Optimal Enforcement of (Timed) Properties with Uncontrollable Events
Matthieu Renard, Yliès Falcone, Antoine Rollet, Thierry Jéron, Hervé Marchand

To cite this version:

HAL Id: hal-01262444
https://hal.archives-ouvertes.fr/hal-01262444v4
Submitted on 9 May 2017

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

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Optimal Enforcement of (Timed) Properties with Uncontrollable Events

MATTHIEU RENARD, YLIÈS FALCONE, ANTOINE ROLLET,
THIERRY JÉRON, and HERVÉ MARCHAND

1 LaBRI, Bordeaux INP, Université Bordeaux, Bordeaux, France.
3 Inria Rennes Bretagne-Atlantique, Rennes, France.

Received April 2017

This paper deals with runtime enforcement of untimed and timed properties with uncontrollable events. Runtime enforcement consists in defining and using mechanisms that modify 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 described by a deterministic automaton 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 should satisfy important properties, namely soundness, compliance, and optimality - meaning that enforcement mechanisms should output as soon as possible correct executions that are as close as possible to the input execution. We define the conditions for a property to be enforceable with uncontrollable events. Moreover, we synthesise sound, compliant, and optimal descriptions of runtime enforcement mechanisms at two levels of abstraction to facilitate their design and implementation.

1. Introduction

Runtime verification (Leucker and Schallhart, 2009; Falcone et al., 2013) is a powerful technique which aims at checking the conformance of the executions of a system under scrutiny with respect to some specification. It consists in running a mechanism that assigns verdicts to a sequence of events produced by the instrumented system with respect to a property formalising the specification. This paper focuses on runtime enforcement (cf. (Schneider, 2000; Ligatti et al., 2009; Falcone et al., 2011; Basin et al., 2013)) 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, respectively. When dealing with timed properties, EMs can act as delayers over the input sequence of events (Pinisetty et al., 2012; Pinisetty et al., 2014b; Pinisetty et al., 2014c). That is, whenever possible, EMs buffer input events for some time and then release them in such a way that the output sequence of events satisfies the property. The general scheme is given in Fig. 1.
Motivations. We focus on enforcement of properties with uncontrollable events\(^1\). Introducing uncontrollable events is a step towards more realistic runtime enforcement. Uncontrollable events naturally occur in many application 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 critical domain, one usually adds control mechanisms in specific points of the architecture in order to verify that nothing wrong happens. Some events may only be observed by these mechanisms in order to decide if a situation is abnormal, but they cannot be acted upon, meaning that they are uncontrollable. For instance, the “spoiler activation”\(^2\) command triggered 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 spoilers prevents events leading to an incoherent state by blocking them, according to the pilot commands. The pilot commands are out of the scope of the EM, i.e. observable but uncontrollable. 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 from occurring in the monitored program.

Challenges. Considering uncontrollable events in the timed setting induces new challenges. Indeed, EMs may now receive events that cannot be buffered and have to be output immediately. Since uncontrollable events influence the satisfaction of the property under scrutiny, the dates of the controllable events stored in memory have to be recomputed upon the reception of each uncontrollable event to guarantee that the property is still satisfied after outputting them. Moreover, it is necessary to prevent the system from reaching a bad state upon reception of any sequence of uncontrollable events. Since uncontrollable events can occur at any time, the EM must take their potential reception into account when computing the sequence to be emitted. Then, 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. 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. Thus, new enforcement strategies are necessary for both untimed and timed properties.

Contributions. We introduce a framework for the enforcement monitoring of regular untimed and timed properties with uncontrollable events. We define EMs at two levels of abstraction. The synthesised EMs are sound, compliant and optimal. When considering uncontrollable events, it turns out that the usual notion of transparency has to be weakened. As we shall see, the initial

\(^1\) This notion of uncontrollable event should not be confused with the notion of uncontrollable transition used in some supervision and game theory.

\(^2\) The spoiler is a device used to reduce the lift of an aircraft.
order between uncontrollable and controllable events can change in output, contrary to what is
prescribed by transparency. Thus, we replace transparency with a new notion, namely compliance, prescribing 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 EM for any input sequence. In the timed setting, the executions are associated with dates from which the property is enforceable.

This paper revisits and extends a first approach in (Renard et al., 2015). Most definitions were modified to ensure optimality of the EMs for any regular property. Some definitions have been rewritten in a more formal, more modular, and clearer way. All the proofs of soundness, compliance, optimality and equivalence between the different descriptions of the enforcement mechanism are provided. This new framework can also be used without uncontrollable events.

Remark 1. There exist similarities between supervisory control theory (Ramadge and Wonham, 1987; Ramadge and Wonham, 1989) and runtime enforcement. For instance, a supervisor is usually implemented as a monitor deciding at runtime if a command should be activated or not. Supervisory control usually needs a model of the system, and consists in building a supervisor from this model by cutting forbidden states and transitions of uncontrollable events leading to them. Usually, an EM only uses a high-level property. In our work, an EM is equipped with a memory providing many more possibilities of actions, such as keeping and releasing events.

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. In each setting, we define enforcement mechanisms at two levels of abstraction. Section 5 discusses related work. Section 6 presents conclusions and perspectives. Proofs are in Appendix A.

2. Preliminaries and Notation

Untimed Notions. An alphabet is a finite, non-empty 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 \( L \subseteq \Sigma^* \). The concatenation of two words \( w \) and \( w' \) is noted \( w.w' \) (the dot is omitted when clear from the context). A word \( w' \) is a prefix of a word \( w \), noted \( w' \preceq w \) if there exists a word \( w'' \) s.t. \( w = w'.w'' \). The word \( w'' \) is called the residual of \( w \) after reading the prefix \( w' \), noted \( w'' = w^{-1}.w \). Note that \( w'.w'' = w'.w'' \cdot w = w \). These definitions are extended to languages in the natural way. A language \( L \subseteq \Sigma^* \) is extension-closed if for any words \( w \in L \) and \( w' \in \Sigma^* \), \( w.w' \in L \). Given a word \( w \) and an integer \( i \) s.t. \( 1 \leq i \leq |w| \), 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 \( I_i(e) \), the projection on the \( i \)-th coordinate, i.e. \( I_i(e) = e_i \). The tuple \( (e_1, e_2, \ldots, e_n) \) is sometimes noted \( (e_1, e_2, \ldots, e_n) \) in order to help reading. It can be used, for example, if a tuple contains a tuple. 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 \( \forall \sigma \in \Sigma^*, \forall a \in \Sigma, (w.a)|_{\Sigma'} = w|_{\Sigma'} \cdot a \) if \( a \in \Sigma' \), and \( (w.a)|_{\Sigma} = w|_{\Sigma} \) otherwise. We also note \( =_{w'} \) the equality of the restrictions of two words to \( \Sigma' \); if \( \sigma \) and \( \sigma' \) are two words, \( \sigma =_{w'} \sigma' \) if \( \sigma|_{\Sigma'} = \sigma'|_{\Sigma'} \). We define in the same way \( \leq_{w'} \sigma \leq_{w'} \sigma' \) if \( \sigma|_{\Sigma'} \leq \sigma'|_{\Sigma'} \).
Automata. An automaton is a tuple \((Q, q_0, \Sigma, \rightarrow, F)\), where \(Q\) is the set of states, \(q_0 \in Q\) is the initial state, \(\Sigma\) is the alphabet, \(\rightarrow \subseteq Q \times \Sigma \times Q\) is the transition relation and \(F \subseteq Q\) is the set of accepting states. Whenever \((q, a, q') \in \rightarrow\), we note it \(q \xrightarrow{a} q'\). Relation \(\rightarrow\) is extended to words \(\sigma \in \Sigma^*\) by noting \(q \xrightarrow{\sigma} q'\) whenever there exists \(q''\) s.t. \(q \xrightarrow{\sigma_1} q''\) and \(q'' \xrightarrow{\sigma_2} q'\). Moreover, for any \(q \in Q\), \(q \xrightarrow{\varepsilon} q\) always holds. An automaton \(A = (Q, q_0, \Sigma, \rightarrow, F)\) is deterministic if \(\forall q \in Q, \forall a \in \Sigma, (q \xrightarrow{a} q' \land q \xrightarrow{a} q'') \implies q' = q''\). 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 \(A\) if there exists \(q \in F\) such that \(q_0 \xrightarrow{w} q\). The language (i.e. set of all words) accepted by \(A\) is denoted by \(L(A)\). A property is a language over an alphabet. A regular property is a language accepted by an automaton. In the sequel, we assume that a property \(\varphi\) is represented by a deterministic and complete automaton \(A_\varphi\). For example, in Fig. 2, \(Q = \{q_0, q_1, q_2, q_3\}\), the initial state is \(q_0\), \(\Sigma = \{\text{Auth, LockOff, LockOn, Write}\}\), \(F = \{q_1, q_2\}\), and the transition relation \(\rightarrow\) contains for instance \((q_0, \text{Auth, } q_1), (q_1, \text{LockOn, } q_2)\), and \((q_3, \text{LockOn, } q_3)\).

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 \(\text{date}(t, a) = t\) and \(\text{act}(t, a) = a\) the projections of dates on events 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 \(\forall i \in [1; |\sigma| - 1], \text{date}(w(i)) \leq \text{date}(w(i + 1))\). \(\text{tw}(\Sigma)\) 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 \(i\) s.t. \(1 \leq i \leq n\), \(t_i\) is the time elapsed before action \(a_i\) occurs. We naturally extend the notions of prefix and residual to timed words. We note \(\text{time}(\sigma) = \text{date}(\sigma(|\sigma|))\) for \(\sigma \neq \varepsilon\), and \(\text{time}(\varepsilon) = 0\). We define the observation of \(\sigma\) at time \(t\) as the timed word \(\text{obs}(\sigma, t) = \max_{\epsilon \geq t}\{\{\sigma' | \sigma' \leq \sigma \land \text{time}(\sigma') \leq t\}\}\), corresponding to the word that would be observed at date \(t\) if events were received at the date they are associated with. We also define the remainder of the observation of \(\sigma\) at time \(t\) as \(\text{nobs}(\sigma, t) = (\text{obs}(\sigma, t))^{-1}\sigma\), which corresponds to the events that are to be received after date \(t\). 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 \(\sigma + t\). \(s.t.\) \(t\) is added to all dates: \(\sigma + t = (t_1 + t, a_1), (t_2 + t, a_2), \ldots, (t_n + t, a_n)\). Similarly, we define \(\sigma - t\), when \(t_1 \geq t\), to be the word \((t_1 - t, a_1), (t_2 - t, a_2), \ldots, (t_n - t, a_n)\). We also extend the definition of the restriction of \(\sigma\) to \(\Sigma' \subseteq \Sigma\) to timed words, s.t. \(\epsilon_{\Sigma'} = \varepsilon\), and for \(\sigma \in \text{tw}(\Sigma)\) and \((t, a)\) s.t. \(\sigma(t, a) \in \text{tw}(\Sigma), (\sigma(t, a))_{|\Sigma'} = \sigma|_{|\Sigma'}(t, a)\) if \(a \in \Sigma'\), and \((\sigma(t, a))_{|\Sigma'} = \sigma|_{|\Sigma'}\) otherwise. The notations \(\varepsilon_{|\Sigma'}\) and \(\subseteq_{|\Sigma'}\) are then naturally extended to timed words. A timed language is any subset of \(\text{tw}(\Sigma)\). The notion of extension-closed languages is naturally extended to timed languages. We also extend the notion of extension-closed languages to sets of elements composed of a timed word and a date: a set \(S \subseteq \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0}\) is time-extension-closed if for any \((\sigma, t) \in S\), for all \(w \in \text{tw}(\Sigma)\) s.t. \(\sigma.w \in \text{tw}(\Sigma)\), for all \(t' \geq t\), \((\sigma.w, t') \in S\). In other words, \(S\) is time-extension-closed if for every \(\sigma \in \text{tw}(\Sigma)\), there exists a date \(t\) from which \(\sigma\) and all its extensions are in \(S\), that is, associated with a date greater or equal to \(t\). Moreover, we define an order on timed words: we say that \(\sigma'\) is a delayed prefix of \(\sigma\), noted \(\sigma' \leq_d \sigma\), whenever \(\Pi_\Sigma(\sigma') \leq \Pi_\Sigma(\sigma)\) and \(\forall i \in [1; |\sigma'| - 1], \text{date}(\sigma(i)) \leq \text{date}(\sigma'(i))\).
Note that the order is not the same in the different constraints: $\Pi_\Sigma(\sigma')$ is a prefix of $\Pi_\Sigma(\sigma)$, but dates in $\sigma'$ exceed dates in $\sigma$. As for the equality $=\Pi_\Sigma(\sigma)$, we note $\sigma' \leq_{\text{lex}}^* \sigma$ whenever $\sigma'_|_{\Sigma'} \leq_{\text{lex}} \sigma|_{\Sigma'}$. We also define a *lexicographical order* $\leq_{\text{lex}}$ on timed words with identical untimed projections, s.t. $\epsilon \leq_{\text{lex}} \epsilon$, and for two words $\sigma$ and $\sigma'$ s.t. $\Pi_\Sigma(\sigma) = \Pi_\Sigma(\sigma')$, and two events $(t, a)$ and $(t', a), \sigma' \leq_{\text{lex}} (t, a) \sigma$ if $t' < t \lor (t = t' \land \sigma' \leq_{\text{lex}} \sigma)$.

Consider for example the timed word $\sigma = (1, a), (2, b), (3, c), (4, a)$ over the alphabet $\Sigma = \{a, b, c\}$. Then, $\Pi_\Sigma(\sigma) = a.b.c.a, \text{obs}(\sigma, 3) = (1, a), (2, b), (3, c), \text{obs}(\sigma, 3) = (4, a)$, and if $\Sigma' = \{b, c\}, \sigma|_{\Sigma'} = (2, b), (3, c)$, and for instance $(1, a), (2, b), (4, c) \leq_{\text{lex}} (1, a), (3, b), (3, c), (3, a)$. Moreover, if $w = (1, a), (2, b)$, then $w^{-1} \sigma = (3, c), (4, a)$.

*Timed Automata.* Let $X = \{X_1, X_2, \ldots, X_n\}$ be a finite set of *clocks*, i.e. variables that increase regularly with time. 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 $V(X)$, i.e., $V(X) = \{\nu \mid \nu : X \rightarrow \mathbb{R}_{\geq 0}\}$. 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 $X'$. $G(X)$ denotes the set of guards consisting of boolean combinations of constraints of the form $x_i \triangleright c$ with $x_i \in X, c \in \mathbb{N}$, and $\triangleright \in \{<, =, \leq, \geq, \triangleright\}$. Given $g \in G(X)$ and a valuation $\nu$, we write $\nu \models g$ when for every constraint $x_i \triangleright c$ in $g, \nu(x_i) \models c$ holds.

**Definition 1** (Timed automaton [Alur and Dill, 1992]). A timed automaton (TA) is a tuple $\mathcal{A} = (L, l_0, X, \Sigma, \Delta, G)$, s.t. $L$ 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, $\Delta \subseteq L \times G(X) \times \Sigma \times 2^X \times L$ 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 $g$, and with the clocks in $X'$ to be reset.

The semantics of a timed automaton $\mathcal{A}$ is a timed transition system $[\mathcal{A}] = \langle Q, q_0, \Gamma, \rightarrow, F_G \rangle$ where $Q = L \times V(X)$ is the (infinite) set of states, $q_0 = (l_0, \nu_0)$ is the initial state, with $\nu_0 = \nu[X \leftarrow 0], F_G = G \times V(X)$ 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 $\rightarrow \subseteq Q \times \Gamma \times Q$ is a set of transitions of the form $(l, \nu) \xrightarrow{(\delta, a)} (l', \nu')$ with $\nu' = (\nu + \delta)[Y \leftarrow 0]$ whenever there is a transition $(l, g, a, Y, l') \in \Delta$ s.t. $\nu + \delta \models g$, for $\delta \geq 0$.

A timed automaton $\mathcal{A} = (L, l_0, X, \Sigma, \Delta, G)$ is deterministic if for any different transitions $(l, g_1, a, Y_1, l_1)$ and $(l, g_2, a, Y_2, l_2)$ in $\Delta, g_1 \cap g_2$ 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 labelled by $a$ is valid (i.e., it holds for any clock valuation). An example of a timed automaton is given in Fig. 3.

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 $\text{Run}(\mathcal{A})$ and $\text{Run}_{F_G}(\mathcal{A})$ 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 $(t_1, a_1), (t_2, a_2) \ldots (t_n, a_n)$, with, for $1 \leq i \leq n, t_i = \sum_{k=1}^{i} \delta_k$. Thus, given the trace $\sigma = (t_1, a_1), (t_2, a_2) \ldots (t_n, a_n)$ of a run $\rho$ from a state $q \in Q$ to $q' \in Q$, we can define $w = (\delta_1, a_1), (\delta_2, a_2) \ldots (\delta_n, a_n)$, 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 $q \xrightarrow{w} q'$ whenever $q \xrightarrow{w} q'$ for the previously defined $w$. Note that to concatenate two traces
σ₁ and σ₂, it is needed to delay σ₂ to obtain a trace: the concatenation σ of σ₁ and σ₂ is the trace defined as σ = σ₁.(σ₂ +₁ \time(σ₁)). 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 \( \varphi \subseteq \text{tw}(\Sigma) \) accepted by a timed automaton. For a timed word \( \sigma \), we say that \( \sigma \) satisfies \( \varphi \), noted \( \sigma \models \varphi \) whenever \( \sigma \in \varphi \). We only consider regular timed properties whose associated automaton is complete and deterministic.

![Figure 3: Property \( \varphi_t \) modelling writes on a shared storage device](image)

Given a complete and deterministic automaton \( \mathcal{A} \) s.t. \( Q \) is the set of states of \( \left[ \mathcal{A} \right] \) and \( \rightarrow \) its transition relation, and a word \( \sigma \), for \( q \in Q \), we note \( q \after \sigma = q' \), where \( q' \) is s.t. \( q \xrightarrow{\sigma} q' \). Since \( \mathcal{A} \) is complete and deterministic, there exists only one such \( q' \). We note Reach(\( \sigma \)) = \( q_0 \) after \( \sigma \). We extend these definitions to languages: if \( L \) is a language, \( q \after L = \bigcup_{q \in L} q \after \sigma \) and Reach(\( L \)) = \( q_0 \) after \( L \). These definitions are valid both in the untimed and timed cases. For the timed case, we also allow to add an extra parameter to after and Reach, that represents an observation time. For \( q \in Q \), \( t \in \mathbb{R}_{\geq 0} \), and \( \sigma \in \text{tw}(\Sigma) \), \( q \after (\sigma, t) = (l, \nu + t - \time(\obs(\sigma, t))) \), where \( (l, \nu) = q \after(\obs(\sigma, t)) \), and Reach(\( \sigma, t \)) = \( q_0 \after(\sigma, t) \). This allows to consider states of the semantics that are reached after the last action of the input word, by letting time elapse. In particular, note that for \( (l, \nu) \in Q \), \( (l, \nu) \after(\varepsilon, t) = (l, \nu + t) \) is the state reached from \( (l, \nu) \) by letting time elapse of \( t \) units of time. Moreover, for \( (l, \nu) \in Q \), we note up(\( l, \nu \)) = \( \{ (l, \nu + t) \mid t \in \mathbb{R}_{\geq 0} \} \). This definition is extended to sets of states: for \( S \subseteq Q \), up(\( S \)) = \( \bigcup_{q \in S} \text{up}(q) \). We also define a predecessor operator: for \( q \in Q \) and \( a \in \Sigma \), Predₜ(\( q \)) = \( \{ q' \in Q \mid q' \after(0, a) = q \} \) for the untimed setting, and Predₜₓ(\( q \)) = \( \{ q' \in Q \mid q' \after (0, a) = q \} \) for the timed setting. This definition is extended to words: if \( \sigma \in T^* \) (or \( \sigma \in \text{tw}(\Sigma) \)), then Predₜₓ(\( q \)) = \( \{ q' \in Q \mid q' \after \sigma = q \} \).

Example 1 (Shared Data Storage). Consider the property \( \varphi_t \) described in Fig. 3 and representing writes on a shared data storage. A more detailed description of this property is given in Section 4.3. This property is similar to \( \varphi_{\text{ex}} \) (Fig. 2), but a clock has been added to impose that writes should not occur before two time units have elapsed since the reception of the last LockOff event. Thus, the set of locations of \( \varphi_t \) is \( L = \{ l₀, l₁, l₂, l₃ \} \), the initial location is \( l₀ \), the set of clocks is \( X = \{ x \} \), the alphabet is \( \Sigma = \{ \text{Auth}, \text{LockOn}, \text{LockOff}, \text{Write} \} \), the set of accepting locations is \( G = \{ l₁, l₂ \} \), and the set of transitions contains for instance transitions \( (l₀, \top, \text{Auth}, \emptyset, l₁) \), \( (l₁, \top, \text{Auth}, \emptyset, l₂) \), and \( (l₃, \top, \text{LockOn}, \emptyset, l₃) \), where \( \top \) is the guard that holds for every clock valuation.

