [\varphi]_+ \wedge [\psi]_+ & [\varphi \wedge^{\textsc{AndSupL}} \psi]_- &= [\varphi]_- \wedge^{R\omega} \psi\\
[\varphi \wedge^{\textsc{AndSupR}} \psi]_- &= \varphi \wedge^{L\omega} [\psi]_- & [\bigcirc_I \varphi]_p &= \bigcirc_I [\varphi]_p\\
[\varphi\ \mathsf{S}_I\ \psi]_+ &= \varphi\ \mathsf{S}_I^{L\omega+}\ [\psi]_+ & [\varphi\ \mathsf{S}_I^{\textsc{SinceSupLR}}\ \psi]_- &= [\varphi]_-\ \mathsf{S}_I^{R\omega}\ \psi\\
[\varphi\ \mathsf{S}_I^{\textsc{SinceSupR}}\ \psi]_- &= \varphi\ \mathsf{S}_I^{L\omega-}\ [\psi]_- & [\varphi\ \mathsf{U}_I^{\textsc{UntilCauLR}}\ \psi]_+ &= [\varphi]_+\ \mathsf{U}_I\ [\psi]_+\\
[\varphi\ \mathsf{U}_I^{\textsc{UntilCauR}}\ \psi]_+ &= \varphi\ \mathsf{U}_I^{L\omega+}\ [\psi]_+ & [\varphi\ \mathsf{U}_I\ \psi]_- &= \varphi\ \mathsf{U}_I^{L\omega-}\ [\psi]_-
\end{aligned}
$$

The transformed formulae soundly approximate the original formulae:

Lemma 4. *For any φ such that $\Gamma \vdash \varphi : \{\mathsf{Cau}\}_p$, if $v, i \vDash p[\varphi]_p$ holds, then $v, i \vDash p\varphi$ holds. In particular, $\mathcal{L}([\varphi]_+) \subseteq \mathcal{L}(\varphi)$.*

Proof. By straightforward structural induction on the typing rules using the definition of $[\cdot]$, and observing that

$$
\begin{aligned}
\forall \sigma'.\ v, k \vDash_{\sigma|_{..i}\cdot\sigma'} \varphi &\implies v, k \vDash_\sigma \varphi\\
\neg\left(\exists \sigma'.\ v, k \vDash_{\sigma|_{..i}\cdot\sigma'} \varphi\right) &\implies v, k \vDash_\sigma \neg\varphi.
\end{aligned}
$$

B.3 Satisfaction-checking under assumptions (Section 5.2)

We inductively define the set of future obligations of a (typed) formula φ at time-point i and timestamp ts, denoted $\mathsf{FO}^+_{i,ts}(\varphi)$, as

$$
\begin{aligned}
\mathsf{FO}^+_{i,ts}(\varphi) = \{(fo, v, p) \mid{}& (\forall x \in \mathsf{fv}(\varphi).\, v(x) \in \mathsf{AD}_i(\varphi)) \wedge (\forall x \notin \mathsf{fv}(\varphi).\, v(x) = 0)\\
&\wedge (fo, p) \in \mathsf{FO}^+_{ts}(\varphi)\}
\end{aligned}
$$

where

$$
\begin{aligned}
\mathsf{FO}^p_{ts}(\neg\varphi_1) &= \mathsf{FO}^{-p}_{ts}(\varphi_1)\\
\mathsf{FO}^+_{ts}(\varphi_1 \wedge \varphi_2) &= \mathsf{FO}^+_{ts}(\varphi_1) \cup \mathsf{FO}^+_{ts}(\varphi_2)\\
\mathsf{FO}^-_{ts}(\varphi_1 \wedge^{\textsc{AndSupL}} \varphi_2) &= \mathsf{FO}^-_{ts}(\varphi_1)\\
\mathsf{FO}^-_{ts}(\varphi_1 \wedge^{\textsc{AndSupR}} \varphi_2) &= \mathsf{FO}^-_{ts}(\varphi_2)\\
\mathsf{FO}^p_{ts}(\exists x.\ \varphi_1) &= \mathsf{FO}^p_{ts}(\varphi_1)\\
\mathsf{FO}^p_{ts}(\bigcirc_I \varphi_1) &= \{(\mathsf{fo}_{ts,\bigcirc,I,\varphi_1}, p)\}\\
\mathsf{FO}^+_{ts}(\varphi_1\ \mathsf{S}_I\ \varphi_2) &= \mathsf{FO}^+_{ts}(\varphi_2)\\
\mathsf{FO}^-_{ts}(\varphi_1\ \mathsf{S}_I^{\textsc{SinceSupL}}\ \varphi_2) &= \mathsf{FO}^-_{ts}(\varphi_1)\\
\mathsf{FO}^-_{ts}(\varphi_1\ \mathsf{S}_I^{\textsc{SinceSupR}}\ \varphi_2) &= \mathsf{FO}^-_{ts}(\varphi_2)\\
\mathsf{FO}^+_{ts}(\varphi_1\ \mathsf{U}_I^{\textsc{UntilCauLR}}\ \varphi_2) &= \mathsf{FO}^+_{ts}(\varphi_1) \cup \mathsf{FO}^+_{ts}(\varphi_2) \cup \{(\mathsf{fo}_{ts,\mathsf{U},I,\varphi_1,\varphi_2}, +)\}\\
\mathsf{FO}^+_{ts}(\varphi_1\ \mathsf{U}_I^{\textsc{UntilCauR}}\ \varphi_2) &= \mathsf{FO}^-_{ts}(\varphi_2) \cup \{(\mathsf{fo}_{ts,\mathsf{U},I,\varphi_1,\varphi_2}, +)\}\\
\mathsf{FO}^-_{ts}(\varphi_1\ \mathsf{U}_I\ \varphi_2) &= \mathsf{FO}^-_{ts}(\varphi_2) \cup \{(\mathsf{fo}_{ts,\mathsf{U},I,\varphi_1,\varphi_2}, -)\}.
\end{aligned}
$$

Note that the above sets are always finite since $\mathsf{AD}_i(\varphi)$ and $\mathsf{fv}(\varphi)$ are finite, too. We prove

Lemma 2. *The proof system of [43] extended with the rules from Figure 6 yields a decision procedure* Sat *that satisfies (⋆).*

by showing the stronger lemma

Lemma 7. *The proof system of [43] extended with the rules from Figure 6 yields a decision procedure* Sat *that satisfies*

$$\begin{aligned}&\textsc{Sat}(v, \varphi, \sigma', X)\\ \overset{(\star\star)}{\Longrightarrow} \Big(&\forall ts \in \mathbb{N}, D \in \mathbb{DB}, \sigma'' \in \mathbb{T}_\omega.\ (\forall (\xi, v', p') \in X.\ v', |\sigma'| \vDash_{\sigma' \cdot \overline{(ts,D)\cdot\sigma''}^{\mathrm{TP}}} p'\xi(ts))\\ &\qquad \Longrightarrow v, |\sigma'| - 1 \vDash_{\sigma' \cdot \overline{(ts,D)\cdot\sigma''}^{\mathrm{TP}}} [\varphi]_+\Big).\end{aligned}$$

Proof. First, note that for all v', σ', ts, d, ts', D', σ'', p', τ, I, φ_1, and φ_2:

$$\begin{aligned}& & v', |\sigma'| + 1 &\vDash_{\sigma' \cdot (ts,D) \cdot \overline{(ts',D')\cdot\sigma''}^{\mathrm{TP}}} p'\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1}(ts')\\ &\iff & v', |\sigma'| + 1 &\vDash_{\sigma' \cdot (ts,D) \cdot \overline{(ts',D')\cdot\sigma''}^{\mathrm{TP}}} p' \bigcirc_I \varphi_1\\ \text{and}\quad & & v', |\sigma'| + 1 &\vDash_{\sigma' \cdot (ts,D) \cdot \overline{(ts',D')\cdot\sigma''}^{\mathrm{TP}}} p'\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}(ts')\\ &\iff & v', |\sigma'| + 1 &\vDash_{\sigma' \cdot (ts,D) \cdot \overline{(ts',D')\cdot\sigma''}^{\mathrm{TP}}} p'(\varphi_1\ \mathsf{U}_I\ \varphi_2)\end{aligned}$$

by definition of the fo and $\overline{\bullet}^{\mathrm{TP}}$. The conclusion follows by induction on φ, using the definition of $[\bullet]_+$ and the standard unrolling rules for $\bigcirc$ and U.

Lemma 3. *Whenever $X \supseteq \mathsf{FO}^+_{|\sigma|,\tau_{|\sigma|}}(\varphi)$, the converse of (⋆) also holds for* Sat *constructed as in Lemma 2.*

Proof. By structural induction of φ, applying the additional rules greedily. The premisses of the additional rules that depend on future obligations are always satisfied, since $X \supseteq \mathsf{FO}^+$. The parts of the proof that remain to be discharged using rules from the original proof system do not depend on the future. The conclusion follows from the completeness of the proof system in [42].

B.4 Soundness (Section 5.4)

For safety properties, we have the following characterization of soundness:

Lemma 8. *An enforcer $\mathcal{E}$ is sound with respect to a safety formula φ iff for any $\sigma \in \mathbb{T}_\omega$ and any prefix σ' of $\mathcal{E}(\sigma)$, there exists σ'' such that $\sigma' \cdot \sigma'' \in P$.*

Proof. Straightforward by definition of safety.

In the following, we write $\sigma \preceq \sigma'$, or equivalenty, $\sigma' \succeq \sigma$, when $\sigma \in \mathbb{T}_f$, $\sigma' \in \mathbb{T}$, and σ is a prefix of σ', i.e., there exists $\sigma'' \in \mathbb{T}$ such that $\sigma' = \sigma \cdot \sigma''$.

Lemma 9. *For any φ such that $\Gamma \vdash \varphi : \mathsf{Cau}$, the formula $[\varphi]_+$ is safety.*

Proof. We prove, for all φ:

$$\begin{aligned} P(\varphi) \equiv\ & \forall p \in \{+,-\}, \sigma_0 \in \mathbb{T}_f \setminus \{\varepsilon\}, v \in \mathbb{V} \to \mathbb{D}, \sigma \in \mathbb{T}_\omega. \\ & \Gamma \vdash \varphi : \{\mathsf{Cau}\}_p \wedge v, |\sigma_0| - 1 \vDash_{\sigma_0 \cdot \sigma} (-p)[\varphi]_p \\ & \implies \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma'.\ v, |\sigma_0| - 1 \vDash_{\sigma_0 \cdot \sigma''} (-p)[\varphi]_p \end{aligned}$$

by induction on φ. Let $\varphi, p, \sigma_0 \neq \varepsilon, v, \sigma$ such that $\Gamma \vdash p : \{\mathsf{Cau}\}_p$, $v, |\sigma_0|-1 \vDash_{\sigma_0 \cdot \sigma} (-p)[\varphi]_p$. For any $\sigma'' \in \mathbb{T}$, denote $\widehat{\sigma''} := \sigma_0 \cdot \sigma''$. Let $i := |\sigma_0| - 1$.

