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

To cite this version:
Matthieu Renard, Yliès Falcone, Antoine Rollet, Thierry Jéron, Hervé Marchand. Optimal Enforcement of (Timed) Properties with Uncontrollable Events. Mathematical Structures in Computer Science, Cambridge University Press (CUP), 2017, <10.1017/S0960129517000123>. <hal-01262444v4>

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\textsuperscript{1}, YLIÈS FALCONE\textsuperscript{2}, ANTOINE ROLLET\textsuperscript{1},

THIERRY JÉRON\textsuperscript{3}, and HERVÉ MARCHAND\textsuperscript{3}

\textsuperscript{1} LaBRI, Bordeaux INP, Université Bordeaux, Bordeaux, France.
\textsuperscript{2} Univ. Grenoble-Alpes, Inria, Laboratoire d’Informatique de Grenoble, F-38000 Grenoble, France.
\textsuperscript{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\(^\dagger\). 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”\(^\ddagger\) 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 spoiler 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

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

\(^\ddagger\) 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 $Σ$ is a sequence over $Σ$. The set of finite words over $Σ$ is denoted $Σ^*$. The *length* of a finite word $w$ is noted $|w|$, and the empty word is noted $ε$. $Σ^*$ stands for $Σ^* \setminus \{ε\}$. A *language* over $Σ$ is any subset $L \subseteq Σ^*$. 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 $w'.w'' = w'.w^{-1}.w = w$. These definitions are extended to languages in the natural way. A language $L \subseteq Σ^*$ is *extension-closed* if for any words $w \in L$ and $w' \in Σ^*$, $w.w' \in L$. 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 $Π_i(e)$, the projection on the $i$-th coordinate, i.e. $Π_i(e) = e_i$. 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 $w \in Σ^*$ and $Σ' \subseteq Σ$, we define the *restriction* of $w$ to $Σ'$, noted $w|_{Σ'}$, as the word $w' \in Σ'^*$ whose letters are the letters of $w$ belonging to $Σ'$ in the same order. Formally, $Π_i(\epsilon) = \epsilon$ and $∀σ \in Σ^*, ∀a \in Σ$, $(w.a)|_{Σ'} = w|_{Σ'}$, $a$ if $a \in Σ'$, and $(w.a)|_{Σ'} = w|_{Σ'}$ otherwise. We also note $=_{Σ'}$ the equality of the restrictions of two words to $Σ'$; if $σ$ and $σ'$ are two words, $σ =_{Σ'} σ'$ if $σ|_{Σ'} = σ'|_{Σ'}$. We define in the same way $\preceq_{Σ'}$, $\preceq_{Σ'}$ if $σ|_{Σ'} \preceq σ'|_{Σ'}$. 


Automata. An automaton is a tuple $\langle Q, q_0, \Sigma, \rightarrow, F \rangle$, 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 $q'' \xrightarrow{a} q'$. Moreover, for any $q \in Q$, $q \xrightarrow{w}$ $q$ always holds. An automaton $A = \langle Q, q_0, \Sigma, \rightarrow, F \rangle$ 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 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 events on dates and actions respectively. A timed word over $\Sigma$ is a word over $\mathbb{R}_{\geq 0} \times \Sigma$ whose real parts are ascending, i.e. $\sigma$ is a timed word if $\sigma \in (\mathbb{R}_{\geq 0} \times \Sigma)^*$ and $\forall i \in [1; |\sigma| - 1], \text{date}(w(i)) \leq \text{date}(w(i + 1))$. $\text{tw}(\Sigma)$ 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 $\sigma \neq \epsilon$, and $\text{time}(\epsilon) = 0$. We define the observation of $\sigma$ at time $t$ as the timed word $\text{obs}(\sigma, t) = \max_{\epsilon \leq t} \{\{\sigma' \mid \sigma' \subseteq \sigma \land \text{time}(\sigma') \leq t\}\}$. 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. $\epsilon_{|\Sigma'} = \epsilon$, and for $\sigma \in \text{tw}(\Sigma)$ and $(t, a) \in \sigma$, $\text{time}(t, a) \in \text{tw}(\Sigma)$, and $(\sigma, t, a)_{|\Sigma'} = \sigma_{|\Sigma'}$ if $a \in \Sigma'$, and $(\sigma, t, a)_{|\Sigma'} = \sigma_{|\Sigma'}$ otherwise. The notations $|_{\Sigma'}$ 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 $\sigma' \preceq_{\text{d}} \sigma$, whenever $\Pi_{\Sigma}(\sigma') \preceq_{\Sigma} \Pi_{\Sigma}(\sigma)$ 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 \( = \) and the prefix order \( \preceq \), we note \( \sigma' \preceq_{\Pi_{\Sigma}} \sigma \) whenever \( \sigma'_{|_{\Sigma'}} \preceq_d \sigma_{|_{\Sigma'}} \). We also define a lexical 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 \( t' < t \) \( \lor \) \( t = t' \) \& \( \sigma' \leq_{\text{lex}} \sigma \).

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 \((1, a)(2, b)(4, c) \preceq_d \sigma \) and \( \sigma \leq_{\text{lex}} (1, a)(3, b)(3, c)(3, a) \). 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 \( \mathcal{V}(X) \), i.e., \( \mathcal{V}(X) = \{\nu \mid \nu : X \rightarrow \mathbb{R}_{\geq 0}\} \). 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 \( X' \). \( \mathcal{G}(X) \) denotes the set of guards consisting of boolean combinations of constraints of the form \( X_i \not\in c \) with \( X_i \in X, c \in \mathbb{N} \), and \( \not\in \in \{<, \leq, =, \geq, >\} \). Given \( g \in \mathcal{G}(X) \) and a valuation \( \nu \), we write \( \nu \models g \) when for every constraint \( X_i \not\in c \) in \( g, \nu(X_i) \models c \) holds.

**Definition 1 (Timed automaton (Alur and Dill, 1992)).** A timed automaton (TA) is a tuple \( A = (L, l_0, X, \Sigma, \Delta, G) \), 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, \( \Delta \subseteq L \times \mathcal{G}(X) \times \Sigma \times 2^X \times L \) is the transition relation, and \( G \subseteq L \) is a set of accepting locations. A transition \((l, g, a, X', l') \in \Delta \) is a transition from \( l \) to \( l' \), labelled with event \( a \), with guard \( g \), and with the clocks in \( X' \) to be reset.

The semantics of a timed automaton \( A \) is a timed transition system \([A] = \langle Q, q_0, \Gamma, \rightarrow, F_G \rangle \) where \( Q = L \times \mathcal{V}(X) \) is the (infinite) set of states, \( q_0 = (l_0, \nu_0) \) is the initial state, with \( \nu_0 = \nu[X \leftarrow 0] \), \( F_G = G \times \mathcal{V}(X) \) is the set of accepting states, \( \Gamma = \mathbb{R}_{>0} \times \Sigma \) 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 \( A = (L, l_0, X, \Sigma, \Delta, G) \) is deterministic if for any different transitions \((l, g, a, Y_1, l'_1) \) and \((l, g_2, a, Y_2, l'_2) \) in \( \Delta \), \( g_1 \land g_2 \) is unsatisfiable, meaning that only one transition can be fired at any time. \( A \) 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 \([A] \) 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 \( \text{Rtm}(A) \) and \( \text{Run}_{F_G}(A) \) denotes the subset of runs accepted by \( A \), 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
\( \sigma_1 \) and \( \sigma_2 \), it is needed to delay \( \sigma_2 \) to obtain a trace: the concatenation \( \sigma \) of \( \sigma_1 \) and \( \sigma_2 \) is the trace defined as \( \sigma = \sigma_1 . (\sigma_2 + t \cdot \text{time}(\sigma_1)) \). Thus, if \( q \xrightarrow{\sigma_1} q' \xrightarrow{\sigma_2} q'' \), then \( q \xrightarrow{\sigma} q'' \).

Timed Properties. A regular timed property is a timed language \( \varphi \subseteq \text{tw}(\Sigma) \) 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.

Given a complete and deterministic automaton \( A \) s.t. \( Q \) is the set of states of \( [A] \) and \( \rightarrow \) its transition relation, and a word \( \sigma \), for \( q \in Q \), we note \( q \text{ after } \sigma = q' \), where \( q' \) is s.t. \( q \xrightarrow{\tau} q' \). Since \( A \) is complete and deterministic, there exists only one such \( q' \). We note \( \text{Reach}(\sigma) = q_0 \text{ after } \sigma \).

We extend these definitions to languages: if \( L \) is a language, \( q \text{ after } L = \bigcup_{q \in L} q \text{ after } \sigma \) and \( \text{Reach}(L) = q_0 \text{ after } L \). These definitions are valid both in the untimed and timed cases. For the timed case, we also allow to add an extra parameter to \( \text{after} \) and \( \text{Reach} \), that represents an observation time. For \( q \in Q \), \( t \in \mathbb{R}_{\geq 0} \), and \( \sigma \in \text{tw}(\Sigma) \), \( q \text{ after } (\sigma,t) = (l(\sigma,t), \nu + t \cdot \text{time}(\sigma,t)) \), where \( (l, \nu) = q \text{ after } (\text{obs}(\sigma,t)) \), and \( \text{Reach}(\sigma,t) = q_0 \text{ after } (\sigma,t) \). 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 \( (l, \nu) \in Q \), \( (l, \nu) \text{ after } (\varepsilon, t) = (l, \nu + t) \) is the state reached from \( (l, \nu) \) by letting time elapse of \( t \) units. Moreover, for \( (l, \nu) \in Q \), we note \( \text{up}(l(\nu)) = \{ (l, \nu + t) \mid t \in \mathbb{R}_{\geq 0} \} \). This definition is extended to sets of states: for \( S \subseteq Q \), \( \text{up}(S) = \bigcup_{q \in S} \text{up}(q) \). We also define a predecessor operator: for \( q \in Q \) and \( a \in \Sigma \), \( \text{Pred}_a(q) = \{ q' \in Q \mid q' \text{ after } a = q \} \) for the untimed setting, and \( \text{Pred}_a(q) = \{ q' \in Q \mid q' \text{ after } (0,a) = q \} \) for the timed setting. This definition is extended to words: if \( \sigma \in \Sigma^* \) (or \( \sigma \in \text{tw}(\Sigma) \)), then \( \text{Pred}_a(\sigma) = \{ q' \in Q \mid q' \text{ after } \sigma = q \} \).

**Example 1** (Shared Data Storage). Consider the property \( \varphi_1 \) described in Fig. 3 and representing writes on a shared storage device. 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 \( \text{LockOff} \) event. Thus, the set of locations of \( \varphi_1 \) is \( L = \{ l_0, l_1, l_2, l_3 \} \), the initial location is \( l_0 \), 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 \( G = \{ l_1, l_2 \} \), and the set of transitions contains for instance transitions \( (l_0, T, \text{Auth}, \emptyset, l_1) \), \( (l_2, T, \text{Auth}, \emptyset, l_2) \), and \( (l_3, T, \text{LockOn}, \emptyset, l_3) \), where \( T \) 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 \( \varphi_1 \), where the clock valuations are
replaced by the value of the unique clock $x$. Then, $\text{Reach}((2, \text{Auth})) = (t_0, 0) \text{after} (2, \text{Auth}) = (t_1, 2)$, and, for example, $((l_2, 3) \text{ after} ((2, \text{LockOff}), 4) = (l_1, 2)$, 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). 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 $S^c = Q \setminus S$.

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 $\preceq$: $\forall (\sigma, \sigma') \in (\Sigma^*)^2, \sigma \preceq \sigma' \implies E(\sigma) \preceq E(\sigma')$.

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 $\forall \sigma \in \Sigma^*, E(\sigma) \preceq_{\Sigma_c} E(\sigma) \land \forall u \in \Sigma_u, E(\sigma).u \preceq E(\sigma.u)$.

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-
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 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

\footnote{Uncontrollable events are emphasised in italics.}
it will be in $S(\sigma.a)$ or in $I(\sigma.a)$. 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. $I(a.\sigma)$ is defined as the set of all states from which following the transition labelled by $a$ 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 $q \in Q$, $\sigma \in \Sigma^*_c$, $G(q, \sigma) = \{ w \in \Sigma^*_c \mid w \preceq \sigma \land q \text{ after } w \in S(w^{-1}, \sigma) \}$.

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$_\varphi$, $E_\varphi$).** $\parallel$ Function store$_\varphi : \Sigma^* \rightarrow \Sigma^* \times \Sigma^*$ is defined as:

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

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

where:

$$\kappa(\varphi, q, w) = \max_{\varphi}(G(q, w) \cup \{ \epsilon \}),$$

$$\sigma'_s = \kappa(\varphi(\text{Reach}(\sigma_s.a), \sigma_e), \sigma'_e = \sigma^{-1}_s, \sigma_e, \sigma''_e = \kappa(\varphi(\text{Reach}(\sigma_s, \sigma_e.a)), \sigma''_e = \sigma^{-1}_s. \sigma_e.a).$$

The enforcement function $E_\varphi : \Sigma^* \rightarrow \Sigma^*$ is defined as $E_\varphi(\sigma) = \Pi_1(\text{store}_\varphi(\sigma))$, for any $\sigma \in \Sigma^*$.

$\sigma \xrightarrow{E} \sigma_s$

\begin{figure}[h]
\centering
\begin{tikzpicture}
\node (s) {$\sigma$};
\node (e) [right of=s, yshift=-1.5cm] {$\sigma_c$};
\node (es) [right of=e, yshift=-1.5cm] {$\sigma_s$};
\draw (s) -- node[anchor=west] {$E$} (es);
\end{tikzpicture}
\caption{Enforcement function}
\end{figure}

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 $\Sigma_u$, then it is output, as required by compliance. Then, the longest prefix of $\sigma_s$ 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 $\sigma_s$, 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

\[\parallel E_\varphi \text{ and } \text{store}_\varphi \text{ depend on } \Sigma_u \text{ and } \Sigma_c, \text{ but we did not write it in order to lighten the notations.} \]
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:

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

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 emitting only the uncontrollable events of $\sigma$, and the corresponding buffer would then be the controllable events of $\sigma$.