Let \( Q = L \times \mathbb{R}_{\geq 0} \) be the set of states of the semantics of \( \varphi_t \), where the clock valuations
replaced by the value of the unique clock \( x \). Then, \( \text{Reach}((2, \text{Auth})) = (l_0, 0) \) after \((2, \text{Auth}) = (l_1, 2)\), and, for example, \((l_2, 3)\) after \((2, \text{LockOff}), 4 = (l_1, 2)\), because the clock is reset when the \( \text{LockOff} \) action occurs, and then \( 4 - 2 = 2 \) time units remain to reach date 4. Also, \( \text{Pred}_{\text{Write}}((l_1, 3)) = \{(l_1, 3)\} \), but, for instance, \( \text{Pred}_{\text{Write}}((l_1, 1)) = \emptyset \) since the only transition labelled by \( \text{Write} \) and leading to \( l_1 \) has guard \( x \geq 2 \).

3. Enforcement Monitoring of Untimed Properties

In this section, \( \varphi \) is a regular property defined by an automaton \( A_\varphi = (Q, q_0, \Sigma, \rightarrow, F) \). Recall that the general scheme of an enforcement mechanism (EM) is given in Fig. 1, where \( S \) represents the running system, \( \sigma \) its execution, \( E \) the enforcement mechanism, \( \varphi \) the property to enforce, and \( E(\sigma) \) the output of the enforcement mechanism, which should satisfy \( \varphi \).

We consider uncontrollable events in the set \( \Sigma_u \subseteq \Sigma \). These events cannot be modified by an EM, i.e. they cannot be suppressed nor buffered, so they must be output by the EM whenever they are received. Let us note \( \Sigma_c = \Sigma \setminus \Sigma_u \) the set of controllable events, which can be modified by the EM. An 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, an EM may interleave controllable and uncontrollable events.

In this section, for \( q \in Q \), we note \( u\text{Pred}(q) = \bigcup_{u \in \Sigma_u} \text{Pred}_u(q) \), and we extend this definition to sets of states: for \( S \subseteq Q \), \( u\text{Pred}(S) = \bigcup_{q \in S} u\text{Pred}(q) \). For \( S \subseteq Q \), we also note \( \overline{S} = Q \setminus S \).

3.1. Enforcement Functions and their Requirements

In this section, we consider an alphabet of actions \( \Sigma \). An enforcement function is a description of the input/output behaviour of an EM. Formally, we define enforcement functions as follows:

**Definition 2** (Enforcement Function). An enforcement function is a function from \( \Sigma^* \) to \( \Sigma^* \), that is increasing on \( \Sigma^* \) with respect to \( \leq \): \( \forall (\sigma, \sigma') \in (\Sigma^*)^2, \sigma \leq \sigma' \implies E(\sigma) \leq E(\sigma') \).

An enforcement function is a function that modifies an execution, and that cannot remove events it has already output.

In the sequel, we define the requirements on an EM and express them on enforcement functions. As stated previously, an EM should ensure that the executions of a running system satisfy \( \varphi \), thus its enforcement function has to be sound, meaning that its output always satisfies \( \varphi \):

**Definition 3** (Soundness). An enforcement function \( E : \Sigma^* \rightarrow \Sigma^* \) is sound with respect to \( \varphi \) in an extension-closed set \( S \subseteq \Sigma^* \) if \( \forall \sigma \in S, E(\sigma) \models \varphi \).

Since there are some uncontrollable events that are only observable by the EM, receiving uncontrollable events could lead to the property not being satisfied by the output of the enforcement mechanism. Moreover, some uncontrollable sequences could lead to a state of the property that would be a non-accepting sink state, leading to the enforcement mechanism not being able to satisfy the property any further. Consequently, in Definition 3, soundness is not defined for all words in \( \Sigma^* \), but in a subset \( S \), since it might happen that it is impossible to ensure it from the

\(^5\) This choice appeared to us as the most realistic one. Extending the notions presented in this section in order to handle enforcement mechanisms with suppression is rather simple.
initial state. Thus for an EM to be effective, $S$ needs to be extension-closed to ensure that the property is always satisfied once it has been. If $S$ were not extension-closed, soundness would only mean that the property is sometimes satisfied (in particular, the identity function would be sound in $\varphi$). In practice, there may be an initial period where the enforcement mechanism does not ensure the property (which is unavoidable), but as soon as a safe state is reached, the property becomes enforceable forever (and the property is guaranteed to hold). This approach appears to be the closest to the usual one without uncontrollable events.

The usual notion of transparency (cf. (Schneider, 2000; Ligatti et al., 2009)) states that the output of an EM 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 with respect to $\Sigma_u$ and $\Sigma_c$, noted compliant($E$, $\Sigma_u$, $\Sigma_c$), if $\forall \sigma \in \Sigma^*, E(\sigma) \equiv_{\Sigma_c} \sigma \land E(\sigma) \equiv_{\Sigma_u} \sigma \land \forall u \in \Sigma_u, E(\sigma).u \equiv E(\sigma.u)$.

Intuitively, compliance states that the EM does not change the order of the controllable events and emits uncontrollable events simultaneously with their reception, possibly followed by stored controllable events. We chose to consider enforcement mechanisms that can delay controllable events. To our opinion, it corresponds to the most common choice in practice. However, other primitives, such as deletion or reordering of controllable events could be easily considered. Using other enforcement primitives would require only few changes, especially adapting the definitions of compliance and optimality, and the construction of $G$ (see below). When clear from the context, the partition is not mentioned: $E$ is said to be compliant, and we note it compliant($E$).

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

In addition, an enforcement mechanism should be optimal in the sense that its output sequences should be maximal while preserving soundness and compliance. In the same way we defined soundness in an extension-closed set, we define optimality as follows:

Definition 5 (Optimality). An enforcement function $E : \Sigma^* \rightarrow \Sigma^*$ that is compliant with respect to $\Sigma_u$ and $\Sigma_c$, and sound in an extension-closed set $S \subseteq \Sigma^*$ is optimal in $S$ if:

$\forall E' : \Sigma^* \rightarrow \Sigma^*, \forall \sigma \in S, \forall a \in \Sigma,$

$(\text{compliant}(E')) \land E'(\sigma) = E(\sigma) \land |E'(\sigma.a)| > |E(\sigma.a)|) \Rightarrow (\exists \sigma_u \in \Sigma_u^*, E'(\sigma.a.\sigma_u) \neq \varphi).$

Intuitively, optimality states that if there exists a compliant enforcement function that outputs a longer word than an optimal enforcement function, then there must exist a sequence of uncontrollable events that would lead the output of that enforcement function to violate $\varphi$. This would imply that this enforcement function is not sound because of $\sigma.a.\sigma_u$. Thus, an enforcement function that outputs a longer word than an optimal enforcement function can not be sound and compliant. Since it is not always possible to satisfy the property from the beginning, this condition is restrained to an extension-closed subset of $\Sigma^*$, as is for soundness (Definition 3).

Example 2. We consider a simple untimed shared storage device. After Authentication, a user can write a value only if the storage is unlocked. (Un)locking the device is decided by another entity, meaning that it is not controllable by the user. Property $\varphi_{ex}$ (see Fig. 2) formalises the above re-
quirement, $\varphi_{eq}$ is not enforceable if the uncontrollable alphabet is $\{\text{LockOn, LockOff, Auth}\}$ since reading the word LockOn from $q_0$ leads to $q_3$, which is not an accepting state. However, the existence of such a word does not imply that it is impossible to enforce $\varphi_{eq}$ for some other input words. If word Auth is read, then state $q_1$ is reached, and from this state, it is possible to enforce $\varphi_{eq}$ by emitting Write only when in state $q_1$.

### 3.2. Synthesising Enforcement Functions

Example 2 shows that some input words cannot be corrected by the EM because of uncontrollable events. Nevertheless, since the received events may lead to a state from which it is possible to ensure that $\varphi$ will be satisfied (meaning that for any events received as input, the enforcement mechanism can output a sequence that satisfies $\varphi$), it would then be possible to define a subset of $\Sigma^*$ in which an enforcement function would be sound.

To be compliant, an enforcement mechanism can buffer the controllable events it has received to emit them later (i.e. after having received another events). Thus, the set of states from which an enforcement mechanism can ensure soundness, i.e. ensure it can always compute a prefix of the buffer that leads to an accepting state, whatever uncontrollable events are received, depends on its buffer. Thus, to synthesise a sound and compliant enforcement function, one needs to compute the set of words that can be emitted from a certain state with a given buffer, ensuring that an accepting state is always reachable. This set will be called $G$, and to define it, the set of states from which the enforcement mechanism can wait some events knowing an accepting state will always be reachable should be known (this set has to be a subset of $F$ since it is possible that no event is to be received). This set of states, which depends on the buffer, will be noted $S$, and is defined in conjunction with another set of states, $I$, that is used only to compute $S$. Thus, for a buffer $\sigma \in \Sigma_c^*$, we define the sets of states $I(\sigma)$ and $S(\sigma)$, that represent the states from which the enforcement mechanism can output the first event of $\sigma$, and the states in which the enforcement mechanism can wait for another event, respectively.

**Definition 6 (I, S).** Given a sequence of controllable events $\sigma \in \Sigma_c^*$, we define the sets of states of $\varphi$, $I(\sigma)$ and $S(\sigma)$ by induction as follows: $I(\epsilon) = \emptyset$, $S(\epsilon) = \{q \in F \mid q \text{ after } \Sigma_u^* \subseteq F\}$ and, for $\sigma \in \Sigma_c^*$ and $a \in \Sigma_c$,

$$I(a.\sigma) = \text{Pred}_{a}(S(\sigma) \cup I(\sigma)),$$

$$S(\sigma.a) = S(\sigma) \cup \max_{\subseteq}(\{Y \subseteq F_G \mid Y \cap \text{uPred}(Y \cup I(\sigma.a)) = \emptyset\}).$$

Intuitively, $S(\sigma)$ is the set of “winning” states, i.e. if an enforcement mechanism has reached a state in $S(\sigma)$ with buffer $\sigma$, it will always be able to reach $F$, whatever events are received afterwards, controllable or uncontrollable. Note that since there is a possibility of not receiving any other event, $S(\sigma) \subseteq F$, because the EM could end in any of these states, thus this condition is needed to ensure that the output of the EM satisfies the property. $S(\sigma.a)$ is defined as the biggest subset of $F$ such that no uncontrollable event leads outside of it or $I(\sigma.a)$, meaning that whatever uncontrollable event is received from a state in $S(\sigma.a)$, the state reached will be either in $F$ (since

---

* Uncontrollable events are emphasised in italics.
it will be in $S(\sigma.\alpha)$ or in $I(\sigma.\alpha)$. In both cases, this means that the enforcement mechanism can reach an accepting state, whatever uncontrollable events are received.

$I(\sigma)$ is the set of intermediate states, the states that can be “crossed” while emitting a prefix of the buffer. The states in $I(\sigma)$ do not need to be in $F$ since no event can be received while the EM is in these states, because it emits all the controllable word it wishes to emit at once. $I(\alpha.\sigma)$ is defined as the set of all states from which following the transition labelled by $\alpha$ leads either to $I(\sigma)$ or $S(\sigma)$, meaning that the EM can emit the first event of its buffer to be able to reach an accepting state, whatever uncontrollable events are received.

Now, we can use $S$ to define $G$, the set of words that can be emitted from a state $q \in Q$ by an enforcement mechanism with a buffer $\sigma \in \Sigma^*_c$.

**Definition 7 (G).** For $q \in Q$, $\sigma \in \Sigma^*_c$, $G(q, \sigma) = \{ w \in \Sigma^*_c \mid w \not\in \sigma \land q \land w \in S(w^{-1}, \sigma) \}$.

Intuitively, $G(q, \sigma)$ is the set of words that can be output by a compliant enforcement mechanism to ensure soundness from state $q$ with buffer $\sigma$. When clear from context, the parameters could be omitted; $G$ is the value of the function for the state reached by the output of an enforcement mechanism with its buffer.

Now, we use $G$ to define the functional behaviour of the enforcement mechanism.

**Definition 8 (Functions $store_\varphi$, $E_\varphi$).** The function $store_\varphi : \Sigma^* \to \Sigma^* \times \Sigma^*$ is defined as:

- $store_\varphi(\epsilon) = (\epsilon, \epsilon)$;
- for $\sigma \in \Sigma^*$ and $a \in \Sigma$, let $(\sigma_s, \sigma_c) = store_\varphi(\sigma)$, then:

$$store_\varphi(\sigma.a) = \begin{cases} (\sigma_s.a, \sigma'_s, \sigma'_c) & \text{if } a \in \Sigma_a \\ (\sigma_s, \sigma''_s, \sigma''_c) & \text{if } a \in \Sigma_c \end{cases}$$

where:

$$\kappa_\varphi(q, w) = \max_\varphi(G(q, w) \cup \{ \epsilon \}), \text{ for } q \in Q \land w \in \Sigma^*_c,$$

$$\sigma'_s = \kappa_\varphi(\text{Reach}(\sigma_s.a), \sigma_c), \quad \sigma'_c = \sigma'^{-1}_{\text{c}}, \sigma_c,$$

$$\sigma''_s = \kappa_\varphi(\text{Reach}(\sigma_s), \sigma_c.a), \quad \sigma''_c = \sigma''^{-1}_{\text{c}}.(\sigma_c.a).$$

The enforcement function $E_\varphi : \Sigma^* \to \Sigma^*$ is defined as $E_\varphi(\sigma) = \Pi_1(store_\varphi(\sigma))$, for any $\sigma \in \Sigma^*$.

Figure 4 gives a scheme of the behaviour of the enforcement function. 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 state from which it would be possible to violate the property reading only uncontrollable events. Upon receiving a new event $a$, the enforcement mechanism distinguishes two cases:

- If $a$ belongs to $\Sigma_a$, then it is output, as required by compliance. Then, the longest prefix of $\sigma_c$ that satisfies $\varphi$ and leads to a state in $S$ for the associated buffer is also output.
- If $a$ is in $\Sigma_c$, then it is added to $\sigma_c$, and the longest prefix of this new buffer that satisfies $\varphi$ and leads to a state in $S$ for the associated buffer is emitted, if it exists.

In both cases, $\kappa_\varphi$ is used to compute the longest word that can be output, that is the longest word in $G$ for the state reached so far with the current buffer of the enforcement mechanism, or $\epsilon$ if this

$\parallel E_\varphi$ and $store_\varphi$ depend on $\Sigma_a$ and $\Sigma_c$, but we did not write it in order to lighten the notations.
set is empty. The parameters of $\kappa_c$ are those which are passed to $G$. They correspond to the state reached so far by the output of the enforcement mechanism, and its current buffer, respectively.

As seen in Example 2, some properties are not enforceable, but receiving some events may lead to a state from which it is possible to enforce. Therefore, it is possible to define a set of words, called $\text{Pre}(\varphi)$, such that $E_\varphi$ is sound in $\text{Pre}(\varphi)$, as stated in Proposition 2:

**Definition 9 (Pre).** The set of input words $\text{Pre}(\varphi) \subseteq \Sigma^*$ is defined as follows:

$$\text{Pre}(\varphi) = \{ \sigma \in \Sigma^* \mid G(\text{Reach}(\sigma|_{\Sigma_c}), \sigma|_{\Sigma_u}) \neq \emptyset \}. \Sigma^*$$

Intuitively, $\text{Pre}(\varphi)$ is the set of words in which $E_\varphi$ is sound. This set is extension-closed, as required by Definition 3. In $E_\varphi$, using $S$ ensures that once $G$ is not empty, then it will never be afterwards, whatever events are received. Thus, $\text{Pre}(\varphi)$ is the set of input words such that the output of $E_\varphi$ would belong to $G$. Since $E_\varphi$ outputs only uncontrollable events until $G$ becomes non-empty, the definition of $\text{Pre}(\varphi)$ considers that the state reached is the one that is reached by omitting only the uncontrollable events of $\sigma$, and the corresponding buffer would then be the controllable events of $\sigma$.

**Example 3.** Considering property $\varphi_{\text{ex}}$ (Fig. 2), with the uncontrollable alphabet $\Sigma_u = \{ \text{Auth}, \text{LockOff}, \text{LockOn} \}$, $\text{Pre}(\varphi_{\text{ex}}) = \text{Write}^* \cdot \text{Auth} \cdot \Sigma^*$. Indeed, from the initial state $q_0$, if an uncontrollable event, say $\text{LockOff}$, is received, then $q_3$ is reached, which is a non-accepting sink state, and is thus not in $S(\varphi)$. In order to reach a state in $S$ (i.e. $q_1$ or $q_2$), it is necessary to read $\text{Auth}$. Once $\text{Auth}$ is read, $q_1$ is reached, and from there, all uncontrollable events lead to either $q_1$ or $q_2$. The same holds true from $q_2$. Thus, it is possible to stay in the accepting states $q_1$ and $q_2$, by delaying $\text{Write}$ events when in $q_2$ until a $\text{LockOff}$ event is received. Consequently, $q_1$ and $q_2$ are in $S(\sigma)$ for all $\sigma \in \Sigma^*_c$, and thus $\text{Pre}(\varphi_{\text{ex}}) = \text{Write}^* \cdot \text{Auth} \cdot \Sigma^*$. Since $\text{Write}$ events can be buffered while in state $q_0$ until event $\text{Auth}$ is received, leading to $q_1 \in S(\text{Write}^*)$.

$E_\varphi$, as defined in Definition 8, is an enforcement function that is sound with respect to $\varphi$ in $\text{Pre}(\varphi)$, compliant with respect to $\Sigma_u$ and $\Sigma_c$, and optimal in $\text{Pre}(\varphi)$.

**Proposition 1.** $E_\varphi$ as defined in Definition 8 is an enforcement function.

**Sketch of proof.** We have to show that for all $\sigma$ and $\sigma'$ in $\Sigma^*$, $E_\varphi(\sigma) \preceq E_\varphi(\sigma \cdot \sigma')$. Following the definition of $\text{store}_{\varphi}$, this holds provided that $\sigma' \in \Sigma$ (i.e. $\sigma'$ is a word of size 1). Since $\preceq$ is an order, it follows that the proposition holds for all $\sigma' \in \Sigma^*$.

**Proposition 2.** $E_\varphi$ is sound with respect to $\varphi$ in $\text{Pre}(\varphi)$, as per Definition 3.

**Sketch of proof.** We have to show that if $\sigma \in \text{Pre}(\varphi)$, then $E_\varphi(\sigma) \models \varphi$. The proof is made by induction on $\sigma$. In the induction step, considering $a \in \Sigma$, we distinguish three different cases:

- $\sigma . a \not\in \text{Pre}(\varphi)$. Then the proposition holds.
- $\sigma . a \in \text{Pre}(\varphi)$, but $\sigma \not\in \text{Pre}(\varphi)$. Then the input reaches $\text{Pre}(\varphi)$, and since it is extension-closed, all extensions of $\sigma$ also are in $\text{Pre}(\varphi)$, and we prove that the proposition holds considering the definition of $\text{Pre}(\varphi)$.
- $\sigma \in \text{Pre}(\varphi)$ (and thus, $\sigma . a \in \text{Pre}(\varphi)$ since it is extension-closed). Then, we prove that the proposition holds, based on the definition of $\text{store}_{\varphi}$, and more precisely on the definition of $S$, that ensures that there always exists a compliant output that satisfies $\varphi$.

**Proposition 3.** $E_\varphi$ is compliant, as per Definition 4.
Figure 5: Property that can be enforced by blocking all controllable events $c$.

Sketch of proof. The proof is made by induction on the input $\sigma \in \Sigma^*$. Considering $\sigma \in \Sigma^*$ and $a \in \Sigma$, the proof is straightforward by considering the different values of $\text{store}_\varphi(\sigma.a)$, $(\sigma.a)_{\Sigma_u}$, and $(\sigma.a)_{\Sigma_c}$ when $a \in \Sigma_c$ and $a \in \Sigma_u$.

Remark 2. Notice that for some properties, an enforcement function that would block all controllable events may still be sound and compliant. Consider for instance the property represented in Fig. 5, where $c$ is a controllable event, and $u$ an uncontrollable event. Then, outputting only the events $u$ and buffering all the $c$ events allows to stay in state $q_0$, which is accepting and in $S(\sigma)$ for every word $\sigma \in c^*$. This means that an enforcement mechanism that blocks all controllable events would be sound and compliant. Nevertheless, if two controllable events $c$ are received, they can be output to reach state $q_2$, which is also accepting and safe for all possible sequences. Then it is possible to release more events. Therefore, an enforcement mechanism that would output two $c$ events when they are received would be “better” than the first one blocking all of them, in the sense that its output would be longer (and thus closer to the input).

