Runtime Enforcement of Regular Timed Properties

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand

Runtime Enforcement of Regular Timed Properties

Srinivas Pinisetty
INRIA Rennes - Bretagne Atlantique, France
Srinivas.Pinisetty@inria.fr

Yliès Falcone
Laboratoire d'Informatique de Grenoble
Université Grenoble I, France
Ylies.Falcone@ujf-grenoble.fr

Thierry Jéron
INRIA Rennes - Bretagne Atlantique, France
Thierry.Jeron@inria.fr

Hervé Marchand
INRIA Rennes - Bretagne Atlantique, France
Herve.Marchand@inria.fr

ABSTRACT

Runtime enforcement is a verification/validation technique aiming at correcting (possibly incorrect) executions of a system of interest. In this paper, we consider enforcement monitoring for systems with timed specifications (modeled as timed automata). We consider runtime enforcement of any regular timed property specified by a timed automaton. To ease their design and their correctness-proof, enforcement mechanisms are described at several levels: enforcement functions that specify the input-output behavior, constraints that should be satisfied by such functions, enforcement monitors that implement an enforcement function as a transition system, and enforcement algorithms that describe the implementation of enforcement monitors. The feasibility of enforcement monitoring for timed properties is validated by prototyping the synthesis of enforcement monitors.

1. INTRODUCTION

Runtime enforcement [10, 6, 7, 5, 9] is a verification/validation technique aiming at correcting (possibly incorrect) executions of a system of interest. In traditional (untimed) approaches, the enforcement mechanism is a *monitor* that is modeled as a decision procedure that inputs, corrects, and outputs a sequence of events. How a monitor transforms the input sequence is done according to a high-level specification, formalized as a property, that indicates correct and incorrect sequences. Moreover, a monitor should only output correct sequences (the monitor is sound) and should minimally alter the input sequence (the monitor is transparent). An important endeavor in the previous enforcement monitoring approaches is to determine the set of *enforceable properties* which, from an abstract point of view, consists in determining the set of properties for which an enforcement monitor can be synthesized. Several sets of enforceable properties were delineated according to the enforcement primitives conferred to enforcement monitors, e.g., safety properties for security automata [10, 6], renewal properties for edit-automata [7], response properties for generalized enforcement monitors [5]. In addition to enforcement primitives, enforceability limitations arose because properties were expressed over infinite sequences, thus imposing an enforcement monitor to consistently take decisions (on finite sequences) with the possible infinite continuations of the observed sequences. Only considering properties over finite sequences allows to get rid of the aforementioned limitations [4]: any property over finite sequences is enforceable with a monitor endowed with the primitives of an edit-automaton, basically inserting and suppressing events from the input sequence. Most of the runtime enforcement endeavors consider logical time, as opposed to physical time. Suppressing events and (later) inserting them is assumed without consequence on the execution nor on the satisfiability of the property.

In this paper, we consider *runtime enforcement of timed properties*, initially introduced in [9]. In timed properties (over finite sequences), the time that elapses between two events matters and affects the satisfiability of the property. Moreover, it turns out that considering time when specifying the behavior of systems brings some expressiveness that can be particularly useful in some application domains when, for instance, specifying usage of resources. For example, the two following properties could specify the behavior of a server.

P1 *"Resource grants and releases alternate, starting with a grant, and every grant should be released within 15 to 20 time units (t.u.)"*.

P2 *"Every 10 t.u., there should be a request for a resource followed by a grant. The request should occur within 5 t.u."*.

In this paper, we generalize and extend the initial approach to runtime enforcement of timed properties [9] in several directions. (See Section 7 for a detailed description of the improvements over [9].) First the approach in [9] tackled only safety and co-safety properties that allow to respectively express that "something bad should never happen" and that "something good should happen within a finite amount of time"; thus ruling out specifications P1 and P2. Indeed, in the space of regular properties (over timed words), many interesting properties of systems are neither safety nor co-safety properties but belongs to a larger set: the set of the so-called *response* properties that allow to specify some form of transactional behavior and cannot be considered in [9]. We propose to synthesize enforcement mechanisms for *all regular timed properties*. We consider enforcement mechanisms as *time retardants*, i.e., mechanisms should keep the same order of actions, but are allowed to increase the delay between two actions. We specify the mechanisms at several 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 implemen-

SAC'14 March 24-28, 2014, Gyeongju, Korea.

tation of enforcement monitors and serve to guide the concrete implementation of enforcement mechanisms. The difficulty that arises when considering response properties is that the aforementioned enforcement mechanisms should consider input (corrected) sequences of events that alternate between satisfying and not satisfying the underlying property.

