Real-time Policy Enforcement with Metric First-Order Temporal Logic (Ext. rep.)

François Hublet, David Basin, Srđan Krstić

Institute of Information Security, Department of Computer Science, ETH Zürich, Switzerland

Abstract. Correctness and regulatory compliance of today's software systems are crucial for our safety and security. This can be achieved with policy enforcement: the process of correcting system behavior to satisfy a given policy. The enforcer's capabilities determine which policies are enforceable.
We study the enforceability of policies specified in metric first-order temporal logic (MFOTL) with enforcers that can cause and suppress different system actions in real time. We show that a formula from an expressive safety fragment of MFOTL is enforceable if and only if it is equivalent to a formula in a simpler, syntactically defined MFOTL fragment. We propose an enforcement algorithm for all monitorable formulae (i.e., formulae whose violations can be detected by manipulating finite sets of satisfying valuations) from the latter fragment, and show that our EnfPoly enforcer tool outperforms state-of-the-art enforcers.

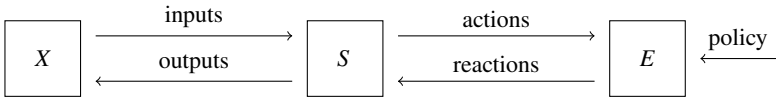
1 Introduction

Modern software systems are increasingly complex, ubiquitous and intransparent. In this context, allowing individuals to scrutinize and control the systems that affect their daily lives is an important technical and societal challenge. One way to achieve this goal is to develop systems that can *monitor* and *control* other, target systems, by enforcing *policies* that describe the acceptable target system's behaviors.

Policy enforcement [56] (Figure 1) is a form of execution monitoring where a system, called an *enforcer*, observes a target system's actions, *detects* attempted policy violations, and reacts to *prevent* them. In contrast, policy monitoring (or runtime verification) [8,29] provides *monitors* that only passively detect policy violations by the target system. Both problems have an offline and an online variant: the former considers only a trace of recorded target system actions, while the latter observes the target system in real time.

Policy enforcement has been studied in different communities (Section 2) like controller synthesis [1,48,49], security [5,56], and operating systems [53,54], each defining and solving the problem in a different, more specialized context.

Fig. 1: Policy enforcement



An enforcer E observes actions in a target system S and reacts (e.g., causes or suppresses some actions in S) to ensure policy compliance. S interacts with an environment X, which E cannot control.

Schneider [56] studied the general form of the policy enforcement problem in the context of security. He proposed security automata as enforcers that prevent policy violations by simply terminating the target system. Schneider, and later others [13], identified classes of policies that were *enforceable* using such an enforcer. As policy enforceability depends on the enforcer's powers (e.g., its ability to suppress, cause, or delay system's actions) over the target system, other enforceable policy classes were suggested [11].

Automata and temporal logic are popular formalisms for policy specification. Existing security policy enforcers typically focus on propositional policies expressed as variants of (security, edit, or timed) automata [30]. Controller synthesis tools, on the other hand, often enforce specifications expressed in LTL [18] or in (fragments of) metric temporal logics [19, 21, 35, 42]. However, propositional temporal logic is limited in its expressiveness: like automata, it regards system actions as atomic and thus cannot formulate dependencies between the data values coming from an infinite domain that the actions may carry as parameters. For instance, data values may encode personally identifiable data items: then each system action that uses an item must be preceded by an action that receives a consent for the item's particular use [7]. To the best of our knowledge, there is no tool that supports enforcement of first-order logic specifications.

In this paper, we consider online policy enforcement (Section 3) of policies expressed in metric first-order temporal logic (MFOTL) [22], which extends LTL with metric constraints and first-order quantification (Section 4). To enforce MFOTL policies, our enforcer can observe the target system in real time, actively cause or suppress certain types of actions (but no action can be both caused and suppressed), and only observe all other actions of the target system. All actions, caused either by the enforcer or the target system, are instantaneous and tagged with a timestamp. The restriction that no action can both be caused and suppressed makes it easier to check enforceability of MFOTL formulae and develop an efficient enforcer.

To make enforcement practical, we consider restrictions to two "well-behaved" fragments of MFOTL: (1) we study enforceability of $\text{MFOTL}^{\mathcal{F}}_{\Box}$, a safety fragment of MFOTL comprising closed formulae of the form $\Box\varphi$ ("always φ") where φ's satisfaction does not depend on future information; and (2) we take advantage of an existing efficient monitoring algorithm for MFOTL to design an enforcement algorithm for both monitorable and enforceable $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae. Violations of monitorable formulae [14] can be detected by manipulating only finite sets of satisfying valuations. Knowing that these sets are always finite allows us to use simple, yet efficient, data structures and reuse the existing, highly-optimized monitoring algorithm [15].

Overall, we characterize the enforceability of $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae by an enforcer with the ability to suppress or cause different system actions, and propose and implement an enforcer for monitorable $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae. We make the following contributions:

- For an enforcer with the ability to suppress or cause (disjoint sets of) actions, we characterize enforceability of $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae. We show that it is undecidable whether an $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formula is enforceable and propose an expressively complete syntactical approximation (Section 5).
- We develop an enforcement algorithm for monitorable $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae and prove its correctness (Section 6).

– Finally, we describe our enforcer's implementation (Section 7) and evaluate its time and memory usage against other state-of-the-art tools (Section 8).

2 Related work

We group related work into conceptual approaches and those that implement enforcers.

Theory. The policy enforcement problem was introduced by Schneider and Erlingsson in the context of security [27,56]. Schneider defined security automata, a class of Büchi automata, as enforcers. Violations were prevented by terminating the system. Bauer *et al.* [16] extended Schneider's work by considering enforcers that can suppress and cause events. Basin *et al.* [13] distinguished between suppressable and (only-)observable events and refined Schneider's enforceability accordingly, but only discussed enforcement via suppression. Falcone *et al.* [28] later studied the enforcement of propositional timed policies by suppressing and delaying events. Recently, Aceto *et al.* [2] proposed *bidirectional* enforcers that treat input and output system actions differently. We see this distinction as a more refined event type partition (Section 3).

Policy enforcement is closely related to the controller synthesis problem [48], where a controller ($\approx$ enforcer) wants to ensure compliance of a plant ($\approx$ system) with a specification ($\approx$ policy). Specification *realizability* corresponds to enforceability, while controller synthesis (i.e., generating an automaton from a specification) corresponds to generating an enforcement algorithm tailored for a particular specification. Our enforcer does not explicitly generate code for a specific policy, but rather takes a policy as input directly. Early work by Pnueli, Rosner and Abadi [1,48,49] studied LTL realizability and LTL (controller) synthesis problems. More efficient approaches later emerged [32,41,55], as well as synthesis techniques for metric extensions of LTL [19,21,35,42].

Tools. Policy enforcement approaches typically rely on different classes of automata both as enforcers and policies [13,16,24,27,28,30,31,43,44,47,51,52,56]. A recent survey [29] listed three enforcer tools: GREP [51], Proactive Libraries [52], and TiPEX [47]. Both GREP and TiPEX use timed automata as a specification language, and could thus support propositional temporal logics such as MITL [4] via conversion [20,45]. They do not, however, natively support temporal logic.

The state-of-the-art MonPoly tool [15] can detect violations of monitorable MFOTL policies [14]. Other tools for first-order temporal logics include Verimon [10,40,57] and DejaVu [33,34], none of which provides enforcement support for violation prevention.

Many controller synthesis tools have been developed for LTL like Lily [38], Unbeast [26], Acacia+ [18] and SSyft [59]. Further tools synthesize controllers for systems described by timed automata to comply with specifications written in TCTL [17,46], MTL [35], or its fragment $\text{MTL}_{0,\infty}$ [42]. BluSTL [25,50] is a MATLAB toolbox for generating controllers from signal temporal logic (STL) specifications.

3 Policy enforcement

We fix a signature $\Sigma = (\mathbb{D}, \mathbb{E}, a)$, containing an infinite set $\mathbb{D}$ of constant symbols, a finite set of *event names* $\mathbb{E}$, and an arity function $a(e) \in \mathbb{N}, e \in \mathbb{E}$. An *event* is a pair $(e, (d_1, \ldots, d_{a(e)})) \in \mathbb{E} \times \mathbb{D}^{a(e)}$ of an event name e and $a(e)$ arguments.

Events model system actions *observable* by the enforcer. While some of these observable events can also be controlled (i.e., suppressed or caused) by the enforcer, others can *only* be observed. To capture these different cases, we partition $\mathbb{E}$ into two sets: a set of controllable event names, and a set of only-observable event names. Among the controllable event names, we further distinguish between *suppressable* event names $\mathsf{Sup} \subseteq \mathbb{E}$ and *causable* event names $\mathsf{Cau} \subseteq \mathbb{E}$. The set of only-observable event names is $\mathsf{Obs} = (\mathbb{E} \setminus \mathsf{Sup}) \setminus \mathsf{Cau}$. In general, some controllable events might be both suppressable and causable. However, we will assume that no such events exist, i.e. $\mathsf{Sup} \cap \mathsf{Cau} = \emptyset$. The reason for this assumption will become apparent when we consider the enforcement of MFOTL policies (Section 6), and we will discuss ways in which it can be relaxed.

Example 1. As a running example, we use the signature $(\mathbb{N}, \{\texttt{Open}, \texttt{Close}, \texttt{Knock}\}, a)$, where $a(\cdot) = 1$, $\mathsf{Sup} = \{\texttt{Open}\}$, and $\mathsf{Cau} = \{\texttt{Close}\}$. The target system controls a set of doors indexed by integers, which an enforcer can mechanically close and/or keep closed, but not hold open. Each door i is equipped with a sensor that causes a $\texttt{Knock}(i)$ event whenever a human knocks on the door. `Knock` events are only-observable ($\mathsf{Obs} = \{\texttt{Knock}\}$), since they reflect the environment's behavior.

Given a signature Σ, we define the set of *(event) databases* $\mathbb{DB}^*$ as $2^{\{(e,d) \mid e \in \mathbb{E}, d \in \mathbb{D}^{a(e)}\}}$. Databases represent structures over Σ. We restrict ourselves to considering *automatic* databases, i.e., databases which can be represented by a collection of finite automata [39]. This setup is the most general one used for MFOTL monitoring in [14].

Definition 1 (Automatic event database). *An event database D is* automatic *iff for all $e \in \mathbb{E}$, $D \cap \{(e,d) \mid d \in \mathbb{D}^{a(e)}\}$ is a regular set. $\mathbb{DB}$ is the set of automatic event databases.*

Finally, for any $E \subseteq \mathbb{E}$, we denote by $\mathsf{Ev}(E)$ the set of all databases with event names in E only, i.e. $\mathsf{Ev}(E) := \{D \in \mathbb{DB} \mid \forall (e, (d_1, \ldots, d_{a(e)})) \in D.\ e \in E\}$.

Traces are finite or infinite sequences $(\tau_i, D_i)_{1 \leq i \leq k}, k \in \mathbb{N} \cup \{\infty\}$, where $\tau_i \in \mathbb{N}$ are nondecreasing timestamps, and $D_i \in \mathbb{DB}$ are databases. The smallest timestamp of a trace σ is denoted by $\mathsf{sts}(\sigma) = \tau_1 \in \mathbb{N}$, its largest timestamp is denoted by $\mathsf{lts}(\sigma) = \sup_{1 \leq i \leq k} \tau_i \in \mathbb{N} \cup \{\infty\}$. The empty trace is denoted by ε, the set of traces by $\mathbb{T}$, and the set of finite traces by $\mathbb{T}_f = \{\sigma \in \mathbb{T} \mid |\sigma| < \infty\}$. If σ, σ' are two traces such that σ is finite, $\sigma \cdot \sigma'$ denotes the concatenation of σ and σ'. A *(trace) property* is a subset $P \subseteq \mathbb{T}$. For all $\sigma, \sigma' \in \mathbb{T}$, we write $\sigma \preceq \sigma'$ iff σ is a prefix of σ', and denote by $\mathsf{pre}(\sigma)$ the set of all prefixes of σ. The *limit closure* of a set $A \subseteq \mathbb{T}$, denoted by $\mathsf{cl}(A)$, contains all traces whose finite prefixes are all in A, i.e., $\mathsf{cl}(A) = \{\sigma \in \mathbb{T} \mid \forall \sigma' \in \mathsf{pre}(\sigma).\ |\sigma'| < \infty \Rightarrow \sigma' \in A\}$. The *truncation* of A is $\mathsf{trunc}(A) = \{\sigma \in A \mid \mathsf{pre}(\sigma) \subseteq A\}$, the largest prefix-closed subset of A.

Finite databases $\mathbb{DB}^\dagger \subseteq \mathbb{DB}$ are a specific type of automatic databases. We also consider traces with finite databases $\mathbb{T}^\dagger \subseteq \mathbb{T}$, and finite traces with finite databases $\mathbb{T}_f^\dagger$.

We consider the following notion of safety [13] that differs from Alpern and Schneider's [3] by considering both finite and infinite traces. Considering finite traces is necessary, since practical enforcers work with finite traces.

Definition 2 (∞-safety, [13]). *A trace property is a* safety property *iff for all $\sigma \notin P$, there exists $\sigma' \preceq \sigma$ such that $\{\sigma' \cdot \sigma'' \mid \sigma'' \in \mathbb{T}\} \subseteq \mathbb{T} \setminus P$, i.e., a trace σ is not in P iff it has a "bad prefix" σ' which cannot be extended to a trace in P.*

Example 2. The property P_1 that expresses that "the trace contains no `Open` event" is a safety property. The bad prefixes here are all prefixes which contain an `Open` event. In contrast, the property P_2 requiring that "the trace contains some `Open` event" is not a safety property: any finite trace σ containing no `Open` event is not in P_2, but σ can always be extended to a trace $\sigma \cdot ((\tau, D)) \in P_2$ with $\tau \geq \mathsf{lts}(\sigma)$ and e.g. $(\texttt{Open}, (1)) \in D$.

We now extend the definition of enforceability [13] to support causable events.

Definition 3 (Enforceability). *A property $P \subseteq \mathbb{T}$ is enforceable iff there is a deterministic Turing machine (TM) $\mathcal{M}$ accepting a set of finite traces S such that*

(i) $\mathsf{cl}(\mathsf{trunc}(S)) = P$;
(ii) $\mathcal{M}$ *accepts* ε;
(iii) For all $\sigma \in \mathsf{trunc}(S)$, $\tau \geq \mathsf{lts}(\sigma)$, *and* $D \in \mathbb{DB}$, $\mathcal{M}$ *halts on* $\sigma \cdot ((\tau, D))$; *and*
(iv) For all $\sigma \in \mathsf{trunc}(S)$, $\tau \geq \mathsf{lts}(\sigma)$, *and* $D \in \mathbb{DB}$, *there exists* $S \in \mathsf{Ev}(\mathsf{Sup})$ *and* $C \in \mathsf{Ev}(\mathsf{Cau})$ *such that* $\mathcal{M}$ *accepts* $\sigma \cdot ((\tau, (D \setminus S) \cup C))$.

Properties are sets of infinite traces, while enforcers (that do not know the system's implementation) can only observe finite traces. Hence, an enforceable property must be checked "prefix-wise": a trace is in a property iff an enforcer accepts all of its prefixes. Enforceable properties must hold on the empty trace, i.e., the system must initially comply with the property. For any extension of a (non-violating) prefix, the enforcer must be able to decide on its compliance to the property. Whenever a valid prefix is extended with an additional database, there must exist sets of suppressable and causable events which the enforcer can respectively suppress and cause to ensure satisfaction of the property.

We have the following equivalent characterization, which is easier to manipulate in proofs, since it removes the need for existential quantification in (iv).

Lemma 1. *In Definition 3, condition (iv) can be replaced by*

(iv') For all $\sigma \in \mathsf{trunc}(S)$, $\tau \geq \mathsf{lts}(\sigma)$, *and* $O \in \mathsf{Ev}(\mathsf{Obs})$, $\mathcal{M}$ *accepts* $\sigma \cdot ((\tau, O \cup AC))$, *where* $AC = \{(e, (d_1, \ldots, d_{a(e)})) \mid e \in \mathsf{Cau}, (d_1, \ldots, d_{a(e)}) \in \mathbb{D}^{a(e)}\})$ *is the database containing all events with names in* Cau.

Proof. Let $P \subseteq \mathbb{T}$ and $\mathcal{M}$ such that (i–iii) are fulfilled. We show that (iv) from Definition 3 and (iv') are equivalent.

$\Rightarrow$ Assume (iv). Let $\sigma \in \mathsf{trunc}(S)$, $\tau \geq \mathsf{lts}(\sigma)$, and $O \in \mathsf{Ev}(\mathsf{Obs})$. Let AC as above. Consider $D = O \cup AC \in \mathbb{DB}$. By (iv), there exists $S \in \mathsf{Ev}(\mathsf{Sup})$ and $C \in \mathsf{Ev}(\mathsf{Cau})$ such that $\sigma \cdot ((\tau, O \cup AC \setminus S \cup C)) \in P$. Since $C \subseteq AC$ and $S \cap (O \cup AC) \subset \mathsf{Sup} \cap (\mathsf{Obs} \cup \mathsf{Cau}) = \emptyset$, we have $O \cup AC \setminus S \cup C = O \cup AC$. Therefore, $\sigma \cdot ((\tau, O \cup AC)) \in P$.