**Example 3.** Considering property $\varphi_{ex}$ (Fig. 2), with the uncontrollable alphabet $\Sigma_u = \{ \text{Auth}, \text{LockOff}, \text{LockOn} \}$, $\text{Pre}(\varphi_{ex}) = \text{Write}^*.\text{Auth}.$ $\Sigma^*$. 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 $S(\epsilon)$. In order to reach a state in $S$ (i.e. $q_1$ or $q_2$), it is necessary to read Auth. Once 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 Write events when in $q_2$ until a LockOff event is received. Consequently, $q_1$ and $q_2$ are in $S(\sigma)$ for all $\sigma \in \Sigma^*$, and thus $\text{Pre}(\varphi_{ex}) = \text{Write}^*.\text{Auth}.\Sigma^*$, since Write events can be buffered while in state $q_0$ until event 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 $\Sigma^*$, $E_\varphi(\sigma) \succeq E_\varphi(\sigma').$ Following the definition of store$_{\varphi}$, this holds provided that $\sigma' \in \Sigma$ (i.e. $\sigma'$ is a word of size 1). Since $\succeq$ 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:

- $\sigma.a \notin \text{Pre}(\varphi)$. Then the proposition holds.
- $\sigma.a \in \text{Pre}(\varphi)$, but $\sigma \notin \text{Pre}(\varphi)$. 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 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 $\text{store}_e(\sigma.a)$, $(\sigma.a)|_{\Sigma_c}$, and $(\sigma.a)|_{\Sigma_u}$ 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 $\text{store}_e$, $\kappa_\varphi$ 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 $\varphi_{\text{ex}}$ (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 \((\sigma_s, \sigma_c) = \text{store}_{\psi, \sigma}(\sigma)\), with input \text{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>\text{Auth}</td>
<td>\text{Auth}</td>
<td>(\epsilon)</td>
</tr>
<tr>
<td>\text{Auth.LockOn}</td>
<td>\text{Auth.LockOn}</td>
<td>(\epsilon)</td>
</tr>
<tr>
<td>\text{Auth.LockOn.Write}</td>
<td>\text{Auth.LockOn}</td>
<td>\text{Write}</td>
</tr>
<tr>
<td>\text{Auth.LockOn.Write.LockOff}</td>
<td>\text{Auth.LockOn.LockOff.Write}</td>
<td>(\epsilon)</td>
</tr>
</tbody>
</table>

An enforcement monitor \(E\) for \(\varphi\) is a transition system \((C^E, c^E_0, \Gamma^E, \rightarrow_E)\) such that:

- \(C^E = Q \times \Sigma^*\) is the set of configurations.
- \(c^E_0 = (q_0, \epsilon)\) is the initial configuration.
- \(\Gamma^E = \Sigma^* \times \{\text{dump}(.), \text{pass-uncont}(.), \text{store-cont}(.), \text{pass-cont}(.)\} \times \Sigma^*\) 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 \(w/\triangleright a \triangleright w'\) stands for \((w,\triangleright a, w') \in \Gamma^E\)):

- **Dump:** \(\langle q, a.\sigma_c \rangle \xrightarrow{c/\text{dump}(a)/\alpha} \langle q', \sigma_c \rangle\), if \(a \in \Sigma_c, G(q, a.\sigma_c) \neq \emptyset\) and \(G(q, a.\sigma_c) \neq \emptyset\), with \(q' = q\) after \(a\).
- **Pass-uncont:** \(\langle q, \sigma_c \rangle \xrightarrow{\alpha/\text{pass-uncont}(a)/\alpha} \langle q', \sigma_c \rangle\), with \(a \in \Sigma_a\) and \(q' = q\) after \(a\).
- **Store-cont:** \(\langle q, \sigma_c \rangle \xrightarrow{\alpha/\text{store-cont}(a)/\alpha} \langle q, \sigma_c, a \rangle\), 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 \(\text{store}_{\psi, \sigma}\), 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 \(\sigma\) is \(E_{\varphi}(\sigma)\).

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 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 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 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, can not be used in this situation. Moreover, when delaying some events to have the property satisfied in the future, one must consider the fact that some uncontrollable events could occur at any moment (and cannot be delayed). Finally, some properties 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 of uncontrollable events, meaning that for these properties it is impossible to obtain sound EMs, as shown in Example 5.

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 is a function \( E : \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0} \to \text{tw}(\Sigma) \) 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 \( S \subset tw(\Sigma) \times \mathbb{R}_{\geq 0} \) if for any \((\sigma, t)\) 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 \( tw(\Sigma) \), 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 for all \( \sigma \in tw(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, E(\sigma, t) \preceq_{\Sigma_u} \sigma \land E(\sigma, t) = \text{obs}(\sigma, t) \land \forall u \in \Sigma_u, E(\sigma, t, \langle t, u \rangle) \prec E(\sigma, \langle t, u \rangle, t) \).

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 \( E : tw(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow tw(\Sigma) \), and a timed word \( \sigma \in tw(\Sigma) \), 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 \( E : tw(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow tw(\Sigma) \) that is compliant with respect to \( \Sigma_c \) and \( \Sigma_u \) and sound in a time-extension-closed set \( S \subseteq tw(\Sigma) \times \mathbb{R}_{\geq 0} \) is optimal in \( S \) if for any enforcement function \( E' : tw(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow tw(\Sigma) \), for all \( \sigma \in tw(\Sigma) \), for all \( \langle t, a \rangle \) s.t. \( \sigma, \langle t, a \rangle \in tw(\Sigma) \),

\[
\text{compliant}(E', \Sigma_u, \Sigma_c) \land (\sigma, t) \in S \land E'(\sigma, t) = E(\sigma, t) \land E'(\sigma, \langle t, a \rangle) \preceq d E'(\sigma, \langle t, a \rangle) \\
\quad \quad \quad \implies \exists \sigma_u \in tw(\Sigma_u), E'(\sigma, \langle t, a \rangle, \sigma_u) \not\models \varphi.
\]

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 $A_\varphi$, $S(\sigma)$ 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 $[A_\varphi]$, $I(\sigma)$ and $S(\sigma)$, are inductively defined over sequences of controllable events as follows: $I(\epsilon) = \emptyset$ and $S(\epsilon) = \{ q \in F_G \mid q \text{ after } tw(\Sigma_\emptyset) \subseteq F_G \}$ and, for $\sigma \in \Sigma^*_c$ and $a \in \Sigma_c$,

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

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, $S(\epsilon) = \{ q \in F_G \mid q \text{ after } tw(\Sigma_\emptyset) \subseteq F_G \}$. 
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 \( Y \not\subseteq F_G \), 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 \((X \cup Y) \cap \nupred(X \cup Y \cup I(\sigma, a)) = \emptyset \). 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 \( Y = \up(\sigma) \). 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 \( G : Q \times \Sigma_c \rightarrow 2^{tw(\Sigma)} \) 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 set corresponding to what remains from \( \sigma \)):

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

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

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

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

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

\[
\text{store}_{\varphi}(\sigma, (t', a), t) = \begin{cases} 
(\sigma_a, (t', a), \text{obs}(\sigma_b, t), \sigma_c', \sigma_c) & \text{if } a \in \Sigma_u \\
(\sigma_a, \text{obs}(\sigma_b, t), \sigma_c', \sigma_c) & \text{if } a \in \Sigma_c
\end{cases}
\]

with:

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

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

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

\[
\sigma_b' = \kappa_{\varphi}(\text{Reach}(\sigma_a, (t', a), \min\{\{t, t_1\}\}), \text{buffer}_c) + t \min\{\{t, t_1\}\},
\]

\[
\sigma_c' = \Pi_{\Sigma}(\sigma_c')^{-1}. \text{buffer}_c,
\]
\[ t_2 = \min(\{t'' \in \mathbb{R} \geq 0 \mid t'' \geq t' \land \tau(\text{Reach}(s, t''), \text{buffer}_c.a) \neq \emptyset \} \cup \{+\infty\}), \]
\[ \sigma_b'' = \kappa(\text{Reach}(s, \min(\{t, t_2\})), \text{buffer}_c.a) + \min(\{t, t_2\}), \]
\[ \sigma_c'' = \text{min}(\text{buffer}_c.a). \]

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

The function store 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 \( \text{transitions} \). 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 \( \text{transitions} \).

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 \( \sigma_s \) 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 \( S(\sigma_b) \) 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 \((\sigma_s, \sigma_b, \sigma_c) = \text{store}(\sigma, t) \), then \( \sigma \) is in \( \text{Pre}(\varphi, t) \) if and only if the set \( G(\text{Reach}(\sigma_s, t), \Pi(\text{obs}(\sigma_b, t)).\sigma_c) \) 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 (Pre(\varphi)). \( \text{Pre}(\varphi) = \{(\sigma, t) \mid \sigma \in \text{Pre}(\varphi, t)\} \), where, for \( \sigma \in \text{tw}(\Sigma) \) and \( t \in \mathbb{R} \geq 0 \),

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

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 \( G(\text{Reach}(\text{obs}(\sigma, t)|_{\Sigma_1}, t), \Pi_{\Sigma_2}(\text{obs}(\sigma, t)|_{\Sigma_2})) \) 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 \( t' \geq t \), \( E_{\varphi}(\sigma, t) \preccurlyeq E_{\varphi}(\sigma, t') \), and for all \( (t, a) \) s.t. \( \sigma(t, a) \in \text{tw}(\Sigma) \), \( E_{\varphi}(\sigma, t) \preccurlyeq E_{\varphi}(\sigma(t, a), t) \). To prove this, we first show by induction that \( E_{\varphi}(\sigma, t) \preccurlyeq E_{\varphi}(\sigma, t') \). 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 \( E_{\varphi}(\sigma(t'', a), t) \preccurlyeq E_{\varphi}(\sigma(t', a), t') \), or \( t \geq t_1 \), and then there exists \( \sigma_b \) s.t. \( E_{\varphi}(\sigma(t'', a), t) = \sigma_b \cdot \text{obs}(\sigma_b, t) \) and \( E_{\varphi}(\sigma(t', a), t') = \sigma_b \cdot \text{obs}(\sigma_b, t') \), meaning that \( E_{\varphi}(\sigma(t', a), t) \preccurlyeq E_{\varphi}(\sigma(t', a), t') \).

- \( t'' > t' \). 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 \( \text{store}_{\varphi} \), \( E_{\varphi}(\sigma, t', a) \preccurlyeq E_{\varphi}(\sigma(t', a), t') \).

Thus, \( E_{\varphi}(\sigma, t) \preccurlyeq E_{\varphi}(\sigma, t') \). Then, what remains to show is that if \( \sigma(t, a) \in \text{tw}(\Sigma) \), then \( E_{\varphi}(\sigma, t) \preccurlyeq E_{\varphi}(\sigma(t, a), t) \).

**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 \( \preccurlyeq_a \)) that satisfies \( \varphi \) and
is safe to output. It is maximal because in the definition of \( \text{store}_e \), \( \kappa_e \) 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, \( E_\varphi \) 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 \( E_\varphi \), as defined in Definition 16.

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

- \( C^E = \text{tw}(\Sigma) \times \Sigma_0^* \times Q \times \mathbb{R}_{\geq 0} \times \{\top, \bot\} \) is the set of configurations.
- \( c^E_0 = (\epsilon, \epsilon, q_0, 0, \bot) \in C^E \) is the initial configuration.
- \( \Gamma^E = (\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\} \times \text{Op} \times ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}) \) is the alphabet, composed of an optional input, an operation and an optional output.

The set of operations is \{ compute(\_), dump(\_), pass-uncont(\_), store-cont(\_), delay(\_) \}.

Whenever \( (\sigma, \rho, \sigma') \in \Gamma^E \), it will be noted \( \sigma / \bowtie / \sigma' \).

- \( \rightarrow_E \) is the transition relation defined as the smallest relation obtained by applying the following rules given by their priority order:

  - **Compute:** \( (\epsilon, \sigma_c, q, t, \bot) \xrightarrow{\epsilon/\text{compute}(\_)/\epsilon} (\sigma'_c, \sigma'_c, q, t, \top) \), if \( G(q, \sigma_c) \neq \emptyset \), with \( \sigma'_c = \kappa_c(q, \sigma_c) + t \), and \( \sigma'_e = \Pi(\sigma'_c) \).

  - **Dump:** \( (t, a).\sigma_b, \sigma_c, q, t_b, \top) \xrightarrow{\epsilon/\text{dump}(t, a)/(t, a)} (\sigma_b, \sigma_c, q', t_b, \top) \), with \( q' = q \) after \( (0, a) \).

  - **Pass-uncont:** \( (\epsilon, \sigma_c, q, t, b) \xrightarrow{(t, a)/\text{pass-uncont}(t, a)/(t, a)} (\epsilon, \Pi(\sigma_b).\sigma_c, q', t, \bot) \), with \( q' = q \) after \( (0, a) \).

  - **Store-cont:** \( (\sigma_b, \sigma_c, q, t, b) \xrightarrow{(t, c)/\text{store-cont}(t, c)/(t, c)} (\epsilon, \Pi(\sigma_b).\sigma_c, q, t, \bot) \).

  - **Delay:** \( (\sigma_b, \sigma_c, q, t, b) \xrightarrow{\epsilon/\text{delay}(\delta)/\epsilon} (\sigma_b, \sigma_c, (l, v + \delta), t + \delta, b) \).

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 \( \text{store}_e \), whereas \( \sigma_c \) is the same as in the definition of \( \text{store}_e \). The state \( q \) represents \( \sigma_s \), s.t. \( q = \text{Reach}(\sigma_s, l) \).

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

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: Write synchronisation. Each time the lock is taken and released, we must also wait before issuing a new received, we are able to send the controllable event Write uncontrollable event is sent by the device to authorise writings. Once the lock has been locked by someone else, and that it is unlocked, respectively. The definition of LockOn can get the status of a lock through the uncontrollable events LockOff can be computed. If the input is in Pre(ϕ), then rule compute() can be applied, and then the second member of the configuration will have the same value as the second member of storec, 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(σb, t), where σb was computed by rule compute() as second member. Considering the definition of storeb, it follows that the output of E with input σ at date t is Eϕ(σ, 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: Σe = {Write} and Σu = {Auth, LockOff, LockOn}.

Now, let us follow the output of the storeϕ function over time with the word σ = (1, Auth). (2, LockOn). (4, Write). (5, LockOff). (6, LockOn). (7, Write). (8, LockOff) as input: let (σa, σb, σc) = storeϕ(obs(σ, t), t). Then the values taken by σa, σb and σc over time are given in Table 2. To calculate them, notice that for all valuation ν : {x} → ℝ≥0, (l1, ν) ∈ S(ε), and (l2, ν) ∈ S(ε), since all uncontrollable words from l1 and l2 lead to l1 or l2, 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
### Optimal Enforcement of (Timed) Properties with Uncontrollable Events

<table>
<thead>
<tr>
<th>t</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 0</td>
<td>$\epsilon/(\epsilon, (l_0, 0), 0, \bot)/(1, Auth).(2, On).(4, w).(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ delay(1)</td>
</tr>
<tr>
<td>t = 1</td>
<td>$\epsilon/(\epsilon, (l_0, 1), 1, \bot)/(1, Auth).(2, on).(4, w).(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ pass-uncont((1, Auth))</td>
</tr>
<tr>
<td>t = 1</td>
<td>$(1, Auth)/(\epsilon, (l_1, 1), 1, \bot)/(2, on).(4, w).(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ compute()</td>
</tr>
<tr>
<td>t = 1</td>
<td>$(1, Auth)/(\epsilon, (l_1, 1), 1, \top)/(2, on).(4, w).(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ delay(1)</td>
</tr>
<tr>
<td>t = 2</td>
<td>$(1, Auth).(2, on)/(\epsilon, (l_2, 2), 2, \top)/(4, w).(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ compute()</td>
</tr>
<tr>
<td>t = 2</td>
<td>$(1, Auth).(2, on)/(\epsilon, (l_2, 2), 2, \bot)/(4, w).(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ delay(2)</td>
</tr>
<tr>
<td>t = 4</td>
<td>$(1, Auth).(2, on)/(\epsilon, (l_4, 4), 4, \top)/(4, w).(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ store-cont((4, w))</td>
</tr>
<tr>
<td>t = 4</td>
<td>$(1, Auth).(2, on)/(\epsilon, (l_4, 4), 4, \bot)/(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ compute()</td>
</tr>
<tr>
<td>t = 5</td>
<td>$(1, Auth).(2, on)/(\epsilon, (l_5, 5), 5, \top)/(5, off).(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ pass-uncont((5, off))</td>
</tr>
<tr>
<td>t = 5</td>
<td>$(1, Auth).(2, on).(5, off)/(\epsilon, (7, w), (l_1, 0), 5, \bot)/(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ compute()</td>
</tr>
<tr>
<td>t = 6</td>
<td>$(1, Auth).(2, on).(5, off)/(\epsilon, (7, w), (l_1, 1), 6, \top)/(6, on).(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ pass-uncont((6, on))</td>
</tr>
<tr>
<td>t = 6</td>
<td>$(1, Auth).(2, on).(5, off).(6, on)/(\epsilon, (7, w), (l_2, 1), 6, \bot)/(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ compute()</td>
</tr>
<tr>
<td>t = 7</td>
<td>$(1, Auth).(2, on).(5, off).(6, on)/(\epsilon, (7, w), (l_2, 2), 7, \top)/(7, w).(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ store-cont((7, w))</td>
</tr>
<tr>
<td>t = 7</td>
<td>$(1, Auth).(2, on).(5, off).(6, on)/(\epsilon, (7, w), (l_2, 2), 7, \bot)/(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ compute()</td>
</tr>
<tr>
<td>t = 7</td>
<td>$(1, Auth).(2, on).(5, off).(6, on)/(\epsilon, (7, w), (l_2, 2), 7, \top)/(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ delay(1)</td>
</tr>
<tr>
<td>t = 8</td>
<td>$(1, Auth).(2, on).(5, off).(6, on)/(\epsilon, (7, w), (l_2, 3), 8, \top)/(8, off)$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ pass-uncont((8, off))</td>
</tr>
<tr>
<td>t = 8</td>
<td>$(1, Auth).(2, on).(5, off).(6, on).(8, off)/(\epsilon, (10, w),(10, w), (l_1, 0), 8, \bot)/\epsilon$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ compute()</td>
</tr>
<tr>
<td>t = 8</td>
<td>$(1, Auth).(2, on).(5, off).(6, on).(8, off)/(\epsilon, (10, w), (10, w), (l_1, 0), 8, \top)/\epsilon$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ delay(2)</td>
</tr>
<tr>
<td>t = 10</td>
<td>$(1, Auth).(2, on).(5, off).(6, on).(8, off)/(\epsilon, (10, w), (10, w), (l_1, 2), 10, \top)/\epsilon$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ dump((10, w))</td>
</tr>
<tr>
<td>t = 10</td>
<td>$(1, Auth).(2, on).(5, off).(6, on).(8, off).(10, w)/(\epsilon, (10, w), (l_1, 2), 10, \top)/\epsilon$</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$ dump((10, w))</td>
</tr>
<tr>
<td>t = 10</td>
<td>$(1, Auth).(2, on).(5, off).(6, on).(8, off).(10, w).(10, w)/(\epsilon, (l_1, 2), 10, \top)/\epsilon$</td>
</tr>
</tbody>
</table>

Figure 8: Execution of an enforcement monitor with input $(1, Auth) \cdot (2, LockOn) \cdot (4, Write) \cdot (5, LockOff) \cdot (6, LockOn) \cdot (7, Write) \cdot (8, LockOff)$
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 \( A_\varphi = (Q, q_0, \Sigma, \to, F) \). 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 \( E_\varphi(\sigma) = E_\varphi(\sigma, \sigma') \preceq E_\varphi(\sigma, \sigma') \).

