# Enforcement of (Timed) Properties with Uncontrollable Events

Matthieu Renard, Yliès Falcone, Antoine Rollet, Srinivas Pinisetty, Thierry Jéron, Hervé Marchand

# Enforcement of (Timed) Properties with Uncontrollable Events

Matthieu Renard[1], Yliès Falcone[2], Antoine Rollet[1], Srinivas Pinisetty[3], Thierry Jéron[4], and Hervé Marchand[4]

[1] LaBRI, Bordeaux INP, Bordeaux, France, First.Last@labri.fr
[2] Univ. Grenoble Alpes, Inria, LIG, F-38000 Grenoble, France, ylies.falcone@imag.fr
[3] Aalto University, Finland, srinivas.pinisetty@aalto.fi
[4] Inria Rennes Bretagne-Atlantique, Rennes, France, First.Last@inria.fr

**Abstract.** This paper deals with runtime enforcement of untimed and timed properties with uncontrollable events. Runtime enforcement consists in modifying the executions of a running system to ensure their correctness with respect to a desired property. We introduce a framework that takes as input any regular (timed) property over an alphabet of events, with some of these events being uncontrollable. An uncontrollable event cannot be delayed nor intercepted by an enforcement mechanism. Enforcement mechanisms satisfy important properties, namely soundness and compliance - meaning that enforcement mechanisms output correct executions that are close to the input execution. We discuss the conditions for a property to be enforceable with uncontrollable events, and we define enforcement mechanisms that modify executions to obtain a correct output, as soon as possible. Moreover, we synthesize sound and compliant descriptions of runtime enforcement mechanisms at two levels of abstraction to facilitate their design and implementation.

## 1 Introduction

Verifying a user-provided specification at runtime consists in running a mechanism that assigns verdicts to a sequence of events produced by an instrumented system w.r.t. a property formalizing the specification. This paper focuses on *runtime enforcement* (cf. [21,15,12,13,3]) which goes beyond pure verification at runtime and studies how to react to a violation of specifications. In runtime enforcement, an enforcement mechanism (EM) takes a (possibly incorrect) execution sequence as input, and outputs a new sequence. Enforcement mechanisms should be *sound* and *transparent*, meaning that the output should satisfy the property under consideration and should be as close as possible to the input. When dealing with timed properties, EMs can act as *delayers* over the input sequence of events [19,17,18]. That is, EMs buffer input events for some time and then release them in such a way that the output sequence of events satisfies the property.

*Motivations.* In this paper, we focus on online enforcement of properties with uncontrollable events. Introducing uncontrollable events is a step towards more realistic runtime enforcement. As a matter of fact, uncontrollable events naturally occur in many

applications scenarios where the EM has no control over certain input events. For instance, certain events from the environment may be out of the scope of the mechanism at hand. This situation arises for instance in avionic systems where a command of the pilot has consequences on a specific component. In this domain, it is usual to add control mechanisms to check the validity of an event on particular points according to observations. For instance, the "spoiler activation"[5] command decided by the pilot is sent by the panel to a control flight system, and this leads finally to a specific event on the spoilers. Placing an EM directly on the spoiler permits to avoid incoherent events, according to the pilot commands (which are events out of the scope of the EM). In the timed setting, uncontrollable events may be urgent messages that cannot be delayed by an enforcement mechanism. Similarly, when a data-dependency exists between two events (e.g., between a *write* event that displays a value obtained from a previous *read* event), the first *read* event is somehow uncontrollable as it cannot be delayed by the enforcement mechanism without preventing the *write* event to occur in the monitored program.

*Challenges.* Considering uncontrollable events in the timed setting induces new challenges. Indeed, enforcement mechanisms may now receive events that cannot be buffered and have to be released immediately in output. Since they influence the satisfaction of the property under scrutiny, delays of controllable events stored in memory have to be recomputed upon each uncontrollable event. Moreover, the occurrence of such events has to be anticipated, meaning that all possible sequences of uncontrollable events have to be considered by the enforcement mechanism. Thus, new enforcement strategies are necessary for both untimed and timed properties.

*Contributions.* We introduce a framework for runtime enforcement of regular untimed and timed properties with uncontrollable events. It turns out that the usual notion of transparency has to be weakened. As we shall see, the initial order between uncontrollable and controllable events can change in output, contrary to what is prescribed by transparency. Thus, we propose to replace transparency with a new notion, namely *compliance*, ensuring that the order of controllable events is maintained while uncontrollable events are output as soon as they are received. We define a property to be enforceable with uncontrollable events when it is possible to obtain a sound and compliant enforcement mechanism for any input sequence. It turns out that a property may not be enforceable because of certain input sequences. Intuitively, enforceability issues arise because some sequences of uncontrollable events that lead the property to be violated cannot be avoided. We give a condition, represented by a property, that indicates whether soundness is guaranteed by the enforcement monitor or not, depending on the input given so far. We describe enforcement mechanisms at two levels of abstraction. The synthesized enforcement mechanisms are sound and compliant whenever the previously mentioned condition holds.

*Outline.* Section 2 introduces preliminaries and notations. Sections 3 and 4 present the enforcement framework with uncontrollable events in the untimed and timed settings, respectively, where enforcement mechanisms are defined at two levels of abstraction. Section 5 discusses related work. Section 6 presents conclusions and perspectives.

[5] The spoiler is a device used to reduce the lift of an aircraft.

## 2 Preliminaries and Notation

*Untimed notions.* An *alphabet* is a finite set of symbols. A *word* over an alphabet $\Sigma$ is a sequence over $\Sigma$. The set of finite words over $\Sigma$ is denoted $\Sigma^*$. The *length* of a finite word $w$ is noted $|w|$, and the *empty word* is noted $\epsilon$. $\Sigma^+$ stands for $\Sigma^* \setminus \{\epsilon\}$. A *language* over $\Sigma$ is any subset $\mathcal{L} \subseteq \Sigma^*$. The concatenation of two words $w$ and $w'$ is noted $w.w'$ (the dot could sometimes be omitted). A word $w'$ is a *prefix* of a word $w$, noted $w' \preccurlyeq w$ if there exists a word $w''$ such that $w = w'.w''$. The word $w''$ is called the *residual* of $w$ after reading the prefix $w'$, noted $w'' = w'^{-1}.w$. The word $w''$ is then called a *suffix* of $w$. Note that $w'.w'' = w'.w'^{-1}.w = w$. These standard definitions are extended to languages in the natural way. Given a word $w$ and an integer $i$ such that $1 \leq i \leq |w|$, we note $w(i)$ the $i$-th element of $w$. Given a tuple $e = (e_1, e_2, \ldots, e_n)$ of size $n$, for an integer $i$ such that $1 \leq i \leq n$, we note $\Pi_i$ the projection on the $i$-th coordinate, i.e. $\Pi_i(e) = e_i$. Given a word $w \in \Sigma^*$ and $\Sigma' \subseteq \Sigma$, we define the *restriction* of $w$ to $\Sigma'$, noted $w_{|\Sigma'}$, as the word $w' \in \Sigma'$ whose letters are the letters of $w$ belonging to $\Sigma'$ in the same order. Formally, $\epsilon_{|\Sigma'} = \epsilon$ and $\forall \sigma \in \Sigma^*, \forall a \in \Sigma$, $(w.a)_{|\Sigma'} = w_{|\Sigma'}.a$ if $a \in \Sigma'$, and $(w.a)_{|\Sigma'} = w_{|\Sigma'}$ otherwise.

*Automata.* An *automaton* is a tuple $\langle Q, q_0, \Sigma, \rightarrow, F \rangle$, where $Q$ is the set of states, called *locations*, $q_0 \in Q$ is the initial location, $\Sigma$ is the alphabet, $\rightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation and $F \subseteq Q$ is the set of accepting locations. Any location in $F$ is called accepting. Whenever there exists $(q, a, q') \in \rightarrow$, we note it $q \xrightarrow{a} q'$. Relation $\rightarrow$ is extended to words $\sigma \in \Sigma^*$ by noting $q \xrightarrow{\sigma.a} q'$ whenever there exists $q''$ such that $q \xrightarrow{\sigma} q''$ and $q'' \xrightarrow{a} q'$. Moreover, for $q \in Q$, $q \xrightarrow{\epsilon} q$ always holds. An automaton $\mathcal{A} = \langle Q, q_0, \Sigma, \rightarrow, F \rangle$ is *deterministic* if $\forall q \in Q, \forall a \in \Sigma, (q \xrightarrow{a} q' \wedge q \xrightarrow{a} q'') \implies q' = q''$. $\mathcal{A}$ is *complete* if $\forall q \in Q, \forall a \in \Sigma, \exists q' \in Q, q \xrightarrow{a} q'$. A word $w$ is *accepted* by $\mathcal{A}$ if there exists $q \in F$ such that $q_0 \xrightarrow{w} q$. The language (i.e. set of all words) accepted by $\mathcal{A}$ is noted $\mathcal{L}(\mathcal{A})$. A *property* is a language over an alphabet $\Sigma$. A regular property is a language accepted by an automaton. In the sequel, we shall assume that a property $\varphi$ is represented by a deterministic and complete automaton $\mathcal{A}_\varphi$.

*Timed languages.* Let $\mathbb{R}_{\geq 0}$ be the set of non-negative real numbers, and $\Sigma$ a finite alphabet of actions. An event is a pair $(t, a) \in \mathbb{R}_{\geq 0} \times \Sigma$. We define $\mathrm{date}(t, a) = t$ and $\mathrm{act}(t, a) = a$ the projections of events on dates and actions respectively. A *timed word* over $\Sigma$ is a word over $\mathbb{R}_{\geq 0} \times \Sigma$ whose real parts are ascending, i.e. $\sigma$ is a timed word if $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and $\forall i \in [1; |\sigma| - 1], \mathrm{date}(w(i)) \leq \mathrm{date}(w(i+1))$. $\mathrm{tw}(\Sigma)$ denotes the set of timed words over $\Sigma$. For a timed word $\sigma = (t_1, a_1).(t_2, a_2) \ldots (t_n, a_n)$ and an integer $i$ such that $1 \leq i \leq |\sigma|$, $t_i$ is the time elapsed before action $a_i$ occurs. We naturally extend the notions of *prefix* and *residual* to timed words. We note $\mathrm{time}(\sigma) = \mathrm{date}(\sigma(|\sigma|))$, and define the *observation* of $\sigma$ at time $t$ as the timed word $\mathrm{obs}(\sigma, t) = \max_{\preccurlyeq}(\{\sigma' \mid \sigma' \preccurlyeq \sigma \wedge \mathrm{time}(\sigma') \leq t\})$. We also define the remainder of the observation of $\sigma$ at time $t$ as $\mathrm{nobs}(\sigma, t) = (\mathrm{obs}(\sigma, t))^{-1}.\sigma$. The *untimed projection* of $\sigma$ is $\Pi_\Sigma(\sigma) = a_1.a_2 \ldots a_n$, it is the sequence of actions of $\sigma$ with dates ignored. $\sigma$ *delayed by* $t \in \mathbb{R}_{\geq 0}$ is the word noted $\sigma +_{\mathrm{t}} t$ such that $t$ is added to all dates: $\sigma +_{\mathrm{t}} t = (t_1 + t, a_1).(t_2 + t, a_2) \ldots (t_{|\sigma|} + t, a_{|\sigma|})$. We also extend the definition of the restriction of $\sigma$ to $\Sigma' \subseteq \Sigma$ to timed words, such that $\epsilon_{|\Sigma'} = \epsilon$, and for $\sigma \in \mathrm{tw}(\Sigma)$

and $(t,a)$ such that $\sigma.(t,a) \in \mathrm{tw}(\Sigma)$, $(\sigma.(t,a))_{|\Sigma'} = \sigma_{|\Sigma'}.(t,a)$ if $a \in \Sigma'$, and $(\sigma.(t,a))_{|\Sigma'} = \sigma_{|\Sigma'}$ otherwise. A *timed language* is any subset of $\mathrm{tw}(\Sigma)$. Moreover, we define an order on timed words: we say that $\sigma'$ *delays* $\sigma$, noted $\sigma \preccurlyeq_d \sigma'$, whenever $\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\sigma)$ and $\forall i \in [1; |\sigma|-1], \mathrm{date}(\sigma(i)) \leq \mathrm{date}(\sigma'(i))$. Note that the order is not the same in the different constraints: $\sigma'$ is a prefix of $\sigma$, but dates in $\sigma'$ exceed dates in $\sigma$. We also define a *lexical order* $\preccurlyeq_{\mathrm{lex}}$ on timed words with identical untimed projections, such that $\epsilon \preccurlyeq_{\mathrm{lex}} \epsilon$, and for two words $\sigma$ and $\sigma'$ such that $\Pi_\Sigma(\sigma) = \Pi_\Sigma(\sigma')$, and two events $(t,a)$ and $(t',a)$, $(t',a).\sigma' \preccurlyeq_{\mathrm{lex}} (t,a).\sigma$ if $t' < t \vee (t = t' \wedge \sigma' \preccurlyeq_{\mathrm{lex}} \sigma)$.

Consider for example the timed word $\sigma = (1,a).(2,b).(3,c).(4,a)$ over the alphabet $\Sigma = \{a,b,c\}$. Then, $\Pi_\Sigma(\sigma) = a.b.c.a$, $\mathrm{obs}(\sigma,3) = (1,a).(2,b).(3,c)$, $\mathrm{nobs}(\sigma,3) = (4,a)$, and if $\Sigma' = \{b,c\}$, $\sigma_{|\Sigma'} = (2,b).(3,c)$, and for instance $\sigma \preccurlyeq_d (1,a).(2,b).(4,c)$, and $\sigma \preccurlyeq_{\mathrm{lex}} (1,a).(3,b).(3,c).(3,a)$.

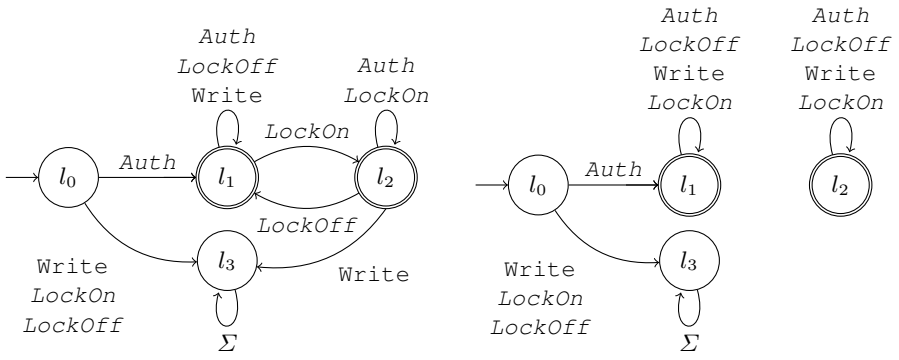
*Timed automata.* Let $X = \{X_1, X_2, \ldots, X_n\}$ be a finite set of *clocks*. A *clock valuation* is a function $\nu$ from $X$ to $\mathbb{R}_{\geq 0}$. The set of clock valuations for the set of clocks $X$ is noted $\mathcal{V}(X)$, i.e., $\mathcal{V}(X) = \{\nu \mid \nu : X \to \mathbb{R}_{\geq 0}\}$. We consider the following operations on valuations: for any valuation $\nu$, $\nu + \delta$ is the valuation assigning $\nu(X_i) + \delta$ to every clock $X_i \in X$; for any subset $X' \subseteq X$, $\nu[X' \leftarrow 0]$ is the valuation assigning 0 to each clock in $X'$, and $\nu(X_i)$ to any other clock $X_i$ not in $X'$. $\mathcal{G}(X)$ denotes the set of guards consisting of 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 a valuation $\nu$, we write $\nu \models g$ when for every simple constraint $X_i \bowtie c$ in $g$, $\nu(X_i) \bowtie c \equiv true$.

