Math. Struct. in Comp. Science (2019), *vol.* 29, *pp.* 169–214. © Cambridge University Press 2017
doi:10.1017/S0960129517000123 First published online 29 May 2017

Optimal enforcement of (timed) properties with uncontrollable events

MATTHIEU RENARD[†], YLIÈS FALCONE[‡], ANTOINE ROLLET[†], THIERRY JÉRON[§] and HERVÉ MARCHAND[§]

[†]*LaBRI, Bordeaux INP, Université Bordeaux, Bordeaux, France*
Emails: matthieu.renard@labri.fr, antoine.rollet@labri.fr
[‡]*Univ. Grenoble-Alpes, Inria, Laboratoire d'Informatique de Grenoble, F-38000 Grenoble, France*
Email: Ylies.Falcone@univ-grenoble-alpes.fr
[§]*Inria Rennes Bretagne-Atlantique, Rennes, France*
Emails: thierry.jeron@inria.fr, herve.marchand@inria.fr

Received 29 April 2016; revised 14 March 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 (Falcone et al. 2013; Leucker and Schallhart 2009) 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. Basin et al. 2013; Falcone et al. 2011; Ligatti et al. 2009; Schneider 2000) 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. EMs 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. 2014a,c, 2012). That is, whenever possible, EMs buffer input events for some time and

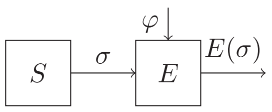

Fig. 1. Enforcement mechanism E, modifying the execution σ of the system S to $E(\sigma)$, so that it satisfies property φ.

then release them in such a way that the output sequence of events satisfies the property. The general scheme is given in Figure 1.

Motivations. We focus on enforcement of properties with uncontrollable events[†]. 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'[‡] 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 EM. 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 EM 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

† This notion of uncontrollable event should not be confused with the notion of uncontrollable transition used in some supervision and game theory.

‡ The spoiler is a device used to reduce the lift of an aircraft.

have to be considered by the EM. 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 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 EM are provided. This new framework can also be used without uncontrollable events.

Remark 1.1. There exist similarities between supervisory control theory (Ramadge and Wonham 1987, 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 EMs 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 $\Sigma^* \setminus \{\epsilon\}$. A *language* over Σ is any subset $L \subseteq \Sigma^*$. The concatenation of two words w and w' is noted

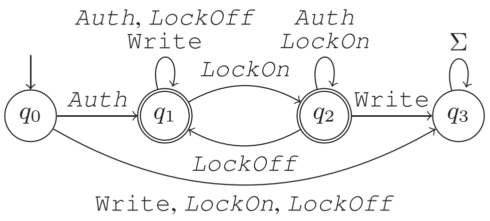

Fig. 2. Property φ_{ex} modelling writes on a shared storage device.

$w.w'$ (the dot is omitted when clear from the context). A word w' is a *prefix* of a word w, noted $w' \preccurlyeq w$ 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 $L \subseteq \Sigma^*$ is *extension-closed* if for any words $w \in L$ and $w' \in \Sigma^*$, $w.w' \in L$. Given a word w and an integer i s.t. $1 \leqslant i \leqslant |w|$, we note $w(i)$ the ith element of w. Given a tuple $e = (e_1, e_2, \ldots, e_n)$ of size n, for an integer i such that $1 \leqslant i \leqslant n$, we note Π_i the projection on the ith coordinate, i.e., $\Pi_i(e) = e_i$. The tuple $(e_1, e_2, \ldots, e_n)$ is sometimes noted $\langle e_1, e_2, \ldots, e_n \rangle$ in order to help reading. It can be used, for example, if a tuple contains a tuple. Given a word $w \in \Sigma^*$ and $\Sigma' \subseteq \Sigma$, we define the *restriction* of w to Σ', noted $w_{|\Sigma'}$, as the word $w' \in \Sigma'^*$ whose letters are the letters of w belonging to Σ' in the same order. Formally, $\epsilon_{|\Sigma'} = \epsilon$ and $\forall \sigma \in \Sigma^*, \forall a \in \Sigma$, $(w.a)_{|\Sigma'} = w_{|\Sigma'}.a$ if $a \in \Sigma'$ and $(w.a)_{|\Sigma'} = w_{|\Sigma'}$ otherwise. We also note $=_{\Sigma'}$ the equality of the restrictions of two words to Σ': if σ and σ' are two words, $\sigma =_{\Sigma'} \sigma'$ if $\sigma_{|\Sigma'} = \sigma'_{|\Sigma'}$. We define in the same way $\preccurlyeq_{\Sigma'}$: $\sigma \preccurlyeq_{\Sigma'} \sigma'$ if $\sigma_{|\Sigma'} \preccurlyeq \sigma'_{|\Sigma'}$.