Otherwise, let us consider \( (\sigma_s, \sigma_d) = store_\varphi(\sigma), a = \sigma'(1), \) and \( (\sigma_s, \sigma_d) = store_\varphi(\sigma, a) \). Then, if \( a \in \Sigma_u \), \( \sigma_l = \sigma_s.a.\sigma'_{+} \), where \( \sigma'_{+} \) defined in Definition 8, meaning that \( \sigma_s \preceq \sigma_l \). If \( a \in \Sigma_c \), then \( \sigma_l = \sigma_s.a'_{+} \), where \( \sigma'_{+} \) defined in Definition 8, thus again, \( \sigma_s \preceq \sigma_l \). In both cases, \( E_\varphi(\sigma) = \sigma_s \preceq \sigma_l = E_\varphi(\sigma, a) \).

Since the order \( \preceq \) is transitive, this means that \( E_\varphi(\sigma) \preceq E_\varphi(\sigma, a) \preceq E_\varphi(\sigma, a.\sigma'(2)) \) \( \ldots \preceq E_\varphi(\sigma, \sigma') \).

Thus \( E_\varphi \) is an enforcement function. \( \square \)

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

*Proof.* For \( \sigma \in \Sigma^*_c \), let \( P(\sigma) \) be the predicate \( "\forall a \in \Sigma_c, I(\sigma) \subseteq I(\sigma, a)" \). Let us show by induction that \( P(\sigma) \) holds for all \( \sigma \in \Sigma^*_c \).

--- Induction basis: if \( a \in \Sigma_c \), then since \( I(\epsilon) = \emptyset \), \( I(\epsilon) \subseteq I(a) \). Thus, \( P(\epsilon) \) holds.

--- Induction step: let us suppose that for \( \sigma \in \Sigma^*_c \), \( |\sigma| \leq n \), \( P(\sigma) \) holds. Let us then consider \( \sigma \in \Sigma^*_c \) s.t. \( |\sigma| = n + 1 \), and \( a \in \Sigma_c \). Let \( (h, \sigma_0) \in \Sigma_c \times \Sigma^*_c \) 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 \( I(\sigma_0) \subseteq I(\sigma_0, a) \). 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 \( I(\sigma) = I(h.\sigma_0) \subseteq \text{Pred}_h(S(\sigma_0) \cup I(\sigma_0)) \subseteq \text{Pred}_h(S(\sigma_0, a) \cup I(\sigma_0, a)) = I(h.\sigma_0, a) = I(\sigma, a) \).

This means that \( P(\sigma, a) \) holds.

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

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

*Proof.* For \( \sigma \in \Sigma^*_c \), 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 \( u \in \Sigma_u \), \( u \in \Sigma^*_c \), and following the definition of \( S(\epsilon) \), \( q \text{ after } u \in S(\epsilon) \cup I(\epsilon) \). Thus, \( q \text{ after } u \in S(\epsilon) \cup I(\epsilon) \).
—**Induction step:** let us suppose that for \( \sigma \in \Sigma^*_n \), \( P(\sigma) \) holds. Let us then consider \( u \in \Sigma_u \), \( a \in \Sigma_c \), and \( q \in S(\sigma.a) \). Then, either \( q \in S(\sigma) \) or \( q \in \max_<(\{Y \subseteq F_C \mid Y \cap u\text{Pred}(Y \cup I(\sigma.a)) = \emptyset\}) \). 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 \( 1 \), \( I(\sigma) \subseteq S(\sigma.a) \), 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, \( q \in \max_<(\{Y \subseteq F_C \mid Y \cap u\text{Pred}(Y \cup I(\sigma.a)) = \emptyset\}) \), 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 \( u \in \Sigma_u \), for all \( q \in Q \), \( (q \) 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 \( \langle \epsilon, \epsilon \rangle \), and \( q \) after \( \epsilon = q \in S(\epsilon) = S(\epsilon^{-1}, \epsilon) \). Thus, \( \epsilon \in G(\sigma, q) \), meaning that \( G(q, \epsilon) \neq \emptyset \), and thus that \( P(\sigma) \) holds.

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

- \( q \in S(\sigma.a) \), then \( \epsilon \in S(\sigma) \cup I(\sigma) \) is s.t. \( \epsilon \in S(\sigma.a) \) and \( \epsilon \) after \( \sigma \) in \( S(\sigma.a) = S(\sigma^{-1}, \sigma.a) \), 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, \( I(\sigma.a) = I(h, \sigma_0) = \text{Pred}_a(S(\sigma_0 \cup I(\sigma_0))) \), meaning that \( q \in \text{Pred}_a(S(\sigma_0 \cup I(\sigma_0))) \). By induction hypothesis, \( S(\sigma_0) = |\sigma_0| = n \), \( P(\sigma_0) \) holds, meaning that \( G(q, h, \sigma_0) \neq \emptyset \). Let us then consider \( w \in G(q, h, \sigma_0) \). Then, \( w \) is \( \langle s, w \rangle \) for all \( s \in S(\sigma.a) \) and \( q \) after \( h \) in \( w \in S(\sigma^{-1}, \sigma.a) \). Thus, \( h.w \leq h.\sigma_0 \) and \( q \) after \( h \) in \( w \in S(\sigma^{-1}, \sigma.a) = S((h.w)^{-1}.(h.\sigma_0)) \). Thus, \( h.w \in G(q, h.\sigma_0) = G(q, \sigma.a) \).

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.** \( \forall \sigma \in \Sigma^*_n \), \( \sigma \notin \text{Pre}(\phi) \land (\sigma, \sigma.e) = \text{store}_\phi(\sigma) \) \( \Rightarrow \) \( \sigma_s = \sigma|\Sigma_u \land \sigma_e = \sigma|\Sigma_c \).

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

—**Induction basis:** \( \text{store}_\phi(\epsilon) = (\epsilon, \epsilon) \), and since \( \epsilon|\Sigma_u = \epsilon|\Sigma_c = \epsilon = \epsilon \), \( P(\epsilon) \) holds.

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

- \( a \in \Sigma_u \), then \( \sigma_1 = \sigma.a, \sigma_2 \) with \( \sigma_1 \in G(\text{Reach}(\sigma.a)), \sigma_2 \in \{\epsilon\} \). Since \( \sigma.a \notin \text{Pre}(\phi) \), \( G(\text{Reach}((\sigma.a)|\Sigma_u), (\sigma.a)|\Sigma_u) = \emptyset \). Moreover, since \( a \in \Sigma_u \), \( (\sigma,a)|\Sigma_u = \sigma|\Sigma_u, a = a_1, a_2 \), and \( (\sigma,a)|\Sigma_u = \sigma|\Sigma_u \), thus \( G(\text{Reach}(\sigma.a), \sigma_e) = \emptyset \). It follows that \( \sigma_s' \in \{\epsilon\} \), meaning that \( \sigma_1 = \sigma.a, \sigma_s = \sigma|\Sigma_u, a = (\sigma,a)|\Sigma_u \), and \( \sigma_2 = \sigma_2 \). Thus, \( \sigma_e = \sigma|\Sigma_c \).

- \( a \in \Sigma_c \), then \( \sigma_1 = \sigma.a, \sigma_2 \) with \( \sigma_1 \) \( \notin \text{G}(\sigma|\Sigma_u, \sigma.e) \) \( \cup \{\epsilon\} \). Since \( \sigma.a \notin \text{Pre}(\phi) \), \( G(\text{Reach}((\sigma.a)|\Sigma_u), (\sigma.a)|\Sigma_u) = \emptyset \). Moreover, since \( a \in \Sigma_c \), \( (\sigma,a)|\Sigma_u = \sigma|\Sigma_u = \sigma.a, \) and \( (\sigma,a)|\Sigma_u = \sigma|\Sigma_u \), thus \( G(\text{Reach}(\sigma.a), \sigma.e) = \emptyset \). It follows that \( \sigma_s' \in \{\epsilon\} \), meaning that \( \sigma_1 = \sigma.a, \sigma_s = \sigma|\Sigma_u, a = (\sigma,a)|\Sigma_u \), and \( \sigma_2 = \sigma_2.\epsilon = \sigma_e = \sigma|\Sigma_c \).

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

Proof. Let $P(\sigma)$ be the predicate: "$(\sigma \in \text{Pre}(\varphi) \land (\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma)) \implies (E_\varphi(\sigma) \models \varphi \land \text{Reach}(\sigma_s) \in S(\sigma_c))"$. 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 $a \in \Sigma_s(\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma)$, and $(\sigma_t, \sigma_u) = \text{store}_\varphi(\sigma, a)$. 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 $G(\text{Reach}((\sigma \cdot a)|_{\Sigma_s}, (\sigma \cdot a)|_{\Sigma_c}) \neq \emptyset$. Moreover, since $\sigma \not\in \text{Pre}(\varphi)$, following lemma 4, $\sigma_s = \sigma_s^{\Sigma_s}$ and $\sigma_c = \sigma_c^{\Sigma_c}$. Now, we consider two cases:

- If $a \in \Sigma_a$, then $(\sigma, a)|_{\Sigma_a} = \sigma_s^{\Sigma_s} \cdot a = \sigma_s \cdot a$, and $(\sigma, a)|_{\Sigma_c} = \sigma_c^{\Sigma_c} = \sigma_c$. Thus, $G(\text{Reach}(\sigma, a), \sigma_c) \neq \emptyset$, meaning that $\sigma'_{\Sigma} = (\sigma_{\Sigma})^{-1} \cdot \sigma_t \in G(\text{Reach}(\sigma, a), \sigma_c)$. Then, following the definition of $G$, $\text{Reach}(\varphi, a)$ after $\sigma'_{\Sigma} = \text{Reach}(\sigma, a, \sigma'_{\Sigma}) = \text{Reach}(\sigma_t) \in S(\sigma^{-1}_{\Sigma_c}, \sigma_c = S(\sigma_d))$. Moreover, since $S(\sigma_d) \subseteq F_G$, $E_\varphi(\sigma) = \varphi$. This means that $P(\sigma, a)$ holds.

- If $a \in \Sigma_a$, then $(\sigma, a)|_{\Sigma_a} = \sigma_s^{\Sigma_s} = \sigma_s$ and $(\sigma, a)|_{\Sigma_c} = \sigma_c^{\Sigma_c} = \sigma_c$. Thus, $G(\text{Reach}(\sigma, a), \sigma_c) \neq \emptyset$, meaning that $\sigma'_{\Sigma} = (\sigma_a)^{-1} \cdot \sigma_t \in G(\text{Reach}(\sigma, a), \sigma_c)$, as in the case where $a \in \Sigma_a$, it follows that $\text{Reach}(\sigma_t) \in S(\sigma_d)$ and $E_\varphi(\sigma) = \varphi$. This means that $P(\sigma, a)$ holds.

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

$\sigma \in \text{Pre}(\varphi)$ (and then $(\sigma, a) \in \text{Pre}(\varphi)$ since $\text{Pre}(\varphi)$ is extension-closed). Then, by induction hypothesis, $P(\sigma)$ holds, meaning that $\text{Reach}(\sigma) \in S(\sigma_d)$ and $E_\varphi(\sigma) \models \varphi$. Again, we consider two cases:

- If $a \in \Sigma_a$, then, since $\text{Reach}(\sigma) \in S(\sigma_d)$, following lemma 2, $\text{Reach}(\sigma_t) \in S(\sigma_d)$ after $\sigma = \text{Reach}(\sigma, a) \in S(\sigma_d) \cup I(\sigma_d)$. Following lemma 3, $G(\text{Reach}(\sigma, a), \sigma_d) \neq \emptyset$. Thus $\sigma'_{\Sigma} = (\sigma_a)^{-1} \cdot \sigma_t \in G(\text{Reach}(\sigma, a), \sigma_c)$. It follows that $\text{Reach}(\sigma, a, \sigma'_{\Sigma}) = \text{Reach}(\sigma_t) \in S(\sigma^{-1}_{\Sigma_c}, \sigma_c) = S(\sigma_d)$, and thus, since $S(\sigma_d) \subseteq F_G$, $E_\varphi(\sigma, a) = \sigma_t \models \varphi$. Henceforth, $P(\sigma, a)$ holds.

- If $a \in \Sigma_a$, then, since $\text{Reach}(\sigma) \in S(\sigma_d)$ and $S(\sigma_a) \subseteq S(\sigma_a)$, $\text{Reach}(\sigma_t) \in S(\sigma_d)$. Following lemma 3, $G(\text{Reach}(\sigma, a), \sigma_d) \neq \emptyset$. Thus, $\sigma''_{\Sigma} = (\sigma_a)^{-1} \cdot \sigma_t \in G(\text{Reach}(\sigma, a), \sigma_d)$. As in the case where $a \in \Sigma_a$, it leads to $\text{Reach}(\sigma_t) \in S(\sigma_d)$ and $E_\varphi(\sigma, a) \models \varphi$. Henceforth, $P(\sigma, a)$ holds.

Thus, if $\sigma \in \text{Pre}(\varphi)$, $P(\sigma, a)$ holds.

In all cases, $P(\sigma, a)$ holds. Thus, $P(\sigma) \implies P(\sigma, a)$. By induction on $\sigma$, $\forall \sigma \in \Sigma^*, (\sigma \in \text{Pre}(\varphi) \land (\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma)) \implies (E_\varphi(\sigma) \models \varphi \land \text{Reach}(\sigma) \in S(\sigma_d))$. 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: "$(\sigma_s, \sigma_c) = \text{store}_\varphi(\sigma)) \implies (\sigma_s|_{\Sigma_s}, \sigma_c = \sigma_s|_{\Sigma_s} \land \sigma_c|_{\Sigma_c} = \sigma_c|_{\Sigma_c})"$. Let us prove that for all $\sigma \in \Sigma^*$, $P(\sigma)$ holds.