$\Leftarrow$ Assume (iv'). Let $\sigma \in \mathsf{trunc}(S)$, $\tau \geq \mathsf{lts}(\sigma)$ and $D \in \mathbb{DB}$. Let $O = D \cap \mathsf{Ev}(\mathsf{Obs})$, $S = D \cap \mathsf{Ev}(\mathsf{Sup})$ and $C = AC$ as above. By (iv'), we have $\sigma \cdot ((\tau, O \cup AC)) \in P$. But $O \cup AC = D \cap \mathsf{Ev}(\mathsf{Obs}) \cup AC = (D \setminus D \cap \mathsf{Ev}(\mathsf{Sup}) \setminus D \cap \mathsf{Ev}(\mathsf{Cau})) \cup AC = (D \setminus S \setminus D \cap \mathsf{Ev}(\mathsf{Cau})) \cup AC = D \setminus S \cup AC$ since $D \cap \mathsf{Ev}(\mathsf{Cau}) \subseteq \mathsf{Ev}(\mathsf{Cau}) \subseteq AC$. Hence, $\sigma \cdot ((\tau, D \setminus S \cup C)) \in P$.

This condition now requires that $\mathcal{M}$ accepts any trace obtained by extending an accepted trace with a database containing some only-observable events and *all* causable events. Note that the database containing all causable events is co-finite, thus automatic.

Our notion of enforceability implies safety:

Lemma 2. *Any enforceable property $P \subseteq \mathbb{T}$ is a safety property.*

Proof. Let $\mathcal{M}$ and S be as in Lemma 1. Let $\sigma \in \mathbb{T} \setminus P$. Since $P = \mathsf{cl}(\mathsf{trunc}(S))$, there exists a finite $\sigma' \preceq \sigma$ such that $\sigma' \notin \mathsf{trunc}(S)$, and hence $\sigma'' \preceq \sigma' \preceq \sigma$ such that $\sigma'' \notin S$. For all $\sigma^* \succeq \sigma''$, $\sigma^* \notin \mathsf{cl}(\mathsf{trunc}(S))$ and therefore $\sigma^* \notin P$. Hence, P is a safety property.

The converse is not true: a safety property that requires that no `Knock` event ever happens is not enforceable, as `Knock` events are only-observable and cannot be suppressed.

An *enforcer* can be seen as a TM which, given a finite trace, returns a pair of sets of events to be respectively suppressed and caused in the last database of the trace, with the additional requirement that events to be suppressed (resp. caused) should be suppressable (resp. causable) and present (resp. not already present) in this database.

Definition 4 (Enforcer). *An* enforcer *is a computable function $\mu : \mathbb{T}_f \to \mathbb{DB} \times \mathbb{DB}$ such that for all $\sigma \in \mathbb{T}_f$, $\tau \geq \mathsf{lts}(\sigma)$, $D \in \mathbb{DB}$, and $(B,C) = \mu(\sigma \cdot ((\tau, D)))$:*
(i) For all $(e,d) \in B$, $e \in \mathsf{Sup}$ and $(e,d) \in D$; and
(ii) For all $(e,d) \in C$, $e \in \mathsf{Cau}$ and $(e,d) \notin D$.

An enforcer μ is correct with respect to a property P if, for all $\sigma \in P$, any trace σ' obtained by adding a single database at the end of σ and then updating it (to some σ'') according to μ ensures $\sigma'' \in P$.

Definition 5 (Correct enforcement). *An enforcer μ is called* correct *with respect to a property $P \subseteq \mathbb{T}$ and a set of databases $\Delta \subseteq \mathbb{DB}$ if for all $\sigma \in P \cap \mathbb{T}_f$, $\tau \geq \mathsf{lts}(\sigma)$, $D \in \Delta$, and $(B,C) = \mu(\sigma \cdot ((\tau, D)))$, we have $\sigma \cdot ((\tau, (D \setminus B) \cup C)) \in P$.*

Transparent enforcers [12] do not to alter traces that belong to the enforced property:

Definition 6 (Transparent enforcement). *An enforcer μ is called* transparent *with respect to a property $P \subseteq \mathbb{T}$ and a set of databases $\Delta \subseteq \mathbb{DB}$ if for all $\sigma \in P \cap \mathbb{T}_f$, $\tau \geq \mathsf{lts}(\sigma)$, $D \in \Delta$, we have $\sigma \cdot ((\tau, D)) \in P \Longrightarrow \mu(\sigma \cdot ((\tau, D))) = (\emptyset, \emptyset)$.*

Given $A \subseteq \mathbb{T}$ and $B, C \subseteq \mathbb{E}$, $\mathsf{extend}(A, B, C)$ is the set of all traces $\sigma \cdot (\tau, D)$ obtained by appending to any trace $\sigma \in A$ the pair $(\tau, D \cup D')$ with $\tau \geq \mathsf{sts}(\sigma)$, $D \in 2^{B \times \mathbb{D}^*}$ and $D' = \{(c,d) \mid c \in C, d \in \mathbb{D}^{a(c)}\}$. Intuitively, set $\mathsf{extend}(A, B, C)$ is obtained from the set A by appending *some* events from B and *all* events from C to A. We have:

Lemma 3. *Let $P \subseteq \mathbb{T}$ such that P is enforceable. Then there exists a correct and transparent enforcer with respect to P and $\mathbb{DB}$.*

Proof. Consider the following function μ:

$$\begin{aligned} \mu(\sigma) &= (\emptyset, \emptyset) && \text{if } \sigma \in P \cup \{\varepsilon\} \\ \mu(\sigma \cdot ((\tau, D))) &= (\{(e,d) \in D \mid e \in \mathsf{Sup}\}, \\ &\qquad \{(e,d) \in \mathbb{E} \times \mathbb{D}^{a(e)} \mid e \in \mathsf{Cau}, (e,d) \notin D\}) && \text{if } \sigma \cdot ((\tau, D)) \notin P. \end{aligned}$$

Clearly, μ is an enforcer.

Now, let $\sigma \in P$, $\tau \geq \mathsf{lts}(\sigma)$, $D \in \mathbb{DB}$, and $(B,C) = \mu(\sigma \cdot ((\tau, D)))$. Further, let $\sigma' = \sigma \cdot ((\tau, D \setminus B \cup C))$. If $\sigma \cdot ((\tau, D)) \in P$, then $\sigma' = \sigma \cdot ((\tau, D)) \in P$. If $\sigma \cdot ((\tau, D)) \notin P$, then by the above definition of μ, we have $\sigma' \in \mathsf{extend}(P, \mathsf{Obs}, \mathsf{Cau})$, and hence $\sigma' \in P$ by Lemma 1. Therefore, μ is correct with respect to P and $\mathbb{DB}$. Since $\sigma' = \sigma \cdot (\tau, D)$ when $\sigma \cdot ((\tau, D)) \in P$, then μ is also transparent.

$v,i \models_\sigma r(t_1,...,t_n)$ iff $(r,(v(t_1),...,v(t_n))) \in D_i$ | $v,i \models_\sigma \neg\varphi$ iff $v,i \not\models_\sigma \varphi$

$v,i \models_\sigma \exists x.\ \varphi$ iff $v[x \mapsto d], i \models_\sigma \varphi$ for $d \in \mathbb{D}$ | $v,i \models_\sigma \varphi \vee \psi$ iff $v,i \models_\sigma \varphi$ or $v,i \models_\sigma \psi$

$v,i \models_\sigma \bullet_I \varphi$ iff $i > 1$ and $v,i-1 \models_\sigma \varphi$ and $\tau_i - \tau_{i-1} \in I$ | $v,i \models_\varepsilon \varphi$

$v,i \models_\sigma \bigcirc_I \varphi$ iff $i+1 \leq |\sigma|$ and $v,i+1 \models_\sigma \varphi$, and $\tau_{i+1} - \tau_i \in I$

$v,i \models_\sigma \varphi\, \mathsf{S}_I\, \psi$ iff $v,j \models_\sigma \psi$ for some $j \leq i, \tau_i - \tau_j \in I$, and $v,k \models_\sigma \varphi$ for all k, $j < k \leq i$

$v,i \models_\sigma \varphi\, \mathsf{U}_I\, \psi$ iff $v,j \models_\sigma \psi$ for some $|\sigma| \geq j \geq i, \tau_j - \tau_i \in I$, and $v,k \models_\sigma \varphi$ for all k, $j > k \geq i$

Fig. 2: MFOTL semantics

4 Metric first-order temporal logic

Metric first-order temporal logic (MFOTL) extends first-order logic with the metric temporal operators "previous" ($\bullet_I$), "next" ($\bigcirc_I$), "since" (S_I), and "until" (U_I). We write $\mathbb{I}$ for the set of intervals over $\mathbb{N}$ and $\mathbb{V}$ for a countable set of variables. MFOTL formulae over signature Σ are defined by the grammar

$$\varphi ::= r(t_1,\ldots,t_{a(r)}) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\ \varphi \mid \bullet_I \varphi \mid \bigcirc_I \varphi \mid \varphi\, \mathsf{S}_I\, \varphi \mid \varphi\, \mathsf{U}_I\, \varphi,$$

where $t_1,\ldots,t_{a(r)} \in \mathbb{V} \cup \mathbb{D}$, $r \in \mathbb{E}$, and $I \in \mathbb{I}$. We define shorthands $\top := p \vee \neg p$, $\bot := \neg\top$, $\varphi \Rightarrow \psi := \neg\varphi \vee \psi$, and the operators "once" ($\blacklozenge_I \varphi := \top\, \mathsf{S}_I\, \varphi$), "eventually" ($\Diamond_I \varphi := \top\, \mathsf{U}_I\, \varphi$), "always" ($\Box_I \varphi := \neg \Diamond_I \neg\varphi$), and "historically" ($\blacksquare_I \varphi := \neg \blacklozenge_I \neg\varphi$). Temporal operators with no interval have $[0,\infty)$ instead. Predicate is a formula of the form $r(t_1,\ldots,t_{a(r)})$.

A *valuation* is a mapping $v : \mathbb{V} \to \mathbb{D}$. We extend v's domain to $\mathbb{D}$ by setting $v(d) = d$ for all $d \in \mathbb{D}$. We write $v[x \mapsto d]$ for the mapping equal to v, except that $v(x)$ is d. We use $\mathsf{fv}(\varphi)$ for the set of φ's free variables. Given a trace σ, a timepoint $1 \leq i \leq |\sigma|$, a valuation v, and an MFOTL formula φ, satisfaction relation $\models$ is defined in Figure 2. Note that $\models$ is well-defined for both finite and infinite traces. We write $v \models_\sigma \varphi$ for $v,1 \models_\sigma \varphi$.

We say that two MFOTL formulae φ and ψ are *equivalent*, written $\varphi \equiv \psi$, iff for all v, $\sigma \in \mathbb{T}$, $1 \leq i \leq |\sigma|$, we have $v,i \models_\sigma \varphi \Leftrightarrow v,i \models_\sigma \psi$.

If φ is closed, i.e., $\mathsf{fv}(\varphi) = \emptyset$, φ's satisfaction does not depend on v. We then write $\models_\sigma \varphi$ as shorthand for $\forall v.\ v \models_\sigma \varphi$. Given a closed formula φ, we denote by $\mathcal{L}(\varphi) \subseteq \mathbb{T}$ the set of all traces that satisfy φ, i.e., $\mathcal{L}(\varphi) := \{\sigma \in \mathbb{T} \mid \models_\sigma \varphi\}$. Finally, we denote by $\mathcal{L}_f(\varphi)$ the set of finite traces in $\mathcal{L}(\varphi)$, i.e., $\mathcal{L}_f(\varphi) = \{\sigma \in \mathcal{L}(\varphi) \mid |\sigma| < \infty\}$. Extending the previous terminology, we say that a formula φ is enforceable iff $\mathcal{L}(\varphi)$ is enforceable.

Definition 7 (Future-free formulae). *An MFOTL formula φ is called* future-free *iff for all $\sigma \in \mathbb{T}$, valuation v, and $\sigma' \preceq \sigma$ such that $|\sigma'| = i$, we have $v,i \models_\sigma \varphi \Leftrightarrow v,i \models_{\sigma'} \varphi$.*

The truth value of a future-free formula at a given timepoint only depends on the content of the trace at past or present timepoints. Therefore, an enforcer can determine the satisfaction of a future-free formula for each trace prefix. For instance, formulae without future operators (U_I, $\bigcirc_I$, $\Diamond_I$, $\Box_I$) are future-free, but also some that have them.

Example 3. The formula $\varphi_1 = \blacklozenge_{[3,4]}(\exists x.\ \texttt{Close}(x))$ uses no future temporal operators, and is therefore future-free. The formula $\varphi_2 = \blacklozenge_{[3,4]}(\exists x.\ \texttt{Close}(x) \wedge \Diamond_{[1,2]}\, \texttt{Open}(x))$ contains a future operator, but is still future-free, since the future operator $\Diamond_{[1,2]}$ (looking at most 2 time units into the future) is nested in a $\blacklozenge_{[3,4]}$ operator that is always evaluated at least 3 time units in the past. The formula $\varphi_3 = \Diamond_{[1,2]}\, \texttt{Open}(x)$ is not future-free: its truth value depends on events happening up to 2 time units in the future.

In the rest of this paper, we consider the fragment $\text{MFOTL}_{\square}^{\mathcal{F}}$ that contains all closed formulae of the form $\square\varphi$, where φ is future-free. Given the correctness of the monitoring algorithm [14] for MFOTL formulae of the form $\square\varphi$, where all future operators in φ have bounded intervals and the fact that future-free formulae are a subset of the algorithm's supported formulae, we have:

Lemma 4. *For any $\varphi \in \text{MFOTL}_{\square}^{\mathcal{F}}$, there exists a TM that decides $\mathcal{L}_f(\varphi)$.*

In fact, the algorithm determines without delay whether a future-free formula is satisfied.

5 MFOTL enforceability

In this section, we characterize the enforceability of $\text{MFOTL}_{\square}^{\mathcal{F}}$ formulae with an enforcer as described in Section 3. Our first result is negative: a reduction shows that the enforceability of $\text{MFOTL}_{\square}^{\mathcal{F}}$ formulae in undecidable.

Theorem 1. *Assume that* Sup *contains at least one event of arity at least 2 and* $\mathsf{Obs} \neq \emptyset$. *The subset $\mathcal{E} = \{\varphi \in \text{MFOTL}_{\square}^{\mathcal{F}} \mid \varphi \text{ is enforceable}\}$ of $\text{MFOTL}_{\square}^{\mathcal{F}}$ is not computable.*

Proof. Let ENFORCEABLE denote the problem

$$\begin{aligned} &\text{Input} : \varphi \in \text{MFOTL}_{\square}^{\mathcal{F}} \\ &\text{Output} : \texttt{True} \text{ if } \varphi \text{ is enforceable}, \texttt{False} \text{ otherwise}, \end{aligned}$$

and VALIDFOL denote the Entscheidungsproblem of FOL with events in Sup:

$$\begin{aligned} &\text{Input} : \psi \in \text{FOL}, \psi \text{ closed, all predicates in } \mathsf{Sup} \\ &\text{Output} : \texttt{True} \text{ if } \psi \text{ is universally valid}, \texttt{False} \text{ otherwise}, \end{aligned}$$

which is undecidable by Church's theorem [23, 58] since we assumed that Sup contains at least one predicate of arity at least 2.

As $\mathsf{Obs} \neq \emptyset$, choose $\mathtt{0} \in \mathsf{Obs}$. Let $k := a(\mathtt{0})$, $x \in \mathbb{D}$ and $\mathbf{x} = (x, \dots, x) \in \mathbb{D}^k$.

Let $\mathcal{M}$ be a TM deciding ENFORCEABLE. Now, consider the TM $\mathcal{N}$ that takes any closed $\psi \in \text{FOL}$ with predicates in Sup, computes $\varphi := \square[\mathtt{0}(\mathbf{x}) \Rightarrow \hat{\psi}]$, where $\hat{\psi}$ is obtained by replacing every atom $E(a_1, \dots, a_p)$ in ψ by $\blacklozenge E(a_1, \dots, a_p)$, passes φ to $\mathcal{M}$ and returns the verdict $\mathcal{N}(\psi) := \mathcal{M}(\varphi)$.

Clearly, φ is in $\text{MFOTL}_{\square}^{\mathcal{F}}$, and the TM $\mathcal{N}$ always terminates.

If $\mathcal{N}(\psi) = \texttt{True}$, then φ is enforceable. By contradiction, assume that ψ is not universally valid, and let μ be an interpretation on Sup and v a valuation such that $v \not\models_\mu \psi$. Now, let $\sigma = ((0, \mu))$ and $\sigma' = ((0, \mu), (1, \{(\mathtt{0}, \mathbf{x})\} \cup \{(c, d) \mid c \in \mathsf{Cau}, d \in \mathbb{D}^{a(c)}\})) \in \mathsf{extend}(\{\sigma\}, \mathsf{Obs}, \mathsf{Cau})$. Clearly, $\models_\sigma \varphi$ (since the LHS of the implication in φ is always false in σ), but $\not\models_{\sigma'} \varphi$, contradicting φ's enforceability. Hence, ψ is universally valid.