For any $\sigma \in \text{Pre}(\varphi)$, $E_\varphi(\sigma)$ is the longest possible word that ensures soundness and compliance, that is controllable events are blocked only when necessary. Thus, $E_\varphi$ is also optimal in $\text{Pre}(\varphi)$:

**Proposition 4.** $E_\varphi$ is optimal in $\text{Pre}(\varphi)$, as per Definition 5.

Sketch of proof. The proof is made by induction on the input $\sigma \in \Sigma^*$. Once $\sigma \in \text{Pre}(\varphi)$, we know that $E_\varphi(\sigma) \models \varphi$ since $E_\varphi$ is sound in $\text{Pre}(\varphi)$. $E_\varphi$ is optimal because in $\text{store}_\varphi$, $\kappa_\varphi$ provides the longest possible word. If a longer word were output, then either the output would not satisfy $\varphi$, or it would lead to a state that is not in $S$ for the corresponding buffer, meaning that there would exist an uncontrollable word leading to a non-accepting state that would not be in $S$ for the buffer. Then, the enforcement mechanism would have to output some controllable events from the buffer to reach an accepting state, but since the state is not in $S$, there would exist again an uncontrollable word leading to a non-accepting state that is not in $S$ for the updated buffer. By iterating, the buffer would become $\epsilon$ whereas the output of the enforcement mechanism would be leading to a non-accepting state. Therefore, outputting a longer word would mean that the function is not sound. This means that $E_\varphi$ is optimal in $\text{Pre}(\varphi)$, since it outputs the longest word that allows to be both sound and compliant.

**Example 4.** Consider property $\varphi_{ex}$ (Fig. 2). We illustrate in table 1 the enforcement mechanism by showing the evolution of $\sigma_s$ and $\sigma_c$ with input $\sigma = \text{Auth.LockOn.Write.LockOff}$.

3.3. Enforcement Monitors

Enforcement monitors are operational descriptions of EMs. We give a representation of an EM for a property $\varphi$ as an input/output transition system. The input/output behaviour of the enforce-
Table 1: Example of the evolution of \((\sigma_s, \sigma_c) = \text{store}_{\varphi, e}(\sigma), \text{ with input } Auth.LockOn.Write.LockOff\)

<table>
<thead>
<tr>
<th>(\sigma)</th>
<th>(\sigma_s)</th>
<th>(\sigma_c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\epsilon)</td>
<td>(\epsilon)</td>
<td>(\epsilon)</td>
</tr>
<tr>
<td>Auth</td>
<td>Auth</td>
<td>(\epsilon)</td>
</tr>
<tr>
<td>Auth.LockOn</td>
<td>Auth.LockOn</td>
<td>(\epsilon)</td>
</tr>
<tr>
<td>Auth.LockOn.Write</td>
<td>Auth.LockOn</td>
<td>Write</td>
</tr>
</tbody>
</table>

Enforcement monitor is the same as the one of the enforcement function \(E_\varphi\) defined in Section 3.2. Enforcement monitors are purposed to ease the implementation of EMs.

**Definition 10** (Enforcement Monitor). An enforcement monitor \(E\) for \(\varphi\) is a transition system \(\langle C^E, c^E_0, \Gamma^E, \rightarrow_E \rangle\) such that:

- \(C^E = Q \times \Sigma^*\) is the set of configurations.
- \(c^E_0 = (q_0, \epsilon)\) is the initial configuration.
- \(\Gamma^E = \Sigma^* \times \{\text{dump}(.), \text{pass-uncont}(.), \text{store-cont}(.)\} \times \Sigma^*\) is the alphabet, where the first, second, and third members are an input sequence, an enforcement operation, and an output sequence, respectively.
- \(\rightarrow_E \subseteq C^E \times \Gamma^E \times C^E\) is the transition relation, defined as the smallest relation obtained by applying the following rules in order (where \(w/\exists a/ \exists w'\) stands for \((w, \exists a, w') \in \Gamma^E\)):
  - **Dump**: \(\langle q, a, \sigma_c \rangle \xrightarrow{\epsilon/\text{dump}(a)/a} \langle q', \sigma_c \rangle\), if \(a \in \Sigma_c\), \(G(q, a, \sigma_c) \neq \emptyset\) and \(G(q, a, \sigma_c) \neq \{\epsilon\}\), with \(q' = q\) after \(a\),
  - **Pass-uncont**: \(\langle q, \sigma_c \rangle \xrightarrow{a/\text{pass-uncont}(a)/a} \langle q', \sigma_c \rangle\), with \(a \in \Sigma_a\) and \(q' = q\) after \(a\),
  - **Store-cont**: \(\langle q, \sigma_c \rangle \xrightarrow{a/\text{store-cont}(a)/a} \langle q, \sigma_c, a \rangle\), with \(a \in \Sigma_c\).

In \(E\), a configuration \(c = \langle q, \sigma \rangle\) represents the current state of the enforcement mechanism. The state \(q\) is the one reached so far in \(A_\varphi\) with the output of the monitor. The word of controllable events \(\sigma_c\) represents the buffer of the monitor, i.e. the controllable events of the input that it has not output yet. Rule **dump** outputs the first event of the buffer if it can ensure soundness afterwards (i.e. if there is a non-empty word in \(G\), that must begin with this event). 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 \(\text{store}_{\varphi, e}\), whereas \(\sigma_s\) is here represented by state \(q\) which is the first member of the configuration, such that \(q = \text{Reach}(\sigma_s)\).

**Proposition 5.** The output of the enforcement monitor \(E\) for input \(\sigma = E_{\varphi}(\sigma)\).

In Proposition 5, the output of the enforcement monitor is the concatenation of all the outputs of the word labelling the path followed when reading \(\sigma\). A more formal definition is given in the proof of this proposition, in appendix A.1.

**Sketch of proof.** The proof is made by induction on the input \(\sigma \in \Sigma^*\). We consider the rules applied when receiving a new event. If the event is controllable, then rule **store-cont** can be applied, possibly followed by rule **dump** applied several times. If the event is uncontrollable, then rule **pass-uncont** can be applied, again possibly followed by rule **dump** applied several
times. Since rule \texttt{dump()} applies only when there is a non-empty word in G, then this word must begin with the first event of the buffer, and the rule \texttt{dump()} can be applied again if there was a word in G of size at least 2, meaning that there is another non-empty word in the new set G, and so on. Thus, the output of all the applications of the rule \texttt{dump()} corresponds to the computation of $\kappa_\varphi$ in the definition of $\text{store}_\varphi$, and consequently the outputs of $E$ and $E_\varphi$ are the same.

Remark 3. Enforcement monitors as per Definition 10 are somewhat similar to the configuration description of EMs in (Falcone et al., 2011). The main difference with the EMs considered in (Falcone et al., 2011) is that the rule to be applied depends on the memory (the buffer), whereas in (Falcone et al., 2011) it only depends on the state and the event received.

4. Enforcement Monitoring of Timed Properties

We extend the framework in Section 3 to enforce timed properties. EMs and their properties need to be redefined to fit with timed properties. Enforcement functions need an extra parameter representing the date at which the output is observed. Soundness needs to be weakened so that, at any time instant, the property is allowed not to hold, provided that it will hold in the future.

Considering uncontrollable events with timed properties raises several difficulties. First, as in the untimed case, the order of events might be modified. Thus, previous definitions of transparency (Pinisetty et al., 2012), stating that the output of an enforcement function will eventually be a delayed prefix of the input, cannot 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 become not enforceable because of uncontrollable events, meaning that for these properties it is impossible to obtain sound EMs, as shown in Example 5.

In this section, $\varphi$ is a timed property defined by a timed automaton $A_\varphi = \langle L, l_0, X, \Sigma, \Delta, G \rangle$ with semantics $[A_\varphi] = \langle Q, q_0, \Gamma, \rightarrow, F_G \rangle$. As in the untimed setting, for $q \in Q$, we define $u\text{Pred}(q) = \bigcup_{u \in \Sigma_u} \text{Pred}_u(q)$, and for $S \subseteq Q$, $u\text{Pred}(S) = \bigcup_{q \in S} u\text{Pred}(q)$ and $\overline{S} = Q \setminus S$.

Example 5 (Non-Enforceable Property). Consider the property defined by the automaton in Fig. 6 with alphabet $\{a, b\}$. If all actions are controllable ($\Sigma_u = \emptyset$), the property is enforceable because an EM just 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.

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:

**Definition 11 (Enforcement Function).** Given an alphabet of actions $\Sigma$, an enforcement function $E$ is a function $E : \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow \text{tw}(\Sigma)$ s.t.:

$$\forall \sigma \in \text{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, \forall t' \geq t, \quad E(\sigma, t) \preceq E(\sigma, t') \quad \land \quad (\sigma.(t, a) \in \text{tw}(\Sigma) \implies E(\sigma, t) \preceq E(\sigma.(t, a), t)).$$
Definition 11 models physical constraints: an enforcement function can not remove something already output. The two conditions correspond to letting time elapse and reading a new event, respectively. In both cases, the new output must be an extension of what has been output so far.

Soundness states that the output of an enforcement function should eventually satisfy the property:

**Definition 12 (Soundness).** An enforcement function $E$ is sound with respect to $\varphi$ in a time-extension-closed set $S \subseteq \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0}$ if $\forall (\sigma, t) \in S, \exists t' \geq t, \forall t'' \geq t', E(\sigma, t'') \models \varphi$.

An enforcement function is sound in a time-extension-closed set $S$ if for any $(\sigma, t)$ in $S$, the value of the enforcement function with input $\sigma$ from date $t$ satisfies the property in the future. As in the untimed setting, soundness is not defined for all words in $\text{tw}(\Sigma)$, but in a set of words, this time associated with dates. The reason is the same as in the untimed setting: the EM might not be able to ensure soundness from the beginning, because of bad uncontrollable sequences. Moreover, in the definition of soundness, the set $S$ needs to be time-extension-closed to ensure that the property remains satisfied once the EM starts to operate.

Remark 4. Soundness could have been defined in the same way as in the untimed setting, however, with such alternative stronger definition, where the output of the EM must always satisfy the property, less properties could be enforced. Weakening soundness allows to enforce more properties, and to let enforcement mechanisms produce longer outputs.

Compliance states that uncontrollable events should be emitted instantaneously upon reception, and that controllable events can be delayed, but their order must remain unchanged:

**Definition 13 (Compliance).** Given an enforcement function $E$ defined on an alphabet $\Sigma$, we say that $E$ is compliant with respect to $\Sigma_u$ and $\Sigma_c$, noted compliant($E, \Sigma_u, \Sigma_c$), if $\forall \sigma \in \text{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, E(\sigma, t) \triangleq \exists \Delta \in \Sigma_u. \sigma \land E(\sigma, t) =_{\Sigma_u} \text{obs}(\sigma, t) \land \forall u \in \Sigma_u, E(\sigma, t)(t, u) \triangleq E(\sigma, (t, u), t)$.

Compliance is similar to the one in the untimed setting except that the controllable events can be delayed. However, their order must not be modified by the EM, that is, when considering the projections on controllable events, the output should be a delayed prefix of the input. Any uncontrollable event is released immediately when received, that is, when considering the projections on uncontrollable events, the output should be equal to the input.

We say that a property is enforceable whenever there exists a sound and compliant enforcement function for this property.

For a compliant enforcement function $E : \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow \text{tw}(\Sigma)$, and a timed word $\sigma \in \text{tw}(\Sigma)$, we note $E(\sigma)$ the value of $E$ with input $\sigma$ at infinite time (i.e. when it has stabilised). More formally, $E(\sigma) = E(\sigma, t)$, where $t \in \mathbb{R}_{\geq 0}$ is s.t. for all $t' \geq t$, $E(\sigma, t') = E(\sigma, t)$. Since $\sigma$ is finite, and $E$ is compliant, the output of $E$ with input word $\sigma$ is finite, thus such a $t$ must exist.

As in the untimed setting, we define a notion of optimality in a set:

**Definition 14 (Optimality).** We say that an enforcement function $E : \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow \text{tw}(\Sigma)$ that is compliant with respect to $\Sigma_c$ and $\Sigma_u$ and sound in a time-extension-closed set $S \subseteq \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0}$ is optimal in $S$ if for any enforcement function $E' : \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow \text{tw}(\Sigma)$, for all $\sigma \in \text{tw}(\Sigma)$, for all $(t, a)$ s.t. $\sigma.(t, a) \in \text{tw}(\Sigma)$,

$$\text{compliant}(E', \Sigma_u, \Sigma_c) \land (\sigma, t) \in S \land E'(\sigma, t) = E(\sigma, t) \land E(\sigma, (t, a)) \triangleq E'(\sigma, (t, a)) \Rightarrow (\exists \sigma_u \in \text{tw}(\Sigma_u), E'(\sigma, (t, a), \sigma_u) \not\models \varphi).$$

Optimality states that outputting a greater word (with respect to $\preceq_d$) than the output of an
optimal enforcement function leads to either compliance or soundness not being guaranteed. This holds from the point where the input begins to belong to the set in which the function is optimal, and since it is time-extension-closed, the input will belong to this set afterwards. In Definition 14, \(E\) is an optimal enforcement function, and \(E'\) is another compliant enforcement function, that we consider having a greater output (with respect to \(\leq_d\)) than \(E\) for some input word \(\sigma.(t,a)\). Then, since \(E\) is optimal, \(E'\) is not sound, because there exists a word of uncontrollable events s.t. the output of \(E'\) after receiving it eventually violates \(\varphi\).

An EM delaying events should buffer them until it can output them. Being able to enforce \(\varphi\) depends on the possibility of computing a timed word with the events of the buffer, even when receiving some uncontrollable events, that leads to an accepting state from the current one. Thus, we define, for every sequence \(\sigma\) of controllable actions, two sets of states of the semantics of \(\mathcal{A}_\varphi\), \(S(\sigma)\) and \(I(\sigma)\). \(S(\sigma)\) is the largest set s.t. from any of its states, it is possible to wait before emitting a word that leads to \(F_G\), knowing that all along the path, receiving an uncontrollable event will not prevent from computing such a word again. \(I(\sigma)\) is the set of states from which it is possible to emit the first event of \(\sigma\) and reach a state from which it is possible to compute a word that leads to \(F_G\), again s.t. receiving uncontrollable events does not prevent from eventually reaching \(F_G\).

**Definition 15 (I, S).** The sets of states of \([\mathcal{A}_\varphi]\), \(I(\sigma)\) and \(S(\sigma)\), are inductively defined over sequences of controllable events as follows: \(I(\epsilon) = \emptyset\) and \(S(\epsilon) = \{q \in F_G \mid q \text{ after } tw(\Sigma_u) \subseteq F_G\}\) and, for \(\sigma \in \Sigma^*_c\) and \(a \in \Sigma_c\),

\[
I(a.\sigma) = \text{Pred}_{\epsilon}(I(\sigma) \cup S(\sigma)),
\]

\[
S(\sigma.a) = S(\sigma) \cup \max_{\leq}(\{X \cup Y \subseteq Q \mid Y \subseteq F_G \land Y = \text{up}(Y) \land (\forall x \in X, \exists i \in I(\sigma.a), \exists \delta \in \mathbb{R}_{\geq 0}, x \text{ after } (\epsilon, \delta) = i \land \forall t < \delta, x \text{ after } (\epsilon, t) \in X) \land (X \cup Y) \cap u\text{Pred}(\overline{X \cup Y \cup I(\sigma.a)}) = \emptyset\})
\]

Intuitively, in Definition 15, \(S(\sigma)\) is the set of states of the semantics of \(\varphi\) that our EM will be allowed to reach with a buffer \(\sigma\). It corresponds to the states from which the EM will be able to reach \(F_G\), meaning that its output will satisfy the property, even if some uncontrollable events are received. From any state in \(S(\sigma)\), the EM can compute a word of controllable events (taken from its buffer \(\sigma\)) leading to \(F_G\), and if some uncontrollable events are received, it will also be able to compute a new word to reach \(F_G\), with events taken from its (possibly modified due to previous emissions of events) buffer. The set \(I(\sigma)\) is the set of states that the output of the enforcement mechanism will be authorised to “traverse”, meaning that the enforcement mechanism can emit the first event of its buffer \(\sigma\) immediately from these states, but not wait in them (contrary to the states in \(S(\sigma)\), from which the EM could choose to wait before emitting a new event).

These sets are defined by induction on \(\sigma\), which represents the buffer of the EM. If the EM has its buffer empty (\(\sigma = \epsilon\)), then the set of states from which it can emit a controllable event is empty, since it can only emit events from its buffer: \(I(\epsilon) = \emptyset\). Nevertheless, some states in \(F_G\) can be s.t. all uncontrollable words lead to a state in \(F_G\), meaning that from these states, the property will be satisfied even if some uncontrollable events are received. Consequently, \(S(\epsilon) = \{q \in F_G \mid q \text{ after } tw(\Sigma_u) \subseteq F_G\}\).
If a new controllable event \( a \) is received, it is added to the buffer, and then the EM can decide to emit the first event of its buffer to reach a state that is in \( S \) or \( I \) for its new buffer, this explains the definition of \( I(a, \sigma) \). Adding a new event to the buffer gives more possibilities to the EM (since it could act as if it had not received this event), thus \( S(\sigma) \subseteq S(\sigma.a) \). Moreover, \( S(\sigma.a) \) is made of the union of two sets, \( X \) and \( Y \). \( X \) is the set of states from which the EM can decide to wait before emitting the first event of its buffer, thus waiting from a state of \( X \) has to lead to a state in \( I(\sigma.a) \). \( Y \) is the set of states that are in \( F_G \) and from which the EM can decide to wait for a new uncontrollable event before doing anything. Since \( Y \subseteq F_G \), if no uncontrollable event is to be received, the property is satisfied, and otherwise, the EM can decide what to emit to reach \( F_G \). In order to ensure that receiving uncontrollable events do not prevent from being able to reach \( F_G \) with events from the buffer, \( X \) and \( Y \) are s.t. every uncontrollable event received from a state in \( X \) or \( Y \) leads to a state in \( X \), \( Y \), or \( I(\sigma.a) \). This is the purpose of the condition \((X \cup Y) \cap \text{upPred}(X \cup Y \cup I(\sigma.a)) = \emptyset \). On top of this, it is necessary to ensure that all the states reached while waiting from \( X \) or \( Y \) are in \( X \) or \( Y \), otherwise there could be a state reached by the EM for which there is an uncontrollable event leading to a state from which it is impossible to reach \( F_G \) with events of the buffer, meaning that the enforcement would not be sound. This is ensured by the conditions \( x \) after \((e, t) \in X \), and \( Y = \text{up}(Y) \). To have the best EM possible, these sets are as large as possible.

Note that if \( X_1 \) and \( X_2 \) satisfy the conditions required for \( X \), then \( X_1 \cup X_2 \) also satisfies them. Thus, the bigger set satisfying these properties exists. The same holds for \( Y \).

For convenience, we also define \( G : Q \times \Sigma_c \to 2^{\text{tw}(\Sigma)} \) which gives, for a state \( q \) and a sequence of controllable events \( \sigma \), the set of timed words made with the actions of \( \sigma \) that can be output from \( q \) in a safe way (i.e. all the states reached while emitting the word are in the \( S \) set corresponding to what remains from \( \sigma \)):

\[
G(q, \sigma) = \{ w \in \text{tw}(\Sigma) \mid \Pi_\Sigma(w) \not\subseteq \sigma \land \text{q after } w \in F_G \land \forall t \in \mathbb{R}_{\geq 0}, q \text{ after } (w, t) \in S(\Pi_\Sigma(\text{obs}(w, t))^{-1}.\sigma) \}.
\]

It is now possible to use \( G \) to define an enforcement function for \( \varphi \), denoted as \( E_\varphi \):

**Definition 16 (Functions store_\varphi, E_\varphi).** Let \( \text{store}_\varphi \) be the function : \( \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0} \to \text{tw}(\Sigma) \times \Sigma_c^* \times \Sigma_c^* \) defined inductively by:

\[
\forall t \in \mathbb{R}_{\geq 0}, \text{store}_\varphi(\epsilon, t) = (\epsilon, \epsilon, \epsilon),
\]

and for \( \sigma \in \text{tw}(\Sigma), (t', a) \) s.t. \( \sigma(t', a) \in \text{tw}(\Sigma) \), and \( t \geq t' \), if \( (\sigma_x, \sigma_b, \sigma_c) = \text{store}_\varphi(\sigma, t') \), then

\[
\text{store}_\varphi(\sigma, (t', a), t) = \begin{cases} 
(\sigma_x, (t', a), \text{obs}(\sigma'_x, t), \sigma'_b, \sigma'_c) & \text{if } a \in \Sigma_u \\
(\sigma_x, \text{obs}(\sigma''_b, t), \sigma''_b, \sigma''_c) & \text{if } a \in \Sigma_c
\end{cases}
\]

with:

\[
\kappa_\varphi(q, w) = \min(\max(G(q, w) \cup \{\epsilon\})), \text{for } q \in Q \text{ and } w \in \Sigma_c^*.
\]

\[
\text{buffer}_c = \Pi_\Sigma(\text{noobs}(\sigma_b, t'), \sigma_c),
\]

\[
t_1 = \min\{t'' \in \mathbb{R}_{\geq 0} \mid t'' \geq t' \land G(\text{Reach}(\sigma_x, (t', a), t''), \text{buffer}_c) \neq \emptyset \} \cup \{+\infty\},
\]

\[
\sigma'_b = \kappa_\varphi(\text{Reach}(\sigma_x, (t', a), \min\{\{t, t_1\}\}), \text{buffer}_c) + t \land \min\{t, t_1\},
\]

\[
\sigma'_c = \Pi_\Sigma(\sigma''_b)^{-1}.\text{buffer}_c,
\]

\[
t_2 = \min\{t'' \in \mathbb{R}_{\geq 0} \mid t'' \geq t' \land G(\text{Reach}(\sigma_x, (t', a), \sigma_c), \text{buffer}_c) \neq \emptyset \} \cup \{+\infty\},
\]

\[
\sigma'_b = \kappa_\varphi(\text{Reach}(\sigma_x, (t', a), t''), \text{buffer}_c) + t \land \min\{t, t_2\},
\]

\[
\sigma'_c = \Pi_\Sigma(\sigma''_b)^{-1}.\text{buffer}_c.
\]
\[ t_2 = \min\{t'' \in \mathbb{R}_{\geq 0} \mid t'' \preceq t' \wedge \text{G(Reach}(s, t''), \text{buffer}_c.a) \neq \emptyset\} \cup \{+\infty\}, \]
\[ \sigma_b'' = \kappa_\varphi(\text{Reach}(s, \min\{(t, t_2)\}), \text{buffer}_c.a) + 1 \min\{(t, t_2)\}, \]
\[ \sigma_c'' = \Pi_\Sigma(\sigma_b'')^{-1}.(\text{buffer}_c.a). \]

For \( \sigma \in \text{tw}(\Sigma) \) and \( t \in \mathbb{R}_{\geq 0} \), we define \( E_\varphi(\sigma, t) = (\Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t), t))). \)

The function \( \text{store}_\varphi \) takes a timed word \( \sigma \) and a date \( t \) as input, and outputs three words: \( \sigma_s, \sigma_b, \) and \( \sigma_c. \) \( \sigma_s \) is the output of the enforcement function at time \( t. \) \( \sigma_b \) is the timed word, composed of controllable events, that is to be output after the date of the last event of the input, if no uncontrollable event is received. \( \sigma_c \) is the untimed word composed of the remaining controllable actions of the buffer. When time elapses, after the last event of the input, \( \sigma_s \) is modified to output the events of \( \sigma_b \) when the dates are reached. Since letting time elapse can disable some transitions, it is possible to reach a state in \( S \) or \( I \) without emitting any event, and thus \( \sigma_b \) can change at this moment, changing from \( \epsilon \) to a word in \( G. \) This change of \( \sigma_b \) when letting time elapse can only happen once, since \( G \) will not be empty anymore once it has become non-empty.