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

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

— Case 1: $a \in \Sigma_a$. Then, $\sigma_s = \sigma_s \cdot a', \sigma_c'$ is defined in Definition 8, and $\sigma_s \cdot a' = \sigma_s \cdot a \cdot \sigma_c'$. Therefore, $\sigma_s|_{\Sigma_s} = (\sigma_t, \sigma_d)|_{\Sigma_s}$, since $\sigma_d \in \Sigma_d$, and $\sigma_s \in \Sigma_s$. Thus, $\sigma_t|_{\Sigma_s} = \sigma_t|_{\Sigma_s} \cdot \sigma_c$. Since $P(\sigma)$ holds, $\sigma_t|_{\Sigma_s} \cdot \sigma_c = \sigma_s|_{\Sigma_s} = (\sigma_t, \sigma_d)|_{\Sigma_s}$.
Moreover, since $\sigma_1' \in \Sigma^*_u, \sigma_1' |_{\Sigma_u^u} = \sigma_2 |_{\Sigma_u^u}.a$. Since $P(\sigma)$ holds, this means that $\sigma_1 |_{\Sigma_u^u} = \sigma_2 |_{\Sigma_u^u}.a = (\sigma.a) |_{\Sigma_u^u}$. Thus $P(\sigma.a)$ holds.

Case 2: $a \in \Sigma_e$. Then $\sigma_1 = \sigma_s, \sigma_2', \sigma_3'$ where $\sigma_2'$ is defined in Definition 8, and $\sigma_1, \sigma_2 = \sigma_s, \sigma_e.a$. Therefore, $\sigma_1 |_{\Sigma_u^u}, \sigma_2 = (\sigma_1, \sigma_2) |_{\Sigma_u^u} = (\sigma_1, \sigma_2) |_{\Sigma_u^u} = \sigma_3 |_{\Sigma_u^u}.a. Since$ $P(\sigma)$ holds, this means that $\sigma_1 |_{\Sigma_u^u}, \sigma_2 = (\sigma.e) |_{\Sigma_u^u}$.

Moreover, since $\sigma_1' \in \Sigma_s, \sigma_1' |_{\Sigma_u^u} = \sigma_2 |_{\Sigma_u^u}$. Since $P(\sigma)$ holds, this means that $\sigma_1 |_{\Sigma_u^u} = \sigma_2 |_{\Sigma_u^u} = (\sigma.a) |_{\Sigma_u^u}$. Thus $P(\sigma.a)$ holds.

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

By induction on $\sigma$, for all $\sigma \in \Sigma^*_u, (\sigma_s, \sigma_e) = \text{store}_\varphi(\sigma) \Rightarrow (\sigma_s |_{\Sigma_u^u}, \sigma_e = \sigma |_{\Sigma_u^u} \land \sigma_2 |_{\Sigma_u^u} = \sigma |_{\Sigma_u^u}.$

Moreover, if $\sigma \in \Sigma^*_u, (\sigma_s, \sigma_e) = \text{store}_\varphi(\sigma)$, and $(\sigma_2, \sigma_2) = \text{store}_\varphi(\sigma.a), then \sigma_2 = \sigma.s.a.\sigma_3'$, where $\sigma_3'$ is defined in Definition 8. Thus $\sigma.s. \leq \sigma_1$, and since $\sigma_s = \text{E}_\varphi(\sigma)$, and $\sigma_1 = \text{E}_\varphi(\sigma.a)$, it follows that $\text{E}_\varphi(\sigma.a) \leq \text{E}_\varphi(\sigma.a)$. Thus, for all $\sigma \in \Sigma^*_u, \text{E}_\varphi(\sigma) \leq \text{E}_\varphi(\sigma.a) \land \text{E}_\varphi(\sigma.a) = \sigma |_{\Sigma_u^u} \land \forall u \in \Sigma_u, \text{E}_\varphi(\sigma.a), u \leq \text{E}_\varphi(\sigma.a)$. meaning that $E_\varphi$ is compliant.

**Lemma 5.** \(\forall \sigma \in \Sigma^*_u, \forall q \in Q, (q \not\in S(\sigma)) \Rightarrow \exists \sigma' \in \Sigma^*_u, q \sigma u \in F \land \forall \sigma'' \in \Sigma^*_u, q \sigma u \not\in \epsilon \Rightarrow q \sigma u \not\in S(\sigma) \cup I(\sigma).$$

**Proof.** For $\sigma \in \Sigma^*_u$ and $q \in Q$, let $P(\sigma, q)$ be the predicate “$\forall u \sigma u \in \Sigma^*_u, q \sigma u \in F \land \forall \sigma'' \in \Sigma^*_u, q \sigma u \not\in \epsilon \Rightarrow q \sigma u \not\in S(\sigma) \cup I(\sigma).$ Let us show the contrapositive of the lemma, that is, for all $\sigma \in \Sigma^*_u$ and $q \in Q, P(\sigma, q) \Rightarrow q \in S(\sigma)$. We consider two cases:

- If $\sigma = \epsilon$, let us consider $q \in Q$ s.t. $P(\epsilon, q)$ holds. Then, since $\epsilon \in \Sigma^*_u$ and there does not exist a word $w$ satisfying $w \leq \epsilon \land w \not= \epsilon$, it follows that $q = q \sigma \epsilon \epsilon \in F$. Let us consider $\sigma \in \Sigma^*_u$. Then, since $P(\epsilon, q)$ holds, either $q \sigma u \in F$, or there exists $\sigma' \leq \sigma_1$ such that $\sigma' \not= \epsilon$ and $q \sigma u \not\in S(\epsilon) \cup I(\epsilon)$. In this last case, since $I(\epsilon) = \emptyset$, $q \sigma u \not\in S(\epsilon)$. Following the definition of $S(\epsilon)$, since $\sigma' \not\in S(\epsilon)$, $(q \sigma u \sigma') \not\in S(\epsilon)$. Thus, in all cases $q \sigma u \not\in F$. Thus, for all $\sigma \in \Sigma^*_u$, $q \sigma u \not\in F$, meaning that $q \in S(\sigma)$.

- If $\sigma \not= \epsilon$, there exists $\sigma' \in \Sigma^*_u$ and $\alpha \in \Sigma$ s.t. $\sigma = \sigma' \alpha$, meaning that $S(\sigma)$ is s.t. $S(\sigma) = S(\sigma') \cup \max_{\Sigma_u}((Z \subseteq F \mid Z \cap u\text{Pred}(Z \cup I(\sigma)) = \emptyset)).$ Let us consider $q \in Q$ s.t. $P(\sigma, q)$ holds. Then, we define $S(\sigma) = S(\epsilon) \cup I(\epsilon)$. Since $P(\sigma, q)$ holds, $Y \subseteq F$. Moreover, if $y \in Y$ and $u \in \Sigma_u$, then:

- either $y \sigma u \not\in S(\sigma) \cup I(\sigma)$, and then $y \sigma u \not\in (Y \cup S(\sigma)) \cup I(\sigma)$,
- or $y \sigma u \not\in S(\sigma) \cup I(\sigma)$. Then, if $\sigma u \in \Sigma^*_u$ is s.t. $y \sigma u = q \sigma u \not\in S(\sigma)$, then $y \sigma u = (q \sigma u \sigma) \sigma u \not\in S(\sigma) \cup I(\sigma).$ Since $\sigma u \in \Sigma^*_u$ and $y \sigma u \not\in (Y \cup S(\sigma)) \cup I(\sigma)$.

Thus, $y \sigma u \not\in (Y \cup S(\sigma)) \cup I(\sigma)$, and since following lemma 2, $S(\sigma) \cap u\text{Pred}(S(\sigma) \cup I(\sigma)) = \emptyset$, it means that $(Y \cup S(\sigma)) \cap u\text{Pred}(Y \cup S(\sigma) \cup I(\sigma)) = \emptyset$. It follows that $(Y \cup S(\sigma)) \subseteq \max_{\Sigma_u}((Z \subseteq F \mid Z \cap u\text{Pred}(Z \cup I(\sigma)) = \emptyset))) \subseteq S(\sigma).$ Since $q \in Y \subseteq S(\sigma)$, this means that $q \in S(\sigma)$.

Thus, for $\sigma \in \Sigma^*_u$ and $q \in Q, P(\sigma, q) \Rightarrow q \in S(\sigma)$, this means that the contrapositive also holds, thus $q \in S(\sigma) \Rightarrow \neg P(\sigma, q)$, meaning that $q \not\in S(\sigma) \Rightarrow \exists \sigma u \in \Sigma_u, q \sigma u \not\in S(\sigma) \cup I(\sigma).$
a ∈ Σo. Then, since E is compliant, and E(σ) = Eϕ(σ) = σ, there exists σ′ ∈ τ such that E(σ.a) = E(σ.a.a).σ = σ.a.a.σ. Moreover, there exists σ′ ∈ τ such that E(σ.a) = E(σ.a.a).σ = σ.a.a.σ′. Since |E(σ.a)| > |Eϕ(σ)|, |σ| > |σ′|, it follows that σ′ /∈ G(Reach(σ, a), σ). Therefore, the definition of G, this means that either σ′ /∈ τ, but since E’ is compliant, this is not possible, or that Reach(σ, a) after σ′ /∈ S(σ′, a, σ). Let us consider q = Reach(σ, a, σ) and σ = σ′, σ. Then, q /∈ S(σ, a). Following lemma 5, this means that there exists σu ∈ Σo such that q after σu /∈ F and for all σu /∈ σu, σu /∈ ϕ → q after σu /∈ S(σ, a) ∪ I(σ, a).

Then, we consider two cases:

- If E(σ.a.a, σu) = σ.a.a, σu, then Reach(E(σ.a.a, σu)) /∈ F, meaning that E(σ.a.a, σu) /∉ ϕ.

- Otherwise, since E is compliant, there exists σu and σu /∈ σ u.t. σu /∈ σ, σu /∈ ϕ, and E(σ.a.a, σu) = σ.a.a, σu, σu, σu. Let us consider q = q after σu, σu and σu = σu, σu. Then, since σu and σu /∈ q after σu, σu /∈ S(σu) ∪ I(σu). Thus, q = q after σu, σu /∈ S(σu) ∪ I(σu), because otherwise, q after σu = Pred σu(q) = Pred σu(S(σu) ∪ I(σu)) ⊆ I(σu), which is absurd. Then, we can again use lemma 5 to find a word σu /∈ Σo such that q after σu /∈ F and for any σ′ /∈ σu, q′ after σ′ /∈ S(σu) ∪ I(σu). Since σu /∈ ϕ, |σu| < |σu|, thus the operation can be repeated a finite number of times (at most until all the controllable events of σ appear in the output of E). Thus, there exists n ∈ N, there exists σu, σu, ..., σu, and σ1, σ2, ..., σm, such that E(σ1, σ2, ..., σm), E(σ1, σ2, ..., σm), E(σ1, σ2, ..., σm) = σ1.a, σ1.a, σ1.a, σ2.a, σ2.a, ..., σm.a, σm.a, and Reach(σ1.a, σ1.a, σ1.a, σ2.a, σ2.a, ..., σm.a, σm.a) /∈ F. This means that, if σu = σu, σ2, ..., σm, then σu /∈ Σo and E(σ.a, a, σu) /∉ ϕ.

Thus, in all cases, there exists σu ∈ Σo s.t. E(σ.a, a, σu) /∉ ϕ.

a ∈ Σo. The proof is the same as in the case where a ∈ Σo, by replacing occurrences of “σ, a” by “σ”, and occurrences of “σ, a” by “σ, a”.

Thus, if E is an enforcement function s.t. there exists σ ∈ Pre(ϕ), and a ∈ Σ compliant(E, Σo, Σo), E(σ) = Eϕ(σ), and |E(σ.a)| > |Eϕ(σ.a)|, then there exists σu ∈ Σo s.t. E(σ.a, a, σu) /∉ ϕ. This means that Eϕ is optimal in Pre(ϕ).

Proposition 5. The output of the enforcement monitor E for input σ is Eϕ(σ).

Proof. Let us introduce some notation for this proof: for a word w ∈ Γ∗, we note input(w) = Π1(w(1)).Π1(w(2))...Π1(w(|w|)), the word obtained by concatenating the first members (the inputs) of w. In a similar way, we note output(w) = Π1(|w(1)|)...Π1(|w(|w|)|), the word obtained by concatenating all the last members (the outputs) of w. Since all configurations are not reachable from c0, for w ∈ Γ∗, we note Reach(w) = c whenever c0 c0...c w c, and Reach(w) = ⊥ if such a c does not exist. We also define the Rules function as follows:

\[
\begin{align*}
\text{Rules} : \quad \Sigma^* & \rightarrow \Gamma^* \\
\sigma & \mapsto \max_{\epsilon}(\{w \in \Gamma^* \mid \text{input}(w) = \sigma \land \text{Reach}(w) = \bot\})
\end{align*}
\]

For a word σ ∈ Σ∗, Rules(σ) is the trace of the last valid output in E, i.e. the sequence of all the rules that can be applied with input. We then extend the definition of output to words in Σ∗; for σ ∈ Σ∗, output(σ) = output(Rules(σ)). We also note ε the empty word of Σ∗, and ε the empty word of Γ∗. For σ ∈ Σ∗, let P(σ) be the predicate: “Eϕ(σ) = output(σ) ∧ ((σ, ϕ) = storeϕ(σ) \land \text{Reach}(\text{Rules}(σ)) = (q, σ)|).”

Let us prove by induction that for all σ ∈ Σ∗, P(σ) holds.

**Induction basis:** Eϕ(ε) = ε = output(ε). Moreover, storeϕ(ε) = (ε, ε), and Reach(ε) = c0. Therefore, as c0 = (q0, ε), P(ε) holds, because Reach(ε) = q0.

**Induction step:** Let us suppose now that for some σ ∈ Σ∗, P(σ) holds. Let us consider (σ, σo) = storeϕ(σ), q = Reach(σo), a ∈ Σ, and (σ1, σo) = storeϕ(σ.a). Let us prove that P(σ.a) holds.
Since \( P(\sigma) \) holds, Reach(Rules(\sigma)) = \( \{q, \sigma_c\} \) and \( \sigma_s = \text{output}(\sigma) \). We consider two cases:

\(<a \in \Sigma_a. Then, considering \( \sigma'_s = (\sigma_s.a)^{-1}.\sigma_t \), \( \sigma_t = \sigma_s.a.\sigma'_s \). Since \( a \in \Sigma_a \), rule pass-untcont can be applied: let us consider \( q' = q \) after \( a \). Then, \( \langle q, \sigma'_c \rangle \xrightarrow{a/\text{pass-untcont}(a)/\epsilon} \langle q', \sigma_c \rangle \). Then, if \( \sigma'_s = \epsilon \), \( G(q', \sigma_c) = \emptyset \) or \( G(q', \sigma_c) = \{\epsilon\} \), meaning that no other rule can be applied, and thus \( P(\sigma.a) \) would hold. Otherwise, \( \sigma'_s \neq \epsilon \), and thus \( \sigma'_s \in G(q', \sigma_c) \), meaning that \( G(q', \sigma_c) \neq \emptyset \) and \( G(q', \sigma_c) \neq \{\epsilon\} \), thus rule dump(\( \sigma_c(1) \)) can be applied. Since \( \sigma'_s \neq \sigma_c, \sigma'_s(1) = \sigma_c(1) \), thus if \( q_1 = q' \) after \( \sigma_c(1), q_1 = q' \) after \( \sigma'_s(1) \). If \( \sigma'_s(1)^{-1}.\sigma'_s \neq \epsilon \), then \( \sigma'_s(1)^{-1}.\sigma'_s \in G(q_1, \sigma_c(1)^{-1}.\sigma_c) \), meaning that rule dump can be applied again. Rule dump can actually be applied \( |\sigma'_s| \) times, since for all \( w \neq \sigma'_s \), then \( w^{-1}.\sigma'_s \neq \epsilon \) and \( w^{-1}.\sigma'_s \in G(q, w^{-1}.\sigma_{-c}) \). Thus, after rule dump has been applied \( |\sigma'_s| \) times, the configuration reached is \( \langle q' \rangle \sigma'_s. \sigma_s^{-1}.\sigma_c \rangle. Moreover, the output produced by all the rules dump is \( \sigma'_s \). Since no rule can be applied after the \( \sigma'_s \) applications of the rule dump, \( \text{output}(\sigma.a) = \langle q' \rangle \sigma'_s \sigma_s^{-1}.\sigma_c \rangle = \langle q' \rangle \sigma'_s \sigma_s^{-1}.\sigma_c \rangle \sigma_{-c} \) (after \( a \) after \( \sigma'_s, \sigma_{-c} = \langle \text{Reach}(\sigma_{-c}) \rangle \sigma_{-c} = \langle \text{Reach}(\sigma_{-c}, a.\sigma'_s), \sigma_{-c} = \langle \text{Reach}(\sigma_{-c}) \rangle \sigma_{-c} \rangle \). Thus, if \( a \in \Sigma_a, P(\sigma.a) \) holds.