Paper organization.

Some motivating examples for the enforcement monitoring of response properties and illustrating the enforceability issue are given in Section 2. Section 3 introduces some preliminaries and notations. Section 4 explains how properties are specified as TA with some examples. Section 6 describes the enforcement mechanisms (enforcement function, monitor, and algorithm) in detail. Related work are discussed in Section 7. Finally, conclusions and open perspectives are drawn in Section 8.

2. SOME MOTIVATING EXAMPLES

Let us consider again the two properties P1 and P2 mentioned in the introduction. We shall see in Section 4 how to formalize these specifications as properties represented by timed automata. Let gr and rel denote the grant and release events, respectively.

Let us consider P1 and illustrate how an enforcement monitor corrects the input sequence $(3, gr)\cdot(10, rel)\cdot(3, gr)\cdot(5, gr)$ (where each event is associated with a delay, indicating the time elapsed after the previous event or the system initialization for the first event). Let t be the total time since the beginning of the sequence. The monitor receives the first event gr at $t = 3$, the second event rel at $t = 13$, etc. The monitor cannot output the first received event gr because the event alone does not satisfy the property (and the monitor does not know yet the next events). If the next event is rel, then it can output the events gr followed by rel, 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 gr will be 13 t.u. If the monitor chooses the same delay for the second action rel, then the property cannot be satisfied. The monitor chooses a delay of 15 t.u. which is the minimal delay satisfying the constraint and greater than the delay in the input. When the monitor observes the second gr at $t = 16$, it will not release it as output, and again waits for the next event. Since the next input event observed at $t = 21$ is not rel, the sequence violates the property and cannot be corrected by the monitor. Hence the output of the monitor will be $(13, gr) \cdot (15, rel)$.

Let us now consider P2. Consider an input sequence $(3, req) \cdot (4, gr) \cdot (2, req) \cdot (6, gr)$. The monitor will be observing the events req followed by a grant only when $t = 7$. Hence, the delay associated with the first event in the output should be at least 7 t.u., and if the monitor chooses a delay which is greater than or equal to 7, the timing constraints cannot be satisfied. So, the output of the monitor will be empty. However, notice here that the input sequence provided to the monitor satisfies the property. Nevertheless, the monitor cannot release any event as output as it cannot take a decision until it receives a gr, which effects the delay of the first event req, thus falsifying the constraints.

In addition to the "expressiveness" of response properties, properties P1 and P2 illustre another important feature of the timed case, that we exhibit in this paper: not all properties (over finite sequences) can be enforced. For instance, we will see that property P1 is enforceable, while P2 is not. It turns out that enforcement monitors face some physical constraints when they input and memorize timed events, e.g., memorizing events takes time and influences satisfiability of the considered property. Nevertheless, the synthesis of enforcement mechanisms proposed in this paper works for all regular timed properties and the synthesized enforcement mechanisms remain sound.

3. PRELIMINARIES AND NOTATION

3.1 Untimed Languages

A (finite) word over an alphabet A is a finite sequence of elements of A. The *length* of a word w is noted $|w|$. The empty word over A is denoted by ϵ when clear from the context. The set of all (respectively non-empty) 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 w and w' is noted $w \cdot w'$. For an interval $[j, k]$ of $\mathbb{N}$, by $\bigodot_{i\in[j,k]}(a_i)$ we denote the concatenation $a_j \cdot a_{j+1} \cdots a_k$. A word w' is a prefix of a word w, noted $w' \preccurlyeq w$, whenever there exists a word w'' such that $w = w' \cdot w''$, and $w' \prec w$ whenever $w' \preccurlyeq w \wedge |w'| < |w|$. For a word w and $1 \leq i \leq |w|$, the i-th letter (resp. prefix of length i, suffix starting at position i) of w is noted $w(i)$ (respectively $w_{[\cdots i]}$, $w_{[i\cdots]}$– with the convention $w_{[\cdots 0]} = w_{[0\cdots]} \stackrel{\text{def}}{=} \epsilon$). Given a word w and two integers i, j, s.t. $i \leq j$ and $|w| \geq j$, the subword from index i to j is noted $w_{[i\cdots j]}$. The set $\mathrm{pref}(w)$ denotes the set of prefixes of w and by extension, $\mathrm{pref}(\mathcal{L}) \stackrel{\text{def}}{=} \{\mathrm{pref}(w) \mid w \in \mathcal{L}\}$ the set of prefixes of words in $\mathcal{L}$. Given an n-tuple of symbols $e = (e_1, \ldots, e_n)$, $\Pi_i(e)$ is the projection of e on its i^{th} element ($\Pi_i(e) \stackrel{\text{def}}{=} e_i$).