If $\mathcal{N}(\psi) = \texttt{False}$, then φ is not enforceable. We get $\sigma \in \mathbb{T}$, $\sigma' \in \mathsf{extend}(\{\sigma\}, \mathsf{Obs}, \mathsf{Cau})$ such that (a) $\models_\sigma \varphi$ but (b) $\not\models_{\sigma'} \varphi$. By construction, if $\models_\sigma \hat{\psi}$, then $\models_{\sigma'} \hat{\psi}$ (the truth value of $\hat{\psi}$ depends only on Sup events, which only occur at timepoint 0), thus contradicting (b). Hence, $\not\models_\sigma \hat{\psi}$. Collecting all Sup atoms that occur in σ, we obtain an interpretation μ such that $\not\models_\mu \psi$. Hence, ψ is not universally valid.

We conclude that $\mathcal{N}$ decides VALIDFOL, which is absurd.

The proof relies on the undecidability of universal validity in FOL. Therefore, it is sensible to ask whether some syntactical characterization of enforceability can be recovered by reasoning *modulo equivalence of formulae*. Is there a decidable and enforceable fragment of MFOTL$^{\mathcal{F}}_{\Box}$ that contains all enforceable policies modulo equivalence? If so, such a fragment would not only provide a sound approximation of enforceable MFOTL$^{\mathcal{F}}_{\Box}$ formulae, but also an approximation that is *expressively complete*. All enforceable MFOTL$^{\mathcal{F}}_{\Box}$ policies could be expressed using the fragment via an appropriate (manual) rewriting. Rather surprisingly, such a fragment exists. Consider the following:

Definition 8 (GMFOTL). Guarded MFOTL *(*GMFOTL*) is defined inductively by:*

$$\psi ::= \bot \mid s(t_1,\ldots,t_n) \mid \neg c(t_1,\ldots,t_n) \mid \psi \wedge \varphi \mid \psi \vee \psi \mid \exists x.\, \psi$$

where $s \in \mathsf{Sup}, c \in \mathsf{Cau}$, *and* φ *is an* MFOTL *formula.*

In GMFOTL, all subformulae (and, in particular, all temporal subformulae) are *guarded* by an instance of a predicate $r(t_1,\ldots,t_n)$ with r being suppressable, or by an instance of a negated predicate $\neg r(t_1\ldots,t_n)$ with r being causable. In the following, we call such a (possibily negated) predicate a *guard*. The presence of a guard ensures that, whenever an GMFOTL formula is true on a trace prefix, it can always be made false by suppressing or causing appropriate events in the last database of the prefix.

Example 4. Consider the formula $\varphi_4 = \neg\texttt{Close}(x) \wedge \psi$, where ψ is an arbitrary future-free formula and $\mathsf{fv}(\psi) = \{x\}$. For φ_4 to be true on a trace prefix σ, ψ must hold for some valuations of x and $\{(\texttt{Close},(a)) \mid v, |\sigma| \models_\sigma \psi, v(x) = a\}$ must not be in the last database of the prefix. Hence, φ_4 can be made false by causing the appropriate `Close` events.

This way, it can be shown that all closed formulae of the form $\Box\neg\psi$ with $\psi \in$ GMFOTL and future-free are enforceable. Since enforceability is defined in terms of the language recognized by a given formula, we obtain that all MFOTL$^{\mathcal{F}}_{\Box}$ formulae equivalent to some $\Box\neg\psi$ with $\psi \in$ GMFOTL closed and future-free are enforceable. In fact, the converse is also true: all future-free MFOTL$^{\mathcal{F}}_{\Box}$ formulae are equivalent to a formulae of the above form. We have thus obtained an expressively complete fragment of enforceable MFOTL$^{\mathcal{F}}_{\Box}$. To show this, we first introduce an alternative definition of enforceability for closed formulae:

Lemma 5. *A closed formula* $\varphi \in$ MFOTL *is enforceable iff the following conditions are fulfilled:*

(i) There exists a TM $\mathcal{M}$ *that decides* $\mathcal{L}_f(\varphi)$.
(ii) φ *is a safety formula, i.e.,* $\mathcal{L}_f(\varphi)$ *is a safety property.*
(iii) For all finite $\sigma \in \mathbb{T}$*, for all* $\sigma' \in \mathsf{extend}(\{\sigma\}, \mathsf{Obs}, \mathsf{Cau})$, $\models_\sigma \varphi \Longrightarrow \models_{\sigma'} \varphi$.

Proof. Let $\varphi \in$ MFOTL closed and fulfilling (i)–(iii). We get a TM $\mathcal{M}$ that terminates on every $\sigma \in \mathbb{T}$ and decides $\mathcal{L}_f(\varphi) \ni \varepsilon$. The formula φ is also safety, hence $\mathcal{L}(\varphi) = \mathsf{cl}(\mathsf{trunc}(\mathcal{L}_f(\varphi)))$. Finally, consider $\sigma' \in \mathsf{extend}(\mathsf{trunc}(\mathcal{L}(\varphi)), \mathsf{Obs}, \mathsf{Cau})$. Let $\sigma \in \mathsf{trunc}(\mathcal{L}_f(\varphi))$ such that $\sigma' \in \mathsf{extend}(\{\sigma\}, \mathsf{Obs}, \mathsf{Cau})$. The fact that $\sigma \in \mathsf{trunc}(\mathcal{L}_f(\varphi))$ yields $\models_\sigma \varphi$. By (iii), we obtain $\models_{\sigma'} \varphi$; hence $\mathcal{M}$ accepts σ', and $\mathcal{L}(\varphi)$ (and φ) is enforceable.

Conversely, let $\varphi \in \text{MFOTL}$ be a closed enforceable formula. By definition, $\mathcal{L}(\varphi)$ is enforceable. We obtain a TM $\mathcal{M}$ for φ deciding a set of finite traces S with $\mathsf{cl}(\mathsf{trunc}(S)) = \mathcal{L}(\varphi)$, such that $\mathsf{extend}(\mathsf{trunc}(S), \mathsf{Obs}, \mathsf{Cau}) \subseteq S$. Clearly, φ is safety. Now, observe that since S is computable, then $\mathsf{trunc}(S)$ is computable as well. But since $\mathsf{cl}(\mathsf{trunc}(S)) = \mathcal{L}(\varphi)$, it is easy to see that $\mathsf{trunc}(S) = \mathcal{L}_f(\varphi)$. Hence $\mathcal{L}_f(\varphi)$ is computable. Finally, let $\sigma \in \mathbb{T}$ be a finite trace such that $\models_\sigma \varphi$; in particular, $\sigma \in \mathcal{L}(\varphi)$. Since $\mathsf{cl}(\mathsf{trunc}(S)) = \mathcal{L}(\varphi)$, we get $\sigma \in \mathsf{trunc}(S)$. Now, if we consider some $\sigma' \in \mathsf{extend}(\{\sigma\}, \mathsf{Obs}, \mathsf{Cau})$, we get $\sigma' \in \mathsf{extend}(\mathsf{trunc}(S), \mathsf{Obs}, \mathsf{Cau}) \subseteq S$, then $\sigma' \in \mathsf{cl}(\mathsf{trunc}(S))$ (since all prefixes of σ' are in S), and finally $\models_{\sigma'} \varphi$.

Then, we rewrite the definition of formula enforceability for closed $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae in terms of the universal validity of a formula $\Phi(\varphi)$. Formula $\Phi(\varphi)$ captures the fact that as soon as φ is violated, at least one suppressable event occurs at the current timepoint or at least one causable event does not.

Lemma 6. *A formula $\Box\varphi \in \text{MFOTL}^{\mathcal{F}}_{\Box}$ is enforceable iff $\Phi(\varphi) := \neg\varphi \Rightarrow \mathcal{C} \vee \bullet\blacklozenge\neg\varphi$ is universally valid, where*

$$\mathcal{C} = \bigvee_{r \in \mathsf{Sup}} (\exists y_1, \ldots, y_{a(r)}).\ r(y_1, \ldots, y_{a(r)})) \vee \bigvee_{r \in \mathsf{Cau}} (\exists y_1, \ldots, y_{a(r)}).\ \neg r(y_1, \ldots, y_{a(r)})).$$

Proof. Let φ be a closed and future-free form. Assume that $\Box\varphi$ is enforceable. Let $\sigma' \in \mathbb{T}$ such that $\models_{\sigma'} \neg\mathcal{C} \wedge \bullet\blacksquare\varphi$. Let $\sigma = (\sigma'_1, \ldots, \sigma'_{|\sigma|-1})$. Since $\models_{\sigma'} \neg\mathcal{C}$, we have $\sigma' \in \mathsf{extend}(\{\sigma\}, \mathsf{Obs}, \mathsf{Cau})$. Since $\models_{\sigma'} \bullet\blacksquare\varphi$ and φ is future-free, then for all $\sigma'' \preceq \sigma$, $\models_{\sigma''} \varphi$. As $\Box\varphi$ is enforceable, we get $\models_{\sigma'} \varphi$, and finally $\models_{\sigma'} \Phi(\varphi)$. Conversely, assume that $\Phi(\varphi)$ is universally valid. Let $\sigma \in \mathbb{T}$, $\sigma' \in \mathsf{extend}(\{\sigma\}, \mathsf{Obs}, \mathsf{Cau})$ such that for all $\sigma'' \preceq \sigma$, $\models_{\sigma''} \varphi$. Since φ is future-free, we get $\models_{\sigma'} \neg\mathcal{C} \wedge \bullet\blacksquare\varphi$, and finally $\models_{\sigma'} \varphi$. Since, for all $\varphi \in \text{MFOTL}^{\mathcal{F}}_{\Box}$, φ is safety (Lemma 2) and there exists a TM deciding $\mathcal{L}_f(\varphi)$ (Lemma 4), this concludes the proof.

We now prove our main result:

Theorem 2. *A formula $\Box\varphi \in \text{MFOTL}^{\mathcal{F}}_{\Box}$ is enforceable iff there exists $\psi \in \text{GMFOTL}$ such that $\Box\varphi \equiv \Box\neg\psi$.*

Proof. First, we show that for any $\psi \in \text{GMFOTL}$ with $\mathsf{fv}(\psi) = \{x_1, \ldots, x_k\}$, the formula $\forall x_1, \ldots, x_n.\ [\psi \Rightarrow \mathcal{C}]$ is universally valid. The proof is by structural induction on ψ.

- If $\psi = r(t_1, \ldots, t_n)$ with $r \in \mathsf{Sup}$, $\psi = \neg r(t_1, \ldots, t_n)$ with $r \in \mathsf{Cau}$, or $\psi = \bot$, the result is trivial.
- If $\psi = \varphi_1 \wedge \varphi_2$ where $\varphi_1 \in \text{GMFOTL}$, assume that $\forall x_1, \ldots, x_n.\ [\varphi_1 \Rightarrow \mathcal{C}]$ is universally valid. Let v, σ such that $v \models_\sigma \psi$. Then $v \models_\sigma \varphi_1$, and by our assumption $v \vDash_\sigma \mathcal{C}$. Hence, $\forall x_1, \ldots, x_n.\ [\psi \Rightarrow \mathcal{C}]$ is universally valid.
- If $\psi = \varphi_1 \vee \varphi_2$ where $\varphi_1, \varphi_2 \in \text{GMFOTL}$, assume that $\forall x_1, \ldots, x_n.\ [\varphi_1 \Rightarrow \mathcal{C}]$ and $\forall x_1, \ldots, x_n.\ [\varphi_2 \Rightarrow \mathcal{C}]$ are universally valid. Let v, σ such that $v \vDash_\sigma \psi$. Then $v \vDash_\sigma \varphi_1$ or $v \vDash_\sigma \varphi_2$. W.l.o.g., assume the former. Hence, by our first assumption, $v \vDash_\sigma \mathcal{C}$. We conclude that $\forall x_1, \ldots, x_n.\ [\psi \Rightarrow \mathcal{C}]$ is universally valid.

– If $\psi = \exists x_0.\, \varphi_1$ where $x_0 \in \mathsf{fv}(\varphi_1)$ and $\varphi_1 \in \text{GMFOTL}$, assume that $\forall x_0, \ldots, x_n.\ [\varphi_1 \Rightarrow \mathcal{C}]$. Let v, σ such that $v \vDash_\sigma \psi$. Then there exists $d \in \mathbb{D}$ and $v' = v[x_0 \mapsto d]$ such that $v' \vDash_\sigma \varphi_1$. By our assumption, $v' \vDash_\sigma \mathcal{C}$, and since $\mathsf{fv}(\mathcal{C}) = \emptyset$, we finally get $v \vDash_\sigma \varphi_1$. Therefore, $\forall x_1, \ldots, x_n.\ [\psi \Rightarrow \mathcal{C}]$ is universally valid.

In particular, if ψ is closed, then $\psi \Rightarrow \mathcal{C}$ is universally valid. Therefore, $\psi \Rightarrow \mathcal{C} \vee \bullet \blacklozenge \psi$ is universally valid, and, by Lemma 6, $\Box \neg \psi$ is enforceable. If $\Box \varphi \equiv \Box \neg \psi$, then $\Box \varphi$ is enforceable as well.

Conversely, if $\Box \varphi$ is enforceable, then $\Phi(\varphi)$ is universally valid by Lemma 6. We have

$$\begin{aligned}
\Box \varphi &\equiv \neg(\Diamond \neg \varphi) \\
&\equiv \neg(\Diamond(\neg\varphi \wedge (\mathcal{C} \vee \bullet \blacklozenge \neg\varphi))) && (\Phi(\varphi) \text{ is universally valid}) \\
&\equiv \neg \Diamond(\neg\varphi \wedge \mathcal{C}) && (*) \\
&\equiv \Box \neg(\mathcal{C} \wedge \neg\varphi)
\end{aligned}$$

where we obtain equality $(*)$ by observing that at the first timepoint for which $\neg\varphi \wedge (\mathcal{C} \vee \bullet \blacklozenge \neg\varphi)$ is fulfilled, we necessarily have $\neg\varphi \wedge \mathcal{C}$: we need $\neg\varphi$ to hold, and as there is no earlier timepoint at which $\neg\varphi$ holds, $\mathcal{C}$ must hold.

Finally, we easily check that $\mathcal{C} \in \text{GMFOTL}$: formula $\mathcal{C}$ is a disjunction of atoms of the form $r(t_1, \ldots, t_n)$ for $r \in \mathsf{Sup}$ and $\neg r(t_1, \ldots, t_n)$ for $r \in \mathsf{Cau}$. Therefore, $\mathcal{C} \wedge \neg\varphi \in \text{GMFOTL}$. We choose $\psi := \mathcal{C} \wedge \neg\varphi$ to obtain $\Box \varphi \equiv \Box \neg \psi$, which concludes the proof.

Example 5. Consider the formula $\varphi_5 = \Box \forall x.\, (\mathtt{Open}(x) \Rightarrow \neg \blacklozenge_{[2,5]} \mathtt{Open}(x))$. This formula is enforceable: `Open` events which lead to a violation (i.e., those occurring 2 to 5 time units after a previous `Open` event with the same argument) can always be suppressed. The formula φ_5 is equivalent to $\Box \neg \psi$ where $\psi \in \text{GMFOTL}$ is

$$(\exists x.\, \mathtt{Open}(x)) \wedge \neg(\forall x.\, (\mathtt{Open}(x) \Rightarrow \neg \blacklozenge_{[2,5]} \mathtt{Open}(x))).$$

6 MFOTL enforcement in the finite case

In the previous section, we have presented GMFOTL, a syntactic class of MFOTL that is expressively complete for enforceable $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae. Lemma 3 implies the existence of an enforcer for such formulae. However, the naive enforcer constructed in the lemma's proof may be inefficient—in fact, it may cause an infinite number of new events.

In this section, we focus on traces with finite databases and MFOTL formulae from the intersection of enforceable $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae with monitorable MFOTL formulae [14]. Similarly as we approximated enforceable $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae with GMFOTL, we will do so for monitorable and enforceable $\text{MFOTL}^{\mathcal{F}}_{\Box}$ formulae . We show that, in this case, we can exhibit a correct and transparent enforcer that produces only a finite number of events to be suppressed or caused.

6.1 Monitoring MFOTL formulae

Basin *et al.* [14] describe an algorithm that efficiently monitors monitorable MFOTL formulae. Variants of this algorithm and the fragment it supports are used in several state-of-the-art tools [15, 57]. We now briefly recall the algorithm and some of its properties.

The algorithm encodes each database $D \in \mathbb{DB}^\dagger$ as a finite set of tables, one for each event name in the database. Row d is in the table corresponding to the event name e if $(e,d) \in D$. Set of satisfying valuations of a formula can be similarly encoded as a table with rows representing valuations restricted to the domain of the formula's free variables.

The algorithm computes the table of satisfying valuations for a monitorable MFOTL formula bottom-up, using well-known table operations (e.g., join, anti-join, union, projection and others). The syntactic monitorable fragment ensures that table operations always produce finite tables. In the rest of the section, we assume that this algorithm is available as a subroutine $\textsc{Sat}(\varphi,\sigma) = \{v \mid v, |\sigma| \models_\sigma \varphi\}$ that returns the set of satisfying valuations of a monitorable MFOTL formula φ with respect to finite trace $\sigma \in \mathbb{T}^\dagger$ and timepoint $|\sigma|$.