- If $\varphi = \top$, $\varphi = \bot$, or $\varphi = e(t_1, \ldots, t_k)$, then φ is non-temporal and $[\varphi]_p = \varphi$, and hence setting $\sigma' = \sigma$ guarantees $\forall \sigma'' \succeq \sigma'.\ v, |\sigma_0| \vDash_{\widehat{\sigma''}} (-p)[\varphi]_p$.
- If $\varphi = \neg\varphi_1$, assume $P(\varphi_1)$. Since $\Gamma \vdash \varphi : \{\mathsf{Cau}\}_p$, then $\Gamma \vdash \varphi_1 : \{\mathsf{Cau}\}_{-p}$ (using rules NotCau and NotSup). From $v, i \vDash_{\widehat{\sigma}} (-p)[\varphi]_p$ and $[\varphi]_p = \neg[\varphi_1]_{-p}$ we get $v, i \vDash_{\widehat{\sigma}} p[\varphi_1]_{-p}$ and obtain $\sigma' \preceq \sigma$ such that $\forall \sigma'' \succeq \sigma'.\ v, i \vDash_{\widehat{\sigma''}} p[\varphi_1]_{-p}$ by $P(\varphi_1)$. Hence, $\forall \sigma'' \succeq \sigma'.\ v, i \vDash_{\widehat{\sigma''}} (-p)[\varphi]_p$.
- If $\varphi = \exists x.\, \varphi_1$, assume $P(\varphi_1)$; there are two cases:
  - If $p = +$, then $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ using rule ExistsCau. Furthermore, $[\varphi]_+ = [\varphi_1]_+[x/0]$. From $v, i \vDash_{\widehat{\sigma}} \neg[\varphi]_+$ we get $v[x \mapsto 0], i \vDash_{\widehat{\sigma}} \neg[\varphi_1]_+$ and obtain a finite $\sigma' \preceq \sigma$ such that $\forall \sigma'' \succeq \sigma'.\ v[x \mapsto 0], i \vDash_{\widehat{\sigma''}} \neg[\varphi_1]_+ \iff \forall \sigma' \succeq \sigma'.\ v, i \vDash_{\widehat{\sigma''}} \neg[\varphi]_+$.
  - If $p = -$, then $\Gamma \vdash \varphi : \mathsf{Sup}$ using rule ExistsSup and $[\exists x.\ \varphi]_- = \exists x.\ [\varphi_1]_-$. Then

$$\begin{aligned} v, i \vDash_{\widehat{\sigma}} [\varphi]_- &\overset{\text{def. } \vDash}{\iff} \exists d.\ v[x := d], i \vDash_{\widehat{\sigma}} [\varphi_1]_- \\ &\overset{P(\varphi_1)}{\implies} \exists d.\ \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma'.\ v[x := d], i \vDash_{\widehat{\sigma''}} [\varphi_1]_- \\ &\implies \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma'.\ \exists d.\ v[x := d], i \vDash_{\widehat{\sigma''}} [\varphi_1]_- \\ &\overset{\text{def. } \vDash}{\iff} \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma'.\ v, i \vDash_{\widehat{\sigma''}} [\varphi]_-. \end{aligned}$$

- If $\varphi = \varphi_1 \wedge \varphi_2$, we have $P(\varphi_1)$ and $P(\varphi_2)$; there are two cases:
  - If $p = +$, then $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ and $\Gamma \vdash \varphi_2 : \mathsf{Cau}$ by rule AndCau and $[\varphi_1 \wedge \varphi_2]_+ = [\varphi_1]_+ \wedge [\varphi_2]_+$. Then

$$\begin{aligned} v, i \vDash_{\widehat{\sigma}} \neg[\varphi]_+ &\overset{\text{def. } \vDash}{\iff} v, i \vDash_{\widehat{\sigma}} \neg[\varphi_1]_+ \text{ or } v, i \vDash_{\widehat{\sigma}, } \neg[\varphi_2]_+ \\ &\overset{P(\varphi_1), P(\varphi_2)}{\implies} \\ &\qquad \left(\exists \sigma_1' \preceq \sigma. \forall \sigma'' \succeq \sigma_1'. v, i \vDash_{\widehat{\sigma''}} \neg[\varphi_1]_+\right) \\ &\qquad \text{or } \left(\exists \sigma_2' \preceq \sigma. \forall \sigma'' \succeq \sigma_2'. v, i \vDash_{\widehat{\sigma''}} \neg[\varphi_2]_+\right) \\ &\overset{(\star)}{\implies} \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma'. (v, i \vDash_{\widehat{\sigma}} \neg[\varphi_1]_+ \text{ or } v, i \vDash_{\widehat{\sigma}} \neg[\varphi_2]_+) \\ &\overset{\text{def. } \vDash}{\iff} \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma'. v, i \vDash_{\widehat{\sigma''}} \neg[\varphi]_+ \end{aligned}$$

    where $(\star)$ is obtained by choosing as witness for σ' the shortest of the witnesses for σ_1' and σ_2'.

- If $p = -$, assume w.l.o.g. that rule AndSupL is applied (the case of AndSupR is similar up to symmetry). Then $\Gamma \vdash \varphi_1 : \mathsf{Sup}$ by rule AndSupL and $[\varphi]_- = [\varphi_1]^- \wedge^{R\omega} \varphi_2$. We have, for arbitrary σ,

$$\begin{aligned} v, i \vDash_{\widehat{\sigma}} [\varphi]_- &\overset{\text{def. } \vDash}{\iff} v, i \vDash_{\widehat{\sigma}} [\varphi_1]_- \text{ and } \exists \sigma_2'.\ v, i \vDash_{\widehat{\sigma_2'}} \varphi_2 \\ &\overset{P(\varphi_1)}{\implies} \exists \sigma_1' \preceq \sigma.\, \forall \sigma'' \succeq \sigma_1'.\, v, i \vDash_{\widehat{\sigma''}} [\varphi_1]_- \text{ and } \exists \sigma_2'.\ v, i \vDash_{\widehat{\sigma_2'}} \varphi_2 \\ &\implies \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma'.\ (v, i \vDash_{\widehat{\sigma''}} [\varphi_1]_- \text{ and } \exists \sigma_2'.\ v, i \vDash_{\widehat{\sigma_2'}} \varphi_2) \\ &\overset{\text{def. } \vDash}{\implies} \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma'.\, v, i \vDash_{\widehat{\sigma''}} [\varphi]_-. \end{aligned}$$

– If $\varphi = \bigcirc_I \varphi_1$, we have $P(\varphi_1)$. Let $\sigma = (\tau', D') \cdot \sigma_1$; there are two cases:

- If $p = +$, then $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ and $I = [0, \infty)$ by rule NextCau and $[\bigcirc_I \varphi_1]_+ = \bigcirc_I [\varphi_1]_+$. We have

$$\begin{aligned} v, i \vDash_{\widehat{\sigma}} \neg[\varphi]_+ &\overset{\text{def. } \vDash}{\iff} v, i+1 \vDash_{\sigma_0 \cdot \langle (\tau', D') \rangle \cdot \sigma_1} \neg[\varphi_1]_+ \\ &\overset{P(\varphi_1)}{\implies} \exists \sigma_1' \preceq \sigma_1.\, \forall \sigma'' \succeq \sigma_1'.\, v, i+1 \vDash_{\sigma_0 \cdot \langle (\tau', D') \rangle \cdot \sigma''} \neg[\varphi_1]_+ \\ &\overset{(\star)}{\implies} \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma'.\, v, i+1 \vDash_{\sigma_0 \cdot \sigma''} \neg[\varphi_1]_+ \\ &\overset{\text{def. } \vDash}{\iff} \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma'.\, v, i \vDash_{\widehat{\sigma''}} \neg[\varphi]_+ \end{aligned}$$

  where $(\star)$ is obtained by choosing as witness for σ' the prefix $\sigma_1' \cdot (\tau', D')$ where σ_1' is a witness of the LHS.
- If $p = -$, then $\Gamma \vdash \varphi_1 : \mathsf{Sup}$ by rule NextSup and $[\bigcirc_I \varphi_1]_- = \bigcirc_I [\varphi_1]_-$. We have

$$\begin{aligned} v, i \vDash_{\widehat{\sigma}} [\varphi]_- &\overset{\text{def. } \vDash}{\iff} v, i+1 \vDash_{\sigma_0 \cdot \sigma_1} [\varphi_1]_- \text{ and } \tau' - \tau \in I \\ &\overset{P(\varphi_1)}{\implies} \exists \sigma_1' \preceq \sigma_1.\, \forall \sigma'' \succeq \sigma_1'.\, v, i+1 \vDash_{\sigma_0 \cdot \langle (\tau', D') \rangle \cdot \sigma''} [\varphi_1]_- \\ &\qquad \text{and } \tau' - \tau \in I \\ &\overset{(\star)}{\implies} \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma'.\, v, i+1 \vDash_{\sigma_0 \cdot \sigma''} [\varphi_1]_- \text{ and } \tau' - \tau \in I \\ &\overset{\text{def. } \vDash}{\iff} \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma'.\ v, i \vDash_{\widehat{\sigma''}} [\varphi]_- \end{aligned}$$

  where $(\star)$ is obtained as in the previous case.

– If $\varphi = \varphi_1 \mathsf{S}_I \varphi_2$, we have $P(\varphi_1)$ and $P(\varphi_2)$ for all j. There are three cases:

- If $p = +$, then $\Gamma \vdash \varphi_2 : \mathsf{Cau}$ and $0 \in I$ by rule SinceCau, and $[\varphi]_+ = \varphi_1 \, \mathsf{S}_I^{L\omega+} \, [\varphi_2]_+$. We have

$$\begin{aligned}
v, i \vDash_{\widehat{\sigma}} \neg[\varphi]_+ &\overset{\text{def. } \vDash}{\iff} \forall j \leq i.\ \tau_i - \tau_j \in I \Rightarrow \\
&\qquad \big(\exists \sigma_1'.\ v, j \vDash_{\sigma_{|..i} \cdot \sigma_1'} \neg[\varphi_2]_+ \\
&\qquad \text{or } \exists k \in [j+1..i].\ v, k \vDash_{\widehat{\sigma_{|..i} \cdot \sigma_1'}} \neg\varphi_1\big) \\
&\overset{P(\varphi_2)}{\Longrightarrow} \forall j \leq i.\ \tau_i - \tau_j \in I \Rightarrow \\
&\qquad \big(\exists \sigma_2' \preceq \sigma.\, \forall \sigma'' \succeq \sigma_2'.\ v, j \vDash_{\widehat{\sigma''}} \neg[\varphi_2]_+ \\
&\qquad \text{or } \exists k \in [j+1..i], \sigma_1'.\ v, k \vDash_{\widehat{\sigma_1'}} \neg\varphi_1\big) \\
&\Longrightarrow \exists \sigma' \preceq \sigma.\, \forall j \leq i.\ \tau_i - \tau_j \in I \Rightarrow \\
&\qquad \big(\forall \sigma'' \succeq \sigma_2'.\ v, j \vDash_{\widehat{\sigma''}} \neg[\varphi_2]_+ \\
&\qquad \text{or } \exists k \in [j+1..i], \sigma_1'.\ v, k \vDash_{\widehat{\sigma_1'}} \neg\varphi_1\big) \\
&\overset{\text{def. } \vDash}{\Longrightarrow} \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma'.\ v, i \vDash_{\widehat{\sigma''}} \neg[\varphi]_+.
\end{aligned}$$

  - If $p = -$ and either SinceSupL or SinceSupLR, the proof is similar, using the semantics of $\varphi_1 \, \mathsf{S}^{L\omega-} \, \varphi_2$ or $\varphi_1 \, \mathsf{S}^{R\omega-} \, \varphi_2$.