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, Σ 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 $q \xrightarrow{\sigma.a} q'$ whenever there exists q'' s.t. $q \xrightarrow{\sigma} q''$ and $q'' \xrightarrow{a} q'$. Moreover, for any $q \in Q$, $q \xrightarrow{\epsilon} q$ always holds. An automaton $\mathcal{A} = \langle Q, q_0, \Sigma, \rightarrow, F \rangle$ is *deterministic* if $\forall q \in Q, \forall a \in \Sigma, (q \xrightarrow{a} q' \wedge q \xrightarrow{a} q'') \implies q' = q''$. $\mathcal{A}$ is *complete* if $\forall q \in Q, \forall a \in \Sigma, \exists q' \in Q, q \xrightarrow{a} q'$. A word w is *accepted* by $\mathcal{A}$ if there exists $q \in F$ such that $q_0 \xrightarrow{w} q$. The language (i.e., set of all words) accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$. A *property* is a language over an alphabet. A regular property is a language accepted by an automaton. In the sequel, we assume that a property φ is represented by a deterministic and complete automaton $\mathcal{A}_\varphi$. For example, in Figure 2, $Q = \{q_0, q_1, q_2, q_3\}$, the initial state is q_0, $\Sigma = \{\mathtt{Auth}, \mathtt{LockOff}, \mathtt{LockOn}, \mathtt{Write}\}$, $F = \{q_1, q_2\}$, and the transition relation $\rightarrow$ contains for instance $(q_0, \mathtt{Auth}, q_1)$, $(q_1, \mathtt{LockOn}, q_2)$ and $(q_3, \mathtt{LockOn}, q_3)$.

Timed languages. Let $\mathbb{R}_{\geqslant 0}$ be the set of non-negative real numbers, and Σ a finite alphabet of actions. An event is a pair $(t, a) \in \mathbb{R}_{\geqslant 0} \times \Sigma$. 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 Σ is a word over $\mathbb{R}_{\geqslant 0} \times \Sigma$ whose real parts are ascending, i.e., σ is a timed word if $\sigma \in (\mathbb{R}_{\geqslant 0} \times \Sigma)^*$ and $\forall i \in [1; |\sigma| - 1], \text{date}(w(i)) \leqslant \text{date}(w(i+1))$. $\text{tw}(\Sigma)$ denotes the set of timed words over Σ. For a timed word, $\sigma = (t_1, a_1).(t_2, a_2) \ldots (t_n, a_n)$ and an integer i s.t. $1 \leqslant i \leqslant n$, 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 $\mathrm{time}(\sigma) = \mathrm{date}(\sigma(|\sigma|))$ for $\sigma \neq \epsilon$ and $\mathrm{time}(\epsilon) = 0$. We define the *observation* of σ at time t as the timed word $\mathrm{obs}(\sigma, t) = \max_{\preccurlyeq}(\{\sigma' \mid \sigma' \preccurlyeq \sigma \wedge \mathrm{time}(\sigma') \leqslant t\})$, corresponding to the word that would be observed at date t if events were received at the date they are associated with. We also define the remainder of the observation of σ at time t as $\mathrm{nobs}(\sigma, t) = (\mathrm{obs}(\sigma, t))^{-1}.\sigma$, which corresponds to the events that are to be received after date t. The *untimed projection* of σ is $\Pi_\Sigma(\sigma) = a_1.a_2 \dots a_n$, it is the sequence of actions of σ with dates ignored. σ *delayed by* $t \in \mathbb{R}_{\geqslant 0}$ is the word noted $\sigma +_{\mathrm{t}} t$ s.t. t is added to all dates: $\sigma +_{\mathrm{t}} t = (t_1 + t, a_1).(t_2 + t, a_2) \dots (t_n + t, a_n)$. Similarly, we define $\sigma -_{\mathrm{t}} t$, when $t_1 \geqslant t$, to be the word $(t_1 - t, a_1).(t_2 - t, a_2) \dots (t_n - t, a_n)$. We also extend the definition of the restriction of σ to $\Sigma' \subseteq \Sigma$ to timed words, s.t. $\epsilon_{|\Sigma'} = \epsilon$, and for $\sigma \in \mathrm{tw}(\Sigma)$ and (t, a) s.t. $\sigma.(t, a) \in \mathrm{tw}(\Sigma)$, $(\sigma.(t, a))_{|\Sigma'} = \sigma_{|\Sigma'}.(t, a)$ if $a \in \Sigma'$, and $(\sigma.(t, a))_{|\Sigma'} = \sigma_{|\Sigma'}$, otherwise. The notations $=_{\Sigma'}$ and $\preccurlyeq_{\Sigma'}$ are then naturally extended to timed words. A *timed language* is any subset of $\mathrm{tw}(\Sigma)$. 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 $S \subseteq \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geqslant 0}$ is *time-extension-closed* if for any $(\sigma, t) \in S$, for all $w \in \mathrm{tw}(\Sigma)$ s.t. $\sigma.w \in \mathrm{tw}(\Sigma)$, for all $t' \geqslant t$, $(\sigma.w, t') \in S$. In other words, S is time-extension-closed if for every $\sigma \in \mathrm{tw}(\Sigma)$, there exists a date t from which σ 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 σ' is a *delayed prefix* of σ, noted $\sigma' \preccurlyeq_{\mathrm{d}} \sigma$, whenever $\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\sigma)$ and $\forall i \in [1; |\sigma'| - 1], \mathrm{date}(\sigma(i)) \leqslant \mathrm{date}(\sigma'(i))$. 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 σ' exceed dates in σ. As for the equality $=$ and the prefix order $\preccurlyeq$, we note $\sigma' \preccurlyeq_{\mathrm{d}_{\Sigma'}} \sigma$ whenever $\sigma'_{|\Sigma'} \preccurlyeq_{\mathrm{d}} \sigma_{|\Sigma'}$. We also define a *lexical order* $\leqslant_{\mathrm{lex}}$ on timed words with identical untimed projections, s.t. $\epsilon \leqslant_{\mathrm{lex}} \epsilon$, and for two words σ and σ' s.t. $\Pi_\Sigma(\sigma) = \Pi_\Sigma(\sigma')$, and two events (t, a) and (t', a), $(t', a).\sigma' \leqslant_{\mathrm{lex}} (t, a).\sigma$ if $t' < t \vee (t = t' \wedge \sigma' \leqslant_{\mathrm{lex}} \sigma)$.