The monitorable MFOTL fragment [57] is given in Figure 5 in the Appendix. It also ensures that for any valuation v satisfying a formula φ from the fragment with respect to a finite trace σ and a time point $1 \leq i \leq |\sigma|$, for every $x \in \mathsf{fv}(\varphi)$ value $v(x) \in \mathbb{D}$ is contained in some event argument in a database in σ or a constant term in φ. Formally:

Lemma 7. *For all monitorable $\varphi \in$ MFOTL, valuation v, trace $\sigma \in \mathbb{T}^\dagger$, and timepoint $1 \leq i \leq |\sigma|$, assuming $v, i \models_\sigma \varphi$, we have*

$$\forall x \in \mathsf{fv}(\varphi).\ \exists 1 \leq j \leq |\sigma|.\ (e,d) \in D_j, 1 \leq k \leq a(e).\ d_k = v(x) \vee d_k \in \mathsf{cst}(\varphi)$$

where $\mathsf{cst}(\varphi) \subset \mathbb{D}$ denotes the (finite) set of constant terms that appear in φ.

Proof. Let φ, v, σ and i as above. Assume $v, i \models_\sigma \varphi$, $\mathsf{fv}(\varphi) \neq \emptyset$, and choose $x_0 \in \mathsf{fv}(\varphi)$.

The proof is by induction on φ, following the rules defining monitorable MFOTL in Figure 5.

- If $\varphi = t_1 \approx t_2$, as $\mathsf{fv}(\varphi) \neq \emptyset$, exactly one of t_1 or t_2 is constant. W.l.o.g., assume $\varphi = x_0 \approx c_0$ with $c_0 \in \mathbb{D}$. Since v satisfies φ, we get $v(x_0) = c_0 \in \mathsf{cst}(\varphi)$.
- If $\varphi = r(t_1, \ldots, t_n)$, since v satisfies φ at timepoint i, then there exists $(r,d) = (r, (v(t_1), \ldots, v(t_n))) \in D_i$. Moreover, by definition of $\mathsf{fv}(\varphi)$, there exists $1 \leq j \leq n$ such that $t_j = x_0$. For such j and d, we get $d_j = v(x_0)$.
- If $\varphi = \varphi_1 \wedge \varphi_2$ for monitorable φ_1 and φ_2, then $x_0 \in \mathsf{fv}(\varphi) = \mathsf{fv}(\varphi_1) \cup \mathsf{fv}(\varphi_2)$ implies $x_0 \in \mathsf{fv}(\varphi_1)$ or $x_0 \in \mathsf{fv}(\varphi_2)$. W.l.o.g., assume $x_0 \in \mathsf{fv}(\varphi_1)$. Since $v, i \models_\sigma \varphi$, we also have $v, i \models_\sigma \varphi_1$. We conclude by using the induction hypothesis on φ_1.
- If $\varphi = \neg\varphi_1 \wedge \varphi_2$ for monitorable φ_1 and φ_2 with $\mathsf{fv}(\varphi_1) \subseteq \mathsf{fv}(\varphi_2)$, then $\mathsf{fv}(\varphi) = \mathsf{fv}(\varphi_2)$ and we now that $x_0 \in \mathsf{fv}(\varphi_2)$. Since $v, i \models_\sigma \varphi$, we have $v, i \models_\sigma \varphi_2$, and we conclude by using the induction hypothesis on φ_2.
- If $\varphi = \exists x_1.\ \varphi_1$ for monitorable φ_1, then $\mathsf{fv}(\varphi) \subseteq \mathsf{fv}(\varphi_1)$, hence $x_0 \in \mathsf{fv}(\varphi_1)$. Since $v, i \models_\sigma \varphi$, we have $v', i \models_\sigma \varphi_1$ for some $v' = v[x_0 \mapsto c_0]$, $c_0 \in \mathbb{D}$. The induction hypothesis on φ_1 and v' provides us with some $\exists 1 \leq j \leq |\sigma|$, $(e,d) \in D_j$, and $1 \leq k \leq a(e)$ such that $d_k = v'(x_0)$ or $d_k \in \mathsf{cst}(\varphi)$. In the latter case, we obtain the desired result. In the former, we only need to observe that $v(x_0) = v'(x_0) = d_k$, and the conclusion follows.
- If $\varphi = \varphi_1 \,\mathsf{S}_I\, \varphi_2$ for monitorable φ_1 and φ_2 with $\mathsf{fv}(\varphi_1) \subseteq \mathsf{fv}(\varphi_2)$, then $x_0 \in \mathsf{fv}(\varphi_2) = \mathsf{fv}(\varphi)$. Since $v, i \models_\sigma \varphi$, we get $1 \leq i' \leq i$ such that $v, i' \models_\sigma \varphi_2$. We conclude by using the induction hypothesis on φ_2 and i'.
- The proof is similar for all other temporal operators.

– If $\varphi = \neg\varphi_1$ for monitorable φ_1 and $\mathsf{fv}(\varphi_1) = \emptyset$, then $\mathsf{fv}(\varphi) = \emptyset$, contradicting the above assumption.

We will use this lemma, as well as the termination of subroutine SAT [14], to prove the termination of our enforcer.

6.2 Enforcer

Given $\sigma \in \mathbb{T}_f$, $\tau \geq \mathsf{lts}(\sigma)$, and $D, D^-, D^+ \in \mathbb{DB}$, we first define function update as

$$\mathsf{update}(\sigma \cdot ((\tau, D)), (D^-, D^+)) := \sigma \cdot ((\tau, (D \cup D^+) \setminus D^-))$$

as the trace obtained by adding all events from D^+ and removing all events from D^- in the last database of σ.

For any $\sigma \in \mathbb{T}_f$ and enforcer μ, we define $\ell_\mu(\sigma) \in \mathbb{T}_f$ as the limit of the sequence $(u_i)_{i\in\mathbb{N}} \in \mathbb{T}_f^{\mathbb{N}}$ defined by $u_0 = \sigma$ and for all $i \in \mathbb{N}$, $u_{i+1} = \mathsf{update}(u_i, \mu(u_i))$. This limit is always well-defined [1], and if $u_{i+1} = u_i$ for some $i \in \mathbb{N}$, we have $\ell_\mu(\sigma) = u_i$. This allows us to define a routine FIXPOINT(σ, μ) which iteratively computes $u_0, u_1, \ldots, u_i, \ldots$, returns $\ell_\mu(\sigma) = u_i$ as soon as $(u_i)_{i\in\mathbb{N}}$ reaches a fixpoint $u_{i+1} = u_i$, and does not terminate otherwise. We will later show that, in our setup, this procedure always terminates.

Our enforcer relies on the function enf described in Algorithm 1, which takes as an input a future-free and monitorable GMFOTL formula φ, a finite trace σ, and a valuation v such that $v, |\sigma| \models_\sigma \varphi$, and returns a pair of sets of events to be respectively suppressed and caused at the last timepoint in σ in order to obtain some new trace σ' such that $v, |\sigma| \not\models_{\sigma'} \varphi$. For notational convenience, we denote by $\Cup$ the elementwise union of pairs of sets $(A, B) \Cup (C, D) = (A \cup C, B \cup D)$.

The intuition behind enf is as follows. If the formula φ is reduced to an atom $r(t_1, \ldots, t_n)$ or $\neg r(t_1, \ldots, t_n)$, we can make it false by suppressing or causing a single event. If φ is of the form $\varphi_1 \wedge \varphi_2$ with $\varphi \in$ GMFOTL, it is sufficient to make φ_1 false to make φ false: enf looks for events to be suppressed or caused in φ_1.

For formulae of the form $\varphi_1 \vee \varphi_2$, additional care is needed. At first glance, the strategy used for $\wedge$ seems applicable, modulo a simple case distinction: if both φ_1 and φ_2 are satisfied by a given pair of a trace and a valuation, we need to find events to suppress or cause in *both* subformulae; if only one conjunct is satisfied, we look for events to suppress or cause in this subformula only. But such a one-step strategy is insufficient.

[1] The function update only affects the last database of its argument. For all $i \in \mathbb{N}$, denote by D_i the last database of u_i. Since μ is an enforcer and $\mathsf{Sup} \cap \mathsf{Cau} = \emptyset$, we have

$$D_{i+1} = \left(D_0 \setminus \bigcup_{j=1}^{i} \mu^-(u_i) \right) \cup \bigcup_{j=1}^{i} \mu^+(u_i)$$

where $\mu^-(\sigma) := (\mathsf{let}\ (B, C) = \mu(\sigma)\ \mathsf{in}\ B)$ returns the first component of $\mu(\sigma)$, and $\mu^+(\sigma) := (\mathsf{let}\ (B, C) = \mu(\sigma)\ \mathsf{in}\ C)$ returns the second component. We see that D_i is the union of a monotonically decreasing sequence and a monotonically increasing sequence of sets of events. Therefore, $\lim D_i$ exists and $\ell_\mu(\sigma)$ is well-defined.

Algorithm 1 Function enf

function $\mathsf{enf}(\varphi, \sigma, v)$
  if $\varphi = r(t_1, \ldots, t_n), r \in \mathsf{Sup}$ **then**
    return $(\{(r, (v(t_1), \ldots, v(t_n)))\}, \emptyset)$
  else if $\varphi = \neg r(t_1, \ldots, t_n), r \in \mathsf{Cau}$ **then**
    return $(\emptyset, \{(r, (v(t_1), \ldots, v(t_n)))\})$
  else if $\varphi = \varphi_1 \wedge \varphi_2$ **then**
    return $\mathsf{enf}(\varphi_1, \sigma, v)$
  else if $\varphi = \varphi_1 \vee \varphi_2$ **then**
    return $\textsc{Fixpoint}(\sigma, \mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v})$
  else if $\varphi = \exists x.\ \varphi_1$ **then**
    return $\textsc{Fixpoint}(\sigma, \mathsf{enf}_{\mathsf{ex},\varphi_1,v})$

function $\mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v}(\sigma)$
  $(D^-, D^+) \leftarrow (\emptyset, \emptyset)$
  if $v \in \textsc{Sat}(\varphi_1, \sigma)$ **then**
    $(D^-, D^+) \leftarrow (D^-, D^+) \uplus \mathsf{enf}(\varphi_1, \sigma, v)$
  if $v \in \textsc{Sat}(\varphi_2, \sigma)$ **then**
    $(D^-, D^+) \leftarrow (D^-, D^+) \uplus \mathsf{enf}(\varphi_2, \sigma, v)$
  return (D^-, D^+)

function $\mathsf{enf}_{\mathsf{ex},\varphi_1,v}(\sigma)$
  $(D^-, D^+) \leftarrow (\emptyset, \emptyset)$
  for $v \in \mathbb{D}$ s.t. $v[x \mapsto v] \in \textsc{Sat}(\varphi_1, \sigma)$ **do**
    $(D^-, D^+) \leftarrow (D^-, D^+) \uplus \mathsf{enf}(\varphi_1, \sigma, v[x \mapsto v])$
  return (D^-, D^+)

Example 6. Consider the formula $\varphi_6 = \mathtt{Open}(1) \vee (\neg \mathtt{Close}(2) \wedge \neg \mathtt{Open}(1)) \in \text{GMFOTL}$ and the trace $\sigma_6 = ((0, \{(\mathtt{Open}, (1))\}))$. Only the left disjunct is satisfied. Hence, applying the above strategy would produce the trace $\sigma_6' = ((0, \emptyset))$, which again satisfies φ_6 as it satisfies the right disjunct now. Hence, after having suppressed $(\mathtt{Open}, (1))$ we must check for satisfaction of φ_6 again, and, if necessary, select additional events to be suppressed or caused, here causing $(\mathtt{Close}, (2))$ suffices. This results in the trace $\sigma_6'' = ((0, \{(\mathtt{Close}, (2))\}))$, which now does not satisfy φ_6.

The above iterative approach, which performs a fixpoint computation, is formalized as a call to $\textsc{Fixpoint}(\sigma, \mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v})$, where $\mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v}$ performs the above case distinction for a fixed valuation v satisfying $\varphi_1 \vee \varphi_2$.

The same problem arises with existentially quantified formulae of the form $\exists x.\ \varphi_1$. For fixed v, function $\mathsf{enf}_{\mathsf{ex},\varphi_1,v}$ identifies events that must be suppressed or caused to prevent the satisfaction of φ_1 using any valuation v' extending v, and a call to $\textsc{Fixpoint}(\sigma, \mathsf{enf}_{\mathsf{ex},\varphi_1,v})$ computes the corresponding fixpoint.

Finally, for any closed, monitorable and future-free $\varphi \in \text{GMFOTL}$, we define our tentative enforcer for $\square \neg \varphi$ as

$$\hat{\mu}_\varphi(\rho) = \begin{cases} \mathsf{enf}(\varphi, \rho, \emptyset) & \text{if } |\sigma| \models_\rho \varphi \\ (\emptyset, \emptyset) & \text{otherwise.} \end{cases}$$

Example 7. Consider the GMFOTL monitorable formula

$$\varphi_7 = \underbrace{(\exists x.\ \mathtt{Open}(x) \wedge \blacklozenge_{[0,5]}\, \mathtt{Close}(x))}_{\varphi_7^1} \vee \underbrace{(\exists y.\ \neg \mathtt{Close}(y) \wedge \neg \mathtt{Close}(y)\, \mathsf{S}_{[5,\infty)}\, \mathtt{Open}(y))}_{\varphi_7^2},$$

which holds true whenever an $(\mathtt{Open}, (\mathsf{x}))$ event follows a $(\mathtt{Close}, (\mathsf{x}))$ within 5 time units for some $\mathsf{x} \in \mathbb{D}$, or there is a $(\mathtt{Close}, (\mathsf{y}))$ event for some $\mathsf{y} \in \mathbb{D}$ which is not followed by any $(\mathtt{Close}, (\mathsf{y}))$ event within 5 time units. Assume the following trace:

$$\sigma_7 = ((0, \{(\mathtt{Open}, (1))\}), (1, \{(\mathtt{Close}, (2))\}), (5, \{(\mathtt{Open}, (2))\})).$$

We have $\models_{\sigma_7} \varphi_7$: events $(\texttt{Close},(2))$ and $(\texttt{Open},(2))$ at timestamps 1 and 5 satisfy the left disjunct, while the $(\texttt{Open},(1))$ event at timestamp 0 and the lack of a $(\texttt{Close},(1))$ event between timestamps 0 and 5 make the right disjunct true. As φ_7 is closed, the set of valuations satisfying it is $\{\emptyset\}$, where $\emptyset$ denotes the empty application. We compute $\mathsf{enf}(\varphi_7,\sigma_7,\emptyset) = \textsc{Fixpoint}(\sigma_7, \mathsf{enf}_{\mathsf{or},\varphi_7^1,\varphi_7^2,\emptyset})$.

Since σ_7 satisfies both φ_7^1 and φ_7^2, we get:

$$\begin{aligned}
\mathsf{enf}_{\mathsf{or},\varphi_7^1,\varphi_7^2,\emptyset}(\sigma_7) &= \mathsf{enf}(\varphi_7^1,\sigma_7,\emptyset) \Cup \mathsf{enf}(\varphi_7^2,\sigma_7,\emptyset) \\
&= \mathsf{enf}(\texttt{Open}(x) \wedge \blacklozenge_{[0,5]}\, \texttt{Close}(x), \sigma_7, \{x \mapsto 2\}) \Cup \\
&\quad \mathsf{enf}(\neg\texttt{Close}(y) \wedge \neg\texttt{Close}(y)\, \mathsf{S}_{[5,\infty)}\, \texttt{Open}(y), \sigma_7, \{y \mapsto 1\}) \\
&= \mathsf{enf}(\texttt{Open}(x), \sigma_7, \{\{x \mapsto 2\}\}) \Cup \mathsf{enf}(\neg\texttt{Close}(y), \sigma_7, \{\{y \mapsto 1\}\}) \\
&= (\{(\texttt{Open},(2))\},\emptyset) \Cup (\emptyset, \{(\texttt{Close},(1))\}) \\
&= (\{(\texttt{Open},(2))\}, \{(\texttt{Close},(1))\}).
\end{aligned}$$

We then update σ_7:

$$\begin{aligned}
\sigma_7' &= \mathsf{update}(\sigma_7, \mathsf{enf}_{\mathsf{or},\varphi_7^1,\varphi_7^2,\emptyset}(\sigma_7)) \\
&= ((\{0,\texttt{Open},(1)\}), (1,\{\texttt{Close},(2)\}), (5,\{\texttt{Close},(1)\}))
\end{aligned}$$

and check that $\sigma_7' = \mathsf{update}(\sigma_7', \mathsf{enf}_{\mathsf{or},\varphi_7^1,\varphi_7^2,\emptyset}(\sigma_7'))$, i.e., that $\not\models_{\sigma_7'} \varphi_7$.

Hence, our enforcer computes $\hat{\mu}_{\varphi_7}(\sigma_7) = \mathsf{enf}(\varphi_7,\sigma_7,\emptyset) = (\{(\texttt{Open},(2))\}, \{(\texttt{Close},(1))\})$.

6.3 Correctness and transparency

For any monitorable, future-free and closed $\varphi \in$ GMFOTL and finite $\sigma \in \mathbb{T}^\dagger$, the enforcer $\hat{\mu}_\varphi$ always terminates.