\(<a \in \Sigma_a. Then, considering \( \sigma''_s = \sigma_s^{-1}.\sigma_t \), \( \sigma_t = \sigma_s.\sigma''_s \). Since \( a \in \Sigma_a \), it is possible to apply the store-cont rule, and \( \langle q, \sigma_{-c} \rangle \) after \( a/\text{store-cont}(a)/\epsilon = \langle q, \sigma_{-c} \rangle \). Then as in the case where \( a \in \Sigma_a \), rule dump can be applied \( |\sigma''_s| \) times, meaning that the configuration reached would then be \( \langle q \rangle \sigma_{-c}(1). \sigma_{-c}(2)^{-1}. \ldots. \sigma_{-c}(a^{-1}). \sigma_{-c}(a). \sigma_{-c}(a). \sigma_{-c}(a). \sigma_{-c}(a). \sigma_{-c}(a) \rangle = \sigma''_s \). Since \( \sigma''_s \neq \sigma_{-c}, \sigma''_s(1) = \sigma_{-c}(1) \), thus \( \text{Reach}(\sigma_{-c}) = \langle \text{Reach}(\sigma_{-c}, a.\sigma''_s), \sigma_{-c} = \langle \text{Reach}(\sigma_{-c}) \rangle \sigma_{-c} \rangle \). Moreover, \( \text{output}(\sigma.a) = \text{output}(\sigma).\sigma''_s = \sigma_{-c}.\sigma''_s = \sigma_t = \langle q \rangle \text{Reach}(\sigma_{-c}) \rangle \). Thus, if \( a \in \Sigma_a, P(\sigma.a) \) holds. This means that \( P(\sigma) \Rightarrow P(\sigma.a) \). Thus, by induction on \( \sigma \), for all \( \sigma \in \Sigma^\ast, P(\sigma) \) holds. In particular, for all \( \sigma \in \Sigma^\ast, E_{\epsilon}(\sigma) = \text{output}(\sigma) \).

A.2. Proofs for the Timed Setting

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

Proof. For \( a \in \text{tw}(\Sigma) \), let \( P(\sigma) \) be the predicate: “\( \forall t \in \mathbb{R}_{\geq 0}, \forall t' \geq t, E_\epsilon(\sigma, t) \leq E_\epsilon(\sigma, t') \)”.

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

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

- If \( t \geq t' \), then let us consider \( \sigma, \sigma_0, \sigma_c = \text{store}_a(\sigma, t'), (\sigma_1, \sigma_{a1}, \sigma_{a1}) = \text{store}_a(\sigma(t', a), t) \), and \( \sigma_{a2} = \text{store}_a(\sigma(t', a), t) \). Then, \( \text{output}(\sigma(t', a)) = \sigma_{a1} = \text{E}(\sigma(t', a), t) = \sigma_{a2} \).

- If \( a \in \Sigma_a \), then considering \( t_1 \) as defined in Definition 16, \( t_1 = \min\{t_0 \in \mathbb{R}_{\geq 0} \mid t_0 \geq t' \wedge \text{Reach}(\sigma(t', a), t_0, \Pi_{\text{obs}(\sigma(t', a)))} \neq 0\} \). \( \sigma_{a2} = \min_{\text{max}_{\text{abs}}}(G(\text{Reach}(\sigma_{a2}(t', a), t_1)), \Pi_{\text{obs}(\sigma_{a2}(t'))}.(\sigma_{a2}) \cup \{\epsilon\} +_1 \min\{t, t_1\}, \} \), and \( \sigma_{a2} = \min_{\text{max}_{\text{abs}}}(G(\text{Reach}(\sigma_{a2}(t'), a, t_1)), \Pi_{\text{obs}(\sigma_{a2}(t'))}.(\sigma_{a2}) \cup \{\epsilon\} +_1 \min\{t, t_1\}) \).

- Case 1: \( t \geq t_1 \). Since \( t' \geq t \), then \( t' \geq t_1 \). Only \( \min\{t, t_1\} = t_1 \), thus \( \sigma_{a1} = \sigma_{a2} \).

- Case 2: \( t < t_1 \). Then, \( \min\{t, t_1\} = t \). Since \( t < t_1 \), by definition of \( t_1 \), this means that \( G(\text{Reach}(\sigma_{a2}(t')) \neq 0 \), and thus \( \sigma_{a1} = \epsilon \). Since \( \sigma_{a1} = \epsilon, \sigma_{a1} = \sigma_{a2} \).

It follows that:

\( \sigma_{a1} = \sigma_{a2} \).

- Case 2: \( t < t_1 \). Then, \( \min\{t, t_1\} = t \). Since \( t < t_1 \), by definition of \( t_1 \), this means that \( G(\text{Reach}(\sigma_{a2}(t'), a, t)) = 0 \), and thus \( \sigma_{a1} = \epsilon \). Since \( \sigma_{a1} = \epsilon, \sigma_{a1} = \sigma_{a2} \).

\( \sigma_{a2} \neq \sigma_{a2} \).
Thus, if $t' \geq t \geq t''$ and $a \in \Sigma_n$, $P(\sigma) \implies E_\varphi(\sigma(t'',a),t) \leq E_\varphi(\sigma(t',a),t')$.

- Otherwise, $a \in \Sigma_n$, and then considering $t_2$ as defined in Definition 16, $t_2 = \min\{t_0 \in \mathbb{R}_{\geq 0} \mid t_0 \geq t'' \land G(\text{Reach}(\sigma,t_0), \Pi_2(\text{obs}(\sigma,t'')), \sigma\not= \emptyset)\}$. Then, $\sigma\not=a = \min_{\text{exec}}(\max_{\text{exec}}(G(\text{Reach}(\sigma, \min \{t_1, t_2\})), \Pi_2(\text{obs}(\sigma,t''))), \sigma, a) \cup \{t\}) + \min \{t_1, t_2\}$, and:

\[
\sigma_{d_2} = \min \{\text{exec}(G(\text{Reach}(\sigma), \min \{t_1, t_2\})) \}, \Pi_2(\text{obs}(\sigma,t''), \sigma, a) \cup \{t\}\} + \min \{t_1, t_2\}.
\]

- 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 $\sigma_{d_1} = \sigma_{d_2}$. It follows that $\sigma_{d_1} = \sigma_{s_0, \text{obs}(\sigma_{d_1}, t')} = \sigma_{s_0, \text{obs}(\sigma_{d_2}, t')} = \sigma_{d_2}$.

- Case 2: $t < t_2$. Then, $G(\text{Reach}(\sigma, \min \{t_1, t_2\}), \Pi_2(\text{obs}(\sigma, t''), \sigma, a)) = \emptyset$, meaning that $\sigma_{d_1} = \sigma_{s_0, \text{obs}(\sigma, t')} = \emptyset$. Thus, $\sigma_{d_1} = \sigma_{s} = \sigma_{d_2}$.

Thus, if $t' \geq t \geq t''$ and $a \in \Sigma_n, P(\sigma) \implies E_\varphi(\sigma(t'',a),t) \leq E_\varphi(\sigma(t',a),t')$.

Therefore, if $t' \geq t \geq t''$, for all $a \in \Sigma, P(\sigma) \implies E_\varphi(\sigma(t'',a),t) \leq E_\varphi(\sigma(t',a),t')$.

- If $t' < t''$, then $t < t'$, and $\sigma(t',a), t) = \text{obs}(\sigma, t)$, and $\text{obs}(\sigma(t',a), t') = \text{obs}(\sigma, t')$. Thus, $E_\varphi(\sigma(t'',a), t) = \text{store}_\varphi(\text{obs}(\sigma(t'',a), t), t) = \text{store}_\varphi(\text{obs}(\sigma, t), t) = E_\varphi(\sigma, t)$, and $E_\varphi(\sigma(t',a), t') = \text{store}_\varphi(\text{obs}(\sigma(t',a), t'), t') = \text{store}_\varphi(\text{obs}(\sigma, t'), t') = E_\varphi(\sigma, t')$. Since $P(\sigma)$ holds, then $E_\varphi(\sigma(t'',a), t) = E_\varphi(\sigma(t',a), t')$.

- If $t < t'' < t'$, then $\text{obs}(\sigma(t',a), t) = \text{obs}(\sigma, t)$, and $\text{obs}(\sigma(t',a), t') = \text{obs}(\sigma, t')$. Let $\sigma_s, \sigma_{s_0}, \sigma_{s_0'}$ = $\text{store}_\varphi(\sigma(t',a), t)$, and $\text{store}_\varphi(\sigma(t'',a), t)$. Then, $\sigma = \sigma_{s_0}, \sigma_{s_0'}$, $\text{obs}(\sigma, t')$ if a $\in \Sigma_n$, and $\sigma_s = \sigma_{s_0}, \text{obs}(\sigma, t')$ if a $\in \Sigma_n$. In both cases, $\sigma_s = \sigma_{s_0}, \sigma_{s_0'}$. This means that $E_\varphi(\sigma(t',a), t) = E_\varphi(\sigma(t',a), t')$, and $E_\varphi(\sigma(t'',a), t) = E_\varphi(\sigma(t'',a), t')$. Thus, $E_\varphi(\sigma(t',a), t) = E_\varphi(\sigma(t',a), t') = E_\varphi(\sigma(t'',a), t')$.

Finally, $P(\sigma) \implies E_\varphi(\sigma(t',a), t) \leq E_\varphi(\sigma(t'',a), t')$. Consequently, in all cases, if $t \leq t'$, then $P(\sigma) \implies E_\varphi(\sigma(t'',a), t) \leq E_\varphi(\sigma(t',a), t')$. Finally, $P(\sigma) \implies E_\varphi(\sigma(t'',a), t) \leq E_\varphi(\sigma(t',a), t')$.

By induction, for all $\sigma \in \text{tw}(\Sigma)$, $P(\sigma)$ holds. Thus, for all $\sigma \in \text{tw}(\Sigma)$, for all $t \in \mathbb{R}, t' \geq t$, $E_\varphi(\sigma, t) \leq E_\varphi(\sigma, t')$.

Now, let us consider $\sigma \in \text{tw}(\Sigma)$, and $(t, a) \text{ s.t. } \sigma(t, a) \in \text{tw}(\Sigma)$. Then, if $(\sigma_s, \sigma_{s_0}, \sigma_{s_0'}) = \text{store}_\varphi(\sigma(t, a), t)$, then either $\sigma_t = \sigma_{s_0}(t), \sigma'_t$, or $\sigma_t = \sigma_s, \sigma'_s$, whether $a$ is controllable or uncontrollable respectively, where $\sigma_t, \sigma'_t, \sigma_s, \sigma'_s$ are defined in Definition 16. In both cases, $\sigma_{s_0} \preceq \sigma$. Thus, $E_\varphi(\sigma, t) = \Pi_2(\text{store}_\varphi(\text{obs}(\sigma, t), t)) = \sigma_{s_0} \preceq \sigma_t = \Pi_2(\text{store}_\varphi(\text{obs}(\sigma(t, a), t), t)) = E_\varphi(\sigma(t, a), t)$. 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 $t \in \mathbb{R}$ and $t' \geq t$, $E_\varphi(\sigma, t) \leq E_\varphi(\sigma(t'), t')$ and $E_\varphi(\sigma, t) \leq E_\varphi(\sigma(t, a), t)$. This means that $E_\varphi(\sigma, t)$ is an enforcement function.

**Lemma 6.** \forall t \in \mathbb{R}, \forall \sigma \in \text{tw}(\Sigma), (\sigma \not\in \text{Pre}(\varphi, t) \land (\sigma_s, \sigma_{s_0}, \sigma_{s_0'}) = \text{store}_\varphi(\sigma(t), t)) \implies (\sigma_s = \sigma_{s_0} \land \sigma_{s_0'} = \epsilon \land \sigma_{s_0} = \Pi_2(\sigma_{s_0})).

**Proof.** Let $\sigma \in \text{tw}(\Sigma)$, let $P(\sigma)$ be the predicate \"$\forall t \geq \text{time}(\sigma), (\sigma \not\in \text{Pre}(\varphi, t) \land (\sigma_s, \sigma_{s_0}, \sigma_{s_0'}) = \text{store}_\varphi(\sigma(t), t)) \implies (\sigma_s = \sigma_{s_0} \land \sigma_{s_0'} = \epsilon \land \sigma_{s_0} = \Pi_2(\sigma_{s_0}))\". Let us prove by induction that for all $\sigma \in \text{tw}(\Sigma), P(\sigma)$ holds.

- **Induction basis:** for $\sigma = \epsilon$, let us consider $t \in \mathbb{R}$. Thus, $\text{store}_\varphi(\epsilon, t) = (\epsilon, \epsilon, \epsilon)$. Considering that $\epsilon \in \text{tw}(\Sigma_0)$ and $\epsilon = \Pi_2(\epsilon_{\Sigma_0})$, $P(\epsilon)$ trivially holds (whether $\epsilon \in \text{Pre}(\varphi, t)$ or not).

- **Induction step:** suppose that for $\sigma \in \text{tw}(\Sigma), P(\sigma)$ holds. Let us consider $t' < t$. Let us also consider $\sigma_s, \sigma_{s_0}, \sigma_{s_0'} = \text{store}_\varphi(\sigma(t), t)$, and $\sigma_t = \text{store}_\varphi(\sigma(t', a), t)$. Then, if $\sigma(t, a) \in \text{Pre}(\varphi, t), P(\sigma(t, a), t)$ trivially holds. Thus, let us suppose that $\sigma(t', a) \not\in \text{Pre}(\varphi, t)$. Since $\sigma_s \preceq \sigma(t', a)$ and $t' \geq t$, it follows that $\sigma \not\in \text{Pre}(\varphi, t')$. By induction hypothesis, this means that $\sigma_s = \sigma_{s_0}, \sigma_{s_0'} = \epsilon, \text{ and } \sigma_t = \Pi_2(\sigma_{s_0})$. Then, since $\sigma(t', a) \not\in \text{Pre}(\varphi, t)$, following the definition of $\text{Pre}(\varphi, t)$, this means that for all $t'' \leq t, G(\text{Reach}(\text{obs}(\sigma(t', a), t''), t''), \Pi_2(\text{obs}(\sigma(t', a), t''), t'')) \subseteq \text{Pre}(\varphi, t)$.
Lemma 8. Optimal Enforcement of (Timed) Properties with Uncontrollable Events
Suppose now that for \( u \subseteq \sigma \), \( \Pi_{\Sigma}((\sigma,(t',a)))_{\Sigma_u} = \emptyset \) (since \( t \geq t' \), \( \text{obs}(\sigma,(t',a),t) = \sigma.(t',a) \)). Then, there are two cases:

- Case 1: \( a \in \Sigma_u \). Then, since \( (\sigma,(t',a))_{\Sigma_u} = \sigma_{\Sigma_u}(t',a) = \sigma.(t',a) \), and \( \Pi_{\Sigma}((\sigma,(t',a)))_{\Sigma_u} = \Pi_{\Sigma}(\text{obs}(\sigma,b)) \), we have \( G(\text{reach}(\sigma,(t',a),t),\Pi_{\Sigma}(\sigma,(t',a),t)) = \emptyset \). This means that \( t < t_2 \), where \( t_1 \) is defined in Definition 16, and thus \( \sigma\epsilon = \epsilon \). Since \( \sigma\epsilon = \sigma.b \), \( \text{obs}(\sigma\epsilon,t) = \sigma.b \). Let us consider \( \sigma.b = (\sigma.(t',a))_{\Sigma_u} \), and \( \sigma\epsilon = \sigma.b = \sigma.(t',a)_{\Sigma_u} \). Thus, \( P(\sigma,(t',a)) \) holds if \( a \in \Sigma_u \).