– If $\varphi = \varphi_1 \, \mathsf{U}_{[a,b]} \, \varphi_2$, assume $P(\varphi_1)$ and $P(\varphi_2)$. There are again three cases:

  - If $p = +$ and rule UntilCauLR is used, then $\Gamma \vdash \varphi_1 : \mathsf{Cau}$, $\Gamma \vdash \varphi_2 : \mathsf{Cau}$, and $b \neq \infty$. Furthermore, $[\varphi]_+ = [\varphi_1]_+ \, \mathsf{U}_I \, [\varphi_2]_+$. We have:

$$\begin{aligned}
v, i \vDash_{\widehat{\sigma}} \neg[\varphi]_+ &\overset{\text{def. } \vDash}{\iff} \forall j \geq i.\ \tau_j - \tau_i \in I \Rightarrow \\
&\qquad (v, j \vDash_{\widehat{\sigma}} \neg[\varphi_2]_+ \text{ or } \exists k \in [i..j-1].\ v, k \vDash_{\widehat{\sigma}} \neg[\varphi_1]_+) \\
&\overset{P(\varphi_1), P(\varphi_2)}{\Longrightarrow} \forall j \geq i.\ \tau_j - \tau_i \in I \Rightarrow \\
&\qquad \big(\exists \sigma_2' \preceq \sigma.\, \forall \sigma'' \succeq \sigma_2'.\ v, j \vDash_{\widehat{\sigma''}} \neg[\varphi_2]_+ \\
&\qquad \text{or } \exists k \in [i..j-1].\, \exists \sigma_1' \preceq \sigma.\, \forall \sigma'' \succeq \sigma_1'.\ v, k \vDash_{\widehat{\sigma''}} \neg[\varphi_1]_+\big) \\
&\overset{(\star)}{\Longrightarrow} \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma_2'.\, \forall j \geq i.\ \tau_j - \tau_i \in I \Rightarrow \\
&\qquad \big(v, j \vDash_{\widehat{\sigma''}} \neg[\varphi_2]_+ \text{ or } \exists k \in [i..j-1].\ v, k \vDash_{\widehat{\sigma''}} \neg[\varphi_1]_+\big) \\
&\overset{\text{def. } \vDash}{\Longrightarrow} \exists \sigma' \preceq \sigma.\, \forall \sigma'' \succeq \sigma_2'.\ v, i \vDash_{\widehat{\sigma''}} \neg[\varphi]_+
\end{aligned}$$

    where $(\star)$ is obtained by choosing as witness for σ' the longest of a witness for σ_2' and either a witness for σ_1' for each value of j such that $\tau_j - \tau_i \in I$. The progress property of the trace σ guarantees that the set of such indices j is finite, and hence the witness for σ' is finite.
  - If $p = +$ and rule UntilCauR is used, the proof is similar, using the semantics of $\varphi_1 \, \mathsf{U}^{L\omega+} \, \varphi_2$.

- If $p = -$ and rule UntilSup is used, then $\Gamma \vdash \varphi_1 : \mathsf{Sup}$. Furthermore, $[\varphi]_- = [\varphi_1]_- \; \mathsf{U}^{R\omega} \; \varphi_2$. We have:

$$\begin{aligned}
v, i \vDash_{\widehat{\sigma}} [\varphi]_- &\overset{\text{def.} \vDash}{\Longleftrightarrow} \exists j \geq i. \big(\tau_j - \tau_i \in I \text{ and } \exists \sigma_2'.\ v, j \vDash_{\widehat{\sigma}|_{..i} \cdot \sigma_2'} \varphi_2 \\
&\qquad \text{and } \forall k \in [i..j-1].\ v, k \vDash_{\widehat{\sigma}} [\varphi_1]_-\big) \\
&\overset{P(\varphi_1)}{\Longrightarrow} \exists j \geq i. \Big(\tau_j - \tau_i \in I \text{ and } \exists \sigma_2'.\ v, j \vDash_{\widehat{\sigma_2'}} \varphi_2 \\
&\qquad \text{and } \forall k \in [i..j-1]. \exists \sigma_1' \preceq \sigma. \forall \sigma'' \succeq \sigma_1'.\ v, k \vDash_{\widehat{\sigma''}} [\varphi_1]_-\Big) \\
&\overset{(\star)}{\Longrightarrow} \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma'. \exists j \geq i. \\
&\qquad \Big(\tau_j - \tau_i \in I \text{ and } \exists \sigma_2'.\ v, j \vDash_{\widehat{\sigma_2'}} \varphi_2 \\
&\qquad \text{and } \forall k \in [i..j-1].\ v, k \vDash_{\widehat{\sigma''}} [\varphi_1]_-\Big) \\
&\overset{\text{def.} \vDash}{\Longrightarrow} \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma_2'.\ v, i \vDash_{\widehat{\sigma''}} [\varphi]_-.
\end{aligned}$$

  where $(\star)$ is obtained by choosing as witness for σ' the longest of the witnesses for σ_1' for each value of j such that $\tau_j - \tau_i \in I$. The progress property of the trace σ guarantees that the set of such indices j is finite, and hence the witness for σ' is also finite.

Given any trace $\langle(\tau, D)\rangle \cdot \sigma \in \mathbb{T}_\omega$, we can now use $P(\varphi)$ instantiated with $p = +$, $\sigma_0 = \langle(\tau, D)\rangle$, and σ to obtain

$$\Gamma \vdash \varphi : \mathsf{Cau} \wedge v, 0 \vDash_{\langle(\tau,D)\rangle \cdot \sigma} \neg[\varphi]_p \Longrightarrow \exists \sigma' \preceq \sigma.\ \left(\forall \sigma'' \succeq \sigma'.\ v, i \vDash_{(\tau,D)\cdot\sigma''} \neg[\varphi]_p\right),$$

which implies

$$\forall \sigma \in \mathbb{T}_\omega. \Gamma \vdash \varphi : \mathsf{Cau} \wedge \sigma \notin \mathcal{L}(\varphi) \Longrightarrow \exists \sigma' \preceq \sigma. \forall \sigma'' \succeq \sigma'.\ \sigma'' \notin \mathcal{L}(\varphi),$$

i.e., φ is safety.

Having proved in Lemma 4 that the transformed formulae soundly approximate the original formulae, we will use the following weakening of $(\star\star)$ where the *transformed* future obligations are assumed to be satisfied.

$$\begin{aligned}
&\textsc{Sat}(v, \varphi, \sigma', X) \\
&\overset{(\star\star\star)}{\Longrightarrow} \Big(\forall ts \in \mathbb{N}, D \in \mathbb{DB}, \sigma'' \in \mathbb{T}_\omega.\ (\forall (\xi, v', p') \in X.\ v', |\sigma'| \vDash_{\sigma' \cdot \overline{(ts,D)\cdot\sigma''}^{\mathrm{TP}}} p'[\xi(ts)]_{p'}) \\
&\qquad\qquad \Longrightarrow v, |\sigma'| - 1 \vDash_{\sigma' \cdot \overline{(ts,D)\cdot\sigma''}^{\mathrm{TP}}} [\varphi]_+\Big).
\end{aligned}$$

Lemma 10. *Let $\sigma \in \mathbb{T}_f$, $(\tau', D') \cdot \sigma' \in \mathbb{T}_\omega$, $\tau \in \mathbb{N}$ such that $\tau' \geq \tau$, and $\mathsf{enf}^p_{\tau,b}(\varphi, \sigma \cdot (\tau, D), X', v) = (S, C, X)$ (in particular, $\mathsf{enf}^p_{\tau,b}$ terminates on these inputs). Assume further that the following hold:*

$P_1 \equiv\ \Gamma \vdash \varphi : \{\mathsf{Cau}\}_p$*;*
$P_2 \equiv\ \tau' > \tau \Longrightarrow b$*;*

$$P_3 \equiv \forall(\xi, v', p') \in X.\, v', |\sigma| + 1 \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau', D')\cdot\sigma'}^{\mathrm{TP}}} p'[\xi(\tau')]_{p'}$$

Then $Q \equiv v, |\sigma| \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau', D')\cdot\sigma'}^{\mathrm{TP}}} p[\varphi]_p$ *holds.*

Proof. Fix $\Gamma, X', \sigma, \sigma', \tau, \tau', D'$. We prove

$$\begin{aligned} & P(\varphi) \\ \equiv\ & \forall D, v, b, S, C, X, p.\ \mathsf{enf}^p_{\tau,b}(\varphi, \sigma\cdot(\tau, D), X', v) = (S, C, X) \\ & \wedge P_1(\Gamma, \varphi, p) \wedge P_2(b) \wedge P_3(X, D, S, C) \Longrightarrow Q(v, D, S, C, p, \varphi) \end{aligned}$$

by structural induction on φ.

- If $\varphi = \bot$, then P_1 yields $p = -$ (rule FALSE). In this case, Q is simply $v, |\sigma| \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau', D')\cdot\sigma'}^{\mathrm{TP}}} -\bot \equiv \top$, which always holds.
- If $\varphi = \top$, then P_1 yields $p = +$ (rule TRUE). In this case, Q is simply $v, |\sigma| \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau', D')\cdot\sigma'}^{\mathrm{TP}}} \top$, which always holds.
- If $\varphi = e(t_1, \ldots, t_k)$, we know by P_1 that either $p = -$, $e \in \mathsf{Sup}$, and $\Gamma(e) = \mathsf{Sup}$ (rule SUP); or $p = +$, $e \in \mathsf{Cau}$, and $\Gamma(e) = \mathsf{Cau}$ (rule CAU). Consider the former case. By definition of enf^-, we have $S = \{(r, (v(t_1), \ldots, v(t_n)))\}$, $C = \emptyset$, $X = \emptyset$. Hence $(r, (v(t_1), \ldots, v(t_n))) \notin D\setminus S\cup C$, and $Q \equiv v, |\sigma| \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau', D')\cdot\sigma'}^{\mathrm{TP}}} -r(t_1, \ldots, t_n)$ holds. The other case is similar using the definition of enf^+.
- If $\varphi = \neg\varphi'$, assume $P(\varphi')$. We know by P_1 (rules NOTCAU and NOTSUP) that $\Gamma \vdash \varphi' : \{\mathsf{Cau}\}_{-p} \equiv P_1(\Gamma, \varphi', -p)$. Observe that P_2 and P_3 do not depend on φ. Hence, we can use $P(\varphi')$ to show $Q(v, D, S, C, -p, \varphi')$, which is exactly $Q(v, D, S, C, p, \varphi)$ by the semantics of $\neg$ and the definition of $[\cdot]$.
- If $\varphi = \exists x.\ \varphi'$, assume $P(\varphi')$; there are two cases:
  - If $p = +$, then $P_1(\Gamma, \varphi, p)$ (rule EXISTSCAU) yields $\Gamma \vdash \varphi' : \mathsf{Cau} \equiv P_1(\Gamma, \varphi', p)$. Furthermore, $\mathsf{enf}^+_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{enf}^+_{\tau,b}(\varphi', \sigma, X, v[x \mapsto 0])$. Observe that P_2 and P_3 do not depend on φ and v. Hence, we can use $P(\varphi')$ to show $Q(v[x \mapsto 0], D, S, C, p, \varphi')$, which implies $Q(v, D, S, C, p, \varphi)$ by the semantics of $\exists$.
  - If $p = -$, then $P_1(\Gamma, \varphi, p)$ (rule EXISTSSUP) yields $\Gamma \vdash \varphi' : \mathsf{Cau} \equiv P_1(\Gamma, \varphi', p)$ and $\vdash \varphi' : \mathsf{PG}(x)^+$. Furthermore, $\mathsf{enf}^-_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{fp}(\sigma, X, \mathsf{enf}^-_{\mathsf{ex},\varphi',v,\tau,b})$. Let $\sigma_1 = \sigma\cdot(\tau, D \setminus S \cup C)$. Since $\mathsf{enf}^-_{\tau,b}$ terminates, the fixpoint compu-

tation also does, and we have

$$
\begin{aligned}
&\forall d \in \mathsf{AD}_{|\sigma_1|-1}(\varphi').\, \textsc{Sat}(v[x \mapsto d], \neg\varphi', \sigma_1, X)\\
\overset{(\star\star\star)}{\Longrightarrow} &\forall d \in \mathsf{AD}_{|\sigma_1|-1}(\varphi').\, (\forall (\xi, v', p') \in X.\ v', |\sigma_1| \vDash_{\sigma_1 \cdot \overline{(\tau', D) \cdot \sigma'}^{\mathrm{TP}}} p'[\xi(\tau')]_{p'})\\
&\qquad \Longrightarrow v[x \mapsto d], |\sigma_1| - 1 \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} \neg[\varphi']_-\\
\overset{\text{Lm. 1}}{\Longrightarrow} &\forall d \in \mathbb{D}.\, (\forall (\xi, v', p') \in X.\ v', |\sigma_1| \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} p'[\xi(\tau')]_{p'})\\
&\qquad \Longrightarrow v[x \mapsto d], |\sigma_1| - 1 \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} \neg[\varphi']_-\\
\overset{\text{def. } \vDash}{\Longrightarrow} &\Big((\forall (\xi, v', p') \in X.\ v', |\sigma_1| \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} p'[\xi(\tau')]_{p'})\\
&\quad \Longrightarrow v, |\sigma_1| - 1 \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} \neg[\varphi]_-\Big)\\
\Longleftrightarrow &(P_3 \Longrightarrow Q)
\end{aligned}
$$

which concludes the proof.