Lemma 8. *Let $\varphi \in$ GMFOTL be closed, monitorable and strictly relative-past, and $\sigma \in \mathbb{T}^\dagger$. Then $\hat{\mu}_\varphi$ terminates on input σ.*

Proof. We first make two important observations. First, when running the enforcer, $\mathsf{enf}(\varphi,\sigma,v)$ is only called on values of φ, σ and v such that $v, |\sigma| \models_\sigma \varphi$. Given the definition of enf for atoms, this implies that all suppressable (resp. causable) events selected by enf are in the last database of σ (resp. not in the last database of σ). Second, every call to SAT with arguments ψ,σ terminates, since for all such calls it is easily seen that ψ is always monitorable and future-free.

We now show by induction on φ that for all monitorable $\varphi \in$ MFOTL, finite $\sigma \in \mathbb{T}^\dagger$, and valuation v such that $v, |\sigma| \models_\sigma \varphi$, function $\mathsf{enf}(\varphi,\sigma,v)$ terminates. The case of $\sigma = \mathsf{fv}(\varphi) = \emptyset$ then provides the desired result.

Only the cases of $\varphi = \varphi_1 \vee \varphi_2$ and $\varphi = \exists x_0.\ \varphi_1$ need to be considered, since the other cases are either trivial (for atoms) or only diverge if one of the recursive calls they perform does (for conjunctions).

If $\varphi = \varphi_1 \vee \varphi_2$, consider the traces $u_0 = \sigma, \ldots, u_i, \ldots$ produced by the iterative computation of $\textsc{Fixpoint}(\sigma, \mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v})$. By induction hypothesis, each u_{i+1} can be computed in finite time from the previous u_i. If two consecutive u_i, u_{i+1} are equal, then we

have found the limit. If they are not equal, there must exist some suppressable (resp. causable) event that is absent (resp. present) in u_{i+1} but not in u_i. This event has the form $(r,(v(t_1),\dots,v(t_n)))$ and was returned by some function call to $\mathsf{enf}(r(t_1,\dots,t_n),\sigma,v)$. If this event was suppressed, we know that $r(v(t_1),\dots,v(t_n))$ is in the last database of σ. If it was caused, function enf was executed on a negated atom. Since φ is monitorable and enf does not descend into temporal operators, this is only possible if there exist two subformulae ψ_1 and ψ_2 of φ such that $\psi_1 = \neg r(t_1,\dots,t_n),\sigma,v) \wedge_2 \psi$ and $\mathsf{fv}(\neg r(t_1,\dots,t_n)) \subseteq \mathsf{fv}(\psi_1)$, and the call to $\mathsf{enf}(\neg r(t_1,\dots,t_n),\sigma,v)$ was made by $\mathsf{enf}(\psi_1,\sigma,v)$. Now, by the above observation, this implies $v,|\sigma| \models_\sigma \psi_1$, hence $v,|\sigma| \models_\sigma \psi_2$. By Lemma 7, we obtain that for all $1 \leq i \leq n$, $t_i \in \mathsf{fv}(\neg r(t_1,\dots,t_n)) \subseteq \mathsf{fv}(\psi_1)$ is such that there exists (e,d) in the last database of σ and $1 \leq j \leq a(e)$ such that $v(t_i) = d_j$. In any case, any new event $(r,(t_1,\dots,t_n))$ that is suppressed (resp. caused) in u_{i+1} has all its t_i in the active domain of σ. As this active domain is finite and $\mathbb{E}$ is finite as well, the number of events that can be suppressed or caused in this way is also finite. Hence, the sequence $(u_i)_{i\in\mathbb{N}}$ must be equal to its limit for large enough i, and $\textsc{Fixpoint}(\sigma,\mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v})$.

If $\varphi = \exists x_0.\ \varphi_1$, the proof is essentially the same, with the following additional observation: since φ and all the nonatomic subformulae which enf descends into are monitorable, and since $\sigma \in \mathbb{T}^\dagger$, the number of valuations v satisfying φ_1 at a given timepoint must be finite. Moreover, these valuations can be computed in finite time using the algorithm from [14]. Hence, $\mathsf{enf}_{\mathsf{ex},\varphi_1,v}$ computes a union over a computable, finite set; as the function used within the union terminates by induction hypothesis, the function $\mathsf{enf}_{\mathsf{ex},\varphi_1,v}$ also terminates. The fact that the limit is reached in a finite number of steps can be proved as before.

Having established termination, we can finally prove that our enforcer is correct and transparent:

Theorem 3. *Let $\varphi \in$ GMFOTL be closed, monitorable and future-free. Then $\hat{\mu}_\varphi$ is a correct and transparent enforcer with respect to $\mathcal{L}(\square\neg\varphi) \cap \mathbb{T}^\dagger$ and $\mathbb{DB}^\dagger$.*

Proof. As we have seen in the proof of the previous proposition, all suppressable (resp. causable) events selected by enf are in the last database of σ (resp. not in the last database of σ). Therefore, $\hat{\mu}$, which always terminates by virtue of Lemma 8, defines an enforcer.

Transparency is straightforward, since for any $\sigma \in \mathbb{T}^\dagger \cap \mathcal{L}(\square\neg\varphi)$, we have $v,|\sigma| \not\models_\sigma \varphi$ and therefore $\hat{\mu}_\varphi(\sigma) = \emptyset,\emptyset$.

For correctness, we first show that for all monitorable $\psi \in$ GMFOTL, for all $\sigma \in \mathbb{T}^\dagger$, and valuation v such that that $v,|\sigma| \models_\sigma \psi$, we have $v,|\sigma| \not\models_{\mathsf{update}(\sigma,\mathsf{enf}(\psi,\sigma,v))} \psi$ $(*)$. The proof is by induction on ψ.

- If $\psi = r(t_1,\dots,t_n)$ with $r \in \mathsf{Sup}$, then $v,|\sigma| \models_\sigma \psi$ entails that $(r,(v(t_1),\dots,v(t_n)))$ occurs at timepoint $|\sigma|$. Suppressing this event ensures that φ is no longer satisfied.
- The case of $\psi = \neg r(t_1,\dots,t_n)$ with $r \in \mathsf{Cau}$ is similar.
- If $\psi = \varphi_1 \wedge \varphi_2$ where $\varphi_1 \in$ GMFOTL, then $v,|\sigma| \models_\sigma \psi$ entails $v,|\sigma| \models_\sigma \varphi_1$. By induction hypothesis on φ_1, we have $v,|\sigma| \not\models_{\mathsf{update}(\sigma,\mathsf{enf}(\varphi_1,\sigma,v))} \varphi_1$, and therefore $v,|\sigma| \not\models_{\mathsf{update}(\sigma,\mathsf{enf}(\varphi_1,\sigma,v))} \psi$. As $\mathsf{update}(\sigma,\mathsf{enf}(\psi,\sigma,v)) = \mathsf{update}(\sigma,\mathsf{enf}(\varphi_1,\sigma,v))$, we get $v,|\sigma| \not\models_{\mathsf{update}(\sigma,\mathsf{enf}(\psi,\sigma,v))} \varphi_1$.

- If $\psi = \varphi_1 \vee \varphi_2$ where $\varphi_1, \varphi_2 \in$ GMFOTL, we know by Lemma 8 that function $\mathsf{enf}(\varphi, \sigma, v)$ terminates, i.e., $\textsc{Fixpoint}(\sigma, \mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v})$ terminates, returning the limit σ_ℓ. Since σ_ℓ is computed by iteratively applying $\mathsf{update}(\cdot, \mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v}(\cdot))$ until a fixpoint is reached, we have $\mathsf{update}(\sigma_\ell, \mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v}(\sigma_\ell)) = \sigma_\ell$. As enf only suppresses (resp. causes) events that are (resp. are not) in the trace, this implies $\mathsf{enf}_{\mathsf{or},\varphi_1,\varphi_2,v}(\sigma_\ell) = \emptyset, \emptyset$ and therefore $\mathsf{enf}(\varphi, \sigma_\ell, v) = \emptyset, \emptyset$.
  Now, for all $\varphi \in$ GMFOTL, σ and v such that $v, |\sigma| \models_\sigma \varphi$, a straightforward induction shows that the union of the two sets returned by $\mathsf{enf}(\varphi, \sigma, v)$ is never empty. If the two sets returned by enf are empty, then $v, |\sigma| \not\models_\sigma \varphi$.
  Therefore, $v, |\sigma_\ell| \not\models_{\sigma_\ell} \varphi$. Now $\sigma_\ell = \mathsf{update}(\sigma, \mathsf{enf}(\psi, \sigma, v))$ by definition of enf, and hence $v, |\sigma| \not\models_{\mathsf{update}(\sigma, \mathsf{enf}(\psi,\sigma,v))} \varphi$.
- If $\psi = \exists x_0.\ \varphi_1$ where $x_0 \in \mathsf{fv}(\varphi_1)$ and $\varphi_1 \in$ GMFOTL, the proof is similar to the previous case.

Now, let $\sigma \in \mathcal{L}(\Box\neg\varphi) \cap \mathbb{T}^\dagger_f$, $\tau \geq \mathsf{lts}(\sigma)$, and $D \in \mathbb{DB}^\dagger$. Let $\sigma' = \sigma \cdot ((\tau, D))$, $\sigma'' = \mathsf{update}(\sigma', \hat{\mu}_\varphi(\sigma')) \in \mathbb{T}^\dagger_f$. Since $\sigma \in \mathcal{L}(\Box\neg\varphi)$, then for all $1 \leq i \leq |\sigma|$, $i \not\models_\sigma \varphi$. Moreover, σ,σ' and σ'' coincide on their first $|\sigma|$ databases. Hence, for all $1 \leq i \leq |\sigma|$, $i \not\models_{\sigma''} \varphi$. If $|\sigma'| \models_{\sigma'} \varphi$, then $\sigma'' = \mathsf{update}(\sigma', \hat{\mu}_\varphi(\sigma')) = \mathsf{update}(\sigma', (\emptyset, \emptyset)) = \sigma'$ and hence $|\sigma'| \models_{\sigma''} \varphi$. Otherwise, $\sigma'' = \mathsf{update}(\sigma', \hat{\mu}_\varphi(\sigma')) = \mathsf{update}(\sigma', \mathsf{enf}(\varphi, \sigma', \emptyset))$. By $(*)$, we get $|\sigma'| \models_{\sigma''} \varphi$. We conclude that $\sigma'' \in \mathcal{L}(\Box\neg\varphi) \cap \mathbb{T}^\dagger_f$. Hence, $\hat{\mu}_\varphi$ is a correct enforcer with respect to $\mathcal{L}(\Box\neg\varphi) \cap \mathbb{T}^\dagger$ and $\mathbb{DB}^\dagger$.

At this point, it is worth reflecting on the effect the assumption $\mathsf{Sup} \cap \mathsf{Cau} = \emptyset$ has on the correctness of our enforcer. In general, dropping this assumption results in some non-enforceable formula being equivalent to some formula $\Box\neg\psi$ with $\psi \in$ GMFOTL; thus, Theorem 2 no longer holds. For example, a formula such as $\varphi_7 = \Box\neg(\mathtt{C} \vee \neg\mathtt{C})$ where $\mathtt{C} \in \mathsf{Sup} \cap \mathsf{Cau}$ and $a(\mathtt{C}) = 0$ is not enforceable: given an initially empty trace—on which, by convention, φ_7 evaluates to true—adding any first timepoint makes the formula false, since $\neg(\mathtt{C} \vee \neg\mathtt{C}) \equiv \bot$. This rules out enforceability, which requires that appending only-observable events to a valid trace does not lead to a violation.

To understand why $\mathsf{Sup} \cap \mathsf{Cau} = \emptyset$ ensures correctness of our algorithm, consider the behavior of $\hat{\mu}_{\varphi_7}$ for the (non-enforceable) formula φ_7 above on the trace $\sigma_7 = ((\{\mathtt{C}\}, 0))$. The enforcer calls $\textsc{Fixpoint}(\sigma_7, \mathsf{enf}_{\mathsf{or},\mathtt{C},\neg\mathtt{C},\emptyset})$, which itself calls $\mathsf{enf}_{\mathsf{or},\mathtt{C},\neg\mathtt{C},\emptyset}(\sigma_7)$. This routine finds out that only the left disjunct C is satisfied, and returns the actions $(D^-, D^+) = (\{C\}, \emptyset)$. We get $\sigma'_7 = ((0, \emptyset))$ and call $\mathsf{enf}_{\mathsf{or},\mathtt{C},\neg\mathtt{C}}(\sigma_7)$ again to find a fixpoint. Now, the second disjunct is not satisfied, leading to the actions $(D^-, D^+) = (\emptyset, \{C\})$ and to the updated trace $\sigma''_7 = ((0, \{\mathtt{C}\})) = \sigma_7$. The same process repeats indefinitely.

When $\mathsf{Sup} \cap \mathsf{Cau} = \emptyset$, such a behavior is avoided. Since only suppressable events are suppressed and causable events caused, and since suppressable and causable events are disjoint, the algorithm will never try to suppress (resp. cause) an event that it has previously caused (resp. suppressed). Hence, the sets of caused and suppressed events can only grow during the fixpoint computation. This ensures termination, as any new iteration except the last one must compute at least one new event to cause or suppress.

Note that the assumption $\mathsf{Sup} \cap \mathsf{Cau} = \emptyset$ can be relaxed if we additionally require each suppressable *and* causable event to appear only with, or only without, a negation in

the formula. In the definition of enf, each element from $\mathsf{Sup} \cap \mathsf{Cau}$ can then be considered to belong to Sup or Cau only.

7 Implementation

We have implemented our enforcer in EnfPoly tool [36], which extends the MonPoly tool [15] with ca. 500 lines of OCaml code. Users can specify suppressable and causable events by adding "-" or "+" after the corresponding event description in the signature.

Example 8. The example signature Σ can be specified as:
`Open(int)- Close(int)+ Knock(int)`

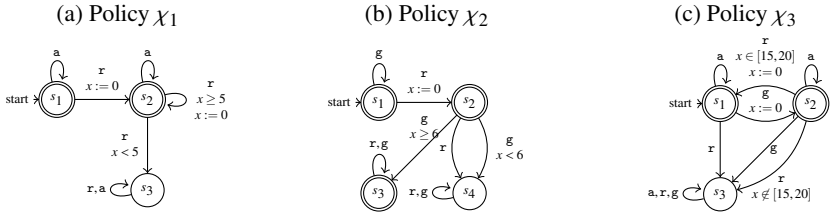
Events that are both enforceable and causable can be specified, e.g. as `SomeE+-`. In this case, for each formula to be enforced, a simple constraint-solving procedure is used to determine whether each such event can be considered only enforceable or only causable in the context of the current formula.

Strictly relative-past MFOTL Recall that determining whether a $\text{MFOTL}^{\mathcal{F}}_{\square}$ formula is enforceable is undecidable. Therefore, we developed a syntactical approximation of future-free formulae, called *strictly relative-past* formulae, which EnfPoly uses in practice. We formally define the fragment in Appendix A. Intuitively, all formulae that use only past temporal operators (i.e. *past-only MFOTL*) are strictly relative-past. Additionally, the strictly relative-past fragment contains many non-past formulae which can be statically checked for non depending on the future. For example, $\varphi_8 = \blacklozenge_{[5,+\infty)}(\texttt{Close}(2)\,\mathsf{U}_{[0,5)}\,\texttt{Open}(3))$ is relative-past, but not past-only. Observe that the intervals of the temporal operators of φ_8 ensure that its truth value does not depend on future events: the evaluation of φ_8 at timestamp τ uses `Close` events from timestamps $\leq \tau - 5$, and `Open` from timestamps $< \tau - 5 + 5 = \tau$, which all lie in the past.

To enforce a formula of the form $\square\neg\varphi$, EnfPoly checks if φ is closed, in GMFOTL, and strict relative-past. Associative and commutative rewriting is used to relax the GMFOTL membership conditions in conjuncts. Then, the enforcement loop starts. At every timepoint, the enforcer reacts either with `OK`, if there is no violation, or with the set of events to cause and a set of events to suppress, otherwise.

Example 9. The output of EnfPoly when enforcing formulae $\square\neg\varphi_6$ and $\square\neg\varphi_7$ (from Examples 6 and 7) on traces σ_6 and σ_7, respectively, is shown in the table below.

