HAL open science

Runtime enforcement of timed properties revisited

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, Omer Landry Nguena Timo

To cite this version:

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, et al.. Runtime enforcement of timed properties revisited. Formal Methods in System Design, 2014, 45 (3), pp.381-422. 10.1007/s10703-014-0215-y . hal-01088136

HAL Id: hal-01088136

https://inria.hal.science/hal-01088136v1

Submitted on 27 Nov 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Formal Methods in System Design manuscript No.
(will be inserted by the editor)

Runtime Enforcement of Timed Properties Revisited

Srinivas Pinisetty · Yliès Falcone · Thierry Jéron · Hervé Marchand · Antoine Rollet · Omer Nguena Timo

Received: date / Accepted: date

Abstract Runtime enforcement is a powerful technique to ensure that a running system satisfies some desired properties. Using an enforcement monitor, an (untrustworthy) input execution (in the form of a sequence of events) is modified into an output sequence that complies with a property. Over the last decade, runtime enforcement has been mainly studied in the context of untimed properties.

This paper deals with runtime enforcement of *timed properties* by revisiting the foundations of runtime enforcement when time between events matters. We propose a new enforcement paradigm where enforcement mechanisms are *time retardants*: to produce a correct output sequence, additional delays are introduced between the events of the input sequence. We consider runtime enforcement of any regular timed property defined by a timed automaton. We prove the correctness of enforcement mechanisms and prove that they enjoy two usually expected features, revisited here in the context of timed properties. The first one is *soundness* meaning that the output sequences (eventually) satisfy the required property. The second one is *transparency*, meaning that input sequences are modified in a minimal way. We also introduce two new features, i) *physical constraints* that describe how a time retardant is physically constrained when delaying a sequence of timed events, and ii) *optimality*, meaning that output sequences are produced as soon as possible. To facilitate the adoption and implementation of enforcement mechanisms, we describe them at several complementary abstraction levels. Our enforcement mechanisms have been implemented and our ex-

S. Pinisetty, T. Jéron, H. Marchand
INRIA Rennes - Bretagne Atlantique, France,
E-mail: First.Last@inria.fr

Y. Falcone
Laboratoire d'Informatique de Grenoble, Université Grenoble I, France,
Tel.: 33 4 76 82 72 14, Fax: 33 4 76 82 72 87
E-mail: Ylies.Falcone@ujf-grenoble.fr

A. Rollet
LaBRI, Université de Bordeaux - CNRS, France,
E-mail: Antoine.Rollet@labri.fr

O. Nguena Timo
CRIM - Centre de recherche informatique de Montréal, Canada
E-mail: Omer.Nguena@crim.ca

perimental results demonstrate the feasibility of runtime enforcement in a timed context and the effectiveness of the mechanisms.

Keywords runtime verification · runtime enforcement · timed properties · timed automata · software engineering

1 Introduction

Runtime verification [1–6] refers to the theories, techniques, and tools aiming at *checking* the conformance of the executions of systems under scrutiny w.r.t. some desired property. Runtime enforcement [7–10] extends runtime verification and refers to the theories, techniques, and tools aiming at *ensuring* the conformance of the executions of systems under scrutiny w.r.t. some desired property. The first step of monitoring approaches consists in instrumenting the underlying system so as to partially observe the events or the parts of its global state that may influence the property under scrutiny. A central concept is the verification or enforcement *monitor* that is generally synthesized from the property expressed in a high-level formalism. Then, the monitor can operate either *online* by receiving events in a lock-step manner with the execution of the system or *offline* by reading a log/sequence of system events/actions. When the monitor is only dedicated to verification, it is a decision procedure emitting verdicts stating the correctness of the (partial) observed trace generated from the system execution. When the monitor additionally has enforcement abilities, it corrects any incorrect execution to meet a desirable behavior (and leaves correct executions unchanged).

Three categories of runtime verification frameworks can be distinguished according to the formalism used to express the input property. In *propositional* approaches, properties refer to events taken from a finite set of propositional names. For instance, a propositional property may rule the ordering of function calls in a program. Monitoring such kind of properties has received a lot of attention. *Parametric* approaches have received a growing interest in the last five years. In this case, events in the property are augmented with formal parameters, instantiated at runtime. In *timed* approaches, the observed time between events may influence the truth-value of the property. It turns out that monitoring of timed properties (where time is continuous) is a much harder problem because of (at least) two reasons. First, modeling timed requirements requires a more complex formalism involving time as a continuous parameter. Second, when monitoring a timed property, the problem that arises is that the overhead induced by the monitor (i.e., the time spent executing the monitoring code) influences the truth-value of the monitored property. Consequently, without assumptions and limitations on the computation performed by monitors (see § *Context and Objectives*), not much information can be gained from the verdicts produced by the monitor. Few attempts have been made on monitoring systems w.r.t. timed properties (see Sec. 5 for a detailed comparison with related work). Roughly speaking, two lines of work can be distinguished: synthesis of automata-based decision procedures for timed formalisms (e.g., [1,3–5]), and, tools for runtime verification of timed properties [11,12].

In runtime enforcement, an enforcement monitor (EM) is used to transform some (possibly) incorrect execution sequence into a correct sequence w.r.t. the property of interest. In the propositional case, the transformation performed by an EM should be *sound* and *transparent*. Soundness means that the resulting sequence obeys the property. Transparency historically means that, the monitor should modify the input sequence in a minimal way (meaning that the input sequence should not be modified if it already conforms to the

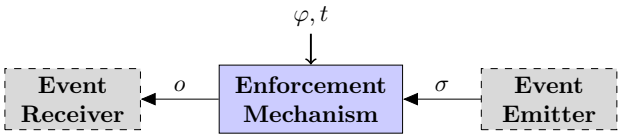


Fig. 1 Enforcement mechanism.

property). According to how a monitor is allowed to modify the input sequence (i.e., the primitives afforded to the monitor), several models of enforcement monitors have been proposed [7–10]. In a nutshell, an EM can definitely block the input sequence (as done by security automata [7]), suppress an event from the input sequence (as done by suppression automata [8]), insert an event to the input sequence (as done by insertion automata [8]), or perform any of these primitives (as is the case with edit-automata [8] or so-called generalized enforcement monitors [10]). Moreover, according to how transparency is effectively formalized, several definitions of runtime enforcement have been proposed (see [9] for an overview). The notion of time has been considered in previous runtime enforcement approaches such as in [13] for discrete-time properties, and in [14] which considers elapsing of time as a series of uncontrollable events ("ticks").

Context and Objectives. The general context is depicted in Fig. 1. We focus on *online enforcement of timed properties*. More specifically, given a timed property φ, we synthesize an enforcement mechanism that operates at runtime. To be as general as possible, an enforcement mechanism is supposed to be placed between an event emitter and an event receiver. The emitter and receiver execute asynchronously. Note, this abstract architecture is generic and can be instantiated to many concrete ones where the emitter and receiver are considered to be e.g., a program or the environment. In all cases, we assume that delaying an event from the emitter does not effect its subsequent events. This assumption is reasonable in many practical application scenarios with architectures compatible with the one described above.

An enforcement mechanism inputs a sequence of timed event σ and transforms it into a sequence of timed events o. No constraint is required on σ, whereas the enforcement mechanism ensures that o is correct w.r.t. property φ. Satisfaction of property φ by the output sequence is considered at the output of the enforcement mechanism and not at the input of the event receiver: we assume a reliable, without delay, and safe communication between the emitter and receiver. As usual in runtime enforcement, we do not consider any security, communication, nor reliability issue with events. The considered enforcement mechanisms are *time retardants*, i.e., their main enforcement primitive consists in delaying the received events. Contrary to edit-automata, enforcement mechanisms, as considered in this paper, are not able to generate nor suppress events because of i) inducing more costly computations in a timed context, and ii) delaying events is already sufficient in many application domains. Since time between events matters, we assume the enforcement mechanism to be infinitely faster than the emitter and receiver. In other words, the computation time of the enforcement mechanism is negligible: at runtime, the computation performed by the enforcement mechanism is done in zero-time. [1] Moreover, we assume that delaying the events of the emitter does not influence its behavior.

[1] As our experiments in Sec. 4 show, the computation time of the monitor upon the reception of an event is relatively low. Moreover, given some average computation time per event and a property, one can determine easily whether the computation time is negligible or not.

To sum up, given some timed property φ and an input timed word σ, we aim to study mechanisms that input σ and output a sequence o that i) satisfies φ (soundness of the mechanism), ii) has the same order of events as σ with possibly increased delays (transparency of the mechanism), iii) is released as fast as possible (optimality of the mechanism). [2]

Motivations. To the best of our knowledge, no approach focusing on enforcement of timed properties has been proposed. Motivations for extending runtime enforcement to timed properties abound. First, timed properties are more precise to specify desired behaviors of systems since they allow to explicitly state how time should elapse between events. Thus, timed properties/specifications can be particularly useful in some application domains [15]. For instance, in the context of security monitoring, enforcement monitors can be used as firewalls to prevent denial of service attacks by ensuring a minimal delay between input events (carrying some request for a protected server). On a network, enforcement monitors can be used to synchronize streams of events together, or, ensuring that a stream of events conforms to the pre-conditions of some service.

The following requirements could specify the expected behavior of a server:

R_1 *"There should be a delay of at least 5 time units between any two user requests (r)".*
R_2 *"The user should perform a successful authentication, that is, he should send a request (r) and receive a grant (g) between 10 and 15 time units".*
R_3 *"Resource grants (g) and releases (r) should alternate, starting with a grant, and every grant should be released within 15 to 20 time units".*
R_4 *"Every 10 time units, there should be a request for a resource followed by a grant. The request should occur within 5 time units".*

R_1 (resp. R_2) can be formalized as a safety (resp. co-safety) property. Safety (resp. co-safety) properties express that "something bad should never happen" (resp. "something good should happen within a finite amount of time"). Moreover, in the space of regular properties (over timed words), many interesting properties of systems are neither safety nor co-safety properties that typically specify some form of transactional behavior. Such behaviors are illustrated by requirements R_3 and R_4. In this paper, we propose to synthesize enforcement mechanisms for *all regular timed properties*.

Some motivating examples illustrating enforcement mechanisms. Let us consider again requirements R_1 and R_3. In Sec. 2 we will describe how to formalize these requirements as properties defined by timed automata. Before going further into the formal definitions, we briefly describe how an enforcement mechanism corrects an input sequence to satisfy a property. To enforce the properties, the enforcement mechanism is placed at the input of the server and operates on command-messages that it receives (destinated to the server). The enforcement mechanism releases the (correct sequence of) command-messages into the server input. In the following discussion of the examples, t denotes the current instant of time, i.e., the total time since the beginning of the considered sequence.

Let us consider requirement R_1. Let r and a be the possible actions where r denotes request of a resource. We now illustrate how an enforcement mechanism corrects the input sequence $\sigma = (1, a) \cdot (3, r) \cdot (1, r)$ (where each event is associated with a delay, indicating the time elapsed after the previous event or the system initialization for the first event). The

[2] Observe that the notions of transparency and optimality in a timed context are interpretations of the historical notion of transparency when dealing with enforcement mechanisms as time retardants: the output sequence is a minimally-delayed prefix of the input sequence.

monitor receives the first action a at $t = 1$. Since, the safety requirement is not violated, the monitor outputs it immediately (the mechanism is sound and optimal as it ouput a correct sequence which is delayed in a minimal way). The second action r is received by the monitor at $t = 4$, and the monitor outputs it immediately. The last action r, is received at $t = 5$. If the action is output immediately, then the safety requirement would be violated (since only 1 time unit has elapsed after the monitor output the previous r action), and thus the mechanism would not be sound anymore. Thus, to satisfy R_1, the monitor outputs the last r action at $t = 9$, introducing a delay of 4 time units (which is the minimal required delay as per optimality requires). The output of the monitor is $(1, a) \cdot (3, r) \cdot (5, r)$. Note that the order of actions is preserved, only the delays between them have been possibly augmented.

Let us now consider requirement R_3. Recall that g and r are the actions denoting the grant and release of the resource, respectively. We illustrate how an enforcement monitor corrects the input sequence $\sigma = (3, g) \cdot (10, r) \cdot (3, g) \cdot (5, g)$. The monitor receives the first action g at $t = 3$, the second action r at $t = 13$, etc. The monitor cannot output the first received action g because the event alone does not satisfy the requirement (and the monitor does not know yet the next events). If the next event is r, then it can output the events g followed by r, if it can choose good delays for both the events satisfying the timing constraints. At $t = 13$, the monitor can decide that the first two events can be released as output. Hence in output, the delay associated with the first g is 13 t.u. If the monitor should choose the same delay for the second action r, then the property cannot be satisfied. The monitor chooses a delay of 15 t.u. which is the minimal delay satisfying the constraint that is greater than the corresponding delay in the input sequence. When the monitor observes the second g at $t = 16$, it releases it as output, and again waits for the next event. Since the next input event observed at $t = 21$ is not r, the sequence violates the property and cannot be corrected by the monitor. Hence, after $t = 21$, the output of the monitor remains $(13, g) \cdot (15, r)$.

Contributions. We introduce runtime enforcement of timed properties. For this purpose, we adapt soundness and transparency to a timed context. We show how to synthesize runtime enforcement mechanisms for any regular property defined by a timed automaton. In contrast with previous runtime enforcement approaches [8,7,10], our mechanisms only have the primitive of being able to delay input events so that the output sequence conforms to the property. To ease the design and implementation of enforcement monitors, we describe them at several abstraction levels: the notion of enforcement function describes the behavior of an enforcement mechanism at an abstract level as an input-output relation between timed words. An enforcement monitor implements an enforcement function and describes the behavior of an enforcement mechanism in an operational way as a rule-based transition system. Enforcement algorithms describe the implementation of enforcement monitors and serve to guide the concrete implementation of enforcement mechanisms. The difficulty that arises when considering regular properties is that the aforementioned enforcement mechanisms should consider input (corrected) sequences of events that alternate between satisfying and not satisfying the underlying property. Experiments have been performed on prototype monitors to show their effectiveness and the feasibility of our approach.

This paper combines and extends the results of the two papers [16,17]. More specifically, this paper provides the following additional contributions:

- to propose a more complete and revised theoretical framework for runtime enforcement of timed properties: we have re-visited the notations, unified and simplified the main definitions;

- to propose a completely new implementation of our enforcement monitors that i) offers better performance (compared to the ones in [16]), ii) are now loosely-coupled to UPPAAL;
- to synthesize and evaluate enforcement monitors for more properties on longer executions;
- to include correctness proofs of the proposed mechanisms.

Importantly, while our synthesis techniques yield sound, transparent, and optimal enforcement mechanisms for all regular properties, some input execution sequences, while being correct, are not producible by our enforcement mechanisms. Indeed, we exhibit a notion of what we refer to as *non-enforceable properties*, for which physical time constraints prevent the associated enforcement mechanisms from preserving correct sequences. Intuitively, such undesired situation occurs when enforcing for instance co-safety properties: while the enforcement mechanism reads a correct input sequence, at the moment when it receives the event that allows it to determine that the sequence is correct, it is not possible anymore to release the first read event because the delay of the first event is now greater than the maximal value allowed by the guard on the corresponding transition in the underlying timed automaton. As we shall see, requirement R_4 can be formalized as a so-called non-enforceable timed property (Remark 3).

Paper organization. Section 2 introduces preliminaries and notation and also explains how properties are defined as timed automata on some examples. Section 3 details the enforcement mechanisms (enforcement function, monitor, and algorithm). Our prototype implementations of monitors and experiments are presented in Sec. 4. Section 5 discusses related work. Finally, conclusions and open perspectives are drawn in Sec. 6. To facilitate the reading of this paper, after each proposition we propose a sketch of proof. Full versions of the proofs along with some further notation required for the proofs are available in Appendix A.

2 Preliminaries and Notation

2.1 Untimed Notions

$\mathbb{N}$ denotes the set of non-negative integers. An alphabet is a set of elements. A (finite) word over an alphabet A is a finite sequence of elements of A. The *length* of a word σ is denoted as $|\sigma|$. The empty word over A is denoted by ϵ_A or ϵ when clear from the context. The set of all (respectively non-empty) finite words over A is denoted by A^* (respectively A^+). A *language* over A is a set $\mathcal{L} \subseteq A^*$. The concatenation of two words σ and σ' is denoted by $\sigma \cdot \sigma'$. The empty word over A is neutral for concatenation of words of A: $\forall \sigma \in A^* : \sigma \cdot \epsilon_A \stackrel{\text{def}}{=} \epsilon_A \cdot \sigma \stackrel{\text{def}}{=} \sigma$. A word σ' is a *prefix* of a word σ, denoted as $\sigma' \preccurlyeq \sigma$, whenever there exists a word σ'' such that $\sigma = \sigma' \cdot \sigma''$, and σ' is a *strict prefix* of σ denoted as $\sigma' \prec \sigma$ whenever $\sigma' \preccurlyeq \sigma \wedge |\sigma'| < |\sigma|$. For a word σ and $1 \leq i \leq |\sigma|$, the i-th letter (resp. prefix of length i, suffix starting at position i) of σ is denoted $\sigma(i)$ (respectively $\sigma_{[\cdots i]}$, $\sigma_{[i\cdots]}$) – with the convention $\sigma_{[\cdots 0]} \stackrel{\text{def}}{=} \epsilon$. Given a word σ and two integers i, j, such that $i \leq j$ and $|\sigma| \geq j$, the sub-word from index i to j is defined as $\sigma_{[i\cdots j]} \stackrel{\text{def}}{=} \sigma_{[\cdots j][i\cdots]}$. Given a word σ, the last letter, $\sigma(|\sigma|)$, is denoted $\mathrm{last}(\sigma)$. The set $\mathrm{pref}(\sigma)$ denotes the set of prefixes of σ and by extension, $\mathrm{pref}(\mathcal{L}) \stackrel{\text{def}}{=} \{\mathrm{pref}(\sigma) \mid \sigma \in \mathcal{L}\}$ is the set of prefixes of words in $\mathcal{L}$. A language $\mathcal{L}$ is said to be *prefix-closed* whenever $\mathrm{pref}(\mathcal{L}) = \mathcal{L}$ and *extension-closed*

whenever $\mathcal{L} = \mathcal{L} \cdot A^*$. Given an n-tuple of symbols $e = (e_1, \ldots, e_n)$, $\Pi_i(e)$ is the projection of e on its ith element ($\Pi_i(e) \stackrel{\text{def}}{=} e_i$). The projection function is extended to sequences of tuples in the standard way.

2.2 Timed Languages and Properties as Timed Automata

Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers, and Σ a finite alphabet of *actions*. A pair $(\delta, a) \in (\mathbb{R}_{\geq 0} \times \Sigma)$ is called an *event*. We note $\text{delay}(\delta, a) \stackrel{\text{def}}{=} \delta$ and $\text{act}(\delta, a) = a$ the projections of events on delays and actions, respectively. A *timed word* over Σ is a finite sequence of events ranging over $(\mathbb{R}_{\geq 0} \times \Sigma)^*$. A *timed language* is any set $\mathcal{L} \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^*$. We consider a timed word $\sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n)$. For $i \in [2, n]$, δ_i is the delay between a_{i-1} and a_i, and, δ_1 the time elapsed before the first action a_1. Note that even though the alphabet $(\mathbb{R}_{\geq 0} \times \Sigma)$ is infinite in this case, previous notions and notations defined above (related to length, concatenation, prefix, etc) naturally extend to timed words. The *untimed projection* of σ is $\Pi_\Sigma(\sigma) \stackrel{\text{def}}{=} a_1 \cdot a_2 \cdots a_n$ in Σ^* (i.e., delays are ignored). The *duration* of a timed word σ, denoted by $\text{time}(\sigma) \stackrel{\text{def}}{=} \Sigma_{i=1}^{n} \delta_i$, is the sum of its delays.