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

Timed automata. Let $X = \{X_1, X_2, \dots, X_n\}$ be a finite set of *clocks*, i.e., variables that increase regularly with time. A *clock valuation* is a function ν from X to $\mathbb{R}_{\geqslant 0}$. The set of clock valuations for the set of clocks X is noted $\mathcal{V}(X)$, i.e., $\mathcal{V}(X) = \{\nu \mid \nu : X \to \mathbb{R}_{\geqslant 0}\}$. We consider the following operations on valuations: For any valuation ν, $\nu + \delta$ 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 $X_i \bowtie c$ with $X_i \in X$, $c \in \mathbb{N}$, and $\bowtie \in \{<, \leqslant, =, \geqslant, >\}$. Given $g \in \mathcal{G}(X)$ and a valuation ν, we write $\nu \models g$ when for every constraint $X_i \bowtie c$ in g, $\nu(X_i) \bowtie c$ holds.

Definition 2.1 (Timed automaton (Alur and Dill 1992)). A *timed automaton* (TA) is a tuple $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, G \rangle$, s.t. L is a set of locations, $l_0 \in L$ is the initial location, X is a set of clocks, Σ is a finite set of events, $\Delta \subseteq L \times \mathcal{G}(X) \times \Sigma \times 2^X \times L$ is the transition relation, 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 TA $\mathcal{A}$ is a timed transition system $[\![\mathcal{A}]\!] = \langle Q, q_0, \Gamma, \rightarrow, F_G\rangle$ where $Q = L \times \mathcal{V}(X)$ is the (infinite) set of states, $q_0 = (l_0, v_0)$ is the initial state, with $v_0 = v[X \leftarrow 0]$, $F_G = G \times \mathcal{V}(X)$ is the set of accepting states, $\Gamma = \mathbb{R}_{\geqslant 0} \times \Sigma$ 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 $(l, v) \xrightarrow{(\delta,a)} (l', v')$ with $v' = (v + \delta)[Y \leftarrow 0]$ whenever there is a transition $(l, g, a, Y, l') \in \Delta$ s.t. $v + \delta \models g$, for $\delta \geqslant 0$.