\( t_1 \) and \( t_2 \) are used for this purpose, they both represent the time at which \( G \) becomes non-empty, if \( a \in \Sigma_a \) or \( a \in \Sigma_c \), respectively. Words are thus calculated from this point whenever \( G \) has become non-empty, to ensure that what has already been output is not modified. If \( G \) is still empty, then \( \min\{(t, t_1)\} \) or \( \min\{(t, t_2)\} \), depending on whether \( a \in \Sigma_c \) or \( a \in \Sigma_a \) equals to \( t \), meaning that \( \sigma_0 = \epsilon. \) Most of the time, \( t_1 \) or \( t_2 \) is equal to \( t' \), it is not the case only when \( G \) still empty at time \( t' \), but if \( G \) was not empty at date \( t' \), then \( t_1 \) or \( t_2 \) is equal to \( t' \). \( \sigma_c \) contains the controllable actions of the input that have not been output and do not belong to \( \sigma_b. \)

It is used to compute the new value of \( \sigma_b \) when possible. When receiving a new event in the input, it is appended to \( \sigma_s \) if it is an uncontrollable event, or the action is appended to the buffer if it is a controllable one. Then, \( \sigma_b \) is computed again, from the new state reached if it was an uncontrollable event, or with the new buffer if it was controllable. Note that \( t_1 \) and \( t_2 \) may not exist, since they are minima of an interval that can be open, depending on the strictness of the considered guard. In this case, one should consider the infimum instead of the minimum, and add an infinitesimal delay, s.t. the required transition is taken.

As mentioned previously, an EM may not be sound from the beginning of an execution, but some uncontrollable events may lead to a state from which it becomes possible to be sound. Whenever \( \sigma_0 \) is in \( G, \) then it will always be, meaning that the output of \( E_\varphi \) will eventually reach a state in \( F_G, \) i.e. it will eventually satisfy \( \varphi. \) Thus, \( E_\varphi \) eventually satisfies \( \varphi \) as soon as the state reached so far is in \( S(\sigma_0) \) or \( I(\sigma_b). \) This leads to the definition of \( \text{Pre}(\varphi, t) \), which is the set of timed words for which \( E_\varphi \) ensures soundness at time \( t. \) For \( \sigma \in \text{tw}(\Sigma), \) if \( (\sigma_s, \sigma_b, \sigma_c) = \text{store}_\varphi(\sigma, t), \) then \( \sigma \) is in \( \text{Pre}(\varphi, t) \) if and only if the set \( \text{G(Reach}(s, t), \Pi_\Sigma(\text{nobs}(\sigma_b, t)), \sigma_c) \) is not empty. Then, \( \text{Pre}(\varphi, t) \) is used to define \( \text{Pre}(\varphi) \), which is the set in which \( E_\varphi \) is sound:

**Definition 17** (\( \text{Pre}(\varphi) \)). \( \text{Pre}(\varphi) = \{(\sigma, t) \mid \sigma \in \text{Pre}(\varphi, t)\} \), where, for \( \sigma \in \text{tw}(\Sigma) \) and \( t \in \mathbb{R}_{\geq 0}, \)

\[ \text{Pre}(\varphi, t) = \{\sigma \in \text{tw}(\Sigma) \mid \exists \sigma' \preceq \sigma, \exists t' \leq t, \]
\[ \text{G}(\text{Reach}(\text{obs}(\sigma', t')|\Sigma_a, t'), \Pi_\Sigma(\text{obs}(\sigma', t')|\Sigma_c)) \neq \emptyset\}. \]

Note that \( \text{Pre}(\varphi) \) is time-extension-closed, meaning that once \( E_\varphi \) is sound, its output will always eventually satisfy \( \varphi \) in the future.

Since the output of our enforcement function consists only of the uncontrollable events from
the input, if \( G(\text{Reach}(\text{obs}(\sigma, t)_{\Sigma}, t), \Pi_\Sigma(\text{obs}(\sigma, t)|_{\Sigma})) \) is not empty, this means that the enforcement function becomes sound with input \( \sigma \) from time \( t \), since there is a word that is safe to emit. Thus, \( \text{Pre}(\varphi, t) \) is the set of inputs for which \( E_\varphi \) is sound after date \( t \), and then \( E_\varphi \) is sound for any input in \( \text{Pre}(\varphi) \) after its associated date.

**Proposition 6.** \( E_\varphi \) as defined in Definition 16 is an enforcement function, as per Definition 11.

**Sketch of proof.** We have to show that for all \( \sigma \in \text{tw}(\Sigma) \), for all \( t \in \mathbb{R}_{\geq 0} \) and \( t' \geq t \), \( E_\varphi(\sigma, t) \approx E_\varphi(\sigma, t') \), and for all \( (t, a) \) s.t. \( \sigma, (t, a) \in \text{tw}(\Sigma) \), \( E_\varphi(\sigma, t) \approx E_\varphi(\sigma, (t, a)) \). To prove this, we first show by induction that \( E_\varphi(\sigma, t) \approx E_\varphi(\sigma, t') \). Considering \( (t'', a) \) s.t. \( \sigma, (t'', a) \in \text{tw}(\Sigma) \), we distinguish different cases according to the values of \( t'' \) compared to \( t \) and \( t' \):

- \( t'' \leq t \). Then, in the definition of \( \text{store}_\varphi, t_1 \) (or \( t_2 \), if \( a \) is controllable) has the same value in \( \text{store}_\varphi(\sigma, t) \) and \( \text{store}_\varphi(\sigma, (t'', a), t') \). Then, comparing \( t \) to \( t_1 \), either \( E_\varphi(\sigma, (t'', a), t) = \epsilon \) if \( t < t_1 \), and then \( E_\varphi(\sigma, (t'', a), t) \approx E_\varphi(\sigma, (t', a), t') \), or \( t \geq t_1 \), and then there exists \( \sigma_i \) s.t. \( \sigma_i(\sigma, (t'', a), t) = \sigma_i(\sigma, (t', a), t') \) and \( E_\varphi(\sigma, (t'', a), t) = E_\varphi(\sigma, (t', a), t') \), meaning that \( E_\varphi(\sigma, (t'', a), t) \approx E_\varphi(\sigma, (t', a), t') \).

- \( t'' \geq t' \). Then the proposition holds because in the definition of \( E_\varphi \), only the observation of the input word at the given time is considered, meaning that \( E_\varphi(\sigma, (t'', a), t) = E_\varphi(\sigma, t) \) and \( E_\varphi(\sigma, (t', a), t') = E_\varphi(\sigma, t') \). By induction hypothesis, the proposition thus holds.

- \( t < t'' < t' \). Then, \( E_\varphi(\sigma, (t'', a), t) = E_\varphi(\sigma, t) \), and \( E_\varphi(\sigma, (t', a), t') = \Pi_1(\text{store}_\varphi(\sigma, (t'', a), t')) \), meaning that, looking at the definition of \( \text{store}_\varphi \), \( E_\varphi(\sigma, (t'', a), t) \approx E_\varphi(\sigma, (t', a), t') \).

Thus, \( E_\varphi(\sigma, t) \approx E_\varphi(\sigma, t') \). Then, what remains to show is that if \( \sigma, (t, a) \in \text{tw}(\Sigma) \), then \( E_\varphi(\sigma, t) \approx E_\varphi(\sigma, (t', a), t) \). Following the definition of \( \text{store}_\varphi \), it is clear that \( \text{store}_\varphi(\sigma, (t, a), t) \approx \text{store}_\varphi(\sigma, (t', a), t) \), and thus \( E_\varphi(\sigma, (t, a), t) \approx E_\varphi(\sigma, (t', a), t) \).

**Proposition 7.** \( E_\varphi \) is sound with respect to \( \varphi \) in \( \text{Pre}(\varphi) \) as per Definition 12.

**Sketch of proof.** As in the untimed setting, the proof is made by induction on the input \( \sigma \in \text{tw}(\Sigma) \). Similarly to the untimed setting, considering \( \sigma \in \text{tw}(\Sigma), t \in \mathbb{R}_{\geq 0} \), and \( (t', a) \) s.t. \( \sigma, (t', a) \in \text{tw}(\Sigma) \), there are three possibilities:

- \( \sigma, (t', a), t \notin \text{Pre}(\varphi) \). Then, the proposition holds.
- \( \sigma, (t', a), t \in \text{Pre}(\varphi) \), but \( \sigma, t' \notin \text{Pre}(\varphi) \). Then, this is when the input reaches \( \text{Pre}(\varphi) \).

Considering the definition of \( \text{Pre}(\varphi) \), we then prove that it is possible to emit a word with the controllable events seen so far, leading to an accepting state in \( S \).

- \( \sigma, t' \in \text{Pre}(\varphi) \) (and thus \( \sigma, (t', a), t \) too). Then, we prove again that there exists a controllable word made with the events which have not been output yet leading to an accepting state that is in \( S \), but this time considering the definitions of \( S \) and \( I \).

**Proposition 8.** \( E_\varphi \) is compliant, as per Definition 13.

**Sketch of proof.** As in the untimed setting, the proof is made by induction on the input \( \sigma \), considering the different cases where the new event is controllable or uncontrollable. The only difference with the untimed setting is that one should consider dates on top of actions.

**Proposition 9.** \( E_\varphi \) is optimal in \( \text{Pre}(\varphi) \), as per Definition 14.

**Sketch of proof.** This proof is made by induction on the input \( \sigma \). Whenever \( \sigma \in \text{Pre}(\varphi) \), since \( E_\varphi \) is sound in \( \text{Pre}(\varphi) \), then \( E_\varphi(\sigma) \) is the maximal word (with respect to \( \leq_c \)) that satisfies \( \varphi \) and
is safe to output. It is maximal because in the definition of store_c, \( \kappa_c \) returns the longest word with lower delays (for lexicographic order), which corresponds to the maximum with respect to \( \preceq_d \). Thus, outputting a greater word (with respect to \( \preceq_d \)) would lead to \( G \) being empty, meaning that the EM would not be sound. Thus, \( E_c \) is optimal in \( \text{Pre}(\varphi) \), since it outputs the maximal word with respect to \( \preceq_d \) that allows to be sound and compliant.

### 4.2. Enforcement Monitors

As in the untimed setting, we define an operational description of an EM whose output is exactly the output of \( E_c \), as defined in Definition 16.

**Definition 18.** An enforcement monitor \( E \) for \( \varphi \) is a transition system \( \langle C^E, c^E_0, \Gamma^E, \rightarrow^E \rangle \) s.t.:

- \( C^E = tw(\Sigma) \times \Sigma^* \times Q \times \mathbb{R}_{\geq 0} \times \{ \top, \bot \} \) is the set of configurations.
- \( c^E_0 = (\epsilon, \epsilon, q_0, 0, \bot) \in C^E \) is the initial configuration.
- \( \Gamma^E = (\langle (\mathbb{R}_{\geq 0} \times \Sigma) \cup \{ \epsilon \} \rangle \times Op \times (\langle (\mathbb{R}_{\geq 0} \times \Sigma) \cup \{ \epsilon \} \rangle \times \{ \epsilon \}) \) is the alphabet, composed of an optional input, an operation and an optional output.

The set of operations is \( \{ \text{compute}(\cdot), \text{dump}(\cdot), \text{pass-uncont}(\cdot), \text{store-cont}(\cdot), \text{delay}(\cdot) \} \). Whenever \( (\sigma, b, \alpha, \alpha') \in \Gamma^E \), it will be noted \( \sigma / \alpha / \alpha' \).

- \( \rightarrow^E \) is the transition relation defined as the smallest relation obtained by applying the following rules given by their priority order:
  - Compute: \( (\epsilon, \sigma_c, q, t, \bot) \xrightarrow{\text{\scriptsize compute}(\cdot)/\epsilon} (\sigma'_c, \sigma'_c, q, t, \top) \), if \( G(q, \sigma_c) \neq \emptyset \), with \( \sigma'_c = \kappa_c(q, \sigma_c) \) + t, and \( \sigma'_c = \Pi_{\Sigma}(\sigma'_c)^{-1}.\sigma_c \).
  - Dump: \( (t, a).\sigma_b, \sigma_c, q, t, \top) \xrightarrow{\text{\scriptsize dump}(t,a)/(t,a)} (\sigma_b, \sigma_c, q', t, \top) \), with \( q' = q \) after \( (0, a) \).
  - Pass-uncont: \( (\sigma_b, \sigma_c, q, t, b) \xrightarrow{(t,a)/\text{\scriptsize pass-uncont}(t,a)/(t,a)} (\epsilon, \Pi_{\Sigma}(\sigma_b).\sigma_c, q', t, \bot) \), with \( q' = q \) after \( (0, a) \).
  - Store-cont: \( (\sigma_b, \sigma_c, q, t, b) \xrightarrow{\text{\scriptsize store-cont}(t,c)/\epsilon} (\epsilon, \Pi_{\Sigma}(\sigma_b).\sigma_c.c, q, t, \bot) \).
  - Delay: \( (\sigma_b, \sigma_c, \langle l, v, t \rangle, b) \xrightarrow{\text{\scriptsize delay}(\delta)/\epsilon} (\sigma_b, \sigma_c, \langle l, v + \delta, t + \delta \rangle, b) \).

In a configuration \( (\sigma_b, \sigma_c, q, t, b) \), \( \sigma_b \) is the word to be output as time elapses; \( \sigma_c \) is the sequence of controllable actions from the input that are not used in \( \sigma_b \) and have not been output yet; \( q \) 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; and \( b \) indicates whether \( \sigma_b \) and \( \sigma_c \) should be computed (due to the reception of a new event for example).

The timed word \( \sigma_b \) corresponds to \( \text{nobs}(\sigma_b, t) \) from the definition of store_c, whereas \( \sigma_c \) is the same as in the definition of store_c. The state \( q \) represents \( \sigma_c \), s.t. \( q = \text{Reach}(\sigma_c, t) \).

**Proposition 10.** The output of \( E \) for input \( \sigma \) is \( E_c(\sigma) \).

As in the untimed setting, in Proposition 10, the output of the enforcement monitor is the concatenation of the outputs of the word labelling the path followed by the enforcement monitor when reading \( \sigma \). A formal definition is given in the proof of this proposition, in appendix A.2.

**Sketch of proof.** The proof is done by induction on \( \sigma \). When receiving a new event, the rule store-cont() can be applied if it is controllable, or rule pass-uncont() if it is uncontrollable. Doing so, the last member of the configuration is set to \( \bot \), meaning that the word to be emitted
and order. The sets of events are:

synchronisation. Each time the lock is taken and released, we must also wait before issuing a new event is received, we are able to send the controllable event Write

uncontrollable event

ing that the lock has been locked by someone else, and that it is unlocked, respectively. The

can get the status of a lock through the uncontrollable events LockOn and LockOff.

Consider Fig. 3, representing a property modelling the use of some shared writable device. We

can be computed. If the input is in Pre(\varphi), then rule compute() can be applied, and then the

second member of the configuration will have the same value as the second member of store2, and the same goes for the third members. Then, rule delay() can be applied, to reach the date of the first event in the second member of the current configuration, and then rule dump() can be applied to output it. This process can be repeated until the desired date is reached. Thus, when date t is reached, what has been emitted since the last rule store-cont() or pass-uncont() is obs(\sigma, t), where \sigma_2 was computed by rule compute() as second member. Considering the definition of store_2, it follows that the output of \mathcal{E} with input \sigma at date t is \mathcal{E}_\varphi(\sigma, t).

4.3. Example

Consider Fig. 3, representing a property modelling the use of some shared writable device. We can get the status of a lock through the uncontrollable events LockOn and LockOff indicating that the lock has been locked by someone else, and that it is unlocked, respectively. The uncontrollable event Auth is sent by the device to authorise writings. Once the Auth event is received, we are able to send the controllable event Write after having waited some time for synchronisation. Each time the lock is taken 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\}.

\Sigma \setminus \{Write\} \quad \Sigma \setminus \{\text{Write}\} \quad \Sigma

\sum

Figure 7: Example of Property without uncontrollable events

and \((l_2, \nu) \in S(\epsilon)\), since all uncontrollable words from \(l_1\) and \(l_2\) lead to \(l_1\) or \(l_2\), which are both accepting states.

We can also follow the execution of an enforcement monitor enforcing the property in Fig. 3,
watching the evolution of the configurations as semantic rules are applied. 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. The variable $t$ defines the global time of the execution. Fig. 8 shows the execution of the enforcement monitor with input $(1, \text{Auth}).(2, \text{LockOn}).(4, \text{Write}).(5, \text{LockOff}).(6, \text{LockOn}).(7, \text{Write}).(8, \text{LockOff})$. In Fig. 8, valuations are represented as integers, giving the value of the only clock $x$ of the property, $\text{LockOff}$ is abbreviated as $\text{off}$, $\text{LockOn}$ as $\text{on}$, and $\text{Write}$ as $w$. First column depicts the dates of events, then red text is the current output ($\sigma_s$) of the EM, blue text shows the evolution of $\sigma_b$ and green text depicts the remaining input word at this date. We can observe that the final output is the same as the one of the enforcement function: $(1, \text{Auth}),(2, \text{on}),(5, \text{off}),(6, \text{on}),(8, \text{off}),(10, w),(10, w)$.

**Remark 5.** The EM in Definition 16 output longer timed words than the approach in (Pinisetty et al., 2012) and (Pinisetty et al., 2014c) when applied only with controllable events thanks to optimality considerations. Consider the property in Fig. 7 over the set of controllable actions $\Sigma \supseteq \{\text{Write}\}$, and the input timed word $(1, \text{Write}),(1.5, \text{Write})$ input to the EM. The output obtained with our approach at date $t = 4$ is $(4, \text{Write}),(4, \text{Write})$ whereas the output obtained in (Pinisetty et al., 2012) would be $(2, \text{Write})$.

5. Related Work

Runtime enforcement was pioneered by the work of Schneider with security automata (Schneider, 2000), a runtime mechanism for enforcing safety properties. In (Schneider, 2000), 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 insert and suppress events from the execution. Later, more general models were proposed where the monitors can be synthesised from regular properties (Falcone et al., 2011). More recently, Bloem et al. (Bloem et al., 2015) presented a framework to synthesise enforcement monitors for reactive systems, called 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 (Charafeddine et al., 2015)). 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 Dolzhenko et al. (Dolzhenko et al., 2015) introduces Mandatory Result Automata (MRAs). MRAs extend edit-automata by refining the input/output relationship of an EM and thus allowing a more precise description of the enforcement abilities of an EM in concrete application scenarios. All the previously mentioned approaches considered untimed specifications, and do not consider uncontrollable events.

In the timed setting, several monitoring tools exist. RT-Mac (Sammapun et al., 2005) permits to verify at runtime timeliness and reliability correctness. LARVA (Colombo et al., 2009a; Colombo et al., 2009b) takes as input safety properties expressed with DATEs (Dynamic Automata with Events and Timers), a timed model similar to timed automata.