Timed automata. A timed automaton [18] is a finite automaton extended with a finite set of real valued clocks. Let $X = \{x_1, \ldots, x_k\}$ be a finite set of *clocks*. A *clock valuation* for X is a function ν from X to $\mathbb{R}_{\geq 0}$. $\mathbb{R}_{\geq 0}^X$ denotes the valuations of clocks in X. For $\nu \in \mathbb{R}_{\geq 0}^X$ and $\delta \in \mathbb{R}_{\geq 0}$, $\nu + \delta$ is the valuation assigning $\nu(x) + \delta$ to each clock x of X. Given a set of clocks $X' \subseteq X$, $\nu[X' \leftarrow 0]$ is the clock valuation ν where all clocks in X' are assigned to 0. $\mathcal{G}(X)$ denotes the set of clock constraints defined as Boolean combinations of simple constraints of the form $x \bowtie c$ with $x \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Given $g \in \mathcal{G}(X)$ and $\nu \in \mathbb{R}_{\geq 0}^X$, we write $\nu \models g$ when g holds according to ν.

Definition 1 (Timed automata) A *timed automaton* (TA) is a tuple $\mathcal{A} = (L, l_0, X, \Sigma, \Delta, G)$, such that L is a finite set of *locations* with $l_0 \in L$ the *initial location*, X is a finite set of clocks, Σ is a finite set of *actions*, $\Delta \subseteq L \times \mathcal{G}(X) \times \Sigma \times 2^X \times L$ is the *transition relation*. $G \subseteq L$ is a set of accepting locations.

The *semantics* of a TA is a timed transition system $[\![\mathcal{A}]\!] = (Q, q_0, \Gamma, \rightarrow, F_G)$ where $Q = L \times \mathbb{R}_{\geq 0}^X$ is the (infinite) set of *states*, $q_0 = (l_0, \nu_0)$ is the initial state where ν_0 is the valuation that maps every clock in X to 0, $F_G = G \times \mathbb{R}_{\geq 0}^X$ is the set of accepting states, $\Gamma = \mathbb{R}_{\geq 0} \times \Sigma$ is the set of transition *labels*, i.e., pairs composed of a delay and an action. The transition relation $\rightarrow \subseteq Q \times \Gamma \times Q$ is a set of transitions of the form $(l, \nu) \xrightarrow{(\delta, a)} (l', \nu')$ with $\nu' = (\nu + \delta)[Y \leftarrow 0]$ whenever there exists $(l, g, a, Y, l') \in \Delta$ such that $\nu + \delta \models g$ for $\delta \in \mathbb{R}_{\geq 0}$.

In the following, we consider a timed automaton $\mathcal{A} = (L, l_0, X, \Sigma, \Delta, G)$ with its semantics $[\![\mathcal{A}]\!]$. $\mathcal{A}$ is *deterministic* whenever for any two distinct transitions (l, g_1, a, Y_1, l_1') and (l, g_2, a, Y_2, l_2') in Δ, $g_1 \wedge g_2$ is unsatisfiable. $\mathcal{A}$ is *complete* whenever for any location $l \in L$ and any action $a \in \Sigma$, the disjunction of the guards of the transitions leaving l and labeled by a evaluates to *true* (i.e., it holds according to any valuation). In the remainder of this paper, we shall consider only deterministic timed automata, and, automata refer to timed automata.

Remark 1 (Completeness and determinism) In this paper, we restrict the presentation to deterministic TAs. However, results also hold for non-deterministic TAs, with slight adaptations required to the vocabulary and when synthesizing an enforcement monitor. Regarding completeness, if no transition can be triggered upon the reception of an event, a TA implicitly moves to a non-accepting trap state (i.e., a state with no successor).

A *run* ρ from $q \in Q$ is a sequence of moves in $[\![\mathcal{A}]\!]$: $\rho = q \xrightarrow{(\delta_1,a_1)} q_1 \cdots q_{n-1} \xrightarrow{(\delta_n,a_n)} q_n$, for some $n \in \mathbb{N}$. The set of runs from $q_0 \in Q$ is denoted $\mathrm{Run}(\mathcal{A})$ and $\mathrm{Run}_{F_G}(\mathcal{A})$ denotes the subset of runs *accepted* by $\mathcal{A}$, i.e., when $q_n \in F_G$. The *trace* of a run ρ is the timed word $(\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n)$. We note $\mathcal{L}(\mathcal{A})$ the set of traces of $\mathrm{Run}(\mathcal{A})$. We extend this notation to $\mathcal{L}_{F_G}(\mathcal{A})$ in a natural way.

Regular, safety, and co-safety timed properties. In the sequel, we shall be interested in the set of regular timed properties, i.e., the timed properties that can be defined by a TA. Within the set of regular timed properties, we are interested in safety and co-safety properties. Informally, safety (resp. co-safety) properties state that "nothing bad should ever happen" (resp. "something good should happen within a finite amount of time"). In this paper, the classes are characterized as follows:

Definition 2 (Regular, safety, and co-safety properties)

- Regular properties are the properties that can be defined by languages accepted by a TA.
- Safety properties (a subset of regular properties) are the non-empty prefix-closed timed languages (i.e., the languages $\mathcal{L}$ s.t. $\mathrm{pref}(\mathcal{L}) = \mathcal{L}$), that can be defined by a TA.
- Co-safety properties (a subset of regular properties) are the non-universal extension-closed timed languages (i.e., the languages $\mathcal{L}$ s.t. $\mathcal{L} = \mathcal{L} \cdot A^*$), that can be defined by a TA.

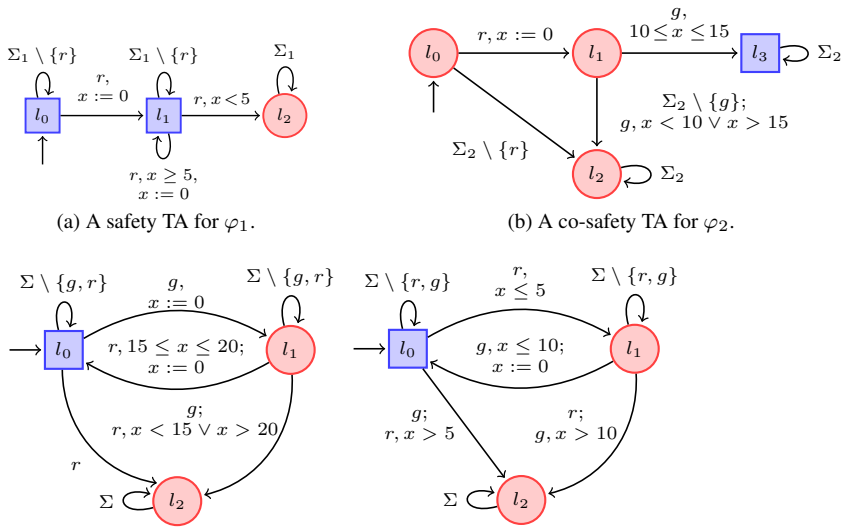
Remark 2 Usually, safety properties are defined as prefix-closed languages, and co-safety as extension-closed languages. With the usual definitions, the two properties $\emptyset$ and $(\mathbb{R}_{\geq 0} \times \Sigma)^*$ (the empty and universal properties, respectively vacuously and universally satisfied) are both, at the same time safety and co-safety properties, and are the only ones in the intersection. In this paper, to simplify the presentation and to avoid pathological cases, we separate the two classes, by considering that $(\mathbb{R}_{\geq 0} \times \Sigma)^*$ is a safety (but not a co-safety) property, and $\emptyset$ is a co-safety (but not a safety) property.

Safety and co-safety timed automata. In the sequel, we shall only consider the properties that can be defined by a deterministic timed automaton (Definition 1). Note that some of these properties can be defined using a timed temporal logic such as a subclass of MTL, which can be transformed into timed automata using the technique described in [19, 3].

We now define how to determine whether a regular property defined by a TA defines a safety or a co-safety property by examining its transition relation.

Definition 3 (Safety and co-safety TA) A complete and deterministic TA $(L, l_0, X, \Sigma, \Delta, G)$, where $G \subseteq L$ is the set of accepting locations, is said to be:

- a *safety* TA if $l_0 \in G \wedge \nexists (l, g, a, Y, l') \in \Delta : l \in L \setminus G \wedge l' \in G$;
- a *co-safety* TA if $l_0 \notin G \wedge \nexists (l, g, a, Y, l') \in \Delta : l \in G \wedge l' \in L \setminus G$.



(a) A safety TA for φ_1.

(b) A co-safety TA for φ_2.

(c) A TA for φ_3.

(d) A TA for φ_4.

Fig. 2 Examples of TA modeling timed properties.

It is easy to check that safety and co-safety TAs define safety and co-safety properties. [3]

Example 1 ((Safety and co-safety) timed automata) We consider requirements R_1, R_2, R_3, and R_4, introduced in Sec. 1. These requirements are respectively formalized as properties $\varphi_1, \varphi_2, \varphi_3$, and φ_4 defined by the timed automata in Fig. 2. Accepting locations are denoted by squares. The safety TA in Fig. 2a defines property φ_1 defined over $\Sigma_1 = \{a, r\}$. The co-safety TA in Fig. 2b defines property φ_2 defined over $\Sigma_2 = \{r, g, a\}$. The TA in Fig. 2c defines property φ_3 defined over $\Sigma_3 = \{r, g\}$. The TA in Fig. 2d defines property φ_4 defined over $\Sigma_4 = \{r, g\}$.

Combining properties using Boolean operations. The next definition, as described in [18], provides a way to combine (complete and deterministic) timed automata and will be used in the sequel to define properties expressed as a Boolean combination of other properties.

Definition 4 (Operations on timed automata) Given two properties φ_1 and φ_2 defined by TAs $\mathcal{A}_{\varphi_1} = (L_1, \ell_1^0, X_1, \Sigma, \Delta_1, G_1)$ and $\mathcal{A}_{\varphi_2} = (L_2, \ell_2^0, X_2, \Sigma, \Delta_2, G_2)$, respectively. The $\times_{op}$-product of $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$, where $op \in \{\cup, \cap\}$, is the timed automaton defined as $\mathcal{A}_{\varphi_1} \times_{op} \mathcal{A}_{\varphi_2} \stackrel{\text{def}}{=} (L, l_0, X, \Sigma, \Delta, G)$ where $L = L_1 \times L_2$, $l_0 = (l_0^1, l_0^2)$, $X = X_1 \cup X_2$ (disjoint union), $\Delta \subseteq L \times \mathcal{G}(X) \times \Sigma \times 2^X \times L$ is the *transition relation*, where $((l_1, l_2), g_1 \wedge g_2, a, Y_1 \cup Y_2, (l_1', l_2')) \in \Delta$ iff $(l_1, g_1, a, Y_1, l_1') \in \Delta_1$ and $(l_2, g_2, a, Y_2, l_2') \in \Delta_2$. G_{op} is a set of accepting locations with:

- $G_\cap = G_1 \times G_2$,
- $G_\cup = (L_1 \times G_2) \cup (G_1 \times L_2)$.

[3] As one can observe, these definitions of safety and co-safety TAs slightly differ from the usual ones by expressing constraints on the initial state. As a consequence of these constraints, consistently with Definition 2, the empty and universal properties are ruled out from the set of safety and co-safety properties, respectively.

Definition 5 (Negation of a timed automaton) Given a property φ defined by a TA $\mathcal{A}_\varphi = (L, \ell^0, X, \Sigma, \Delta, G)$ its negation is defined as $\neg\mathcal{A}_\varphi \stackrel{\text{def}}{=} (L, \ell^0, X, \Sigma, \Delta, L \setminus G)$.

The proposition below states that when performing the $\cap$-product (resp. $\cup$-product) between two TAs, it amounts to perform the intersection (resp. union) of the recognized languages.

Proposition 1 *Consider two properties φ_1 and φ_2 defined by TAs $\mathcal{A}_{\varphi_1} = (L_1, \ell_1^0, X_1, \Sigma, \Delta_1, G_1)$ and $\mathcal{A}_{\varphi_2} = (L_2, \ell_2^0, X_2, \Sigma, \Delta_2, G_2)$, respectively. The following facts hold:*

- $\mathcal{L}_{(\mathcal{A}_{\varphi_1} \times_\cap \mathcal{A}_{\varphi_2})(G_\cap)} = \varphi_1 \cap \varphi_2$,
- $\mathcal{L}_{(\mathcal{A}_{\varphi_1} \times_\cup \mathcal{A}_{\varphi_2})(G_\cup)} = \varphi_1 \cup \varphi_2$,
- $\mathcal{L}_{\neg\mathcal{A}_{\varphi_1}} = (\mathbb{R}_{\geq 0} \times \Sigma_1)^* \setminus \mathcal{L}_{(\mathcal{A}_{\varphi_1})}$,
- $\mathcal{L}_{\neg\mathcal{A}_{\varphi_2}} = (\mathbb{R}_{\geq 0} \times \Sigma_2)^* \setminus \mathcal{L}_{(\mathcal{A}_{\varphi_2})}$.

The above proposition entails that the classes of safety and co-safety properties are closed under union and intersection. However, the properties resulting of any other operation between safety, co-safety, and regular properties is a regular property. Finally, note that the negation of a safety (resp. co-safety) property is a co-safety (resp. safety) property. From the results shown in [18], the following proposition holds.

Proposition 2 (Closure of safety, co-safety and regular properties under Boolean operations)

- *Safety and co-safety properties are closed under (finite) union and intersection.*
- *The negation of a safety property is a co-safety property, and vice-versa.*
- *Regular properties are closed under Boolean operations.*

2.3 Preliminaries to Runtime Enforcement

Given $t \in \mathbb{R}_{\geq 0}$, and a timed word $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$, we define the *observation of σ at time t* as the timed word:

$$\text{obs}(\sigma, t) \stackrel{\text{def}}{=} \max_{\preccurlyeq}\{\sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid \sigma' \preccurlyeq \sigma \wedge \text{time}(\sigma') \leq t\},$$

where $\max_{\preccurlyeq}$ takes the maximal sequence according to the prefix ordering $\preccurlyeq$ (unique in this case). That is $\text{obs}(\sigma, t)$ is the longest prefix of σ of duration smaller than t. By definition, $\text{time}(\text{obs}(\sigma, t)) \leq t$, meaning that the duration of an observation at time t never exceeds t.

The *maximal strict prefix of σ that belongs to φ* is denoted as $\max^{\varphi}_{\prec,\epsilon}(\sigma)$ and defined as: $\max^{\varphi}_{\prec,\epsilon}(\sigma) \stackrel{\text{def}}{=} \max_{\preceq}\big(\{\sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid \sigma' \prec \sigma \wedge \sigma' \in \varphi\} \cup \{\epsilon\}\big)$.

Orders on timed words. Apart from the prefix order $\preccurlyeq$, we define the following partial orders on timed words:

Delaying order $\preccurlyeq_{\text{d}}$: For $\sigma, \sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$, we say that σ' *delays* σ (denoted $\sigma' \preccurlyeq_{\text{d}} \sigma$) iff

$$\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\sigma) \text{ and } \forall i \leq |\sigma'| : \text{delay}(\sigma(i)) \leq \text{delay}(\sigma'(i)),$$

which means that σ' is "a delayed prefix" of σ. This order will be used to characterize outputs w.r.t. to inputs in enforcement monitoring.

Lexical order $\preceq_{\text{lex}}$: Given any two timed words σ, σ' with same untimed projection, i.e., $\Pi_\Sigma(\sigma) = \Pi_\Sigma(\sigma')$, and any two timed events with identical actions (δ, a) and (δ', a), we define $\preceq_{\text{lex}}$ inductively as follows: $\epsilon \preceq_{\text{lex}} \epsilon$, and $(\delta, a) \cdot \sigma \preceq_{\text{lex}} (\delta', a) \cdot \sigma'$ iff $\delta \leq \delta' \vee (\delta = \delta' \wedge \sigma \preceq_{\text{lex}} \sigma')$. This ordering is defined for timed words with identical actions, and is useful to choose a unique timed word among some with same actions.

3 Enforcement Monitoring in a Timed Context

Roughly speaking, the purpose of enforcement monitoring is to read some (possibly incorrect) input sequence produced by a running system (input to the enforcement mechanism), and to transform it into an output sequence that is correct w.r.t. a property φ. To ease the design and implementation of enforcement monitoring mechanisms in a timed context, we introduce two sorts of mechanisms: enforcement functions and enforcement monitors. From an abstract point of view, an enforcement function describes the transformation of an input timed word into an output timed word according to time. An enforcement monitor is a transition system, whose input/output behavior realizes an enforcement function. In other words, an enforcement function serves as an abstract description (black-box view) of an enforcement monitor, and, an enforcement monitor is an operational description of an enforcement function.

3.1 Enforcement Functions

The input of an enforcement function is considered to be a timed word σ where every action is associated to the delay since the previous action (or the initialization). Moreover, at this abstract level, even if σ is partially known, it is considered to be fully determined from the beginning. At time t, the sequence observed by the enforcement mechanism is $\mathrm{obs}(\sigma, t)$. The output of the enforcement function at time t should be a timed word representing a sequence of actions with delays between them computed from $\mathrm{obs}(\sigma, t)$.

Definition 6 (Enforcement function) An *enforcement function* for a property φ is a function E_φ from $(\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0}$ to $(\mathbb{R}_{\geq 0} \times \Sigma)^*$.

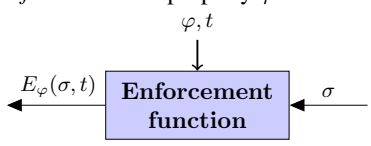


Fig. 3 Enforcement function E_φ

An enforcement function E_φ for property φ transforms some (possibly incorrect) timed word σ given as input (see Fig. 3). In this paper, we assume that enforcement mechanisms do not modify the actions they receive but are rather *time retardants*, i.e., their output at time t is a timed word $E_\varphi(\sigma, t)$ with same actions as a prefix of their observation, but with possibly increased delays between actions, in such a way that the output timed word satisfies the property.

Constraints on enforcement functions. Similarly to the untimed setting, several constraints, namely *soundness* and *transparency*, are required on how E_φ transforms timed words. Since our enforcement monitors are time retardants, the physical constraints in the following definition also apply to E_φ.

Definition 7 (Constraints on an Enforcement Mechanism) For a timed property φ, an *enforcement mechanism* behaves as a function E_φ from $(\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0}$ to $(\mathbb{R}_{\geq 0} \times \Sigma)^*$, satisfying the following constraints:

- **Physically time retardant:**

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t, t' \in \mathbb{R}_{\geq 0} : \; t \leq t' \implies E_\varphi(\sigma, t) \preccurlyeq E_\varphi(\sigma, t') \qquad (\mathbf{Phy1}).$$

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : \; \mathrm{time}(E_\varphi(\sigma, t)) \leq t \qquad (\mathbf{Phy2}).$$

- **Soundness:**

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : \; E_\varphi(\sigma, t) \neq \epsilon \implies (\exists t' \geq t : E_\varphi(\sigma, t') \models \varphi) \qquad (\mathbf{Snd}).$$

- **Transparency:**

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) \preccurlyeq_d \mathrm{obs}(\sigma, t) \qquad (\mathbf{Tr}).$$

The requirements on the enforcement function are specified by three constraints: *physically time retardant*, *soundness*, and *transparency*. **(Phy1)** means that the outputs of the enforcement function are concatenated over time, i.e., what is output cannot be modified. **(Phy2)** expresses that the duration of the output never exceeds t. Soundness **(Snd)** means that if a timed word is released as output by the enforcement function, in the future, the output of the enforcement function should satisfy property φ. In other words, no event is output before being sure that the property will be satisfied by subsequent events. In the particular case of a safety property φ, since φ is prefix closed, soundness reduces to $\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) \models \varphi$. Transparency **(Tr)** expresses that, at any time t, the output is a delayed prefix of the observed input $\mathrm{obs}(\sigma, t)$.

3.2 Functional Definition

We propose first a general definition of an enforcement function (Sec. 3.2.1) and then a simplified definition for safety properties (Sec. 3.2.2).

3.2.1 General Definition

The enforcement function can be described as a composition of functions, each performing the following steps: i) processing the input, ii) computing the delayed timed word satisfying the property, iii) and processing the output sequence. Moreover, the enforcement function describes how these functions are composed to transform an input sequence. We will then prove that it satisfies the physical, soundness, and transparency constraints.