**Definition 1 (Timed automaton [1]).** *A timed automaton (TA) is a tuple $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, G\rangle$, such that $L$ is a set of locations, $l_0 \in L$ is the initial location, $X$ is a set of clocks, $\Sigma$ is a finite set of events, $\Delta \subseteq L \times \mathcal{G}(X) \times \Sigma \times 2^X \times L$ is the transition relation, and $G \subseteq L$ is a set of accepting locations. A transition $(l, g, a, X', l') \in \Delta$ is a transition from $l$ to $l'$, labelled with event $a$, with guard defined by $g$, and with the clocks in $X'$ to be reset.*

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

A timed automaton $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, G\rangle$ is *deterministic* if for any $(l, g_1, a, Y_1, l'_1)$ and $(l, g_2, a, Y_2, l'_2)$ in $\Delta$, $g_1 \wedge g_2$ is unsatisfiable, meaning that only one transition can be fired at any time. $\mathcal{A}$ is *complete* if for any $l \in L$ and any $a \in \Sigma$, the disjunction of the guards of all the transitions leaving $l$ and labeled by $a$ is valid (i.e., it evaluates to true for any clock valuation).

A *run* $\rho$ from $q \in Q$ is a valid sequence of transitions in $[\![\mathcal{A}]\!]$ starting from $q$, of the form $\rho = q \xrightarrow{(\delta_1,a_1)} q_1 \xrightarrow{(\delta_2,a_2)} q_2 \ldots \xrightarrow{(\delta_n,a_n)} q_n$. The set of runs from $q_0$ is noted $\mathrm{Run}(\mathcal{A})$ and $\mathrm{Run}_{F_G}(\mathcal{A})$ denotes the subset of runs accepted by $\mathcal{A}$, i.e. ending in a state in $F_G$. The *trace* of the run $\rho$ previously defined is the timed word




(a) Property $\varphi_{\text{ex}}$ modeling writings on a shared storage device

(b) $\text{Pre}(\varphi_{\text{ex}})$

Fig. 1: A property and its corresponding precondition property of enforceability

$(t_1, a_1).(t_2, a_2) \dots (t_n, a_n)$, with, for $1 \leq i \leq n$, $t_i = \sum_{k=1}^{i} \delta_k$. Thus, given the trace $\sigma = (t_1, a_1).(t_2, a_2) \dots (t_n, a_n)$ of a run $\rho$ from a state $q \in Q$ to $q' \in Q$, we can define $w = (\delta_1, a_1).(\delta_2, a_2) \dots (\delta_n, a_n)$, with $\delta_1 = t_1$, and $\forall i \in [2; n], \delta_i = t_i - t_{i-1}$, and then $q \xrightarrow{w} q'$. To ease the notation, we will only consider traces and note $q \xrightarrow{\sigma} q'$ whenever $q \xrightarrow{w} q'$ for the previously defined $w$. Note that to concatenate two traces $\sigma_1$ and $\sigma_2$, it is needed to delay $\sigma_2$: the concatenation $\sigma$ of $\sigma_1$ and $\sigma_2$ is the trace defined as $\sigma = \sigma_1.(\sigma_2 +_{\text{t}} \text{time}(\sigma_1))$. Thus, if $q \xrightarrow{\sigma_1} q' \xrightarrow{\sigma_2} q''$, then $q \xrightarrow{\sigma} q''$.

*Timed properties.* A *regular timed property* is a timed language $\varphi \subseteq \text{tw}(\Sigma)$ that is accepted by a timed automaton. For a timed word $\sigma$, we say that $\sigma$ *satisfies* $\varphi$, noted $\sigma \models \varphi$ whenever $\sigma \in \varphi$. A *regular timed property* is a timed language accepted by a timed automaton. We only consider deterministic and complete regular timed properties.

Given an automaton $\mathcal{A}$ such that $Q$ is the set of states of $[\![\mathcal{A}]\!]$ and $\rightarrow$ its transition relation, and a word $\sigma$, we note $q \,\text{after}\, \sigma = \{q' \in Q \mid q \xrightarrow{\sigma} q'\}$ for $q \in Q$. We note $\text{Reach}(\sigma) = q_0 \,\text{after}\, \sigma$. These definitions are valid in both the untimed and timed cases. We extend these definitions to languages: if $L$ is a language, $q \,\text{after}\, L = \bigcup_{\sigma \in L} q \,\text{after}\, \sigma$ and $\text{Reach}(L) = q_0 \,\text{after}\, L$.

## 3 Enforcement Monitoring of Untimed Properties

In this section, $\varphi$ is a regular property defined by a complete and deterministic automaton $\mathcal{A}_\varphi = \langle Q, q_0, \Sigma, \rightarrow, F \rangle$. The purpose of an *enforcement mechanism* (EM) for $\varphi$ is to modify the executions of a running system, represented by words so as to satisfy $\varphi$. It takes as input a word, representing an execution, and outputs a word, i.e. an execution. We consider uncontrollable events in the set $\Sigma_u \subseteq \Sigma$. These events cannot be modified by the EM, i.e. they cannot be suppressed nor buffered, so they must be emitted

by the EM whenever they are received. Let us note $\Sigma_c = \Sigma \setminus \Sigma_u$ the set of controllable events, which are on the scope of the EM. The EM can decide to buffer them to delay their emission, but it cannot suppress them (nevertheless, it can delay them endlessly, keeping their order unchanged). Thus, the EM may interleave controllable and uncontrollable events.

### 3.1 Enforcement Functions and their Requirements

An enforcement function is a description of the input/output behavior of an EM. Formally, we define *enforcement functions* as follows:

**Definition 2 (Enforcement function).** *Given an alphabet of actions $\Sigma$, an enforcement function is a function $E : \Sigma^* \to \Sigma^*$, i.e. a function that modifies an execution.*

As stated previously, the usual purpose of an EM is to ensure that the executions of a running system satisfy a property, thus its enforcement function has to be *sound*, meaning that its output always satisfies the property:

**Definition 3 (Soundness).** *$E$ is sound with respect to $\varphi$ if $\forall \sigma \in \Sigma^*, E(\sigma) \models \varphi$.*

The usual notion of *transparency* in enforcement monitoring [21,15] states that the output of an enforcement function is the longest prefix of the input satisfying the property. The name "transparency" stems from the fact that correct executions are left unchanged. However, because of uncontrollable events, events may be released in a different order from the one they are received. Therefore, transparency can not be ensured, and we define the weaker notion of *compliance*.

**Definition 4 (Compliance).** *$E$ is compliant w.r.t. $\Sigma_u$ and $\Sigma_c$, noted* $\mathrm{compliant}(E, \Sigma_c, \Sigma_u)$*, if $\forall \sigma \in \Sigma^*$, $E(\sigma)_{|\Sigma_c} \preccurlyeq \sigma_{|\Sigma_c} \wedge \forall u \in \Sigma_u, E(\sigma).u \preccurlyeq E(\sigma.u)$.*

Intuitively, compliance states that the EM does not change the order of the controllable events and emits uncontrollable events immediately upon their reception, possibly followed by stored controllable events. When clear from the context, the partition is not mentioned: $E$ is said to be compliant.

We say that a property is *enforceable* whenever there exists a compliant function that is sound with respect to that property.

*Example 1.* Figure 1a depicts property $\varphi_{\text{ex}}$ which states that writing to a shared storage device should be authenticated and is authorized only when the device is not locked. $\varphi_{\text{ex}}$ is not enforceable if the uncontrollable alphabet is $\{\texttt{LockOn}, \texttt{LockOff}\}$[6] since reading the word $\texttt{LockOn}$ from $l_0$ leads to $l_3$, which is not an accepting location. However, the existence of such a word does not imply that it is impossible to enforce this property for some other input words. If word $\texttt{Auth}$ is read, then location $l_1$ is reached, and from this location, it is possible to enforce $\varphi_{\text{ex}}$ by emitting $\texttt{Write}$ only when in location $l_1$.

[6] Uncontrollable events are emphasized in italics.

### 3.2 Synthesizing Enforcement Functions

Example 1 shows that some input words cannot be corrected by the EM, because of uncontrollable events. This leads us to define another property that captures the sequences that can be input to an EM while ensuring soundness.

**Definition 5** **($\mathrm{Pre}, \mathrm{Enf}$).**

- $\mathrm{Pre}(\varphi) = \langle Q, q_0, \Sigma, \rightarrow', \mathrm{Q}_{\mathrm{enf}} \rangle$, *with*

$$\rightarrow' = (\rightarrow \cap\, \mathrm{Q}_{\overline{\mathrm{enf}}} \times \Sigma \times Q) \cup \{(q, a, q) \mid q \in \mathrm{Q}_{\mathrm{enf}} \wedge a \in \Sigma\}$$

- $\mathrm{Enf}(\varphi) = \langle Q, q_0, \Sigma, \rightarrow, \mathrm{Q}_{\mathrm{enf}} \rangle$.

*where* $\mathrm{Q}_{\mathrm{enf}} = \{q \in F \mid (q \,\mathrm{after}\, \Sigma_u^*) \subseteq F\}$, $\mathrm{Q}_{\overline{\mathrm{enf}}} = Q \setminus \mathrm{Q}_{\mathrm{enf}}$.

$\mathrm{Q}_{\mathrm{enf}}$ is the set of accepting locations of $\mathcal{A}_\varphi$ from which it is impossible to reach a non-accepting location by reading only uncontrollable events, and thus possible to enforce the property (since it is possible to indefinitely delay all controllable events to ensure the property).

Intuitively, $\mathrm{Pre}(\varphi)$ is the property specifying whether it is possible to enforce $\varphi$ from a location that has already been reached by triggering the output sequence of the enforcement mechanism (i.e. a location reached by a prefix of the output) or not. Thus, it can be used to know if soundness is guaranteed or not (i.e. if a location from $\mathrm{Q}_{\mathrm{enf}}$ has been reached). Since the enforcement mechanism ensures that soundness is satisfied as soon as possible, $\mathrm{Pre}(\varphi)$ is a co-safety property, because once $\mathrm{Q}_{\mathrm{enf}}$ is reached, $\varphi$ can be ensured from then.

*Example 2.* For property $\varphi_{\mathrm{ex}}$, $\mathrm{Q}_{\mathrm{enf}} = \{l_1, l_2\}$, and $\mathrm{Pre}(\varphi_{\mathrm{ex}})$ is the property represented by the automaton in Fig. 1b.

Since it is not possible to enforce $\varphi$ from locations in $Q \setminus \mathrm{Q}_{\mathrm{enf}}$, (because uncontrollable events could lead to a location in $Q \setminus F$ trapped with uncontrollable events), an enforcement function should try to always be able to reach locations in $\mathrm{Q}_{\mathrm{enf}}$ to ensure soundness. Thus, property $\mathrm{Enf}(\varphi)$ holds on a sequence whenever $\mathrm{Q}_{\mathrm{enf}}$ is reached in $\mathcal{A}_\varphi$ with this sequence. Since $\mathrm{Q}_{\mathrm{enf}} \subseteq F$, satisfying $\mathrm{Enf}(\varphi)$ is sufficient to satisfy $\varphi$. Thus, we shall enforce $\mathrm{Enf}(\varphi)$.

Based on the above definition and the enforcement limitation illustrated in Example 1, we synthesise an enforcement function for $\varphi$ that is compliant, and sound w.r.t. a property that is as close as possible to $\varphi$ (see later Propositions 1 and 2).

**Definition 6** **($\mathrm{store}_\varphi$, $\mathrm{E}_\varphi$).** [7] *Function* $\mathrm{store}_\varphi : \Sigma^* \to \Sigma^* \times \Sigma^*$ *is defined as follows:*

- $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon)$;
- *for* $\sigma \in \Sigma^*$ *and* $a \in \Sigma$, *let* $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$, *then:*

$$\mathrm{store}_\varphi(\sigma.a) = \begin{cases} (\sigma_s.a.\sigma_s', \sigma_c') & \text{if } a \in \Sigma_u \\ (\sigma_s.\sigma_s'', \sigma_c'') & \text{if } a \in \Sigma_c \end{cases}, \text{ where:}$$

$$\sigma_s' = \max_{\preccurlyeq}(\{w \preccurlyeq \sigma_c \mid \sigma_s.a.w \models \mathrm{Enf}(\varphi)\}),$$

[7] $\mathrm{E}_\varphi$ and $\mathrm{store}_\varphi$ depend on $\Sigma_u$ and $\Sigma_c$, but we did not add them in order to lighten the notations.

$$\sigma'_c = \sigma'^{-1}_s.\sigma_c,$$
$$\sigma''_s = \begin{cases} \epsilon & \text{if } \sigma_s.\sigma_c.a \not\models \mathrm{Enf}(\varphi), \\ \sigma_c.a & \text{otherwise}, \end{cases}$$
$$\sigma''_c = \sigma''^{-1}_s.(\sigma_c.a).$$

*The enforcement function* $\mathrm{E}_\varphi : \Sigma^* \to \Sigma^*$ *is s.t. for* $\sigma \in \Sigma^*$, $\mathrm{E}_\varphi(\sigma) = \Pi_1(\mathrm{store}_\varphi(\sigma))$.

Intuitively, $\sigma_s$ is the word that can be released as output, whereas $\sigma_c$ is the buffer containing the events that are already read/received, but cannot be released as output yet because they lead to an unsafe location from which it would be possible to violate the property reading only uncontrollable events. If $\sigma_s$ satisfies $\mathrm{Pre}(\varphi)$, then the output will always satisfy the property afterwards.

Upon receiving a new action $a$, it is output if it belongs to $\Sigma_u$, followed by the longest prefix of $\sigma_c$ that leads to $\mathrm{Q}_{\mathrm{enf}}$. If the $a$ is controllable, $\sigma_c.a$ is output if it leads to $\mathrm{Q}_{\mathrm{enf}}$, else $a$ is added to the buffer. Property $\mathrm{Enf}(\varphi)$ is used instead of $\varphi$ to ensure that the output of the enforcement function always leads to locations in $\mathrm{Q}_{\mathrm{enf}}$, so that the property will still be satisfied (if it was) upon receiving uncontrollable events.