In previous work, we introduced runtime enforcement for timed properties (Pinisetty et al., 2012) specified by timed automata (Alur and Dill, 1992). We proposed a model of EMs that work as delayers, that is, mechanisms that are able to delay the input sequence of timed events to correct it. While (Pinisetty et al., 2012) proposed synthesis techniques only for safety and co-safety properties, we then generalised the framework to synthesise an enforcement monitor for
Figure 8: Execution of an enforcement monitor with input $(1, \text{Auth}) \cdot (2, \text{LockOn}) \cdot (4, \text{Write}) \cdot (5, \text{LockOff}) \cdot (6, \text{LockOn}) \cdot (7, \text{Write}) \cdot (8, \text{LockOff})$
any regular timed property (Pinisetty et al., 2014b; Pinisetty et al., 2014c). In (Pinisetty et al., 2014a), we considered parametric timed properties, that is timed properties with data-events containing information from the execution of the monitored system. In our approach, the optimality of the enforcement mechanism is based on the maximisation of the length of the output sequence. When applied in the case of controllable events only, this improves the preceding results.

Basin et al. (Basin et al., 2011) introduced uncontrollable events for security automata (Schneider, 2000). The approach in (Basin et al., 2011) allows to enforce safety properties where some of the events in the specification are uncontrollable. More recently, they proposed a more general approach (Basin et al., 2013) related to enforcement of security policies with controllable and uncontrollable events. They presented several complexity results and how to synthesise EMs. In case of violation of the property, the system stops the execution. They handle discrete time, and clock ticks are considered as uncontrollable events. In our approach, we consider dense time using the expressiveness of timed automata, any regular properties, and our monitor are more flexible since they block the system only when delaying events cannot prevent from violating the property, thus offering the possibility to correct many violations.

6. Conclusion and Future Work

This paper extends previous work on enforcement monitoring with uncontrollable events, which are only observable by an EM. We present a framework for both untimed and timed regular properties, described with (untimed) automata and timed automata, respectively. We provide a functional and an operational description of the enforcement mechanism, and show their equivalence. Adding uncontrollable events leads to the necessity of changing the order between controllable and uncontrollable events, which requires some existing notions to be adapted. Therefore, we replace transparency with compliance, and then give EMs, i.e. enforcement functions and enforcement monitors, for regular properties and regular timed properties. Since not every property can be enforced, we also give a condition, depending on the property and the input word, indicating whether the EM is sound with respect to the property under scrutiny or not. The EMs output immediately all the uncontrollable events received, and store the controllable ones, until soundness can be guaranteed. Then, they output events only when they can ensure that soundness will be satisfied. The proposed EMs are then sound and compliant, even with reception of some uncontrollable events. They are also optimal in the sense that they output the longest possible word, with the least possible dates in the timed setting.

One possible extension is to take some risks, outputting events even if some uncontrollable events could lead to a bad state, and introducing for example some probabilities. Implementing the given enforcement devices for the untimed setting is pretty straightforward, whereas implementation in the timed setting needs more attention due to computing in timed models. Another interesting direction for further investigation is to use game theory in order to compute the behaviour of the EM. This approach should permit to compute the behaviour before the execution, thus leading to an optimised implementation.
Optimal Enforcement of (Timed) Properties with Uncontrollable Events

References


### Appendix 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 $A_\varphi = (Q, q_0, \Sigma, \rightarrow, F)$. In some proofs, we also use notations from Definition 8.

**Proposition 1.** $E_\varphi$ as defined in Definition 8 is an enforcement function.

**Proof.** Let us consider $\sigma \in \Sigma^*$, and $\sigma' \in \Sigma^*$. If $\sigma' = \epsilon$, then $E_{\varphi}(\sigma) = E_{\varphi}(\sigma, \sigma') \not\approx E_{\varphi}(\sigma, \sigma')$. Otherwise, let us consider $(\sigma_s, \sigma_h) = \text{store}_{\varphi}(\sigma), a = \sigma'(1)$, and $(\sigma_s, \sigma_a) = \text{store}_{\varphi}(\sigma, a)$. Then, if $a \in \Sigma_a$, $\sigma_t = \sigma_s.a.\sigma'_a$, where $\sigma'_a$ is defined in Definition 8, meaning that $\sigma_s \not\approx \sigma_t$. If $a \in \Sigma_c$, then $\sigma_1 = \sigma_s.\sigma''_a$, where $\sigma''_a$ is defined in Definition 8, thus again, $\sigma_s \not\approx \sigma_1$. In both cases, $E_{\varphi}(\sigma) = \sigma_s \not\approx \sigma_1 = E_{\varphi}(\sigma, a)$. Since the order $\not\approx$ is transitive, this means that $E_{\varphi}(\sigma) \not\approx E_{\varphi}(\sigma, a) \not\approx E_{\varphi}(\sigma, a, \sigma'(2)) \not\approx \ldots \not\approx E_{\varphi}(\sigma, \sigma')$. Thus $E_\varphi$ is an enforcement function.

**Lemma 1.** $\forall \sigma \in \Sigma^*, \forall a \in \Sigma_a, I(\sigma) \subseteq I(\sigma.a)$.

**Proof.** For $\sigma \in \Sigma^*$, let $P(\sigma)$ be the predicate “$\forall u \in \Sigma_a, I(\sigma) \subseteq I(\sigma.a)$”. Let us show by induction that $P(\sigma)$ holds for every $\sigma \in \Sigma^*$.

--- Induction basis: if $a \in \Sigma_a$, then since $I(\epsilon) = \emptyset$, $I(\epsilon) \not\subseteq I(a)$. Thus, $P(\epsilon)$ holds.

--- Induction step: let us suppose that for $n \in \mathbb{N}$, for all $\sigma \in \Sigma^*_n$ s.t. $|\sigma| \leq n$, $P(\sigma)$ holds. Let us then consider $\sigma \in \Sigma^*_n$ s.t. $|\sigma| = n+1$, and $a \in \Sigma_a$. Let $(h, \sigma_0) \in \Sigma_c \times \Sigma^*_n$ be s.t. $\sigma = h.\sigma_0$ (they must exist since $|\sigma| > 0$). Then, $|\sigma_0| = n$, and by induction hypothesis, $P(\sigma_0)$ holds, meaning that $I(\sigma_0) \not\subseteq I(\sigma_0.a)$. Moreover, following the definition of $S(\sigma_0.a), S(\sigma_0) \subseteq S(\sigma_0,a)$. It follows that $S(\sigma_0) \cup I(\sigma_0) \subseteq S(\sigma_0.a) \cup I(\sigma_0.a)$, and thus $I(\sigma) = I(h.\sigma_0) = \text{Pred}_h(S(\sigma_0) \cup I(\sigma_0)) \subseteq \text{Pred}_h(S(\sigma_0.a) \cup I(\sigma_0.a)) = I(h.\sigma_0.a) = I(\sigma.a)$. This means that $P(\sigma.a)$ holds.

Thus, by induction on the size of $\sigma \in \Sigma^*_n$, for all $\sigma \in \Sigma^*_n$, $P(\sigma)$ holds. This means that for all $\sigma \in \Sigma^*_n$, for all $a \in \Sigma_a$, $I(\sigma) \subseteq I(\sigma.a)$.

**Lemma 2.** $\forall \sigma \in \Sigma^*_n, \forall q \in Q, \forall u \in \Sigma_u, (q \in S(\sigma)) \implies (q \text{ after } u \in S(\sigma) \cup I(\sigma))$.

**Proof.** For $\sigma \in \Sigma^*_n$, let $P(\sigma)$ be the predicate “$\forall q \in Q, \forall u \in \Sigma_u, (q \in S(\sigma)) \implies (q \text{ after } u \in S(\sigma) \cup I(\sigma))$”. Let us show by induction that $P(\sigma)$ holds for any $\sigma \in \Sigma_c$.

--- Induction basis: let us consider $u \in \Sigma_u$ and $q \in S(\epsilon)$. Then, since $u \in \Sigma_u$, $u \in \Sigma^*_u$, and following the definition of $S(\epsilon)$, $q$ after $u \in S(\epsilon)$, Thus, $q$ after $u \in S(\epsilon) \cup I(\epsilon)$. 

--- Induction step: let us consider $a \in \Sigma_a$, and $q \in S(\sigma)$. Then, since $a \in \Sigma_a$, $a \in \Sigma^*_a$, and following the definition of $S(\epsilon)$, $q$ after $a \in S(\epsilon)$, Thus, $q$ after $a \in S(\epsilon) \cup I(\epsilon)$.
— **Induction step:** let us suppose that for \( \sigma \in \Sigma^*_n \), \( P(\sigma) \) holds. Let us then consider \( u \in \Sigma_n \), \( a \in \Sigma_n \), and \( q \in S(\sigma.a) \). Then, either \( q \in S(\sigma) \) or \( q \in \max_{\Sigma}(\{Y \subseteq F_C \mid Y \cap uP_{\Sigma}(V \cup I(\sigma.a)) = \emptyset\}) \). If \( q \in S(\sigma) \), then by induction hypothesis, \( P(\sigma) \) holds, meaning that \( q \) after \( u \in S(\sigma) \cup I(\sigma) \). Following lemma 1, \( I(\sigma) \subseteq \Sigma(\sigma.a) \), and since \( S(\sigma) \subseteq S(\sigma.a) \), it follows that \( S(\sigma) \cup I(\sigma) \subseteq S(\sigma.a) \cup I(\sigma.a) \). Thus, \( q \) after \( u \in S(\sigma.a) \cup I(\sigma.a) \). Otherwise, \( q \in \max_{\Sigma}(\{Y \subseteq F_C \mid Y \cap uP_{\Sigma}(V \cup I(\sigma.a)) = \emptyset\}) \), and thus \( q \) after \( u \in S(\sigma.a) \cup I(\sigma.a) \). Thus, \( P(\sigma.a) \) holds.

By induction on \( \sigma \), it follows that \( P(\sigma) \) holds for any \( \sigma \in \Sigma^*_n \). Thus, for all \( \sigma \in \Sigma^*_n \), for all \( u \in \Sigma_n \), for all \( q \in Q, (q \text{ after } u \in S(\sigma) \cup I(\sigma)) \).

**Lemma 3.** \( \forall \sigma \in \Sigma^*_n, \forall q \in S(\sigma) \cup I(\sigma), G(q, \sigma) \neq \emptyset. \)

**Proof.** For \( \sigma \in \Sigma^*_n \), let \( P(\sigma) \) be the predicate \( \\forall q \in S(\sigma) \cup I(\sigma), G(q, \sigma) \neq \emptyset. \). Let us show by induction that \( P(\sigma) \) holds for any \( \sigma \in \Sigma^*_n \).

**Induction basis:** let us consider \( q \in S(\epsilon) \cup I(\epsilon) \). Then, since \( I(\epsilon) = \emptyset, q \in S(\epsilon) \). Following the definition of \( S(\epsilon) \), this means that \( \epsilon \) is s.t. \( \epsilon \notin \epsilon \) and \( q \) after \( \epsilon = q \in S(\epsilon) = S(\epsilon^{-1}.\epsilon) \). Thus, \( \epsilon \in G(q, \sigma.a) \).

**Induction step:** let us suppose that for \( n \in \mathbb{N} \), for all \( \sigma \in \Sigma^*_n \) s.t. \( |\sigma| \leq n \), \( P(\sigma) \) holds. Let us then consider \( \sigma \in \Sigma^*_n \) s.t. \( |\sigma| = n, a \in \Sigma_n \), and \( q \in S(\sigma.a) \cup I(\sigma.a) \). Then, we consider two cases:

- \( q \in S(\sigma.a) \), then \( \epsilon \) is s.t. \( \epsilon \notin \sigma.a \) and \( q \) after \( \epsilon = q \in S(\sigma.a) = S(\epsilon^{-1}.(\sigma.a)) \), thus \( \epsilon \in G(q, \sigma.a) \).

- \( q \in I(\sigma.a) \), then let \( (h, \sigma_0) \in \Sigma \times \Sigma \) be s.t. \( h.\sigma_0 = \sigma.a \) (they must exist since \( |\sigma.a| > 0 \)). Then, \( I(\sigma.a) = I(h, \sigma_0) \) by induction hypothesis, since \( |\sigma_0| = |\sigma| = n, P(\sigma_0) \) holds, meaning that \( G(q, h, \sigma_0) \neq \emptyset \). Let us then consider \( w = G(q, h, \sigma_0) \). Then, \( w = w \notin \sigma_0 \) and \( q \) after \( h \) after \( w \) after \( S(w^{-1}.\sigma_0) = S((h.w)^{-1}.(h.\sigma_0)) \). Thus, \( h.w \notin \sigma.a \) and \( q \) after \( h \) after \( w \) after \( S(w^{-1}.\sigma_0) = S((h.w)^{-1}.(h.\sigma_0)) \). Thus, \( h.w \in G(q, (h, \sigma_0)) \). In both cases, \( G(q, \sigma.a) \neq \emptyset \), meaning that \( P(\sigma.a) \) holds. By induction on the size of \( \sigma \in \Sigma^*_n \), it follows that \( P(\sigma) \) holds for any \( \sigma \in \Sigma^*_n \), for all \( q \in S(\sigma) \cup I(\sigma), G(q, \sigma) \neq \emptyset. \).

**Lemma 4.** \( \forall \sigma \in \Sigma^*, (\sigma \notin \text{Pre}(\varphi) \land (\sigma.s, \sigma.e) = \text{store}_\varphi(\sigma)) \Rightarrow (\sigma_s = \sigma|_{\Sigma_n} \land \sigma_e = \sigma|_{\Sigma_n}). \)

**Proof.** For \( \sigma \in \Sigma^* \), let \( P(\sigma) \) be the predicate \( (\sigma \notin \text{Pre}(\varphi) \land (\sigma.s, \sigma.e) = \text{store}_\varphi(\sigma)) \Rightarrow (\sigma_s = \sigma|_{\Sigma_n} \land \sigma_e = \sigma|_{\Sigma_n}). \). Let us show by induction that \( P(\sigma) \) holds for any \( \sigma \in \Sigma^* \).

**Induction basis:** \( \text{store}_\varphi(\epsilon) = (\epsilon, \epsilon) \), and since \( \epsilon|_{\Sigma_n} = \epsilon|_{\Sigma_n} = \epsilon = P(\epsilon) \).

**Induction step:** let us suppose that for \( \sigma \in \Sigma^* \), \( P(\sigma) \) holds. Let us then consider \( a \in \Sigma, (\sigma.s, \sigma.e) = \text{store}_\varphi(\sigma) \). Then, if \( \sigma.a \notin \text{Pre}(\varphi) \), \( P(\sigma.a) \) holds. Let us now consider that \( \sigma.a \notin \text{Pre}(\varphi) \). Then, since \( \text{Pre}(\varphi) \) is extension-closed, \( \sigma \notin \text{Pre}(\varphi) \), and thus, by induction hypothesis, \( \sigma_s = \sigma|_{\Sigma_n} \land \sigma_e = \sigma|_{\Sigma_n} \). We consider two cases:

- \( a \in \Sigma_n \), then \( \sigma_1 = \sigma.a, \sigma_2 \), with \( \sigma_1 = G(\text{Reach}(\sigma.a), \sigma.e) \cup \{\epsilon\} \). Since \( \sigma.a \notin \text{Pre}(\varphi) \), \( G(\text{Reach}(\sigma.a), \sigma.e) \cup \{\epsilon\} \).

- \( a \in \Sigma_n \), then \( \sigma_1 = \sigma.a, \sigma_2 \), with \( \sigma_1 = G(\text{Reach}(\sigma.a), \sigma.e) \cup \{\epsilon\} \). Since \( \sigma.a \notin \text{Pre}(\varphi) \), \( G(\text{Reach}(\sigma.a), \sigma.e) \cup \{\epsilon\} \).

In both cases, \( P(\sigma.a) \) holds. By induction on \( \sigma \in \Sigma^* \), if \( \sigma \notin \text{Pre}(\varphi) \) and \( (\sigma.s, \sigma.e) = \text{store}_\varphi(\sigma) \), then \( \sigma_s = \sigma|_{\Sigma_n} \land \sigma_e = \sigma|_{\Sigma_n} \).
Proposition 2. \( E_{\varphi} \) is sound with respect to \( \varphi \) in \( \text{Pre}(\varphi) \), as per Definition 3.

Proof. Let \( P(\varphi) \) be the predicate: \((\sigma \in \text{Pre}(\varphi) \land (\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma)) \implies (E_{\varphi}(\sigma) \models \varphi \land \text{Reach}(\sigma) \in S(\sigma))\). Let us prove by induction that for any \( \sigma \) in \( \Sigma^* \), \( P(\sigma) \) holds.

—Induction basis: if \( \epsilon \in \text{Pre}(\varphi) \), then following the definition of \( \text{Pre}(\varphi) \), \( G(\text{Reach}(\epsilon), \epsilon) \neq \emptyset \). Thus \( \epsilon \in G(\text{Reach}(\epsilon), \epsilon) \) (since \( \epsilon \) is the only word satisfying \( \epsilon \subseteq \epsilon \)). This means that \( \text{Reach}(\epsilon) \) after \( \epsilon = \text{Reach}(\epsilon) \) in \( S(\epsilon) \). Considering that \( \text{store}_\varphi(\epsilon) = (\epsilon, \epsilon) \), it follows that \( E_{\varphi}(\epsilon) = \epsilon \), and thus, since \( S(\epsilon) \subseteq F_G, E_{\varphi}(\epsilon) \models \varphi \). Thus \( P(\epsilon) \) holds.

—Induction step: Suppose now that, for \( \sigma \in \Sigma^* \), \( P(\sigma) \) holds. Let us consider \( a \in \Sigma_c \): \((\sigma, \sigma_a) = \text{store}_\varphi(\sigma)\), and \((\sigma, \sigma_a) = \text{store}_\varphi(\sigma, a)\). Let us prove that \( P(\sigma, a) \) holds. We consider three different cases:

\(- (\sigma, a) \not\in \text{Pre}(\varphi) \), then \( P(\sigma, a) \) holds.

\(- (\sigma, a) \in \text{Pre}(\varphi) \land \sigma \not\in \text{Pre}(\varphi) \). Then, since \( \text{Pre}(\varphi) \) is extension-closed, it follows that \( \sigma \cdot a \in \{ w \in \Sigma^* \mid G(\text{Reach}(w), w) \neq \emptyset \} \), meaning that \( G(\text{Reach}(\sigma \cdot a), \sigma) \neq \emptyset \). Moreover, since \( \sigma \not\in \text{Pre}(\varphi) \), following Lemma 4, \( \sigma_s = \sigma_{\Sigma_s} \) and \( \sigma_c = \sigma_{\Sigma_c} \). Now, we consider two cases:

\(- \) If \( a \in \Sigma_a \), then \((\sigma, a)_{\Sigma_a} = \sigma_{\Sigma_a} \cdot a = \sigma_a, \) and \((\sigma, a)_{\Sigma_c} = \sigma_{\Sigma_c} = \sigma_c. \) Thus, \( G(\text{Reach}(\sigma, a), \sigma_a) \), and \( G(\text{Reach}(\sigma, a), \sigma_c) \neq \emptyset \), following the definition of \( G \).

\(- \) If \( a \in \Sigma_c \), then \((\sigma, a)_{\Sigma_a} = \sigma_{\Sigma_a} = \sigma, \) and \((\sigma, a)_{\Sigma_c} = \sigma_{\Sigma_c} = \sigma_a. \) Thus, \( G(\text{Reach}(\sigma), \sigma_a) \), and \( G(\text{Reach}(\sigma), \sigma_a) \neq \emptyset \), following the definition of \( G \).

Thus, if \( \sigma \in \text{Pre}(\varphi) \) but \( \sigma \not\in \text{Pre}(\varphi) \), \( P(\sigma, a) \) holds.

In all cases, \( P(\sigma, a) \) holds. Thus, \( P(\sigma) \implies P(\sigma, a) \). By induction on \( \sigma, \forall \sigma \in \Sigma^*, (\sigma \in \text{Pre}(\varphi) \land (\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma)) \implies (E_{\varphi}(\sigma) \models \varphi \land \text{Reach}(\sigma) \in S(\sigma_b)) \). In particular, for all \( \sigma \in \Sigma^* \), \((\sigma \in \text{Pre}(\varphi)) \implies (E_{\varphi}(\sigma) \models \varphi) \). This means that \( E_{\varphi} \) is sound with respect to \( \varphi \) in \( \text{Pre}(\varphi) \).

Proposition 3. \( E_{\varphi} \) is compliant, as per Definition 4.

Proof. For \( \sigma \in \Sigma^* \), let \( P(\sigma) \) be the predicate: \((\sigma_s \in \Sigma_s, \sigma_c = \text{store}_\varphi(\sigma)) \implies (E_{\varphi}(\sigma) \models \varphi \land \text{Reach}(\sigma) \in S(\sigma))\). Let us prove that for all \( \sigma \in \Sigma^* \), \( P(\sigma) \) holds.

—Induction basis: \( \text{store}_\varphi(\epsilon) = (\epsilon, \epsilon) \), and \( \epsilon_{\Sigma_s} = \epsilon_{\Sigma_a} \epsilon \), and \( \epsilon_{\Sigma_c} = \epsilon_{\Sigma_a} \epsilon \). Thus \( P(\epsilon) \) holds.