Definition 8 (Enforcement function) The enforcement function for a property φ is $E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ defined as:

$$E_\varphi(\sigma, t) = \mathrm{obs}\Big(\Pi_1\big(\mathrm{store}(\mathrm{obs}(\sigma, t))\big), t\Big).$$

where:

- $\mathrm{store} : (\mathbb{R}_{\geq 0} \times \Sigma)^* \to (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^*$ is defined as

$$\begin{aligned} \mathrm{store}(\epsilon) &= (\epsilon, \epsilon) \\ \mathrm{store}(\sigma \cdot (\delta, a)) &= \begin{cases} (\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K, \epsilon) & \text{if } K \neq \emptyset, \\ (\sigma_s, \sigma_c \cdot (\delta, a)) & \text{otherwise,} \end{cases} \\ & \quad \text{with} \\ & \quad (\sigma_s, \sigma_c) = \mathrm{store}(\sigma), \\ & \quad K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a)), \end{aligned}$$

- $\kappa_\varphi(T, \sigma_s, \sigma_c)$ is the set defined as:

$$\kappa_\varphi(T, \sigma_s, \sigma_c) \stackrel{\mathrm{def}}{=} \{w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \\ \sigma_s \cdot w \models \varphi \wedge \mathrm{delay}(w(1)) \geq T - \mathrm{time}(\sigma_s))\}, \text{ and}$$

– $\min_{\preceq_{\text{lex,time}}}$ stands for minimal timed word according to the lexical order among the timed words with minimal duration.

In the definition of E_φ, $\text{obs}(\sigma, t)$ is the prefix of the input that has been observed at time t, and thus can be processed by the enforcement function. The store function takes as input this observation, and computes a pair of timed words, whose first component extracted by Π_1 is processed by obs to produce the output.

The first element of the output of the store function is the transformation of a prefix of the observation for which delays have been computed (the property is satisfied by this prefix by appropriate delaying); the second element is the suffix of the observation for which delays still have to be computed. The store function is defined inductively: initially, for an empty observation, both elements are empty; if σ has been observed, $\text{store}(\sigma) = (\sigma_s, \sigma_c)$, and a new event (δ, a) is observed, there are two possible cases, according to the vacuity of the set $K = \kappa_\varphi(\text{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))$ (the set of candidate timed words appropriately delaying $\sigma_c \cdot (\delta, a)$ and satisfying φ, see below):

- if $K \neq \emptyset$, among the set of timed words with minimal duration in K, the minimal timed word w.r.t the lexical order is appended to σ_s, and the second element is set to ϵ.
- otherwise, (δ, a) is appended to σ_c and σ_s is not modified.

The function κ_φ has three parameters: T (the duration of the current observation), σ_s, and σ_c. It computes the set of candidate timed words w "appropriately delaying" σ_c such that $\sigma_s \cdot w$ satisfies φ. The appropriate delaying is such that w and σ_c have identical actions, same length but delays of w are greater than or equal to those of σ_c. Moreover the delay of the first action in w should exceed the difference between the duration of the observation and the duration of σ_s. The reason for this constraint is that σ_s will be output entirely after a duration of $\text{time}(\sigma_s)$, while the decision to output w is taken after T t.u., thus a smaller value for $\text{delay}(w(1))$ would cause this delay to be elapsed before the decision is taken. Notice that upon reading an input event (δ, a), in case if no appropriate delays exist, and correcting the sub-sequence $\sigma_c \cdot (\delta, a)$ is impossible (when κ_φ is empty), then the input event (δ, a) is appended to σ_c. If the sub-sequence $\sigma_c \cdot (\delta, a)$ can be corrected, it is appended immediately to σ_s (with appropriate delays) without relying on events that will be read later. The adopted "policy" here is to correct the observation of the input sequence *as soon as possible* (see also Proposition 4 later). Consequently, the input sequence is treated as a series of sub-sequences, each sub-sequence allowing to satisfy the property.

Proposition 3 (Physicality, soundness, and transparency of enforcement functions) *Given some property φ, its enforcement function E_φ as per Definition 8 satisfies:*

1. *the physical constraints* $(\mathbf{Phy1})$ *and* $(\mathbf{Phy2})$*,*
2. *the soundness* $(\mathbf{Snd})$ *and transparency* $(\mathbf{Tr})$ *constraints,*

as per Definition 7.

In addition, the functional definition also ensures that each sub-sequence is output as soon as possible, as expressed by the following proposition:

Proposition 4 (Optimality of enforcement functions) *Given some property φ, its enforcement function E_φ as per Definition 8 satisfies the following optimality constraint* $(\mathbf{Op})$*:*

$$\begin{aligned}
&\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) \neq \epsilon \wedge E_\varphi(\sigma, t) \models \varphi \\
&\quad \Longrightarrow \exists w_{\mathrm{mx}}, w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* : \\
&\qquad w_{\mathrm{mx}} = \max^{\varphi}_{\prec,\epsilon}(E_\varphi(\sigma, t)) \\
&\qquad \wedge E_\varphi(\sigma, t) = w_{\mathrm{mx}} \cdot w \\
&\qquad \wedge \mathrm{time}(w) = \min\{\mathrm{time}(w') \mid \mathrm{delay}(w'(1)) \geq \mathrm{time}(\sigma_{[1 \cdots |E_\varphi(\sigma,t)|]}) - \mathrm{time}(w_{\mathrm{mx}}) \\
&\qquad\qquad \wedge w_{\mathrm{mx}} \cdot w' \models \varphi \wedge \Pi_\Sigma(w') = \Pi_\Sigma(w)\}.
\end{aligned}$$

Intuition of optimality For any input σ, at any time t, if the output $E_\varphi(\sigma, t)$ is not ϵ, and satisfies φ, then the output is considered as two sub-sequences w_{mx}, followed by w, such that w_{mx} is the maximal strict prefix of $E_\varphi(\sigma, t)$, satisfying property φ and w is the remaining sub-sequence such that $E_\varphi(\sigma, t)) = w_{\mathrm{mx}} \cdot w$.

The last sub-sequence of the output which again makes the output to satisfy φ after w_{mx} is w. The optimality constraint expresses that the sum of the delays (i.e., the time required to output) of w is minimal. The delay for the events in w should be chosen such that $E_\varphi(\sigma, t) = w_{\mathrm{mx}} \cdot w$ satisfies φ and the transparency condition, and the delay of the first event is greater than the difference between the duration of the input sequence $\sigma_{[1 \cdots |E_\varphi(\sigma,t)|]}$ and the duration of w_{mx}.

Notice that if $\mathrm{time}(\sigma_{[1 \cdots |E_\varphi(\sigma,t)|]}) - \mathrm{time}(w_{\mathrm{mx}})$ is negative or null, then this means that the delay corresponding to some events in the sequence preceding w (which is w_{mx}) are increased, providing sufficient amount of time to observe the last sub-sequence (which is $\sigma_{[|w_{\mathrm{mx}}|+1 \cdots |w_{\mathrm{mx}}|+|w|]}$) entirely. In case $\mathrm{time}(\sigma_{[1 \cdots |E_\varphi(\sigma,t)|]}) - \mathrm{time}(w_{\mathrm{mx}})$ is positive, all events in w_{mx} have been released as output before the last sub-sequence $\sigma_{[|w_{\mathrm{mx}}|+1 \cdots |w_{\mathrm{mx}}|+|w|]}$ is observed entirely as input. After releasing w_{mx}, $\mathrm{time}(\sigma_{[1 \cdots |E_\varphi(\sigma,t)|]}) - \mathrm{time}(w_{\mathrm{mx}})$ time units have elapsed and thus the last sub-sequence w can be released as output only after a delay of $\mathrm{time}(\sigma_{[1 \cdots |E_\varphi(\sigma,t)|]}) - \mathrm{time}(w_{\mathrm{mx}})$ time units.

Proof (of Propositions 3 and 4 - sketch only) The proofs are given in Appendices A.1 (for physical constraints), A.2 (for soundness and transparency), A.3 (for optimality), in p. 31, 32, 33, respectively. The proofs rely on an induction on the length of the input word σ. The induction step uses a case analysis, depending on whether the input is completely observed or not at time t, whether the input can be delayed into a correct output or not, and whether the memory content (computed by store) is completely dumped or not at time t.

The following example illustrates the notion of enforcement function.

Example 2 (Enforcement function) We now illustrate how Definition 8 is applied to enforce property φ_3 defined by the automaton depicted in Fig. 2c with $\Sigma = \{g, r\}$, and the input timed word $\sigma = (3, g) \cdot (10, r) \cdot (3, g) \cdot (5, g)$. Variable t describes global time. Figure 4 shows the evolution of obs, store, and E_φ over time for the input σ. The resulting output is $(13, g) \cdot (15, r)$, which satisfies property φ_1.

It is worth noticing that not all regular properties are enforceable, as illustrated by the following example.

Example 3 (Enforcement function: a non-enforceable property) Let us consider again property φ_4, formalized by the TA in Fig. 2d, with Σ= $\{g, r\}$, and the input timed word $\sigma = (3, r) \cdot (4, g) \cdot (2, r) \cdot (6, g)$. Figure 5 shows the evolution of obs, store, and E_φ. Variable

$t \in [0,3[$	$\mathrm{obs}(\sigma,t) = \epsilon$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = (\epsilon,\epsilon)$ $E_\varphi(\sigma,t) = \mathrm{obs}(\epsilon,t)$
$t \in [3,13[$	$\mathrm{obs}(\sigma,t) = (3,g)$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = (\epsilon,(3,g))$ $E_\varphi(\sigma,t) = \mathrm{obs}(\epsilon,t)$
$t \in [13,16[$	$\mathrm{obs}(\sigma,t) = (3,g)\cdot(10,r)$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = ((13,g)\cdot(15,r),\epsilon)$ $E_\varphi(\sigma,t) = \mathrm{obs}((13,g)\cdot(15,r),t)$
$t \in [16,21[$	$\mathrm{obs}(\sigma,t) = (3,g)\cdot(10,r)\cdot(3,g)$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = ((13,g)\cdot(15,r),(3,g))$ $E_\varphi(\sigma,t) = \mathrm{obs}((13,g)\cdot(15,r),t)$
$t \in [21,\infty[$	$\mathrm{obs}(\sigma,t) = (3,g)\cdot(10,r)\cdot(3,g)\cdot(5,g)$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = ((13,g)\cdot(15,r),(3,g)\cdot(5,g))$ $E_\varphi(\sigma,t) = \mathrm{obs}((13,g)\cdot(15,r),t)$

Fig. 4 Evolution of the enforcement function for φ_3.

$t \in [0,3[$	$\mathrm{obs}(\sigma,t) = \epsilon$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = (\epsilon,\epsilon)$ $E_\varphi(\sigma,t) = \mathrm{obs}(\epsilon,t)$
$t \in [3,7[$	$\mathrm{obs}(\sigma,t) = (3,r)$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = (\epsilon,(3,r))$ $E_\varphi(\sigma,t) = \mathrm{obs}(\epsilon,t)$
$t \in [7,9[$	$\mathrm{obs}(\sigma,t) = (3,r)\cdot(4,g)$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = (\epsilon,(3,r)\cdot(4,g))$ $E_\varphi(\sigma,t) = \mathrm{obs}(\epsilon,t)$
$t \in [9,15[$	$\mathrm{obs}(\sigma,t) = (3,r)\cdot(4,g)\cdot(2,r)$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = (\epsilon,(3,r)\cdot(4,g)\cdot(2,r))$ $E_\varphi(\sigma,t) = \mathrm{obs}(\epsilon,t)$
$t \in [15,\infty[$	$\mathrm{obs}(\sigma,t) = (3,r)\cdot(4,g)\cdot(2,r)\cdot(6,g)$ $\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t)) = (\epsilon,(3,r)\cdot(4,g)\cdot(2,r)\cdot(6,g))$ $E_\varphi(\sigma,t) = \mathrm{obs}(\epsilon,t)$

Fig. 5 Evolution of the enforcement function for φ_4.

t describes global time. The resulting output of the enforcement function is ϵ at any time instant.

Remark 3 (Non-enforceable properties) From Example 3, notice that the output of the enforcement function is ϵ, though the input sequence itself satisfies the property. The monitor observes action r followed by action g only at $t = 7$. Hence, the delay associated with the first action in output should be at least 7 t.u., but increasing the delay associated with the first action to 7, would falsify the guard on transition between l_0 to l_1, which is a possible move upon the first event req, and there is no transition with a reset of clock x before.

It can be noticed that guards with $<, \leq$, and $=$ impose urgency on releasing an event as output at or before some time. For some properties, some input sequences that can be delayed to satisfy φ cannot be corrected by enforcement, because the delay of the first event of each sub-sequence may be increased, which may falsify a guard with $<, \leq, =$. However, even for such properties, the enforcement function will never produce incorrect outputs. Moreover, note that the notion of non-enforceability exhibited here does not stem from the

fact that we focus on enforcement mechanisms that act as delayers. Indeed, even if our enforcement mechanisms were able to reduce delays between events (by for instance releasing g immediately after r), the property would remain non-enforceable because of the guard "$x \leq 5$" on the transition between locations l_0 and l_1.

Remark 4 (Alternative enforcement strategies) The optimality constraint presented in Proposition 4 allows only to augment delays of events. For each event, a delay greater than or equal to the actual delay is chosen. This condition can be modified or relaxed according to our requirements. This condition can be easily adapted to a given time bound in $\mathbb{R}_{\geq 0}$. It may also be possible to shorten the delay of some events (as long as the duration of the output is greater than the input, i.e. when it satisfies Phy2). Additionally, processing input and output actions is assumed to be done in zero time. Some delay (either fixed or depending on additional parameters) can be considered for this action by modifying the store function, and adding this constraint in the definition of κ_φ. Such modification would reduce the set of enforceable properties.

Remark 5 (Elimination of some acceptable behaviors by the enforcement function) As we saw earlier in Remark 3, some input sequences that can be delayed to satisfy φ cannot be accepted (and released as output) by the enforcement function. This behavior of the enforcement function stems from the following facts:

- At anytime, decision is taken based on the input events received until that time, and the enforcer has no information regarding the events that it will receive in the future.
- Moreover, we defined optimality in such a way that, the decision about releasing events is taken as soon as possible (when a mechanism knows that there is a possibility to correct, it does not wait for any further input events).

Note, this is also the case for control mechanisms in supervisory control theory for discrete event systems.

3.2.2 Simplified Functional Definitions

The functional definitions for enforcement in the case of safety or co-safety properties can be simplified. As an example, we briefly explain how the functional definition can be simplified for safety properties.

Simplified functional definition for safety properties. For safety properties, the store function can be simplified: the output is a timed word instead of a pair of timed words ($\mathrm{store}^{\mathrm{sa}}_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$). In fact for a safety property, the delay of each input event, if there exists one, can be computed immediately. Thus the second element of the store output pair (which stores events with undetermined delays) is unnecessary. The store function for safety properties can be defined as follows:

$$\mathrm{store}^{\mathrm{sa}}_\varphi(\epsilon) = \epsilon$$
$$\mathrm{store}^{\mathrm{sa}}_\varphi(\sigma \cdot (\delta, a)) = \begin{cases} \mathrm{store}^{\mathrm{sa}}_\varphi(\sigma) \cdot (\min(K), a) & \text{if } K \neq \emptyset, \\ \mathrm{store}^{\mathrm{sa}}_\varphi(\sigma) & \text{otherwise,} \end{cases}$$

where $K \stackrel{\mathrm{def}}{=} \{\delta' \in \mathbb{R}_{\geq 0} \mid \delta' \geq \delta \wedge \mathrm{store}^{\mathrm{sa}}_\varphi(\sigma) \cdot (\delta', a) \preccurlyeq_d \sigma \cdot (\delta, a) \wedge \mathrm{store}^{\mathrm{sa}}_\varphi(\sigma) \cdot (\delta', a) \models \varphi\}$.

K is the set of delays δ' that can be associated to a such that the extension $\mathrm{store}^{\mathrm{sa}}_\varphi(\sigma) \cdot (\delta', a)$ of the previous $\mathrm{store}^{\mathrm{sa}}_\varphi(\sigma)$ delays $\sigma \cdot (\delta, a)$ and still satisfies property φ. It depends on φ, σ (more precisely $\mathrm{store}^{\mathrm{sa}}_\varphi(\sigma)$), and (δ, a).

Remark 6 For the particular case of safety properties, Propositions 3 and 4 also hold when using the simplified functional definition (using the $\mathrm{store}^{\mathrm{sa}}_{\varphi}$ function).

Moreover, with the alternative functional definition for safety properties using function $\mathrm{store}^{\mathrm{sa}}_{\varphi}$, the optimality constraint in Proposition 4 which the enforcement function satisfies can be simplified and defined alternatively as follows.

Simplified optimality constraint of an enforcement function for safety properties. If E_φ is sound, transparent, and satisfies the physical constraints **(Phy1)** and **(Phy2)**, E_φ is said to be *optimal* if the following constraint holds:

$$\begin{aligned}&\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0}:\\ &\quad |E_\varphi(\sigma,t)| = \max\left\{|o'| \mid o' \preccurlyeq_d \mathrm{obs}(\sigma,t) \wedge o' \models \varphi \cap (\widehat{E_\varphi}(\sigma,t)\cdot(\mathbb{R}_{\geq 0}\times\Sigma)^*)\right\} \quad \textbf{(Op-saf)}\\ &\qquad \text{where } \widehat{E_\varphi}(\sigma,t) \stackrel{\mathrm{def}}{=} \max_{\preccurlyeq}\{E_\varphi(\sigma,t') \mid t' < t\}.\end{aligned}$$

The optimality constraint **(Op-saf)** expresses that at any time instant t, the output sequence $E_\varphi(\sigma,t)$ should be the longest correct timed word delaying the input sequence $\mathrm{obs}(\sigma,t)$ that extends $\widehat{E_\varphi}(\sigma,t)$, the maximal output sequence strictly before t. Notice that the term $(\mathbb{R}_{\geq 0}\times\Sigma)^*$ takes into account the fact that several timed actions may be output at time t.

3.3 Enforcement Monitor

In what follows, we consider a property φ defined by a TA $\mathcal{A}_\varphi$ whose semantics is a transition system $[\![\mathcal{A}_\varphi]\!] = (Q, q_0, \Gamma, \rightarrow, F_G)$.

An enforcement function E_φ for φ is implemented by an enforcement monitor (EM), defined as a transition system $\mathcal{E}$. An EM is an operational view of the enforcement function. It is equipped with a memory and a set of enforcement operations used to store and dump some timed events to and from the memory, respectively. The memory of an EM consists of two queues, each containing a timed word, one storing the received actions (with increased delays) which are corrected and can be released as output. The other queue stores the actions that are read by the EM, but are yet to be corrected (and cannot be released as output). In addition, an EM also keeps track of the state of the underlying TA, and clock values used to count time between input events and between output events.

Before presenting the definition of enforcement monitor, we introduce update as a function from $Q \times (\mathbb{R}_{\geq 0}\times\Sigma)^+ \times \mathbb{R}_{\geq 0}$ to $(\mathbb{R}_{\geq 0}\times\Sigma)^+ \times \mathbb{B}$. The update function takes as input a triple (q, σ_c, m_t) where $q \in Q$ is the (current) state of $[\![\mathcal{A}_\varphi]\!]$, $\sigma_c \in (\mathbb{R}_{\geq 0}\times\Sigma)^+$ is a non-empty timed word, and $m_t \in \mathbb{R}_{\geq 0}$ is the difference between the duration of the input sequence observed minus the duration of the corrected sequence, and returns a timed word of length $|\sigma_c|$ and a Boolean as output.

$$\mathrm{update}(q,\sigma_c,m_t) \stackrel{\mathrm{def}}{=} \begin{cases} (\sigma_c, \mathtt{ff}) & \text{if } \Lambda(\sigma_c,m_t,q) = \emptyset,\\ (\sigma'_c, \mathtt{tt}) & \text{otherwise;}\end{cases}$$

where $\sigma'_c = \min_{\preceq_{\mathrm{lex,time}}} \Lambda(\sigma_c,m_t,q)$ with $\Lambda : (\mathbb{R}_{\geq 0}\times\Sigma)^+ \times \mathbb{R}_{\geq 0} \times Q \rightarrow 2^{(\mathbb{R}_{\geq 0}\times\Sigma)^*}$ defined as:

$$\begin{aligned}\Lambda(\sigma_c,m_t,q) = \{w \in (\mathbb{R}_{\geq 0}\times\Sigma)^+ \mid w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \mathrm{delay}(w(1)) \geq m_t\\ \wedge q \stackrel{w}{\rightarrow} F_G\}.\end{aligned}$$

$\Lambda(\sigma_c, m_t, q)$ is the set of timed words w of length $|\sigma_c|$ with same actions as σ_c, each delay in the sequence is equal to or greater than the delay at the corresponding index in the provided input sequence σ_c, and the first delay in w should be greater than or equal to m_t, and an accepting state is reachable from state q upon sequence w.