- If $\varphi = \varphi_1 \wedge \varphi_2$, assume $P(\varphi_1)$ and $P(\varphi_2)$; there are two cases:
  - If $p = -$, then $P_1(\Gamma, \varphi, p)$ yields either $P_1(\Gamma, \varphi_1, p)$ (rule AndSupL) or $P_1(\Gamma, \varphi_2, p)$ (rule AndSupR). W.l.o.g., consider the former case (the latter case is similar exchanging the role of φ_1 and φ_2). We have $\mathsf{enf}^+_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{enf}^+_{\tau,b}(\varphi_1, \sigma, X, v)$. Observe that P_2 and P_3 do not depend on φ and v. Hence we get $Q(v, D, S, C, p, \varphi_1)$, which implies $Q(v, D, S, C, p, \varphi)$ by the semantics of $\wedge^{R\omega}$ and $[\varphi \wedge_{\textsc{AndSupL}} \psi]_- = [\varphi_1]_- \wedge^{R\omega} \varphi_2$: the conjunction $[\varphi_1]_- \wedge^{R\omega} \varphi_2$ is false at i if $[\varphi_1]_-$ is false at i.
  - If $p = +$, then $P_1(\Gamma, \varphi, p)$ (rule AndCau) yields $\Gamma \vdash \varphi_1 : \mathsf{Cau} \equiv P_1(\Gamma, \varphi_1, p)$ and $\Gamma \vdash \varphi_2 : \mathsf{Cau} \equiv P_1(\Gamma, \varphi_2, p)$. Furthermore, $\mathsf{enf}^+_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{fp}(\sigma, X, \mathsf{enf}^+_{\mathsf{and},\varphi_1,\varphi_2,v,\tau,b})$. Let $\sigma_1 = \sigma \cdot (\tau, D \setminus S \cup C)$. Since enf^+ terminates, the fixpoint computation also does, and we have

$$
\begin{aligned}
&\textsc{Sat}(v, \varphi_1, \sigma_1, X) \wedge \textsc{Sat}(v, \varphi_2, \sigma_1, X)\\
\overset{(\star\star\star)}{\Longrightarrow} &\Big(\forall (\xi, v', p') \in X.\ v', |\sigma_1| \vDash_{\sigma_1 \cdot \overline{(\tau, D') \cdot \sigma'}^{\mathrm{TP}}} p'[\xi(\tau)']_{p'})\\
&\quad \Longrightarrow v, |\sigma_1| - 1 \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} [\varphi_1]_+ \wedge v, |\sigma_1| - 1 \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} [\varphi_2]_+\\
\overset{\text{def. } \vDash}{\Longleftrightarrow} &\Big(\forall (\xi, v', p') \in X.\ v', |\sigma_1| \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} p'[\xi(\tau')]_{p'})\\
&\quad \Longrightarrow v, |\sigma_1| - 1 \vDash_{\sigma_1 \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}},+} [\varphi]_+\\
\Longleftrightarrow &(P_3 \Longrightarrow Q)
\end{aligned}
$$

which concludes the proof using $[\varphi_1 \wedge \varphi_2]_+ = [\varphi_1]_+ \wedge [\varphi_2]_+$.

- If $\varphi = \bigcirc_I \varphi'$, assume $P(\varphi')$; there are two cases:
  - If $p = -$, then by P_1 (rule NextSup) we get $\Gamma \vdash \varphi : \mathsf{Sup}$. By definition of enf^-, we have $S = C = \emptyset$ and $X = \{(\mathsf{fo}_{\tau,\bigcirc,I,\varphi'}, v, -)\}$. By definition of $\overline{\bullet}^{\mathrm{TP}}$ and $\mathsf{fo}_{\tau,\bigcirc,I,\varphi'}$, then if $\tau' - \tau \in I$, P_3 yields $v, |\sigma|+1 \vDash_{\sigma \cdot (\tau, D \setminus S \cup C) \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} \neg[\varphi']_-$ which implies $Q \equiv v, |\sigma| \vDash_{\sigma \cdot (\tau, D \setminus S \cup C) \cdot \overline{(\tau', D') \cdot \sigma'}^{\mathrm{TP}}} \neg \bigcirc_I [\varphi']_-$. If $\tau' - \tau \notin I$, the semantics of $\bigcirc_I$ similarly guarantees Q.

- If $p = +$, then by P_1 (rule NextCau) we get $\Gamma \vdash \varphi : \mathsf{Cau}$. By definition of enf^+, we have $S = C = \emptyset$ and $X = \{(\mathsf{fo}_{\tau,\bigcirc,I,\varphi'}, v, +)\}$. Then P_3 yields $v, |\sigma| + 1 \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau',D')\cdot\sigma'}^{\mathsf{TP}},+} [\varphi']_+$, which implies $Q \equiv v, |\sigma| \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau',D')\cdot\sigma'}^{\mathsf{TP}},+} \bigcirc[\varphi']_+$.

– If $\varphi = \varphi_1 \,\mathsf{S}_I\, \varphi_2$, assume $P(\varphi_1)$ and $P(\varphi_2)$; there are three cases:
  - If $p = +$, then by P_1 (rule SinceCau) we get $0 \in I$, $\Gamma \vdash \varphi_2 : \mathsf{Cau}$. We have $\mathsf{enf}^+_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{enf}^+_{\tau,b}(\varphi_2, \sigma, X, v)$; as before, P_2 and P_3 are independent of φ and v. Hence we get $Q(v, D, S, C, p, \varphi_2)$, which implies $Q(v, D, S, C, p, \varphi)$ using the semantics of $\mathsf{S}_I^{L\omega+}$, the equation $[\varphi_1 \mathsf{S}_I \varphi_2]_+ = \varphi_1 \,\mathsf{S}_I^{L\omega+}\, [\varphi_2]_+$, and the fact that $0 \in I$.
  - If $p = -$ and SinceSupL is used, we have $0 \notin I$ and $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ by P_1. Further, $\mathsf{enf}^-_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{enf}^-_{\tau,b}(\varphi_1, \sigma, X, v)$, and P_2 and P_3 are independent of φ and v. Hence $Q(v, D, S, C, p, \varphi_2)$, which implies $Q(v, D, S, C, p, \varphi)$ using the semantics of $\mathsf{S}_I^{L\omega-}$, the equation $[\varphi_1 \mathsf{S}_I \varphi_2]_+ = \varphi_1 \,\mathsf{S}_I^{L\omega-}\, [\varphi_2]_+$, and the fact that $0 \notin I$.
  - If $p = -$ and SinceSupLR is used, we have $0 \in I$, $\Gamma \vdash \varphi_1 : \mathsf{Cau}$, and $\Gamma \vdash \varphi_2 : \mathsf{Cau}$ by P_1. Further,

$$\mathsf{enf}^-_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{fp}(\sigma, X, \mathsf{enf}^+_{\mathsf{and},\neg(\varphi_1 \wedge_{\text{AndSupL}} (\varphi_1 \mathsf{S}_I \varphi_2)),\neg\varphi_2,v,\tau,b})$$

    whereby $\Gamma \vdash \neg\varphi_2 : \mathsf{Cau}$ and $\Gamma \vdash \neg(\varphi_1 \wedge_{\text{AndSupL}} (\varphi_1 \,\mathsf{S}_I\, \varphi_2)) : \mathsf{Cau}$. The same proof as for $\wedge$ and $p = +$ above shows $Q(v, D, S, C, -p, \neg\varphi_2 \wedge \neg(\varphi_1 \wedge (\varphi_1 \,\mathsf{S}_I\, \varphi_2))$ since, when $0 \in I$, we have

$$\neg\,(\varphi_1 \,\mathsf{S}_I\, \varphi_2) \equiv \neg\varphi_2 \wedge \neg\,(\varphi_1 \wedge (\varphi_1 \,\mathsf{S}_I\, \varphi_2))$$

    Hence $Q(v, D, S, C, p, \varphi)$ holds.

– If $\varphi = \varphi_1 \,\mathsf{U}_I\, \varphi_2$, assume $P(\varphi_1)$ and $P(\varphi_2)$; there are three cases:
  - If $p = +$, then by P_1 (rule UntilCau) we get $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ and $\Gamma \vdash \varphi_2 : \mathsf{Cau}$. If $I = [0, 0]$ and $b = \top$, we have $\mathsf{enf}^+_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{enf}^+_{\tau,b}(\varphi_2, \sigma, X, v)$; as before, P_2 and P_3 are independent of φ and v. Hence we get $Q(v, D, S, C, p, \varphi_2)$, which implies $Q(v, D, S, C, p, \varphi)$ by the semantics of $[\varphi_1]_+ \,\mathsf{U}_I\, [\varphi_2]_+ = [\varphi_1 \,\mathsf{U}_I\, \varphi_2]_+$ and the fact that $0 \in I$. Otherwise, $\mathsf{enf}^+_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{enf}^+_{\tau,b}(\varphi_1, \sigma, X, v, b) \uplus (\emptyset, \emptyset, \{(\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}, v, +)\}$. Using P_3 and the definition of $\overline{\bullet}^{\mathsf{TP}}$, we get $v, |\sigma| + 1 \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau',D')\cdot\sigma'}^{\mathsf{TP}}} [\varphi_1]_+ \,\mathsf{U}_{I-(\tau'-\tau)}\, [\varphi_2]_+$. Using $P(\varphi_1)$, we also get $Q(v, D, S, C, p, \varphi_1)$. Together, these two facts imply

$$\begin{aligned} & v, |\sigma| \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau',D')\cdot\sigma'}^{\mathsf{TP}}} [\varphi_1]_+ \wedge \bigcirc\left([\varphi_1]_+ \,\mathsf{U}_{I-(\tau'-\tau)}\, [\varphi_2]_+\right) \\ \implies\ & v, |\sigma| \vDash_{\sigma\cdot(\tau, D\setminus S\cup C)\cdot\overline{(\tau',D')\cdot\sigma'}^{\mathsf{TP}}} [\varphi_1]_+ \,\mathsf{U}_I\, [\varphi_2]_+ \equiv Q. \end{aligned}$$

  - If $p = +$ and UntilCauR is used, the proof is similar to the previous case.
  - If $p = -$ and UntilSupL is used, we have $0 \notin I$ and $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ by P_1. Further, $\mathsf{enf}^-_{\tau,b}(\varphi, \sigma, X, v) = \mathsf{enf}^-_{\tau,b}(\varphi_1, \sigma, X, v)$, and P_2 and P_3 are independent of φ and v. Hence $Q(v, D, S, C, p, \varphi_1)$, which implies $Q(v, D, S, C, p, \varphi)$ using the semantics of $\mathsf{U}_I^{R\omega-}$, the equation $[\varphi_1 \,\mathsf{U}_I\, \varphi_2]_- = [\varphi_1]_- \,\mathsf{U}_I^{R\omega-}\, \varphi_2$, and the fact that $0 \notin I$.