- Case 2: \( a \notin \Sigma_u \). Then, \( (\sigma.(t',a))_{\Sigma_u} = \sigma_{\Sigma_u} = \sigma.b \), and \( \Pi_{\Sigma}((\sigma,(t',a)))_{\Sigma_u} = \Pi_{\Sigma}(\sigma_{\Sigma_u}) = \Pi_{\Sigma}(\text{obs}(\sigma,b)) \). Thus, \( G(\text{reach}(\sigma.b),\Pi_{\Sigma}(\sigma.b)) = \emptyset \). This means that \( t < t_2 \), where \( t_2 \) is defined in Definition 16, and thus \( \sigma\epsilon = \epsilon \). Since \( \sigma\epsilon = \sigma.b \), \( \text{obs}(\sigma\epsilon,t) = \sigma.b \). Let us consider \( \sigma.b = (\sigma.(t',a))_{\Sigma_u} \), and \( \sigma\epsilon = \sigma.b = \sigma.(t',a)_{\Sigma_u} \). Thus, \( P(\sigma,(t',a)) \) holds if \( a \in \Sigma_u \).

By induction, for all \( \sigma \in \text{tw}(\Sigma), \), \( P(\sigma) \) holds. Thus, for all \( \sigma \in \text{tw}(\Sigma), \) for all \( t \in \mathbb{R}_{\geq 0}, \) if \( (\sigma.b, \sigma.c) = \text{store}_{\Sigma}(\sigma,t) \) and \( (\sigma,t) \not\in \text{Pre}(\phi) \), then \( \sigma.c = \sigma_{\Sigma_u}, \sigma.b = \epsilon \), and \( \sigma\epsilon = \Pi_{\Sigma}(\sigma_{\Sigma_u}) \).

Lemma 7. \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 \( \forall a \in \Sigma_a, I(\sigma) \subseteq I(\sigma.a) \).

- Induction basis: let us consider a \( \subset \in \Sigma_a \). Then, \( I(\epsilon) = \emptyset \subseteq I(\epsilon.a) \).

- Induction step: suppose now that for \( \sigma \in \Sigma^* \), for all \( \sigma' \in \Sigma^* \) s.t. \( |\sigma'| \leq |\sigma| \), \( P(\sigma') \) holds. Let us then consider a \( \subset \in \Sigma_a \), \( \sigma' \in \Sigma_a \), and \( h, \sigma_0 \in \Sigma \times \Sigma_a \) s.t. \( h.a, \sigma_0 = \sigma.a \). Let us consider \( I(\sigma.a',) = I(h.a, \sigma_0.a') \). Then, \( I(\sigma.a',) = \text{pred}_h(S(\sigma_0.a') \cup I(\sigma_0.a')) \).

Lemma 8. \forall q \in \Sigma, \forall \sigma \in \Sigma^*, (q \in S(\sigma)) \implies (\forall u \in \Sigma_u, q \text{ after } (0,u) \in S(\sigma) \cup I(\sigma)) \).

Proof. For \( \sigma \in \Sigma^* \), let \( P(\sigma) \) be the predicate \( \forall q \in \Sigma, (q \in S(\sigma)) \implies (\forall u \in \Sigma_u, q \text{ after } (0,u) \in S(\sigma) \cup I(\sigma)) \).

- Induction basis: let us consider q \( \in S(\epsilon) \). Then, for all \( u \in \Sigma_a \), \( q \in \text{tw}(\Sigma_u) \), considering the definition of \( S(\epsilon) \), \( q \text{ after } (0,u) \in S(\epsilon) \). Thus, \( q \in S(\epsilon) \cup I(\epsilon) \).

- Induction step: let us suppose that for \( \sigma \in \Sigma^* \), P(\sigma) holds. Let us consider a \( \sigma \in \Sigma_a \), and q \( \in S(\sigma.a) \).

Theorem 8. Let \( \sigma \in \Sigma^* \), P(\sigma) holds. Thus, for all \( \sigma \in \Sigma^* \), for all \( q \in S(\sigma) \), for all \( u \in \Sigma_u \), q after \( (0,u) \in S(\sigma) \cup I(\sigma) \).

In both cases, for all \( u \in \Sigma_u \), q after \( (0,u) \in S(\sigma.a) \cup I(\sigma.a) \), meaning that P(\sigma) holds. Thus, by induction, for all \( \sigma \in \Sigma^* \), P(\sigma) holds. Thus, for all \( \sigma \in \Sigma^* \), for all \( q \in S(\sigma) \), for all \( u \in \Sigma_u \), q after \( (0,u) \in S(\sigma) \cup I(\sigma) \).
Lemma 9. For all $\sigma \in \Sigma^*_t$, for all $q \in Q$, $(q \in S(\sigma) \cup I(\sigma)) \implies (G(q, \sigma) \neq \emptyset)$.

Proof. For $\sigma \in \Sigma^*_t$, 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 $\sigma \in \Sigma^*_t$.

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

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

- If $q \in I(a)$, let us consider $(h, \sigma_0) \in \Sigma_e \times \Sigma^*_t$ 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, $G(q \text{ after } (0, h, \sigma_0)) \neq \emptyset$. Let us consider $w \in G(q \text{ after } (0, h, \sigma_0))$. Then, $(0, h).w$ satisfies $\Pi_G((0, h).w) \subseteq h.\sigma_0$, $q$ 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 $(w, t) \in S(\Pi_G((0, h).w) = S((h.\sigma_0))).$ Thus, $(0, h).w \in G(q, h.\sigma_0) = G(q, a)$. Thus, $G(q, a) \neq \emptyset$.

- 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 $G(q, \sigma) \subseteq G(q, a)$, it follows that $G(q, a) \neq \emptyset$.
  
  - 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 $i \in I(\sigma)$ and $\delta \in \mathbb{R}_{\geq 0}$ s.t. $q$ after $(\epsilon, \delta) = i$, and for all $t \leq \delta$, $q$ after $(\epsilon, t) \in X \subseteq S(\sigma)$, since $i \in I(\sigma)$, we showed previously that $G(i, \sigma) \neq \emptyset$. Let us consider $w \in G(q \text{ after } (0, h, \sigma_0))$. Then, $(w + \delta).w$ satisfies $\Pi_G((w + \delta).w) \subseteq \sigma.a$, $q$ after $(w + \delta).w = q$ after $(w + \delta)$ after $(w, t) \in F_G$, and for all $t \in \mathbb{R}_{\geq 0}$, if $t < \delta$, then $q$ after $(w + \delta, t) = q$ after $(\epsilon, t) \in X \subseteq S(\sigma)$, otherwise, $q$ after $(w + \delta, t) = i$ after $(w, t - \delta) \in S(\sigma)$. Thus, $w + \delta \in G(q, a)$. Thus, $G(q, a) \neq \emptyset$.
    
    - Otherwise, $q \in Y$, and then $\epsilon$ satisfies $\Pi_G(\epsilon) \subseteq \sigma.a$, $q$ after $(0, h, \sigma_0) \in F_G$, and for all $t \in \mathbb{R}_{\geq 0}$, $q$ after $(\epsilon, t) \in up(q) \subseteq up(\epsilon) = Y \subseteq S(\sigma)$. Thus, $\epsilon \in G(q, a)$. Thus, $G(q, a) \neq \emptyset$.

Thus, for all $q \in S(\sigma) \cup I(\sigma)$, $G(q, a) \neq \emptyset$. Thus, $P(\sigma)$ holds. By induction on $\sigma$, $P(\sigma)$ holds for every $\sigma \in \Sigma^*_t$, meaning that for all $\sigma \in \Sigma^*_t$, 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 $\text{Pre}(\varphi)$ as per Definition 12.

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


\[ \kappa_{\varphi}(q, w) = \min_{\text{lex}} \{ \max_{\varphi}(G(q, w) \cup \{\epsilon\}) \}, \text{ for } q \in Q \text{ and } w \in \Sigma_e^* \]

\[ \text{buffer}_{\varphi} = \Pi_G(\text{nobs}(\sigma, t')) \cdot \sigma, \]

\[ t_1 = \min \{ t'' \in R_{\geq 0} \mid t'' \geq t' \wedge \min \{ G(\text{Reach}(\sigma, t', \epsilon), \text{buffer}_{\varphi}) \neq \emptyset \} \cup \{ +\infty \} \} \]

\[ \delta' \wedge \min \{ \text{buffer}_{\varphi} \} \]

\[ t_1 = \min \{ t'' \in R_{\geq 0} \mid t'' \geq t' \wedge \min \{ G(\text{Reach}(\sigma, t', \epsilon), \text{buffer}_{\varphi}) \neq \emptyset \} \cup \{ +\infty \} \} \]

\[ \sigma''_{\varphi} = \min \{ \text{buffer}_{\varphi} \} \]
For $\sigma \in \text{tw}(\Sigma)$, and $t \geq \text{time}(\sigma)$, let $P(\sigma, t)$ be the predicate \((\sigma \in \text{Pre}(\varphi, t) \land (\sigma_x, \sigma_a, \sigma_e) = \text{store}_e(\sigma, t)) \implies (E_{\varphi}(\sigma) = \varphi \land \text{nobs}(\sigma(t), t) \rightarrow t \in \text{G}(\text{Reach}(\sigma_x, t), \Pi_{\text{G}}(\text{nobs}(\sigma(t), t))).)\). Let also $P(\sigma)$ be the predicate: \(\forall t \geq \text{time}(\sigma), P(\sigma, t)\). 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:** $\sigma \in \text{Pre}(\varphi, t)$. Then, $P(\epsilon)$ trivially holds.

**Case 2:** $\epsilon \in \text{Pre}(\varphi, t)$. Then, there exists $t' \leq t$ s.t. $\text{G}(\text{obs}(\epsilon, t'), t'), \epsilon \not\in \emptyset$, meaning that $\text{G}(\text{Reach}(\epsilon, t'), t), \epsilon \not\in \emptyset$, and $\text{Reach}(\epsilon) \in F_{\Sigma}$. Since $E_{\epsilon}(\epsilon) = \epsilon$, and $\text{Reach}(\epsilon) \in F_{\Sigma}, E_{\epsilon}(\epsilon) = \varphi$. Thus, because $\text{store}_e(\epsilon) = (\epsilon, t), P(\epsilon, t)$ holds.

Thus, in both cases, $P(\epsilon, t)$ holds, meaning that $P(\epsilon)$ 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 $(\sigma_x, \sigma_a, \sigma_e) = \text{store}_e(\sigma(t'), t)$, and $(\sigma_x, \sigma_a, \sigma_e) = \text{store}_e(\sigma(t', a), t)$.

**Case 1:** $(t', a) \not\in \text{Pre}(\varphi, t)$. Then, $P(\sigma(t', a), t)$ trivially holds.

**Case 2:** $(t', a) \in \text{Pre}(\varphi, t) \land a \not\in \text{Pre}(\varphi, t')$. Then, $\sigma \not\in \text{Pre}(\varphi, t')$, thus, following lemma 6, $\sigma \in \text{tw}_{\Sigma}, \sigma_a = \epsilon$, and $\sigma_e = \text{Pre}(\sigma_x, t), \sigma_e = \emptyset$. Since $(\sigma_x, t) \in \text{Pre}(\varphi, t)$, and $\sigma \not\in \text{Pre}(\varphi, t')$, there exists $t'' \in \mathbb{R}_{\geq 0}$ s.t. $t' \leq t'' \leq t$, and $\text{G}(\text{obs}(\sigma(t', a), t''), t''), \Pi_{\text{G}}(\text{obs}(\sigma(t', a), t''), t'')) \not\in \emptyset$. Since $t'' \geq t' = \text{time}(\sigma(t', a), t'')$, then $\text{obs}(\sigma(t', a), t'') = \sigma(t', a), t'')$. This means that $\text{G}(\text{Reach}(\sigma(t', a)), t''), \Pi_{\text{G}}(\sigma(t', a), t''), t'') \not\in \emptyset$.

If $a \in \Sigma_a$, then considering that $(\sigma_x, t'))_{\Sigma_{\varnothing}} = \sigma_{\Sigma_a}(t', a) = \sigma_x(t', a), \sigma_b = \epsilon$, and $\sigma_e = \text{Pre}(\sigma_x, t), \sigma_e = \emptyset$. Thus, $t_1 \leq t' \leq t$, meaning that $\text{nobs}(\sigma(t), t), t_1, t \in \text{G}(\sigma_x(t', a), t', t), \Pi_{\text{G}}(\text{nobs}(\sigma(t), t), t) \not\in \emptyset$. Thus, considering the definition of $G$, it follows that $\text{nobs}(\sigma(t), t) \rightarrow t \in \text{G}(\text{Reach}(\sigma_x(t', a), t, t), t), t \in \text{G}(\text{reach}(\sigma_x(t', a), t, t), t) \not\in \emptyset$.

If $a \in \Sigma_a$, then $\text{store}_e(\sigma(t', a)) \in \text{tw}_{\Sigma_a} = \Sigma_{\varnothing}$, thus $\Pi_{\text{G}}(\text{obs}(\sigma(t), t), t) \not\in \emptyset$. Thus, $\Pi_{\text{G}}(\sigma(t), t), t) \not\in \emptyset$. Thus, $P(\sigma(t', a), t)$ holds.

**Otherwise,** $(t', a) \in \Sigma_a$. Then, $(\sigma_x, t'))_{\Sigma_{\varnothing}} = \sigma_{\Sigma_a}(t', a) = \sigma_x(t', a), \sigma_b = \epsilon$, and $\sigma_e = \text{Pre}(\sigma_x, t'), \Pi_{\text{G}}(\sigma(t), t'), \sigma_e = \emptyset$. Thus, $t_2 \leq t' \leq t$, therefore $\text{nobs}(\sigma(t), t) \rightarrow t \in \text{G}(\sigma_x(t', a), t, t), t \in \text{G}(\text{reach}(\sigma_x(t', a), t, t), t) \not\in \emptyset$. Thus, $\Pi_{\text{G}}(\sigma(t), t'), t) \not\in \emptyset$. Thus, $\Pi_{\text{G}}(\sigma(t), t), t) \not\in \emptyset$. Thus, $\Pi_{\text{G}}(\sigma(t), t), t) \not\in \emptyset$. Thus, $P(\sigma(t', a), t)$ holds.

Thus, if $\sigma(t', a) \in \text{Pre}(\varphi, t) \land \sigma \neq \text{Pre}(\varphi, t')$. Then, consider $\omega_{t'} = \text{nobs}(\sigma(t', a), t')$. By the induction hypothesis, since $\sigma \in \text{Pre}(\varphi, t')$, $E_{\varphi}(\sigma) = \varphi$, and $\omega_{t'} \in \text{G}(\text{reach}(\sigma(t', a), t'), t'), \Pi_{\text{G}}(\sigma(t), t'), \sigma_e = \emptyset$.