- The first case applies when there are no good delays such that an accepting state is reachable from the state q upon with a sequence delaying σ_c ($\Lambda(\sigma_c, m_t, q) = \emptyset$). In this case, the update function returns the same timed word σ_c (which is provided as input), and a Boolean value `ff`, indicating that no accepting state is reachable.
- The second case applies when there are good delays and an accepting state in $q_1 \in Q_F$ is reachable from q upon a sequence delayed from σ_c. In this case, the update function returns a timed word of minimal duration belonging to $\Lambda(\sigma_c, m_t, q)$, chosen according to the lexical order; and a Boolean value `tt`, indicating that an accepting state is reachable.

Definition 9 (Enforcement Monitor) An enforcement monitor $\mathcal{E}$ for φ is a transition system $(C^{\mathcal{E}}, c_0^{\mathcal{E}}, \Gamma^{\mathcal{E}}, \hookrightarrow_{\mathcal{E}})$ s.t.:

- $C^{\mathcal{E}} = (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times Q$ is the set of configurations,
- $c_0^{\mathcal{E}} = \langle \varepsilon, \varepsilon, 0, 0, 0, q_0 \rangle \in C^{\mathcal{E}}$ is the initial configuration,
- $\Gamma^{\mathcal{E}} = \big((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}\big) \times Op \times \big((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}\big)$ is the alphabet, which is composed of triples, comprised of an optional input event, an operation, and an optional output event, where the set of possible operations is $Op = \{\text{store-}\overline{\varphi}(\cdot), \text{store-}\varphi(\cdot), \mathrm{dump}(\cdot), \mathrm{idle}(\cdot)\}$;
- $\hookrightarrow_{\mathcal{E}} \subseteq C^{\mathcal{E}} \times \Gamma_{\mathcal{E}} \times C^{\mathcal{E}}$ is the transition relation defined as the smallest relation obtained by the following rules applied with the priority order below:
  - **1. *store-*$\overline{\varphi}$**:
    $(\sigma_s, \sigma_c, \delta, d, m_t, q) \overset{(\delta,a)/\mathrm{store}-\overline{\varphi}(\delta,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_s, \sigma_c \cdot (\delta, a), 0, d, m_t', q)$,
    if $\Pi_2(\mathrm{update}(q, \sigma_c \cdot (\delta, a), m_t + \delta)) = \mathtt{ff}$, where:
    - $m_t' = m_t + \delta$
  - **2. *store-*φ**:
    $(\sigma_s, \sigma_c, \delta, d, m_t, q) \overset{(\delta,a)/\mathrm{store}-\varphi(\delta,a)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_s \cdot \sigma_c', \epsilon, 0, d, m_t', q')$,
    if $\mathrm{update}(q, \sigma_c \cdot (\delta, a), m_t + \delta)) = (\sigma_c', \mathtt{tt})$, where:
    - $m_t' = m_t' + \delta - \mathrm{time}(\sigma_c')$
    - q' is defined as $q \overset{\sigma_c'}{\to} q'$
  - **3. *dump***:
    $((\delta, a) \cdot \sigma_s, \sigma_c, s, \delta, m_t, q) \overset{\epsilon/\,\mathrm{dump}(\delta,a)/(\delta,a)}{\hookrightarrow_{\mathcal{E}}} (\sigma_s, \sigma_c, s, 0, m_t, q)$
  - **4. *idle***:
    $(\sigma_s, \sigma_c, s, d, m_t, q) \overset{\epsilon/\,\mathrm{idle}(\delta)/\epsilon}{\hookrightarrow_{\mathcal{E}}} (\sigma_s, \sigma_c, s + \delta, d + \delta, m_t, q)$.

A configuration $(\sigma_s, \sigma_c, s, d, m_t, q)$ of the EM consists of the current stored sequence (i.e., the memory content) σ_s, and σ_c. The sequence that is corrected and can be released as output is denoted by σ_s. The sequence σ_c is sort of an internal memory for the store function: this is the input sequence read by the EM, but yet to be corrected. The configuration also contains two clock values s and d indicating respectively the time elapsed since the last store and dump operations, and one more counter m_t indicating the difference between the duration of the observed input sequence and the duration of the corrected sequence. q is the current state of $[\![\mathcal{A}_\varphi]\!]$ reached after processing the sequence already released followed by the timed word in memory σ_s.

Semantic rules can be understood as follows:

- Upon reception of an event (δ, a), one of the following store rules is executed.
  - The ***store***-$\overline{\varphi}$ rule is executed if the update function returns `ff` (indicating that $\sigma_c \cdot (\delta, a)$ cannot be corrected). The clock s is reset to 0, and the event (δ, a) is appended to the internal memory σ_c. The delay corresponding to the input event δ is added to m_t.
  - The ***store***-φ rule is executed if the update function returns `tt`, indicating that φ can be satisfied for the sequence already released as output, followed by the sequence in σ_s, followed by $\sigma_c \cdot (\delta, a)$ with possibly increased delays. When executing this rule, s is reset to 0, and the timed word σ'_c returned by the update function is appended to the content of the output memory σ_s. The delay of the input event δ is added to m_t, and the duration of the corrected sub-sequence returned by the update function, $\mathrm{time}(\sigma'_c)$, is subtracted from m_t.
- The ***dump*** rule is executed if the time elapsed since the last dump operation d, is equal to the delay corresponding to the first event of the timed word σ_s in the memory. The event (δ, a) is released as output and removed from σ_s, and the clock d is reset to 0.
- The ***idle*** rule adds the time elapsed δ to the current values of s and d when neither the ***store*** nor the ***dump*** rule applies.

Example 4 (Execution of an enforcement monitor) We now illustrate how the rules of Definition 9 are applied to enforce property φ_3, defined by the automaton depicted in Fig. 2c. Let us consider the input timed word $\sigma = (3, g) \cdot (10, r) \cdot (3, g) \cdot (5, g)$. Figure 6 shows how semantic rules are applied, and the evolution of the configurations of the enforcement monitor. In a configuration, the input (resp. output) is on the right (resp. left). Variable t describes global time. The resulting output is $(13, g) \cdot (15, r)$, which satisfies property φ_3. From $t = 28$, only the ***idle*** rule can be applied.

Remark 7 (Simplified definitions of enforcement monitor) To synthesize an EM for a safety or co-safety property, one can use simplified definitions. For example, for a safety property, only one timed word is needed in the configuration. Indeed, recall that σ_c is a sort of internal memory used to store the input events used when it may be possible to reach an accepting state if more events are observed in the future. Since a safety property is prefix-closed, upon an event that cannot be delayed to keep satisfying the property, no future extension can. Hence, σ_c is not necessary for safety properties. Therefore, some simplifications, that may lead to performance improvements, are possible.

3.4 Relating Enforcement Functions and Enforcement Monitors

We present how the definitions of enforcement function and enforcement monitor can be related: given a property φ, any input sequence σ, at any time instant t, the output of the associated enforcement function and the output-behavior of the associated enforcement monitor are equal.

We first describe how an enforcement monitor reacts to an input sequence. In the remainder of this section, we consider an enforcement monitor $\mathcal{E} = (C^{\mathcal{E}}, c_0^{\mathcal{E}}, \Gamma^{\mathcal{E}}, \hookrightarrow_{\mathcal{E}})$. Enforcement monitors, as defined in Section 3.3, are deterministic. By determinism, we mean that the observable behavior of our enforcement monitors, i.e., given an input sequence the observable output sequence is unique. Moreover, given $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and $t \in \mathbb{R}_{\geq 0}$, how an enforcement monitor reads σ until time t is unique: it goes through a unique sequence of configurations. However, given an input sequence σ and a time instant t, because

$$
\begin{array}{cc}
t=0 & \epsilon/(\epsilon,\epsilon,0,0,0,(l_0,0))/(3,g)\cdot(10,r)\cdot(3,g)\cdot(5,g) \\
 & \downarrow \mathrm{idle}(3) \\
t=3 & \epsilon/(\epsilon,\epsilon,3,3,0,(l_0,0))/(3,g)\cdot(10,r)\cdot(3,g)\cdot(5,g) \\
 & \downarrow \text{store-}\overline{\varphi}(3,g) \\
t=3 & \epsilon/(\epsilon,(3,g),0,3,3,(l_0,0))/(10,r)\cdot(3,g)\cdot(5,g) \\
 & \downarrow \mathrm{idle}(10) \\
t=13 & \epsilon/(\epsilon,(3,g),0,13,3,(l_0,0))/(10,r)\cdot(3,g)\cdot(5,g) \\
 & \downarrow \text{store-}\varphi(10,r) \\
t=13 & \epsilon/((13,g)\cdot(15,r),\epsilon,0,13,-15,(l_0,15))/(3,g)\cdot(5,g) \\
 & \downarrow \mathrm{dump}(13,g) \\
t=13 & (13,g)/((15,r),\epsilon,0,0,-15,(l_0,15))/(3,g)\cdot(5,g) \\
 & \downarrow \mathrm{idle}(3) \\
t=16 & (13,g)/((15,r),\epsilon,3,3,-15,(l_0,15))/(3,g)\cdot(5,g) \\
 & \downarrow \text{store-}\overline{\varphi}(3,g) \\
t=16 & (13,g)/((15,r),(3,g),0,3,-12,(l_0,15))/(5,g) \\
 & \downarrow \mathrm{idle}(5) \\
t=21 & (13,g)/((15,r),(3,gr),5,8,-12,(l_0,15))/(5,g) \\
 & \downarrow \text{store-}\overline{\varphi}(5,g) \\
t=21 & (13,g)/((15,r),(3,g)\cdot(5,g),0,8,-7,(l_0,15))/\epsilon \\
 & \downarrow \mathrm{idle}(7) \\
t=28 & (13,gr)/((15,rel),(3,gr)\cdot(5,gr),7,15,-7,(l_0,15))/\epsilon \\
 & \downarrow \mathrm{dump}(15,r) \\
t=28 & (13,g)\cdot(15,r)/(\epsilon,(3,g)\cdot(5,g),7,0,-7,(l_0,15))/\epsilon
\end{array}
$$

Fig. 6 Execution of an enforcement monitor

of ϵ's in the input alphabet there is possibly an infinite set of corresponding sequences over the *input-operation-output* alphabet (as in Definition 9). All these sequences are equivalent: they involve the same configurations for the enforcement monitor and the same output sequence. Consequently, the rules of the transition relations are ordered in such a way that reading ϵ will always be the transition with least priority. As a consequence given an input sequence reading ϵ (and doing other operations such as outputting some event) will always be possible when the monitor cannot read an input. This constraint is consistent with our hypothesis stating that enforcement monitors execute infinitely faster than their environment.

More formally, let us define $\mathcal{E}_{\mathrm{ioo}}(\sigma,t) \in (\Gamma^{\mathcal{E}})^*$ to be the unique sequence over the alphabet of actions (triples comprised of an optional input event, an operation, and an optional output event) that is "triggered" when the enforcement monitor reads σ until time t.

Definition 10 (Input-Operation-Output sequence) Given an input sequence $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and some time instant $t \in \mathbb{R}_{\geq 0}$, we define the input-operation-output sequence denoted as $\mathcal{E}_{\mathrm{ioo}}(\sigma,t)$ and which triggered when an enforcement monitor of initial configuration $c_0^{\mathcal{E}}$ reads σ until time t. $\mathcal{E}_{\mathrm{ioo}}(\sigma,t)$ is defined as the unique sequence of $(\Gamma^{\mathcal{E}})^*$ such that:

$$
\begin{aligned}
\exists c \in C^{\mathcal{E}} : c_0^{\mathcal{E}} \overset{\mathcal{E}_{\mathrm{ioo}}(\sigma,t)}{\hookrightarrow}{}^*_{\mathcal{E}}\ c \\
\wedge\ \Pi_1(\mathcal{E}_{\mathrm{ioo}}(\sigma,t)) = \mathrm{obs}(\sigma,t) \\
\wedge\ \mathrm{timeop}(\Pi_2(\mathcal{E}_{\mathrm{ioo}}(\sigma,t))) = t \\
\wedge\ \nexists c' \in C^{\mathcal{E}}, \exists e \in (\mathbb{R}_{\geq 0} \times \Sigma) : c \overset{(\epsilon,\mathrm{dump}(e),e)}{\hookrightarrow} c',
\end{aligned}
$$

where the timeop function indicates the duration of a sequence of enforcement operations and says that only the idle enforcement operation consumes time. Formally:

$$\mathrm{timeop}(op \cdot ops) = \begin{cases} d + \mathrm{timeop}(ops) & \text{if } \exists d \in \mathbb{R}_{\geq 0} : op = \mathrm{idle}(d), \\ \mathrm{timeop}(ops) & \text{otherwise;} \end{cases}$$
$$\mathrm{timeop}(\epsilon) = 0.$$

The input timed word corresponding to $\mathrm{obs}(\sigma, t)$ at any time t is the concatenation of all the input events read/consumed by the enforcement monitor over various steps. Observe that because of the assumptions on $\Gamma^{\mathcal{E}}$, only the ***idle*** rule applies to the configuration c: the ***dump*** rule does not apply by definition of $\mathcal{E}_{\mathrm{ioo}}(\sigma, t)$ and none of the ***store*** rules applies because $\Pi_1(\mathcal{E}_{\mathrm{ioo}}(\sigma, t)) = \mathrm{obs}(\sigma, t)$.

Relating enforcement functions and enforcement monitors. Now, we can relate the enforcement function and the enforcement monitor, for a property φ. Seen from the outside, an enforcement monitor $\mathcal{E}$ behaves as a device reading and producing timed words. Overloading notations, we can characterize this input/output behavior as a function $\mathcal{E} : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ defined as:

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : \mathcal{E}(\sigma, t) = \Pi_3\big(\mathcal{E}_{\mathrm{ioo}}(\sigma, t)\big).$$

The corresponding output timed word $\mathcal{E}(\sigma, t)$ at any time t is the concatenation of all the output events produced by the enforcement monitor over various steps of the enforcement monitor (erasing ϵ's). As before, the ϵ's output by the store operation are erased. In the following, we do not make the distinction between an enforcement monitor and the function that characterizes its behavior.

Finally, we are able to relate enforcement functions, and the functions derived from an enforcement monitor. For this purpose, we define an implementation relation between enforcement monitors and enforcement functions as follows.

Definition 11 (Implementation relation between enforcement functions and enforcement monitors) Given an enforcement function E_φ (as per Definition 6) and an enforcement monitor (as per Definition 9) whose behavior is characterized by a function $\mathcal{E}$, we say that $\mathcal{E}$ *implements* E_φ iff:

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) = \mathcal{E}(\sigma, t).$$

Proposition 5 (Relation between enforcement function and enforcement monitor) *Given a property φ, its enforcement function E_φ (as per Definition 8), and its enforcement monitor $\mathcal{E}$ (as per Definition 9), $\mathcal{E}$ implements E_φ in the sense of Definition 11.*

Proof (of Proposition 5 - sketch only) The proof is given in Appendix A, p. 37. The proof is done by induction on the length of the input sequence and uses a similar case analysis as the proof of Proposition A.2. The proof also uses several intermediate lemmas that characterize some special configurations (e.g., value of the store and dump variables, memory content) of an enforcement monitor at some time instants.

3.5 Implementation of Enforcement Monitors

Let us now see the algorithms, which are a straightforward translation of the EM semantics, showing how enforcement monitors can be implemented. The implementation of an EM consists of two processes running concurrently (Store and Dump) as shown in Figure 7, and a memory. The Store process models the ***store*** rules. The memory contains the timed word σ_s: the corrected sequence that can be released as output. The memory σ_s is realized as a queue, shared by the Store and Dump processes, where the Store process adds events which are processed and corrected to this queue. The Dump process reads events stored in the memory σ_s and releases them as output after the required amount of time. The Store process also makes use of another internal buffer σ_c (not shared with any other process), to store the events which are read, but cannot be corrected (to satisfy the property). In the algorithms the await primitive is used to wait for a trigger event from another process or to wait until some condition becomes true. The wait primitive is used by a process to wait for a certain amount of time, that is determined by the process itself.

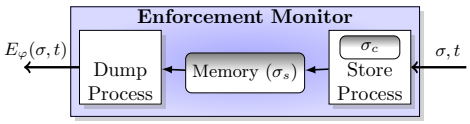


Fig. 7 Realizing an EM

The StoreProcess algorithm (see Algorithm 1) is an infinite loop that scrutinizes the system for input events. In the algorithm, (l, ν) represents the state of the automaton defining the property, where l represents the location and ν is the current clock valuation. It is initialized to $(l_0, [X \leftarrow 0])$. The variable m_t is used to keep track of the difference between the duration of the input sequence read (the sequence which is already corrected followed by the sequence in σ_c), and the duration of the corrected sequence. The update function takes the events stored in the internal memory of the store process σ_c, the current state, and m_t, and returns a timed word of same length as σ_c and a Boolean indicating whether an accepting state is reachable from the current state upon the timed word it returns as a result. The function post takes a state of the automaton defining the property (l, ν), a timed word, and computes the state reached by this automaton.

The algorithm proceeds as follows. The StoreProcess initially waits for an input event. A received event is appended to the internal buffer σ_c, with the corresponding delay δ, and this delay δ is added to m_t. Then the update function is invoked providing the events stored in σ_c as input. If the update function indicates that there is a path leading to an accepting state, (i.e., if $isPath = \mathtt{tt}$), then the timed word σ'_c returned by the update function, is appended to the shared memory σ_s (since it now corrected with respect to the property, and can be released as output). Then, the duration of σ'_c is subtracted from m_t. Before proceeding to the next iteration, the state of the automaton (l, ν) is updated, and the internal memory σ_c is cleared.

The DumpProcess algorithm (see Algorithm 2) is an infinite loop that scrutinizes the memory and proceeds as follows: Initially, the clock d is set to 0. If the memory is empty ($\sigma_s = \epsilon$), the DumpProcess waits until a new element (δ, a) is stored in the memory. Else (the memory is not empty), it proceeds with the first element in the memory. Using the dequeue operation, the first element stored in the memory is removed, and is stored as

Algorithm 1 StoreProcess

```
(l, ν) ← (l_0, [X ← 0])
(σ_s, σ_c) ← (ϵ, ϵ)
m_t ← 0
while tt do
    (δ, a) ← await(event)
    σ_c ← σ_c · (δ, a)
    m_t ← m_t + δ
    (σ'_c, isPath) ← update(l, ν, σ_c, m_t)
    if isPath = tt then
        m_t ← m_t − time(σ'_c)
        σ_s ← σ_s · σ'_c
        (l, ν) ← post(l, ν, σ'_c)
        σ_c ← ϵ
    end if
end while
```

Algorithm 2 DumpProcess

```
d ← 0
while tt do
    await(σ_s ≠ ϵ)
    (δ, a) ← dequeue (σ_s)
    wait(δ − d)
    dump (a)
    d ← 0
end while
```

(δ, a). Meanwhile, d keeps track of the time elapsed since the last dump operation. The DumpProcess waits for $(\delta - d)$ time units before performing the dump(a) operation, releasing the action a as output (which amounts to appending (δ, a) to the output of the enforcement monitor). Finally, the clock d is reset to 0 before the next iteration starts.