Lemma 11. *When $\Gamma \vdash \varphi : \{\mathsf{Cau}\}$, for all p, σ, X, τ, v, b, any call to $\mathsf{enf}^p_{\tau,b}(\varphi, \sigma, X, v)$ terminates.*

Proof. By structural induction on φ. The only non-trivial cases are those that lead to a call to fp: $\varphi = \varphi_1 \wedge \varphi_2$ for $p = -$, and $\varphi = \exists x.\ \varphi_1$, $\varphi_1 \mathsf{S}_{I\,\textsc{SinceSup}\,\mathrm{LR}} \varphi_2$, $\varphi_1 \mathsf{U}_{I\,\textsc{UntilSup}} \varphi_2$ for $p = +$.

In all four cases, a similar proof is obtained by combining three facts:

(1) At each iteration of the loop in fp, $|S| + |C| + |X|$ grows strictly.
(2) Any event added to S or C by any call to enf^- or enf^+ takes its arguments in $\mathsf{AD}_{|\sigma|}(\varphi)$, which is finite; thus $|S| + |C| \leq (|\mathsf{Sup}| + |\mathsf{Cau}|)|\mathsf{AD}(\sigma)|^{\max \iota(\mathbb{E})} < \infty$.
(3) Any triple added to X is in $\mathsf{FO}^+_{|\sigma|,\tau}(\varphi)$, which is also finite, and hence $|X| < \infty$.

Therefore every call fp terminates, and thus $\mathsf{enf}^p_{\tau,b}$ terminates, too.

Lemma 12. *Let φ such that $\Gamma \vdash \varphi : \{\mathsf{Cau}\}_p$, $\sigma \in \mathbb{T}_f$, v a valuation such that $v(\mathsf{fv}(\varphi)) \subseteq \mathsf{AD}_{|\sigma|}(\varphi)$. Then there exists σ' such that $v, |\sigma| \vDash_{\sigma \cdot \sigma'} p[\varphi]_p$.*

Proof. Construct

$$\begin{aligned}
\Delta &:= \mathsf{AD}_{|\sigma|}(\varphi) \\
D &:= \bigcup_{e \in \Gamma^{-1}(\mathsf{Cau})} \{(e, (d_1, \ldots, d_{a(e)})) \mid (d_1, \ldots, d_{a(e)}) \in \Delta^{a(e)}\} \\
\sigma' &:= \langle (\tau', D), (\tau' + 1, D), (\tau' + 2, D), \ldots \rangle .
\end{aligned}$$

We show

$$\begin{aligned}
& P(\varphi) \\
\equiv\ & \forall v, p, i \geq |\sigma|.\ \Gamma \vdash \varphi : \{\mathsf{Cau}\}_p \wedge v(\mathsf{fv}(\varphi)) \subseteq \mathsf{AD}_i(\varphi) \Longrightarrow v, i \vDash_{\sigma \cdot \sigma'} p[\varphi]_p.
\end{aligned}$$

by structural induction on φ. Let φ such that $\Gamma \vdash \varphi : \{\mathsf{Cau}\}_p$, $v(\mathsf{fv}(\varphi)) \subseteq \mathsf{AD}_i(\varphi)$.

- If $\varphi = \bot$ and $p = -$ or $\varphi = \top$ and $p = +$, the result is trivial.
- If $\varphi = e(t_1, \ldots, t_k)$ with $e \in \mathsf{Sup}$ and $p = -$, then the typing guarantees $\Gamma(e) = \mathsf{Sup}$, and therefore $e \notin \Gamma^{-1}(\mathsf{Cau})$ and $(e, (v(t_1), \ldots, v(t_k))) \notin D$, yielding $v, i \vDash_{\sigma \cdot \sigma'} \neg[\varphi]_-$.
- If $\varphi = e(t_1, \ldots, t_k)$ with $e \in \mathsf{Cau}$ and $p = +$, then the typing guarantees $\Gamma(e) = \mathsf{Cau}$. Furthermore, $v(t_j) \in \mathsf{AD}_{|\sigma|}(\varphi) = \Delta$ for every $j \in [1..a(e)]$ by our assumption. Therefore $(e, (v(t_1), \ldots, v(t_k))) \in D$, yielding $v, i \vDash_{\sigma \cdot \sigma'} [\varphi]_+$.
- If $\varphi = \neg\varphi_1$, assume $P(\varphi_1)$. The typing of φ yields $\Gamma \vdash \varphi_1 : \{\mathsf{Cau}\}_{-p}$. By $P(\varphi_1)$, we get $v, i \vDash_{\sigma \cdot \sigma'} (-p)[\varphi_1]_{-p}$, and hence $v, i \vDash_{\sigma \cdot \sigma'} p[\varphi]_p$.
- If $\varphi = \exists x.\ \varphi_1$, assume $P(\varphi_1)$; there are two cases:
  - If $p = +$, the typing of φ yields $\Gamma \vdash \varphi_1 : \mathsf{Cau}$. By $P(\varphi_1)$ with $v' = v[x \mapsto 0]$ and the fact that $0 \in \mathsf{AD}_i(\varphi)$, we get $v', i \vDash_{\sigma \cdot \sigma'} [\varphi_1]_+$. Hence $v, i \vDash_{\sigma \cdot \sigma'} [\varphi]_+[x/0]$ and $v, i \vDash_{\sigma \cdot \sigma'} [\varphi]_+$.
  - If $p = -$, the typing of φ guarantees $\Gamma \vdash \varphi_1 : \mathsf{Sup}$ and $\vdash \varphi_1 : \mathsf{PG}(x)^+$. Let $d \in \mathsf{AD}_i(\varphi)$ and $v_d = v[x \mapsto d]$. By $P(\varphi_1)$ with v', we get $v', i \vDash_{\sigma \cdot \sigma'} \neg[\varphi_1]_-$. As this holds for any d in $\mathsf{AD}_i(\varphi)$, we get $v, i \vDash_{\sigma \cdot \sigma'} \neg[\varphi]_-$.

- If $\varphi = \varphi_1 \wedge \varphi_2$, assume $P(\varphi_1)$ and $P(\varphi_2)$; there are three cases:
  - If $p = +$, the typing of φ yields $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ and $\Gamma \vdash \varphi_2 : \mathsf{Cau}$. By $P(\varphi_1)$ and $P(\varphi_2)$, we get $v, i \vDash_{\sigma\cdot\sigma'} [\varphi_1]_+$ and $v, i \vDash_{\sigma\cdot\sigma'} [\varphi_2]_+$, and hence $v, i \vDash_{\sigma\cdot\sigma'} [\varphi]_+$.
  - If $p = -$ and φ is typed using AndSupL, then $\Gamma \vdash \varphi_1 : \mathsf{Sup}$. By $P(\varphi_1)$, we get $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi_1]_-$, and hence $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi]_-$.
  - If $p = -$ and φ is typed using AndSupR, then $\Gamma \vdash \varphi_2 : \mathsf{Sup}$. By $P(\varphi_2)$, we get $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi_2]_-$, and hence $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi]_-$.
- If $\varphi = \bigcirc_I \varphi_1$, assume $P(\varphi_1)$; there are two cases:
  - If $p = +$, the typing of φ yields $I = [0..\infty)$ and $\Gamma \vdash \varphi_1 : \mathsf{Cau}$. By $P(\varphi_1)$ with v and $i+1$, we get $v, i+1 \vDash_{\sigma\cdot\sigma'} [\varphi_1]_+$, and hence $v, i \vDash_{\sigma\cdot\sigma'} [\varphi]_+$.
  - If $p = -$, the typing of φ yields $\Gamma \vdash \varphi_1 : \mathsf{Sup}$. By $P(\varphi_1)$ with v and $i+1$, we get $v, i+1 \vDash_{\sigma\cdot\sigma'} \neg[\varphi_1]_-$, and hence $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi]_-$.
- If $\varphi = \varphi \,\mathsf{S}_I\, \psi$, assume $P(\varphi_1)$ and $P(\varphi_2)$; there are three cases:
  - If $p = +$, the typing of φ yields $0 \in I$ and $\Gamma \vdash \varphi_2 : \mathsf{Cau}$. By $P(\varphi_2)$, we get $v, i \vDash_{\sigma\cdot\sigma'} [\varphi_2]_+$, and hence $v, i \vDash_{\sigma\cdot\sigma'} [\varphi]_+$ since $0 \in I$.
  - If $p = -$ and φ is typed using SinceSupL, then $\Gamma \vdash \varphi_1 : \mathsf{Sup}$ and $0 \notin I$. By $P(\varphi_1)$, we get $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi_1]_-$, and hence $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi]_-$ since $0 \notin I$.
  - If $p = -$ and φ is typed using SinceSupLR, then $\Gamma \vdash \varphi_2 : \mathsf{Sup}$, and $0 \in I$. By $P(\varphi_2)$, we get $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi_2]_-$, and hence $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi]_-$.
- If $\varphi = \varphi \,\mathsf{U}_I\, \psi$, assume $P(\varphi_1)$ and $P(\varphi_2)$; there are three cases:
  - If $p = +$, the typing of φ gives $I = [a, b]$ with $b < \infty$, $\Gamma \vdash \varphi_1 : \mathsf{Cau}$, and $\Gamma \vdash \varphi_2 : \mathsf{Cau}$. Let $\delta \in I$ and $k = i + \delta$. By $P(\varphi_2)$, we have $v, k \vDash_{\sigma\cdot\sigma'} [\varphi_2]_p$. By $P(\varphi_1)$, we also have $\forall j \in [i..k-1].\ v, j \vDash_{\sigma\cdot\sigma'} [\varphi_1]_+$. Hence $v, i \vDash_{\sigma\cdot\sigma'} [\varphi]_+$.
  - If $p = -$ and φ is typed using UntilSupL, then $\Gamma \vdash \varphi_1 : \mathsf{Sup}$ and $0 \notin I$. By $P(\varphi_1)$, we get $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi_1]_-$, and hence $v, i \vDash_{\sigma\cdot\sigma'} \neg[\varphi]_-$ since $0 \notin I$.
  - If $p = -$ and φ is typed using UntilSup, then $\Gamma \vdash \varphi_2 : \mathsf{Sup}$. By $P(\varphi_2)$, we get $\forall j \in I + i.\ v, j \vDash_{\sigma\cdot\sigma'} \neg[\varphi_2]$. Hence $v, i \vDash_{\sigma\cdot\sigma',-} [\varphi]_-$.