If $a \in \Sigma_a$, then since $\omega_{t'} \in \text{G}(\text{reach}(\sigma(t', a), t'), \Pi_{\text{G}}(\sigma(t), t'), \sigma_e = \emptyset), \text{reach}(\sigma(t', a) \rightarrow (a, 0) = \text{Reach}(\sigma(t', a), a)) = \text{S}(\Pi_{\text{G}}(\text{nobs}(\sigma(t'), a))),$ Thus, following lemma 8, since $a \in \Sigma_a, \text{reach}(\sigma(t', a) \rightarrow (0, a) = \text{Reach}(\sigma(t', a)), a)) \in \Sigma(\Pi_{\text{G}}(\text{nobs}(\sigma(t'), a), \sigma_e = \emptyset).$ Thus, following the definition of $G$, $\text{nobs}(\sigma(t), t), t) \rightarrow t \in \text{G}(\text{reach}(\sigma(t', a), t), t), t \in \text{G}(\text{reach}(\sigma(t', a), t), t) \not\in \emptyset$. Thus, $\Pi_{\text{G}}(\sigma(t), t), t) \not\in \emptyset$. Thus, $\Pi_{\text{G}}(\sigma(t), t), t) \not\in \emptyset$. Thus, $\Pi_{\text{G}}(\sigma(t), t), t) \not\in \emptyset$. Thus, $P(\sigma(t', a), t)$ holds.
Proof. For $\sigma \in tw(\Sigma)$, let $P(\sigma)$ be the predicate: “$\forall t \geq time(\sigma), (\sigma_s, \sigma_a) = store_\varphi(\sigma, t) \implies \sigma_{|\Sigma_u} = \sigma_{|\Sigma_u} \land P(\sigma_{|\Sigma_u} \setminus \text{obs}(\sigma, t)) \land \sigma_e = P(\sigma_{|\Sigma_u}) \land \sigma_{|\Sigma_u} \equiv_2 \sigma_{|\Sigma_u}$”. Let us prove by induction that for all $\sigma \in tw(\Sigma)$, $P(\sigma)$ holds.

—Induction basis: for $\sigma = \epsilon$, $store_\varphi(\epsilon) = (\epsilon, \epsilon, \epsilon)$, and $\epsilon_{|\Sigma_u} = \epsilon_{|\Sigma_u} = P(\epsilon) = \epsilon$. 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. $\sigma, (t', a) \in tw(\Sigma)$, $t \geq time(\sigma)$, $(\sigma_s, \sigma_a) = store_\varphi(\sigma, t')$, and $(\sigma_s, \sigma_a, \sigma_e) = store_\varphi((\sigma_s, t'), a)$, and $\sigma_{|\Sigma_u} \equiv_2 \sigma_{|\Sigma_u}$.

- Case $a \in \Sigma_u$. By construction, $\sigma_s$ satisfies $P(\sigma_s) \equiv P(\sigma_{|\Sigma_u} \setminus \text{obs}(\sigma, t')) \land \sigma_e = P(\sigma_{|\Sigma_u}) \land \sigma_{|\Sigma_u} \equiv_2 \sigma_{|\Sigma_u}$, and $\sigma_{|\Sigma_u} \not\equiv_2 \sigma_{|\Sigma_u}$.

- Case $a \not\in \Sigma_u$. Since $a \in \Sigma_u$, $\sigma_{|\Sigma_u} = (\sigma_s, (t', a) \cdot \text{obs}(\sigma, t))_{|\Sigma_u}$, $\sigma_e \in tw(\Sigma_u)$, thus $\sigma_{|\Sigma_u} = \sigma_{|\Sigma_u} \cdot (t', a) = (\sigma_s, (t', a))_{|\Sigma_u}$.

- Following the same reasoning as in the case where $a \in \Sigma_u$.

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

- Case $a \not\in \Sigma_u$. By construction, $\sigma_s$ satisfies $P(\sigma_s) \equiv P(\sigma_{|\Sigma_u} \cdot \sigma_s, a)$, and $\sigma_{|\Sigma_u} \not\equiv_2 \epsilon \implies date(\sigma_{|\Sigma_u}) \geq t'$.

- Following the same reasoning as in the case where $a \in \Sigma_u$.

Moreover, $\sigma_{|\Sigma_u} = (\sigma_s, (t', a))_{|\Sigma_u}$. Thus $\sigma_{|\Sigma_u} \equiv_2 (\sigma_{|\Sigma_u})_{|\Sigma_u}$. Moreover, $\sigma_{|\Sigma_u} \equiv_2 (\sigma_{|\Sigma_u})_{|\Sigma_u}$.

This proves $P(\sigma_{|\Sigma_u}) \implies P(\sigma_{|\Sigma_u} \setminus \text{obs}(\sigma, t')) \land \sigma_e = P(\sigma_{|\Sigma_u} \setminus \text{obs}(\sigma, t')) \land \sigma_{|\Sigma_u} \equiv_2 \sigma_{|\Sigma_u}$.

Thus, $E_\varphi(\sigma) \equiv \varphi$. But $E_\varphi(\sigma)$ is sound in $Pre(\varphi)$.

□
Optimal Enforcement of (Timed) Properties with Uncontrollable Events

Thus the contrapositive of the proposition, that is that for all \(i \in \{1, |\sigma_d|\}, \text{ date}(\sigma_d(i)) \geq t' \). Thus, for all \(i \in \{1, |\sigma_d|\}, \text{ date}(\sigma_d(i)) \geq t' \). This still holds if \(\sigma_d = \epsilon \), because then \(\{1, |\sigma_d|\} = \emptyset \). Since \(\sigma_{\downarrow|\Sigma|} \trianglerighteq \sigma_{\uparrow|\Sigma|} \), for all \(i \in \{1, |\sigma_{\downarrow|\Sigma|}|, \text{ date}(\sigma_{\downarrow|\Sigma|}(i)) \geq \text{ date}(\sigma_{\uparrow|\Sigma|}(i)) \). Thus, for all \(i \in \{1, |\sigma_{\downarrow|\Sigma|}|, \text{ date}(\sigma_{\downarrow|\Sigma|}(i)) \geq \text{ date}(\sigma_{\uparrow|\Sigma|}(i)) \). Since \(\Pi_2(\sigma_{\downarrow|\Sigma|} \cdot \text{ nob}(\sigma_d, t)). s_e = \Pi_2((\sigma(t', a))_{|\Sigma|}) \), 
\(\Pi_2(\sigma_{\uparrow|\Sigma|} \trianglerighteq \Pi_2((\sigma(t', a))_{|\Sigma|}) \). Thus \(\sigma_{\downarrow|\Sigma|} \trianglerighteq \sigma_2((\sigma(t', a))_{|\Sigma|}) \).

Thus if \(a \in \Sigma_c, P(\sigma, (t, a)) \) holds.

Thus \(P(\sigma) \implies P(\sigma_{\downarrow|\Sigma|}, (t, a)) \). By induction, for all \(\sigma \in \text{ tw}(\Sigma) \), for all \(t \geq \text{ time}(\sigma), (\sigma_\downarrow|\Sigma|, \sigma_\uparrow|\Sigma|) = \text{ store}_\epsilon((\sigma, t)) \implies (\sigma_{\downarrow|\Sigma|}, \sigma_{\uparrow|\Sigma|} \bigcup \text{ nob}(\sigma_d, t)). s_e = \Pi_2((\sigma_{\downarrow|\Sigma|}) \bigcup \sigma_{\uparrow|\Sigma|} \trianglerighteq \sigma_{\downarrow|\Sigma|}) \). Thus \(P_{s_e} \) is compliant.

**Lemma 10.** \(\forall \sigma \in \Sigma_c^\ast, \forall q \in Q, (q \notin S(\sigma)) \implies (\exists \sigma_u \in \text{ tw}(\Sigma_u), (q \text{ after } \sigma_u \notin F_t) \land (\forall t > 0, q \text{ after } (\sigma_u, t) \notin S(\sigma) \cup I(\sigma)) \land (\forall \sigma_u' \neq \epsilon, \sigma_u' \neq \epsilon \implies q \text{ after } \sigma_u' \notin S(\epsilon) \cup I(\sigma)) \).

**Proof.** For \(\sigma \in \Sigma_c^\ast \) and \(q \in Q \), let \(P(\sigma, q) \) be the predicate \( "(\sigma u \in \text{ tw}(\Sigma_u), (q \text{ after } \sigma_u \in F_t) \lor (\exists t > 0, q \text{ after } (\sigma_u, t) \in S(\sigma) \cup I(\sigma)) \lor (\exists \sigma_u' \neq \epsilon, \sigma_u' \neq \epsilon \text{ and } q \text{ after } \sigma_u' \notin S(\epsilon) \cup I(\epsilon)) \) \). Let us show the contrapositive of the proposition, that is that for all \(\sigma \in \Sigma_c^\ast \), for all \(q \in Q \), \(P(\sigma, q) \implies (q \in S(\sigma)) \).

—If \(\sigma = \epsilon \), let us consider \(q \in Q \) s.t. \(P(\epsilon, q) \) holds. Then, since \(\epsilon \in \text{ tw}(\Sigma_u) \), \(q \text{ after } \epsilon = q \in F_t \), or there exists \(t > 0 \) s.t. \(q \text{ after } (\epsilon, t) \in S(\epsilon) \cup I(\epsilon) \), or there exists \(\sigma_u' \neq \epsilon \text{ s.t. } \sigma_u' \neq \epsilon \text{ and } q \text{ after } \sigma_u' \notin S(\epsilon) \cup I(\epsilon) \). Since \(\sigma_u' \neq \epsilon \text{ and } \sigma_u' = \epsilon \), meaning that this last condition does not hold for \(\sigma_u = \epsilon \). Thus, \(q \in F_t \) or there exists \(t \in \mathbb{R}_{\geq 0} \) s.t. \(q \text{ after } (\epsilon, t) \in S(\epsilon) \cup I(\epsilon) \). Since \(\epsilon = \emptyset \) and \(S(\epsilon) \subseteq F_t \), if the second condition holds, then \(q \text{ after } (\epsilon, t) \in F_t \), meaning that \(q \in F_t \). Thus, \(q \in F_t \).

Moreover, since \(P(\epsilon, q) \) holds, for all \(\sigma_u \in \text{ tw}(\Sigma_u) \), \(q \text{ after } \sigma_u \in F_t \) or there exists \(t > 0 \) s.t. \(q \text{ after } (\sigma_u, t) \in S(\epsilon) \cup I(\epsilon) \), meaning that \(q \text{ after } (\sigma_u, t) \in F_t \), or there exists \(\sigma_u' \neq \sigma_u \text{ s.t. } q \text{ after } \sigma_u' \notin S(\epsilon) \cup I(\epsilon) \). If the last condition holds, since \(\epsilon = \emptyset \), then \(q \text{ after } \sigma_u' \notin S(\epsilon) \). Then, following the definition of \(S(\sigma) \), since \(\sigma_u' \neq \sigma_u \text{ s.t. } q \text{ after } \sigma_u' \notin S(\epsilon) \cup I(\epsilon) \), this follows that \(q \text{ after } \sigma_u' \notin S(\sigma_u) \). Thus, for all \(\sigma_u \in \text{ tw}(\Sigma_u) \), \(q \text{ after } \sigma_u \in F_t \), meaning that \(q \in S(\sigma) \).

—If \(\sigma \neq \epsilon \), there exists \(\sigma' \text{ s.t. } (\sigma', a) \in \Sigma_c^\ast \times \Sigma_c^\ast \text{ s.t. } \sigma' = \sigma' \). Let us consider \(q \in Q \) s.t. \(P(\sigma, q) \) holds. Then, for all \(\sigma_u \in \text{ tw}(\Sigma_u) \), \(q \text{ after } \sigma_u \in F_t \) or there exists \(t > 0 \) s.t. \(q \text{ after } (\sigma_u, t) \in S(\sigma) \cup I(\sigma) \), or there exists \(\sigma_u' \neq \sigma_u \text{ s.t. } \sigma_u' \neq \epsilon \) and \(q \text{ after } \sigma_u' \notin S(\sigma) \cup I(\sigma) \). Let \(X_a \text{ and } Y_a \text{ be s.t. } S(\sigma) = S(\sigma') = S(\sigma') \cup X_a \cup Y_a \text{ with:} \)

\[ \forall a \in X_a, \exists a \in I(\sigma'), \exists \delta \in \mathbb{R}_{> 0}, x \text{ after } (a, \delta) = i \land \forall t \leq \delta, x \text{ after } (a, t) \in X_a, \]
\[ Y_a \subseteq F_t \cup \text{ up}(Y_a) = Y_a, \text{ and} \]
\[ (X_a \cup Y_a) \cap \text{ upPred}(X_a \cup Y_a \cup I(\sigma')) = \emptyset. \]

\(X_a \text{ and } Y_a \text{ correspond to the sets } X \text{ and } Y \text{ in the definition of } S(\sigma' | a) \text{, respectively. Let us consider} \)
\(X_0 = \{q \text{ after } (\sigma, t) \mid \sigma_u \in \text{ tw}(\Sigma_u) \land t \in \mathbb{R}_{> 0} \land \forall t' \in [0, t] \land q \text{ after } (\sigma_u, t') \notin S(\sigma) \cup I(\sigma) \land \forall \sigma_u' \neq \sigma_u \text{ s.t. } q \text{ after } (\sigma_u', t') \notin S(\sigma) \cup I(\sigma) \}, \)
\(Y_0 = \{q \text{ after } (\sigma, t) \mid \sigma_u \in \text{ tw}(\Sigma_u) \land t \in \mathbb{R}_{> 0} \land \forall t' \in [0, t] \land q \text{ after } (\sigma_u, t') \notin S(\sigma) \cup I(\sigma) \land \forall \sigma_u' \neq \sigma_u \text{ s.t. } q \text{ after } (\sigma_u', t') \notin S(\sigma) \cup I(\sigma) \}, \)
\(Y_0 \subseteq X_0 \text{, and upPred}(X_0 \cup Y_0 \cup I(\sigma')) = \emptyset. \)
and for all $t < \delta + \delta'$, $x$ after $(\epsilon, t) \in X$. Moreover, $(X \cup Y) \cap \text{uPred}(X \cup Y \cup \overline{\text{I}(\sigma)}) = \emptyset$ 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 $(\epsilon, 0)$, with $x \in tw(\Sigma_0)$ and $t \in \mathbb{R}_{\geq 0}$, $q \in X_0$, and thus $q \in S(\sigma)$. Thus, if $\sigma \neq \epsilon$ and $q \in Q$, $P(\sigma, q) \implies q \in S(\sigma)$.

Thus, for all $\sigma \in \Sigma^*_s$, for all $q \in Q$, $P(\sigma, q)$ $\implies q \in S(\sigma)$. Thus, the contrapositive also holds, meaning that for all $\sigma \in \Sigma^*_s$, for all $q \in Q$, $q \notin S(\sigma)$ $\implies \neg P(\sigma, q)$, that is $q \notin S(\sigma)$ $\implies (\exists \sigma_u \in tw(\Sigma_0), q_{\sigma_u} \notin F_G \land \forall t > 0, q_{\sigma_u} \notin S(\sigma) \cup I(\sigma) \land \forall \sigma'_u \neq \epsilon \implies q_{\sigma'_u} \notin S(\sigma) \cup I(\sigma))$.

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

**Proof.** Let us consider $E' : tw(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow tw(\Sigma)$, that is compliant with respect to $\Sigma_0$ and $\Sigma_u$.

Let us also consider $\sigma' \in tw(\Sigma)$, and $(t', a)$ s.t. $\sigma(t', a) \in tw(\Sigma)$. Suppose that $(\sigma, t') \in \text{Pre}(\varphi)$, $E'(\sigma, t') = E_{\varphi}(\sigma, t')$, and that $\underline{E}_{\varphi}(\sigma(t', a)) \sim_{\text{d}} E'(\sigma(t', a))$. Consider $(\sigma_s, \sigma_u, \sigma_c) = \text{store}_c(\sigma(t', a))$, and $(\sigma_c, \sigma_u, \sigma_c) = \text{store}_u(\sigma(t', a), t)$, where $t$ is s.t. $\sigma_t = E_{\varphi}(\sigma(t', a))$. Then, considering proof of soundness, since $(\sigma, t') \in \text{Pre}(\varphi)$, $\text{nofs}(\sigma(t', a)) \sim_{\text{t}} t' \in \text{G}(\text{Reach}(\sigma_s, t'), \Pi_{\Sigma}(\text{nofs}(\sigma(t', a)), \sigma_c))$.