Remark 8 (Using non-deterministic TAs to define properties) In this paper, the presentation considers only deterministic TAs. Extending the results to non-deterministic TAs comes directly from the policy used to choose a unique solution to define the update function (which computes the correct and optimal delays). The update function first computes all accepting paths from the current state, for the given input sub-sequence. And from this set of all accepting paths, a unique solution with minimal duration is chosen using the lexical order. Thus, note that the same mechanisms and algorithms remain valid (without requiring any change) to also enforce properties defined with non-deterministic TAs.

Remark 9 (Simplified algorithms) For safety and co-safety properties, if we want to implement monitors following the simplified functional and enforcement monitor definitions, the algorithm for the StoreProcess can also be simplified. In particular, in the algorithm for safety properties, the content of the memory can be maintained by a single sequence of events instead of a tuple. Also, in case of safety properties, the update function is always invoked with a single event as input, instead of a sequence of events. The simplified algorithms for safety and co-safety properties are described in detail in [16].

Remark 10 (Enforcing several properties) Earlier in this section, we described how any regular property can be enforced. When a boolean combination of properties has to be enforced on a system, we can combine the properties in a single one and synthesize one enforcement monitor for the resulting property. Definition 4 in Section 2 describes how two properties

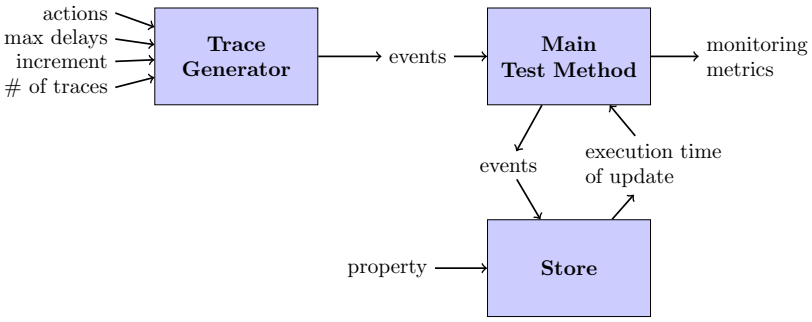


Fig. 8 Experimental framework

defined by TAs can be combined using Boolean operations (e.g., union, intersection and negation).

4 Implementation and Evaluation

We implemented the algorithms in Sec. 3.5 and developed an experimentation framework called TIPEX: (TImed Properties Enforcement during eXecution) in order to:

1. validate through experiments the architecture and feasibility of enforcement monitoring, and
2. measure and analyze the performance of the update function of the StoreProcess in the case of safety and co-safety properties.

From [16], we completely re-implemented the enforcement monitors. Still following the algorithms proposed in [16], TIPEX is more independent and offers better performance. It is now completely independent from UPPAAL at runtime. Moreover, TIPEX does not invoke UPPAAL to realize update anymore, and some other redundancies, such as updating the UPPAAL model file after each event, are also eliminated.

In this paper, we focus on evaluating the performance using some safety and co-safety properties. In Sec. 3, we described how the definitions of enforcement mechanisms can be simplified if we know that the property is safety (or co-safety). Thus, instead of using the algorithms proposed for regular properties in Sec. 3.5, TIPEX is implemented using the algorithms based on the simplified definitions for safety and co-safety properties described in [16]. Extending TIPEX for regular properties based on the algorithms proposed in this paper, and evaluation using some regular properties which are neither safety nor co-safety, is ongoing.

We focus on benchmarking the update function of the StoreProcess for safety and co-safety properties proposed in [16], which are the simplified versions of the general StoreProcess described in section 3.5. Indeed, examining the algorithms, the steps in the algorithm of the DumpProcess of monitors are algorithmically simple and lightweight from a computational point of view. Regarding the StoreProcess function for safety and co-safety properties, their most computationally intensive step is their call to their update function.

Experimental framework. Let us briefly look into the experimental framework, which is depicted in Fig. 8. The Main module uses the module Trace Generator that provides a set of input traces, to test the Store module. The Trace Generator module takes as input the

alphabet of actions, the range of possible delays between actions, the desired number of traces, and the increment in length per trace. For example if the number of traces in 5, and increment in length per trace is 100 then 5 traces will be generated, where the first trace is of length 100 and the second trace of length 200 and so on. It returns a set of traces. For each event, the Trace Generator picks an action (from the set of possible actions), and a delay (from the set of possible delays) randomly using methods from the Python random module.

The Store module takes as input a property and one trace, and returns the total execution of the update function to process the given input trace. The TA modeling the property is a UPPAAL [20] model written in XML. The Store module uses the pyuppaal library to parse the UPPAAL model (input property), and the UPPAAL DBM library to implement the update function. [4] More details about the implementation of the Store module for safety and co-safety properties are in Appendix B. The UPPAAL model also contains another automaton representing the sequence of events received by the enforcement monitor. The Main Test Method sends the sequence to the Store module (using the property), and keeps track of the result returned by the Store module for each trace.

Experiments were conducted on an Intel Core i7-2720QM at 2.20GHz CPU, with 4 GB RAM, and running on Ubuntu 12.04 LTS. The reported numbers are mean values over 10 runs and are represented in seconds. We have chosen to compute the average values over 10 runs because, for all metrics, with 95% confidence, the measurement error was less than 1%. For example, referring to Table 1, for the safety property $\varphi_{\mathrm{s}}^{1.1}$, the mean value of the total execution time of the update function is 8.6306 seconds, and the error is 0.018 seconds. Thus with 95% confidence, the execution time of update for input trace of length 10000 lies within the interval $[8.6126, 8.6486]$ (seconds). For the average time per call, as shown in the table, $\varphi_{\mathrm{s}}^{1.1}$, the mean value is 0.863 milli-seconds, and the error is 0.005 milliseconds. Thus, with 95% confidence, the average time per call to update, for input trace of length 10000 lies within the interval $[0.858, 0.868]$ (milliseconds).

4.1 Performance evaluation of the update function for safety properties

We describe the properties used in our experiments and discuss the results of the performance analysis. The considered safety properties follow different patterns [21].

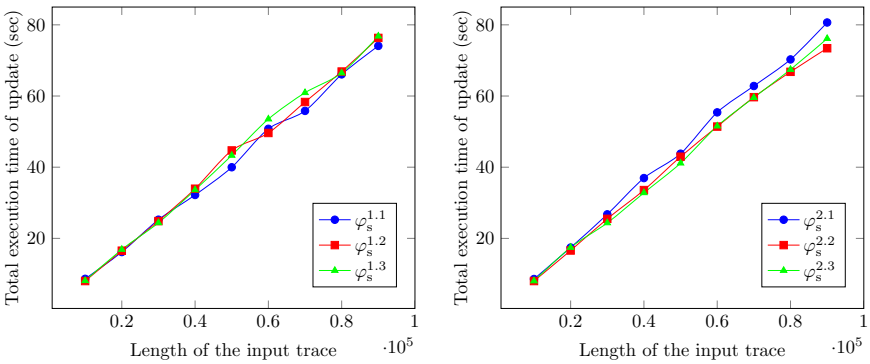
Property φ_{s}^{1} belongs to the absence pattern. It expresses that *"There cannot be n or more a-actions in every k time units"*, where n is a parameter of the pattern. Following this pattern, the considered properties are $\varphi_{\mathrm{s}}^{1.1}$, $\varphi_{\mathrm{s}}^{1.2}$ and $\varphi_{\mathrm{s}}^{1.3}$, each varying in the value of n: $n = 2$ for $\varphi_{\mathrm{s}}^{1.1}$, $n = 10$ for $\varphi_{\mathrm{s}}^{1.2}$, $n = 20$ for $\varphi_{\mathrm{s}}^{1.3}$. Property φ_{s}^{2} belongs to the precedence pattern. It expresses that *"A sequence of n a-actions enables action b after a delay of k time units"*. Following this pattern, the considered properties are $\varphi_{\mathrm{s}}^{2.1}$, $\varphi_{\mathrm{s}}^{2.2}$ and $\varphi_{\mathrm{s}}^{2.3}$, each varying in the value of n: $n = 1$ for $\varphi_{\mathrm{s}}^{2.1}$, $n = 5$ for $\varphi_{\mathrm{s}}^{2.2}$, and $n = 10$ for $\varphi_{\mathrm{s}}^{2.3}$.

Results and analysis. Results of the performance analysis of our running example properties are reported in Table 1. The entry $\mathrm{t_update}$ indicates the total execution time of the update function, and the entry $\mathrm{t_avg}$ is the average time per call. From the results presented in Table 1, as expected for safety properties, we can observe that the time taken per call to update is independent on the length of the trace. This behavior is as expected: since we

[4] The pyuppaal and DBM libraries are provided by Aalborg University. They can be downloaded at `http://people.cs.aau.dk/~adavid/python/`.

Table 1 Performance analysis of enforcement monitors for safety properties

	$\varphi_s^{1.1}$		$\varphi_s^{1.2}$		$\varphi_s^{1.3}$	
$\|tr\|$	t_update	t_avg	t_update	t_avg	t_update	t_avg
10,000	8.6306	0.000863	8.008	0.00080	8.106	0.000810
20,000	16.157	0.000807	16.538	0.000828	16.887	0.000844
30,000	25.251	0.000841	24.855	0.000828	24.3794	0.00812
40,000	32.199	0.000804	33.947	0.000848	33.619	0.000840
50,000	39.982	0.000799	44.704	0.000854	43.314	0.000866
60,000	50.785	0.000846	49.616	0.000826	53.521	0.000892
70,000	55.821	0.000797	58.317	0.000833	60.928	0.000870
80,000	66.080	0.000826	66.876	0.000835	66.461	0.000830
90,000	74.082	0.000823	76.327	0.000848	76.807	0.000853
	$\varphi_s^{2.1}$		$\varphi_s^{2.2}$		$\varphi_s^{2.3}$	
$\|tr\|$	t_update	t_avg	t_update	t_avg	t_update	t_avg
10,000	8.589	0.000858	8.019	0.000801	8.050	0.000805
20,000	17.435	0.000871	16.603	0.000830	17.472	0.000873
30,000	26.760	0.000892	25.507	0.000850	24.353	0.000811
40,000	36.956	0.000923	33.576	0.000839	32.811	0.000820
50,000	43.806	0.000876	42.955	0.000859	41.141	0.000822
60,000	55.410	0.000923	51.417	0.000856	51.550	0.000859
70,000	62.816	0.000897	59.677	0.000852	59.572	0.000851
80,000	70.282	0.000878	66.800	0.000835	67.450	0.000843
90,000	80.659	0.000896	73.423	0.000815	76.137	0.000845



(a) For φ_s^1. (b) For φ_s^2.

Fig. 9 Length of the input trace (Vs) total execution time of update.

update the state of the TA after each event, and after receiving a new event, we explore the possible transitions leading to a good state from the current state. Moreover, from the curves shown in Fig. 9, notice that, for a given trace length, the execution time of update is similar for the two patterns and their variants in size.

Table 2 Performance analysis of enforcement monitors for co-safety properties

	$\varphi_{\text{cs}}^{1.1}$			$\varphi_{\text{cs}}^{1.2}$		
$\lvert tr \rvert$	t_update	t_avg	t_last	t_update	t_avg	t_last
100	3.332	0.0333	0.0776	3.4099	0.034	0.080
200	12.838	0.0641	0.148	13.27	0.0663	0.155
300	29.926	0.0997	0.231	30.687	0.102	0.229
400	51.925	0.129	0.303	53.280	0.133	0.310
500	79.66	0.159	0.372	81.91	0.163	0.382
600	115.05	0.191	0.448	117.24	0.19	0.45
700	159.64	0.228	0.530	161.97	0.231	0.53
800	206.95	0.258	0.608	212.64	0.265	0.62
900	262.2	0.291	0.683	267.61	0.297	0.695

4.2 Performance evaluation of the update function for co-safety properties

We describe the properties used in our experiments and discuss the results of the performance analysis. The considered co-safety properties follow different patterns [21].

Property φ_{cs}^{1} belongs to the existence pattern [21]. It expresses that *"There should be n r-actions, which should be immediately followed by a g-action with a delay of at least k time units"*. Following this pattern, the considered properties are $\varphi_{\text{cs}}^{1.1}$ and $\varphi_{\text{cs}}^{1.2}$ each varying in the value of n: $n = 1$ for $\varphi_{\text{cs}}^{1.1}$, and $n = 5$ for $\varphi_{\text{cs}}^{1.2}$.

Results and analysis. Results of the performance analysis of our running example properties are presented in Table 2. Entry $\mathrm{t_update}$ indicates the execution time of function update. Entry $\mathrm{t_avg}$ is the average time per call, and the entry $\mathrm{t_last}$ is the execution time of update upon the last event. Note that the execution on the last event is the most time consuming. The considered input traces are generated in such a way that an accepting state is reachable only upon the last event. From the results presented in Table 2, notice that $\mathrm{t_last}$ and $\mathrm{t_avg}$ increase with $|tr|$. This behavior is as expected for a co-safety property because the update function starts the computation from the initial state for a given input trace. This behavior is also clearly shown by the curves in Fig. 10, showing the total time taken by the update function versus the length of the input trace. Moreover, notice that, for a given trace length, the execution time of update is similar for the two properties of the same pattern varying in size.

Remark 11 In case of a co-safety property, the monitor starts to output events only after reading an input sequence which can satisfy the property (after the optimal delays are computed for the input sequence). Once an accepting state is reached, then it is not necessary to invoke the update and to correct the delays anymore.

4.3 Discussion

On precision. In theory, the delays between actions and the optimal delay computed by the update function are real numbers. In the implementation, in order to compute optimal delay, we need to set precision.

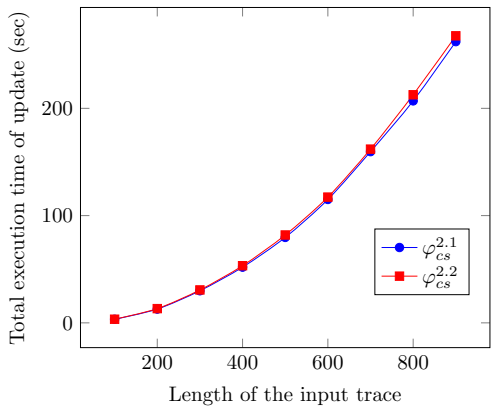


Fig. 10 For φ_{cs}^{1}: Length of the input trace (Vs) total execution time of update.

We use UPPAAL [20] to model the input TA, and some UPPAAL libraries to realize the algorithms. In UPPAAL, only integers can be used to compare the values of clocks in the guards. But, in practice, we may have to use real-numbers to express requirements and timing constraints. This issue can be handled by setting the precision of real-numbers, and representing values on guards with equivalent integers. For example, if we set the precision with four digits after the decimal point, 0.0024 can be represented as 24, and 5.0012 can be represented as 50012. Note that having a large integer value on a guard such as in $x > 50000$ is not an issue with region and zone computations, as computation is done on-the-fly. After each event, we check for possible paths from the current state.

On performance and overhead. Assessing the performance of runtime enforcement monitors is paramount in a timed context as. Using the experimental results, one can determine the guards and the properties for which the assumptions stated in the introduction hold. Regarding safety properties, one can see that, on the used experimental setup, the computation time of the update function is below 1ms. By taking guards with constraints using integers above 0.1s, one can see that the computation time can be negligible in some sense as the impact on the guard is below 1%, and makes the overhead of enforcement monitoring acceptable.

5 Related Work

Several runtime verification and enforcement approaches are related to the one proposed in this paper. We propose a comparison with approaches for the runtime enforcement of untimed properties (Sec. 5.1), for the runtime verification of timed properties (Sec. 5.2), and runtime enforcement of timed properties (Sec. 5.3)

5.1 Runtime Enforcement of Untimed Properties

Most of the work in runtime enforcement was dedicated to untimed properties (see [9] for a short overview). Schneider introduced security automata as the first runtime mechanism for enforcing safety properties [7]. Then the set of enforceable properties was later refined

by Schneider, Hamlen, and Morrisett by showing that security automata were actually restrained by the computational limits exhibited by Viswanathan and Kim [22]: the set of co-recursively enumerable safety properties is a strict upper limit of the power of (execution) enforcement monitors defined as security automata. Ligatti et al. [8] later introduced edit-automata as enforcement monitors. Edit-automata can either insert a new action by replacing the current input, or suppress it. The set of properties enforced by edit-automata is called the set of infinite renewal properties: it is a super-set of safety properties and contains some liveness properties (but not all). Similar to edit-automata are generic enforcement monitors [10] are able to enforce the set of (untimed) response regular properties in the safety-progress classification. Moreover, some variants of edit-automata differ in how they ensure the transparency constraints (see e.g., [23]).

5.2 Runtime Verification of Timed Properties

Several approaches have been proposed for the runtime verification of timed properties. We shall categorize them into i) rather theoretical efforts aiming at synthesizing monitors (Sec. 5.2.1), and ii) tools for runtime monitoring of timed properties (Sec. 5.2.2).

5.2.1 Synthesis of Timed Automata from Timed Logical Formalisms

Bauer et al. propose an approach to runtime verify timed-bounded properties expressed in a variant of Timed Linear Temporal Logic (TLTL) [4]. Contrarily to TLTL, the considered logic, TLTL_3, processes finite timed words and the truth-values of this logic are suitable for monitoring. After reading some timed word u, the monitor synthesized for a TLTL_3 formula φ states the verdict $\top$ (resp. $\bot$) when there is no infinite timed continuation w such that $u \cdot w$ satisfy (resp. does not satisfy) φ. Another variant of LTL in a timed context is Metric Temporal Logic (MTL), a dense extension of LTL. Nickovic et al. [19,3] propose a translation of MTL to timed automata. The translation is defined under the bounded variability assumption stating that, in a finite interval, a bounded number of events can arrive to the monitor. Still for MTL, Thati et al. propose an online monitoring algorithm which works by rewriting of the monitored formula and study its complexity [1]. Basin et al. propose an improvement of the aforementioned approach with a better complexity but considering only the past fragment of MTL [5].

Runtime enforcement of timed properties as presented in this paper is compatible with the previously-described approaches. These approaches synthesize audtomata-based decision procedures for logical formalisms. Decision procedures synthesized for regular properties can be used as input to our framework.

5.2.2 Tools for Runtime Monitoring of Timed Properties

The Analog Monitoring Tool [11] is a tool for monitoring specifications over continuous signals. The input logic of AMT is STL/PSL where continuous signals are abstracted into propositions and operations are defined over signals. Input signal traces can be monitored in an offline or incremental fashion (i.e., online monitoring with periodic trace accumulation).

RT-MaC [24] is a tool for verifying timed properties at runtime. RT-MaC allows to verify timeliness and reliability correctness. Using the time-bound temporal operators provided by the tool, one can specify a deadline after which a property must hold.

LARVA [25, 12] takes as input properties expressed in several notations, e.g., Lustre, duration calculus. Properties are translated to DATE (Dynamic Automata with Timers and Events) which basically resemble timed automata with stop watches but also feature resets, pauses, and can be composed into networks. Transitions are augmented with code that modify the internal system state. DATE target only safety properties. In addition, LARVA is able to compute an upper-bound on the overhead induced on the target system. The authors also identify a subset of the duration calculus, called counter-examples traces, where properties are insensitive to monitoring [26].

Our monitors not only differ by their objectives but also by how they are interfaced with the system. We propose a less restrictive framework where monitors asynchronously read the outputs of the target system. We do not assume our monitors to be able to modify the internal state of the target program. The objective of our monitors is rather to correct the timed sequence of output events before this sequence is released to the environment (i.e., outside the system augmented with a monitor).

5.3 Runtime Enforcement of Timed Properties

Matteucci inspires from partial-model checking techniques to synthesize controller operations to enforce safety and information-flow properties using process-algebra [13]. Monitors are close to Schneider's security automata [7]. The approach targets discrete-time properties and systems are modeled as timed processes expressed in CCS. Compared to our approach, the description of enforcement mechanisms remains abstract, directly restricts the monitored system, and no description of monitor implementation is proposed. Besides, in a general study, Rinard discusses monitoring and enforcement strategies for real-time systems [27], and mentions the fact that enforcement mechanisms could delay input individual events in an input stream when they arrive too early w.r.t. the constraints of the system. In the same way, we consider in our work that an enforcer is time retardant. However, the work in [27] remains at a high-level of abstraction and does not propose any detailed description of enforcement mechanisms.