For $\sigma = \varepsilon$ and $p = +$, we get as a corollary:

Lemma 13. *For any $\varphi \in$ EMFOTL, there exists σ such that $\sigma \in \mathcal{L}([\varphi]_+)$.*

Our choice of σ' in the proof of Lemma 12 depends on Γ and σ, but not on the specific formula φ and valuation v. We thus obtain the following generalization:

Lemma 14. *Fix $\sigma \in \mathbb{T}_f$, $\tau' \geq \mathsf{lts}(\sigma)$, and Γ. Let X be a set of triples (ξ, v, p') all satisfying $\Gamma \vdash \xi(\tau') : \{\mathsf{Cau}\}_{p'}$ and $v(\mathsf{fv}(\xi(\tau'))) \subseteq \mathsf{AD}_{|\sigma|}(\varphi)$. Then there exists σ' such that $\mathsf{fts}(\sigma') = \tau'$ and for all $(\xi, v, p') \in X$, $v, |\sigma| \vDash_{\sigma\cdot\sigma'} p'[\xi(\tau')]_{p'}$.*

Proof. As in Lemma 12, repeating the same proof for each of the triples $(\xi(\tau'), v, p')$.

Lemma 15. *Assume that $\Gamma \vdash \varphi : \{\mathsf{Cau}\}_p$. If $\mathsf{enf}^p_{\tau,b}(\varphi, \sigma \cdot \langle(\tau, D)\rangle, X', v) = (S, C, X)$ and $(\xi, v, p') \in X$, then for all $\tau' \in \{\tau, \tau+1\}$ such that $\tau' = \tau + 1 \Longrightarrow b$, we have $\Gamma \vdash \xi(\tau') : \{\mathsf{Cau}\}_{p'}$.*

Proof. By systematic inspection of Algorithm 2, we see that whenever $\mathsf{enf}^{p_2}(\varphi_2, ...)$ is called recursively in $\mathsf{enf}^{p_1}(\varphi_1, ...)$ and $\Gamma \vdash \varphi_1 : \{\mathsf{Cau}\}_{p_1}$, then $\Gamma \vdash \varphi_2 : \{\mathsf{Cau}\}_{p_2}$. Now assume that some $\mathsf{enf}^{p'}(\varphi', ...)$ with $\Gamma \vdash \varphi' : \{\mathsf{Cau}\}_{p'}$ adds a new triple to X. There are four cases:

- If $\varphi' = \bigcirc_I \varphi_1$, then $\xi(\tau') = (\neg \mathtt{TP})\, \mathsf{U}_{I-(\tau'-\tau)}\, (\mathtt{TP} \wedge \varphi_1)$ if $\tau' - \tau \leq \sup I$, and $\xi(\tau') = \bot$ otherwise. There are two cases:
  - If $p = -$, then $\Gamma \vdash \varphi' : \mathsf{Sup}$ implies $\Gamma \vdash \varphi_1 : \mathsf{Sup}$. We can use UntilSup, AndSupR, and $\Gamma \vdash \varphi_1 : \mathsf{Sup}$ to obtain $\Gamma \vdash \xi(\tau') : \mathsf{Sup}$ if $\tau' - \tau \leq \sup I$. If $\tau' - \tau > \sup I$, then we obtain the same conclusion using rule False.
  - If $p = +$, then $\Gamma \vdash \varphi' : \mathsf{Cau}$ implies $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ and $I = [0, b)$ with $b > 0$ (rule NextCau). Since $\tau' \in \{\tau, \tau + 1\}$, we have $\tau' - \tau \leq 1 \leq \sup I$. Hence $\xi(\tau') = (\neg \mathtt{TP})\, \mathsf{U}_{I-(\tau'-\tau)}\, (\mathtt{TP} \wedge \varphi_1)$, and we obtain $\Gamma \vdash \varphi' : \mathsf{Cau}$ by using UntilCauR, AndCau, and $\mathtt{TP} \in \mathsf{Cau}$.
- If $\varphi' = \varphi_1\, \mathsf{U}_I\, \varphi_2$, then $\xi(\tau') = (\mathtt{TP} \rightarrow \varphi_1)\, \mathsf{U}_{I-(\tau'-\tau)}\, (\mathtt{TP} \wedge \varphi_1)$ if $\tau' - \tau \leq \sup I$, and $\xi(\tau') = \bot$ otherwise. There are again two cases:
  - If $p = -$, then $\Gamma \vdash \varphi' : \mathsf{Sup}$ implies $\Gamma \vdash \varphi_2 : \mathsf{Sup}$ (rule UntilSup). We can use UntilSup, AndSupR, and $\Gamma \vdash \varphi_2 : \mathsf{Sup}$ to obtain $\xi(\tau') \vdash \varphi' : \mathsf{Sup}$ if $\tau' - \tau \leq \sup I$. If $\tau' - \tau > \sup I$, then we obtain the same conclusion using rule False.
  - If $p = +$, we observe that the future obligation can only be in X if $I \neq [0, 0]$ or b is false. This implies that $\sup I \geq 1$ or $\tau' - \tau = 0$, and hence $\tau' - \tau \leq \sup I$ and $\xi(\tau') = (\mathtt{TP} \rightarrow \varphi_1)\, \mathsf{U}_{I-(\tau'-\tau)}\, (\mathtt{TP} \wedge \varphi_2)$. If $\Gamma \vdash \varphi' : \mathsf{Cau}$ is obtained using UntilCauLR, then we have $\Gamma \vdash \varphi_1 : \mathsf{Cau}$ and $\Gamma \vdash \varphi_2 : \mathsf{Cau}$. We prove that $\Gamma \vdash \xi(\tau') : \mathsf{Cau}$ using $\mathtt{TP} \in \mathsf{Cau}$ and rules UntilCauLR, ImpCauR (defined Appendix A), and AndCau. If $\Gamma \vdash \varphi' : \mathsf{Cau}$ is obtained using UntilCauR, then $\Gamma \vdash \varphi_2 : \mathsf{Cau}$ and $I = [0, b]$ for some $b \in \mathbb{N}$. We prove that $\Gamma \vdash \xi(\tau') : \mathsf{Cau}$ using $\mathtt{TP} \in \mathsf{Cau}$, the fact that $I - [0, b] = [0, b - (\tau' - \tau)]$, and rules UntilCauR and AndCau.

Lemma 16. *Let Γ, I, φ_1, φ_2, and p such that $\Gamma \vdash \varphi_1\, \mathsf{U}_I\, \varphi_2 : \{\mathsf{Cau}\}_p$. If $\sup I \geq 1$, then $\Gamma \vdash \varphi_1\, \mathsf{U}_{I-1}\, \varphi_2 : \{\mathsf{Cau}\}_p$.*

Proof. If $\sup I \geq 1$, then $I - 1$ is a non-empty interval. The proof is by case distinction on the rule used to derive $\Gamma \vdash \varphi_1\, \mathsf{U}_I\, \varphi_2$.

- UntilSup, $p = -$: We have $\Gamma \vdash \varphi_2 : \mathsf{Sup}$, hence by applying UntilSup again we get $\Gamma \vdash \varphi_1\, \mathsf{U}_{I-1}\, \varphi_2$.
- UntilCauR, $p = +$: We have $\Gamma \vdash \varphi_2 : \mathsf{Cau}$, and $I = [0, b]$ for some $b \in \mathbb{N}$. Hence $I - 1 = [0, b-1]$. By applying UntilCauR again we get $\Gamma \vdash \varphi_1 \mathsf{U}_{I-1} \varphi_2$.
- UntilCauLR, $p = +$: We have $\Gamma \vdash \varphi_1 : \mathsf{Cau}$, $\Gamma \vdash \varphi_2 : \mathsf{Cau}$, and $\sup I \neq \infty$. By applying UntilCauLR again we get $\Gamma \vdash \varphi_1\, \mathsf{U}_{I-1}\, \varphi_2$.

Lemma 17. *Assume that $\Gamma \vdash \varphi : \mathsf{Cau}$. Let X be the state of the enforcer at the beginning of the kth iteration of run, σ be the trace produced in the first $k - 1$ iterations, and ts the timestamp in the kth iteration. Then $\forall (\xi, v, p) \in X. \Gamma \vdash \xi(ts) : \{\mathsf{Cau}\}_p$.*

Proof. By induction on k. If $k = 0$, then $X = \{(\lambda\_.\, \varphi, \emptyset, +)\}$ and the result if trivial. Let $k > 0$ such that the property holds for $k-1$. Let X' and ts' be the state of the enforcer and the timestamp at the beginning of the $k-1$st iteration, respectively. There are two cases:

- If the $k-1$st iteration returned NoCom, then $X' = X$, $ts' = ts+1$, and b was true in iteration $k-1$. Moreover, there exists C and S such that $(C, S, X) = \mathsf{enf}^+_{ts,b}(\Phi, \sigma, \emptyset, \emptyset)$ and $\mathtt{TP} \notin C$. By systematic inspection of Algorithm 2, we see that none of the ξ in X' is such that $\xi(ts') = \varphi_1 \,\mathsf{U}_{I-(ts'-\tau)}\, \varphi_2$ for some φ_1, φ_2, and τ such that $I - (ts' - \tau) = [0, 0]$ (otherwise, $\mathtt{TP}$ would have been caused at $k-1$ and the returned command would not have been NoCom). Hence, all future obligations in X have a first component of the form $\xi = \lambda\tau'.\, \varphi_1 \,\mathsf{U}_{I-(\tau'-\tau)}\, \varphi_2$ where $\sup I \geq ts' - \tau + 1$. By Lemma 16 and $ts' = ts + 1$, this guarantees that $\forall(\xi, v, p) \in X.\, \Gamma \vdash \xi(ts') : \{\mathsf{Cau}\}_p$.
- Else, there exists C and S such that $(C, S, X) = \mathsf{enf}^+_{ts,b}(\Phi, \sigma, \emptyset, \emptyset)$, where $\Phi = \bigwedge_{(\xi,v,+)\in X'} \xi(ts')[v] \wedge \bigwedge_{(\xi,v,-)\in X'} \neg\xi(ts')[v]$ or $\Phi = \mathtt{TP} \wedge \bigwedge_{(\xi,v,+)\in X'} \xi(ts')[v] \wedge \bigwedge_{(\xi,v,-)\in X'} \neg\xi(ts')[v]$. Using the IH and rules AndCau and NotCau, we get $\Gamma \vdash \Phi : \mathsf{Cau}$. Hence, by Lemma 15, $\mathsf{enf}^+_{ts,b}(\Phi, \sigma, \emptyset, \emptyset)$ yields $\forall(\xi, v, p) \in X.\, \Gamma \vdash \xi(ts) : \{\mathsf{Cau}\}_p$.

Lemma 18. *Let Φ be the formula computed by* enf *at the beginning of the kth iteration of* run*, ts and ts' the timestamps in the kth and $k+1$st iterations respectively, σ the trace produced in the first $k-1$ iterations, and X the state of the formula at the end of the kth iteration. We have*

$$\forall D, \sigma'.\ \left(\forall(\xi, v, p) \in X.\, v, |\sigma| \vDash_{\sigma\cdot\overline{(ts,D)\cdot\sigma'}^{\mathtt{TP}}} p[\xi(ts')]_p\right) \implies v, |\sigma| - 1 \vDash_{\sigma\cdot\overline{(ts,D)\cdot\sigma'}^{\mathtt{TP}}} [\Phi]_+.$$

Proof. Let X' be the state of the enforcer at the beginning of the kth iteration of run. By Lemma 17, we have $\forall(\xi, v, p) \in X'.\, \Gamma \vdash \xi(ts) : \{\mathsf{Cau}\}_p$, and hence, as above, $\Gamma \vdash \Phi : \mathsf{Cau}$. The definition of run ensures that b is set to true whenever $ts' > ts$. Moreover, as above, $\Gamma \vdash \Phi : \mathsf{Cau}$. Using Lemma 10, we conclude that $v, |\sigma| - 1 \vDash_{\sigma\cdot\overline{(ts,D)\cdot\sigma'}^{\mathtt{TP}}} [\Phi]_+$ holds.