3.2 Timed Words and Languages

A timed word over a finite alphabet Σ is a finite sequence $\sigma = (\delta_1, a_1)\cdot(\delta_2, a_2)\cdots(\delta_n, a_n)$ of events. For an event (δ_i, a_i), $\mathrm{act}(\delta, a) \stackrel{\text{def}}{=} a$ is the action and $\mathrm{delay}(\delta_i, a_i) \stackrel{\text{def}}{=} \delta$ is the delay between a_{i-1} (or the initialization for i=1) and a_i. A *timed language* is any set $\mathcal{L} \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^*$. Note that even though the alphabet $(\mathbb{R}_{\geq 0} \times \Sigma)$ is infinite in this case, previous notions and notations defined in the untimed case (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, noted $\mathrm{time}(\sigma) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \delta_i$, is the sum of its delays.

3.3 Preliminaries to Runtime Enforcement

Orders on timed words.

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

Delaying order $\preccurlyeq_d$: For $\sigma, \sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$, we say that σ' *is a delayed prefix of* σ (noted $\sigma' \preccurlyeq_d \sigma$) iff $\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\sigma)$ and $\forall i \leq |\sigma'| : \mathrm{delay}(\sigma'(i)) \geq \mathrm{delay}(\sigma(i))$, which means that the untimed projection of σ' is a prefix of the untimed projection of σ, but delays in σ' may be greater than the delays in σ. This order will be used to characterize outputs with respect to inputs in enforcement monitoring.

Lexical order $\preceq_{\text{lex}}$: Given any two timed words σ, σ' s.t. $\Pi_\Sigma(\sigma) = \Pi_\Sigma(\sigma')$ and any two timed events with identical actions (δ, a) and (δ', a), $\preceq_{\text{lex}}$ is defined recursively 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 order is useful to choose a unique timed word among some with same actions.

Some special sets and sequences.

We consider an input timed word $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and a timed property φ defined by a timed regular language.

The *observation* of σ at time $t \in \mathbb{R}_{\geq 0}$ is the part of the input that can be read by the enforcement monitor at time t, and is defined as the maximal prefix of σ whose duration is smaller than t: $\text{obs}(\sigma, t) \stackrel{\text{def}}{=} \max_{\preccurlyeq}\{\sigma' \in (\mathbb{R}_{\geq 0} \times \Sigma)^* \mid \sigma' \preccurlyeq \sigma \wedge \text{time}(\sigma') \leq t\}$, where $\max_{\preccurlyeq}$ takes the maximal sequence according to the prefix relation $\preccurlyeq$ (unique in this case).

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

4. PROPERTIES AS TIMED AUTOMATA

4.1 Timed Automata

A timed automaton [1] (TA) is an 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 $\mathbb{R}^X_{\geq 0}$ where $\mathbb{R}^X_{\geq 0}$ denotes the valuations of clocks of X in the set $\mathbb{R}_{\geq 0}$. For $\nu \in \mathbb{R}^X_{\geq 0}$ and $\delta \in \mathbb{R}_{\geq 0}$, $\nu + \delta$ is the valuation assigning $\nu(x_i) + \delta$ to each clock x_i 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 $x_i \bowtie c$ with $x_i \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Given $g \in \mathcal{G}(X)$ and $\nu \in \mathbb{R}^X_{\geq 0}$, we write $\nu \models g$ when g holds according to ν.

Definition 1 (Timed automaton) *A* timed automaton *is a tuple $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, F\rangle$, s.t. L is a finite set of* locations*, $l_0 \in L$ is the* initial location*, X is a finite set of clocks, Σ is a finite set of* events*, $\Delta \subseteq L \times \mathcal{G}(X) \times \Sigma \times 2^X \times L$ is the* transition relation*. $F \subseteq L$ is a set of accepting locations.*

The *semantics* of a TA is a timed transition system $[\![\mathcal{A}]\!] = \langle Q, q_0, \Gamma, \rightarrow, Q_F\rangle$ where $Q = L \times \mathbb{R}^X_{\geq 0}$ 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, $Q_F = F \times \mathbb{R}^X_{\geq 0}$ 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 the maximal set of transitions of the form $(l, \nu) \xrightarrow{(\delta, a)} (l', \nu')$ with $\nu' = (\nu + \delta)[Y \leftarrow 0]$ whenever there exists $(l, g, a, Y, l') \in \Delta$ s.t. $\nu + \delta \models g$ for $\delta \in \mathbb{R}_{\geq 0}$.