—Induction step: Let us suppose that for \( \sigma \in \Sigma^* \), \( P(\sigma) \) holds. Let us consider \((\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma), a \in \Sigma \), and \((\sigma_s, \sigma_d) = \text{store}_\varphi(\sigma, a)\). Let us prove that \( P(\sigma, a) \) holds.

—Case 1: \( a \in \Sigma_a \). Then, \( \sigma_s = \sigma_s \cdot a \cdot \sigma_s' \), where \( \sigma_s' \) is defined in Definition 8, and \( \sigma_s \cdot a \cdot \sigma_s = \sigma_s \cdot a \cdot \sigma_s \). Therefore, \( \sigma_s \cdot \sigma_s' = (\sigma_s, \sigma_s) \), since \( \sigma_d \in \Sigma_s \). Thus, \( \sigma_s \cdot \sigma_s = \sigma_s \cdot \sigma_s \). Since \( P(\sigma) \) holds, \( \sigma_s \cdot \sigma_s = \sigma_s \cdot \sigma_s = (\sigma_s, \sigma_s) \).
Moreover, since $\sigma'_u \in \Sigma'_u$, $\sigma_i(\Sigma_u) = \sigma_1(\Sigma_u)\cdot a = (\sigma.a)|_{\Sigma_u}$.
Thus $P(\sigma.a)$ holds.

Case 2: $a \in \Sigma_u$. Then $\sigma_1 = \sigma.s, \sigma'_u$, where $\sigma'_u$ is defined in Definition 8, and $\sigma_1, \sigma_d = \sigma.s, \sigma.e, a$. Therefore, $\sigma_i(\Sigma_u) = (\sigma.s)|_{\Sigma_u} = \sigma.1(\Sigma_u)\cdot a$. Thus, since $P(\sigma)$ holds, this means that $\sigma_i(\Sigma_u) = \sigma.1(\Sigma_u)\cdot a = (\sigma.a)|_{\Sigma_u}$.
Moreover, since $\sigma'_u \in \Sigma_u$, $\sigma_i(\Sigma_u) = \sigma.1(\Sigma_u)$. Thus $P(\sigma)$ holds, this means that $\sigma_i(\Sigma_u) = \sigma.1(\Sigma_u) = (\sigma.a)|_{\Sigma_u}$.

Thus $P(\sigma,a)$ holds.

In both cases, $P(\sigma,a)$ holds. Thus, for all $\sigma \in \Sigma^*$, for all $a \in \Sigma$, $P(\sigma) \implies P(\sigma.a)$.

By induction on $\sigma$, for all $\sigma \in \Sigma^*$. $(\sigma.s, \sigma.e) = \text{store}_\varphi(\sigma) \implies (\sigma_1, \sigma.1) = (\sigma.s, \sigma.e, a) = \text{store}_\varphi(\sigma.a)$, where $\sigma_i$ is defined in Definition 8. Thus $\sigma.s.s \equiv \sigma_1$, and since $\sigma_1 = \text{store}_\varphi(\sigma)$, and $\sigma_1 = \text{store}_\varphi(\sigma.u)$, it follows that $E_\varphi(\sigma.u) \equiv E_\varphi(\sigma.u)$. Thus, for all $\sigma \in \Sigma^*$, $E_\varphi(\sigma) \equiv \text{store}_\varphi(\sigma.u) = \text{store}_\varphi(\sigma.1) \equiv \text{store}_\varphi(\sigma) = \text{store}_\varphi(\sigma.1) \equiv E_\varphi(\sigma.u)$, meaning that $E_\varphi$ is compliant.

Lemma 5. $\forall \sigma \in \Sigma^*_w, \forall q \notin Q, (q \notin S(\sigma)) \implies (\exists \sigma \in \Sigma^*_w, q \notin S(\sigma) \equiv \forall \sigma, \sigma' \equiv q \implies q \notin S(\sigma) \equiv \forall q \notin S(\sigma)$.

Proof. For $\sigma \in \Sigma^*_w$ and $q \in Q$, let $P(\sigma, q)$ be the predicate “$\forall q \in Q, q \notin S(\sigma)$”. Let us show the contrapositive of the lemma, that is, that for all $\sigma \in \Sigma^*_w$ and $q \notin Q, P(\sigma, q)$ $\iff q \in S(\sigma)$. We consider two cases:

If $\sigma = \epsilon$, let us consider $q \in Q$ s.t. $P(\epsilon, q)$ holds. Then, since $\epsilon \in \Sigma^*_w$ and there does not exist a word $w$ satisfying $w \equiv q \wedge w \neq \epsilon$, it follows that $q = q \neq w \notin F$. Let us consider $\sigma = \Sigma^*_w$. Then, since $P(\epsilon, q)$ holds, either $q \notin F$ or there exists $\sigma = \sigma \equiv q$ such that $q \notin \epsilon$ and $q \notin S(\sigma) \equiv \forall q \notin S(\epsilon)$.

In this last case, since $I(\epsilon) = \emptyset$, $q \notin S(\epsilon)$, $q \notin S(\epsilon) \equiv \forall q \notin S(\epsilon)$.

Thus, in all cases $q \notin F$.

If $\sigma \neq \epsilon$, there exists $\sigma' \in \Sigma^*_w$ and $a \in \Sigma$ s.t. $\sigma = a'\cdot a$, meaning that $S(\sigma)$ is s.t. $S(\sigma) = S(\sigma') \cup \max_{\Sigma^*}(\emptyset)$. Let us consider $q \in Q$ s.t. $P(\sigma, q)$ holds. Then, we define $Y = \{q \in S(\sigma) \mid \forall \sigma' \equiv \epsilon \equiv q \implies q \notin S(\sigma) \cup I(\sigma)\}$. Since $P(\sigma, q)$ holds, $\forall Y \subseteq F$. Moreover, if $y \in Y$ and $u \in \Sigma$, then:

Either $y \notin q \in S(\sigma) \cup I(\sigma)$, or $Y \in (Y \cup (S(\sigma) \cup I(\sigma)))$.

Thus $y \notin q \in S(\sigma) \cup I(\sigma)$, and then $y \in (Y \cup (S(\sigma) \cup I(\sigma)))$.

Thus, $y \notin q \in S(\sigma) \cup I(\sigma)$, and since following lemma 2, $S(\sigma) \cap \max_{\Sigma^*}(\emptyset) = \emptyset$, this means that $S(\sigma) \cap \max_{\Sigma^*}(\emptyset) = \emptyset$. It follows that $\forall q \notin S(\sigma) \equiv \forall q \notin S(\sigma) \cup I(\sigma)$.

Thus, for $\sigma \in \Sigma^*_w$ and $q \in Q$, $P(\sigma, q) \iff q \in S(\sigma)$.

Proposition 4. $E_\varphi$ is optimal in $\text{Pre}(\varphi)$, by definition 5.

Proof. Let $E$ be an enforcement function s.t. compliant $(E, \Sigma_w, \Sigma_u)$, and let us consider $\sigma \in \text{Pre}(\varphi)$ and $a \in \Sigma$ s.t. $E(\sigma) = E_\varphi(\sigma)$ and $|E(\sigma.a)| > |E_\varphi(\sigma.a)|$. Let us also consider $(\sigma.s, \sigma.e) = \text{store}_\varphi(\sigma)$.

Let us show that there exists $\sigma_u \in \Sigma_u$ s.t. $E(\sigma.s, \sigma.e) \neq \varphi$. We consider two cases:
Proof. Let us introduce some notation for this proof: for a word $w \in \Gamma^{\ast}$, we note $\text{input}(w) = \Pi_{1}(w(1)), \Pi_{1}(w(2))\ldots \Pi_{1}(w(|w|))$, the word obtained by concatenating the first members (the inputs) of $w$. In a similar way, we note $\text{output}(w) = \Pi_{2}(w(1)), \Pi_{2}(w(2))\ldots \Pi_{2}(w(|w|))$, the word obtained by concatenating all the third members (outputs) of $w$. Since all configurations are not reachable from $c_{0}$, for $w \in \Gamma^{\ast}$, we note $\text{Reach}(w) = c$ whenever $c_{0}^{\ast}$ is \textit{\textbf{\&}} $\text{Reach}(w)$ and $\text{Reach}(w) = \bot$ if such a $c$ does not exist. We also define the Rules function as follows:

\[
\text{Rules : } \left\{ \begin{array}{c}
\Sigma^{\ast} \rightarrow \Gamma^{\ast} \\
\sigma \rightarrow \max_{<}(\{ w \in \Gamma^{\ast} \mid \text{input}(w) = \sigma \land \text{Reach}(w) \neq \bot \})
\end{array}\right.
\]

For a word $\sigma \in \Sigma^{\ast}$, $\text{Rules}(\sigma)$ is the trace of the last valid rules applied in $\epsilon$, i.e. the sequence of all the rules that can be applied with input $\sigma$. We then extend the definition of output to words in $\Sigma^{\ast}$: for $\sigma \in \Sigma^{\ast}$, $\text{output}(\sigma)$ is obtained by concatenating the first members of $\text{Rules}(\sigma)$. We also note $\epsilon$ the empty word of $\Sigma^{\ast}$, and $\epsilon^{\ast}$ the empty word of $\Gamma^{\ast}$.

For $\sigma \in \Sigma^{\ast}$, let $P(\sigma)$ be the predicate: $E_{\varphi}(\sigma) = \text{output}(\sigma) \land ((\sigma_{\epsilon}, \sigma_{\epsilon}) = \text{store}_{\sigma}(\sigma) \land \text{Reach}(\text{Rules}(\sigma)) = (q, \sigma_{\epsilon}^{\ast})$.

Let us prove by induction that for all $\sigma \in \Sigma^{\ast}$, $P(\sigma)$ holds.

-\textit{Induction basis:} $E_{\varphi}(\epsilon) = \epsilon = \text{output}(\epsilon)$. Moreover, $\text{store}(\epsilon) = (\epsilon, \epsilon)$, and $\text{Reach}(\epsilon^{\ast}) = c_{0}^{\ast}$. Therefore, as $c_{0}^{\ast} = (q_{0}, \epsilon), P(\epsilon)$ holds, because $\text{Reach}(\epsilon) = q_{0}$.

-\textit{Induction step:} Let us suppose now that for some $\sigma \in \Sigma^{\ast}$, $P(\sigma)$ holds. Let us consider $(\sigma_{\epsilon}, \sigma_{\epsilon}) = \text{store}_{\sigma}(\sigma), q = \text{Reach}(\sigma_{\epsilon}), a \in \Sigma$, and $(\sigma_{\epsilon}, \sigma_{\epsilon}) = \text{store}_{\sigma}(\sigma_{\epsilon})$. Let us prove that $P(\sigma_{\epsilon})$ holds.
Since $P(\sigma)$ holds, $\text{Reach}(\text{Rules}(\sigma)) = \langle q, \sigma_0 \rangle$ and $\sigma_x = \text{output}(\sigma)$. We consider two cases:

- $a \in \Sigma_u$. Then, considering $\sigma'_x = (\sigma_x.a)^{-1}.\sigma_x$, $\sigma_x = \sigma_x.a.\sigma'_x$. Since $a \in \Sigma_u$, rule pass-uncont can be applied: let us consider $q' = q$ after $a$. Then, $\langle q, \sigma_x \rangle \xrightarrow{a/\text{pass-uncont}(a)/a} \langle q', \sigma_x \rangle$. Thus, if $\sigma'_x = \epsilon$, $G(q', \sigma_x) = \emptyset$ or $G(q', \sigma_x) = \{\{\} \}$. meaning that no other rule can be applied, and thus $P(\sigma.a)$ would hold. Otherwise, $\sigma'_x \neq \epsilon$, and thus $\sigma'_x \in G(q', \sigma_x)$, meaning that $G(q', \sigma_x) \neq \emptyset$ and $G(q', \sigma_x) \neq \{\}$. Thus rule dump$(\sigma_x(1))$ can be applied. Since $\sigma'_x \not\subseteq \sigma_x$, $\sigma'_x(1) = \sigma_x(1)$, thus if $q_1 = q'$ after $\sigma_x(1)$, $q_1 = q'$ after $\sigma'_x(1)$. If $\sigma'_x(1)^{-1}.\sigma'_x \neq \epsilon$, then $\sigma_x(1)^{-1}.\sigma'_x \in G(q_1, \sigma_x(1)^{-1}.\sigma_x)$, meaning that rule dump can be applied again. Rule dump can actually be applied $|\sigma'_x|$ times, since for all $w \not\subseteq \sigma'_x$, then $w^{-1}.\sigma'_x \neq \epsilon$ and $w^{-1}.\sigma'_x \in G(q', w, w^{-1}.\sigma_x)$. Thus, after rule dump has been applied $|\sigma'_x|$ times, the configuration reached is $\langle q' \rangle a \sigma_x^{-1}.\sigma_x$. Moreover, the output produced by all the rules dump is $\sigma'_x$. Since no rule can be applied after the $|\sigma'_x|$ applications of the rule dump, $\text{output}(\sigma.a) = \text{output}(\sigma.a) \sigma_x = \sigma_x$, and $\text{Reach}(\text{Rules}(\sigma.a)) = \langle q' \rangle a \sigma_x^{-1}.\sigma_x = \langle q \rangle a \sigma_x \sigma_x = \langle q \rangle a \sigma_x$ after $\sigma_x, \sigma_x = (\text{Reach}(\sigma_x) a \sigma_x) = (\text{Reach}(\sigma_x, \sigma_x) a \sigma_x) = \text{Reach}(\sigma, \sigma_x) a \sigma_x$. Thus, if $a \in \Sigma_u$, $P(\sigma.a)$ holds.

- $a \in \Sigma_e$. Then, considering $\sigma''_x = \sigma_x^{-1}.\sigma_x$, $\sigma_x = \sigma_x.\sigma''_x$. Since $a \in \Sigma_e$, it is possible to apply the store-cont rule, and $\langle q, \sigma_x \rangle a/\text{store-cont}(a)/a = \langle q, \sigma_x.a \rangle$. Then as in the case where $a \in \Sigma_u$, rule dump can be applied $|\sigma''_x|$ times, meaning that the configuration reached would then be $\langle q \rangle a \sigma_x^{-1}.\sigma_x.\sigma''_x(1). \langle \sigma_x.a(1) \rangle \ldots \langle \sigma_x.a(|\sigma''_x|) \rangle$. Since $\sigma''_x \not\subseteq \sigma_x$, $\sigma''_x(1) = \sigma_x(1)$, thus Rule dump$(\sigma_x.a(1)) = \langle q \rangle a \sigma_x^{-1}.\sigma_x = \langle q \rangle a \sigma_x \sigma_x = \langle q \rangle a \sigma_x$ after $\sigma_x, \sigma_x = (\text{Reach}(\sigma_x.a(1)) a \sigma_x) = (\text{Reach}(\sigma_x.a.\sigma''_x) a) = \text{Reach}(\sigma_x.a) a \sigma_x$. Moreover, $\text{output}(\sigma.a) = \text{output}(\sigma.a) \sigma''_x = \sigma_x, \sigma''_x = \sigma_x = \text{output}(\sigma.a) a \sigma_x$.

Thus, if $a \in \Sigma_e$, $P(\sigma.a)$ holds. This means that $P(\sigma) \implies P(\sigma.a)$. Thus, by induction on $\sigma$, for all $\sigma \in \Sigma^*$, $P(\sigma)$ holds. In particular, for all $\sigma \in \Sigma^*$, $E_x(\sigma) = \text{output}(\sigma)$.

\begin{proof}
For $a \in \text{tw}(\Sigma)$, let $P(\sigma)$ be the predicate: \"$\forall t \in \mathbb{R}_{\geq 0}, \forall t' \geq t, E_x(\sigma, t) \leq E_x(\sigma, t')$\". Let us show by induction that for all $a \in \text{tw}(\Sigma)$, $P(\sigma)$ holds.

- **Induction basis:** $\sigma = \epsilon$. Then, let us consider $t \in \mathbb{R}_{\geq 0}$, and $t' \geq t$. Then, $E_x(\epsilon, t) = \epsilon \leq \epsilon = E_x(\epsilon, t')$.

Thus, $P(\epsilon)$ holds.

- **Induction step:** let us suppose that, for $a \in \text{tw}(\Sigma)$, $P(\sigma)$ holds. Let us consider $\langle t', a \rangle$ s.t. $\sigma.\langle t', a \rangle \in \text{tw}(\Sigma), t \in \mathbb{R}_{\geq 0}$, and $t' \geq t$.

- If $t \geq t'$, then let us consider $\langle \sigma_x, \sigma_x, \sigma_x \rangle = \text{store}_x(\sigma, t'), \langle \sigma_x, \sigma_x, \sigma_x \rangle = \text{store}_x(\sigma, \sigma_x, a, t)$, and $\langle \sigma_x, \sigma_x, \sigma_x \rangle = \text{store}_x(\sigma, t', a, t')$. Then, $E_x(\sigma, (t', a), t) = \sigma_x$ and $E_x(\sigma, (t', a), t') = \sigma_x.

- If $t \leq t'$, then let us consider $t_1 = \min\{\{t_0 \in \mathbb{R}_{\geq 0} | t_0 \geq t' \wedge G(\text{Reach}(\sigma_x, (t', a), t_0), \Pi_x(\text{obs}(\sigma_x, t')), \sigma_x) \neq \emptyset)\}$. Then, $\sigma_{d1} = \sigma_{d2} = \text{store}_x(\sigma, \sigma_x, a, t_1) \cup \{t, t_1\}$. Then, $E_x(\sigma, (t', a), t) = \sigma_x$ and $\sigma_{d2} = \text{store}_x(\sigma, \sigma_x, a, t_1) \cup \{t, t_1\}$.

- Case 1: $t \geq t_1$. Since $t' \geq t$, then $t' \geq t_1$, thus $\min\{\{t_1, t_1\} = \min\{t, t_1\} = t_1$, thus $\sigma_{d1} = \sigma_{d2}$. It follows that:

  - $\sigma_{d1} = \sigma_x(t', a), \sigma_x \sigma_x(a) \sigma_x(t', a), \sigma_x \sigma_x(t', a) \sigma_x \sigma_x(t', a) \sigma_x \sigma_x(t', a) \sigma_x \sigma_x(t', a)$.

- Case 2: $t' < t$. Then, $\min\{\{t_1, t_1\} = t$. Since $t < t_1$, by definition of $t_1$, this means that $G(\text{Reach}(\sigma_x, (t', a), t), \Pi_x(\text{obs}(\sigma_x, t')), \sigma_x) = \emptyset$, and $\sigma_{d1} = \epsilon$. Since $\sigma_{d1} = \epsilon, \sigma_{d1} = \sigma_x(t', a) \sigma_x(t', a) \sigma_x(t', a) \sigma_x(t', a)$.

Thus, $E_x(\sigma, t') = \sigma_x$.
Thus, if \( t' \geq t \geq t'' \) and \( a \in \Sigma_u \), \( P(\sigma) \Rightarrow E_e(\sigma(t'',a),t) \Leftrightarrow E_e(\sigma(t'',a),t') \).

- Otherwise, \( a \in \Sigma_u \), and then considering \( t_2 \) as defined in Definition 16, \( t_2 = \min\{ \{ t_0 \in \mathbb{R}_{\geq 0} \mid t_0 \geq t'' \wedge G(\text{Reach}(\sigma_a, t_0), P_2(\text{obs}(\sigma_b, t'''))) \cup \{ c \} \neq \emptyset \} \} \). Then, \( \sigma_{\text{da}} = \min_{\text{obs}}(\max_{\text{obs}}(G(\text{Reach}(\sigma_a, \{ t, t_2 \})), P_{\Sigma_2}(\text{obs}(\sigma_b, t'''), \sigma_c \cup \{ c \}) + \min(\{ t, t_2 \})) \), and:

\[
\sigma_{\text{da}} = \min_{\text{obs}}(\max_{\text{obs}}(G(\text{Reach}(\sigma), \{ t', t_2 \}), P_{\Sigma_2}(\text{obs}(\sigma_b, t''')) \cup \{ c \}) + \min(\{ t', t_2 \})
\]

- Case 1: \( t \geq t_2 \). Since \( t' \geq t, t' \geq t_2 \), meaning that \( \min(\{ t, t_2 \}) = \min(\{ t', t_2 \}) = t_2 \), and thus \( \sigma_{\text{da}} = \sigma_{\text{da}} \). It follows that \( \sigma_{\text{da}} = \sigma_c \).

- Case 2: \( t < t_2 \). Then, \( G(\text{Reach}(\sigma_a, \min(\{ t, t_2 \})), P_{\Sigma_2}(\text{obs}(\sigma_b, t''')) \cup \{ c \}) = 0 \), meaning that \( \sigma_{\text{da}} = \epsilon \). Thus, \( \sigma_{\text{da}} = \sigma_c \Leftrightarrow \text{obs}(\sigma_{\text{da}}, t') = \sigma_{\text{da}} \).

Thus, if \( t' \geq t \geq t'' \) and \( a \in \Sigma_u \), \( P(\sigma) \Rightarrow E_e(\sigma(t'',a),t) \Leftrightarrow E_e(\sigma(t'',a),t') \).