A TA $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, G\rangle$ is *deterministic* if for any different transitions (l, g_1, a, Y_1, l'_1) and (l, g_2, a, Y_2, l'_2) in Δ, $g_1 \wedge g_2$ is unsatisfiable, meaning that only one transition can be fired at any time. $\mathcal{A}$ is *complete* if for any $l \in L$ and any $a \in \Sigma$, the disjunction of the guards of all the transitions leaving l and labelled by a is valid (i.e., it holds for any clock valuation). An example of a TA is given in Figure 3.

A *run* ρ from $q \in Q$ is a valid sequence of transitions in $[\![\mathcal{A}]\!]$ starting from q, of the form $\rho = q \xrightarrow{(\delta_1,a_1)} q_1 \xrightarrow{(\delta_2,a_2)} q_2 \ldots \xrightarrow{(\delta_n,a_n)} q_n$. The set of runs from q_0 is noted $\mathrm{Run}(\mathcal{A})$ and $\mathrm{Run}_{F_G}(\mathcal{A})$ denotes the subset of runs accepted by $\mathcal{A}$, i.e., ending in a state in F_G. The *trace* of the run ρ previously defined is the timed word $(t_1, a_1).(t_2, a_2) \ldots (t_n, a_n)$, with, for $1 \leqslant i \leqslant n$, $t_i = \sum_{k=1}^{i} \delta_k$. Thus, given the trace $\sigma = (t_1, a_1).(t_2, a_2) \ldots (t_n, a_n)$ of a run ρ 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 $q \xrightarrow{\sigma} q'$ whenever $q \xrightarrow{w} q'$ for the previously defined w. Note that to concatenate two traces σ_1 and σ_2, it is needed to delay σ_2 to obtain a trace: the concatenation σ of σ_1 and σ_2 is the trace defined as $\sigma = \sigma_1.(\sigma_2 +_{\mathrm{t}} \mathrm{time}(\sigma_1))$. 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 $\varphi \subseteq \mathrm{tw}(\Sigma)$ accepted by a TA. For a timed word σ, we say that σ *satisfies* φ, 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 $\mathcal{A}$ s.t. Q is the set of states of $[\![\mathcal{A}]\!]$ and $\rightarrow$ its transition relation, and a word σ, for $q \in Q$, we note q after $\sigma = q'$, where q' is s.t. $q \xrightarrow{\sigma} q'$. Since $\mathcal{A}$ is complete and deterministic, there exists only one such q'. We note $\mathrm{Reach}(\sigma) = q_0$ after σ. We extend these definitions to languages: if L is a language, q after $L = \bigcup_{\sigma \in L} q$ after σ and $\mathrm{Reach}(L) = q_0$ after L. These definitions are valid both in the untimed and timed cases. For the timed case, we also allow to add an extra parameter to after and Reach, that represents an observation time. For $q \in Q$, $t \in \mathbb{R}_{\geqslant 0}$, and $\sigma \in \mathrm{tw}(\Sigma)$, q after $(\sigma, t) = (l, v + t - \mathrm{time}(\mathrm{obs}(\sigma, t)))$, where $(l, v) = q$ after $(\mathrm{obs}(\sigma, t))$, and $\mathrm{Reach}(\sigma, t) = q_0$ after (σ, 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 $(l, v) \in Q$, (l, v) after $(\epsilon, t) = (l, v + t)$ is the state reached from (l, v) by letting time elapse of t time units. Moreover, for $(l, v) \in Q$, we note $\mathrm{up}((l, v)) = \{(l, v + t) \mid t \in \mathbb{R}_{\geqslant 0}\}$. This definition is extended to sets of states: for $S \subseteq Q$, $\mathrm{up}(S) = \bigcup_{q \in S} \mathrm{up}(q)$. We also define a predecessor operator: for $q \in Q$ and $a \in \Sigma$, $\mathrm{Pred}_a(q) = \{q' \in Q \mid q'$ after $a = q\}$ for the

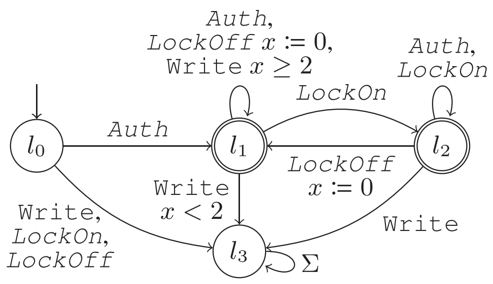

Fig. 3. Property φ_t modelling writes on a shared storage device.

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 $\text{Pred}_\sigma(q) = \{q' \in Q \mid q' \text{ after } \sigma = q\}$.