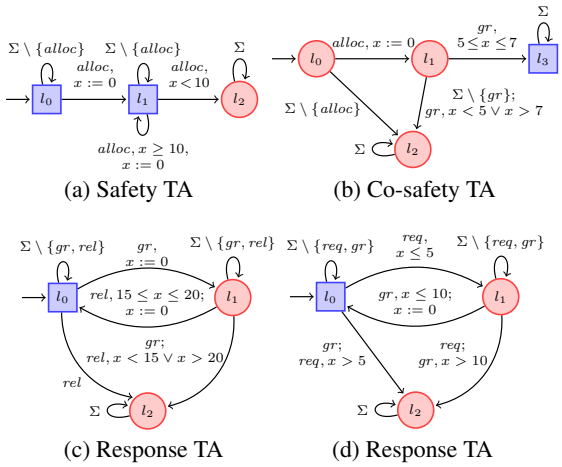
In the following, we consider a timed automaton $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, F\rangle$ with its semantics $[\![\mathcal{A}]\!]$. $\mathcal{A}$ is *deterministic* whenever for any (l, g_1, a, Y_1, l'_1) and (l, g_2, a, Y_2, l'_2) in Δ, $g_1 \wedge g_2$ is unsatisfiable. $\mathcal{A}$ is *complete* whenever for any location $l \in L$ and every event $a \in \Sigma$, 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.

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

4.2 Classes of Timed Properties

In this paper, a timed property is defined by a timed language $\varphi \subseteq (\mathbb{R}_{\geq 0} \times \Sigma)^*$ that can be recognized by a timed automaton.



(a) Safety TA (b) Co-safety TA

(c) Response TA (d) Response TA

Figure 1: Some properties as TAs

That is, we consider the set of regular timed properties. Given a timed word $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$, we say that σ satisfies φ (noted $\sigma \models \varphi$) if $\sigma \in \varphi$.

Definition 2 (Safety, co-safety and response TA) *A TA $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, F\rangle$, with semantics $[\![\mathcal{A}]\!] = \langle Q, q_0, \Gamma, \rightarrow, Q_F\rangle$ is said to be:*

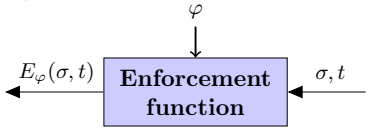
- *a* safety *TA if $q_0 \in Q_F$ and Q_F is unreachable from $Q \setminus Q_F$;*
- *a* co-safety *TA if $q_0 \in Q \setminus Q_F$ and $Q \setminus Q_F$ is unreachable from Q_F;*
- *a* response *TA if it is an unconstrained TA.*

Safety properties are the non-empty prefix-closed languages, i.e., properties φ such that $\varphi \neq \emptyset$ and if $w \preccurlyeq w'$ then $w' \models \varphi \Rightarrow w \models \varphi$. Co-safety properties are the non-universal extension-closed languages, i.e., properties φ such that $\varphi \neq (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and if $w \preccurlyeq w'$ then $w \models \varphi \Rightarrow w' \models \varphi$. Safety and co-safety properties are dual: φ is a safety property iff $(\mathbb{R}_{\geq 0} \times \Sigma)^* \setminus \varphi$ is a co-safety property. Response properties are all properties, including safety and co-safety properties. It is not hard to see that safety TAs (respectively co-safety TAs) define safety (resp. co-safety properties).

Example 1 (Properties modeled as TA's) *Figure 1 presents some properties expressed as TA's, where accepting locations are represented by squares. The safety property "The delay between consecutive allocation requests should be at least 10 t.u." is specified by the safety TA in Figure 1a. The co-safety property "An allocation request should be followed by a grant within 5 to 7 t.u." is specified by the co-safety TA in Figure 1b. The response property P1 is specified by the response TA in Figure 1c. The response property P2 is specified by the response TA in Figure 1d .*

5. CONSTRAINTS ON AN ENFORCEMENT MECHANISM

At an abstract level, an enforcement mechanism can be seen as a function with two parameters, an input timed word, and a given time, and which returns a timed word as output.



An enforcement function E_φ transforms some input timed word σ which is possibly incorrect w.r.t. φ. The resulting output $E_\varphi(\sigma, t)$ at time t is a timed word with same actions, but possibly increased delays between actions so that eventually it will satisfy the property. Our enforcement monitors are time retardants.

Definition 3 (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:*

- **Time retardant:**

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

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

- **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 of 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 events cannot be released as output, before they are read as input. 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 the property φ. In other words, no event is output before being sure that the property will be satisfied by subsequent events. Transparency **(Tr)** expresses that, at any time t, the output is a delayed prefix of the observed input $\mathrm{obs}(\sigma, t)$.