More recently, Basin et al. [14] proposed a general approach related to enforcement of security policies with controllable and uncontrollable events, investigating enforceability (with complexity results), and how to synthesize enforcement mechanisms for several specification formalisms (automata-based or logic-based). A monitor observes the system and terminates it to prevent violations. Timed properties described in MLTL logic are handled in this work. Discrete time is considered, clock ticks are used to determine the enforceability of an MLTL formula. In our approach, we consider dense time, using the expressiveness of timed automata and efficiency of UPPAAL. Moreover, our enforcement mechanisms may modify the execution of the observed system, and termination is decided if correcting the execution by delaying is not possible.

6 Conclusion and Future Work

Conclusion. This paper presents a general enforcement monitoring framework for systems with (dense) timing requirements. We showed how to synthesize enforcement mechanisms for any regular timed property (modeled with a timed automaton). We propose adapted notions of enforcement mechanisms that delay input actions in order to satisfy the required

property. Enforcement mechanisms are described at several levels of abstraction (enforcement function, monitor, and algorithm), thus facilitating the design and implementation of such mechanisms. We describe how to realize the enforcement monitor using concurrent processes. We propose a prototype implementation and our experiments demonstrate the feasibility of enforcement monitoring for timed properties.

Future work. Several avenues for future work are opened by this paper.

First, we believe it is important to study and delineate the set of *enforceable timed properties*. As shown informally by this paper, some timed properties should be characterized as non-enforceable. For this purpose, an enforceability condition should be defined and used to delineate enforceable properties. Such a criterion should ideally also be expressible on timed automata. We conjecture that several enforceability notions exist. Some notions depend on the underlying mechanism used to enforce properties. We believe that there is also one (formalism-independent) enforceability notion that stems only from the physical time constraints faced by enforcement mechanisms.

Note that even for properties which are non-enforceable, the enforcement monitors proposed in this paper can be built, which may not be able to correct some input sequences, but their outputs are always sound.

Properties are currently defined with timed automata. We consider synthesizing enforcement mechanisms from more expressive formalisms. For instance, we will consider formalisms such as context-free timed languages (which can be useful for recursive specifications) or introduce data into requirements (which can be useful in some application domains). Implementing efficient enforcement monitors is another important aspect and should be done w.r.t. a particular application domain.

We implemented the tool in Python with the objectives of i) making a quick prototype that shows feasibility of enforcement monitoring in a timed context, and ii) reusing some existing UPPAAL libraries. In the future, we will consider implementing our enforcement monitors in other languages such as C or Java, and we expect even better performance and a more stand-alone implementation.

Alternative enforcement primitives can be afforded to timed retardants, which could be of interest in some application domains. For instance, we could relax the constraint of only augmenting delays of events. For instance, time retardants that delay the total duration of the observation (while being allowed to shorten the delay of some events) have yet to be studied. Suppressing events also can be considered, by erasing some events which are stored in the memory. An event should be suppressed if it is not possible to satisfy the property in the future, whatever is the remainder of the input sequence (i.e. the TA has reached q non-accepting state from which no accepting states can be reached). Formalizing suppression is however quite involved, requiring to redefine relations between input and output sequences which impacts on transparency.

Also related to expressiveness is the question of how the set of timed enforceable properties is impacted when the underlying memory is limited and/or the primitive operations endowed to the monitor are modified.

A Proofs

A.1 Proof of Proposition 3 - item 1 (physical constraints)

We shall prove that, given a property φ, the associated enforcement function $E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ defined as per Definition 8 satisfies the physical constraints **(Phy1)** and **(Phy2)**. That is, we

shall prove that:

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t, t' \in \mathbb{R}_{\geq 0} : t \leq t' \implies E_\varphi(\sigma, t) \preccurlyeq E_\varphi(\sigma, t'), \quad \textbf{(Phy1)}$$
$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in (\mathbb{R}_{\geq 0} \times \Sigma)^* : \mathrm{time}(E_\varphi(\sigma, t)) \leq t. \quad \textbf{(Phy2)}$$

- Proof that the enforcement function satisfies **(Phy1)**.
  Let us consider $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, t, t' \in \mathbb{R}_{\geq 0}$, such that $t \leq t'$. We will prove that there exists $o \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ s.t $E_\varphi(\sigma, t') = E_\varphi(\sigma, t) \cdot o$. The fact that E_φ satisfies **(Phy1)** is a direct consequence of the following observations:
  - $\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t, t' \in \mathbb{R}_{\geq 0} : t \leq t' \implies \mathrm{obs}(\sigma, t) \preccurlyeq \mathrm{obs}(\sigma, t')$, i.e., obs is monotonic in its second argument;
  - $\forall w, w' \in (\mathbb{R}_{\geq 0} \times \Sigma)^* : w \preccurlyeq w' \implies \mathrm{store}(w) \preccurlyeq \mathrm{store}(w')$, i.e., store is monotonic;
  - $\forall \sigma, \sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : \sigma \preccurlyeq \sigma' \implies \mathrm{obs}(\sigma, t) \preccurlyeq \mathrm{obs}(\sigma', t)$, i.e., obs is monotonic in its first argument.
- Proof that the enforcement function satisfies **(Phy2)**.
  From Definition 8, since $E_\varphi(\sigma, t) = \mathrm{obs}\Big(\Pi_1\big(\mathrm{store}(\mathrm{obs}(\sigma, t))\big), t\Big)$, from the definition of obs and the property of obs that $\forall \sigma : \mathrm{time}(\mathrm{obs}(\sigma, t)) \leq t$, applied to $\Pi_1\left(\mathrm{store}\left(\mathrm{obs}\left(\sigma, t\right)\right)\right)$, we can conclude that $\mathrm{time}(E_\varphi(\sigma, t)) \leq t$. Thus, E_φ satisfies **(Phy2)**.

A.2 Proof of Proposition 3 - item 2 (soundness and transparency)

We shall prove that, given a property φ, the associated enforcement function $E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ as per Definition 8 is sound and transparent as per Definitions 7, i.e., E_φ satisfies the constraints **(Snd)** and **(Tr)**.

Recall that $E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ is defined as:

$$E_\varphi(\sigma, t) = \mathrm{obs}\Big(\Pi_1\big(\mathrm{store}(\mathrm{obs}(\sigma, t))\big), t\Big).$$

where:

- function $\mathrm{obs} : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ is the observation function defined in Sec. 2,
- function $\mathrm{store} : (\mathbb{R}_{\geq 0} \times \Sigma)^* \to (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^*$ is defined as follows in Sec. 3.2.1:

$$\begin{aligned}
\mathrm{store}(\epsilon) &= (\epsilon, \epsilon) \\
\mathrm{store}(\sigma \cdot (\delta, a)) &= \begin{cases} (\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K, \epsilon) & \text{if } K \neq \emptyset \\ (\sigma_s, \sigma_c \cdot (\delta, a)) & \text{otherwise} \end{cases} \\
&\quad \text{with} \\
&\quad\quad (\sigma_s, \sigma_c) = \mathrm{store}(\sigma) \\
&\quad\quad K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))
\end{aligned}$$

with $\kappa_\varphi(T, \sigma_s, \sigma_c) \stackrel{\text{def}}{=} \{w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid w \preccurlyeq_d \sigma_c \wedge |w| = |\sigma_c| \wedge \sigma_s \cdot w \models \varphi \wedge \mathrm{delay}(w(1)) \geq T - \mathrm{time}(\sigma_s))\}$.

From $(\mathbf{Snd})$ and $(\mathbf{Tr})$, we define the constraints that hold for a particular word σ and a particular time instant t (i.e., universal quantifications are removed). We denote these propositions by $(\mathbf{Snd})_{\sigma,t}$ and $(\mathbf{Tr})_{\sigma,t}$ respectively. Thus, we shall prove that E_φ satisfies $(\mathbf{Snd})_{\sigma,t}$ and $(\mathbf{Tr})_{\sigma,t}$ for any $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and $t \in \mathbb{R}_{\geq 0}$. For this purpose, we perform an induction on the length of σ.

Induction Basis. Let us suppose that $|\sigma| = 0$, thus $\sigma = \epsilon$ in $(\mathbb{R}_{\geq 0} \times \Sigma)^*$. First, we have $\forall t \in \mathbb{R}_{\geq 0} : \mathrm{obs}(\sigma, t) = \epsilon$. Second, we have $\mathrm{store}(\epsilon) = (\epsilon, \epsilon)$. Consequently, we have $\forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) = \epsilon$. We now prove that, at any time $t \in \mathbb{R}_{\geq 0}$, E_φ satisfies $(\mathbf{Snd})_{\epsilon,t}$ and $(\mathbf{Tr})_{\epsilon,t}$, successively.

- For $\sigma = \epsilon$, we have $\forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) = \epsilon$. Thus, E_φ satisfies $(\mathbf{Snd})_{\epsilon,t}$, for any time $t \in \mathbb{R}_{\geq 0}$.
- Since $\mathrm{obs}(\epsilon, t) = \epsilon$ and $\epsilon \preccurlyeq_d \epsilon$, we have $\forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\epsilon, t) \preccurlyeq_d \epsilon$. That is, E_φ satisfies $(\mathbf{Tr})_{\epsilon,t}$, for any time $t \in \mathbb{R}_{\geq 0}$.

Induction Step. Let us consider $n \in \mathbb{N}$ and suppose that for any $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ with $|\sigma| \leq n$, any $t \in \mathbb{R}_{\geq 0}$, E_φ satisfies $(\mathbf{Snd})_{\sigma,t}$ and $(\mathbf{Tr})_{\sigma,t}$.

Let us now prove that for any $\sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ with $|\sigma'| = n+1$, for any $t \in \mathbb{R}_{\geq 0}$, E_φ satisfies $(\mathbf{Snd})_{\sigma',t}$ and $(\mathbf{Tr})_{\sigma',t}$. For this purpose, let us consider some input timed word σ' with $|\sigma'| = n+1$. Thus $\sigma' = \sigma \cdot (\delta, a)$ for some σ with $|\sigma| = n$, $\delta \in \mathbb{R}_{\geq 0}$ and $a \in \Sigma$. Let us consider some time instant $t \in \mathbb{R}_{\geq 0}$.

We distinguish two cases according to whether the sum of delays of the timed word $\sigma \cdot (\delta, a)$ is greater than t or not, i.e., whether $\mathrm{time}(\sigma \cdot (\delta, a)) > t$ or not.

- Case $\mathrm{time}(\sigma \cdot (\delta, a)) > t$. In this case, $\mathrm{obs}(\sigma \cdot (\delta, a), t) = \mathrm{obs}(\sigma, t)$. Consequently, $E_\varphi(\sigma \cdot (\delta, a), t) = \mathrm{obs}\Big(\Pi_1\big(\mathrm{store}(\mathrm{obs}(\sigma \cdot (\delta, a), t))\big), t\Big) = E_\varphi(\sigma, t)$. From the induction hypothesis, we directly deduce that E_φ satisfies $(\mathbf{Snd})_{\sigma',t}$ and $(\mathbf{Tr})_{\sigma',t}$.
- Case $\mathrm{time}(\sigma \cdot (\delta, a)) \leq t$. In this case $\mathrm{obs}(\sigma \cdot (\delta, a), t) = \sigma \cdot (\delta, a)$.
  We distinguish two cases, based on whether $K = \emptyset$, or not.
  - Case $K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a)) = \emptyset$. From the definition of store, we have $\mathrm{store}(\sigma \cdot (\delta, a)) = (\sigma_s, \sigma_c \cdot (\delta, a))$, and $\Pi_1\big(\mathrm{store}(\sigma \cdot (\delta, a))\big) = \sigma_s$. We also have $\Pi_1\big(\mathrm{store}(\sigma)\big) = \sigma_s$. From the definition of E_φ, we have $E_\varphi(\sigma \cdot (\delta, a), t) = E_\varphi(\sigma, t)$. From the induction hypothesis, we deduce that E_φ satisfies $(\mathbf{Snd})_{\sigma',t}$ and $(\mathbf{Tr})_{\sigma',t}$.
  - Case $K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a)) \neq \emptyset$.
    From the definition of store, we have $\mathrm{store}(\sigma \cdot (\delta, a)) = ((\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K, \epsilon)$, and $\Pi_1\big(\mathrm{store}(\sigma \cdot (\delta, a))\big) = (\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K)$. We distinguish two cases based on whether $\mathrm{time}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K) > t$ or not.
    - Case $\mathrm{time}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K) > t$.
      We further distinguish two more cases based on whether $\mathrm{time}(\sigma_s) > t$ or not.
      - Case $\mathrm{time}(\sigma_s) > t$. In this case, we have $\mathrm{obs}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K, t) = \mathrm{obs}(\sigma_s, t)$. Thus we have $E_\varphi(\sigma \cdot (\delta, a), t) = E_\varphi(\sigma, t)$. So, from the induction hypothesis, we deduce that E_φ satisfies $(\mathbf{Snd})_{\sigma',t}$ and $(\mathbf{Tr})_{\sigma',t}$.
      - Case $\mathrm{time}(\sigma_s) \leq t$. In this case we have $E_\varphi(\sigma \cdot (\delta, a), t) = \sigma_s \cdot O$, where $O \prec \min_{\preceq_{\mathrm{lex,time}}} K$. From the definition of K and κ_φ we know that $\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K \in \varphi$. Since $\forall t' \geq \mathrm{time}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K)$ and $E_\varphi(\sigma \cdot (\delta, a)) = \sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K$, E_φ satisfies $(\mathbf{Snd})_{\sigma',t}$, for any $t \in \mathbb{R}_{\geq 0}$. From the induction hypothesis, we know that $\sigma_s \preccurlyeq_d \sigma$. From the definition of K and κ_φ, and using the induction hypothesis, we can conclude that $\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K \preccurlyeq_d \sigma \cdot (\delta, a)$, and thus we also have $\sigma_s \cdot O \preccurlyeq_d \sigma \cdot (\delta, a)$. Thus E_φ satisfies $(\mathbf{Tr})_{\sigma',t}$.
    - Case $\mathrm{time}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K) \leq t$. In this case, we have $E_\varphi(\sigma \cdot (\delta, a), t) = \sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K$. From the definition of K and κ_φ we know that $\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K \in \varphi$. Thus E_φ satisfies $(\mathbf{Snd})_{\sigma',t}$. From the induction hypothesis, we know that $\sigma_s \preccurlyeq_d \sigma$. From the definition of K and κ_φ, and using the induction hypothesis, we can conclude that $\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K \preccurlyeq_d \sigma \cdot (\delta, a)$. Thus E_φ satisfies $(\mathbf{Tr})_{\sigma',t}$.

A.3 Proof of Proposition 4

We shall prove that, given a property φ, the associated enforcement function $E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ as per Definition 8 satisfies the optimality constraint $(\mathbf{Op})$ (from Definition 4).

$$
\begin{aligned}
&\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) \neq \epsilon \wedge E_\varphi(\sigma, t) \models \varphi \\
&\quad \Longrightarrow \exists w_{\mathrm{mx}}, w \in (\mathbb{R}_{\geq 0} \times \Sigma)^* : \\
&\quad w_{\mathrm{mx}} = \max^{\varphi}_{\prec,\epsilon}(E_\varphi(\sigma, t)) \\
&\quad \wedge E_\varphi(\sigma, t) = w_{\mathrm{mx}} \cdot w \\
&\quad \wedge \mathrm{time}(w) = \min\{\mathrm{time}(w') \mid \mathrm{delay}(w'(1)) \geq \mathrm{time}(\sigma_{[1 \cdots |E_\varphi(\sigma,t)|]}) - \mathrm{time}(w_{\mathrm{mx}}) \\
&\qquad\qquad \wedge w_{\mathrm{mx}} \cdot w' \models \varphi \wedge \Pi_\Sigma(w') = \Pi_\Sigma(w)\}.
\end{aligned}
$$

where the function E_φ is defined in Sec. 3.2.1 and recalled in the previous proof.

From $(\mathbf{Op})$, we define the constraint dedicated to a particular word σ and a particular time instant t (i.e., universal quantifications are removed). We denote this proposition by $(\mathbf{Op})_{\sigma,t}$. Thus, we shall prove that E_φ

satisfies $(\mathbf{Op})_{\sigma,t}$ for any $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and $t \in \mathbb{R}_{\geq 0}$. For this purpose, we perform an induction on the length of σ.

Induction Basis. Let us suppose that $|\sigma| = 0$, thus $\sigma = \epsilon$ in $(\mathbb{R}_{\geq 0} \times \Sigma)^*$. First, we have $\forall t \in \mathbb{R}_{\geq 0}$: $\mathrm{obs}(\sigma, t) = \epsilon$. Second, we have $\mathrm{store}(\epsilon) = (\epsilon, \epsilon)$. Consequently, we have $\forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) = \epsilon$. For $\sigma = \epsilon$, we have $\forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) = \epsilon$. Thus, E_φ vacuously satisfies $(\mathbf{Op})_{\epsilon,t}$, for any $t \in \mathbb{R}_{\geq 0}$.

Induction Step. Let us consider $n \in \mathbb{N}$ and suppose that for any $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ with $|\sigma| \leq n$, any $t \in \mathbb{R}_{\geq 0}$, E_φ satisfies $(\mathbf{Op})_{\sigma,t}$.

Let us now prove that, for any $\sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ with $|\sigma'| = n + 1$, for any $t \in \mathbb{R}_{\geq 0}$, E_φ satisfies $(\mathbf{Op})_{\sigma',t}$. For this purpose, let us consider some input timed word σ' with $|\sigma'| = n+1$. Thus $\sigma' = \sigma \cdot (\delta, a)$ for some σ with $|\sigma| = n$, $\delta \in \mathbb{R}_{\geq 0}$ and $a \in \Sigma$. Let us consider some time instant $t \in \mathbb{R}_{\geq 0}$.

We distinguish two cases according to whether the sum of delays of the timed word $\sigma \cdot (\delta, a)$ is greater than t or not, i.e., whether $\mathrm{time}(\sigma \cdot (\delta, a)) > t$ or not.

- Case $\mathrm{time}(\sigma \cdot (\delta, a)) > t$. In this case, $\mathrm{obs}(\sigma \cdot (\delta, a), t) = \mathrm{obs}(\sigma, t)$. Consequently, $E_\varphi(\sigma \cdot (\delta, a), t) = \mathrm{obs}\Big(\Pi_1\big(\mathrm{store}(\mathrm{obs}(\sigma \cdot (\delta, a), t))\big), t\Big) = E_\varphi(\sigma, t)$. From the induction hypothesis, we directly deduce that E_φ satisfies $(\mathbf{Op})_{\sigma',t}$.
- Case $\mathrm{time}(\sigma \cdot (\delta, a)) \leq t$. We have $\mathrm{obs}(\sigma \cdot (\delta, a), t) = \sigma \cdot (\delta, a)$.
  We distinguish two cases, based on whether $K = \emptyset$, or not, where $K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))$.
  - Case $K = \emptyset$. From the definition of store, we have $\mathrm{store}(\sigma \cdot (\delta, a)) = (\sigma_s, \sigma_c \cdot (\delta, a))$, and $\Pi_1\big(\mathrm{store}(\sigma \cdot (\delta, a))\big) = \sigma_s$. We also have $\Pi_1\big(\mathrm{store}(\sigma)\big) = \sigma_s$. From the definition of E_φ, we have $E_\varphi(\sigma \cdot (\delta, a), t) = E_\varphi(\sigma, t)$. From the induction hypothesis, we deduce that E_φ satisfies $(\mathbf{Op})_{\sigma',t}$.
  - Case $K \neq \emptyset$. From the definition of store, we have $\mathrm{store}(\sigma \cdot (\delta, a)) = ((\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K, \epsilon)$, and $\Pi_1\big(\mathrm{store}(\sigma \cdot (\delta, a))\big) = (\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K)$. We distinguish two cases based on whether $\mathrm{time}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K) > t$ or not.
    - Case $\mathrm{time}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K) > t$. We further distinguish two more cases based on whether $\mathrm{time}(\sigma_s) > t$ or not.
      - Case $\mathrm{time}(\sigma_s) > t$. We have $\mathrm{obs}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K, t) = \mathrm{obs}(\sigma_s, t)$. Thus, we have $E_\varphi(\sigma \cdot (\delta, a), t) = E_\varphi(\sigma, t)$. Hence, from the induction hypothesis, we deduce that E_φ satisfies $(\mathbf{Op})_{\sigma',t}$.
      - Case $\mathrm{time}(\sigma_s) \leq t$. We have $E_\varphi(\sigma \cdot (\delta, a), t) = \sigma_s \cdot O$, where $O \prec \min_{\preceq_{\mathrm{lex,time}}} K$. According to the definition of store, $\sigma_s \cdot O \not\models \varphi$. Thus, E_φ vacuously satisfies $(\mathbf{Op})_{\sigma',t}$.
    - Case $\mathrm{time}(\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K) \leq t$. We have $E_\varphi(\sigma \cdot (\delta, a), t) = \sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K$. From the definition of K and κ_φ, we know that $\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K \models \varphi$. From the definition of store, we know that σ_s is the maximal strict prefix of $\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K$, which satisfies φ. The last subsequence which again makes the output satisfy the property is $\min_{\preceq_{\mathrm{lex,time}}} K$. From the definition of K and κ_φ, we also know that the $\mathrm{delay}(\min_{\preceq_{\mathrm{lex,time}}} K(1)) \geq \mathrm{time}(\sigma) + \delta - \mathrm{time}(\sigma_s)$. Thus, we can conclude that E_φ satisfies $(\mathbf{Op})_{\sigma',t}$.