Example 2.1 (Shared data storage). Consider the property φ_t described in Figure 3 and representing writes on a shared data storage. A more detailed description of this property is given in Section 4.3. This property is similar to φ_{ex} (Figure 2), but a clock has been added to impose that writes should not occur before two time units have elapsed since the reception of the last *LockOff* event. Thus, the set of locations of φ_t 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 $\Sigma = \{\textit{Auth}, \textit{LockOn}, \textit{LockOff}, \texttt{Write}\}$, the set of accepting locations is $G = \{l_1, l_2\}$ and the set of transitions contains for instance transitions $(l_0, \top, \textit{Auth}, \emptyset, l_1)$, $(l_2, \top, \textit{Auth}, \emptyset, l_2)$ and $(l_3, \top, \textit{LockOn}, \emptyset, l_3)$, where $\top$ is the guard that holds for every clock valuation.

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

3. Enforcement monitoring of untimed properties

In this section, φ is a regular property defined by an automaton $\mathcal{A}_\varphi = \langle Q, q_0, \Sigma, \rightarrow, F \rangle$. Recall that the general scheme of an EM is given in Figure 1, where S represents the running system, σ its execution, E the EM, φ the property to enforce and $E(\sigma)$ the output of the enforcement mechanism, which should satisfy φ.

We consider uncontrollable events in the set $\Sigma_u \subseteq \Sigma$. These events cannot be modified by an EM, i.e., they cannot be suppressed nor buffered, so they must be output by the EM whenever they are received. Let us note $\Sigma_c = \Sigma \setminus \Sigma_u$ the set of controllable events, which can be modified by the EM. An EM can decide to buffer them to delay their emission, but it cannot suppress them (nevertheless, it can delay them endlessly, keeping their order unchanged).[§] Thus, an EM may interleave controllable and uncontrollable events.

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

3.1. *Enforcement functions and their requirements*

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

Definition 3.1 (Enforcement function). An *enforcement function* is a function from Σ^* to Σ^*, that is increasing on Σ^* with respect to $\preccurlyeq$: $\forall(\sigma,\sigma') \in (\Sigma^*)^2, \sigma \preccurlyeq \sigma' \implies E(\sigma) \preccurlyeq E(\sigma')$.

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 φ, thus its enforcement function has to be *sound*, meaning that its output always satisfies φ.

Definition 3.2 (Soundness). An enforcement function $E : \Sigma^* \to \Sigma^*$ is *sound* with respect to φ 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 EM. Moreover, some uncontrollable sequences could lead to a state of the property that would be a non-accepting sink state, leading to the EM not being able to satisfy the property any further. Consequently, in Definition 3.2, soundness is not defined for all words in Σ^*, but in a subset S, since it might happen that it is impossible to ensure it from the 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 φ). In practice, there may be an initial period where the EM 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. Ligatti et al. 2009; Schneider 2000) 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

[§] 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.

they are received. Therefore, transparency cannot be ensured, and we define the weaker notion of *compliance*.

Definition 3.3 (Compliance). E is *compliant* with respect to Σ_u and Σ_c, noted $\text{compliant}(E, \Sigma_u, \Sigma_c)$, if $\forall \sigma \in \Sigma^*$, $E(\sigma) \preccurlyeq_{\Sigma_c} \sigma \wedge E(\sigma) =_{\Sigma_u} \sigma \wedge \forall u \in \Sigma_u, E(\sigma).u \preccurlyeq E(\sigma.u)$.

Intuitively, compliance states that the EM does not change the order of the controllable events and emits uncontrollable events simultaneously with their reception, possibly followed by stored controllable events. We chose to consider EMs 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 $\text{compliant}(E)$.

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

In addition, an EM 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 3.4 (Optimality). An enforcement function $E : \Sigma^* \rightarrow \Sigma^*$ that is compliant with respect to Σ_u and Σ_c, and sound in an extension-closed set $S \subseteq \Sigma^*$ is *optimal* in S if

$$\forall E' : \Sigma^* \rightarrow \Sigma^*, \forall \sigma \in S, \forall a \in \Sigma,$$
$$(\text{compliant}(E') \wedge E'(\sigma) = E(\sigma) \wedge |E'(\sigma.a)| > |E(\sigma.a)|) \Rightarrow (\exists \sigma_u \in \Sigma_u^*, E'(\sigma.a.\sigma_u) \not\models \varphi).$$

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 φ. 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 cannot 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 Σ^*, as is for soundness (Definition 3.2).