Formula: $\square\neg\varphi_6$, Trace: σ_6	Formula: $\square\neg\varphi_7$, Trace: σ_7
`@0 Open(1);`	`@0 Open(1);`
`[Enforcer] Suppress: Open(1)`	`[Enforcer] OK.`
`[Enforcer] Cause: Close(2)`	`@1 Close(2);`
`[Enforcer] OK.`	`[Enforcer] OK.`
	`@5 Open(2);`
	`[Enforcer] Suppress: Open(2)`
	`[Enforcer] Cause: Close(1)`
	`[Enforcer] OK.`



$\chi_1 = \Box\neg(\mathtt{r} \wedge ((\blacklozenge_{[1,5)}\,\mathtt{r}) \vee (\bullet_{[0,0]}\blacklozenge_{[0,0]}\,\mathtt{r})))$ $\quad \mathsf{Sup} = \{\mathtt{r}\}, \mathsf{Obs} = \{\mathtt{a}\}$

$\chi_2 = \Box\neg((\mathtt{g} \wedge \blacklozenge_{[0,6)}(\mathtt{r} \wedge (\neg\bullet\blacklozenge\mathtt{r}))) \vee (\mathtt{r} \wedge (\bullet(\neg\mathtt{g}\,\mathsf{S}\,(\mathtt{r} \wedge (\neg\bullet\blacklozenge\mathtt{r}))))))$ $\quad \mathsf{Sup} = \{\mathtt{r},\mathtt{g}\}, \mathsf{Obs} = \emptyset$

$\chi_3 = \Box\neg(\mathtt{r} \wedge ((\neg\mathtt{r}\,\mathsf{S}_{(20,\infty)}\,\mathtt{g}) \vee (\neg\mathtt{r}\,\mathsf{S}_{[0,15)}\,\mathtt{g}) \vee (\neg\mathtt{g}\,\mathsf{S}\,\mathtt{r}))) \vee \mathtt{g} \wedge \neg\mathtt{r}\,\mathsf{S}\,\mathtt{g})$ $\quad \mathsf{Sup} = \{\mathtt{r},\mathtt{g}\}, \mathsf{Obs} = \{\mathtt{a}\}$

Fig. 3: Policies used to compare EnfPoly to GREP

Timestamped databases (prefixed with `@`) of a trace are incrementally input to EnfPoly, while its output (prefixed with `[Enforcer]`) is shown chronologically interleved with the input. When enforcing $\Box\neg\varphi_6$ on σ_6, the enforcer immediately reacts to the the first database $\{(\mathtt{Open}, (1))\}$ at timestamp 0 with two actions: it suppresses event $(\mathtt{Open}, (1))$ and causes the event $(\mathtt{Close}, (2))$. Finally, it indicates that it has finished enforcing the formula by emitting `OK`. For $\Box\neg\varphi_7$, EnfPoly processes three timestamped databases. The first two do not violate the policy and hence there is no reaction from the enforcer (it responds with `OK`). The third database causes a violation and the enforcer suppresses event $(\mathtt{Open}, (2))$ and causes event $(\mathtt{Close}, (1))$ to satisfy the policy.

8 Evaluation

We now compare our enforcer with state-of-the-art tools. As our tool is the first one to support the enforcement of first-order temporal policies, comparison is only possible with (1) *propositional* temporal enforcers or (2) first-order temporal *monitors*.

Note that when there are no causable events in the signature, online monitoring tools can be used as online enforcers in the following way. First, before the events of every timepoint are sent to the monitor, save the monitor's internal state. Then, have the monitor process the timepoint. If the monitor detects no violation, save the monitor's state again and proceed with the next timepoint. If a violation is detected, restore the previous saved state and re-read *only the only-observable events* from the timepoint that led to a violation. When the formula to monitor is enforceable and there are no causable events in the signature, this construction always provides a valid enforcer. This approach has been used recently [6] to perform MFOTL enforcement with MonPoly.

Our evaluation aims to answer the following research questions:

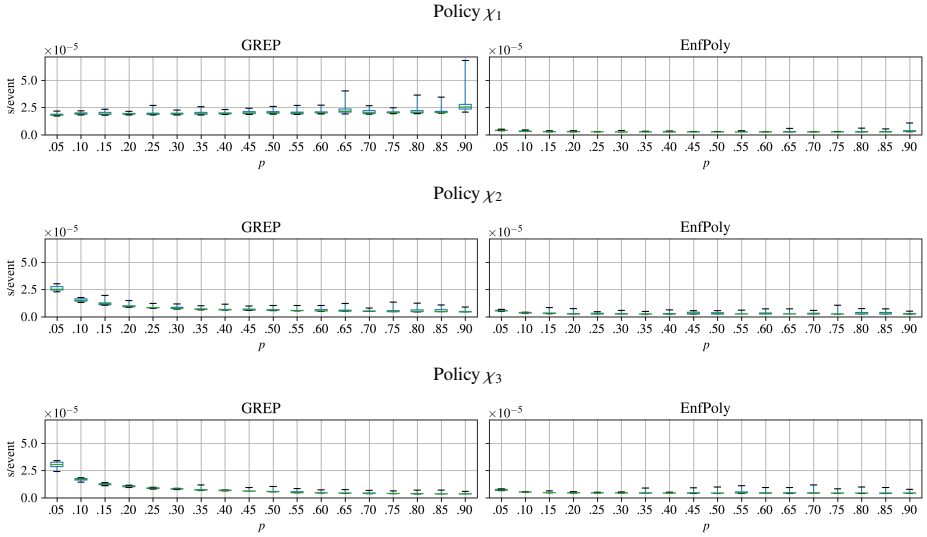
RQ1. Does EnfPoly show better performance than existing propositional enforcers?

RQ2a. Given an MFOTL formula, how much overhead does EnfPoly's enforcement cause compared to MonPoly's monitoring of the same formula?

RQ2b. Does EnfPoly show better performance in enforcing formulae over a signature with no causable events than MonPoly adapted to be an online enforcer?

For RQ1, we focus on runtime enforcement tools, which use a setup similar to ours in terms of enforcement capabilities. We compare EnfPoly to GREP [51]. GREP,

(a) Runtime performance for various choices of p, fixing $N = 25$, $n = 10$, $L = 5000$



(b) Runtime and memory over time for $N = 1000$ executions, fixing $n = 10$, $L = 5000$, $p = 0.1$

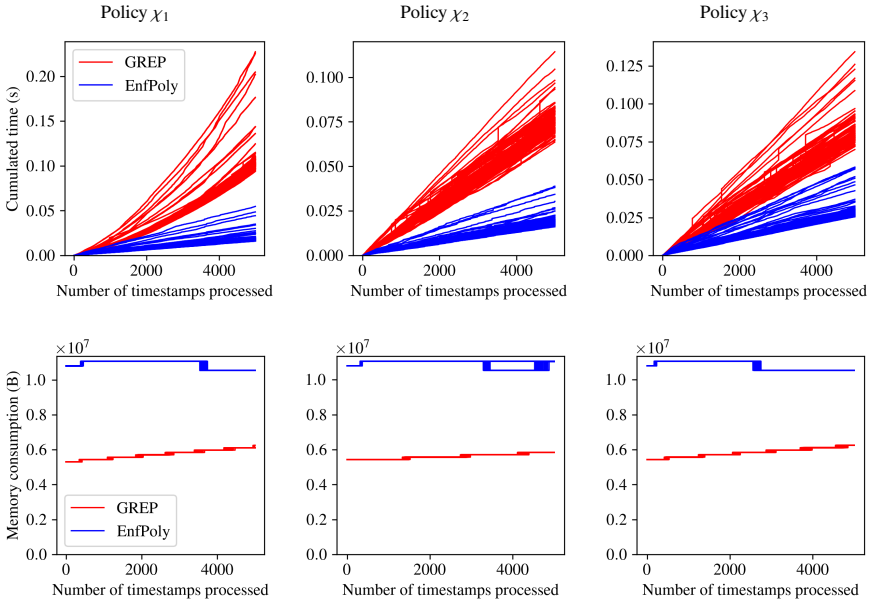


Fig. 4: Runtime and memory consumption of EnfPoly and GREP

along with TiPEX and Proactive Libraries, is one of three tools referenced in a recent survey paper [29]. GREP has been shown to outperform TiPEX by up to two orders of magnitude [51], and, unlike Proactive Libraries, it comes with an publicly available implementation. For RQ2, we compare EnfPoly to MonPoly [15].

In all experiments, we measure enforcers' memory using Python's `psutil`. We also measure enforcers' total runtime, as well as its latency, i.e., the time spent waiting for an enforcer to compute its output, which we normalize by the number of events in the trace. Speedup of our tool with respect to a tool t is computed as the difference between t's and our tool's runtime divided by t's runtime. All experiments are run on an 2.4 GHz Intel Core i5-1135G7 QuadCore CPU with 32 GB RAM.

EnfPoly vs GREP (RQ1). To compare the performance of the two enforcers, we consider the three policies presented on Figure 3, which are slightly adapted from the three benchmark examples used in [47, 51] to evaluate GREP and TiPEX. The original benchmark policies were not enforceable according to Definition 3. To restore enforceability, some previously non-accepting states were made accepting. As GREP takes as input policies specified as timed automata, we provide both an automaton and an MFOTL$^{\mathcal{F}}_{\Box}$ definition for each formula. These specifications are equivalent on traces with at most one event per database. We generate such random traces of length $L \cdot n = 50\,000$ with

- $L = 5\,000$ unique timestamps from $\{1, \ldots, L\}$;
- timestamps τ_i equal to $\lceil \frac{i}{n} \rceil$ for timepoint $i \in \{1 \ldots L \cdot n\}$, where $n = 10$;
- each timepoint containing an event with probability p and no event otherwise; and
- event names sampled uniformly from $\{a, r\}$ for χ_1 and from $\{a, g, r\}$ for χ_2 and χ_3.

For GREP, the duration of a time unit is set to 1 ms. GREP's and EnfPoly's code is instrumented to report the latency of processing inputs (i.e., excluding communication costs). Communication costs were excluded since GREP and EnfPoly receive inputs in a different format (one timepoint per line for EnfPoly, several timepoints per line for GREP). The experiment is repeated $N = 25$ times for various values of p to measure the effect of the *event rate* (i.e., the number of events per time unit) on the enforcers' performance. Note that as the signatures of χ_1, χ_2, and χ_3 contain at most three event names, we can keep the maximal number of events per timestamp small, fixing $n = 10$ and varying p only. GREP is run in online mode with the "fast" option (flag `-f`) activated.

For formulae χ_1 and χ_2, EnfPoly is faster than GREP on average for all values of p, with a speedup comprised between 40% and 90%. For χ_3, GREP outperforms EnfPoly by up to 20% for $p \geq 0.55$, but underperforms it for $p < 0.55$. The corresponding summary figures are presented in Figure 4a. Numerical data is given in Table 1 in the Appendix.

Additionally, in Figure 4b, we plot the cumulated latency and the memory consumption over time for $N = 100$ individual executions of both EnfPoly and GREP. The memory consumption of our tool is constant over time, while GREP's is linear. GREP also displays quadratic latency for policy χ_1, while EnfPoly's latency is constant in all three cases, resulting in linear cumulative latency.

EnfPoly vs MonPoly (RQ2). For RQ2a, we compare the runtime of EnfPoly with the runtime of MonPoly (used as a monitor) on the same traces and formulae. For RQ2b, we repeat this experiment using MonPoly as an enforcer in the way described above.

In both cases, we generate random enforceable and monitorable MFOTL formulae and random traces over a signature $(\mathtt{int}, \mathbb{E}, a)$ with $\mathbb{E} = \mathsf{Sup} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$ and $a(\cdot) = 1$. The random formula generator has a configurable maximal depth d and samples bounds of temporal operator intervals uniformly from $\{(i,j) \in \{0,\dots,I\}^2 \mid i \leq j\}$. Random traces of length 1 000 are generated with timestamps $1, 2, \dots, L$ with $L = 1\,000$ with no repetitions. The number of events in a database is sampled according to the binomial distribution with n trials and success probability p, while event names are sampled uniformly from $\mathbb{E}$. Finally, event's arguments are sampled uniformly from $\{1,\dots,A\}$.

Given parameters $n, A, d, I \in \mathbb{N}$ and $p \in [0,1]$, both tools are executed on pairs of independently generated random traces and enforceable and monitorable $\mathrm{MFOTL}^{\mathcal{F}}_{\Box}$ formulae with the same combinations of parameters repeated $N = 25$ times.

For all values of the parameters, enforcement with EnfPoly adds a 1–50% runtime overhead on top of the costs of monitoring with MonPoly, and does not affect memory consumption. On the other hand, using EnfPoly for enforcement is still 4 to 20 times faster than using MonPoly as an enforcer in the way described above, for a comparable memory consumption. Most of the overhead of MonPoly used as an enforcer is due to loading and saving the (complete) monitor state at each iteration, which EnfPoly avoids. Average runtime costs are under 0.1 ms per event, with most averages in the range 1–10 µs. In individual executions, both tools display constant time and memory consumption. Detailed numerical results can be found in Table 1 in the Appendix (for RQ2b), as well as in Table 1 (for RQ2a).

Discussion The above experiments show that EnfPoly, despite supporting a much larger specification language, displays a runtime and memory performance at least similar to GREP's. Our enforcer's performance is less sensitive to the choice of the input formula and consumes a constant amount of memory over time. Compared to using MonPoly as an MFOTL enforcer, EnfPoly provides a speedup of one order of magnitude. Runtime and memory consumption per event processed is stable or decreasing when more events occur simultaneously, and is not affected by longer trace sizes.

9 Conclusion

We have presented both the theory and practice of enforcing metric first-order temporal logic (MFOTL) formulae with disjoint sets of causable and suppressable events. We have characterized enforceability for MFOTL for such enforcers and proposed an efficient enforcement algorithm. Our enforcer EnfPoly extends the MonPoly monitoring tool and it is the first tool for first-order temporal logic enforcement. We have evaluated EnfPoly and showed that although it supports a more expressive language it can still outperform state-of-the-art enforcers.

In future, we plan generalize our approach to allow events that are both suppressable and causable. At this point, it remains open whether enforceability can be characterized syntactically modulo equivalence (as in Theorem 2) when this assumption is lifted. But even if no such characterization exists, in practice one could develop enforcement algorithms for larger (syntactical) fragments of enforceable policies.

Acknowledgments We thank Dmitriy Traytel for his helpful comments.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and unrealizable specifications of reactive systems. In: Ausiello, G., Dezani-Ciancaglini, M., Ronchi Della Rocca, S. (eds.) 16th International Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 372, pp. 1–17. Springer (1989)
2. Aceto, L., Cassar, I., Francalanza, A., Ingólfsdóttir, A.: On bidirectional runtime enforcement. In: Peters, K., Willemse, T. (eds.) 41st International Conference Formal Techniques for Distributed Objects, Components, and Systems (FORTE). LNCS, vol. 12719, pp. 3–21. Springer (2021)
3. Alpern, B., Schneider, F.: Recognizing safety and liveness. Dist. comp. **2**(3), 117–126 (1987)
4. Alur, R., Feder, T., Henzinger, T.: The benefits of relaxing punctuality. Journal of the ACM **43**(1), 116–146 (1996)
5. Ames, S.R., Gasser, M., Schell, R.R.: Security kernel design and implementation: An introduction. Computer **16**(7), 14–22 (1983)
6. Anonymous authors: Privacy with formal guarantees: The Databank. Tech. rep.,(2022), `https://github.com/databank-anon/databank/blob/main/paper/databank.pdf`
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,LNCS, vol.10457. Springer (2018)
9. Basin, D., Caronni, G., Ereth, S., Harvan, M., Klaedtke, F., Mantel, H.: Scalable offline monitoring of temporal specifications. Formal Methods Syst. Des. **49**(1-2), 75–108 (2016)
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.: 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 (2021)
13. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. **16**(1), 1–26 (2013)
14. Basin, D., Klaedtke, F., Müller, S., Zălinescu, E.: Monitoring metric first-order temporal properties. Journal of the ACM **62**(2), 1–45 (2015)
15. Basin, D., Klaedtke, F., Zalinescu, E.: The MonPoly monitoring tool. In: Reger, G., Havelund, K. (eds.) International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CuBES). Kalpa, vol. 3, pp. 19–28 (2017)
16. Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies. In: Workshop on Foundations of Computer Security (FCS). Citeseer (2002)
17. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) 19th International Conference Computer Aided Verification (CAV). LNCS, vol. 4590, pp. 121–125. Springer (2007)
18. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.: Acacia+, a tool for LTL synthesis. In: Madhusudan, P., Seshia, S.A. (eds.) 24th International Conference Computer Aided Verification (CAV). LNCS, vol. 7358, pp. 652–657. Springer (2012)
19. Bouyer, P., Bozzelli, L., Chevalier, F.: Controller synthesis for MTL specifications. In: Baier, C., Hermanns, H. (eds.) 17th International Conference on Concurrency Theory (CONCUR). LNCS, vol. 4137, pp. 450–464. Springer (2006)