A.4 Preliminaries to the Proof of Proposition 5: Characterizing the Configurations of Enforcement Monitors

We first convey some remarks (Sec. A.4.1), define some notions (Sec. A.4.2) and lemmas (Sec. A.4.3) related to the configurations of enforcement monitors.

A.4.1 Some Remarks

Remark 12 In the following proofs, without loss of generality, we assume that at any time only one of the rules of the enforcement monitor applies. This simplification does not come at the price of reducing the generality nor the validity of the proofs because i) the ***store*** and ***dump*** rules of the enforcement monitor assign different variables and do not rely on the same conditions, and ii) the operations of enforcement

monitors are assumed to be executed in zero time. The considered simplification however reduces the number of (equivalent cases) in the following proofs.

Remark 13 Between the occurrence of two (input or output) events the configuration of the enforcement monitor evolves according to the ***idle*** rule (since it is the rule with lowest priority). To simplify notations we will use a rule to simplify the representation of $\mathcal{E}_{\mathrm{ioo}} \in \big((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}\big) \times Op \times \big((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}\big)$ stating that

$$\sigma \cdot (\epsilon, \mathrm{idle}(\delta_1), \epsilon) \cdot (\epsilon, \mathrm{idle}(\delta_2), \epsilon) \cdot \sigma' \text{ is equivalent to } \sigma \cdot (\epsilon, \mathrm{idle}(\delta_1 + \delta_2), \epsilon) \cdot \sigma'$$

for any $\sigma, \sigma' \in \big((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}\big) \times Op \times \big((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}\big)$ and $\delta_1, \delta_2 \in \mathbb{R}_{\geq 0}$. Thus for $\mathcal{E}_{\mathrm{ioo}}$ we will only consider sequences of $\big((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}\big) \times Op \times \big((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}\big)$ where delays appearing in the idle operation are maximal (i.e., there is no sequence of two consecutive events with an idle operation).

A.4.2 Some notations

Since it is assumed that at most one rule of the enforcement monitor applies at any time, let us define the functions $\mathrm{config}_{\mathrm{in}}, \mathrm{config}_{\mathrm{out}} : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to C^{\mathcal{E}}$ that give respectively the input and output configurations of an enforcement monitor reading an input sequence at some time instant. More formally, given some $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, t \in \mathbb{R}_{\geq 0}$:

- $\mathrm{config}_{\mathrm{in}}(\sigma, t) = c^t_\sigma$ such that $c^{\mathcal{E}}_0 \overset{w(\sigma,t)}{\hookrightarrow^*} c^t_\sigma$ where $w(\sigma, t) \overset{\mathrm{def}}{=} \min_{\preceq}\{ioo \preceq \mathcal{E}_{\mathrm{ioo}}(\sigma, t) \mid \mathrm{timeop}(ioo) = t\}$;
- $\mathrm{config}_{\mathrm{out}}(\sigma, t) = c^t_\sigma$ such that $c^{\mathcal{E}}_0 \overset{\mathcal{E}_{\mathrm{ioo}}(\sigma,t)}{\hookrightarrow^*} c^t_\sigma$.

Observe that, when at some time instant, only the ***idle*** rule applies, $\mathrm{config}_{\mathrm{in}}(\sigma, t) = \mathrm{config}_{\mathrm{out}}(\sigma, t)$ holds.

Moreover, for any two $t, t' \in \mathbb{R}_{\geq 0}$ such that $t' \geq t$, we note $\mathcal{E}(\sigma, t, t')$ for $\mathcal{E}(\sigma, t') \setminus \mathcal{E}(\sigma, t)$, i.e., the output sequence of an enforcement monitor between t and t'.

A.4.3 Some Intermediate Lemmas

Before tackling the proof of Proposition 5, we give a list of lemmas that describes the behavior of an enforcement monitor, describing the configurations or the output at some particular time instant for some input and memory content.

Similarly to the first physical constraint, the following lemma states that the enforcement monitor cannot change what it has output. More precisely, when the enforcement monitor is seen as function $\mathcal{E}$, the output is monotonic w.r.t. $\preceq$.

Lemma 1 (Monotonicity of the outputs of enforcement monitors) *Function* $\mathcal{E} : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ *is monotonic in its second parameter:* $\forall \sigma \in (\Sigma \times \mathbb{R}_{\geq 0})^*, \forall t, t' \in \mathbb{R}_{\geq 0} : t \leq t' \implies \mathcal{E}(\sigma, t) \preceq \mathcal{E}(\sigma, t')$.

The lemma states that for any input sequence σ, if we consider two time instants t, t' such that $t \leq t'$, then the output of the enforcement monitor at time t is a prefix of the output at time t'.

Proof (of Lemma 1) The proof directly follows from the definitions of the function $\mathcal{E}$ associated to an enforcement monitor (see Sec. 3.4, p. 19) which directly depends on $\mathcal{E}_{\mathrm{ioo}}$, which is itself monotonic over time (because of the definition of enforcement monitors). □

As a consequence, one can naturally split the output of the enforcement monitor over time, as it is stated by the following corollary.

Corollary 1 (Separation of the output of the enforcement monitor over time)

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t_1, t_2, t_3 \in \mathbb{R}_{\geq 0} : t_1 \leq t_2 \leq t_3 \implies \mathcal{E}(\sigma, t_1, t_3) = \mathcal{E}(\sigma, t_1, t_2) \cdot \mathcal{E}(\sigma, t_2, t_3).$$

The corollary states that for any sequence σ input to $\mathcal{E}$, if we consider three time instants $t_1, t_2, t_3 \in \mathbb{R}_{\geq 0}$ such that $t_1 \leq t_2 \leq t_3$, the output of $\mathcal{E}$ between t_1 and t_3 is the concatenation of the output between t_1 and t_2 and the output between t_2 and t_3.

Proof (of Corollary 1) The corollary directly follows from Lemma 1. □

The following lemma states that, at some time instant t, the output of the enforcement monitor only depends on what has been observed until time t. In other words, the enforcement monitor can work in an online fashion.

Lemma 2 (Dependency of the output on the observation only)

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : \mathcal{E}(\sigma, t) = \mathcal{E}(\mathrm{obs}(\sigma, t), t).$$

Proof (of Lemma 2) The proof of the lemma directly follows from the definitions of $\mathcal{E}_{\mathrm{ioo}}$ (Definition 10, p. 20) and obs (in Sec. 2). Indeed, using $\mathrm{obs}(\sigma, t) = \mathrm{obs}(\mathrm{obs}(\sigma, t), t)$, we deduce that $\mathcal{E}_{\mathrm{ioo}}(\sigma, t) = \mathcal{E}_{\mathrm{ioo}}(\mathrm{obs}(\sigma, t), t)$, for any $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and $t \in \mathbb{R}_{\geq 0}$. Using $\mathcal{E}(\sigma, t) = \Pi_3(\mathcal{E}_{\mathrm{ioo}}(\sigma, t))$, we can deduce the expected result. □

The following lemma indicates the value of the store variable inside the configurations at some special time instants (corresponding to the time necessary to read prefixes of the input sequence).

Lemma 3 (Values of $\mathrm{config}_{\mathrm{in}}$ when reading events)

$$\begin{aligned}&\forall \sigma, \sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : \\ &\quad \big(\sigma' \preceq \sigma \wedge t = \mathrm{time}(\sigma')\big) \implies \mathrm{config}_{\mathrm{in}}(\sigma, t) = (_, _, \mathrm{time}(\mathrm{last}(\sigma')), _, _, _).\end{aligned}$$

Proof (of Lemma 3) The proof can be done using a straightforward induction on the length of the maximal considered prefix σ' of σ. Moreover, the proof uses the facts that between two prefixes of the input sequence, i) no modification is brought to the store variable, and ii) the store variable is reset upon the store of each event. □

The following lemma states that only the memory content σ_s and the value of the dump variable influence the output of the enforcement monitor. More specifically, if, after reading some sequence, an enforcement monitor reaches some configuration, its future output is fully determined by the memory content σ_s(containing the corrected sequence) and the value of the dump variable (d), during the total time needed to output it.

Lemma 4 (Values of $\mathrm{config}_{\mathrm{out}}$ when releasing events)

$$\begin{aligned}&\forall \sigma, \sigma_s, \sigma_c \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0}, \forall s, d, m_t \in \mathbb{R}_{\geq 0}, \forall q \in Q : \\ &\quad t \geq \mathrm{time}(\sigma) \wedge \mathrm{config}_{\mathrm{out}}(\sigma, t) = (\sigma_s, \sigma_c, s, d, m_t, q) \\ &\qquad \implies \forall \sigma'_s \preceq \sigma_s, \exists \sigma'_c \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \exists s' \in \mathbb{R}_{\geq 0}, \exists m'_t \in \mathbb{R}_{\geq 0}, \exists q' \in Q : \\ &\qquad\quad \mathrm{config}_{\mathrm{out}}(\sigma, t + \mathrm{time}(\sigma'_s) - d) = (\sigma_s \setminus \sigma'_s, \sigma'_c, s', 0, m'_t, q').\end{aligned}$$

The lemma states that whatever is the output configuration $(\sigma_s, \sigma_c, s, d, m_t, q)$ reached by reading some input sequence σ at some time instant $t \geq \mathrm{time}(\sigma)$, then for any prefix σ'_s of σ_s the output configuration reached at time $t + \mathrm{time}(\sigma'_s) - d$ (we add the time needed to read σ'_s minus the value of the dump clock) is such that σ'_s has been released from the memory (the memory is $\sigma_s \setminus \sigma'_s$) and the value of the dump variable has just been reset to 0.

Proof (of Lemma 4) The proof is a straightforward induction on the length of σ'_s. It uses the fact that the considered configurations occur at time instants greater than $\mathrm{time}(\sigma)$, hence implying that no input event can be read any more. Consequently, following the definition of the enforcement monitor (Definition 9, p. 18), on the configurations of the enforcement monitor, only the ***idle*** and ***dump*** rules apply. Between $\mathrm{time}(\sigma'_s)$ and $\mathrm{time}(\sigma'_s \cdot (\delta, a))$ where $\sigma'_s \preceq \sigma'_s \cdot (\delta, a) \preceq \sigma_s$, the configuration of the enforcement monitor evolves only using the ***idle*** rule (no other rule applies) until $\mathrm{config}_{\mathrm{in}}(\sigma, t + \mathrm{time}(\sigma'_s)) = (\sigma_s \setminus \sigma'_s, \sigma_c, s' + \delta, \delta, m_t, q)$ with $\sigma_s \setminus \sigma'_s$. The ***dump*** rule is then applied to get the following derivation $(\sigma_s \setminus \sigma'_s, \sigma_c, s' + \delta, \delta, m_t, q) \overset{\epsilon/dump(\delta,a)/\epsilon}{\hookrightarrow} (\sigma_s \setminus (\sigma'_s \cdot (\delta, a)), \sigma_c, s' + \delta, 0, m_t, q)$. □

The following lemma states that when an enforcement monitor has nothing to read in input anymore, what is output is the observation of its memory content over time.

Lemma 5

$$\begin{aligned}&\forall \sigma, \sigma_s, \sigma_c \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0}, \forall s, d, m_t \in \mathbb{R}_{\geq 0}, \forall q \in Q : \\ &\quad t \geq \mathrm{time}(\sigma) \wedge \mathrm{config}_{\mathrm{out}}(\sigma, t) = (\sigma_s, \sigma_c, s, d, m_t, q) \\ &\qquad \implies \forall \sigma'_s \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t' \in \mathbb{R}_{\geq 0} : \\ &\qquad\quad \sigma'_s \preceq \sigma_s \wedge d \leq t' \leq \mathrm{time}(\sigma'_s) \implies \mathcal{E}(\sigma, t, t + t' - d) = \mathrm{obs}(\sigma'_s, t').\end{aligned}$$

The lemma states that, if after some time t, after reading an input sequence σ, the enforcement monitor is in an output configuration that contains σ_s as a memory content, then whatever is the prefix σ'_s of σ_s we consider, the output of the enforcement monitor between t and $t + \mathrm{time}(\sigma'_s)$ is the observation of σ'_s.

Proof (of Lemma 5) The proof is performed by induction on the length of σ'_s and uses Lemma 4.

- Case $|\sigma'_s| = 0$. In this case, $\sigma'_s = \epsilon$ and since $\mathrm{time}(\epsilon) = 0$ and $\mathcal{E}(\sigma, t, t - d)$ is not defined, the lemma vacuously holds.
- Induction case. Let us suppose that the lemma holds for all prefixes σ'_s of some maximum length $n \in [0, |\sigma_s|-1]$. Let us consider σ' the prefix of σ_s of length $n+1$. On the one hand, at time $t+\mathrm{time}(\sigma')-d$, i.e., when $t' = \mathrm{time}(\sigma') - d$, according to Lemma 4, we have $\mathrm{config}_{\mathrm{out}}(\sigma, t + \mathrm{time}(\sigma') - d) = ((\delta, a), \sigma_c, s, 0, m_t, q)$ for some $s, m_t \in \mathbb{R}_{\geq 0}$ and $\sigma_c \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$. On the other hand, let us consider some $t' \in [0, \delta]$, we have:

  $$\mathcal{E}(\sigma, t, t+\mathrm{time}(\sigma')+t'-d) = \mathcal{E}(\sigma, t, t+\mathrm{time}(\sigma')-d)\cdot\mathcal{E}(\sigma, t+\mathrm{time}(\sigma')-d, t+\mathrm{time}(\sigma')+t'-d).$$

  Using the induction hypothesis, we find $\mathcal{E}(\sigma, t, t + \mathrm{time}(\sigma') - d) = \mathrm{obs}(\sigma', \mathrm{time}(\sigma')) = \sigma'$. Using the semantics of the enforcement monitor (***dump*** and ***idle*** rules), we obtain $\mathcal{E}(\sigma, t+\mathrm{time}(\sigma')+t'-d) = \mathrm{obs}((\delta, a), t'+d)$. Thus, $\mathcal{E}(\sigma, t, t+\mathrm{time}(\sigma')+t'-d) = \sigma'\cdot\mathrm{obs}((\delta, a), t'+d) = \mathrm{obs}(\sigma'\cdot(\delta, a), t'+d)$.

□

A.5 Proof of Proposition 5: Relation between Enforcement Function and Enforcement Monitor

Proof. We shall prove that, given a property φ, the associated enforcement monitor as per Definition 9 (p. 18) implements the associated enforcement function $E_\varphi : (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0} \times \Sigma)^*$ as per Definition 8 (p. 12). That is:

$$\forall \sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*, \forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) = \mathcal{E}(\sigma, t).$$

The proof is done by induction on the length of the input timed word σ.

Induction Basis. Let us suppose that $|\sigma| = 0$, thus $\sigma = \epsilon$ in $(\mathbb{R}_{\geq 0} \times \Sigma)^*$. On the one hand, we have $\forall t \in \mathbb{R}_{\geq 0} : E_\varphi(\sigma, t) = \epsilon$. On the other hand, the word $\mathcal{E}_{\mathrm{ioo}}(\epsilon, t)$ over the input-operation-output alphabet is such that $\forall t \in \mathbb{R}_{\geq 0} : \mathcal{E}_{\mathrm{ioo}}(\epsilon, t) = \epsilon$. Thus, according to the definition of the enforcement monitor, $\mathrm{store}{-}\overline{\varphi}$ and $\mathrm{store}{-}\varphi$ rules cannot be applied. Consequently, the memory of the enforcement monitor remains empty as in the initial configuration. It follows that the ***dump*** rule cannot be applied. We have then $C_0^{\mathcal{E}} \overset{\epsilon/\,\mathrm{idle}(t)/\epsilon}{\hookrightarrow} (\epsilon, \epsilon, t, t, 0, q)$, and thus $\mathcal{E}(\epsilon, t) = \epsilon$.

Induction Step. Let us suppose that $E_\varphi(\sigma, t) = \mathcal{E}(\sigma, t)$ for any timed word $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ of some length $n \in \mathbb{N}$, at any time $t \in \mathbb{R}_{\geq 0}$. Let us now consider some input timed word $\sigma \cdot (\delta, a)$ for some $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ with $|\sigma| = n$, $\delta \in \mathbb{R}_{\geq 0}$, and $a \in \Sigma$. We want to prove that $E_\varphi(\sigma \cdot (\delta, a), t) = \mathcal{E}(\sigma \cdot (\delta, a), t)$, at any $t \in \mathbb{R}_{\geq 0}$.

Let us consider some time instant $t \in \mathbb{R}_{\geq 0}$. We distinguish two cases according to whether $\mathrm{time}(\sigma \cdot (\delta, a)) > t$ or not, that is whether $\sigma \cdot (\delta, a)$ is completely observed or not at time t.

- Case $\mathrm{time}(\sigma \cdot (\delta, a)) > t$. In this case, $\mathrm{obs}\,(\sigma \cdot (\delta, a), t) = \mathrm{obs}(\sigma, t)$, i.e., at time t, the observations of σ and $\sigma \cdot (\delta, a)$ are identical. On the one hand, we have: $E_\varphi\,(\sigma \cdot (\delta, a), t) = \mathrm{obs}\Big(\Pi_1\big(\mathrm{store}(\mathrm{obs}(\sigma \cdot (\delta, a), t))\big), t\Big) = \mathrm{obs}\Big(\Pi_1\big(\mathrm{store}(\mathrm{obs}(\sigma, t))\big), t\Big) = E_\varphi(\sigma, t)$. On the other hand, regarding the enforcement monitor, since $\mathrm{obs}\big(\sigma \cdot (\delta, a), t\big) = \mathrm{obs}(\sigma, t)$, using Lemma 2 (p. 36), we obtain $\mathcal{E}(\sigma \cdot (\delta, a), t) = \mathcal{E}(\sigma, t)$. Using the induction hypothesis, we can conclude that $E_\varphi(\sigma \cdot (\delta, a), t) = \mathcal{E}(\sigma \cdot (\delta, a), t)$.