6. GENERALIZED RUNTIME ENFORCEMENT OF TIMED PROPERTIES

In this section, we present how to synthesize an enforcement monitor from any property specified by a timed automaton. Rather than giving directly the enforcement algorithm, we propose several descriptions of enforcement mechanisms: as an enforcement function, as a transition system, and finally as an algorithm derived from the transition system.

6.1 Functional Definition

The purpose of the functional definition of an enforcement mechanism is to clearly distinguish each step such as 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. The resulting functional definition presented here satisfies the physical, soundness, and transparency constraints.

Definition 4 (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:*

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

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

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

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

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

The store_φ function takes as input an observation, and outputs a pair: the first element is the transformation of a prefix of the observation for which delays have been computed (the property is satisfiable 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_φ function is defined inductively: initially, for an empty observation, both elements are empty; if σ has been observed, $\mathrm{store}_\varphi(\sigma) = (\sigma_s, \sigma_c)$, and a new event (δ, a) is observed, there are two possible cases, according to the vacuity of the set $K = \kappa_\varphi(\mathrm{time}(\sigma) + \delta, \sigma_s, \sigma_c \cdot (\delta, a))$ (the set of candidate timed words appropriately delaying $\sigma_c \cdot (\delta, a)$ and satisfying φ, see below):

- if $K \neq \emptyset$, the minimal timed word in K w.r.t the lexicographic 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 unmodified.

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 $\mathrm{time}(\sigma_s)$, while the decision to output w is taken after T t.u., thus a smaller value for $\mathrm{delay}(w(1))$ would cause this delay to be elapsed before the decision is taken.

Notice that upon reading an input event (δ, a), in case there are no appropriate delays, and if it is not possible to correct the subsequence $\sigma_c \cdot (\delta, a)$ (when κ_φ is empty), then the input event (δ, a) is appended to σ_c, and in case the subsequence $\sigma_c \cdot (\delta, a)$ can be corrected, it is appended immediately to σ_s (with appropriate delays) without relying on events which will be read later. The policy adopted is to correct the observation of the input sequence as soon as possible. Consequently, the input sequence is treated as a series of subsequences, each subsequence allowing to satisfy the property.

Proposition 1 *Given some property φ, its enforcement function E_φ as per Definition 4 satisfies the physical constraints **Phy1** and **Phy2** and is sound and transparent as per Definition 3.*

In addition to the physical constraints, soundness and transparency, the functional definition also ensures that each subsequence is released as output as soon as possible, as expressed by the following proposition:

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

Figure 2: Enforcement function evolution for P1

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

Figure 3: Enforcement function evolution for P2

Proposition 2 (Optimality of an enforcement function) *Given some property φ, its enforcement function E_φ as per Definition 4 satisfies the following optimality constraint:*

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

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 m followed by w, such that m is the maximal strict prefix of $E_\varphi(\sigma,t))$, satisfying the property φ and w is the remaining sub-sequence such that $E_\varphi(\sigma,t)) = m\cdot w$.

The last sub-sequence of the output which again makes the output to satisfy φ after m is w. The optimality constraint expresses that the sum of the delays (time required to output) of w is minimal. The delay for the events in w should be chosen such that $m\cdot w$ satisfies the property, is transparent, 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 m.

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

Example 2 (Enforcement function) *We now illustrate how the Definition 4 is applied to enforce the property P1 defined by the automaton depicted in Fig. 1c with $\Sigma = \{gr, rel\}$, and the input timed word $\sigma = (3,gr)\cdot(10,rel)\cdot(3,gr)\cdot(5,gr)$. Figure 2 shows the evolution of* obs, store_φ *and E_φ. The variable t describes global time. The resulting output is $(13,gr)\cdot(15,rel)$, which satisfies the property P1.*

Example 3 (Enforcement function: A non-enforceable property) *Let us now consider another property P2, formalized by the TA in Figure 1d, with $\Sigma = \{gr, req\}$, and the input timed word $\sigma = (3,req)\cdot(4,gr)\cdot(2,req)\cdot(6,gr)$. Figure 3 shows the evolution of* obs, store_φ *and E_φ. The variable t describes global time. The resulting output of the enforcement function is ϵ at any time instance.*

Remark 1 (Non-enforceable properties) *From Example 3, notice that the output of the enforcement function is ϵ, though the input sequence itself satisfies the property. As explained in Section 2, the delay associated with the first event should be at least 7 time units, but increasing the delay associated with the first event to 7, will 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 $<, \leq, =$ 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 subsequence may be increased, which may falsify a guard with $<, \leq, =$. However, even for such properties, the enforcement function will never produce incorrect output. If the enforcement function outputs some sequence, it will satisfy the physical, soundness and transparency constraints.