—If $a \in \Sigma_u$, this means that $\sigma_u \sim_{\text{t}} t' \in \text{G}(\text{Reach}(\sigma_u(t', a)), \Pi_{\Sigma}(\text{nofs}(\sigma(t', a)), \sigma_c))$. Let us consider $q = \text{Reach}(\sigma_u(t', a))$, and $\text{buff}_u = \Pi_{\Sigma}(\text{nofs}(\sigma_u(t', a)), \sigma_c)$. Then, $\sigma_d \sim_{\text{t}} t' = \min_{\text{max}}(\text{G}(q, \text{buff}_u))$. $E'$ is compliant with respect to $\Sigma_u$ and $\Sigma_0$, thus, since $E_{\varphi}(\sigma(t', a)) = E'(\sigma(t', a))$, there exists $\sigma'_c \in tw(\Sigma)$ s.t. $E'(\sigma(t', a)) = E'(\sigma(t', a)), \sigma_c$. Since $E_{\varphi}(\sigma(t', a)) \sim_{\text{d}} E'(\sigma(t', a))$, then $\sigma_d \sim_{\text{d}} \sigma'_c$, thus $\sigma_u \sim_{\text{t}} t' \sim_{\text{d}} \sigma'_c \sim_{\text{t}} t' = \text{w}_2(u)$, meaning that $\text{w}_2 \notin \text{G}(q, \text{buff}_u)$. Then, following the definitions of $G$ and $S$, there are some cases:

$- \Pi_{\Sigma}(\text{w}_2(u)) \notin \text{buff}_u$. But, since $E'$ is compliant, and $E'(\sigma) = E_{\varphi}(\sigma)$, this is not possible.

$-q \text{ after } \text{w}_2(u) \notin F_G$, meaning that $E'(\sigma(t', a)) \neq \varphi$.

$-\text{There exists } t'' \in \mathbb{R}_{\geq 0} \text{ s.t. } q \text{ after } (\text{w}_2(u), t'') \notin S(\Pi_{\Sigma}(\text{obs}(\text{w}_2(u), t''))^{-1} \text{buff}_u)$. Let us then note $\text{buff}_u = \Pi_{\Sigma}(\text{obs}(\text{w}_2(u), t''))^{-1} \text{buff}_u$. Then, following lemma 10, there exists $\sigma_u \in tw(\Sigma_u)$ s.t. $q_{\sigma_u} \notin F_G$, for all $t > 0$, $q_{\sigma_u} \notin S(\text{buff}_u) \cup I(\text{buff}_u)$, and for all $\sigma'_c \neq \sigma_u$, $\sigma_u \neq \epsilon \implies q_{\sigma'_c} \notin S(\text{buff}_u) \cup I(\text{buff}_u)$. Then, considering that $E'$ is compliant, either $E'(\sigma(t', a), \sigma_u + t'')) = \sigma_u(t', a), \text{obs}(\text{w}_2(u), t', t'') \cdot (\sigma_u + t'')$, meaning that $E'(\sigma(t', a), \sigma_u) \neq \varphi$, or there exists $\sigma'_{u} \neq \sigma_u, q_{\sigma'_{u}} \neq \epsilon$ such that $\Pi_{\Sigma}(\text{w}_2(u)) \neq S(\Pi_{\Sigma}(\text{w}_2(u))) \cup I(\text{buff}_u)$. Considering the definition of I, $q_{\sigma_u} \neq \epsilon$ after $\text{w}_2(u) \notin S(\Pi_{\Sigma}(\text{w}_2(u)))^{-1} \text{buff}_u \cup I(\Pi_{\Sigma}(\text{w}_2(u)))^{-1} \text{buff}_u$, because otherwise $q_{\sigma_u} \neq S(\Pi_{\Sigma}(\text{w}_2(u)))^{-1} \text{buff}_u \cup I(\Pi_{\Sigma}(\text{w}_2(u)))^{-1} \text{buff}_u) = I(\text{buff}_u)$, which is wrong. In the same way, $q_{\sigma_u} \neq \epsilon$ after $\text{w}_2(u) < S(\Pi_{\Sigma}(\text{w}_2(u)))^{-1} \text{buff}_u \cup I(\Pi_{\Sigma}(\text{w}_2(u)))^{-1} \text{buff}_u)$, which is also wrong. Since it is not in $S$, we can find again a word in $tw(\Sigma_u)$ 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 will never reach $S$ nor I, and since $E'$ can only output a limited number of controllable events (no more than $|\text{buff}_u|$, 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 $\sigma_{\text{ag}} \in tw(\Sigma_u)$ s.t. $E'(\sigma(t', a), \sigma_{\text{ag}}) \neq \varphi$.

Thus, if $a \in \Sigma_u$, there exists $\sigma_u \in tw(\Sigma_u)$ such that $E'(\sigma(t', a), \sigma_u) \neq \varphi$.

—If $a \in \Sigma_u$, then since $(\sigma, t') \in \text{Pre}(\varphi)$, $\sigma_u \sim_{\text{t}} t' \in \text{G}(\text{Reach}(\sigma_u, t'), \Pi_{\Sigma}(\text{nofs}(\sigma_u(t', a)), \sigma_c, a))$. Considering $q = \text{Reach}(\sigma)$ and $\text{buff}_u = \Pi_{\Sigma}(\text{nofs}(\sigma_u(t', a)), \sigma_c, a)$, the previous proof (when $a \in \Sigma_u$) still holds. Thus, if $a \in \Sigma_u$, there also exists $\sigma_u \in tw(\Sigma_u)$ s.t. $E'(\sigma(t', a), \sigma_u) \neq \varphi$.

This means that whenever $E'(\sigma) = E_{\varphi}(\sigma) \land E_{\varphi}(\sigma(t', a)) \neq \text{d} E'(\sigma(t', a))$, then there exists $\sigma_u \in tw(\Sigma_u)$ s.t. $E'(\sigma(t', a), \sigma_u) \neq \varphi$. Thus, $E_{\varphi}$ is optimal.
Proposition 10. The output of $E$ for input $\sigma$ is $E_\sigma(\sigma)$.

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

$- C^E = tw(\Sigma) \times \Sigma^* \times Q \times \mathbb{R}_{\geq 0} \times \{\top, \bot\}$ is the set of configurations,
$- c_0^E = (\epsilon, \epsilon, q_0, 0, \bot) \in C^E$ is the initial configuration,
$- \Gamma^E = ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}) \times Op \times ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\})$ is the alphabet, composed of an optional input, an operation and an optional output,
$- The set of operations, to be applied in the given order, is: \{\text{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($w$) = $\Pi_3(w(1)) \Pi_3(w(2)) \ldots \Pi_3(w(|w|))$ the concatenation of all outputs from $w$. Since all configurations are not reachable from $c_0^E$, for a word $w \in \Gamma^E$, we will say that Reach($w$) = $c$ if $c_0^E \xrightarrow{\sigma^E} c$, 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:

$\text{Rules} : tw(\Sigma) \times \mathbb{R}_{\geq 0} \rightarrow \Gamma^E$

$\quad (\sigma, t) \mapsto \max_{w} \{w \in \Gamma^E | \text{input}(w) = \sigma \land \text{Reach}(w) \neq \bot \land \Pi_4(c) = t\}$

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 $t \in \mathbb{R}_{\geq 0}$, let $P(\sigma, t)$ be the predicate: \text{"$E_\sigma(\sigma, t) = \text{output}(\sigma, t) \land ((\sigma_\epsilon, \sigma_b, \sigma_c) = \text{store}_\epsilon(\text{obs}(\sigma, t), t) \land (\sigma^E, \sigma^E_c, q^E, t, b) = \text{Reach}(\text{Rules}(\sigma, t)) \implies \sigma^E_c = \text{nobs}(\sigma_b, t) \land \sigma^E_c \land q^E = \text{Reach}(\sigma_c, t) \land (b = T \implies G(q^E, \sigma^E_c) \neq 0)\""}. Let $P(\sigma)$ be the predicate \text{"$\forall t \in \mathbb{R}_{\geq 0}, P(\sigma, t) \text{ holds}\""}. Let us then prove that for all $\sigma \in tw(\Sigma)$, $P(\sigma)$ holds.

\text{--- Induction basis:} For $\epsilon$, let us consider $t \in \mathbb{R}_{\geq 0}$. Then, store$_\epsilon(\epsilon, t) = (\epsilon, \epsilon, \epsilon)$, and Reach($\epsilon$, $t$) = $\langle lo, vo + t \rangle$. 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($\epsilon$, $t$) = $\epsilon$ / delay($\epsilon$, $t$) / $\epsilon$, or there exists $t' \geq t$ s.t. Rules($\epsilon$, $t$) = $\epsilon$ / delay($\epsilon$, $t$) / $\epsilon$. Let us consider $c = \text{Reach}(\text{Rules}($$\epsilon$, $t$)), then $c = (\epsilon, \epsilon, \langle lo, vo + t, t, b \rangle)$. If rule compute appears in Rules($\epsilon$, $t$), then $b = T$, meaning that $G(q_0) \text{ after } (\epsilon, t'), \epsilon \neq 0$, and thus that $G(q_0) \text{ after } (\epsilon, t), \epsilon \neq 0$ since $t \geq t'$. Otherwise $b = \bot$. All the other values remain unchanged between the two cases. In both cases, output(Rules($\epsilon$, $t$)) = $\epsilon = E_\epsilon(\epsilon, t)$, thus, $P(\epsilon)$ holds.

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

$e = \langle \sigma^E_c, \sigma^E, q^E, t', b \rangle = \text{Reach}(\text{Rules}(\sigma(t'))), (\sigma_b, \sigma_c) = \text{store}_\epsilon(\sigma(t'), a), t), t$. If $t < t'$, then $\text{obs}(\sigma(t', a), t) = \text{obs}(\sigma, t), and P(\sigma(t', a), t)$ trivially holds, since $P(\sigma)$ holds. Thus, in the following, we consider that $t \geq t'$, so that store$_\epsilon(\text{obs}(\sigma(t', a), t), t) =$ $\text{store}_\epsilon(\sigma(t', a), t)$;

If $a \in \Sigma_a$, rule pass-uncont can be applied. Let us consider $c' = c$ after $((t', a)/ \text{pass-uncont}((t', a)))(t', a)$. Then, $c' = (\epsilon, P_2(\sigma^E_c), \sigma^E, q^E, t', \bot)$, with $q' = q^E$ after $\langle 0, a \rangle$. Then, if $t \geq t'$, where $t' = \min(t', t') \geq t' \geq t' \land G(q', a) \text{ after } (\epsilon, t', t)$, $P_2(\sigma^E_c, \sigma^E_c) \neq 0$, then rule delay($\delta^E_c - t'$) can be applied, followed by rule compute. Since $\delta^E_c = \text{Reach}(\sigma_c, t'), \sigma^E_c = \text{nobs}(\sigma_b, t'), and \sigma^E_c = \sigma_c$, by in-
duction hypothesis), then \(G(q' after(e, t'' - t')) = G(Reach(\sigma_e, t''), t''))
\(\Pi_2(\sigma_{\epsilon}^E).\sigma_{\epsilon}^E) = G(Reach(\sigma_e, (t', a), t''))\), \(\Pi_2(\text{obs}(\sigma_e, t''))\), thus \(t_2'' = t_1\), where \(t_1\) is defined in Definition 16. Thus, \(c'\) after \((\epsilon / \text{delay}(t_2'' - t') / \epsilon)\).

\(epsilon / \text{compute}(e)\) = \((\sigma_{\epsilon}^E).\sigma_{\epsilon}^E, q' after(e, t_1 - t'), (t_1, \top)\), with \(\sigma_{\epsilon}^E = \kappa_{\epsilon}(q' after(e, t_1 - t'), \Pi_2(\sigma_{\epsilon}^E).\sigma_{\epsilon}^E)
\(+ t_1 = \kappa_{\epsilon}(\text{Reach}(\sigma_e, (t', a), t_1), \Pi_2(\sigma_e).\sigma_e) + t_1 = \sigma_e\) and thus \(\sigma_{\epsilon}^E = \sigma_e\). Then, rules delay and dump can be applied until date \(t\) is reached. In the end, \(\text{Reach}(\text{Rules}(\sigma_e(t', a), t)) = c' after w, where w is composed of an alternation of rules delay and dump, thus \(\text{Reach}(\text{Rules}(\sigma_e(t', a), t))\).

Thus, by induction, for all \(t \geq t_1\), \(P(\sigma(t', a), t)\) holds. Otherwise, \(t < t_1\) and then rule dump cannot be applied, since \(\Pi_5(\epsilon) = \bot\), and rule compute also cannot be applied. Thus, the only rule that can be applied is delay, so that \(\text{Reach}(\text{Rules}(\sigma_e(t', a), t))\).

\(\Pi_2(\sigma_{\epsilon}^E).\sigma_{\epsilon}^E, q' after(\text{obs}(\sigma_{\epsilon}^E, t) - (t', t'' - t'), (t', t'\epsilon, \top))\).

Then, \(\text{output}(\text{Rules}(\sigma_e(t', a), t')) = \text{output}(\text{Rules}(\sigma_e(t'), (t', a)).\text{obs}(\sigma_{\epsilon}^E, t) = \text{obs}(\sigma_e, t) = \sigma_e\). Thus, \(\text{output}(\text{Rules}(\sigma_e(t', a), t)) = \text{output}(\text{Rules}(\sigma_e(t'), (t', a)).\text{obs}(\sigma_e, t) = \sigma_e\).

Thus, if \(t \geq t_1\), \(P(\sigma(t', a), t)\) holds. Otherwise, \(t < t_1\) and then rule dump cannot be applied, since \(\Pi_5(\epsilon) = \bot\), and rule compute also cannot be applied. Thus, the only rule that can be applied is delay, so that \(\text{Reach}(\text{Rules}(\sigma_e(t', a), t)) = \epsilon, \Pi_2(\sigma_{\epsilon}^E).\sigma_{\epsilon}^E, q' after(\epsilon, (t', t'\epsilon, \top))\).

Since \(t < t_1\), this means that \(\sigma_{\epsilon} = \epsilon\) and \(\sigma_e = \Pi_2(\sigma_e)\). \(\sigma_e\). Thus, \(\text{output}(\text{Rules}(\sigma_e(t', a), t)) = \text{output}(\text{Rules}(\sigma_e(t'), (t', a)).\text{obs}(\sigma_{\epsilon}^E, t) = \text{obs}(\sigma_e, t) = \sigma_e\).

Otherwise, \(a \in \Sigma\). Then, rule store-cont can be applied. Let us consider \(c = c\) after \(((t', a)/ \text{store-cont}(a)/\epsilon)\).

\(\Pi_2(\sigma_{\epsilon}^E).\sigma_{\epsilon}^E, q' after(\text{obs}(\sigma_{\epsilon}^E, t), t, \top) = \text{obs}(\sigma_{\epsilon}^E, t) = \text{obs}(\sigma_e, t) = \sigma_e, \text{Reach}(\sigma_e, (t', a), t, \top)\),
\(\text{output}(\text{Rules}(\sigma_e(t', a), t)) = \text{output}(\text{Rules}(\sigma_e(t'), (t', a)).\text{obs}(\sigma_{\epsilon}^E, t) = \text{obs}(\sigma_e, t) = \sigma_e\).

Thus, \(\text{output}(\text{Rules}(\sigma_e(t', a), t)) = \text{output}(\text{Rules}(\sigma_e(t'), (t', a)).\text{obs}(\sigma_{\epsilon}^E, t) = \text{obs}(\sigma_e, t) = \sigma_e\).

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