Lemma 19. *Assume that $\Gamma \vdash \varphi : \mathsf{Cau}$. Let X be the state of the enforcer at the beginning of the kth iteration of* run*, σ be the trace produced in the first $k-1$ iterations, and ts be the timestamp in the kth iteration of* run*. Then there exists σ' such that $\forall(\xi, v, p) \in X.\, v, |\sigma| \vDash_{\sigma\cdot\sigma'} p[\xi(ts)]_p$.*

Proof. Straightforward from Lemmata 14 and 17.

Lemma 20. *Let X be the state of the enforcer at the beginning of the kth iteration of* run*. For any $(\xi, v, p) \in X$, $\sigma \in \mathbb{T}_\omega$, $i, ts \in \mathbb{N}$, and φ any subformula of $\xi(ts')$, the truth value of $v, i \vDash_\sigma \varphi$ does not depend on the presence of any* $\mathtt{TP}$ *events occurring at time-points $0, \ldots, i-1$ in σ. As a corrolary, we have $v, i \vDash_{\sigma|_{..i-2}\cdot\overline{\sigma|_{i-1..}}^{\mathtt{TP}}} \varphi \iff v, i \vDash_{\sigma|_{..i-1}\cdot\overline{\sigma|_{i..}}^{\mathtt{TP}}} \varphi$.*

Proof. In every iteration, TP predicates are only inserted in $\mathsf{fo}_{\tau,\bigcirc,I,\varphi_1}$ and $\mathsf{fo}_{\tau,\mathsf{U},I,\varphi_1,\varphi_2}$, where they appear above all past operators. In every call to μ, the formula Φ is process top-down, generating future obligations only when reaching a future operator. Hence, one can show by induction that all TP predicates in Φ occur above all past operators. This implies the desired property.

Lemma 21. *Let $k \geq 1$. Let σ the trace produced in the first $k-1$ iterations (1-indexed) of* run*, ts and ts' the timestamps in the $k-1$st iteration (or 0 if $k=0$) and the kth iteration, respectively, and X the state of the formula at the beginning of the kth iteration. We have*

$$\begin{aligned}&\forall D, \sigma'.\ \Big(\forall(\xi, v, p) \in X.\, v, |\sigma| \vDash_{\sigma\cdot\overline{(ts',D)\cdot\sigma'}^{\mathtt{TP}}} p[\xi(ts')]_p\Big)\\ \implies\ &\sigma \cdot (ts', D) \cdot \sigma' \in \mathcal{L}([\varphi]_+).\end{aligned}$$

Proof. We prove

$$\begin{aligned}&\forall k \in \mathbb{N}.\, P(k) \equiv\\ &\forall D, \sigma'.\ \Big(\forall(\xi, v, p) \in X_k.\, v, |\sigma_k| \vDash_{\sigma_k\cdot\overline{(ts_k,D)\cdot\sigma'}^{\mathtt{TP}}} p[\xi(ts_k)]_p\Big)\\ &\qquad \implies \sigma_k \cdot (ts_k, D) \cdot \sigma' \in \mathcal{L}([\varphi]_+)\end{aligned}$$

where X_k, σ_k, and ts_k denote the state of the enforcer, the already generated trace prefix, and the timestamp at the beginning of the kth iteration. The proof is by induction on k.

If $k = 1$, then $\sigma_1 = \varepsilon$, $ts_0 = \tau_0$, and $X_1 = \{(\lambda\_.\, \varphi, v, +)\}$. Let D, and σ'. The LHS of the implication to be proven is $v, 0 \vDash_{\overline{(ts_0,D)\cdot\sigma'}^{\mathtt{TP}}} [\varphi]_+$. As φ does not contain any TP event, then $v, 0 \vDash_{(ts_0,D)\cdot\sigma'} [\varphi]_+$. This is exactly $(ts_0, D) \cdot \sigma' \in \mathcal{L}([\varphi]_+)$.

Let $k > 1$ such that $P(k-1)$ holds. Let D, and σ' such that

$$\forall(\xi, v, p) \in X_k.\, v, |\sigma_k| \vDash_{\sigma_k\cdot\overline{(ts_k,D)\cdot\sigma'}^{\mathtt{TP}}} p[\xi(ts_k)]_p \qquad (*)$$

holds. There are two cases:

- If enf did not return NoCom in the $k-1$st iteration, then there exists $\hat{D}$ such that $\sigma_k = \sigma_{k-1} \cdot (ts_{k-1}, \hat{D})$. By Lemma 18, we get

  $$\begin{aligned}&\forall D', D'', \sigma''.\\ &\Big(\forall(\xi, v, p) \in X_k.\, v, |\sigma_{k-1}| + 1 \vDash_{\sigma_{k-1}\cdot\overline{(ts_{k-1},\hat{D})\cdot(ts_k,D'')\cdot\sigma''}^{\mathtt{TP}}} p[\xi(ts_k)]_p\Big)\\ &\implies v, |\sigma_{k-1}| \vDash_{\sigma_{k-1}\cdot\overline{(ts_{k-1},\hat{D})\cdot(ts_k,D'')\cdot\sigma''}^{\mathtt{TP}}} [\Phi_{k-1}]_+\end{aligned}$$

  where Φ_{k-1} is the formula Φ computed at the beginning of the $k-1$st iteration. Setting $D'' := D$ and $\sigma'' := \sigma'$, we obtain

  $$\begin{aligned}&\Big(\forall(\xi, v, p) \in X_k.\, v, |\sigma_{k-1}| + 1 \vDash_{\sigma_{k-1}\cdot\overline{(ts_{k-1},\hat{D})\cdot(ts_k,D)\cdot\sigma'}^{\mathtt{TP}}} p[\xi(ts_k)]_p\Big)\\ &\implies v, |\sigma_{k-1}| \vDash_{\sigma_{k-1}\cdot\overline{(ts_{k-1},\hat{D})\cdot(ts_k,D)\cdot\sigma'}^{\mathtt{TP}}} [\Phi_{k-1}]_+.\end{aligned}$$

By $(*)$, equation $\sigma_k = \sigma_{k-1} \cdot (ts_{k-1}, \hat{D} \setminus S \cup C)$, and Lemma 20, we obtain

$$\forall (\xi, v, p) \in X_k.\, v, |\sigma_{k-1}| + 1 \vDash_{\sigma_{k-1} \cdot \overline{(ts_{k-1}, \hat{D}) \cdot (ts_k, D) \cdot \sigma'}^{\text{TP}}} p[\xi(ts_k)]_p.$$

Moreover,

$$\begin{aligned} & v, |\sigma_{k-1}| \vDash_{\sigma_{k-1} \cdot \overline{(ts_{k-1}, \hat{D}) \cdot (ts_k, D) \cdot \sigma'}^{\text{TP}}} [\Phi_{k-1}]_+ \\ \iff & \forall (\xi, v, p) \in X_{k-1}.\, v, |\sigma_{k-1}| \vDash_{\sigma_{k-1} \cdot \overline{(ts_{k-1}, \hat{D}) \cdot (ts_k, D) \cdot \sigma'}^{\text{TP}}} p[\xi(ts_{k-1})]_p. \end{aligned}$$

given the construction of Φ using $\wedge^{\text{AndCau}}$, $\neg^{\text{NegCau}}$, and substitution. From $P(k-1)$ and the last two equations, we conclude that

$$\sigma_k \cdot (ts_k, D) \cdot \sigma' = \sigma_{k-1} \cdot (ts_{k-1}, \hat{D}) \cdot (ts_k, D) \cdot \sigma' \in \mathcal{L}([\varphi]_+).$$

– If enf returned NoCom in the $k-1$st iteration, then $\sigma_k = \sigma_{k-1}$. By Lemma 18, we get

$$\begin{aligned} & \forall D', D'', \sigma''. \\ & \Big(\forall (\xi, v, p) \in X_k.\, v, |\sigma_{k-1}| + 1 \vDash_{\sigma_{k-1} \cdot \overline{(ts_k, D'') \cdot \sigma''}^{\text{TP}}} p[\xi(ts_k)]_p\Big) \\ & \implies v, |\sigma_{k-1}| \vDash_{\sigma_{k-1} \cdot \overline{(ts_k, D'') \cdot \sigma''}^{\text{TP}}} [\Phi_{k-1}]_+ \end{aligned}$$

where Φ_{k-1} is the formula Φ computed at the beginning of the $k-1$st iteration. Setting $D'' := D$, and $\sigma'' := \sigma'$, we obtain

$$\begin{aligned} & \Big(\forall (\xi, v, p) \in X_k.\, v, |\sigma_{k-1}| + 1 \vDash_{\sigma_{k-1} \cdot \overline{(ts_k, D) \cdot \sigma'}^{\text{TP}}} p[\xi(ts_k)]_p\Big) \\ & \implies v, |\sigma_{k-1}| \vDash_{\sigma_{k-1} \cdot \overline{(ts_k, D) \cdot \sigma'}^{\text{TP}}} [\Phi_{k-1}]_+. \end{aligned}$$

Moreover,

$$\begin{aligned} & v, |\sigma_{k-1}| \vDash_{\sigma_{k-1} \cdot \overline{(ts_k, D) \cdot \sigma'}^{\text{TP}}} [\Phi_{k-1}]_+ \\ \iff & \forall (\xi, v, p) \in X_{k-1}.\, v, |\sigma_{k-1}| \vDash_{\sigma_{k-1} \cdot \overline{(ts_k, D) \cdot \sigma'}^{\text{TP}}} p[\xi(ts_{k-1})]_p. \end{aligned}$$

given the construction of Φ using $\wedge^{\text{AndCau}}$, $\neg^{\text{NegCau}}$, and substitution. From $P(k-1)$ and the last two equations, we conclude that

$$\sigma_k \cdot (ts_k, D) \cdot \sigma' = \sigma_{k-1} \cdot (ts_k, D) \cdot \sigma' \in \mathcal{L}([\varphi]_+).$$

Lemma 22. *Assume that $\varphi \in$ EMFOTL. Let $\sigma \in \mathbb{T}_f$ be a prefix of $\mathcal{E}_\varphi(\sigma)$. There exists σ' such that $\sigma \cdot \sigma' \in \mathcal{L}(\varphi)$.*

Proof. If $\sigma = \varepsilon$, Lemma 13 provides the desired property.

If $|\sigma| = k > 0$, then by Lemmata 18 and 21, it is sufficient to find some $ts' \geq ts$, D, and σ' such that $\forall (\xi, v, p) \in X.\, v, |\sigma| \vDash^{\varphi}_{\sigma \cdot \overline{(ts', D) \cdot \sigma'}^{\text{TP}}} p[\xi(ts')]_p$, where X is the state of the enforcer at the end of the iteration producing the last time-point in σ. This is exactly what Lemma 19 guarantees, hence the conclusion.

We finally conclude:

Theorem 1. *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.*

Proof. By Lemmata 8 and 22.