Enforcement functions as per Definition 6 are sound and compliant.

**Proposition 1.** $\mathrm{E}_\varphi$ *is sound with respect to* $\mathrm{Pre}(\varphi) \implies \varphi$*, as per Definition 3.*

**Proposition 2.** $\mathrm{E}_\varphi$ *is compliant, as per Definition 4.*

Notice that for some properties, blocking all controllable events may still satisfy soundness and compliance. However, for any given input $\sigma$, $\mathrm{E}_\varphi(\sigma)$ is the longest possible word that ensures to reach $\mathrm{Q}_{\mathrm{enf}}$. Controllable events are blocked only when it is not certain that $\mathrm{Q}_{\mathrm{enf}}$ will be reached.

### 3.3 Enforcement Monitors

Enforcement monitors are operational descriptions of enforcement mechanisms. Here, we give a representation of the previous enforcement function as a transition system whose output should be exactly the output of the enforcement function defined in sections 3.1 and 3.2. The purpose is to ease the implementation, since this representation is closer to the real behavior of a monitor.

**Definition 7 (Enforcement monitor).** *An* enforcement monitor $\mathcal{E}$ *for* $\varphi$ *is a transition system* $\langle C^{\mathcal{E}}, c_0^{\mathcal{E}}, \Gamma^{\mathcal{E}}, \hookrightarrow_{\mathcal{E}} \rangle$ *such that:*

- $C^{\mathcal{E}} = Q \times \Sigma^*$ *is the set of configurations.*
- $c_0^{\mathcal{E}} = \langle q_0, \epsilon \rangle$ *is the initial configuration.*
- $\Gamma^{\mathcal{E}} = \Sigma^* \times \{\mathrm{dump}(.), \text{pass-uncont}(.), \text{store-cont}(.)\} \times \Sigma^*$ *is the alphabet, where the first, second, and third members are an input, an operation, and an output, respectively.*
- $\hookrightarrow_{\mathcal{E}} \subseteq C^{\mathcal{E}} \times \Gamma^{\mathcal{E}} \times C^{\mathcal{E}}$ *is the transition relation, defined as the smallest relation obtained by applying the following rules in order (where* $w/ \bowtie /w'$ *stands for* $(w, \bowtie, w') \in \Gamma^{\mathcal{E}}$*):*

- ***dump:*** $\langle q, \sigma.\sigma_c \rangle \xrightarrow{\epsilon/\,\mathrm{dump}(\sigma)/\sigma} \langle q', \sigma_c \rangle$, *with* $q \xrightarrow{\sigma} q'$, *and* $q' \in \mathrm{Q_{enf}}$,
- ***pass-uncont:*** $\langle q, \sigma_c \rangle \xrightarrow{a/\,\text{pass-uncont}(a)/a} \langle q', \sigma_c \rangle$, *with* $a \in \Sigma_u$ *and* $q \xrightarrow{a} q'$,
- ***store-cont:*** $\langle q, \sigma_c \rangle \xrightarrow{a/\,\text{store-cont}(a)/\epsilon} \langle q, \sigma_c.a \rangle$.

Rule **dump** outputs a prefix of the word in memory (the buffer) whenever it is possible to ensure soundness afterwards. Rule **pass-uncont** releases an uncontrollable event as soon as it is received. Rule **store-cont** simply adds a controllable event at the end of the buffer. Compared to section 3.2, the second member of the configuration represents buffer $\sigma_c$ in the definition of $\mathrm{store}_\varphi$, whereas $\sigma_s$ is here represented by location $q$ which is the first member of the configuration, such that $q = \mathrm{Reach}(\sigma_s)$.

**Proposition 3.** *The output of the enforcement monitor $\mathcal{E}$ for input $\sigma$ is* $\mathrm{E}_\varphi(\sigma)$

*Remark 1.* Enforcement monitors as per Definition 7 are somewhat similar to the ones in [13], except that we choose to explicitly keep the memory as part of the configuration and get uniform definitions in the untimed and timed settings (see Section 4). Hence, enforcement monitors as per Definition 7 can also equivalently be defined using a finite-state machine, extending the definition in [13].

## 4 Enforcement Monitoring of Timed Properties

In this section, we extend the framework presented in section 3 to enforce timed properties. Enforcement mechanisms and their properties should be redefined. Enforcement functions need an extra parameter representing the date at which the output is observed. Soundness has to be adapted because, at any time instant, one has to allow the property not to hold, provided that it will hold in the future.

Considering uncontrollable events with timed properties raises several difficulties. First, the order of the events might be modified. Thus, previous definitions of transparency ([19]), stating that the output of an enforcement function will eventually be a delayed prefix of the input, can not be used in this situation. Moreover, when delaying some events to have the property satisfied in the future, one must consider the fact that some uncontrollable events could occur at any moment (and cannot be delayed). Finally, some properties enforceable in [18] cannot be enforced using uncontrollable events, meaning that it is impossible to ensure the soundness of our enforcement mechanisms, as shown in Example 3. It could be possible to use the same definition of soundness as in section 3, where the output always satisfies the property, but then soundness would have been ensured for less properties (i.e. only for safety properties). Weakening soundness allows to enforce more properties, and to let enforcement mechanisms produce longer outputs.

In this section, $\varphi$ is a timed property defined by a deterministic and complete timed automaton $\mathcal{A}_\varphi = \langle L, l_0, X, \Sigma, \Delta, G \rangle$ with semantics $[\![\mathcal{A}_\varphi]\!] = \langle Q, q_0, \Gamma, \rightarrow, F_G \rangle$.

### 4.1 Enforcement Functions and their Properties

An enforcement function takes a timed word and the current time as input, and outputs a timed word:

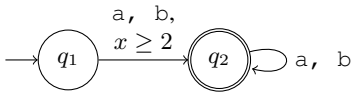



Fig. 2: A timed property enforceable if $\Sigma_u = \emptyset$.

**Definition 8 (Enforcement Function).** *An enforcement function is a function from* $\mathrm{tw}(\Sigma) \times \mathbb{R}_{\geq 0}$ *to* $\mathrm{tw}(\Sigma)$.

As for the untimed case, we define the notions of *soundness* and *compliance*.

**Definition 9 (Soundness).** *An enforcement function $E$ is* sound *w.r.t. $\varphi$ if* $\forall \sigma \in \mathrm{tw}(\Sigma)$, $\forall t \in \mathbb{R}_{\geq 0}, \exists t' \geq t, E(\sigma, t') \models \varphi$.

**Definition 10 (Compliance).** *An enforcement function $E$ is* compliant *if* $\forall \sigma \in \mathrm{tw}(\Sigma)$, $\forall t \in \mathbb{R}_{\geq 0}, \exists t' \geq t, E(\sigma, t')_{|\Sigma_u} = \sigma_{|\Sigma_u} \wedge \sigma_{|\Sigma_c} \preccurlyeq_d E(\sigma, t')_{|\Sigma_c}$.

An enforcement function is sound if for any input timed word, at any time instant, the value of the enforcement function satisfies the property in the future. Compliance is similar to the untimed setting but there are noteworthy differences. First, controllable events can be delayed. Moreover, since timing information is attached to events, it is not necessary to consider an event of $\Sigma_u$. Indeed, the dates of uncontrollable events are the same in the input and in the output, meaning that they are emitted immediately upon their reception. Compliance states that controllable events can be delayed, but their order must be preserved by the enforcement mechanism (i.e. when considering the projections on controllable events, the output should be a delayed prefix of the input). Regarding uncontrollable events, any uncontrollable event is released immediately when received (i.e. when considering the projections on uncontrollable events, the output should be equal to the input).

As in the untimed setting, we say that a property is *enforceable* whenever there exists a sound and compliant enforcement function for this property.

### 4.2 Synthesizing an Enforcement Function

*Example 3 (Non enforceable property).* Consider the property defined by the automaton in Fig. 2 with alphabet $\Sigma = \{a, b\}$. If all actions are controllable ($\Sigma_u = \emptyset$), the property is enforceable because one needs to delay events until clock $x$ exceeds 2. Otherwise, the property is not enforceable. For instance, if $\Sigma_u = \{a\}$, word $(1, a)$ cannot be corrected.

We define a property $\mathrm{Pre}(\varphi)$, indicating whether it is possible to enforce property $\varphi$.

**Definition 11 (Pre).** *Property* $\mathrm{Pre}(\varphi)$ *is defined as the timed property which semantics is* $\langle Q, q_0, \Gamma, \rightarrow', \mathrm{Q}_{\mathrm{enf}} \rangle$ *where:*

- $\mathrm{Q}_{\mathrm{enf}}(\varphi) = \{q \in F_G \mid (q \,\mathrm{after}\, \mathrm{tw}(\Sigma_u)) \subseteq F_G\}$,
- $\mathrm{Q}_{\overline{\mathrm{enf}}}(\varphi) = Q \setminus \mathrm{Q}_{\mathrm{enf}}(\varphi)$,

– $\rightarrow' = (\rightarrow \cap \mathrm{Q}_{\overline{\mathrm{enf}}} \times \Gamma \times Q) \cup \{(q, \gamma, q) \mid q \in \mathrm{Q}_{\mathrm{enf}} \wedge \gamma \in \Gamma\}$.

$\mathrm{Q}_{\mathrm{enf}}(\varphi)$ is the set of states of $[\![\mathcal{A}_\varphi]\!]$ from which it is impossible to reach a bad state reading only uncontrollable events. Thus, it corresponds to the set of states from which it is possible to enforce the property under consideration. $\mathrm{Q}_{\overline{\mathrm{enf}}}$ is the set of states of the semantics of $\varphi$ from which it is not possible to enforce the property, because there is a timed word containing only uncontrollable events (which cannot be modified nor suppressed) leading to a state that is not accepting. In the following, $\mathrm{Q}_{\mathrm{enf}}(\varphi)$ is noted $\mathrm{Q}_{\mathrm{enf}}$ and $\mathrm{Q}_{\overline{\mathrm{enf}}}(\varphi)$ is noted $\mathrm{Q}_{\overline{\mathrm{enf}}}$ to ease the notation.

$\mathrm{Pre}(\varphi)$ is a property indicating whether it is possible to enforce $\varphi$ from the state of the semantics reached after reading a timed word (i.e. every possible continuation leads to $\mathrm{Q}_{\mathrm{enf}}$). Note that once $\mathrm{Q}_{\mathrm{enf}}$ is reached, enforcement becomes effective, then the property will always be satisfied in the future, which explains why $\mathrm{Pre}(\varphi)$ is a co-safety property.

Note that, unlike in the untimed case, $\mathrm{Q}_{\mathrm{enf}}$, $\mathrm{Q}_{\overline{\mathrm{enf}}}$ and $\mathrm{Pre}(\varphi)$ are defined on the semantics of the automaton representing the property and not on the automaton itself. Indeed, the set of states in the semantics in the untimed setting is the same as the set of locations of the property, thus the use of the semantics is not necessary.

We also define function $\mathrm{Safe}$ which, given a state, returns the set of sequences of controllable events that can be emitted safely. Function $\mathrm{Safe}$ is then extended to words:

**Definition 12 (Safe (states)).**

– *Given a state $q$ of the semantics of a timed automaton,*
$\mathrm{Safe}(q) = \{\sigma \in \Sigma_c^* \mid \forall w \in \mathrm{tw}(\Sigma), \Pi_\Sigma(w_{|\Sigma_c}) \preccurlyeq \sigma \implies \exists w' \in \mathrm{tw}(\Sigma), w \preccurlyeq w' \wedge \Pi_\Sigma(w'_{|\Sigma_c}) \preccurlyeq \sigma \wedge \exists q' \in \mathrm{Q}_{\mathrm{enf}}, q \xrightarrow{w'} q'\}$.
– *Given a word $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{Safe}(\sigma) = \mathrm{Safe}(q)$, with $q = \mathrm{Reach}(\sigma)$.*

Intuitively, $\mathrm{Safe}(q)$ is the set of sequences of controllable events for which it is always possible to compute dates to reach $\mathrm{Q}_{\mathrm{enf}}$, even if any uncontrollable event occurs at any time. $\mathrm{Safe}$ shall be used to determine if the enforcement mechanism can release some previously-received controllable events. Contrary to the untimed case, some delay between two consecutive events may be needed to satisfy the property, thus an uncontrollable event could be received by the enforcement mechanism while the delay elapses. Should this happen, the enforcement mechanism needs to compute again the dates for the events it has not output yet in order to reach $\mathrm{Q}_{\mathrm{enf}}$ if possible. $\mathrm{Safe}$ is used to ensure this, i.e. that $\mathrm{Q}_{\mathrm{enf}}$ remains reachable with the events that have not been output yet even if some uncontrollable events occur.

Let us now define an enforcement function for a timed property $\varphi$, denoted as $\mathrm{E}_\varphi$.

**Definition 13 ($\mathrm{E}_\varphi$).** *Let $\mathrm{store}_\varphi$ be the function defined inductively by $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon, \epsilon)$, and for $\sigma \in \mathrm{tw}(\Sigma)$, and $(t, a)$ such that $\sigma.(t, a) \in \mathrm{tw}(\Sigma)$, if $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma)$, then*

$$\mathrm{store}_\varphi(\sigma.(t,a)) = \begin{cases} (\sigma'_s, \sigma'_b, \sigma'_c) & \text{if } a \in \Sigma_u, \\ (\sigma_s.\,\mathrm{obs}(\sigma_b, t), \sigma''_b, \sigma''_c) & \text{if } a \in \Sigma_c, \end{cases}$$

*with:*

$\kappa_\varphi(\sigma_1, \sigma_2, t) = \min_{\text{lex}}(\max_{\preccurlyeq}(\{w \mid \Pi_\Sigma(w) \preccurlyeq \sigma_2 \wedge \text{date}(w(1)) + \text{time}(\sigma_1) \geq t \wedge \Pi_\Sigma(w) \in \text{Safe}(\sigma_1) \wedge \exists q_i \in \text{Reach}(\sigma_1), q_i \xrightarrow{w} q_{\text{enf}} \in \text{Q}_{\text{enf}}\}))$.
$\sigma'_s = \sigma_s.\,\text{obs}(\sigma_b, t).(t, a)$,
$\sigma_{b_1} = \text{nobs}(\sigma_b, t)$,
$\sigma'_b = \kappa_\varphi(\sigma'_s, \Pi_\Sigma(\sigma_{b_1}).\sigma_c, t) +_{\text{t}} \text{time}(\sigma'_s)$,
$\sigma'_c = \Pi_\Sigma(\sigma'_b)^{-1}.(\Pi_\Sigma(\sigma_{b_1}).\sigma_c)$,
$\sigma''_b = \kappa_\varphi(\sigma_s.\,\text{obs}(\sigma_b, t), \Pi_\Sigma(\sigma_{b_1}).\sigma_c.a, t) +_{\text{t}} \text{time}(\sigma_s.\,\text{obs}(\sigma_b, t))$,
$\sigma''_c = \Pi_\Sigma(\sigma''_b)^{-1}.(\Pi_\Sigma(\sigma_{b_1}).\sigma_c.a)$,
*For* $\sigma \in \text{tw}(\Sigma)$*, we define* $\text{E}_\varphi(\sigma, t) = \text{obs}(\Pi_1(s_\sigma).\Pi_2(s_\sigma), t)$*, with* $s_\sigma = \text{store}_\varphi(\sigma)$.