20. Brihaye, T., Geeraerts, G., Ho, H.M., Monmege, B.: MightyL: A compositional translation from MITL to timed automata. In: 29th International Conference on Computer Aided Verification (CAV). pp. 421–440. Springer (2017)
21. Bulychev, P., David, A., Larsen, K., Li, G.: Efficient controller synthesis for a fragment of $\mathrm{MTL}_{0,\infty}$. Acta Informatica **51**(3-4), 165–192 (2014)
22. Chomicki, J.: Efficient checking of temporal integrity constraints using bounded history encoding. ACM Trans. Database Syst. **20**(2), 149–186 (Jun 1995)
23. Church, A.: An unsolvable problem of elementary number theory. American Journal of Mathematics **58**(2), 345–363 (1936)
24. Dolzhenko, E., Ligatti, J., Reddy, S.: Modeling runtime enforcement with mandatory results automata. International Journal of Information Security **14**(1), 47–60 (2015)
25. Donzé, A., Raman, V.: Blustl: Controller synthesis from signal temporal logic specifications. In: Frehse, G., Althoff, M. (eds.) 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, (ARCH@CPSWeek). EPiC, vol. 34, pp. 160–168. EasyChair (2015)
26. Ehlers, R.: Unbeast: Symbolic bounded synthesis. In: Abdulla, P.A., Leino, K.R.M. (eds.) 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 6605, pp. 272–275. Springer (2011)
27. 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)
28. Falcone, Y., Jéron, T., Marchand, H., Pinisetty, S.: Runtime enforcement of regular timed properties by suppressing and delaying events. Sci. of Comp. Programming **123**, 2–41 (2016)
29. 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)
30. Falcone, Y., Mounier, L., Fernandez, J., Richier, J.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods Syst. Des. **38**(3), 223–262 (2011)
31. Falcone, Y., Pinisetty, S.: On the runtime enforcement of timed properties. In: 19th International Conference on Runtime Verification (RV). pp. 48–69. Springer (2019)
32. Filiot, E., Jin, N., Raskin, J.: Antichains and compositional algorithms for LTL synthesis. Formal Methods Syst. Des. **39**(3), 261–296 (2011)
33. Havelund, K., Peled, D., Ulus, D.: DejaVu: A monitoring tool for first-order temporal logic. In: Workshop on Monitoring and Testing of Cyber-Physical Systems (MT-CPS). pp. 12–13. IEEE (2018)
34. Havelund, K., Peled, D., Ulus, D.: First-order temporal logic monitoring with BDDs. Formal Methods Syst. Des. **56**(1), 1–21 (2020)
35. Hofmann, T., Schupp, S.: TACoS: A tool for MTL controller synthesis. In: Calinescu, R., Pasareanu, C.S. (eds.) 19th International Conference on Software Engineering and Formal Methods (SEFM). LNCS, vol. 13085, pp. 372–379. Springer (2021)
36. Hublet, F., Basin, D., Krstić, S.: EnfPoly's development repository. `https://gitlab.ethz.ch/fhublet/mfotl-enforcement` (2022)
37. Hublet, F., Basin, D., Krstić, S.: Real-time policy enforcement with metric first-order temporal logic. Tech. rep., ETH Zürich (2022), `https://gitlab.ethz.ch/fhublet/mfotl-enforcement/-/blob/main/paper/extended.pdf`, Extended Report
38. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: 6th International Conference Formal Methods in Computer-Aided Design (FMCAD). pp. 117–124. IEEE (2006)
39. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: International Workshop on Logic and Computational Complexity. pp. 367–392. Springer (1994)

40. Krstić, S., Schneider, J.: A benchmark generator for online first-order monitoring. In: Deshmukh, J., Ničković, D. (eds.) 20th International Conference on Runtime Verification (RV). vol. 12399, pp. 482–494. Springer (2020)
41. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: 46th Symposium on Foundations of Computer Science (FOCS). pp. 531–542. IEEE (2005)
42. Li, G., Jensen, P., Larsen, K., Legay, A., Poulsen, D.: Practical controller synthesis for $\text{mtl}_{0,\infty}$. In: Erdogmus, H., Havelund, K. (eds.) 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software. pp. 102–111. ACM (2017)
43. Ligatti, J., Bauer, L., Walker, D.: Enforcing non-safety security policies with program monitors. In: European Symposium on Research in Computer Security. pp. 355–373. Springer (2005)
44. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. **12**(3), 1–41 (2009)
45. Maler, O., Ničković, D., Pnueli, A.: From MITL to timed automata. In: International conference on formal modeling and analysis of timed systems. pp. 274–289. Springer (2006)
46. Peter, H., Ehlers, R., Mattmüller, R.: Synthia: Verification and synthesis for timed automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) 23rd International Conference on Computer Aided Verification (CAV). LNCS, vol. 6806, pp. 649–655. Springer (2011)
47. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H.: Tipex: A tool chain for timed property enforcement during execution. In: Runtime Verification. pp. 306–320. Springer (2015)
48. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: 16th ACM Symposium on Principles of Programming Languages (POPL). pp. 179–190. ACM (1989)
49. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Ausiello, G., Dezani-Ciancaglini, M., Rocca, S.R.D. (eds.) 16th International Colloquium on Automata, Languages and Programming (ICALP). LNCS, vol. 372, pp. 652–671. Springer (1989)
50. Raman, V., Donzé, A., Sadigh, D., Murray, R., Seshia, S.: Reactive synthesis from signal temporal logic specifications. In: Girard, A., Sankaranarayanan, S. (eds.) 18th International Conference on Hybrid Systems: Computation and Control (HSCC). pp. 239–248. ACM (2015)
51. Renard, M., Rollet, A., Falcone, Y.: GREP: games for the runtime enforcement of properties. In: Yevtushenko, N., Cavalli, A., Yenigün, H. (eds.) 29th International Conference on Testing Software and Systems (ICTSS). LNCS, vol. 10533, pp. 259–275. Springer (2017)
52. Riganelli, O., Micucci, D., Mariani, L.: Policy enforcement with proactive libraries. In: 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 182–192. IEEE (2017)
53. Rushby, J.: Design and verification of secure systems. In: Howard, J., Reed, D. (eds.) 8th Symposium on Operating System Principles (SOSP). pp. 12–21. ACM (1981)
54. Rushby, J.: Kernels for safety. Safe and Secure Computing Systems pp. 210–220 (1989)
55. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) 5th International Symposium Automated Technology for Verification and Analysis (ATVA). LNCS, vol. 4762, pp. 474–488. Springer (2007)
56. Schneider, F.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
57. 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)
58. Turing, A.: On computable numbers, with an application to the entscheidungsproblem. London mathematical society **2**(1), 230–265 (1937)
59. Zhu, S., Tabajara, L., Li, J., Pu, G., Vardi, M.: A symbolic approach to safety LTL synthesis. In: Strichman, O., Tzoref-Brill, R. (eds.) 13th International Haifa Verification Conference (HVC). LNCS, vol. 10629, pp. 147–162. Springer (2017)

A A rich syntactical subset of future-free MFOTL

Basin *et al.* introduced a notion of *relative interval* providing an overapproximation of the set of (relative) timestamps around the present which are relevant for formula evaluation.

Definition 9 (Relative interval, [9]). *The* relative interval *function* $s : \text{MFOTL} \to \mathbb{I}$ *is defined inductively by*

$$
\begin{aligned}
s(t_1 \approx t_2) &= \{0\} \\
s(r(t_1,\dots,t_n)) &= \{0\} \\
s(\neg\varphi) &= s(\varphi) \\
s(\varphi \vee \psi) &= s(\varphi) \sqcup s(\psi) \\
s(\exists x.\ \varphi) &= s(\varphi) \\
s(\varphi\, \mathsf{S}_{[a,b)}\, \psi) &= (-b,0] \sqcup ((-b,0] + s(\varphi)) \sqcup ((-b,-a] + s(\psi)) && \varphi \neq \top \\
s(\varphi\, \mathsf{U}_{[a,b)}\, \psi) &= [0,b) \sqcup ([0,b) + s(\varphi)) \sqcup ([a,b) + s(\psi)) && \varphi \neq \top \\
s(\bullet_{[a,b)}\, \psi) &= (-b,0] \sqcup ((-b,-a] + s(\psi)) \\
s(\bigcirc_{[a,b)}\, \psi) &= [0,b] \sqcup ([a,b) + s(\psi))
\end{aligned}
$$

where $+$ *denotes the sum on sets defined by* $A + B = \{a + b \mid a \in A, b \in B\}$ *and* $A \sqcup B$ *is the smallest element of* $\mathbb{I}$ *containing both* A *and* B*.*

For example, the relative interval of $\varphi_1 = \forall d.\ \texttt{Open}(d) \Rightarrow \Diamond_{[2,5]}\, \texttt{Close}(d)$ is $s(\varphi_1) = [0,5]$, while the relative interval of $\varphi_2 = \forall d.\ \blacklozenge_{[0,100]}(\texttt{Open}(d) \wedge \neg \Diamond_{[2,5]}\, \texttt{Close}(d)) \Rightarrow \texttt{Open}(d)$ is $s(\varphi_2) = [-100,5]$. We define:

Definition 10 (Relative-past formulae). *An MFOTL formula is* relative-past *iff its relative interval is a subset of* $(-\infty, 0]$.

Clearly, all relative-past formulae are future-bounded, since the relative interval of any non future-bounded formula contains $[0,+\infty)$.

It may seem that the truth value of an MFOTL formula φ may depend on events taking place "in the future" (i.e., after the current timepoint) iff $s(\varphi) \not\subseteq (-\infty, 0]$. In fact, this condition is necessary, but not sufficient: for example, $\varphi_2 = \Diamond_{\{0\}}\, \exists d.\ \texttt{Open}(d)$ is such that $s(\varphi_2) = \{0\} \subseteq (-\infty, 0]$, but the truth value of φ_2 may depend on events taking place at future timepoints, as long as the timestamps of these timepoints are the same as the present timestamp.

To handle this case, we introduce a notion of *strictly relative-past* formulae. A relative-past formula φ is strictly relative-past iff we cannot find any sequence of nested subformulae of φ all guarded by past and (at least one) future operators that contains predicates to be evaluated at timepoint 0. More formally:

Definition 11 (Strictly relative-past formulae). *An MFOTL formula* φ *is* strictly relative-past *iff it is relative-past and there exists no sequences* $\varphi_1, \dots, \varphi_k$ *of subformulae of* φ *and* $O^1_{I_1}, \dots, O^{k-1}_{I_{k-1}}$ *of metric temporal operators such that*

1. $\varphi_k = \varphi$*;*
2. *For all* $i \in \{1, \dots, k-1\}$*, we have either*

- $\varphi_{i+1} = f(O^i_{I_i}(\varphi_i), \psi_1, \ldots, \psi_p)$ *or*
- $\varphi_{i+1} = f(\psi_0 O^i_{I_i} \varphi_i, \psi_1, \ldots, \psi_p)$ *or*
- $\varphi_{i+1} = f(\varphi_i O^i_{I_i} \psi_0, \psi_1, \ldots, \psi_p)$,

where $\psi_0, \ldots, \psi_p \in$ MFOTL*, and* f *is a first-order formula;*

3. *At least one of the* O^i *is a future operator;*
4. $0 \in \alpha_i I_1 + \alpha_2 I_2 + \cdots + \alpha_{k-1} I_{k-1}$*, where* $\alpha_i = -1$ *if* O^i *is a past operator and* $\alpha_i = 1$ *otherwise.*

Example 10. The formula $\varphi_3 = \blacklozenge_{\{0\}} \texttt{Open}(1) \vee \blacklozenge_{[5,+\infty)}(\texttt{Close}(2) \mathsf{U}_{[0,5)} \texttt{Open}(3))$ is strictly relative-past. The only sequences that fulfill conditions 1–3 above are $\varphi^1_3 = \texttt{Close}(2)$ or $\texttt{Open}()$, $\varphi^2_3 = \texttt{Close}(2) \mathsf{U}_{[0,5)} \texttt{Open}(3)$ and $\varphi^3_3 = \blacklozenge_{\{0\}} \texttt{Open}(1) \vee \blacklozenge_{[5,+\infty)}(\texttt{Close}(2) \mathsf{U}_{[0,5)}$ $\texttt{Open}(3)) = \varphi$, but $I_1 - I_2 = [0,5) - [5,+\infty) = (-\infty, 0) \not\ni 0$. In contrast, the formula $\varphi_4 = \blacklozenge_{[4,+\infty)}(\texttt{Close}(4) \mathsf{U}_{[0,5)} \texttt{Open}(5))$ is relative-past but not strictly relative-past: we have $s(\varphi_4) = (-\infty, 0]$ but there exists $\varphi^1_4 = \texttt{Close}(4)$, $\varphi^2_4 = \texttt{Close}(4) \mathsf{U}_{[0,5)} \texttt{Open}(5)$ and $\varphi^3_4 = \varphi_4$ such that conditions 1–3 are fulfilled and $I_1 - I_2 = [0,5) - [4,+\infty) = (-\infty, 0] \ni 0$.

We have the following sufficient condition:

Lemma 9. *Any strictly relative-past MFOTL formula is future-free.*

Proof. Let $\varphi \in$ MFOTL be strictly relative-past. Let $\sigma = (\sigma_1, \ldots, \sigma_k) \in \mathbb{T}$.

For every $1 \leq i \leq k$, let $(\tau_i, D_i) := \sigma_i$.

One can show by straightforward induction that for all $i \leq k$, $v, i \models_\sigma \varphi \Leftrightarrow v, i \models_{\sigma'} \varphi$ where $\sigma' = ((\tau_j, D_j) \mid 1 \leq j \leq k, \tau_j \in i + s(\phi))$. If $\tau_{i+1} > \tau_i$, this concludes the proof, as φ, being strictly relative-past, is such that $s(\varphi) \subseteq (-\infty, 0]$.

If $\tau_i = \tau_{i+1} = \cdots = \tau_{i+\ell} \neq \tau_{i+\ell+1}$, we must additionally check that events logged at $i, i+1, \ldots, i+\ell$ do not affect the verdict. If they do, the formula φ must contain a future operator (as otherwise, clearly, only past timepoints would matter), as well as a sequence of subformulae as in Definition 11 whose summed intervals contain 0. But then, φ is no longer *strictly* past only. This provides the desired result.

Note that the converse does not hold: there exist nontrivial formulae, such as $\blacklozenge_{\{1\}}(\forall x.\, \texttt{Open}(x) \Rightarrow \lozenge_{\{1\}}(\texttt{Close}(x) \wedge \neg \bullet_{\{0\}} \top))$, which are future-free but not strictly relative-past, and do not possess obvious strictly relative-past equivalents.[2] However, when future-freeness needs to be checked statically, one must inevitably consider a subset of future-free MFOTL rather than full future-free MFOTL. The reason for this is contained in the following proposition:

Lemma 10. *If* $\mathbb{E}$ *contains at least one event of arity 2, the set of future-free MFOTL formulae is not computable.*

Proof. For each closed $\varphi \in$ FOL, we can construct $\psi := \lozenge_{\{1\}} \neg \varphi \in$ MFOTL, which is future-free iff φ is universally valid. Hence, if we could decide future-freeness of MFOTL formulae, we could also solve the Entscheidungsproblem of FOL, which is impossible by virtue of Church's Theorem if $\mathbb{E}$ contains an event of arity 2 [23, 58].

[2] We conjecture that the provided formula does not have a relative-past equivalent.

$$\frac{t_1 \in \mathbb{D} \vee t_2 \in \mathbb{D}}{\mathsf{mf}(t_1 \approx t_2)} \qquad \frac{}{\mathsf{mf}(r(t_1,\ldots,t_n))}$$

$$\frac{\mathsf{mf}(\varphi_1) \quad \mathsf{mf}(\varphi_2)}{\mathsf{mf}(\varphi_1 \wedge \varphi_2)} \qquad \frac{\mathsf{mf}(\varphi_1) \quad \mathsf{mf}(\varphi_2) \quad \mathsf{fv}(\varphi_1) \subseteq \mathsf{fv}(\varphi_2)}{\mathsf{mf}(\neg\varphi_1 \wedge \varphi_2)}$$

$$\frac{\mathsf{mf}(\varphi_1) \quad \mathsf{mf}(\varphi_2) \quad \mathsf{fv}(\varphi_1) = \mathsf{fv}(\varphi_2)}{\mathsf{mf}(\varphi_1 \vee \varphi_2)} \qquad \frac{\mathsf{mf}(\varphi_1) \quad x_0 \in \mathsf{fv}(\varphi_1)}{\mathsf{mf}(\exists x_0.\, \varphi_1)}$$

$$\frac{\mathsf{mf}(\varphi_1) \quad \mathsf{mf}(\varphi_2) \quad \mathsf{fv}(\varphi_1) \subseteq \mathsf{fv}(\varphi_2)}{\mathsf{mf}(\varphi_1\, \mathsf{S}_I\, \varphi_2)} \qquad \frac{\mathsf{mf}(\varphi_1) \quad \mathsf{mf}(\varphi_2) \quad \mathsf{fv}(\varphi_1) \subseteq \mathsf{fv}(\varphi_2)}{\mathsf{mf}(\neg\varphi_1\, \mathsf{S}_I\, \varphi_2)}$$

$$\frac{\mathsf{mf}(\varphi_1) \quad \mathsf{mf}(\varphi_2) \quad \mathsf{fv}(\varphi_1) \subseteq \mathsf{fv}(\varphi_2) \quad \sup I < \infty}{\mathsf{mf}(\varphi_1\, \mathsf{U}_I\, \varphi_2)} \qquad \frac{\mathsf{mf}(\varphi_1) \quad \mathsf{mf}(\varphi_2) \quad \mathsf{fv}(\varphi_1) \subseteq \mathsf{fv}(\varphi_2) \quad \sup I < \infty}{\mathsf{mf}(\neg\varphi_1\, \mathsf{U}_I\, \varphi_2)}$$

$$\frac{\mathsf{mf}(\varphi)}{\mathsf{mf}(\bullet_I\, \varphi)} \qquad \frac{\mathsf{mf}(\varphi)}{\mathsf{mf}(\bigcirc_I\, \varphi)}$$

$$\frac{\mathsf{mf}(\varphi) \quad \mathsf{fv}(\varphi) = \emptyset}{\mathsf{mf}(\neg\varphi)}$$

Fig. 5: Monitorable fragment of MFOTL, encoded as a predicate mf

B Evaluation data

Table 1 shows the raw evaluation data produced by our experiments. The table on the left contains the data obtained when answering RQ1, while the data in the table on the right is obtained when answering RQ2. In the former we use three policies χ_1, χ_2, and χ_3, while in the latter we generate random enforceable and monitorable MFOTL formulae.

Parameter d is the depth of the generated random formulae, while I defines the sample space for the bounds of temporal operator intervals: $\{(i,j) \in \{0,\ldots,I\}^2 \mid i \leq j\}$.