6.2 Enforcement Monitor

An enforcement function E_φ for a property φ specified by a TA $\mathcal{A}_\varphi$, is implemented by an enforcement monitor (EM), defined as a transition system $\mathcal{E}$. An EM 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 is basically a queue containing a timed word, the received actions (with increased delays) that are not released yet. 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 the (current) state ($q \in Q$) of $[\![\mathcal{A}]\!]$, a timed word $\sigma_c \in (\mathbb{R}_{\geq 0}\times\Sigma)^+$, and a real number m_t which 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.

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

where $\sigma'_c = \min_{\preceq_{\text{lex}}} \Delta(\sigma_c, m_t, q)$ with Δ defined as:

$$\begin{aligned} \Delta(\sigma_c, m_t, q) = \{ & \bigodot_{i \in [1, |\sigma_c|]} (\delta_i, \text{act}(\sigma_c(i))) \mid \\ & \forall i \leq |\sigma_c|, \delta_i \geq \text{delay}(\sigma(i)) \wedge \delta_1 \geq m_t \\ & \wedge \exists q_1 \in Q_F : q \overset{\bigodot_{i \in [1, |\sigma_c|]} (\delta_i, \text{act}(\sigma_c(i)))}{\rightarrow} q_1 \} \end{aligned}$$

Δ is the set of timed words σ'_c 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 should be greater than or equal to m_t, and an accepting state is reachable from state q upon the sequence σ'_c.

- In the first case, there are no good delays such that an accepting state is reachable from the state q upon with a sequence delaying σ_c ($\Delta = \emptyset$). 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 Δ, chosen according to the lexicographic order; and a Boolean value `tt`, indicating that an accepting state is reachable.

Definition 5 (Enforcement Monitor) *An enforcement monitor $\mathcal{E}$ for φ is a transition system $(C, C_0, \Gamma, \hookrightarrow)$ s.t.:*

- *$C = (\mathbb{R}_{\geq 0} \times \Sigma)^* \times (\mathbb{R}_{\geq 0} \times \Sigma)^* \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times Q$ is the set of configurations,*
- *$C_0 = \langle \varepsilon, \varepsilon, 0, 0, 0, q_0 \rangle \in C$ 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 are triples comprised of an optional input event, an operation, and an optional output event, where the set of possible operations is $Op = \{\text{store-}\overline{\varphi}(\cdot), \text{store-}\varphi(\cdot), \text{dump}(\cdot), \text{idle}(\cdot)\}$;*
- *$\hookrightarrow \subseteq C \times \Gamma_{\mathcal{E}} \times C$ is the transition relation defined as the smallest relation obtained by the following rules applied with the priority order below:*
  - **1. *store-$\overline{\varphi}$:***
    $(\sigma_s, \sigma_c, \delta, d, m_t, q) \overset{(\delta,a)/\text{store}-\overline{\varphi}(\delta,a)/\epsilon}{\hookrightarrow} (\sigma_s, \sigma_c \cdot (\delta, a), 0, d, m'_t, q)$
    if $\Pi_2(\text{update}(q, \sigma_c \cdot (\delta, a), m_t + \delta)) = \texttt{ff}$
    - $m'_t = m_t + \delta$
  - **2. *store-φ:***
    $(\sigma_s, \sigma_c, \delta, d, m_t, q) \overset{(\delta,a)/\text{store}-\varphi(\delta,a)/\epsilon}{\hookrightarrow} (\sigma_s \cdot \sigma'_c, \epsilon, 0, d, m'_t, q')$
    if $(\text{update}(q, \sigma_c \cdot (\delta, a), m_t + \delta)) = (\sigma'_c, \texttt{tt})$*, where:*
    - $m'_t = m'_t + \delta - \text{time}(\sigma'_c)$
    - *q' is defined as $q \overset{\sigma'_c}{\rightarrow} q'$*
  - **3. *dump*:**
    $((\delta, a) \cdot \sigma_s, \sigma_c, s, \delta, m_t, q) \overset{\epsilon/\text{dump}(\delta,a)/(\delta,a)}{\hookrightarrow} (\sigma_s, \sigma_c, s, 0, m_t, q)$
  - **4. *idle*:**
    $(\sigma_s, \sigma_c, s, d, m_t, q) \overset{\epsilon/\text{idle}(\delta)/\epsilon}{\hookrightarrow} (\sigma_s, \sigma_c, s + \delta, d + \delta, m_t, q).$

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 which 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 $[\![\mathcal{A}]\!]$ 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, $\text{time}(\sigma'_c)$, 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 no store nor dump operation is possible.