In the definition of $\text{store}_\varphi$, the actions of the input belong to one of the three words $\sigma_s$, $\sigma_b$ and $\sigma_c$. Word $\sigma_s$ represents what has already been output and cannot be modified anymore. The timed word $\sigma_b$ contains the controllable events that are about to be output, such that if $\sigma_b$ is concatenated to $\sigma_s$, the concatenation satisfies $\varphi$. The untimed word $\sigma_c$ contains the controllable actions that remain, meaning that, whatever dates are associated to these actions, it is not sure that $\text{Q}_{\text{enf}}$ would be reached if it was emitted after $\sigma_b$. Yet, the actions of $\sigma_c$ might be released later (because of the occurrence of an uncontrollable event for instance). Thus it is used to compute the new values for $\sigma_b$ and $\sigma_c$ when needed. Note that only the events of $\sigma_c$ are stored with no dates. Indeed, $\sigma_c$ is used only when recomputing dates, thus there is not any date to associate to the events in $\sigma_c$. $\kappa_\varphi$ is computable: even though the number of words satisfying $\Pi_\Sigma(w) \preccurlyeq \sigma_2$ is infinite, since there is an infinite number of possible dates, it is possible to consider only a finite number of words, by considering only words that lead to different regions [1] of the automaton. Moreover, checking if $\Pi_\Sigma(w) \in \text{Safe}(\sigma_1)$ is also computable, because it is a reachability issue, that is computable in the region automaton.

Roughly speaking, the enforcement mechanism described in the previous definition waits for the controllable events of the input to belong to the safe words of the current state, reached with the uncontrollable events (i.e. in $\sigma_c \in \text{Safe}(q)$ if $q$ is the current state), and then starts to emit as many controllable events as possible, with minimum dates greater than the current time. Since the input is safe, $\text{Q}_{\text{enf}}$ will be reached at some point in the future, and then the enforcement mechanism starts again to wait for the input to be safe for the state reached so far, and goes on like previously.

**Proposition 4.** $\text{E}_\varphi$ *is sound with respect to* $\text{Pre}(\varphi) \implies \varphi$*, as per Definition 9.*

**Proposition 5.** $\text{E}_\varphi$ *is compliant, as per Definition 10.*

### 4.3 Enforcement Monitors

As in the untimed setting, we give here an operational description of an enforcement mechanism whose output is exactly the output of $\text{E}_\varphi$, as defined in Definition 13.

**Definition 14.** *An* enforcement monitor $\mathcal{E}$ *for* $\varphi$ *is a transition system* $\langle C^{\mathcal{E}}, c_0^{\mathcal{E}}, \Gamma^{\mathcal{E}}, \hookrightarrow_{\mathcal{E}} \rangle$ *such that:*

- $C^{\mathcal{E}} = \text{tw}(\Sigma) \times \Sigma_c^* \times Q \times \mathbb{R}_{\geq 0}$ *is the set of configurations.*
- $c_0^{\mathcal{E}} = \langle \epsilon, \epsilon, q_0, 0 \rangle \in C^{\mathcal{E}}$ *is the initial configuration.*

– $\Gamma^{\mathcal{E}} = ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}) \times Op \times ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\})$ *is the alphabet, composed of an optional input, an operation and an optional output. The set of operations is* $\{\mathrm{dump}(.), \text{pass-uncont}(.), \text{store-cont}(.), \mathrm{delay}(.)\}$. *Whenever* $(\sigma, \bowtie, \sigma') \in \Gamma^{\mathcal{E}}$, *it is noted* $\sigma / \bowtie / \sigma'$.
– $\hookrightarrow_{\mathcal{E}}$ *is the transition relation defined as the smallest relation obtained by applying the following rules given by their priority order:*
  - ***Dump:*** $\langle (t_b, a).\sigma_b, \sigma_c, q, t\rangle \xrightarrow{\epsilon/\,\mathrm{dump}(t_b,a)/(t_b,a)} \langle \sigma_b, \sigma_c, q', t\rangle$ *if* $t_b = t$ *with* $q \xrightarrow{(t_b,a)} q'$,
  - ***Pass-uncont:*** $\langle \sigma_b, \sigma_c, q, t\rangle \xrightarrow{(t,a)/\,\text{pass-uncont}(t,a)/(t,a)} \langle \sigma'_b, \sigma'_c, q', t\rangle$, *with* $q \xrightarrow{(t,a)} q'$ *and* $(\sigma'_b, \sigma'_c) = \mathrm{update}(q', \sigma_b, \sigma_c, t)$,
  - ***Store-cont:*** $\langle \sigma_b, \sigma_c, q, t\rangle \xrightarrow{(t,c)/\,\text{store-cont}((t,c))/\epsilon} \langle \sigma'_b, \sigma'_c, q, t\rangle$, *with* $(\sigma'_b, \sigma'_c) = \mathrm{update}(q, \sigma_b, \sigma_c.c, t)$,
  - ***Delay:*** $\langle \sigma_b, \sigma_c, (l, v), t\rangle \xrightarrow{\epsilon/\,\mathrm{delay}(\delta)/\epsilon} \langle \sigma_b, \sigma_c, (l, v+\delta), t+\delta\rangle$,

  *where* update *is a function computing* $\kappa_\varphi$ *from previous definition of* $\mathrm{E}_\varphi$.

In a configuration $\langle \sigma_b, \sigma_c, q, t\rangle$, $\sigma_b$ is the word to be output as time elapses; $\sigma_c$ is what is left from the input; $q$ is the state of the semantics reached after reading what has already been output; $t$ is the current time instant, i.e. the time elapsed since the beginning of the run.

Sequences $\sigma_b$ and $\sigma_c$ are as in the definition of $\mathrm{store}_\varphi$, whereas $q$ represents $\sigma_s$, such that $q = \mathrm{Reach}(\sigma_s)$. Function update computes the values of $\sigma_b$ and $\sigma_c$ to ensure soundness. Function update represents the computation of function $\kappa_\varphi$ in the definition of $\mathrm{store}_\varphi$.

**Proposition 6.** *The output of* $\mathcal{E}$ *for input* $\sigma$ *is* $\mathrm{E}_\varphi(\sigma)$.

### 4.4 Example

Figure 3 depicts a property modeling the use of some shared writable device. One can get the status of a lock through the uncontrollable events `LockOn` and `LockOff` indicating that the lock has been acquired, and that it is available, respectively. The uncontrollable event `Auth` is sent by the device to authorize writings. Once event `Auth` is received, the controllable event `Write` can be sent after having waited a little bit for synchronization. Each time the lock is acquired and released, we must also wait before issuing a new `Write` order. The sets of events are: $\Sigma_c = \{$`Write`$\}$ and $\Sigma_u = \{$`Auth`, `LockOff`, `LockOn`$\}$.

Let us follow the output of function $\mathrm{store}_\varphi$ over time with the following input word: $\sigma = (1,$`Auth`$).(2,$`LockOn`$).(4,$`Write`$).(5,$`LockOff`$).(6,$`LockOn`$).(7,$`Write`$).(8,$`LockOff`$)$. Let $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\mathrm{obs}(\sigma, t))$. Table 1 gives the values taken by $\sigma_s$, $\sigma_b$, and $\sigma_c$ over time. To compute them, first notice that $\mathrm{Q}_{\mathrm{enf}} = \{l_1, l_2\} \times \mathcal{V}(\{x\})$. Moreover, we can see that `Write` $\in \mathrm{Safe}(l_1)$ because it is always possible to delay the `Write` event in such a way that the current state remains in $\mathrm{Q}_{\mathrm{enf}}$, whatever are the uncontrollable events. Consequently, whenever $\sigma_s$ leads to $l_1$, $\sigma_b$ is empty (because `Write` is the only controllable event, thus $\sigma_b$ must start with a `Write` event).

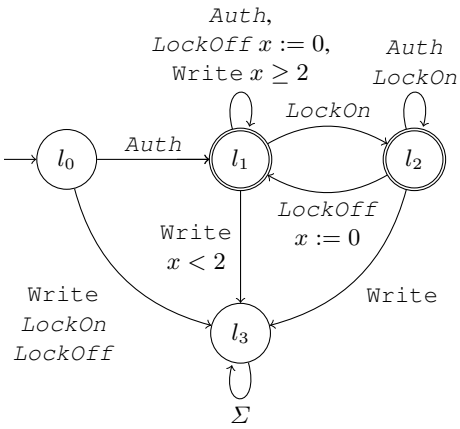



Fig. 3: Property modeling writes on a shared storage device

Figure 4 shows the execution of the enforcement monitor with input $(1, \texttt{Auth})$. $(2, \texttt{LockOn})$. $(4, \texttt{Write})$. $(5, \texttt{LockOff})$. $(6, \texttt{LockOn})$. $(7, \texttt{Write})$.$(8, \texttt{LockOff})$. In a configuration, the input is on the right, the output on the left, and the middle is the current configuration of the enforcement monitor. Variable $t$ keeps track of global time. A valuation is represented as an integer – the value of the (unique) clock $x$. Observe that the final output at $t = 10$ is the same as the one of the enforcement function: $(1, \texttt{Auth})$.$(2, \texttt{LockOn})$.$(5, \texttt{LockOff})$.$(6, \texttt{LockOn})$.$(8, \texttt{LockOff})$.$(10, \texttt{Write})$. $(10, \texttt{Write})$

## 5 Related Work

Runtime enforcement was pioneered by Schneider with security automata [21], a runtime mechanism for enforcing safety properties. In this work monitors are able to stop the execution of the system once a deviation of the property has been detected. Later, Ligatti et al. proposed edit-automata, a more powerful model of enforcement monitors able to introduce and suppress events from the execution. Later, more general models were proposed where the monitors can be synthesized from regular properties [13]. More recently, Bloem et al. [6] presented a framework to synthesize enforcement monitors for reactive systems, called as *shields*, from a set of safety properties. A shield acts instantaneously and cannot buffer actions. Whenever a property violation is unavoidable, the shield allows to deviate from the property for $k$ consecutive steps (as in [7]). Whenever a second violation occurs within $k$ steps, then the shield enters into a *fail-safe* mode, where it ensures only correctness. Another recent approach by Dolzehnko et al [11] introduces Mandatory Result Automata (MRAs). MRAs extend edit-automata by refining the input-output relationship of an enforcement mechanism and thus allowing a more precise description of the enforcement abilities of an enforcement mechanism in concrete application scenarios. All the previously mentioned approaches considered untimed specifications, and do not consider uncontrollable events.

$$
\begin{array}{ll}
\text{t = 0} & \epsilon/(\epsilon, \epsilon, (l_0, 0), 0)/(1, \texttt{Auth}).(2, on).(4, \texttt{w}).(5, \texttt{off}).(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{delay}(1) \\
\text{t = 1} & \epsilon/(\epsilon, \epsilon, (l_0, 1), 1)/(1, \texttt{Auth}).(2, on).(4, \texttt{w}).(5, \texttt{off}).(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{pass-uncont}((1, \texttt{Auth})) \\
\text{t = 1} & (1, \texttt{Auth})/(\epsilon, \epsilon, (l_1, 1), 1)/(2, on).(4, \texttt{w}).(5, \texttt{off}).(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{delay}(1) \\
\text{t = 2} & (1, \texttt{Auth})/(\epsilon, \epsilon, (l_1, 2), 2)/(2, on).(4, \texttt{w}).(5, \texttt{off}).(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{pass-uncont}((2, on)) \\
\text{t = 2} & (1, \texttt{Auth}).(2, on)/(\epsilon, \epsilon, (l_2, 2), 2)/(4, \texttt{w}).(5, \texttt{off}).(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{delay}(2) \\
\text{t = 4} & (1, \texttt{Auth}).(2, on)/(\epsilon, \epsilon, (l_2, 4), 4)/(4, \texttt{w}).(5, \texttt{off}).(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{store-cont}((4, \texttt{w})) \\
\text{t = 4} & (1, \texttt{Auth}).(2, on)/(\epsilon, (4, \texttt{w}), (l_2, 4), 4)/(5, \texttt{off}).(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{delay}(1) \\
\text{t = 5} & (1, \texttt{Auth}).(2, on)/(\epsilon, (4, \texttt{w}), (l_2, 5), 5)/(5, \texttt{off}).(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{pass-uncont}((5, \texttt{off})) \\
\text{t = 5} & (1, \texttt{Auth}).(2, on).(5, \texttt{off})/((7, \texttt{w}), \epsilon, (l_1, 0), 5)/(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{delay}(1) \\
\text{t = 6} & (1, \texttt{Auth}).(2, on).(5, \texttt{off})/((7, \texttt{w}), \epsilon, (l_1, 1), 6)/(6, on).(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{pass-uncont}((6, on)) \\
\text{t = 6} & (1, \texttt{Auth}).(2, on).(5, \texttt{off}).(6, on)/(\epsilon, (7, \texttt{w}), (l_2, 1), 6)/(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{delay}(1) \\
\text{t = 7} & (1, \texttt{Auth}).(2, on).(5, \texttt{off}).(6, on)/(\epsilon, (7, \texttt{w}), (l_2, 2), 7)/(7, \texttt{w}).(8, \texttt{off}) \\
& \downarrow \text{store-cont}((7, \texttt{w})) \\
\text{t = 7} & (1, \texttt{Auth}).(2, on).(5, \texttt{off}).(6, on)/(\epsilon, (7, \texttt{w}).(7, \texttt{w}), (l_2, 2), 7)/(8, \texttt{off}) \\
& \downarrow \text{delay}(1) \\
\text{t = 8} & (1, \texttt{Auth}).(2, on).(5, \texttt{off}).(6, on)/(\epsilon, (7, \texttt{w}).(7, \texttt{w}), (l_2, 3), 8)/(8, \texttt{off}) \\
& \downarrow \text{pass-uncont}((8, \texttt{off})) \\
\text{t = 8} & (1, \texttt{Auth}).(2, on).(5, \texttt{off}).(6, on).(8, \texttt{off})/((10, \texttt{w}).(10, \texttt{w}), \epsilon, (l_1, 0), 8)/\epsilon \\
& \downarrow \text{delay}(2) \\
\text{t = 10} & (1, \texttt{Auth}).(2, on).(5, \texttt{off}).(6, on).(8, \texttt{off})/((10, \texttt{w}).(10, \texttt{w}), \epsilon, (l_1, 2), 10)/\epsilon \\
& \downarrow \text{dump}((10, \texttt{w})) \\
\text{t = 10} & (1, \texttt{Auth}).(2, on).(5, \texttt{off}).(6, on).(8, \texttt{off}).(10, \texttt{w})/((10, \texttt{w}), \epsilon, (l_1, 2), 10)/\epsilon \\
& \downarrow \text{dump}((10, \texttt{w})) \\
\text{t = 10} & (1, \texttt{Auth}).(2, on).(5, \texttt{off}).(6, on).(8, \texttt{off}).(10, \texttt{w}).(10, \texttt{w})/(\epsilon, \epsilon, (l_1, 2), 10)/\epsilon
\end{array}
$$