B.5 Transparency (Section 5.4)

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

Proof. Let P be a property and $\mathcal{E}$ be a transparent enforcer for φ. Let $\sigma \in \mathbb{T}_\omega \setminus P$. Since $\mathcal{E}$ is an enforcer for φ, then $\mathcal{E}(\sigma) \in P$ and hence $\mathcal{E}(\sigma) \neq \sigma$. Set $i \in \mathbb{N}$ such that $\mathcal{E}(\sigma)|_i \neq \sigma|_i$. Observe that since enforcers can neither suppress time-points nor modify the past, the value of $\mathcal{E}(\sigma)|_i$ was computed using information from $\sigma|_{..i}$ only. Therefore, for any $\sigma' \in \mathbb{T}_\omega$, we have $\mathcal{E}(\sigma|_i \cdot \sigma')|_i = \mathcal{E}(\sigma)|_i$. Hence, $\mathcal{E}(\sigma|_i \cdot \sigma')|_i \neq (\sigma|_i \cdot \sigma')|_i$ and finally $\mathcal{E}(\sigma|_i \cdot \sigma') \neq \sigma|_i \cdot \sigma'$. By the transparency of $\mathcal{E}$, we conclude that $\sigma|_i \cdot \sigma' \notin P$. Hence P is a safety property.

Theorem 2. *If $\varphi \in$ EMFOTL, the enforcer $\mathcal{E}_\varphi$ is transparent w.r.t. $\mathcal{L}([\varphi]_+)$.*

Proof. Fix $\sigma \in \mathcal{L}([\varphi]_+])$. We prove by induction on k that at every iteration k (1) the goal Φ_k computed at the begin of iteration k satisfies $\emptyset, |\sigma_k| \vDash_{\overline{\sigma}^{\mathtt{TP}}} [\Phi_k]_+$, where σ_k is the trace produced in the first $k-1$ iterations and (2) the trace is not modified by the enforcer (no causation, suppression, or insertion of time-points) in the kth iteration. For $k = 0$, $\Phi_k = \mathtt{TP} \wedge \varphi$ and (1) is trivial by our choice of σ. Given (1), a straightforward induction on the structure of φ shows that no events are caused or suppressed and that all generated future obligations, evaluated with the second timestamp in σ, are satisfied on σ at $|\sigma_k| + 1 = 1$. This proves (2). For $k > 0$, (1) is obtained through the same argument about the generated future obligations at $k-1$, observing that when all future obligations generated in iteration $k-1$ are satisfied at time-point $|\sigma_k|$, then Φ_k is satisfied at time-point $|\sigma_k|$, too. Using (1), (2) is proved as in the previous case.

C Transparently enforceable subset of EMFOTL

Theorem 2 guarantees that for any $\varphi \in$ EMFOTL, the enforcer $\mathcal{E}_\varphi$ transparently enforces $[\varphi]_+$ and hence, any $\varphi \in$ EMFOTL such that $[\varphi]_+ = \varphi$ is transparently enforceable. In this section, we describe TEMFOTL, a syntactically defined fragment of EMFOTL satisfying this condition and transparently enforced by our algorithm. WhyEnf's type-checker also checks membership in TEMFOTL, issuing a warning when transparent enforcement cannot be guaranteed.

To define TEMFOTL, we use the notion of *strictly relative-past formulae* introduced by Hublet et al. [36]. Strictly relative-past formulae constitute a syntactically defined subset of MFOTL. The truth value of such formulae at a given point in time only depends on events that happened before that time, i.e., all strictly relative-past formulae are *future-free*:

Definition 7 (adapted from [35]). *An MFOTL formulae is* future-free *iff for all trace prefix $\sigma \in \mathbb{T}_f$, valuation v, and $i = |\sigma| - 1$, we have $\forall \sigma_1, \sigma_2.\ v, i \vDash_{\sigma \cdot \sigma_1} \varphi \iff \forall \sigma_1, \sigma_2.\ v, i \vDash_{\sigma \cdot \sigma_2} \varphi$.*

The set of strictly relative-past formulae contains all past-only formulae, but also some formulae with future operators that can only be evaluated on past

time-points. For example, the formula $\blacklozenge_{[10,20]} \lozenge_{[0,5]} \mathsf{use}(c, d, u)$ is strictly relative-past, although it contains a future operator. In the following, SRP denotes the set of strictly relative-past formulae. The reader is referred to Hublet et al.'s work [36] for details.

TEMFOTL is defined as the fragment of EMFOTL containing all formulae that are typable under the following stricter typing rules (the typing rules from Figure 3 not listed below being unchanged):

$$\frac{\Gamma \vdash \varphi : \mathsf{Sup} \quad \psi \in \mathsf{SRP}}{\Gamma \vdash \varphi \wedge \psi : \mathsf{Sup}}\ \textsc{AndSupL} \qquad \frac{\Gamma \vdash \psi : \mathsf{Sup} \quad \varphi \in \mathsf{SRP}}{\Gamma \vdash \varphi \wedge \psi : \mathsf{Sup}}\ \textsc{AndSupR}$$

$$\frac{0 \in I \quad \Gamma \vdash \psi : \mathsf{Cau} \quad \varphi, \psi \in \mathsf{SRP}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{Cau}}\ \textsc{SinceCau} \qquad \frac{0 \notin I \quad \Gamma \vdash \varphi : \mathsf{Sup} \quad \varphi, \psi \in \mathsf{SRP}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{Sup}}\ \textsc{SinceSupL}$$

$$\frac{0 \in I \quad \Gamma \vdash \varphi, \psi : \mathsf{Sup} \quad \varphi, \psi \in \mathsf{SRP}}{\Gamma \vdash \varphi\, \mathsf{S}_I\, \psi : \mathsf{Sup}}\ \textsc{SinceSupLR} \qquad \frac{b \neq \infty \quad \Gamma \vdash \psi : \mathsf{Cau} \quad \varphi \in \mathsf{SRP}}{\Gamma \vdash \varphi\, \mathsf{U}_{[0,b]}\, \psi : \mathsf{Cau}}\ \textsc{UntilCauR}$$

$$\frac{\Gamma \vdash \psi : \mathsf{Sup} \quad \varphi \in \mathsf{SRP}}{\Gamma \vdash \varphi\, \mathsf{U}_I\, \psi : \mathsf{Sup}}\ \textsc{UntilSup}$$

Fig. 12. Modified typing rules for TEMFOTL

With the future-free property of strictly relative-past formulae, we can show that

Lemma 23. *For all $\varphi \in$ TEMFOTL and $p \in \{+, -\}$, we have $[\varphi]_p \equiv \varphi$.*

Proof. By straightforward induction on φ, using the definition of $[\bullet]_p$ and the future-freeness assumption.

D Additional graphs

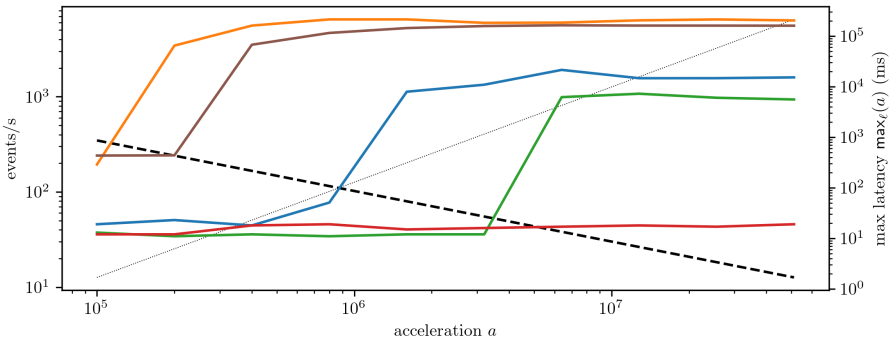

Fig. 13. WhyMon's latency and event rate for the formulae in Figure 7, except φ_{lim}.

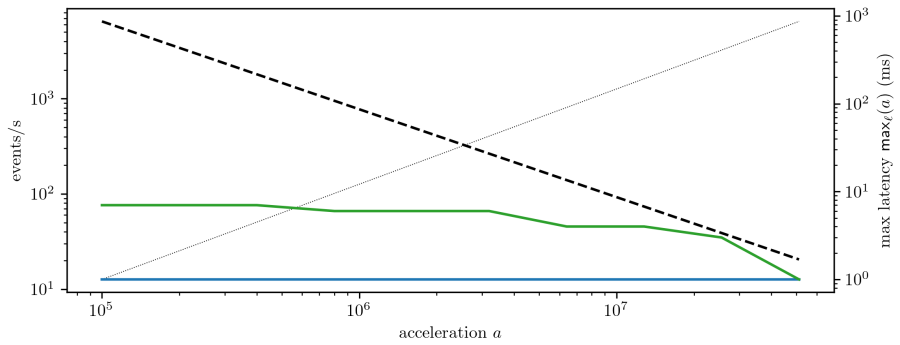

Fig. 14. EnfPoly's latency and event rate for the formulae φ_{law} and φ_{con} of Figure 7.

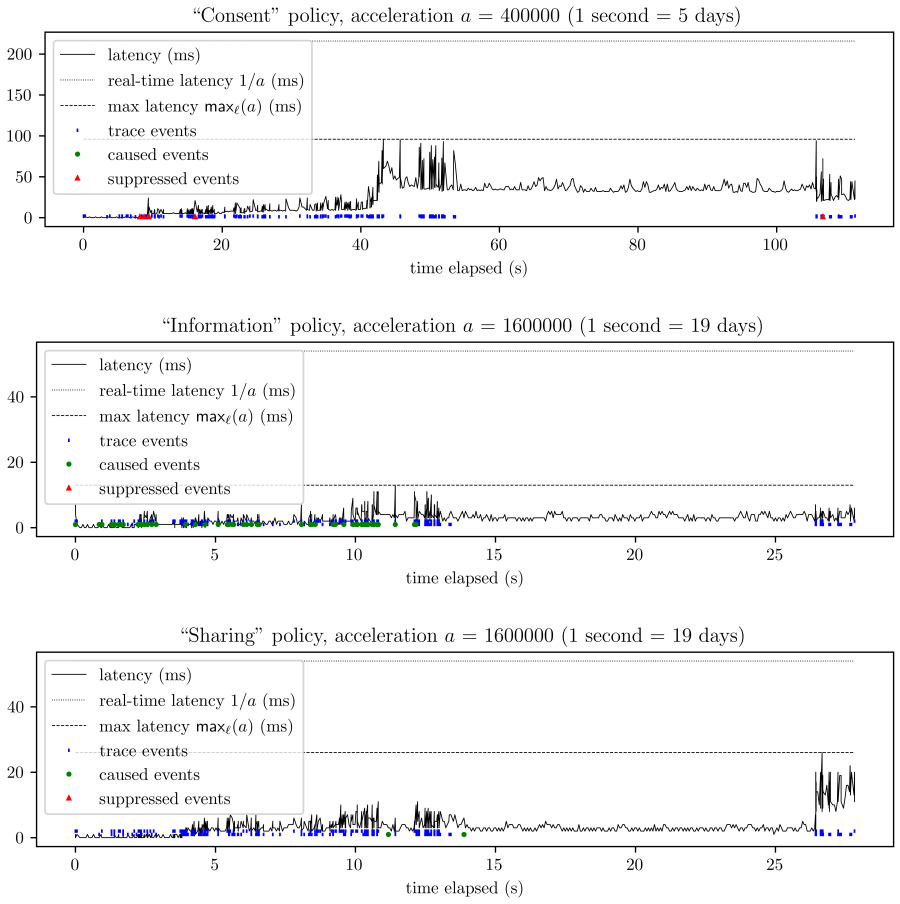

Fig. 15. WhyEnf's latency when enforcing φ_{con}, φ_{inf}, and φ_{sha} over the log [24].