Example 4 (Execution of an enforcement monitor) *We now illustrate how the rules of Definition 5 are applied to enforce the property P1. Let us consider the input timed word $\sigma = (3, gr) \cdot (10, rel) \cdot (3, gr) \cdot (5, gr)$. Figure 4 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). The variable t describes global time. The resulting output is $(13, gr) \cdot (15, rel)$, which satisfies the property P1. From $t = 28$, only the* delay *rule can be applied.*

Remark 2 (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.*

Remark 3 (Relation between enforcement function and monitor) *An enforcement monitor is an operational description of an enforcement function. For any property φ, input σ, and at any time t, the input sequence processed by the enforcement function* $\text{obs}(\sigma, t)$ *is the concatenation of all events read by EM (store) over various steps, until time t. At any time t, the output of the enforcement function E_φ, is equal to the output behavior of the associated EM (the concatenation of all the released events (dump) until time t). Since, E_φ is sound, transparent, and satisfies the physical constraints,*

$t = 0$ $\quad \epsilon/(\epsilon, \epsilon, 0, 0, 0, (l_0, 0))/(3, gr) \cdot (10, rel) \cdot (3, gr) \cdot (5, gr)$

$\downarrow$idle(3)

$t = 3$ $\quad \epsilon/(\epsilon, \epsilon, 3, 3, 0, (l_0, 0))/(3, gr) \cdot (10, rel) \cdot (3, gr) \cdot (5, gr)$

$\downarrow$store-$\overline{\varphi}$

$t = 3$ $\quad \epsilon/(\epsilon, (3, gr), 0, 3, 3, (l_0, 0))/(10, rel) \cdot (3, gr) \cdot (5, gr)$

$\downarrow$idle(10)

$t = 13$ $\quad \epsilon/(\epsilon, (3, gr), 0, 13, 3, (l_0, 0))/(10, rel) \cdot (3, gr) \cdot (5, gr)$

$\downarrow$store-φ

$t = 13$ $\quad \epsilon/((13, gr) \cdot (15, rel), \epsilon, 0, 13, -15, (l_0, 15))/(3, gr) \cdot (5, gr)$

$\downarrow$dump

$t = 13$ $\quad (13, gr)/((15, rel), \epsilon, 0, 0, -15, (l_0, 15))/(3, gr) \cdot (5, gr)$

$\downarrow$idle(3)

$t = 16$ $\quad (13, gr)/((15, rel), \epsilon, 3, 3, -15, (l_0, 15))/(3, gr) \cdot (5, gr)$

$\downarrow$store-$\overline{\varphi}$

$t = 16$ $\quad (13, gr)/((15, rel), (3, gr), 0, 3, -12, (l_0, 15))/(5, gr)$

$\downarrow$idle(5)

$t = 21$ $\quad (13, gr)/((15, rel), (3, gr), 5, 8, -12, (l_0, 15))/(5, gr)$

$\downarrow$store-$\overline{\varphi}$

$t = 21$ $\quad (13, gr)/((15, rel), (3, gr) \cdot (5, gr), 0, 8, -7, (l_0, 15))/\epsilon$

$\downarrow$idle(7)

$t = 28$ $\quad (13, gr)/((15, rel), (3, gr) \cdot (5, gr), 7, 15, -7, (l_0, 15))/\epsilon$

$\downarrow$dump

$t = 28$ $\quad (13, gr) \cdot (15, rel))/(\epsilon, (3, gr) \cdot (5, gr), 7, 0, -7, (l_0, 15))/\epsilon$

Figure 4: Execution of an enforcement monitor

with relating the input-output behavior of an EM to E_φ, it can be proved that EM is sound, transparent and satisfies the physical constraints.

6.3 Implementation

The implementation of an EM consists of two processes running concurrently (Store and Dump) as shown in Figure 5, 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, which is determined by the process itself.

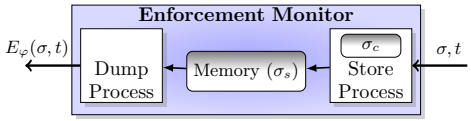


Figure 5: Realizing an EM

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 property automaton, 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 property automaton (l, ν), a timed word, and computes the state reached by the property automaton.

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
```

The algorithm proceeds as follows. The StoreProcess initially waits for an input event. After receiving an event as input, it is appended to the internal buffer σ_c, with the corresponding delay, and the delay corresponding to the received event 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 $isPath = \texttt{tt}$), 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.

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
```

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 dump process 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 (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.