Fig. 4: Execution of an enforcement monitor with input $(1, \texttt{Auth})$. $(2, \texttt{LockOn})$. $(4, \texttt{Write})$. $(5, \texttt{LockOff})$. $(6, \texttt{LockOn})$. $(7, \texttt{Write})$. $(8, \texttt{LockOff})$. $\texttt{LockOff}$ is abbreviated as $\texttt{off}$, $\texttt{LockOn}$ as $on$, and $\texttt{Write}$ as $\texttt{w}$

Table 1: Values of $(\sigma_s, \sigma_b, \sigma_c) = \text{store}_\varphi((1, \mathit{Auth}).\ (2, \mathit{LockOn}).\ (4, \texttt{Write}).\ (5, \mathit{LockOff}).\ (6, \mathit{LockOn}).\ (7, \texttt{Write}).\ (8, \mathit{LockOff}))$ over time.

| t | $\sigma_s$ | $\sigma_b$ | $\sigma_c$ |
|---|---|---|---|
| 1 | $(1, \mathit{Auth})$ | $\epsilon$ | $\epsilon$ |
| 2 | $(1, \mathit{Auth}).(2, \mathit{LockOn})$ | $\epsilon$ | $\epsilon$ |
| 4 | $(1, \mathit{Auth}).(2, \mathit{LockOn})$ | $\epsilon$ | $\texttt{Write}$ |
| 5 | $(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff})$ | $(7, \texttt{Write})$ | $\epsilon$ |
| 6 | $(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff}).$ $(6, \mathit{LockOn})$ | $\epsilon$ | $\texttt{Write}$ |
| 7 | $(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff}).$ $(6, \mathit{LockOn})$ | $\epsilon$ | $\texttt{Write}.\texttt{Write}$ |
| 8 | $(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff}).$ $(6, \mathit{LockOn}).(8, \mathit{LockOff})$ | $(10, \texttt{Write}).(10, \texttt{Write})$ | $\epsilon$ |
| 10 | $(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff}).$ $(6, \mathit{LockOn}).(8, \mathit{LockOff}).(10, \texttt{Write}).$ $(10, \texttt{Write})$ | $\epsilon$ | $\epsilon$ |

In the timed setting, several monitoring tools for timed specifications have been proposed. RT-Mac [20] permits to verify at runtime timeliness and reliability correctness. LARVA [8,9] takes as input safety properties expressed with DATEs (Dynamic Automata with Times and Events), a timed model similar to timed automata.

In previous work, we introduced *runtime enforcement for timed properties* [19] specified by timed automata [1]. We proposed a model of enforcement monitors that work as *delayers*, that is, mechanisms that are able to delay the input sequence of timed events to correct it. While [19] proposed synthesis techniques only for safety and co-safety properties, we then generalized the framework to synthesize an enforcement monitor for any regular timed property [17,18]. In [16], we considered parametric timed properties, that is timed properties with data-events containing information from the execution of the monitored system. None of our previous research endeavors considered uncontrollable events. Considering uncontrollable events entailed us to revisit and redefine all the notions related to enforcement mechanisms (soundness, transparency weaken into compliance, enforcement function, and enforcement monitor). Moreover, we define an enforcement condition as a property, that allows to determine when an enforcement mechanism can safely output controllable events, independently of the uncontrollable events that can be received by the enforcement mechanism.

Basin et al. [4] introduced uncontrollable events in safety properties enforced with security automata [21]. More recently, Basin et al. proposed a more general approach [3] related to enforcement of security policies with controllable and uncontrollable events. Basin et al. presented several complexity results and showed how to synthesize enforcement mechanisms. In case of violation of the property, the system stops the execution. The approaches in [4,3] only handle discrete time, and clock ticks are considered as uncontrollable events. In our framework, we consider dense time using the expressiveness of timed automata. We synthesize enforcement mechanisms for any regular property, and not just safety properties. Moreover, our monitor are more flexible since they block

the input word only when delaying events cannot prevent the violation of the desired property, thus offering the possibility to correct violations due to the timing of events.

## 6 Conclusion and Future Work

This paper extends the research endeavors on runtime enforcement and focuses on the use of uncontrollable events. An enforcement mechanism can only observe uncontrollable events, but cannot delay nor suppress them. Considering uncontrollable events entails to change the order between controllable and uncontrollable events, and to adapt the usual requirements on enforcement mechanisms. Therefore, we weaken transparency into compliance. We define enforcement mechanisms at two levels of abstraction (enforcement functions and enforcement monitors), for regular properties and regular timed properties. Since not all properties can be enforced, we also give a condition, depending on the property and the input word, indicating whether or not the enforcement mechanism can be sound with respect to the property under scrutiny. An enforcement mechanism outputs all received uncontrollable events, and stores the controllable ones, until soundness can be guaranteed. Then, it outputs events only when it can ensure that soundness will be satisfied.

Several extensions of this work are possible. A first extension is to consider more risky strategies regarding uncontrollable events, outputting events even if some uncontrollable events could lead to a bad state. Following such strategies could be guided by an additional probabilistic model on the occurrence of input events. A second extension is to implement the enforcement mechanisms using UPPAAL libraries [14]. A third extension is to use runtime enforcement to modify at runtime the parameters of a system with stochastic behavior, as done offline in [2]. A fourth extension is to define a theory of runtime enforcement for distributed systems where enforcement monitors are decentralised on the components of the verified system, as is the case with verification monitors in [5]. A fifth extension is to distinguish inputs from outputs in properties, and consider for instance timed i/o automata [10] to formalise properties.

## References


1. Alur, R., Dill, D.: The theory of timed automata. In: de Bakker, J., Huizing, C., de Roever, W., Rozenberg, G. (eds.) Real-Time: Theory in Practice, Lecture Notes in Computer Science, vol. 600, pp. 45–73. Springer Berlin Heidelberg (1992)
2. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: System design of stochastic models using robustness of temporal properties. Theor. Comput. Sci. 587, 3–25 (2015)
3. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. 16(1), 3:1–3:26 (Jun 2013)
4. 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). Lecture Notes in Computer Science, vol. 7186, pp. 260–275. Springer-Verlag (2011)
5. Bauer, A.K., Falcone, Y.: Decentralised LTL monitoring. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012: Formal Methods - 18th International Symposium, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7436, pp. 85–100. Springer (2012)

6. Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis: - runtime enforcement for reactive systems. In: Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015. Proceedings. pp. 533–548 (2015)
7. Charafeddine, H., El-Harake, K., Falcone, Y., Jaber, M.: Runtime enforcement for component-based systems. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015. pp. 1789–1796 (2015)
8. 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). pp. 33–37. IEEE Computer Society (2009)
9. 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). Lecture Notes in Computer Science, vol. 5813, pp. 103–117. Springer (2009)
10. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed i/o automata: A complete specification theory for real-time systems. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control. pp. 91–100. HSCC '10, ACM (2010)
11. Dolzhenko, E., Ligatti, J., Reddy, S.: Modeling runtime enforcement with mandatory results automata. International Journal of Information Security 14(1), 47–60 (Feb 2015)
12. 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.) Runtime Verification - First International Conference, RV, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6418, pp. 89–105. Springer (2010)
13. Falcone, Y., Mounier, L., Fernandez, J., Richier, J.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods in System Design 38(3), 223–262 (2011)
14. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer 1, 134–152 (1997)
15. Ligatti, J., Bauer, L., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. 12(3), 19:1–19:41 (Jan 2009)
16. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H.: Runtime enforcement of parametric timed properties with practical applications. In: Lesage, J., Faure, J., Cury, J.E.R., Lennartson, B. (eds.) 12th International Workshop on Discrete Event Systems, WODES 2014, 2014. pp. 420–427. International Federation of Automatic Control (2014)
17. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H.: Runtime enforcement of regular timed properties. In: Cho, Y., Shin, S.Y., Kim, S., Hung, C., Hong, J. (eds.) Symposium on Applied Computing, SAC, 2014. pp. 1279–1286. ACM (2014)
18. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A., Nguena-Timo, O.: Runtime enforcement of timed properties revisited. Formal Methods in System Design 45(3), 381–422 (2014)
19. Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A., Nguena-Timo, O.L.: Runtime enforcement of timed properties. In: Qadeer, S., Tasiran, S. (eds.) Runtime Verification, Third International Conference, RV 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7687, pp. 229–244. Springer (2012)
20. 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, 147–153 (2005)
21. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1), 30–50 (Feb 2000)

# A Proofs

## A.1 Proofs for the Untimed Setting

In all this section, we will use the notations from section 3, meaning that $\varphi$ is a property whose associated automaton is $\mathcal{A}_\varphi = \langle Q, q_0, \Sigma, \rightarrow, F\rangle$, $\mathrm{Q}_{\mathrm{enf}} = \{q \in F \mid (q \,\mathrm{after}\, \Sigma_u^*) \subseteq F\}$, and $\mathrm{Q}_{\overline{\mathrm{enf}}} = Q \setminus \mathrm{Q}_{\mathrm{enf}}$. In some proofs, we also use notations from Definition 6.

**Lemma 1.** *If $\mathcal{A}_\varphi$ is deterministic and complete, then $\mathrm{Pre}(\varphi)$ is deterministic and complete.*

*Proof.* The modifications made to $\rightarrow$ only modify the destination state, never the source state nor the action. Thus, if $\mathcal{A}$ is deterministic and complete, $\mathrm{Pre}(\varphi)$ is too.

**Lemma 2.** $\mathrm{Enf}(\varphi) \implies \varphi$.

*Proof.* $\mathrm{Q}_{\mathrm{enf}} \subseteq F$, thus, for all $\sigma \in \Sigma^*$, if $\sigma$ satisfies $\mathrm{Enf}(\varphi)$, then $\sigma$ also satisfies $\varphi$.

**Lemma 3.** $\forall \sigma \in \Sigma^*, \forall a \in \Sigma, (\sigma \not\models \mathrm{Pre}(\varphi) \wedge \sigma.a \models \mathrm{Pre}(\varphi)) \implies \sigma.a \models \mathrm{Enf}(\varphi)$.

*Proof.* Let us consider $\sigma \in \Sigma^*$ such that $\sigma \not\models \mathrm{Pre}(\varphi)$, and $a \in \Sigma$ such that $\sigma.a \models \mathrm{Pre}(\varphi)$. Let us note $\rightarrow_1 = \rightarrow \cap\, \mathrm{Q}_{\overline{\mathrm{enf}}} \times \Sigma \times Q$, $\rightarrow_2 = \{(q, a, q) \mid q \in \mathrm{Q}_{\mathrm{enf}} \wedge a \in \Sigma\}$, and $\rightarrow_c = \rightarrow_1 \cup \rightarrow_2$, so that $\mathrm{Pre}(\varphi) = \langle Q, q_0, \Sigma, \rightarrow_c, \mathrm{Q}_{\mathrm{enf}}\rangle$.

$\mathrm{Pre}(\varphi)$ is a co-safety property. Thus $\nexists \sigma' \preccurlyeq \sigma, q_0 \xrightarrow{\sigma'}_c q$, with $q \in \mathrm{Q}_{\mathrm{enf}}$. Thus, if $q$ is such that $q_0 \xrightarrow{\sigma}_c q$, then $q_0 \xrightarrow{\sigma}_1 q$. Since $\sigma \not\models \mathrm{Pre}(\varphi)$, $q \in \mathrm{Q}_{\overline{\mathrm{enf}}}$. Moreover, $\sigma.a \models \mathrm{Pre}(\varphi)$, thus $\exists q' \in \mathrm{Q}_{\mathrm{enf}}, q \xrightarrow{a}_c q'$. Since $q \in \mathrm{Q}_{\overline{\mathrm{enf}}}$, $q \xrightarrow{a}_1 q'$. Furthermore, $\rightarrow_1 \subseteq \rightarrow$, thus $q_0 \xrightarrow{\sigma.a} q'$, meaning that $\sigma.a \models \mathrm{Enf}(\varphi)$ because $q' \in \mathrm{Q}_{\mathrm{enf}}$.

**Proposition 1.** $\mathrm{E}_\varphi$ *is sound with respect to* $\mathrm{Pre}(\varphi) \implies \varphi$*, as per Definition 3.*

*Proof.* Let $\mathrm{P}(\sigma)$ be the predicate: "$\mathrm{E}_\varphi(\sigma) \models \mathrm{Pre}(\varphi) \implies \mathrm{Enf}(\varphi)$". Let us prove that $\forall \sigma \in \Sigma^*, \mathrm{P}(\sigma)$.

- *Induction basis*: $\mathrm{E}_\varphi(\epsilon) = \epsilon$. If $\epsilon \models \mathrm{Pre}(\varphi)$, then $q_0 \in \mathrm{Q}_{\mathrm{enf}}$, thus $\epsilon \models \mathrm{Enf}(\varphi)$. This means that $\mathrm{P}(\epsilon)$ holds.
- *Induction step*: Suppose now that, for some $\sigma \in \Sigma^*$, $\mathrm{P}(\sigma)$ holds. Let us consider $a \in \Sigma$. Let us prove that $\mathrm{P}(\sigma.a)$ holds.
  - If $\mathrm{E}_\varphi(\sigma.a) \not\models \mathrm{Pre}(\varphi)$, then $\mathrm{P}(\sigma.a)$ holds.
  - If $\mathrm{E}_\varphi(\sigma) \not\models \mathrm{Pre}(\varphi) \wedge \mathrm{E}_\varphi(\sigma.a) \models \mathrm{Pre}(\varphi)$:
    - If $a \in \Sigma_u$, then considering $\sigma'_s$ from Definition 6, either $\sigma'_s \neq \epsilon$ and then $\mathrm{E}_\varphi(\sigma.a) \models \mathrm{Enf}(\varphi)$, or $\sigma'_s = \epsilon$, meaning that $\mathrm{E}_\varphi(\sigma.a) = \mathrm{E}_\varphi(\sigma).a$. $\mathrm{E}_\varphi(\sigma) \not\models \mathrm{Pre}(\varphi)$, and $\mathrm{E}_\varphi(\sigma).a \models \mathrm{Pre}(\varphi)$. Thus, using lemma 3, $\mathrm{E}_\varphi(\sigma).a = \mathrm{E}_\varphi(\sigma.a) \models \mathrm{Enf}(\varphi)$. In both cases, $\mathrm{P}(\sigma.a)$.
    - If $a \in \Sigma_c$, then, since $\mathrm{E}_\varphi(\sigma) \models \mathrm{Pre}(\varphi)$ and $\mathrm{E}_\varphi(\sigma.a) \not\models \mathrm{Pre}(\varphi)$, $\mathrm{E}_\varphi(\sigma) \neq \mathrm{E}_\varphi(\sigma.a)$. Thus, $\sigma''_s \neq \epsilon$, which means that $\mathrm{E}_\varphi(\sigma.a) \models \mathrm{Enf}(\varphi)$. Thus $\mathrm{P}(\sigma.a)$.