Therefore, if \( t' \geq t \geq t'' \), for all \( a \in \Sigma_u \), \( P(\sigma) \Rightarrow E_e(\sigma(t'',a),t) \Leftrightarrow E_e(\sigma(t'',a),t') \).

- If \( t' < t'' \), then \( t < t'' \), and \( \text{obs}(\sigma(t'',a),t) = \text{obs}(\sigma,t) \), and \( \text{obs}(\sigma(t'',a),t') = \text{obs}(\sigma,t) \). Thus, \( E_e(\sigma(t'',a),t) = \text{store}_e(\text{obs}(\sigma(t'',a),t'),t) = \text{store}_e(\text{obs}(\sigma,t'),t) = \text{store}_e(\text{obs}(\sigma,t') = E_e(\sigma,t) \). Since \( P(\sigma) \) holds, then \( E_e(\sigma(t'',a),t) = E_e(\sigma,t) \). \( \text{iff} \) \( E_e(\sigma(t'',a),t') = E_e(\sigma,t') \).

- If \( t < t' \leq t'' \), then \( \text{obs}(\sigma(t'',a),t) = \text{obs}(\sigma,t) \). Since \( P(\sigma) \) holds, then \( E_e(\sigma,t) \Leftrightarrow E_e(\sigma,t') \).

Let \( \sigma_{\text{b}}, \sigma_{\text{c}} = \text{store}_e(\sigma,t) \) and \( \text{store}_e(\sigma(t'',a),t) = \text{store}_e(\sigma(t'',a),t') \). Then, \( \sigma_{\text{b}} = \sigma_{\text{b}}, \sigma_{\text{c}} \).

Thus, if \( t < t' \leq t'' \), then \( P(\sigma) \Rightarrow E_e(\sigma(t'',a),t) \Leftrightarrow E_e(\sigma,t') \).

Consequently, in all cases, if \( t \leq t' \), then \( P(\sigma) \Rightarrow E_e(\sigma(t'',a),t) \Leftrightarrow E_e(\sigma(t'',a),t') \).

Finally, \( P(\sigma) \Rightarrow P(\sigma(t'',a),t) \).

By induction, for all \( \sigma \in \text{tw}(\Sigma) \), \( P(\sigma) \) holds. Thus, for all \( \sigma \in \text{tw}(\Sigma) \), for all \( t \in \mathbb{R}_{\geq 0} \), for all \( t' \geq t \), \( E_e(\sigma, t) \Leftrightarrow E_e(\sigma,t') \).

Now, let us consider \( \sigma \in \text{tw}(\Sigma) \), and \( (t, a) \) s.t. \( \sigma(t,a) \in \text{tw}(\Sigma) \). Then, if \( (\sigma, \sigma_b, \sigma_c) = \text{store}_e(\sigma,t) \) and \( (\sigma, \sigma_d, \sigma_e) = \text{store}_e(\sigma(t,a),t) \), then either \( \sigma_t = \sigma_{t'} \), or \( \sigma = \sigma'_{\Sigma_{\text{c}}} \), or \( \sigma = \sigma''_{\Sigma_{\text{c}}} \). This means that \( \sigma_{\text{c}} \in \Sigma_{\text{c}} \).

Thus, \( E_e(\sigma,t) = \text{store}_e(\text{obs}(\sigma,t),t) = \sigma_{\text{c}} = \sigma_{\text{c}} = \text{store}_e(\text{obs}(\sigma,t),t) = \text{store}_e(\text{obs}((\sigma,t),t)) \).

This holds because, since \( \sigma(t,a) \in \text{tw}(\Sigma) \), \( \text{time}(t) \leq t \), thus \( \text{obs}(\sigma,t) = \sigma \).

Thus, for all \( \sigma \in \text{tw}(\Sigma) \), for all \( t \in \mathbb{R}_{\geq 0} \), \( \text{time}(t) \leq t \), \( E_e(\sigma,t) \Leftrightarrow E_e(\sigma,t') \) and \( E_e(\sigma,t) \Leftrightarrow E_e(\sigma(t,a),t) \). This means that \( \text{time}(t) \) is an enforcement function.

Lemma 6. \( \forall t \in \mathbb{R}_{\geq 0}, \forall \sigma \in \text{tw}(\Sigma), (\sigma \notin \text{Pre}(\varphi,t) \wedge (\sigma_{\text{b}}, \sigma_{\text{c}}) = \text{store}_e(\sigma,t)) \Rightarrow (\sigma_{\text{b}} = \sigma_{\Sigma_{\text{c}}} = \sigma_{\text{e}} = \epsilon = \sigma = \Sigma_{\Sigma_{\text{c}}}) \).

Proof. For \( \sigma \in \text{tw}(\Sigma) \), let \( P(\sigma) \) be the predicate "\( \forall t \geq \text{time}(t), (\sigma \notin \text{Pre}(\varphi,t) \wedge (\sigma_{\text{b}}, \sigma_{\text{c}}) = \text{store}_e(\sigma,t)) \Rightarrow (\sigma = \sigma_{\Sigma_{\text{c}}} = \sigma_{\text{e}} = \epsilon = \sigma = \Sigma_{\Sigma_{\text{c}}}) \).

- Induction basis: \( \sigma = \epsilon \). Let us consider \( t \in \mathbb{R}_{\geq 0} \). Then, \( \text{store}_e(\epsilon,t) = (\epsilon,\epsilon,\epsilon) \). Considering that \( \epsilon \in \text{tw}(\Sigma) \), and \( \epsilon = \Pi_{\Sigma_{\text{c}}}(\epsilon_{\Sigma_{\text{c}}}) \).

- Induction step: suppose that for \( \sigma \in \text{tw}(\Sigma) \), \( P(\sigma) \) holds. Let us consider \( (t',a) \) s.t. \( (\sigma(t',a) \in \text{tw}(\Sigma) \) and \( \text{time}(t') \leq t' \). Let us also consider \( \sigma(a) = \text{store}_e(\sigma,t') \) and \( \sigma(t,a) = \text{store}_e(\sigma(t',a),t) \).

Then, if \( \sigma(t',a) \in \text{Pre}(\varphi,t), \text{Pre}(\sigma(t',a)) \) trivially holds. Thus, let us suppose that \( \sigma(t',a) \notin \text{Pre}(\varphi,t) \). Since \( \sigma(t',a) \notin \text{Pre}(\varphi,t) \) and \( t' \geq t' \), it follows that \( \sigma \notin \text{Pre}(\varphi,t) \). By induction hypothesis, this means that \( \sigma = \sigma_{\Sigma_{\text{c}}} = \sigma_{\text{e}} = \epsilon = \sigma = \Pi_{\Sigma_{\text{c}}}(\sigma_{\Sigma_{\text{c}}}) \). Then, since \( \sigma(t',a) \notin \text{Pre}(\varphi,t) \), following the definition of \( \text{Pre}(\varphi,t) \), this means that for all \( t' \leq t \), \( G(\text{Reach}(\text{obs}(\sigma(t',a),t'''),t'')) = \text{store}_e(\text{obs}(\sigma(t',a),t'')) \).

Therefore, if \( t' \geq t \geq t'' \) and \( a \in \Sigma_u \), \( P(\sigma) \Rightarrow E_e(\sigma(t'',a),t) \Leftrightarrow E_e(\sigma(t'',a),t') \).
Induction step:

Induction basis:

Let us consider \( \emptyset \). In particular, \( G(\text{Reach}(\sigma,t))_{\Sigma_\sigma} = \emptyset \), where \( t \in \text{Definition} \). This means that \( t < t_1 \), where \( t_1 \) is defined in Definition 16, and thus \( \sigma_{t_1} = \epsilon \). Since \( \sigma_0 = \epsilon \), for all \( t \) such that \( t < t_2 \), where \( t_2 = \text{Definition} \), we have \( G(\text{Reach}(\sigma,t))_{\Sigma_\sigma} = \emptyset \). Thus, \( P(\sigma.(t,a)) \) holds if \( a \in \Sigma_\sigma \).

Case 1: \( a \in \Sigma_\sigma \). Then, for all \( t \in \emptyset \), let us consider \( \epsilon = \emptyset \). In particular, \( G(\text{Reach}(\sigma,t))_{\Sigma_\sigma} = \emptyset \). Thus, \( P(\sigma.(t,a)) \) holds if \( a \in \Sigma_\sigma \).

Case 2: \( a \in \Sigma_\sigma \). Then, \( \text{Definition} \). Therefore, \( \emptyset \). For all \( t \in \emptyset \), let us consider \( \epsilon = \emptyset \). In particular, \( G(\text{Reach}(\sigma,t))_{\Sigma_\sigma} = \emptyset \). Thus, \( P(\sigma.(t,a)) \) holds if \( a \in \Sigma_\sigma \).

By induction, for all \( t \in \Sigma_\sigma \), \( P(\sigma) \) holds. Thus, for all \( t \in \Sigma_\sigma \), \( P(\sigma) \) holds for all \( t \in \emptyset \), if \( \sigma_0 = \emptyset \), \( \sigma_2 = \emptyset \), \( \sigma_2 = \emptyset \), \( \sigma_2 = \emptyset \), \( \sigma_2 = \emptyset \), \( \sigma_2 = \emptyset \), \( \sigma_2 = \emptyset \), \( \sigma_2 = \emptyset \). Thus, \( P(\sigma) \) holds if \( a \in \Sigma_\sigma \).

\( \forall \sigma \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \text{Proof} \)

For all \( \sigma \in \Sigma_\sigma \), let us consider \( \forall \sigma \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \). Thus, \( P(\sigma) \) holds if \( a \in \Sigma_\sigma \).

\( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \). Thus, \( P(\sigma) \) holds if \( a \in \Sigma_\sigma \).

\( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \). Thus, \( P(\sigma) \) holds if \( a \in \Sigma_\sigma \).

\( \forall q \in Q \), \( \forall a \in \Sigma_\sigma \), \( \forall q \in Q \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \). Thus, \( P(\sigma) \) holds if \( a \in \Sigma_\sigma \).

\( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \), \( \forall a \in \Sigma_\sigma \). Thus, \( P(\sigma) \) holds if \( a \in \Sigma_\sigma \).
Lemma 9. For all $\sigma \in \Sigma^*$, for all $q \in Q$, $(q \in S(\sigma) \cup I(\sigma)) \implies (G(q, \sigma) \neq \emptyset)$.

Proof. Let $\sigma \in \Sigma^*$, let $P(\sigma)$ be the predicate “$\forall q \in Q, (q \in S(\sigma) \cup I(\sigma)) \implies (G(q, \sigma) \neq \emptyset)$”. Let us then prove by induction on $\sigma$ that $P(\sigma)$ holds for every $\sigma \in \Sigma^*$.

—Induction basis: let us consider $q \in S(e) \cup I(e)$. Since $I(e) = \emptyset$, this means that $q \in S(e)$. Then, $e$ satisfies $e \in \Pi_2(e)$. Moreover, since $S(e) \subseteq F_G$, $q$ after $e = q \in F_G$, and for all $t \in \mathbb{R}_{\geq 0}$, $q$ after $(e, t) \in S(e)$, because otherwise there would exist $\sigma_n \in \text{tw}(\Sigma_0) \ast q$ after $(e, t)$ after $\sigma_n \notin F_G$, meaning that $q$ after $(\sigma_n \cup t) \notin F_G$, and thus $q$ would not be in $S(e)$. Thus, $e \in G(q, e)$. This means that $G(q, e) \neq \emptyset$, and thus that $P(e)$ holds.

—Induction step: let us suppose that for $n \in \mathbb{N}$, for all $\sigma \in \Sigma^*$, $|\sigma| \leq n \implies P(\sigma)$. Let us consider $\sigma \in \Sigma^*$ s.t. $|\sigma| = n$, $a \in \Sigma_e$, and $q \in S(a) \cup I(a)$.

If $q \in I(a)$, let us consider $(h, \sigma_0) \in \Sigma_e \times \Sigma^*$ s.t. $\sigma.a = h.\sigma_0$. Then, $q \in I(h.\sigma_0) = \text{Pred}_h(S(\sigma_0) \cup I(\sigma_0))$, and since $|\sigma_0| = |\sigma| = n \leq n$, by induction hypothesis, $G(q$ after $(0, h), \sigma_0) \neq \emptyset$. Let us consider $w \in G(q$ after $(0, h), \sigma_0)$. Then, $(0, h).w$ satisfies $w(\sum_2((0, h).w) = h.\sigma_0$. $q$ after $(0, h).w = q$ after $(0, h)$ after $w \in F_G$, and for all $t \in \mathbb{R}_{\geq 0}$, $q$ after $(0, h).w, t) = q$ after $(0, h)$ after $(w, t) \in S(\Pi_2((0, h).w)^{-1}.\sum_2) = S(\Pi_2((0, h).w)^{-1}.(h.\sigma_0))$. Thus, $(0, h).w \in G(q, h.\sigma_0) = G(q, \sigma.a)$. Thus, $G(q, \sigma.a) \neq \emptyset$.

If $q \in S(a)$, then there are again two cases:

• if $q \in S(\sigma)$, then by induction hypothesis, $G(q, \sigma) \neq \emptyset$. Since $G(q, \sigma) \subseteq G(q, \sigma.a)$, it follows that $G(q, \sigma.a) \neq \emptyset$.

• otherwise, $q \in X \cup Y$, where $X$ and $Y$ are defined in the definition of $S(\sigma.a)$.

If $q \in X$, then there exists $i \in I(a)$ and $t \in \mathbb{R}_{\geq 0}$ s.t. $q$ after $(e, t) = i$, and for all $t \leq \delta$, $q$ after $(e, t) \in X \subseteq S(\sigma.a)$. Since $i \in I(a)$, we showed previously that $G(i, \sigma.a) \neq \emptyset$. Let us consider $w \in G(i, \sigma.a)$. Then, $w + \delta$ satisfies $\Pi_2(w + \delta) = \sigma.a$, $q$ after $w + \delta = q$ after $(w, t) \in S(\sigma.a)$, otherwise, $q$ after $(w + \delta, t) = i$ after $(w, t - \delta) \in S(\sigma.a)$. Thus, $w + \delta \in G(q, \sigma.a)$. Thus, $G(q, \sigma.a) \neq \emptyset$.

Otherwise, $q \in Y$, and then $q$ satisfies $w(\sum_2(e) \subseteq \sigma.a$, $q$ after $e \in F_G$, and for all $t \in \mathbb{R}_{\geq 0}$, $q$ after $(e, t) \in \text{up}(q) \subseteq \text{up}(Y) = Y \subseteq S(\sigma.a)$. Thus, $e \in G(q, \sigma.a)$. Thus, $G(q, \sigma.a) \neq \emptyset$.

Thus, for all $q \in S(\sigma.a) \cup I(\sigma.a)$, $G(q, \sigma.a) \neq \emptyset$. Thus, $P(\sigma.a)$ holds. By induction on $\sigma$, $P(\sigma)$ holds for every $\sigma \in \Sigma^*$, meaning that for all $\sigma \in \Sigma^*$, for all $q \in S(\sigma) \cup I(\sigma)$, $G(q, \sigma) \neq \emptyset$.

□

Proposition 7. $E_\varphi$ is sound with respect to $\varphi$ in $\text{Pred}(\varphi)$ as per Definition 12.

Proof. Notation from Definition 16 is to be used in this proof:

\[ \kappa_\varphi(q, w) = \min_{i \in I_\varphi} \left( \max_{w \in \sum^*} (G(q, w) \cup \{i\}) \right) \]
\[ \text{buffer}_e = \Pi_2(\text{nobs}(\sigma_0, t')) \ast e, \]
\[ t_1 = \min \{ t' \in \mathbb{R}_{\geq 0} \mid t' \geq t \land \right. \]
\[ \left. G(\text{Reach}(\sigma, (t', a), t''), \text{buffer}_e) \neq \emptyset \} \cup \{+\infty \} \right), \]
\[ \kappa_\varphi'(\sigma) = \min \{ \text{buffer}_e + \min \{ t, t_1 \} \}, \]
\[ \kappa_\varphi''(\sigma) = \left. \Pi_2(\kappa_\varphi' (\sigma)) \right. \ast \text{buffer}_e, \]
\[ t_2 = \min \{ t'' \in \mathbb{R}_{\geq 0} \mid t'' \geq t \land \right. \]
\[ \left. G(\text{Reach}(\sigma, t'', \text{buffer}_e) \neq \emptyset \} \cup \{+\infty \} \right), \]
\[ \kappa_\varphi''(\sigma) = \min \{ \text{buffer}_e + \min \{ t, t_2 \} \}, \]
\[ \kappa_\varphi''(\sigma) = \left. \Pi_2(\kappa_\varphi'' (\sigma)) \right. \ast \text{buffer}_e. \]
For $\sigma \in \text{tw}(\Sigma)$, and $t \geq \text{time}(\sigma)$, let $P(\sigma, t)$ be the predicate “$(\sigma \in \text{Pre}(\varphi, t) \land (\sigma_0, \sigma, \sigma_0) = \text{store}_0(\sigma, t))$” $\implies (E_\omega(\sigma) \models \varphi \land \text{nobs}(\sigma_0, t) \implies G(\text{Reach}(\sigma_0, t), \Pi_G(\text{nobs}(\sigma_0, t), \sigma_0)))$. Let also $P(\sigma)$ be the predicate: “$t \geq \text{time}(\sigma), P(\sigma, t)$”. Let us show that for all $\sigma \in \text{tw}(\Sigma), P(\sigma)$ holds.

**Induction basis:** For $\sigma = \epsilon$, let us consider $t \in \mathbb{R}_{\geq 0}$.

- Case 1: $t \not\in \text{Pre}(\varphi, t)$. Then, $P(\epsilon, t)$ trivially holds.
- Case 2: $t \in \text{Pre}(\varphi, t)$. Then, there exists $t' \leq t$ s.t. $G(\text{Reach}(\epsilon, t'), \epsilon) \neq \emptyset$, meaning that $G(\text{Reach}(\epsilon, t'), \epsilon) \neq \emptyset$, following the definition of $G(\text{Reach}(\epsilon, t'), \epsilon)$. Thus, in both cases, $P(\epsilon, t)$ holds.

**Induction step:** Suppose that for $\sigma \in \text{tw}(\Sigma), P(\sigma)$ holds. Let us consider $(t', a) \in 
\text{tw}(\Sigma)$, and $t \geq t' = \text{time}(\sigma(t', a))$. Let us also consider $(\sigma_0, \sigma_1, \sigma_0) = \text{store}_0(\sigma(t', a))$ and $(\sigma_1, \sigma_2, \sigma_0) = \text{store}_0(\sigma(t', a), t)$.

- Case 1: $\sigma(t', a) \not\in \text{Pre}(\varphi, t')$. Then, $P(\sigma(t', a), t)$ trivially holds.
- Case 2: $\sigma(t', a) \in \text{Pre}(\varphi, t') \cap \sigma \not\in \text{Pre}(\varphi, t')$. Then, $\sigma(t', a) \not\in \text{Pre}(\varphi, t')$, and following lemma 6, $\sigma = \sigma_0, \sigma_0 = \epsilon$, and $\sigma = \Pi_G(\sigma_0)$. Since $\sigma(\sigma(t', a)) \in \text{Pre}(\varphi, t)$, and $\sigma(t', a) \not\in \text{Pre}(\varphi, t')$, there exists $t'' \in \mathbb{R}_{\geq 0}$ s.t. $t' \leq t'' \leq t$, and $G(\text{Reach}(\sigma(t', a), t''), \sigma(t', a), \Pi_G(\text{nobs}(\sigma(t', a), t'')) \neq \emptyset$. Since $t'' \geq t' = \text{time}(\sigma(t', a))$, then $\text{nobs}(\sigma(t', a), t'') = \text{nobs}(\sigma(t', a))$. This means that $G(\text{Reach}(\sigma(t', a)), \Pi_G(\text{nobs}(\sigma(t', a), t'')), t'')$ holds. Let us consider $(nobs(\sigma(t', a), t''))$. Thus, $P(\sigma(t', a), t)$ holds.

**Otherwise, $a \in \Sigma_0$.** Then, $(\sigma(t', a)) = \sigma_0 = \sigma_0 = \epsilon$, and $\sigma = \Pi_G(\sigma_0)$. This means that $G(\text{Reach}(\sigma(t', a)), \Pi_G(\text{nobs}(\sigma(t', a), t'')), \sigma_0) \neq \emptyset$. Thus, $t_1 \leq t' \leq t$, and $\Pi_G(\text{nobs}(\sigma(t', a), t_1), \Pi_G(\text{nobs}(\sigma(t', a), t_1), \sigma_1) \neq \emptyset$. Moreover, $\Pi_G(\text{nobs}(\sigma(t', a), t), \sigma_0, \Pi_G(\text{nobs}(\sigma(t', a), t), \sigma_0) \neq \emptyset$. This means that $P(\sigma(t', a), t)$ holds.

Thus, if $\sigma(t', a) \in \text{Pre}(\varphi, t') \land \sigma \not\in \text{Pre}(\varphi, t')$, $P(\sigma, t) \implies P(\sigma(t', a), t)$.
t') \cup \sigma_t$, it follows that \(\text{nobs}(\sigma_t, t) - t \in G(\text{Reach}(\sigma_t, t), \Pi_2(\text{nobs}(\sigma_t, t), \sigma_t))\). This means that \(P(\sigma(t', a), t)\) holds.