7. RELATED WORK

Several approaches for the runtime enforcement of properties are related to the one proposed in this paper.

Most of the work on this topic was dedicated to the enforcement of untimed properties (see [3] for a short overview). Schneider introduced security automata as the first runtime mechanism for enforcing safety properties [10]. 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 [11]: 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. [7] 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 proposed that are able to enforce the set of response properties in the safety-progress classification of (untimed) properties. Moreover, some variants of edit-automata differ in how they ensure the transparency constraints (see e.g., [2]). However, none of these approaches is able to synthesize an enforcement mechanism from a property.

In previous work in [9] we introduced the problem of runtime enforcement for timed properties. We similarly proposed several notions of enforcement mechanisms: enforcement function, enforcement monitor, and enforcement algorithm. Only safety and co-safety properties were considered and different definitions of mechanisms were proposed for each of the classes. In this paper, given any timed automaton (representing any regular property), we synthesize enforcement functions, monitors and algorithms according to one general definition and for more properties: the set of regular response properties. Moreover, when the input timed automaton recognizes a safety or a co-safety property, we could optimize the data-structures used by the enforcement monitors and algorithms. Finally, for the enforcement of co-safety properties, the approach in [9] assumes that time elapses differently for input and output sequences (the sequences are desynchronized). More precisely, the delay of the first event of the output sequence is computed from the moment an enforcement mechanism detects that its input sequence can be corrected (that is, the mechanisms has read a sequence that can be delayed into a correct sequence). Our approach is more realistic as it does not suffer from this "shift" problem.

Finally, Matteucci proposed to synthesize controller operations to enforce safety and information-flow properties using process-algebra [8]. The approach targets discrete-time properties and systems are modelled as timed processes expressed in CCS. Monitors are described at an abstract level and resemble Schneider's security automata: monitors only halt the underlying program and do not consider timing issues.

8. CONCLUSION AND FUTURE WORK

Conclusion.

This paper presented a general enforcement monitoring framework for systems with timing requirements. We generalized the results of [9], showed how to synthesize enforcement mechanisms for any regular timed property (modeled with a timed automaton). Enforcement mechanisms are described at several levels of abstraction (enforcement function, monitor, and algorithm), thus facilitating the design and implementation of such mechanisms.

Future work.

Several avenues for future work are open 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. Note that even for properties which are non-enforceable, enforcement monitors can be built, which may not be able to correct some input sequences, but outputs are always sound.

Specifications are currently modeled 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 in a particular application domain.

Alternative enforcement primitives can be afforded to timed retardant. For instance, we could relax the constraint of only augmenting delays of events. For instance, time retardants that delay the current observation of the input (while being allowed to shorten the delay of some events) have yet to be studied. We believe that suppressing events also can be considered, by erasing some events which are stored in the memory.

A prototype is implemented based on the algorithms described in Section 6.3. Regarding efficiency, some experiments should be done using the prototype to determine to what extent using more specific enforcement mechanisms for a sub-class (e.g., the safety class), can give better performance, compared to using the general enforcement monitoring mechanism.

9. REFERENCES

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Comp. Science*, 126:183–235, 1994.

[2] N. Bielova and F. Massacci. Do you really mean what you actually enforced? - edited automata revisited. *Int. J. Inf. Sec.*, 10(4):239–254, 2011.

[3] Y. Falcone. You should better enforce than verify. In *Runtime Verification*, LNCS, pages 89–105, 2010.

[4] Y. Falcone, J.-C. Fernandez, and L. Mounier. What can you verify and enforce at runtime? *STTT*, 14(3):349–382, 2012.

[5] Y. Falcone, L. Mounier, J.-C. Fernandez, and J.-L. Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *FMSD*, 38(3):223–262, 2011.

[6] K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *ACM Trans. Program. Lang. Syst.*, 28(1):175–205, 2006.

[7] J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12(3):19:1–19:41, 2009.

[8] I. Matteucci. Automated synthesis of enforcing mechanisms for security properties in a timed setting. *Electr. Notes Theor. Comput. Sci.*, 186:101–120, 2007.

[9] S. Pinisetty, Y. Falcone, T. Jéron, H. Marchand, A. Rollet, and O. L. N. Timo. Runtime enforcement of timed properties. In *Runtime Verification*, LNCS, 2012.

[10] F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[11] M. Viswanathan and M. Kim. Foundations for the run-time monitoring of reactive systems - Fundamentals of the MaC language. In *ICTAC*, LNCS, pages 543–556, 2004.