- If $\mathrm{E}_\varphi(\sigma) \models \mathrm{Pre}(\varphi)$, then $\mathrm{E}_\varphi(\sigma) \models \mathrm{Enf}(\varphi)$ because $\mathrm{P}(\sigma)$ holds.
  - If $a \in \Sigma_u$, then, considering $\sigma'_s$ as in Definition 6, if $\sigma'_s \neq \epsilon$ then $\mathrm{E}_\varphi(\sigma.a) \models \mathrm{Enf}(\varphi)$, meaning that $\mathrm{P}(\sigma.a)$ holds. Otherwise, $\sigma'_s = \epsilon$, and then $\mathrm{E}_\varphi(\sigma.a) = \mathrm{E}_\varphi(\sigma).a$. Since $\mathrm{E}_\varphi(\sigma) \models \mathrm{Enf}(\varphi)$, $\exists q \in \mathrm{Q}_{\mathrm{enf}}, q_0 \xrightarrow{\mathrm{E}_\varphi(\sigma)} q$. Then $\exists q' \in \mathrm{Q}_{\mathrm{enf}}, q \xrightarrow{a} q'$, because $a \in \Sigma_u$. Thus $q_0 \xrightarrow{\mathrm{E}_\varphi(\sigma).a} q'$, meaning that $\mathrm{E}_\varphi(\sigma).a = \mathrm{E}_\varphi(\sigma.a) \models \mathrm{Enf}(\varphi)$ because $q' \in \mathrm{Q}_{\mathrm{enf}}$. Thus $\mathrm{P}(\sigma.a)$.
  - If $a \in \Sigma_c$, then considering $\sigma''_s$ as defined in Definition 6, if $\sigma''_s \neq \epsilon$ then $\mathrm{E}_\varphi(\sigma.a) \models \mathrm{Enf}(\varphi)$. Otherwise, $\sigma''_s = \epsilon$, meaning that $\mathrm{E}_\varphi(\sigma.a) = \mathrm{E}_\varphi(\sigma) \models \mathrm{Enf}(\varphi)$. In both cases, $\mathrm{P}(\sigma.a)$ holds.

  In all cases, $\mathrm{P}(\sigma.a)$ holds. Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.a)$.

By induction, $\forall \sigma \in \Sigma^*, \mathrm{E}_\varphi(\sigma) \models \mathrm{Pre}(\varphi) \implies \mathrm{Enf}(\varphi)$. Thus, using lemma 2, $\forall \sigma \in \Sigma^*, \mathrm{E}_\varphi(\sigma) \models \mathrm{Pre}(\varphi) \implies \varphi$.

**Proposition 2.** $\mathrm{E}_\varphi$ *is compliant, as per Definition 4.*

*Proof.* Let $\mathrm{P}(\sigma)$ be the predicate: "$(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma) \implies \sigma_{s|\Sigma_c}.\sigma_c = \sigma_{|\Sigma_c}$". Let us prove that $\forall \sigma \in \Sigma^*, \mathrm{P}(\sigma)$ holds.

- *Induction basis*: $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon)$, and $\epsilon_{|\Sigma_c} = \epsilon_{|\Sigma_c}.\epsilon$. Thus $\mathrm{P}(\epsilon)$ holds.
- *Induction step*: suppose now that for $\sigma \in \Sigma^*, \mathrm{P}(\sigma)$ holds. Let us consider $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$, $a \in \Sigma$, and $(\sigma_t, \sigma_d) = \mathrm{store}_\varphi(\sigma.a)$. Let us prove that $\mathrm{P}(\sigma.a)$ holds.
  - If $a \in \Sigma_u$, then $\sigma_t = \sigma_s.a.\sigma'_s$ and $\sigma_t.\sigma_d = \sigma_s.a.\sigma_c$. Therefore, $\sigma_{t|\Sigma_c}.\sigma_d = \sigma_{s|\Sigma_c}.\sigma'_s.\sigma'^{-1}_s.\sigma_c = \sigma_{s|\Sigma_c}.\sigma_c$. Since $\mathrm{P}(\sigma)$, $\sigma_{t|\Sigma_c}.\sigma_d = \sigma_{|\Sigma_c} = (\sigma.a)_{|\Sigma_c}$. Thus $\mathrm{P}(\sigma.a)$.
  - If $a \in \Sigma_c$, then $\sigma_t = \sigma_s.\sigma''_s$ and $\sigma_t.\sigma_d = \sigma_s.\sigma_c.a$. Therefore, $\sigma_{t|\Sigma_c}.\sigma_d = \sigma_{s|\Sigma_c}.\sigma''_s.\sigma''^{-1}_s.\sigma_c.a = \sigma_{s|\Sigma_c}.\sigma_c.a = \sigma_{|\Sigma_c}.a = (\sigma.a)_{|\Sigma_c}$. Thus $\mathrm{P}(\sigma.a)$.

  Thus, $\forall \sigma \in \Sigma^*, \forall a \in \Sigma, \mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.a)$.

By induction on $\sigma$, $\forall \sigma \in \Sigma^*$, $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma) \implies \sigma_{s|\Sigma_c}.\sigma_c = \sigma_{|\Sigma_c}$.

Moreover, $\forall \sigma \in \Sigma^*, \forall u \in \Sigma_u$, if $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$ and $(\sigma_t, \sigma_d) = \mathrm{store}_\varphi(\sigma.u)$, then $\sigma_t = \sigma_s.u.\sigma'_s$, thus $\sigma_s.u \preccurlyeq \sigma_t$, meaning that $\mathrm{E}_\varphi(\sigma).u \preccurlyeq \mathrm{E}_\varphi(\sigma.u)$.

Thus, $\forall \sigma \in \Sigma^*, \mathrm{E}_\varphi(\sigma)_{|\Sigma_c} \preccurlyeq \sigma_{|\Sigma_c} \wedge \forall u \in \Sigma_u, \mathrm{E}_\varphi(\sigma).u \preccurlyeq \mathrm{E}_\varphi(\sigma.u)$, meaning that $\mathrm{E}_\varphi$ is compliant.

**Proposition 3.** *The output of the enforcement monitor* $\mathcal{E}$ *for input* $\sigma$ *is* $\mathrm{E}_\varphi(\sigma)$

*Proof.* Let us introduce some notation for this proof: for a word $w \in \Gamma^{\mathcal{E}*}$, we note $\mathrm{input}(w) = \Pi_1(w(1)).\Pi_1(w(2)) \ldots \Pi_1(w(|w|))$, the word obtained by concatenating the first members (the inputs) of $w$. In a similar way, we note $\mathrm{output}(w) = \Pi_3(w(1)).\Pi_3(w(2)) \ldots \Pi_3(w(|w|))$, the word obtained by concatenating all the third members (outputs) of $w$. We also define the $\mathrm{Rules}$ function as follows:

$$\mathrm{Rules} : \begin{cases} \Sigma^* \to \Gamma^{\mathcal{E}*} \\ \sigma \quad \mapsto \max_{\preccurlyeq}(\{w \in \Gamma^{\mathcal{E}*} \mid \mathrm{input}(w) = \sigma \wedge \exists c_1 \in C^{\mathcal{E}}, c_0^{\mathcal{E}} \xrightarrow{w} c_1\}) \end{cases}$$

For a word $\sigma \in \Sigma^*$, $\mathrm{Rules}(\sigma)$ is the trace of the longest valid run in $\mathcal{E}$, i.e. the sequence of all the rules that can be applied with input $\sigma$. We then extend the definition of $\mathrm{output}$ to words in $\Sigma^*$: for $\sigma \in \Sigma^*$, $\mathrm{output}(\sigma) = \mathrm{output}(\mathrm{Rules}(\sigma))$.

Let $\mathrm{P}(\sigma)$ be the predicate: "$\mathrm{E}_\varphi(\sigma) = \mathrm{output}(\sigma) \wedge ((\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma) \wedge c_0^{\mathcal{E}} \xrightarrow{\mathrm{Rules}(\sigma)} \langle q, \sigma_c' \rangle) \implies (q = \mathrm{Reach}(\sigma_s) \wedge \sigma_c = \sigma_c')$".
Let us prove that for all $\sigma \in \Sigma^*$, $\mathrm{P}(\sigma)$ holds.

- *Induction basis*: $\mathrm{E}_\varphi(\epsilon) = \epsilon = \mathrm{output}(\epsilon)$. Moreover, $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon)$, and $c_0^{\mathcal{E}} \xrightarrow{\epsilon} c_0^{\mathcal{E}}$. Therefore, as $c_0^{\mathcal{E}} = \langle q_0, \epsilon \rangle$, $\mathrm{P}(\epsilon)$ holds, because $\mathrm{Reach}(\epsilon) = q_0$.
- *Induction step*: Let us suppose now that for some $\sigma \in \Sigma^*$, $\mathrm{P}(\sigma)$ holds. Let us consider $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$, $q = \mathrm{Reach}(\sigma_s)$, $a \in \Sigma$, and $(\sigma_t, \sigma_d) = \mathrm{store}_\varphi(\sigma.a)$. Let us prove that $\mathrm{P}(\sigma.a)$ holds.

  Since $\mathrm{P}(\sigma)$ holds, $c_0^{\mathcal{E}} \xrightarrow{\mathrm{Rules}(w)} \langle q, \sigma_c \rangle$ and $\sigma_s = \mathrm{output}(\sigma)$.
  - If $a \in \Sigma_u$, then considering $\sigma_s'$ as defined in Definition 6, $\sigma_t = \sigma_s.a.\sigma_s'$. Moreover, $a \in \Sigma_u$, thus rule pass-uncont can be applied: there exists $q' \in Q$ such that $q \xrightarrow{a} q'$, and then $\langle q, \sigma_c \rangle \xrightarrow{a/\,\text{pass-uncont}(a)/a} \langle q', \sigma_c \rangle$. If $\sigma_s' \neq \epsilon$, then $\sigma_t = \sigma_s.a.\sigma_s' \models \mathrm{Enf}(\varphi)$, meaning that there exists $q'' \in \mathrm{Q}_{\mathrm{enf}}$ such that $q' \xrightarrow{\sigma_s'} q''$, and then rule $\mathrm{dump}$ can be applied: $\langle q', \sigma_c \rangle \xrightarrow{\epsilon/\,\mathrm{dump}(\sigma_s')/\sigma_s'} \langle q'', \sigma_s'^{-1}.\sigma_c \rangle$. Thus, $\mathrm{output}(\sigma.a) = \mathrm{output}(\sigma).a.\sigma_s' = \sigma_s.a.\sigma_s' = \sigma_t$, $q'' = \mathrm{Reach}(\sigma_t)$, and $\sigma_d = \sigma_s'^{-1}.\sigma_c$, meaning that $\mathrm{P}(\sigma.a)$ holds. Otherwise, $\sigma_s' = \epsilon$, and then it is impossible to apply $\mathrm{dump}$ rule, because it is impossible to reach $\mathrm{Q}_{\mathrm{enf}}$ with a prefix of $\sigma_c$. Thus, $\mathrm{output}(\sigma.a) = \mathrm{output}(\sigma).a = \sigma_s.a = \sigma_t$, $q' = \mathrm{Reach}(\sigma_t)$, and $\sigma_d = \sigma_c$. Thus, $\mathrm{P}(\sigma.a)$ holds.
    Thus, if $a \in \Sigma_u$, $\mathrm{P}(\sigma.a)$ holds.
  - Otherwise, $a \in \Sigma_c$, and then, considering $\sigma_s''$ as defined in Definition 6, $\sigma_t = \sigma_s.\sigma_s''$. Since $a \in \Sigma_c$, it is possible to apply the store-cont rule: $\langle q, \sigma_c \rangle \xrightarrow{a/\,\text{store-cont}(a)/\epsilon} \langle q, \sigma_c.a \rangle$. Then, if $\sigma_s'' = \sigma_c.a$, it is possible to apply the $\mathrm{dump}$ rule. Therefore, there exists $q' \in \mathrm{Q}_{\mathrm{enf}}$ such that $q \xrightarrow{\sigma_c.a} q'$, and then $\langle q, \sigma_c.a \rangle \xrightarrow{\epsilon/\,\mathrm{dump}(\sigma_c.a)/\sigma_c.a} \langle q', \epsilon \rangle$. Thus, $\mathrm{output}(\sigma.a) = \mathrm{output}(\sigma).\sigma_c.a = \sigma_s.\sigma_c.a = \sigma_t$, $q' = \mathrm{Reach}(\sigma_t)$, and $\sigma_d = \sigma_s''^{-1}.(\sigma_c.a) = \epsilon$, meaning that $\mathrm{P}(\sigma.a)$ holds.
    Otherwise, $\sigma_s'' = \epsilon$, and no rule can be applied anymore, thus $\mathrm{output}(\sigma.a) = \mathrm{output}(\sigma) = \sigma_s = \sigma_t$, $q = \mathrm{Reach}(\sigma_t)$, and $\sigma_d = \sigma_s''^{-1}.(\sigma_c.a) = \sigma_c.a$. Thus, $\mathrm{P}(\sigma.a)$ holds.
    Thus, if $a \in \Sigma_u$, $\mathrm{P}(\sigma.a)$ holds.

  Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.a)$.

Thus, by induction on $\sigma$, $\forall \sigma \in \Sigma^*, \mathrm{P}(\sigma)$ holds, meaning that $\forall \sigma \in \Sigma^*, \mathrm{E}_\varphi(\sigma) = \mathrm{output}(\sigma)$.

### A.2 Proofs for the Timed Setting

In this section, we use the notations from section 4, meaning that $\varphi$ is a timed property, represented by a deterministic and complete timed automaton $\mathcal{A}_\varphi = \langle L, l_0, X, \Sigma, \Delta, G \rangle$ with semantics $[\![\mathcal{A}_\varphi]\!] = \langle Q, q_0, \Gamma, \to, F_G \rangle$.

**Lemma 4.** $\forall \sigma \in \mathrm{tw}(\Sigma), \forall (t,a)$ *such that* $\sigma.(t,a) \in \mathrm{tw}(\Sigma), (\sigma \not\models \mathrm{Pre}(\varphi) \wedge \sigma.(t,a) \models \mathrm{Pre}(\varphi)) \implies \sigma.(t,a) \models \varphi'$*, with* $\varphi' = \langle Q, q_0, \Gamma, \rightarrow, \mathrm{Q_{enf}} \rangle$.