Random traces have length $L \cdot n$ with timestamps $1,2,\ldots,L$, each repeated n times. Event names are sampled uniformly from $\mathbb{E} = \{\mathtt{A},\mathtt{B},\mathtt{C}\}$, while their arguments are sampled uniformly from $\{1,\ldots,A\}$. The number of events in a database is sampled according to the binomial distribution with n trials and success probability p.

Given parameters $n,A,d,I \in \mathbb{N}$ and $p \in [0,1]$, both tools are executed on pairs of independently generated random traces and enforceable and monitorable MFOTL$^{\mathcal{F}}_{\Box}$ formulae with the same combinations of parameters repeated N times.

Table 1: Mean runtime performance (standard deviation) for various parameter values

RQ1. EnfPoly vs. GREP: $N = 25, n = 10, L = 5000$

	χ_1				χ_2				χ_3			
p	GREP s/event	EnfPoly s/event	Speedup %	p	GREP s/event	EnfPoly s/event	Speedup %	p	GREP s/event	EnfPoly s/event	Speedup %
.05	1.86e-05 (1.01e-06)	4.16e-06 (4.14e-07)	77.7% (2.0%)	.05	2.60e-05 (2.01e-06)	5.56e-06 (4.45e-07)	78.5% (2.3%)	.05	3.03e-05 (2.51e-06)	7.18e-06 (4.70e-07)	76.1% (2.5%)
.10	1.95e-05 (1.05e-06)	3.21e-06 (3.48e-07)	83.6% (1.6%)	.10	1.55e-05 (1.25e-06)	3.63e-06 (3.14e-07)	76.3% (3.1%)	.10	1.68e-05 (1.09e-06)	5.20e-06 (1.64e-07)	69.0% (2.4%)
.15	1.94e-05 (1.18e-06)	2.90e-06 (2.34e-07)	85.1% (1.1%)	.15	1.24e-05 (1.97e-06)	3.58e-06 (1.47e-06)	71.5% (8.1%)	.15	1.25e-05 (7.15e-07)	4.76e-06 (3.80e-07)	61.8% (4.0%)
.20	1.93e-05 (7.45e-07)	2.82e-06 (2.79e-07)	85.3% (1.5%)	.20	1.01e-05 (1.53e-06)	3.18e-06 (1.09e-06)	68.7% (6.7%)	.20	1.05e-05 (5.64e-07)	4.50e-06 (2.33e-07)	57.0% (3.5%)
.25	1.99e-05 (2.02e-06)	2.72e-06 (1.33e-07)	86.2% (1.4%)	.25	8.66e-06 (1.07e-06)	3.00e-06 (6.34e-07)	65.5% (4.9%)	.25	8.87e-06 (4.87e-07)	4.54e-06 (3.08e-07)	48.7% (4.2%)
.30	1.95e-05 (1.07e-06)	2.74e-06 (2.90e-07)	86.0% (1.5%)	.30	8.37e-06 (1.34e-06)	3.10e-06 (9.00e-07)	63.3% (6.1%)	.30	8.14e-06 (3.74e-07)	4.49e-06 (2.35e-07)	44.7% (4.2%)
.35	1.98e-05 (1.51e-06)	2.66e-06 (1.94e-07)	86.5% (1.2%)	.35	7.28e-06 (8.33e-07)	2.83e-06 (6.38e-07)	61.3% (5.6%)	.35	7.60e-06 (1.14e-06)	4.83e-06 (1.32e-06)	37.0% (8.8%)
.40	1.99e-05 (1.10e-06)	2.77e-06 (2.92e-07)	86.0% (1.8%)	.40	6.88e-06 (1.25e-06)	3.04e-06 (9.51e-07)	56.4% (6.6%)	.40	6.76e-06 (2.77e-07)	4.34e-06 (2.30e-07)	35.7% (3.6%)
.45	2.04e-05 (1.51e-06)	2.63e-06 (1.16e-07)	87.1% (1.0%)	.45	6.78e-06 (1.02e-06)	3.29e-06 (9.20e-07)	51.9% (8.5%)	.45	6.53e-06 (8.48e-07)	4.68e-06 (1.09e-06)	28.7% (9.0%)
.50	2.08e-05 (1.78e-06)	2.65e-06 (1.71e-07)	87.2% (1.1%)	.50	6.57e-06 (1.18e-06)	3.17e-06 (9.69e-07)	52.2% (8.5%)	.50	6.00e-06 (1.08e-06)	4.76e-06 (1.44e-06)	21.3% (12.9%)
.55	2.06e-05 (2.12e-06)	2.69e-06 (2.80e-07)	86.8% (1.8%)	.55	6.05e-06 (1.09e-06)	2.95e-06 (9.52e-07)	51.8% (8.7%)	.55	5.34e-06 (1.28e-06)	5.43e-06 (1.88e-06)	-0.4% (15.6%)
.60	2.10e-05 (1.75e-06)	2.68e-06 (1.86e-07)	87.2% (1.2%)	.60	6.19e-06 (1.57e-06)	3.17e-06 (1.21e-06)	49.6% (7.0%)	.60	4.65e-06 (7.65e-07)	4.76e-06 (1.23e-06)	-1.9% (14.6%)
.65	2.32e-05 (4.84e-06)	2.98e-06 (7.76e-07)	87.2% (1.4%)	.65	6.06e-06 (1.88e-06)	3.29e-06 (1.46e-06)	46.5% (13.2%)	.65	4.46e-06 (7.68e-07)	4.74e-06 (1.38e-06)	-5.6% (17.7%)
.70	2.11e-05 (2.04e-06)	2.70e-06 (1.76e-07)	87.1% (1.5%)	.70	5.51e-06 (8.15e-07)	3.05e-06 (8.90e-07)	45.2% (9.0%)	.70	4.33e-06 (7.80e-07)	5.09e-06 (1.82e-06)	-15.6% (21.6%)
.75	2.06e-05 (1.23e-06)	2.62e-06 (1.33e-07)	87.3% (1.0%)	.75	5.63e-06 (1.87e-06)	3.32e-06 (1.90e-06)	41.4% (27.7%)	.75	4.16e-06 (6.60e-07)	4.61e-06 (1.16e-06)	-10.3% (14.5%)
.80	2.22e-05 (4.37e-06)	2.92e-06 (8.19e-07)	86.9% (1.4%)	.80	5.77e-06 (1.94e-06)	3.50e-06 (1.58e-06)	40.2% (13.6%)	.80	4.05e-06 (7.03e-07)	4.67e-06 (1.38e-06)	-14.5% (19.4%)
.85	2.19e-05 (3.68e-06)	2.94e-06 (7.48e-07)	86.6% (2.0%)	.85	5.69e-06 (1.58e-06)	3.42e-06 (1.37e-06)	40.8% (12.2%)	.85	3.91e-06 (8.13e-07)	4.60e-06 (1.47e-06)	-16.5% (16.1%)
.90	2.74e-05 (8.95e-06)	3.74e-06 (1.68e-06)	86.5% (2.6%)	.90	5.01e-06 (1.06e-06)	2.87e-06 (7.66e-07)	42.7% (8.9%)	.90	3.77e-06 (5.52e-07)	4.61e-06 (1.01e-06)	-21.8% (12.1%)

RQ2b. EnfPoly vs. MonPoly used as an enforcer: $N = 25, n = 10, L = 1000$

	$I = 50, A = 16, n = 10, p = .50$				$d = 5, A = 16, n = 10, p = .50$				$d = 5, I = 50, n = 10, p = .50$			
d	MonPoly s/event	EnfPoly s/event	Speedup %	I	MonPoly s/event	EnfPoly s/event	Speedup %	A	MonPoly s/event	EnfPoly s/event	Speedup %
2	3.44e-04 (2.50e-04)	2.60e-06 (3.12e-07)	94.3% (9.8%)	1	1.90e-04 (2.17e-04)	3.36e-06 (5.58e-07)	88.5% (11.8%)	2	1.34e-04 (1.96e-04)	3.32e-06 (5.24e-07)	83.3% (10.8%)
3	3.15e-04 (2.17e-04)	3.31e-06 (1.14e-06)	95.1% (8.6%)	5	3.00e-04 (2.59e-04)	4.06e-06 (9.41e-07)	90.4% (12.2%)	4	1.92e-04 (2.60e-04)	3.54e-06 (7.41e-07)	85.4% (11.5%)
4	2.28e-04 (2.55e-04)	3.34e-06 (7.80e-07)	88.2% (11.7%)	10	2.98e-04 (2.63e-04)	4.08e-06 (1.16e-06)	90.1% (12.6%)	8	2.19e-04 (2.65e-04)	4.09e-06 (9.88e-07)	86.2% (11.8%)
5	2.68e-04 (2.49e-04)	4.71e-06 (2.59e-06)	90.3% (10.3%)	20	2.74e-04 (2.62e-04)	4.42e-06 (1.56e-06)	89.3% (12.7%)	16	2.68e-04 (2.49e-04)	4.71e-06 (2.59e-06)	90.3% (10.3%)
6	2.59e-04 (2.79e-04)	5.47e-06 (3.38e-06)	88.0% (10.5%)	50	2.68e-04 (2.49e-04)	4.71e-06 (2.59e-06)	90.3% (10.3%)	32	1.72e-04 (2.50e-04)	5.41e-06 (4.38e-06)	84.1% (10.4%)
7	1.91e-04 (2.39e-04)	8.94e-06 (4.74e-06)	84.3% (11.4%)	100	2.05e-04 (2.52e-04)	7.16e-06 (1.22e-05)	85.8% (15.6%)	64	3.46e-04 (2.66e-04)	6.69e-06 (4.97e-06)	90.2% (11.8%)
8	2.48e-04 (2.81e-04)	1.56e-05 (1.10e-05)	82.7% (12.5%)	200	2.17e-04 (2.31e-04)	5.65e-06 (4.12e-06)	88.6% (10.1%)	128	1.75e-04 (2.17e-04)	5.50e-06 (4.46e-06)	84.8% (12.0%)
				500	2.56e-04 (2.72e-04)	1.82e-05 (5.21e-05)	85.0% (14.1%)	256	1.69e-04 (2.41e-04)	5.37e-06 (4.13e-06)	82.7% (11.8%)

	$d = 5, I = 50, A = 16, p = .50$				$d = 5, I = 50, A = 16, n = 10$			
n	MonPoly s/event	EnfPoly s/event	Speedup %	p	MonPoly s/event	EnfPoly s/event	Speedup %
1	3.39e-04 (5.57e-04)	2.32e-05 (2.82e-06)	82.4% (7.8%)	0.00	1.17e-04 (1.27e-04)	3.31e-06 (1.19e-06)	87.7% (11.5%)
2	4.61e-04 (6.81e-04)	1.35e-05 (3.74e-06)	86.0% (10.7%)	0.01	7.80e-04 (5.14e-04)	1.06e-04 (1.31e-05)	82.1% (7.5%)
5	2.37e-04 (3.83e-04)	6.92e-06 (2.78e-06)	83.3% (10.6%)	0.05	6.46e-04 (7.35e-04)	2.47e-05 (5.20e-06)	87.0% (10.5%)
10	2.68e-04 (2.49e-04)	4.71e-06 (2.59e-06)	90.3% (10.3%)	0.10	4.02e-04 (4.48e-04)	1.23e-05 (2.22e-06)	88.1% (9.8%)
20	1.38e-04 (1.42e-04)	2.98e-06 (1.07e-06)	86.5% (13.1%)	0.25	3.36e-04 (3.82e-04)	8.33e-06 (9.35e-06)	87.6% (11.0%)
50	5.62e-05 (5.80e-05)	2.57e-06 (1.53e-06)	80.3% (17.4%)	0.50	2.68e-04 (2.49e-04)	4.71e-06 (2.59e-06)	90.3% (10.3%)
100	2.89e-05 (2.95e-05)	2.45e-06 (1.74e-06)	73.4% (21.4%)	0.75	2.35e-04 (1.94e-04)	3.62e-06 (1.57e-06)	90.1% (11.5%)
200	1.64e-05 (1.47e-05)	2.52e-06 (2.98e-06)	68.3% (22.8%)	0.88	1.95e-04 (1.56e-04)	3.22e-06 (7.72e-07)	90.8% (11.2%)

Table 2: Mean runtime performance (standard deviation) for various parameter values (cont'd)

RQ2a. EnfPoly vs. MonPoly (used as a monitor); $N = 25, n = 10, L = 1000$

$I = 50, A = 16, n = 10, p = .50$

d	MonPoly s/event	EnfPoly s/event	Speedup %
2	2.91e-06 (1.78e-07)	2.96e-06 (5.31e-07)	-2.0% (16.1%)
3	3.07e-06 (1.96e-07)	3.41e-06 (5.66e-07)	-11.1% (16.8%)
4	3.16e-06 (2.38e-07)	3.62e-06 (6.25e-07)	-14.8% (18.5%)
5	5.92e-06 (1.20e-06)	7.47e-06 (2.20e-06)	-25.6% (23.1%)
6	5.63e-06 (2.09e-06)	7.47e-06 (3.69e-06)	-31.5% (37.9%)
7	8.98e-06 (4.63e-06)	1.09e-05 (5.66e-06)	-21.4% (20.3%)
8	1.80e-05 (2.08e-05)	2.67e-05 (4.27e-05)	-31.6% (38.0%)

$d = 5, A = 16, n = 10, p = .50$

I	MonPoly s/event	EnfPoly s/event	Speedup %
1	3.32e-06 (2.41e-07)	3.98e-06 (7.63e-07)	-20.3% (23.1%)
5	3.44e-06 (4.67e-07)	4.02e-06 (8.98e-07)	-16.9% (20.4%)
10	3.52e-06 (4.34e-07)	4.31e-06 (9.47e-07)	-21.7% (15.9%)
20	3.76e-06 (1.03e-06)	4.28e-06 (1.14e-06)	-14.5% (16.8%)
50	5.92e-06 (1.20e-06)	7.47e-06 (2.20e-06)	-25.6% (23.1%)
100	4.02e-06 (1.13e-06)	5.37e-06 (2.02e-06)	-32.6% (25.3%)
200	3.85e-06 (1.55e-06)	5.17e-06 (2.24e-06)	-36.3% (45.3%)
500	4.58e-06 (3.88e-06)	7.80e-06 (8.63e-06)	-68.1% (127.1%)

$d = 5, A = 16, n = 10, p = .50$

A	MonPoly s/event	EnfPoly s/event	Speedup %
2	3.44e-06 (3.25e-07)	4.10e-06 (1.00e-06)	-19.4% (28.3%)
4	3.67e-06 (5.82e-07)	4.50e-06 (9.35e-07)	-22.6% (16.4%)
8	9.77e-06 (2.18e-06)	1.19e-05 (3.69e-06)	-21.2% (22.2%)
16	5.92e-06 (1.20e-06)	7.47e-06 (2.20e-06)	-25.6% (23.1%)
32	9.66e-06 (1.14e-05)	1.31e-05 (2.06e-05)	-25.4% (27.8%)
64	9.50e-06 (9.57e-06)	9.30e-06 (8.45e-06)	-9.4% (23.3%)
128	7.72e-06 (6.09e-06)	9.44e-06 (6.53e-06)	-28.1% (30.3%)
256	2.15e-05 (3.24e-05)	2.35e-05 (3.44e-05)	-16.7% (20.0%)

$d = 5, I = 50, A = 16, p = .50$

n	MonPoly s/event	EnfPoly s/event	Speedup %
1	4.22e-05 (2.93e-06)	4.32e-05 (5.23e-06)	-2.3% (9.7%)
2	2.24e-05 (3.15e-06)	2.43e-05 (5.63e-06)	-7.6% (12.1%)
5	1.06e-05 (3.97e-06)	1.26e-05 (4.50e-06)	-19.6% (17.4%)
10	5.92e-06 (1.20e-06)	7.47e-06 (2.20e-06)	-25.6% (23.1%)
20	5.14e-06 (4.09e-06)	5.42e-06 (1.98e-06)	-18.9% (27.3%)
50	4.87e-06 (9.17e-06)	4.40e-06 (2.70e-06)	-29.0% (48.0%)
100	2.54e-06 (1.27e-06)	3.53e-06 (2.14e-06)	-38.6% (59.1%)
200	1.74e-06 (8.95e-07)	2.21e-06 (1.22e-06)	-31.2% (59.6%)

$d = 5, I = 50, A = 16, n = 10$

p	MonPoly s/event	EnfPoly s/event	Speedup %
0.00	3.40e-06 (1.54e-06)	4.89e-06 (2.43e-06)	-44.7% (47.6%)
0.01	1.54e-04 (2.75e-05)	1.54e-04 (3.26e-05)	0.1% (7.9%)
0.05	4.57e-05 (5.65e-06)	4.79e-05 (8.04e-06)	-4.6% (8.7%)
0.10	2.41e-05 (6.50e-06)	2.63e-05 (8.07e-06)	-8.7% (10.1%)
0.25	9.94e-06 (1.21e-06)	1.12e-05 (1.94e-06)	-12.7% (12.7%)
0.50	5.92e-06 (1.20e-06)	7.47e-06 (2.20e-06)	-25.6% (23.1%)
0.75	4.95e-06 (1.63e-06)	6.40e-06 (2.40e-06)	-29.5% (23.1%)
0.88	3.91e-06 (1.55e-06)	4.75e-06 (1.38e-06)	-26.6% (30.0%)