Example 3.1. 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 φ_{ex} (see Figure 2) formalises the above requirement. φ_{ex} is not enforceable if the uncontrollable alphabet is $\{\mathit{LockOn}, \mathit{LockOff}, \mathit{Auth}\}$[¶] since reading the word LockOn from q_0 leads to q_3, which is not an accepting state. However, the existence of such a word does not imply that it is impossible to enforce φ_{ex} for some other input words. If word Auth is read, then

[¶] Uncontrollable events are emphasised in italics.

state q_1 is reached, and from this state, it is possible to enforce φ_{ex} by emitting `Write` only when in state q_1.

3.2. *Synthesising enforcement functions*

Example 3.1 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 φ will be satisfied (meaning that for any events received as input, the EM can output a sequence that satisfies φ), it would then be possible to define a subset of Σ^* in which an enforcement function would be sound.

To be compliant, an EM 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 EM 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 EM 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 $\mathrm{I}(\sigma)$ and $\mathrm{S}(\sigma)$, which represent the states from which the EM can output the first event of σ, and the states in which the EM can wait for another event, respectively.

Definition 3.5 (I, S). Given a sequence of controllable events $\sigma \in \Sigma_c^*$, we define the sets of states of φ, $\mathrm{I}(\sigma)$ and $\mathrm{S}(\sigma)$ by induction as follows: $\mathrm{I}(\epsilon) = \emptyset$, $\mathrm{S}(\epsilon) = \{q \in F \mid q \,\text{after}\, \Sigma_u^* \subseteq F\}$ and, for $\sigma \in \Sigma_c^*$ and $a \in \Sigma_c$,

$$\mathrm{I}(a.\sigma) = \mathrm{Pred}_a(\mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)),$$

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

Intuitively, $\mathrm{S}(\sigma)$ is the set of 'winning' states, i.e., if an EM has reached a state in $\mathrm{S}(\sigma)$ with buffer σ, 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, $\mathrm{S}(\sigma) \subseteq F$, 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. $\mathrm{S}(\sigma.a)$ is defined as the biggest subset of F such that no uncontrollable event leads outside of it or $\mathrm{I}(\sigma.a)$, meaning that whatever uncontrollable event is received from a state in $\mathrm{S}(\sigma.a)$, the state reached will be either in F (since it will be in $\mathrm{S}(\sigma.a)$) or in $\mathrm{I}(\sigma.a)$. In both cases, this means that the EM can reach an accepting state, whatever uncontrollable events are received.

$\mathrm{I}(\sigma)$ is the set of intermediate states, the states that can be 'crossed' while emitting a prefix of the buffer. The states in $\mathrm{I}(\sigma)$ 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. $\mathrm{I}(a.\sigma)$ is defined as the set of all states from which following the

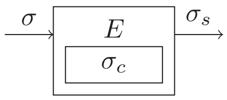

Fig. 4. Enforcement function.

transition labelled by a leads either to $\mathrm{I}(\sigma)$ or $\mathrm{S}(\sigma)$, 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 $G(q, \sigma)$, the set of words that can be emitted from a state $q \in Q$ by an EM with a buffer $\sigma \in \Sigma_c^*$.

Definition 3.6 (G). For $q \in Q$, $\sigma \in \Sigma_c^*$, $\mathrm{G}(q, \sigma) = \{w \in \Sigma_c^* \mid w \preccurlyeq \sigma \wedge q \text{ after } w \in \mathrm{S}(w^{-1}.\sigma)\}$.

Intuitively, $\mathrm{G}(q, \sigma)$ is the set of words that can be output by a compliant EM to ensure soundness from state q with buffer σ. 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 EM with its buffer.

Now, we use G to define the functional behaviour of the EM.

Definition 3.7 (Functions store_φ**,** E_φ**).** ‖ Function $\mathrm{store}_\varphi : \Sigma^* \to \Sigma^* \times \Sigma^*$ is defined as

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

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

$$\kappa_\varphi(q, w) = \max\nolimits_\preccurlyeq(\mathrm{G}(q, w) \cup \{\epsilon\}), \text{ for } q \in Q \text{ and } w \in \Sigma_c^*,$$
$$\sigma'_s = \kappa_\varphi(\mathrm{Reach}(\sigma_s.a), \sigma_c),$$
$$\sigma'_c = \sigma_s'^{-1}.\sigma_c,$$
$$\sigma''_s = \kappa_\varphi(\mathrm{Reach}(\sigma_s), \sigma_c.a),$$
$$\sigma''_c = \sigma_s''^{-1}.(\sigma_c.a).$$

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

Figure 4 gives a scheme of the behaviour of the enforcement function. Intuitively, σ_s is the word that can be released as output, whereas σ_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 EM distinguishes the following two cases:

— If a belongs to Σ_u, then it is output, as required by compliance. Then, the longest prefix of σ_c that satisfies φ and leads to a state in S for the associated buffer is also output.