*Proof.* Let us consider $\sigma \in \mathrm{tw}(\Sigma)$, and $(t,a)$ such that $\sigma \not\models \mathrm{Pre}(\varphi) \wedge \sigma.(t,a) \models \mathrm{Pre}(\varphi)$. Let us note $\rightarrow_1 = \rightarrow \cap (\mathrm{Q_{\overline{enf}}} \times \Gamma \times Q)$, $\rightarrow_2 = \{(q,a,q) \mid q \in \mathrm{Q_{enf}} \wedge a \in \Gamma\}$, and $\rightarrow_3 = \rightarrow_1 \cup \rightarrow_2$, so that $\mathrm{Pre}(\varphi) = \langle Q, q_0, \Gamma, \rightarrow_3, \mathrm{Q_{enf}} \rangle$.

Then, $\sigma \not\models \mathrm{Pre}(\varphi)$, thus $\forall \sigma' \preccurlyeq \sigma, \sigma' \not\models \mathrm{Pre}(\varphi)$, since $\mathrm{Pre}(\varphi)$ is a co-safety property. So, $\nexists q \in \mathrm{Q_{enf}}, q_0 \xrightarrow{\sigma'} q$, thus $q_0 \xrightarrow{\sigma}_1 q$. Moreover, since $\sigma \not\models \mathrm{Pre}(\varphi)$, $q \notin \mathrm{Q_{enf}}$, so $\exists q' \in Q$, $q \xrightarrow{(t,a)}_1 q'$. Since $\sigma.(t,a) \models \mathrm{Pre}(\varphi)$, $q' \in \mathrm{Q_{enf}}$. Thus, because $\rightarrow_1 \subseteq \rightarrow$, $q_0 \xrightarrow{\sigma.(t,a)} q'$, meaning that $\sigma.(t,a) \models \varphi'$.

**Lemma 5.** $\forall \sigma \in \mathrm{tw}(\Sigma), \sigma \models \varphi' \implies \epsilon \in \mathrm{Safe}(\sigma)$*, with* $\varphi' = \langle Q, q_0, \Gamma, \rightarrow, \mathrm{Q_{enf}} \rangle$.

*Proof.* Let us consider $\sigma \in \mathrm{tw}(\sigma)$ such that $\sigma \models \varphi'$. Then $q = \mathrm{Reach}(\sigma) \in \mathrm{Q_{enf}}$, thus if $w$ is such that $\Pi_\Sigma(w_{|\Sigma_c}) \preccurlyeq \epsilon$, then $w \in \mathrm{tw}(\Sigma_u)$, and $w$ satisfies $\Pi_\Sigma(w_{|\Sigma_c}) \preccurlyeq \Pi_\Sigma(w_{|\Sigma_c}) \preccurlyeq \epsilon \wedge \forall i \in [1; |w_{|\Sigma_c}|], \mathrm{date}(w_{|\Sigma_c}(i)) \leq \mathrm{date}(w_{|\Sigma_c}(i)) \wedge w_{|\Sigma_u} = w_{\Sigma_u}$. Moreover, since $q \in \mathrm{Q_{enf}}$, and $w \in \mathrm{tw}(\Sigma_u)$, $\exists q' \in \mathrm{Q_{enf}}, q \xrightarrow{w} q'$. Thus $\epsilon \in \mathrm{Safe}(q) = \mathrm{Safe}(\sigma)$.

**Lemma 6.** $\forall \sigma \in \mathrm{tw}(\Sigma), (\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma) \implies ((\sigma_s.\sigma_b \not\models \mathrm{Pre}(\varphi)) \implies \sigma_b = \epsilon)$.

*Proof.* Let us consider the predicate $\mathrm{P}(\sigma)$ : "$(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma) \implies ((\sigma_s.\sigma_b \not\models \mathrm{Pre}(\varphi)) \implies \sigma_b = \epsilon)$". Let us prove that $\forall \sigma \in \mathrm{tw}(\Sigma), \mathrm{P}(\sigma)$.

- $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon, \epsilon)$, thus $\mathrm{P}(\epsilon)$.
- Suppose now that for some $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$, and let us consider $(t,a)$ such that $\sigma.(t,a) \in \mathrm{tw}(\Sigma)$, $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma)$, and $(\sigma_t, \sigma_d, \sigma_e) = \mathrm{store}_\varphi(\sigma.(t,a))$.
  - If $\sigma_t.\sigma_d \models \mathrm{Pre}(\varphi)$, then $\mathrm{P}(\sigma.(t,a))$.
  - If $\sigma_t.\sigma_d \not\models \mathrm{Pre}(\varphi)$, then if $\sigma_d \neq \epsilon$, and $q = \mathrm{Reach}(\sigma_t)$ then there exists $q' \in \mathrm{Q_{enf}}$ such that $q \xrightarrow{\sigma_d} q'$, meaning that $\sigma_t.\sigma_d \models \mathrm{Pre}(\varphi)$, which is absurd. Thus $\sigma_d = \epsilon$, meaning that $\mathrm{P}(\sigma.(t,a))$ holds.

  Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.(t,a))$.

By induction, $\forall \sigma \in \mathrm{tw}(\Sigma), (\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma) \implies (\sigma_s.\sigma_b \not\models \mathrm{Pre}(\varphi) \implies \sigma_b = \epsilon)$.

**Proposition 4.** $\mathrm{E}_\varphi$ *is sound with respect to* $\mathrm{Pre}(\varphi) \implies \varphi$*, as per Definition 9.*

*Proof.* Let $\varphi' = \langle Q, q_0, \Gamma, \rightarrow, \mathrm{Q_{enf}} \rangle$. We then show that $\mathrm{E}_\varphi$ is correct with respect to $\mathrm{Pre}(\varphi) \implies \varphi'$. Let $\mathrm{P}(\sigma)$ be the predicate: "$((\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma)) \implies ((\sigma_s.\sigma_b \models \mathrm{Pre}(\varphi)) \implies (\Pi_\Sigma(\sigma_b) \in \mathrm{Safe}(\sigma_s) \wedge \sigma_s.\sigma_b \models \varphi'))$". Let us prove that $\forall \sigma \in \mathrm{tw}(\Sigma), \mathrm{P}(\sigma)$ holds.

- *Induction basis*: $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon, \epsilon)$. If $\epsilon \models \mathrm{Pre}(\varphi)$, then $q_0 \in \mathrm{Q_{enf}}$, thus $\epsilon \models \varphi'$, and, using lemma 5, $\epsilon \in \mathrm{Safe}(\epsilon)$. Thus $\mathrm{P}(\epsilon)$ holds.

– *Induction step*: Suppose now that, for some $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Let us consider $(t,a) \in \mathbb{R}_{\geq 0} \times \Sigma$ such that $\sigma.(t,a) \in \mathrm{tw}(\Sigma)$, $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma)$ and $(\sigma_t, \sigma_d, \sigma_e) = \mathrm{store}_\varphi(\sigma.(t,a))$.
  - If $\sigma_t.\sigma_d \not\models \mathrm{Pre}(\varphi)$, then $\mathrm{P}(\sigma.(t,a))$ holds.
  - If $\sigma_s.\sigma_b \not\models \mathrm{Pre}(\varphi)$ and $\sigma_t.\sigma_d \models \mathrm{Pre}(\varphi)$:
    * If $a \in \Sigma_u$, then if $\sigma_d \neq \epsilon$ then $\mathrm{P}(\sigma.(t,a))$ holds, following the construction of $\sigma_d$. Otherwise, $\sigma_d = \epsilon$, and then, since $\sigma_s.\sigma_b \not\models \mathrm{Pre}(\varphi)$, using lemma 6, $\sigma_b = \epsilon$. Thus $\sigma_s.\sigma_b = \sigma_s \not\models \mathrm{Pre}(\varphi)$, and $\sigma_t = \sigma_s.\mathrm{obs}(\sigma_b, t).(t,a) = \sigma_s.(t,a)$. So, $\sigma_t.\sigma_d = \sigma_t = \sigma_s.(t,a)$. Since $\sigma_s \not\models \mathrm{Pre}(\varphi)$, and $\sigma_t.\sigma_d = \sigma_s.(t,a) \models \mathrm{Pre}(\varphi)$, then, using lemma 4, $\sigma_t.\sigma_d \models \varphi'$, and then, using lemma 5 $\sigma_d = \epsilon \in \mathrm{Safe}(\sigma_t)$. Thus $\mathrm{P}(\sigma.(t,a))$ holds.
    * If $a \in \Sigma_c$, then either $\sigma_d \neq \epsilon$, and then $\mathrm{P}(\sigma.(t,a))$ holds (following the construction of $\sigma_d$), or $\sigma_d = \epsilon$, in which case $\sigma_t.\sigma_d = \sigma_t = \sigma_s.\mathrm{obs}(\sigma_b, t)$. Since $\sigma_s.\sigma_b \not\models \mathrm{Pre}(\varphi)$, then, using lemma 4, $\sigma_b = \epsilon$, thus $\sigma_t = \sigma_s$. Thus, $\sigma_t.\sigma_d = \sigma_t \models \mathrm{Pre}(\varphi)$, and $\sigma_t = \sigma_s \not\models \mathrm{Pre}(\varphi)$, which is absurb, thus $\sigma_d \neq \epsilon$ in this case.
  - If $\sigma_s.\sigma_b \models \mathrm{Pre}(\varphi)$ and $\sigma_t.\sigma_d \models \mathrm{Pre}(\varphi)$:
    * If $a \in \Sigma_u$, then either $\sigma_d \neq \epsilon$, and then $\mathrm{P}(\sigma.(t,a))$ holds (by construction of $\sigma_d$), or $\sigma_d = \epsilon$, leading to $\sigma_t.\sigma_d = \sigma_t = \sigma_s.\mathrm{obs}(\sigma_b, t).(t,a)$. Since $\mathrm{P}(\sigma)$ holds and $\sigma_s.\sigma_b \models \mathrm{Pre}(\varphi)$, $\sigma_b \in \mathrm{Safe}(\sigma_s)$, meaning that if $q_i = \mathrm{Reach}(\sigma_s), \exists w \in \mathrm{tw}(\Sigma), \mathrm{obs}(\sigma_b, t).(t,a) \preccurlyeq w \wedge \Pi_\Sigma(w_{|\Sigma_c}) \preccurlyeq \sigma_b \wedge w_{|\Sigma_u} = \mathrm{obs}(\sigma_b, t)_{|\Sigma_u}.(t,a) \wedge \exists q \in \mathrm{Q}_{\mathrm{enf}}, q_i \xrightarrow{w} q$, (because $\Pi_\Sigma((\mathrm{obs}(\sigma_b, t).(t,a))_{|\Sigma_c}) \preccurlyeq \Pi_\Sigma(\sigma_b)$). Since $\sigma_d = \epsilon, w = \mathrm{obs}(\sigma_b, t).(t,a)$. Thus $q_0 \xrightarrow{\sigma_s} q_i \xrightarrow{\mathrm{obs}(\sigma_b,t).(t,a)} q$,meaning that $q = \mathrm{Reach}(\sigma_t) \in \mathrm{Q}_{\mathrm{enf}}$. Thus $\sigma_t.\sigma_d = \sigma_t \models \varphi'$, thus, using lemma 5, $\sigma_d = \epsilon \in \mathrm{Safe}(\sigma_t)$. Thus $\mathrm{P}(\sigma.(t,a))$ holds.
    * If $a \in \Sigma_c$, then either $\sigma_d \neq \epsilon$ and $\mathrm{P}(\sigma.(t,a))$ holds (following the construction of $\sigma_d$), or $\sigma_d = \epsilon$ and then $\sigma_t.\sigma_d = \sigma_t = \sigma_s.\mathrm{obs}(\sigma_b, t)$. Since $\mathrm{P}(\sigma)$ holds and $\sigma_s.\sigma_b \models \mathrm{Pre}(\varphi)$, $\Pi_\Sigma(\sigma_b) \in \mathrm{Safe}(\sigma_s)$. Thus, if $q_i = \mathrm{Reach}(\sigma_s)$, then there exists $w \in \mathrm{tw}(\Sigma)$ such that $\mathrm{obs}(\sigma_b, t) \preccurlyeq w \wedge \Pi_\Sigma(w_{|\Sigma_c}) \preccurlyeq \Pi_\Sigma(\sigma_b) \wedge w_{|\Sigma_u} = \epsilon \wedge \exists q \in \mathrm{Q}_{\mathrm{enf}}, q_i \xrightarrow{w} q$. Since $\sigma_d = \epsilon$, $w = \mathrm{obs}(\sigma_b, t)$, so $\sigma_t \models \varphi'$, thus, using lemma 5 $\sigma_d = \epsilon \in \mathrm{Safe}(\sigma_t)$. Thus $\mathrm{P}(\sigma.(t,a))$ holds.

  In all cases, $\mathrm{P}(\sigma.(t,a)$ holds.

Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.(t,a))$.

By induction, $\forall \sigma \in \mathrm{tw}(\Sigma), \mathrm{P}(\sigma)$ holds, thus $\forall \sigma \in \mathrm{tw}(\Sigma), ((\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma)) \implies (\sigma_s.\sigma_b \models \mathrm{Pre}(\varphi) \implies \varphi')$. Therefore, $\forall t \in \mathbb{R}_{\geq 0}, \exists t' \geq t, \mathrm{E}_\varphi(\sigma, t) \models \mathrm{Pre}(\varphi) \implies \varphi'$. Since $\varphi' \implies \varphi, \forall t \in \mathbb{R}_{\geq 0}, \exists t' \geq t, \mathrm{E}_\varphi(\sigma, t) \models \mathrm{Pre}(\varphi) \implies \varphi$. This means that $\mathrm{E}_\varphi$ is sound with respect to $\mathrm{Pre}(\varphi) \implies \varphi$.

**Proposition 5.** $\mathrm{E}_\varphi$ *is compliant, as per Definition 10.*

*Proof.* Let $\mathrm{P}(\sigma)$ be the predicate: "$(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma) \implies (\Pi_\Sigma(\sigma_{s|\Sigma_c}.\sigma_b).\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_c}) \wedge (\sigma_s.\sigma_b)_{|\Sigma_u} = \sigma_{|\Sigma_u}) \wedge \forall i \in [1; |\sigma_{s|\Sigma_c}.\sigma_b|], \mathrm{date}((\sigma_{s|\Sigma_c}.\sigma_b)(i)) \geq \mathrm{date}(\sigma_{|\Sigma_c}(i))$". Let us prove that $\forall \sigma \in \mathrm{tw}(\Sigma), \mathrm{P}(\sigma)$ holds.