- Case $\mathrm{time}(\sigma \cdot (\delta, a)) \leq t$. In this case, we have $\mathrm{obs}(\sigma \cdot (\delta, a), t) = \sigma \cdot (\delta, a)$ (i.e., $\sigma \cdot (\delta, a)$ has been observed entirely). Using Lemma 3 (p. 36), we know that the configuration of the enforcement monitor at time $\mathrm{time}(\sigma \cdot (\delta, a))$ is $\mathrm{config}_{\mathrm{in}}\big(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))\big) = (\sigma_s, \sigma_c, \delta, d, m_t, q_\sigma)$ for some $\sigma_s, \sigma_c \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$, $\delta, d, m_t \in \mathbb{R}_{\geq 0}$, $q_\sigma \in Q$. Observe that $\mathrm{config}_{\mathrm{in}}\big(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))\big) = \mathrm{config}_{\mathrm{in}}\big(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))\big)$ because of i) the definition of $\mathrm{config}_{\mathrm{in}}$ using the definition of $\mathcal{E}_{\mathrm{ioo}}$ and ii) the event (δ, a) has not been yet consumed through none of the ***store*** rules by the enforcement monitor at time $\mathrm{time}(\sigma \cdot (\delta, a))$.
  We distinguish two cases according to whether $\sigma_c \cdot (\delta, a)$ can be delayed into a word satisfying φ or not, i.e., whether $K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a)) = \emptyset$, or not.
  - Case $K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a)) = \emptyset$. From the definition of store function, we have $\mathrm{store}(\sigma \cdot (\delta, a)) = (\sigma_s, \sigma_c \cdot (\delta, a))$, and $\Pi_1\big(\mathrm{store}(\sigma \cdot (\delta, a))\big) = \sigma_s$. We also have $\Pi_1\big(\mathrm{store}(\sigma)\big) = \sigma_s$. From the definition of E_φ, we have $E_\varphi(\sigma \cdot (\delta, a), t) = E_\varphi(\sigma, t)$.
    Now, regarding $\mathcal{E}$, according to the definition of update, we have $\mathrm{update}(q_\sigma, \sigma_c \cdot (\delta, a), m_t + \delta) = (\sigma_c, \mathtt{ff})$. According to the definition of the transition relation, we have:

$$(\sigma_s, \sigma_c, \delta, d, m_t, q_\sigma) \overset{(\delta,a)/\mathrm{store}-\overline{\varphi}(\delta,a)/\epsilon}{\hookrightarrow} (\sigma_s, \sigma_c \cdot (\delta, a), 0, d, m_t + \delta, q_\sigma).$$

    Thus $\mathrm{config}_{\mathrm{out}}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) = (\sigma_s, \sigma_c \cdot (\delta, a), 0, d, m_t + \delta, q_\sigma)$.
    Let us consider $t_\epsilon \in \mathbb{R}_{\geq 0}$ such that between $\mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon$ and $\mathrm{time}(\sigma \cdot (\delta, a))$, the enforcement monitor does not read any input nor produce any output, i.e., for all $t \in [\mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon, \mathrm{time}(\sigma \cdot (\delta, a))]$, $\mathrm{config}(t)$ is such that only the ***idle*** rule applies.
    Let us examine $\mathcal{E}(\sigma \cdot (\delta, a), t)$. We have:

$$\begin{aligned}\mathcal{E}(\sigma \cdot (\delta, a), t) = {} & \mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon) \\ & \cdot \mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon, \mathrm{time}(\sigma \cdot (\delta, a))) \\ & \cdot \mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)), t).\end{aligned}$$

    Let us examine $\mathcal{E}(\sigma, t)$. We have:

$$\begin{aligned}\mathcal{E}(\sigma, t) = {} & \mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon) \\ & \cdot \mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon, \mathrm{time}(\sigma \cdot (\delta, a))) \\ & \cdot \mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)), t).\end{aligned}$$

    Observe that $\mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon) = \mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon)$ because $\mathrm{obs}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon) = \sigma$ according to the definition of obs. Moreover, $\mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon, \mathrm{time}(\sigma \cdot (\delta, a))) = \epsilon$ since only the ***idle*** rule applies during the considered time interval. Furthermore, according to Lemma 5, since $\mathrm{config}_{\mathrm{out}}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))) = (\sigma_s, \sigma_c \cdot (\delta, a), 0, d, m_t + \delta, q_\sigma)$, we get $\mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)), t) = \mathrm{obs}(\sigma_s, t - \mathrm{time}(\sigma \cdot (\delta, a)) + d) = \mathrm{obs}(\sigma_s, t - \mathrm{time}(\sigma \cdot (\delta, a)) + d)$. Moreover, we know that $\mathrm{config}_{\mathrm{in}}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) = (\sigma_s, \sigma_c, \delta, d, m_t, q_\sigma)$. Since the enforcement monitor is deterministic, and from Remark 12 (p. 34), we also get that $\mathrm{config}_{\mathrm{out}}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) = (\sigma_s, \sigma_c, \delta, d, m_t, q_\sigma)$. Using Lemma 5 (p. 36) again, we get $\mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)), t) = \mathrm{obs}(\sigma_s, t - \mathrm{time}(\sigma \cdot (\delta, a)) + d)$.
    Consequently we can deduce that $\mathcal{E}(\sigma \cdot (\delta, a), t) = \mathcal{E}(\sigma, t) = E_\varphi(\sigma, t) = E_\varphi(\sigma \cdot (\delta, a))$.
  - Case $K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a)) \neq \emptyset$. Regarding E_φ, from the definition of store function, we have $\mathrm{store}(\sigma \cdot (\delta, a)) = ((\sigma_s \cdot \min_{\preceq_{\mathrm{lex}}, time} K, \epsilon)$, and $\Pi_1\big(\mathrm{store}(\sigma \cdot (\delta, a))\big) = (\sigma_s \cdot \min_{\preceq_{\mathrm{lex,time}}} K)$. Regarding the enforcement monitor, according to the definition of update, we have $\mathrm{update}(q_\sigma, \sigma_c \cdot (\delta, a), m_t + \delta) = (\sigma'_c, \mathtt{tt})$. From the definition of the transition relation, we have:

$$(\sigma_s, \sigma_c, \delta, d, m_t, q_\sigma) \overset{(\delta,a)/\mathrm{store}-\varphi(\delta,a)/\epsilon}{\hookrightarrow} (\sigma_s \cdot \sigma'_c, \epsilon, 0, d, m'_t, q'),$$

    where:
    - $m'_t = m'_t + \delta - \mathrm{time}(\sigma'_c)$
    - q' is defined as $q \overset{\sigma'_c}{\rightarrow} q'$

    Thus $\mathrm{config}_{\mathrm{out}}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) = (\sigma_s \cdot \sigma'_c, \epsilon, 0, d, m'_t, q')$.
    Let us consider t_ϵ such that during $\mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon$ and $\mathrm{time}(\sigma \cdot (\delta, a))$, the enforcement monitor does not read any input nor produce any output, i.e., for all $t \in [\mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon, \mathrm{time}(\sigma \cdot (\delta, a))]$, $\mathrm{config}(t)$ is such that only the ***idle*** rule applies.

Let us examine $\mathcal{E}(\sigma \cdot (\delta, a), t)$. We have:

$$\begin{aligned}\mathcal{E}(\sigma \cdot (\delta, a), t) = \mathcal{E}(&\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon) \\ &\cdot \mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon, \mathrm{time}(\sigma \cdot (\delta, a))) \\ &\cdot \mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)), t).\end{aligned}$$

Let us examine $\mathcal{E}(\sigma, t)$. We have:

$$\begin{aligned}\mathcal{E}(\sigma, t) = \mathcal{E}(&\sigma, \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon) \\ &\cdot \mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon, \mathrm{time}(\sigma \cdot (\delta, a))) \\ &\cdot \mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)), t).\end{aligned}$$

Observe that $\mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon) = \mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon)$ because $\mathrm{obs}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon) = \sigma$ according to the definition of obs.
Moreover, $\mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)) - t_\epsilon, \mathrm{time}(\sigma \cdot (\delta, a))) = \epsilon$ since only the *idle* rule applies during the considered time interval.
Furthermore, according to Lemma 5, since $\mathrm{config}_{\mathrm{out}}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))) = (\sigma_s \cdot \sigma'_c, \epsilon, 0, d, m'_t, q')$, we get $\mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)), t) = \mathrm{obs}(\sigma_s \cdot \sigma'_c, t - \mathrm{time}(\sigma \cdot (\delta, a)) + d)$.
Now we further distinguish two more sub-cases, based on whether $\mathrm{time}(\sigma_s \cdot \sigma'_c) > t - \mathrm{time}(\sigma \cdot (\delta, a)) + d$ or not (whether all the elements in the memory can be released as output by time t or not).

- Case $\mathrm{time}(\sigma_s \cdot \sigma'_c) > t - \mathrm{time}(\sigma \cdot (\delta, a)) + d$.
  We further distinguish two more sub-cases based on whether $\mathrm{time}(\sigma_s) > t - \mathrm{time}(\sigma \cdot (\delta, a)) + d$, or not.
  - Case $\mathrm{time}(\sigma_s) > t - \mathrm{time}(\sigma \cdot (\delta, a)) + d$. In this case, we know that $\mathrm{obs}(\sigma_s \cdot \sigma'_c, t - \mathrm{time}(\sigma \cdot (\delta, a)) + d) = \mathrm{obs}(\sigma_s, t - \mathrm{time}(\sigma \cdot (\delta, a)) + d)$. Hence, we can derive that $\mathcal{E}(\sigma \cdot (\delta, a), t) = \mathcal{E}(\sigma, t)$. Also, from the induction hypothesis, we know that $\mathcal{E}(\sigma, t) = E_\varphi(\sigma, t)$.
    Regarding E_φ, we have $\mathrm{store}(\sigma \cdot (\delta, a)) = \Pi_1\big(\mathrm{store}(\sigma)\big) \cdot \min_{\preceq_{\mathrm{lex,time}}} K$. Moreover

    $$\begin{aligned}E_\varphi(\sigma \cdot (\delta, a), t) &= \mathrm{obs}\Big(\Pi_1\big(\mathrm{store}(\mathrm{obs}(\sigma, t))\big), t\Big) \\ &= \mathrm{obs}(\Pi_1\big(\mathrm{store}(\sigma)\big) \cdot \min_{\preceq_{\mathrm{lex,time}}} K, t).\end{aligned}$$

    We can have $E_\varphi(\sigma \cdot (\delta, a), t) = \Pi_1\big(\mathrm{store}(\sigma)\big) \cdot O$ where $O \preccurlyeq \min_{\preceq_{\mathrm{lex,time}}} K$ which is equal to $E_\varphi(\sigma, t) \cdot O$, only if the delays computed by the update function are different from the delays computed by E_φ. This would violate the induction hypothesis stating that $\mathcal{E}(\sigma, t) = E_\varphi(\sigma, t)$. Hence, we have $E_\varphi(\sigma \cdot (\delta, a), t) = \mathrm{obs}(\ \Pi_1\big(\mathrm{store}(\sigma)\big), t) = E_\varphi(\sigma, t)$. Thus, $E_\varphi(\sigma \cdot (\delta, a), t) = \mathcal{E}(\sigma \cdot (\delta, a), t)$.
  - Case $\mathrm{time}(\sigma_s) \leq t - \mathrm{time}(\sigma \cdot (\delta, a)) + d$. In this case, we can follow the same reasoning as in the previous case to obtain the expected result.
- Case $\mathrm{time}(\sigma_s \cdot \sigma'_c) \leq t - \mathrm{time}(\sigma \cdot (\delta, a)) + d$.
  In this case, similarly following Lemma 5 (p. 36), we have $\mathcal{E}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)), t) = \mathrm{obs}(\sigma_s \cdot \sigma'_c, t - \mathrm{time}(\sigma \cdot (\delta, a) + d) = \sigma_s \cdot \sigma'_c$. We can also derive that $\mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)), t) = \sigma_s$. Consequently, we have $\mathcal{E}(\sigma \cdot (\delta, a), t) = \mathcal{E}(\sigma, t) \cdot \sigma'_c$. From the induction hypothesis, we know that $E_\varphi(\sigma, t) = \mathcal{E}(\sigma, t)$, and we have $\mathcal{E}(\sigma \cdot (\delta, a), t) = E_\varphi(\sigma, t) \cdot \sigma'_c$. We have $\mathrm{store}(\sigma \cdot (\delta, a)) = \Pi_1\big(\mathrm{store}(\sigma)\big) \cdot \min_{\preceq_{\mathrm{lex,time}}} K$, and thus $E_\varphi(\sigma \cdot (\delta, a), t) = \mathrm{obs}\left(\Pi_1\left(\mathrm{store}\left(\sigma\right)\right) \cdot \min_{\preceq_{\mathrm{lex,time}}} K, t\right)$. Hence, in this case, we have $E_\varphi(\sigma \cdot (\delta, a), t) = \mathrm{store}(\sigma) \cdot \min_{\preceq_{\mathrm{lex,time}}} K = E_\varphi(\sigma, t) \cdot \min_{\preceq_{\mathrm{lex,time}}} K$, since the delays computed for the subsequence $\sigma_c \cdot (\delta, a)$ by E_φ and $\mathcal{E}$ are equal. Finally, we have $E_\varphi(\sigma \cdot (\delta, a), t) = \mathcal{E}(\sigma \cdot (\delta, a), t)$.

B Implementation of Store Processes

We provide a brief description related to how the store process, which processes the input events (checking for the satisfaction of the property and computing delays) is implemented.

B.1 Implementation of $\mathsf{StoreProcess}_{\mathsf{safety}}$

We now describe the implementation of $\mathsf{StoreProcess}_{\mathsf{safety}}$ used by an enforcement monitor for safety properties. The $\mathsf{StoreProcess}_{\mathsf{safety}}$ first parses the input model, and performs the necessary initialization. For each event of the trace in order, first the automaton representing the input trace is updated with the new event. Then the $\mathrm{update}_{\mathrm{s}}$ function is invoked, with the current state information and the event. The $\mathrm{update}_{\mathrm{s}}$ function returns `ff` if an accepting state is not reachable upon the event (δ, a) from the current state. In case if an accepting state is reachable, then the $\mathrm{update}_{\mathrm{s}}$ function returns the optimal delay with information about the state that is reachable. Before continuing with the next event, the $\mathsf{StoreProcess}$ updates the current state information. In the $\mathsf{StoreProcess}_{\mathsf{safety}}$, the execution time of $\mathrm{update}_{\mathrm{s}}$ is measured. The $\mathsf{StoreProcess}_{\mathsf{safety}}$ keeps track of the total time of $\mathrm{update}_{\mathrm{s}}$, by adding the $\mathrm{update}_{\mathrm{s}}$ time measured after each event to the total time, which is returned as a result of invoking the $\mathsf{StoreProcess}_{\mathsf{safety}}$.

B.2 Implementation of $\mathsf{StoreProcess}_{\mathsf{co-safety}}$

The $\mathsf{StoreProcess}_{\mathsf{co-safety}}$ is implemented following the algorithm and steps described in [16]. Below we provide an abstract description of the code implementing the $\mathrm{update}_{\mathrm{cs}}$ function. The $\mathrm{update}_{\mathrm{cs}}$ function takes a timed word (the events read by the enforcement monitor), and returns a new delay for each input event, if an accepting state is reachable for the given input timed word. First, all paths starting from the initial state for the given input sequence are computed. Then the set of all accepting paths is computed. In each path, if the location in the last state is accepting, then the path is accepting, and is a non-accepting path otherwise. If the set of accepting paths is empty, then $\mathrm{update}_{\mathrm{cs}}$ returns `ff`. If there are accepting paths, for each state in the path, a corresponding delay is computed, and the paths whose sums of delays are minimal are filtered. Among these paths, one path is chosen according to the lexical order, and the delays corresponding to this path are returned as the result.

References

1. Thati, P., Rosu, G.: Monitoring algorithms for metric temporal logic specifications. Electronic Notes in Theoretical Computer Science **113** (2005) 145–162
2. Chen, F., Rosu, G.: Parametric trace slicing and monitoring. In Kowalewski, S., Philippou, A., eds.: Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009). Volume 5505 of Lecture Notes in Computer Science., Springer (2009) 246–261
3. Nickovic, D., Piterman, N.: From MTL to deterministic timed automata. In Chatterjee, K., Henzinger, T.A., eds.: Proceedings of the 8th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2010). Volume 6246 of Lecture Notes in Computer Science., Springer (2010) 152–167
4. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20** (2011) 14:1–14:64
5. Basin, D., Klaedtke, F., Zalinescu, E.: Algorithms for monitoring real-time properties. In Khurshid, S., Sen, K., eds.: Proceedings of the 2nd International Conference on Runtime Verification (RV 2011). Volume 7186 of Lecture Notes in Computer Science., Springer-Verlag (2011) 260–275
6. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.: Quantified Event Automata: Towards expressive and efficient runtime monitors. In Giannakopoulou, D., Mèry, D., eds.: Proceedings of the 18th International Symposium on Formal Methods (FM 2012). Volume 7436 of Lecture Notes in Computer Science., Springer-Verlag (2012) 68–84
7. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3** (2000) 30–50
8. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. **12** (2009) 19:1–19:41
9. Falcone, Y.: You should better enforce than verify. In Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G.J., Rosu, G., Sokolsky, O., Tillmann, N., eds.: Proceedings of the 1st international conference on Runtime verification (RV 2010). Volume 6418 of Lecture Notes in Computer Science., Springer-Verlag (2010) 89–105
10. Falcone, Y., Mounier, L., Fernandez, J.C., Richier, J.L.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods in System Design **38** (2011) 223–262

11. Nickovic, D., Maler, O.: AMT: a property-based monitoring tool for analog systems. In Raskin, J.F., Thiagarajan, P.S., eds.: Proceedings of the 5th International Conference on Formal modeling and analysis of timed systems (FORMATS 2007). Volume 4763 of Lecture Notes in Computer Science., Springer-Verlag (2007) 304–319
12. Colombo, C., Pace, G.J., Schneider, G.: LARVA — safer monitoring of real-time Java programs (tool paper). In Hung, D.V., Krishnan, P., eds.: Proceedings of the 7th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2009), IEEE Computer Society (2009) 33–37
13. Matteucci, I.: Automated synthesis of enforcing mechanisms for security properties in a timed setting. Electron. Notes Theor. Comput. Sci. **186** (2007) 101–120
14. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. **16** (2013) 3:1–3:26
15. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H.: Runtime enforcement of parametric timed properties with practical applications. In: IEEE International Workshop on Discrete Event Systems. (2014) To appear.
16. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A., Timo, O.L.N.: Runtime enforcement of timed properties. In Qadeer, S., Tasiran, S., eds.: Proceedings of the 3rd International Conference on Runtime Verification (RV 2012). Volume 7687 of Lecture Notes in Computer Science., Springer (2012) 229–244
17. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H.: Runtime enforcement of regular timed properties. In: Software Verification and Testing, track of the Symposium on Applied Computing ACM-SAC 2014. (2014) 1279–1286
18. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
19. Maler, O., Nickovic, D., Pnueli, A.: From MITL to timed automata. In Asarin, E., Bouyer, P., eds.: Proceedings of the 4th international conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2006). Lecture Notes in Computer Science, Springer-Verlag (2006) 274–289
20. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer **1** (1997) 134–152
21. Gruhn, V., Laue, R.: Patterns for timed property specifications. Electronic Notes in Theoretical Computer Science **153** (2006) 117–133
22. Viswanathan, M., Kim, M.: Foundations for the run-time monitoring of reactive systems - Fundamentals of the MaC language. In: ICTAC: International Colloquium on Theoretical Aspects of Computing. Lecture Notes in Computer Science (2004) 543–556
23. Bielova, N., Massacci, F.: Do you really mean what you actually enforced? - edited automata revisited. Int. J. Inf. Sec. **10** (2011) 239–254
24. Sammapun, U., Lee, I., Sokolsky, O.: RT-MaC: Runtime monitoring and checking of quantitative and probabilistic properties. 2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications **0** (2005) 147–153
25. Colombo, C., Pace, G.J., Schneider, G.: Dynamic event-based runtime monitoring of real-time and contextual properties. In Cofer, D.D., Fantechi, A., eds.: Proceedings of the 13th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2008). Volume 5596 of Lecture Notes in Computer Science., Springer (2008) 135–149
26. Colombo, C., Pace, G.J., Schneider, G.: Safe runtime verification of real-time properties. In Ouaknine, J., Vaandrager, F.W., eds.: Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2009). Volume 5813 of Lecture Notes in Computer Science., Springer (2009) 103–117
27. Rinard, M.: Acceptability-oriented computing. In Crocker, R., Jr., G.L.S., eds.: Proceedings of the 2003 ACM SIGPLAN Conference On Object-Oriented Programming Sytems, Languages, and Applications Companion (OOPSLA 03 COMPANION), ACM Press (2003) 221–239