‖ E_φ and store_φ depend on Σ_u and Σ_c, but we did not write it in order to lighten the notations.

— If a is in Σ_c, then it is added to σ_c, and the longest prefix of this new buffer that satisfies φ and leads to a state in S for the associated buffer is emitted, if it exists.

In both cases, κ_φ 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 EM, or ϵ if this set is empty. The parameters of κ_φ are those which are passed to G. They correspond to the state reached so far by the output of the EM, and its current buffer, respectively.

As seen in Example 3.1, 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 $\mathrm{Pre}(\varphi)$, such that E_φ is sound in $\mathrm{Pre}(\varphi)$, see Proposition 3.2:

Definition 3.8 **(Pre).** The set of input words $\mathrm{Pre}(\varphi) \subseteq \Sigma^*$ is defined as follows:

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

Intuitively, $\mathrm{Pre}(\varphi)$ is the set of words in which E_φ is sound. This set is extension closed, as required by Definition 3.2. In E_φ, using S ensures that once G is not empty, then it will never be afterwards, whatever events are received. Thus, $\mathrm{Pre}(\varphi)$ is the set of input words such that the output of E_φ would belong to G. Since E_φ outputs only uncontrollable events until G becomes non-empty, the definition of $\mathrm{Pre}(\varphi)$ considers that the state reached is the one that is reached by emitting only the uncontrollable events of σ, and the corresponding buffer would then be the controllable events of σ.

Example 3.2. Considering property φ_{ex} (Figure 2), with the uncontrollable alphabet $\Sigma_u = \{\mathit{Auth}, \mathit{LockOff}, \mathit{LockOn}\}$, $\mathrm{Pre}(\varphi_{\mathrm{ex}}) = \mathtt{Write}^*.\mathit{Auth}.\Sigma^*$. Indeed, from the initial state q_0, if an uncontrollable event, say $\mathit{LockOff}$, is received, then q_3 is reached, which is a non-accepting sink state, and is thus not in $\mathrm{S}(\epsilon)$. 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 $\mathtt{Write}$ events when in q_2 until a $\mathit{LockOff}$ event is received. Consequently, q_1 and q_2 are in $\mathrm{S}(\sigma)$ for all $\sigma \in \Sigma_c^*$, and thus $\mathrm{Pre}(\varphi_{\mathrm{ex}}) = \mathtt{Write}^*.\mathit{Auth}.\Sigma^*$, since $\mathtt{Write}$ events can be buffered while in state q_0 until event Auth is received, leading to $q_1 \in \mathrm{S}(\mathtt{Write}^*)$.

E_φ, as defined in Definition 3.7, is an enforcement function that is sound with respect to φ in $\mathrm{Pre}(\varphi)$, compliant with respect to Σ_u and Σ_c, and optimal in $\mathrm{Pre}(\varphi)$.

Proposition 3.1. E_φ *as defined in Definition 3.7 is an enforcement function.*

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

Proposition 3.2. E_φ *is sound with respect to* φ *in* $\mathrm{Pre}(\varphi)$*, as per Definition 3.2.*

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

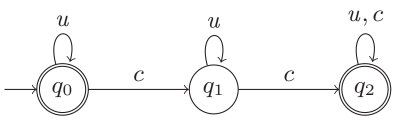

Fig. 5. Property that can be enforced by blocking all controllable events c.

— $\sigma.a \notin \mathrm{Pre}(\varphi)$. Then the proposition holds.
— $\sigma.a \in \mathrm{Pre}(\varphi)$, but $\sigma \notin \mathrm{Pre}(\varphi)$. Then the input reaches $\mathrm{Pre}(\varphi)$, and since it is extension closed, all extensions of σ also are in $\mathrm{Pre}(\varphi)$, and we prove that the proposition holds considering the definition of $\mathrm{Pre}(\varphi)$.
— $\sigma \in \mathrm{Pre}(\varphi)$ (and thus, $\sigma.a \in \mathrm{Pre}(\varphi)$ since it is extension closed). Then, we prove that the proposition holds, based on the definition of store_φ, and more precisely on the definition of S, that ensures that there always exists a compliant output that satisfies φ.