– *Induction basis*: $\text{store}_\varphi(\epsilon) = (\epsilon, \epsilon, \epsilon)$, and $\epsilon_{|\Sigma_c} = \epsilon_{|\Sigma_u} = \epsilon$. Thus, $\text{P}(\epsilon)$ holds.
– *Induction step*: Suppose now that for some $\sigma \in \text{tw}(\Sigma)$, $\text{P}(\sigma)$ holds. Let us consider $(\sigma_s, \sigma_b, \sigma_c) = \text{store}_\varphi(\sigma)$, $(t, a)$ such that $\sigma.(t, a) \in \text{tw}(\Sigma)$, and $(\sigma_t, \sigma_d, \sigma_e) = \text{store}_\varphi(\sigma.(t, a))$.
  - If $a \in \Sigma_u$:
    * $(\sigma_t.\sigma_d)_{|\Sigma_u} = \sigma_{t|\Sigma_u}$ ($\sigma_d \in \text{tw}(\Sigma_c)$), thus $(\sigma_t.\sigma_d)_{|\Sigma_u} = \sigma_{s|\Sigma_u}.(t, a)$, because $\sigma_b \in \text{tw}(\Sigma_c)$, and $\sigma_c \in \Sigma_c^*$. Thus $(\sigma_t.\sigma_d)_{|\Sigma_u} = \sigma_{|\Sigma_u}.(t, a) = (\sigma.(t, a))_{|\Sigma_u}$, since $\text{P}(\sigma)$ holds and $a \in \Sigma_u$.
    * $\Pi_\Sigma(\sigma_{t|\Sigma_c}.\sigma_d).\sigma_e = \Pi_\Sigma(\sigma_{s|\Sigma_c}.\sigma_b).\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_c})$, since $\text{P}(\sigma)$ holds.
    * $\sigma_t = \sigma_s.\,\text{obs}(\sigma_b, t).(t, a)$, thus $\sigma_{t|\Sigma_c} = \sigma_s.\,\text{obs}(\sigma_b, t)$. Since $\text{P}(\sigma)$ holds, $\forall i \in [1; |\sigma_{t|\Sigma_c}|], \text{date}(\sigma_{t|\Sigma_c}(i)) \geq \text{date}(\sigma_{|\Sigma_c}(i))$. Moreover, by construction of $\sigma_d$, $\text{date}(\sigma_d(1)) \geq t$, meaning that $\forall i \in [1; |\sigma_d|], \text{date}(\sigma_d(i)) \geq t$. Thus, $\forall i \in [1; |\sigma_{t|\Sigma_c}.\sigma_d|], \text{date}((\sigma_{t|\Sigma_c}.\sigma_d)(i)) \geq \text{date}((\sigma.(t, a))_{|\Sigma_c}(i))$.

    Thus $\text{P}(\sigma.(t, a))$ holds.
  - If $a \in \Sigma_c$:
    * $(\sigma_t.\sigma_d)_{|\Sigma_u} = \sigma_{s|\Sigma_u} = \sigma_{|\Sigma_u}$, because $\sigma_b \in \text{tw}(\Sigma_c)$, $\sigma_d \in \text{tw}(\Sigma_c)$ and $\text{P}(\sigma)$ holds.
    * $\Pi_\Sigma(\sigma_{t|\Sigma_c}.\sigma_d).\sigma_e = \Pi_\Sigma(\sigma_{s|\Sigma_c}.\sigma_b).\sigma_c.a = \Pi_\Sigma(\sigma_{|\Sigma_c}).a = \Pi_\Sigma((\sigma.(t, a))_{|\Sigma_c})$.
    * $\sigma_t = \sigma_s.\,\text{obs}(\sigma_b, t)$. Since $\text{P}(\sigma)$ holds, $\forall i \in [1; |\sigma_t|], \text{date}(\sigma_{t|\Sigma_c}(i)) \geq \text{date}((\sigma.(t, a))_{|\Sigma_c}(i))$. Moreover, by construction of $\sigma_d$, $\forall i \in [1; |\sigma_d|], \text{date}(\sigma_d(i)) \geq t$. Thus, $\forall i \in [1; |\sigma_{t|\Sigma_c}.\sigma_d|], \text{date}((\sigma_{t|\Sigma_c}.\sigma_d)(i)) \geq \text{date}((\sigma.(t, a))_{|\Sigma_c}(i))$.

    Thus $\text{P}(\sigma.(t, a))$ holds.

  Thus $\text{P}(\sigma) \implies \text{P}(\sigma.(t, a))$.

By induction, $\forall \sigma \in \text{tw}(\Sigma), (\sigma_s, \sigma_b, \sigma_c) = \text{store}_\varphi(\sigma) \implies \Pi_\Sigma(\sigma_{s|\Sigma_c}.\sigma_b).\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_c}) \wedge (\sigma_s.\sigma_b)_{|\Sigma_u} = \sigma_{|\Sigma_u} \wedge \forall i \in [1; |\sigma_{s|\Sigma_c}.\sigma_b|], \text{date}((\sigma_{s|\Sigma_c}.\sigma_b)(i)) \geq \text{date}(\sigma_{|\Sigma_c}(i))$. Thus, $(\sigma_s.\sigma_b)_{|\Sigma_c} \preccurlyeq_d \sigma_{|\Sigma_c} \wedge (\sigma_s.\sigma_b)_{|\sigma_u} = \sigma_{|\Sigma_u}$.

Thus $\text{E}_\varphi$ is compliant.

**Proposition 6.** *The output of $\mathcal{E}$ for input $\sigma$ is* $\text{E}_\varphi(\sigma)$.

*Proof.* For this proof, let us introduce some notation. For a word $w \in \Gamma^{\mathcal{E}*}$, we note $\text{input}(w) = \Pi_1(w(1)).\Pi_1(w(2)) \dots \Pi_1(w(|w|))$ the concatenation of all inputs from $w$. In the same way, we define $\text{output}(w) = \Pi_3(w(1)).\Pi_3(w(2)) \dots \Pi_3(w(|w|))$ the concatenation of all outputs from $w$. Let us also define function Rules which, given a timed word and a date, give the longest sequence of rules that can be applied with the given word as input at the date given:

$$\text{Rules} : \begin{cases} \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0} \to \Gamma^{\mathcal{E}} \\ \quad (\sigma, \quad t) \quad \mapsto \max_{\preccurlyeq}(\{\, w \in \Gamma^{\mathcal{E}} \mid \text{input}(w) = \sigma \wedge \exists c \in C^{\mathcal{E}}, \\ \qquad\qquad c_0^{\mathcal{E}} \xrightarrow{w} c \wedge \Pi_4(c) = t\}) \end{cases}$$

Since time is not discrete, the rule delay can be applied an infinite number of times by slicing time. Thus, we consider that the rule delay is always applied a minimum number of times, i.e., when two rules delay are consecutive, they are merged into one rule delay, whose parameter is the sum of the parameters of the two rules. The runs

obtained are equivalent, but it allows to consider the maximum (for prefix order) of the set used in the definition of $\mathrm{Rules}$.

We then extend $\mathrm{output}$ to timed words with a date: for $\sigma \in \mathrm{tw}(\Sigma)$, and a date $t$, $\mathrm{output}(\sigma, t) = \mathrm{output}(\mathrm{Rules}(\sigma, t))$.

Let $\mathrm{P}(\sigma)$ be the predicate: "$\forall t \in \mathbb{R}_{\geq 0}, \mathrm{E}_\varphi(\sigma, t) = \mathrm{output}(\sigma, t) \wedge (((\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma) \wedge c_0^{\mathcal{E}} \xrightarrow{\mathrm{Rules}(\sigma,t)} c \wedge \langle l, v\rangle = \mathrm{Reach}(\mathrm{E}_\varphi(\sigma, t))) \implies \Pi_1(c) = \mathrm{nobs}(\sigma_b, t) \wedge \Pi_2(c) = \sigma_c \wedge \Pi_3(c) = \langle l, v + t - \mathrm{time}(\mathrm{E}_\varphi(\sigma, t))\rangle)$". Let us then prove that for all $\sigma \in \mathrm{tw}(\Sigma), \mathrm{P}(\sigma)$ holds.

- *Induction basis*: For $\sigma = \epsilon$, let us consider $t \in \mathbb{R}_{\geq 0}$. Then, $\mathrm{E}_\varphi(\epsilon, t) = \epsilon$, and $\mathrm{Reach}(\mathrm{E}_\varphi(\epsilon, t)) = q_0 = \langle l_0, v_0\rangle$. The only rule that can be applied is $\mathrm{delay}$, since there is not any input, nor any element to dump. Thus, $\mathrm{Rules}(\epsilon, t) = \epsilon / \mathrm{delay}(t)/\epsilon$. Let us consider $c \in C^{\mathcal{E}}$ such that $c_0^{\mathcal{E}} \xrightarrow{\mathrm{Rules}(\epsilon,t)} c$. Then, $c = \langle \epsilon, \epsilon, \langle l_0, v_0 + t\rangle, t\rangle$, thus, if $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma) = (\epsilon, \epsilon, \epsilon)$, then $\Pi_1(c) = \epsilon = \mathrm{nobs}(\sigma_b, t)$, $\Pi_2(c) = \epsilon = \sigma_c$, and $\Pi_3(c) = \langle l_0, v_0 + t\rangle = \langle l_0, v_0 + t - \mathrm{time}(\epsilon)\rangle$. Moreover, $\mathrm{output}(\sigma, t) = \epsilon = \mathrm{E}_\varphi(\sigma, t)$. Thus, $\mathrm{P}(\epsilon)$ holds.
- *Induction step*: Let us suppose now that for some $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Let us consider $(t, a) \in \mathbb{R}_{\geq 0} \times \Sigma$ such that $\sigma.(t, a) \in \mathrm{tw}(\Sigma)$. Let us then prove that $\mathrm{P}(\sigma.(t, a))$ holds. Let us consider $t' \in \mathbb{R}_{\geq 0}$, $c \in C^{\mathcal{E}}$ such that $c_0^{\mathcal{E}} \xrightarrow{\mathrm{Rules}(\sigma,t)} c$, $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma)$, and $(\sigma_t, \sigma_d, \sigma_e) = \mathrm{store}_\varphi(\sigma.(t, a))$. If $t' < t$, then the rules applied at date $t'$ with input $\sigma.(t, a)$ are the same as the ones applied with input $\sigma$, and $\mathrm{E}_\varphi(\sigma.(t, a), t') = \mathrm{E}_\varphi(\sigma, t')$, thus $\mathrm{P}(\sigma.(t, a))$ holds. Thus, in the following, we will consider that $t' \geq t$:
  - If $a \in \Sigma_u$, rule $\mathrm{pass\text{-}uncont}$ can be applied. Let us consider $c' = c\,\mathrm{after}((t, a)/\,\mathrm{pass\text{-}uncont}((t, a))/(t, a))$. Then, $c' = \langle \sigma_d, \sigma_e, q', t\rangle$, with $\Pi_3(c) \xrightarrow{(t,a)} q'$, since $\mathrm{update}$ computes $\kappa_\varphi$ from Definition 13. Then, $\mathrm{obs}(\sigma_d, t')$ can be dumped, by applying $\mathrm{delay}$ followed by $\mathrm{dump}$ for each event, followed by a last $\mathrm{delay}$ rule to reach date $t'$. Thus, if $c_f = c_0^{\mathcal{E}}\,\mathrm{after}\,\mathrm{Rules}(\sigma.(t, a), t')$, then $\Pi_1(c_f) = \mathrm{nobs}(\sigma_d, t')$ (what remains after having dumped $\mathrm{obs}(\sigma_d, t')$), $\Pi_2(c_f) = \sigma_e$, since the second member does not change when applying rules $\mathrm{delay}$ and $\mathrm{dump}$, and $\Pi_3(c_f) = \langle l, v + t' - \mathrm{time}(\sigma_s.(t, a).\,\mathrm{obs}(\sigma_d, t'))\rangle = \langle l, v + t' - \mathrm{time}(\mathrm{E}_\varphi(\sigma.(t, a)))\rangle$, with $\langle l, v\rangle = \mathrm{Reach}(\sigma_s.(t, a).\,\mathrm{obs}(\sigma_d, t')) = \mathrm{Reach}(\mathrm{E}_\varphi(\sigma.(t, a)))$, since $\Pi_3(c) \xrightarrow{(t,a).\,\mathrm{obs}(\sigma_d,t')} \Pi_3(c_f)$ and $\Pi_3(c) = \mathrm{Reach}(\mathrm{E}_\varphi(\sigma, t))$. Moreover, $\mathrm{output}(\sigma.(t, a), t') = \sigma_s.\,\mathrm{obs}(\sigma_b, t).(t, a).\,\mathrm{obs}(\sigma_d, t') = \mathrm{output}(\sigma, t).(t, a).\,\mathrm{obs}(\sigma_d, t') = \mathrm{E}_\varphi(\sigma, t).(t, a).\,\mathrm{obs}(\sigma_d, t') = \mathrm{E}_\varphi(\sigma.(t, a), t')$. Thus, $\mathrm{P}(\sigma.(t, a))$ holds.
  - If $a \in \Sigma_c$, rule $\mathrm{store\text{-}cont}$ can be applied. Let us consider $c' = c\,\mathrm{after}((t, a)/\,\mathrm{store\text{-}cont}(a)/\epsilon)$. Then, $c' = \langle \sigma_d, \sigma_e, \Pi_3(c), t\rangle$, and then, like previously, $\mathrm{obs}(\sigma_d, t')$ can be dumped, by using rules $\mathrm{dump}$ and $\mathrm{delay}$, such that, if $c_f = c_0^{\mathcal{E}}\,\mathrm{after}\,\mathrm{Rules}(\sigma.(t, a), t')$, then $\Pi_1(c_f) = \mathrm{nobs}(\sigma_d, t)$, $\Pi_2(c_f) = \sigma_e$, and $\Pi_3(c_f) = \langle l, v + t' - \mathrm{time}(\sigma_s.\,\mathrm{obs}(\sigma_b, t).\,\mathrm{obs}(\sigma_d, t'))\rangle = \langle l, v + t' - \mathrm{time}(\mathrm{E}_\varphi(\sigma.(t, a), t'))\rangle$, with $\langle l, v\rangle = \mathrm{Reach}(\mathrm{E}_\varphi(\sigma, t))$. Moreover, $\mathrm{output}(\sigma.(t, a), t') = \mathrm{output}(\sigma, t).\,\mathrm{obs}(\sigma_d, t') = \mathrm{E}_\varphi(\sigma.t).\,\mathrm{obs}(\sigma_d, t') = \mathrm{E}_\varphi(\sigma.(t, a), t')$. Thus $\mathrm{P}(\sigma.(t, a))$ holds.

Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.(t,a))$.

Thus, by induction, for all $\sigma \in \mathrm{tw}(\Sigma), \mathrm{P}(\sigma)$ holds. In particular, for all $\sigma \in \mathrm{tw}(\Sigma)$, and for all $t \in \mathbb{R}_{\geq 0}$, $\mathrm{output}(\sigma, t) = \mathrm{E}_{\varphi}(\sigma, t)$, meaning that the output of the enforcement monitor $\mathcal{E}$ with input $\sigma$ at time $t$ is exactly the output of function $\mathrm{E}_{\varphi}$ with the same input and the same date.