6. Otherwise, \(a \in \Sigma_a\). Since \(w_b \in G(\text{Reach}(\sigma_b, t'), \Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b))\), \(w_b\) satisfies \(\Pi_2(w_b) \equiv \Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b \equiv \Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b, a, \text{Reach}(\sigma_b, t') \text{ after } w_b \in F_G\), and for all \(t'' \in \mathbb{R}_{\geq 0}\), \(\text{Reach}(\sigma_b, t')\) after \(w_b, t'') \in S(\Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b))\). Since \(\Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b \equiv \Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b, a, S(\Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b, a)) \subseteq S(\Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b, a)). Thus, for all \(t'' \in \mathbb{R}_{\geq 0}\), \(\text{Reach}(\sigma_b, t')\) after \(w_b, t'') \in S(\Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b, a)). This means that \(w_b \in G(\text{Reach}(\sigma_b, t'), \Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b, a)). It follows that \(G(\text{Reach}(\sigma_b, t'), \Pi_2(\text{nobs}(\sigma_b, t'), \sigma_b, a)) \neq \emptyset\) and thus, using the same reasoning as in the case where \(a \in \Sigma_a, t_2 = t', \sigma_a\) and \(\sigma_a\) is s.t. \(\text{Reach}(\sigma_a, t')\) after \(\sigma_a \in F_G\), meaning that \(E_\varphi(\sigma(t', a)) = \varphi\), and \(\text{nobs}(\sigma_a, t) - t \in G(\text{Reach}(\sigma_a, t), \Pi_2(\text{nobs}(\sigma_a, t), \sigma_a))\). Thus, \(P(\sigma(t', a), t)\) holds.

Thus, in all cases, for all \(t \geq t', P(\sigma) \implies P(\sigma(t', a), t)\). This means that \(P(\sigma) \implies \forall t \geq t', P(\sigma(t', a), t)\). Thus, \(P(\sigma) \implies P(\sigma(t', a))\). By induction, for all \(\sigma \in tw(\Sigma), P(\sigma)\) holds. In particular, for all \((\sigma, t) \in \text{Pre}(\varphi), E_\varphi(\sigma) = \varphi\). This means that \(E_\varphi\) is sound in \(\text{Pre}(\varphi)\).

\[\square\]

**Proposition 8.** \(E_\varphi\) is compliant, as per Definition 13.

**Proof.** For \(\sigma \in tw(\Sigma), \) let \(P(\sigma)\) be the predicate: “\(\forall t \geq \text{time}(\sigma), (\sigma_s, \sigma_a, \sigma_c) = \text{store}_\varphi(\sigma, t) \implies \sigma_{c|\Sigma_a} = \sigma_{c|\Sigma} \land \Pi_2(\sigma_{c|\Sigma_a}, \text{nobs}(\sigma_t, t)).\sigma_c = \Pi_2(\sigma_{c|\Sigma_a}) \land \sigma_{c|\Sigma_a} \leq \sigma_a, \sigma_c\)”. Let us prove by induction that for all \(\sigma \in tw(\Sigma), P(\sigma)\) holds.

---

**Induction basis:** for \(\sigma = (\epsilon, \text{store}_\varphi(\epsilon) = (\epsilon, \epsilon, \epsilon), \) and \(\epsilon_{\Sigma_a} = \epsilon_{\Sigma} = \Pi_2(\epsilon) = \epsilon\). Thus, \(P(\epsilon)\) trivially holds.

---

**Induction step:** suppose now that for some \(\sigma \in tw(\Sigma), P(\sigma)\) holds. Let us consider \((t', a)\) s.t. \(\sigma(t', a) \in tw(\Sigma), t \geq \text{time}(\sigma), (\sigma_s, \sigma_a, \sigma_c) = \text{store}_\varphi(\sigma(t', a))\), and \((\sigma_s, \sigma_a, \sigma_c) = \text{store}_\varphi(\sigma(t', a), t)\). Then, by induction hypothesis, \(\sigma_{c|\Sigma_a} = \sigma_{c|\Sigma}, \Pi_2(\sigma_{c|\Sigma_a}, \sigma_b), \sigma_c = \Pi_2(\sigma_{c|\Sigma})\), and \(\sigma_{c|\Sigma_a} \leq \sigma|\Sigma_a\).

\(-a \in \Sigma_a\). By construction, \(\sigma_a\) satisfies \(\Pi_2(\sigma_a) \equiv \Pi_2(\text{nobs}(\sigma_b, t')), \sigma_a \) and \(\sigma_a \neq \epsilon \implies \text{date}(\sigma_a(1)) \geq t'.\)

---

**Projection on \(\Sigma_a\):** Since \(a \in \Sigma_a\), \(\sigma_{c|\Sigma_a} = (\sigma_s(t', a) \cup \text{obs}(\sigma_a, t))|_{\Sigma_a}, \sigma_d \in tw(\Sigma_a)\), thus \(\sigma_{c|\Sigma_a} = \Pi_2(\sigma_{c|\Sigma_a}, \text{nobs}(\sigma_a, t'))|_{\Sigma_a}\).

**Projection on \(\Sigma_a\):** \(\Pi_2(\sigma_{c|\Sigma_a}, \text{nobs}(\sigma_a, t'))\) \(\sigma_c = \Pi_2((\sigma_s(t', a) \cup \text{obs}(\sigma_a, t))|_{\Sigma_a}, \text{nobs}(\sigma_a, t'))\).

**Projection on \(\Sigma_a\):** \(\Pi_2(\sigma_{c|\Sigma_a}, \text{nobs}(\sigma_a, t'))\) \(\sigma_c = \Pi_2((\sigma_s(t', a) \cup \text{obs}(\sigma_a, t))|_{\Sigma_a}, \text{nobs}(\sigma_a, t'))\).

---

This means that if \(a \in \Sigma_a\), \(P(\sigma(t', a))\) holds.

---

**Projection on \(\Sigma_a\):** \(\sigma_{c|\Sigma_a} = (\sigma_s(t', a) \cup \text{obs}(\sigma_a, t))|_{\Sigma_a}. \sigma_d \in tw(\Sigma_a), \sigma_{c|\Sigma_a} = \sigma|_{\Sigma_a} = (\sigma(t', a))|_{\Sigma_a}\).

---

**Projection on \(\Sigma_a\):** \(\Pi_2(\sigma_{c|\Sigma_a}, \text{nobs}(\sigma_a, t'))\) \(\sigma_c = \Pi_2((\sigma_s(t', a) \cup \text{obs}(\sigma_a, t))|_{\Sigma_a}, \text{nobs}(\sigma_a, t'))\).

---

Thus, \(\sigma_{c|\Sigma_a} = \sigma_{c|\Sigma} = (\sigma(t', a))|_{\Sigma_a}\).

---

This means that if \(a \in \Sigma_a\), \(P(\sigma(t', a))\) holds.
Lemma 10. Optimal Enforcement of (Timed) Properties with Uncontrollable Events

Moreover, since \( F(\sigma) := \{ q \in Q \mid (q \not\in S(\sigma)) \implies (\exists \sigma' \in tw(\Sigma)_t (q \not\in F_G) \vee (\exists t > 0, q \not\in S(\sigma) \vee I(\sigma)) \implies (\exists \sigma'' \not\in \sigma, \sigma'' \not\in I(\sigma)) \implies \rho after \sigma' \not\in S(\sigma) \vee I(\sigma)) \} \),

Proof. For \( q \in Q \), let \( P(\sigma, q) \) be the predicate \( "\forall \sigma \in tw(\Sigma)_t \nobs(\sigma) = \emptyset \rightarrow (\exists \sigma' \not\in \sigma, \sigma' \not\in I(\sigma)) \implies \rho after \sigma' \not\in S(\sigma) \vee I(\sigma)" \). Let us show the contrapositive of the proposition, that is that for all \( \sigma \in tw(\Sigma)_t \), and for all \( q \in Q \), \( P(\sigma, q) \) holds for the second condition holds, then \( q after \sigma \not\in S(\sigma) \vee I(\sigma) \), meaning that \( q \in F_G \).

--If \( q \not\in S(\sigma) \), then there exists \( (\exists \sigma' \not\in \sigma, \sigma' \not\in I(\sigma)) \implies \rho after \sigma' \not\in S(\sigma) \vee I(\sigma) \).

--If \( \sigma \not\in S(\sigma) \), there exists \( (\exists \sigma' \not\in \sigma, \sigma' \not\in I(\sigma)) \implies \rho after \sigma' \not\in S(\sigma) \vee I(\sigma) \).

For all \( \sigma \in tw(\Sigma)_t \), let \( X_\sigma \) be the sets \( X \) and \( Y \) in the definition of \( S(\sigma') \), correspond to the sets \( \forall \sigma \in tw(\Sigma)_t \), and \( \forall q \in Q \), \( P(\sigma, q) \) holds for the second condition holds, then \( q after \sigma \not\in S(\sigma) \vee I(\sigma) \), meaning that \( q \in F_G \).
and for all $t < \delta + \delta', x$ after $(\varepsilon, t) \in X$. Moreover, $(X \cup Y) \cap u\text{Pred}(X \cup Y \cup I(\sigma)) = \emptyset$ since $Y = Y_1 \cup Y_2 \subseteq S(\sigma) \cup X_0$, $X \subseteq X_1 \cup X_0$, and $X \cup Y = X_0 \cup S(\sigma)$. This means that $X \cup Y \subseteq S(\sigma)$, and since $X_0 \subseteq X \cup Y$, $X_0 \subseteq S(\sigma)$. Since $q = q$ after $(\varepsilon, 0)$, with $\varepsilon \in tw(\Sigma_0)$ and $t \in \mathbb{R}_{>0}$, $q \in X_0$, and thus $q \in S(\sigma)$. Thus, if $\sigma \neq \varepsilon$ and $q \in Q$, $P(\sigma, q) \implies q \in S(\sigma)$.

Thus, for all $\sigma \in \Sigma^*$, for all $q \in Q$, $P(\sigma, q) \implies q \in S(\sigma)$. Thus, the contrapositive also holds, meaning that for all $\sigma \in \Sigma^*$, for all $q \in Q$, $q \notin S(\sigma) \implies \neg P(\sigma, q)$, that is $q \notin S(\sigma)$ implies $(\exists \sigma_\varepsilon \in tw(\Sigma_0), q \notin F_G \land \forall t > 0, q \text{ after } (\sigma_\varepsilon, t) \notin S(\sigma) \cup I(\sigma) \land \forall \sigma_\varepsilon \neq \varepsilon \implies q \text{ after } \sigma_\varepsilon \notin S(\sigma) \cup I(\sigma))$.

\begin{proposition}
$E_\varepsilon$ is optimal in $Pre(\varphi)$, as per Definition 14.
\end{proposition}

\begin{proof}
Let us consider $E' : tw(\Sigma) \times \mathbb{R}_{>0} \rightarrow tw(\Sigma)$, that is compliant with respect to $\Sigma_0$ and $\Sigma_1$.

Let us also consider $\sigma' \in tw(\Sigma)$, and $(t', a)$ s.t. $\sigma(t', a) \in tw(\Sigma)$. Suppose now that $(\sigma, t') \in Pre(\varphi)$, $E'(\sigma, t') = E_\varepsilon(\sigma, t')$, and that $P(\sigma_\varepsilon, q)$, where $\sigma_\varepsilon \in tw(\Sigma_0), q \notin F_G \land \forall t > 0, q \text{ after } (\sigma_\varepsilon, t) \notin S(\sigma) \cup I(\sigma) \land \forall \sigma_\varepsilon \neq \varepsilon \implies q \text{ after } \sigma_\varepsilon \notin S(\sigma) \cup I(\sigma))$.

Thus, if $\sigma \in \Sigma_0$, this means that $\sigma_\varepsilon \neq t' \in G(Reach(\sigma(t', a)), \Pi_\varepsilon(\text{nob}(\sigma(t'), \sigma_\varepsilon)))$. Let us consider $q = Reach(\sigma(t', a))$, and $buff = \Pi_\varepsilon(\text{nob}(\sigma(t'), \sigma_\varepsilon))$. Then, $\sigma_\varepsilon = q = \min_{\sigma, t'}(\max_{\sigma, t'}(G(q, buff)))$. $E'$ is compliant with respect to $\Sigma_0$ and $\Sigma_1$, thus, since $E_\varepsilon(\sigma, t') = E'(\sigma, t')$, there exists $\sigma_\varepsilon \in tw(\Sigma)$ s.t. $E'(\sigma(t', a)) = \sigma_\varepsilon(t', a), \sigma_\varepsilon$. Since $E_\varepsilon(\sigma(t', a)) \neq \sigma_\varepsilon(t', a), \sigma_\varepsilon$, then $\sigma_\varepsilon \neq \sigma_\varepsilon$, thus $\sigma_\varepsilon \neq \sigma_\varepsilon$, $\sigma_\varepsilon \notin S(\sigma) \cup I(\sigma)$.

Thus, since it is not in $S$ and $\Sigma_0$, we can find again a word in $tw(\Sigma_0)$ s.t. the output of $E'$ will never be in $S$ nor $I$, and end up outside of $F_G$. Whatever controllable events $E'$ will output, its output will never reach $S$ nor $I$, and since $E'$ can only output a limited number of controllable events (no more than $|buff|$, at some point it will not be able to output controllable events anymore, and then there will be an uncontrollable word leading its output outside of $F_G$). Concatenating all the uncontrollable words obtained from lemma 10, there would be $\sigma_\varepsilon \in tw(\Sigma_0)$ s.t. $E'(\sigma, t') \neq \varepsilon$.

Thus, if $\sigma \in \Sigma_0$, there exists $\sigma_\varepsilon \in tw(\Sigma_0)$ such that $E'(\sigma(t', a), \sigma_\varepsilon) \neq \varepsilon$.

This means that whenever $E'(\sigma) = E_\varepsilon(\sigma) \land E_\varepsilon(\sigma(t', a)) \neq \sigma_\varepsilon(t', a)$, then there exists $\sigma_\varepsilon \in tw(\Sigma_0)$ s.t. $E'(\sigma(t', a), \sigma_\varepsilon) \neq \varepsilon$. Thus, $E_\varepsilon$ is optimal.
**Proposition 10.** The output of \( \mathcal{E} \) for input \( \sigma \) is \( E_\sigma(\sigma) \).

**Proof.** In this proof, we use some notation from Section 4.2:

\[ C^E = tw(\Sigma) \times Q \times R_{\geq 0} \times \{ \top, \bot \} \] is the set of configurations,

\[ c^E_0 = (c, e, q_0, 0, \bot) \in C^E \] is the initial configuration,

\[ \Gamma^E = ((R_{\geq 0} \times \Sigma) \cup \{ e \}) \times Op \times ((R_{\geq 0} \times \Sigma) \cup \{ e \}) \] is the alphabet, composed of an optional input, an operation and an optional output,

The set of operations, to be applied in the given order, is:

\{ compute, dump, pass-uncont, store-cont, delay \}.

Let us also introduce some specific notation. For a sequence of rules \( w \in \Gamma^E \), we note \( input(w) = \Pi_1(w(1)) \Pi_1(w(2)) \ldots \Pi_1(w(|w|)) \) the concatenation of all inputs from \( w \). In the same way, we define \( output(w) = \Pi_2(w(1)) \Pi_2(w(2)) \ldots \Pi_2(w(|w|)) \) the concatenation of all outputs from \( w \). Since all configurations are not reachable from \( c^E_0 \), for a word \( w \in \Gamma^E \), we will say that \( Reach(w) = c \) if \( c^E_0 \Rightarrow c \), or \( Reach(w) = \bot \) if such a \( c \) does not exist. Let us also define function \( Rules \) which, given a timed word and a date, returns the longest sequence of rules that can be applied with the given word as input at the given date:

\[
\begin{align*}
\text{Rules} : & \{ tw(\Sigma) \times R_{\geq 0} \rightarrow \Gamma^E \\
& (\sigma, t) \mapsto \max_\prec \{(w \in \Gamma^E \mid input(w) = \sigma \land Reach(w) \neq \bot \land \Pi_4(c) = t)\} \\
\end{align*}
\]

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 Rules. We then extend output to timed words with a date: for \( \sigma \in tw(\Sigma) \), and a date \( t \), \( output(\sigma, t) = output(Rules(\sigma, t)) \). For \( \sigma \in tw(\Sigma) \) and \( t \in R_{\geq 0} \), let \( P(\sigma, t) \) be the predicate: "\( E_\sigma(\sigma, t) = output(\sigma, t) \land \{(\sigma, c_b, \sigma_e) = store_e(\text{obs}(\sigma, t), t) \land \langle \sigma_e^F, \sigma^e_q, q^e, t, b \rangle = \text{Reach}(Rules(\sigma, t)) \} \Rightarrow \sigma^e_q = \text{obs}(\sigma_e, t) \land \sigma^e_q = \sigma_e \land q^e = \text{Reach}(\sigma, t) \land (b = \top \Rightarrow G(q^e, \sigma^F) \neq \emptyset)". Let \( L(\sigma) \) be the predicate "\( \forall t \in R_{\geq 0}, P(\sigma, t) \) holds". Let us then prove that for all \( \sigma \in tw(\Sigma), P(\sigma) \) holds.

**Induction basis:** For \( \sigma = e \), let us consider \( t \in R_{\geq 0} \). Then, \( store_e(e, t) = (e, e, e) \), and \( Reach(e, t) = (l_0, v_0 + t) \). On the other hand, the only rules that can be applied are delay, and possibly compute, since there is no input, nor any element to dump. Thus, Rules(e, t) = e/\( \text{delay}(t)/e \), or there exists \( t' \geq t \) s.t. Rules(e, t) = e/\( \text{delay}(t')/e \). \( e/\text{compute}(t)/e \). If rule compute appears in Rules(e, t), then \( b = \top \), meaning that \( G(q_0) = 0, e, t' \neq 0, \) and thus that \( G(q_0) = 0, e, t \neq 0 \) since \( t \geq t' \). Otherwise \( b = \bot \). All the other values remain unchanged between the two cases. In both cases, output(Rules(e, t)) = e = \( E_e(e, e) \). Thus, \( P(e) \) holds.

**Induction step:** Let us suppose now that for some \( \sigma \in tw(\Sigma), P(\sigma) \) holds. Let us consider \( (t', a) \in R_{\geq 0} \times \Sigma \) s.t. \( \sigma(t', a) \in tw(\Sigma) \). Let us then prove that \( P(\sigma(t', a)) \) holds. Let us consider \( t \in R_{\geq 0} \).

\[ e = \langle \sigma^F_e, \sigma^F, q^e, t', b \rangle = \text{Reach}(Rules(\sigma, t'), \langle \sigma_s, \sigma_b, \sigma_e \rangle = \text{store}_e(\text{obs}(\sigma, t'), t), t) \].

If \( t < t' \), then \( obs(\sigma, t', a, t) = obs(\sigma, t), and P(\sigma, (t', a), t) \) trivially holds, since \( P(\sigma) \) holds. Thus, in the following, we consider that \( t \geq t' \), so that \( store_e(\text{obs}(\sigma, t', a), t, t) = store_e(\sigma, t', a), t) \):

If \( a \in \Sigma_a \), rule pass-uncont can be applied. Let us consider \( c' = c \) after \((t', a)/\text{pass-uncont}((t', a))=(t', a), a, \). Then, \( c' = (e, \Pi_2(\sigma^F_i), \sigma^F_q, q^F, t', \bot) \), with \( q^F = q^F_0 \) after \((0, a) \). Then, if \( t \geq t'_F \), where \( t'_F = \min(t' + t' = t' \land G(q^F_0) = 0) \), then rule delay(\( t'_F - t' \)) can be applied, followed by rule compute. Since \( q^F = \text{Reach}(\sigma, t'), \sigma^F_q = \text{obs}(\sigma_b, t'), and \sigma^F_e = \sigma_e, \)} by in-
duction hypothesis), then \( G(q' \text{ after}(e, t'' - t')) = G(\text{Reach}(\sigma_e, (t', a), t')) \), it follows that \( t_b' = t_2 \) as defined in Definition 16. If \( t_b' > t_2 \), then rule \( \text{delay}(t_2 - t') \) can be applied, followed by rule compute. Then, \( q' \text{ after}(e, t'' - t') = \Phi' \text{ compute}(t_2 - t') \), \( \text{Reach}(\sigma_e, (t', a), t') \). Otherwise, \( t_2' = t_2 \). Then, \( \text{output}(\sigma, (t', a), t') = \Phi' \text{ output}(\sigma, (t', a), t) \). Moreover, \( \Phi = \text{Reach}(\sigma, (t', a), t) \).

Thus, by induction, for all \( \sigma \in tw(\Sigma) \), \( \Phi = \text{ Reach}(\sigma, (t', a), t) \) holds. In particular, for all \( \sigma \in tw(\Sigma) \), and for all \( t \in \mathbb{R}_{\geq 0} \), \( \text{output}(\sigma, t) = \Phi = \text{ Reach}(\sigma, (t', a), t) \), meaning that the output of the enforcement monitor \( \Phi' \) with input \( \sigma \) at time \( t \) is exactly the output of function \( E_{\Phi} \) with the same input and the same date.