Proposition 3.3. E_φ *is compliant as per Definition 3.3.*

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 $\mathrm{store}_\varphi(\sigma.a)$, $(\sigma.a)_{|\Sigma_u}$, and $(\sigma.a)_{|\Sigma_c}$ when $a \in \Sigma_c$ and $a \in \Sigma_u$.

Remark 3.1. 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 Figure 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 $\mathrm{S}(\sigma)$ for every word $\sigma \in c^*$. This means that an EM 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 EM 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 $\sigma \in \mathrm{Pre}(\varphi)$, $\mathrm{E}_\varphi(\sigma)$ is the longest possible word that ensures soundness and compliance, that is controllable events are blocked only when necessary. Thus, E_φ is also optimal in $\mathrm{Pre}(\varphi)$:

Proposition 3.4. E_φ *is optimal in* $\mathrm{Pre}(\varphi)$*, as per Definition 3.4.*

Sketch of proof. The proof is made by induction on the input $\sigma \in \Sigma^*$. Once $\sigma \in \mathrm{Pre}(\varphi)$, we know that $\mathrm{E}_\varphi(\sigma) \models \varphi$ since E_φ is sound in $\mathrm{Pre}(\varphi)$. E_φ is optimal because in store_φ, κ_φ provides the longest possible word. If a longer word were output, then either the output would not satisfy φ, 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 EM 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

Table 1. Example of the evolution of $(\sigma_s, \sigma_c) = \text{store}_{\varphi_{\text{ex}}}(\sigma)$, with input $\mathtt{Auth.LockOn.Write.LockOff}$

σ	σ_s	σ_c
ϵ	ϵ	ϵ
Auth	*Auth*	ϵ
Auth.LockOn	*Auth.LockOn*	ϵ
Auth.LockOn.Write	*Auth.LockOn*	Write
Auth.LockOn.Write.*LockOff*	*Auth.LockOn.LockOff*.Write	ϵ

that is not in S for the updated buffer. By iterating, the buffer would become ϵ whereas the output of the EM 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_φ is optimal in $\mathrm{Pre}(\varphi)$, since it outputs the longest word that allows to be both sound and compliant.

Example 3.3. Consider property φ_{ex} (Figure 2). We illustrate in Table 1 the EM by showing the evolution of σ_s and σ_c with input $\sigma = \mathit{Auth} \,.\, \mathit{LockOn} \,.\, \mathtt{Write} \,.\, \mathit{LockOff}$.

3.3. *Enforcement monitors*

Enforcement monitors are operational descriptions of EMs. We give a representation of an EM for a property φ as an input/output transition system. The input/output behaviour of the enforcement monitor is the same as the one of the enforcement function E_φ defined in Section 3.2. Enforcement monitors are purposed to ease the implementation of EMs.

Definition 3.9 (Enforcement monitor). An *enforcement monitor* $\mathcal{E}$ for φ is a transition system $\langle C^{\mathcal{E}}, c_0^{\mathcal{E}}, \Gamma^{\mathcal{E}}, \hookrightarrow_{\mathcal{E}}\rangle$ such that

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

  – **Dump:** $\langle q, a.\sigma_c\rangle \xhookrightarrow{\epsilon/\,\text{dump}(a)/a}_{\mathcal{E}} \langle q', \sigma_c\rangle$, if $a \in \Sigma_{\text{c}}$, $\mathrm{G}(q, a.\sigma_c) \neq \emptyset$ and $\mathrm{G}(q, a.\sigma_c) \neq \{\epsilon\}$, with $q' = q$ after a,

  – **Pass-uncont:** $\langle q, \sigma_c\rangle \xhookrightarrow{a/\,\text{pass-uncont}(a)/a}_{\mathcal{E}} \langle q', \sigma_c\rangle$, with $a \in \Sigma_{\text{u}}$ and $q' = q$ after a,

  – **Store-cont:** $\langle q, \sigma_c\rangle \xhookrightarrow{a/\,\text{store-cont}(a)/\epsilon}_{\mathcal{E}} \langle q, \sigma_c.a\rangle$, with $a \in \Sigma_{\text{c}}$.

In $\mathcal{E}$, a configuration $c = \langle q, \sigma\rangle$ represents the current state of the EM. The state q is the one reached so far in $\mathcal{A}_\varphi$ with the output of the monitor. The word of controllable events σ_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

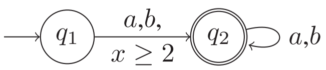

Fig. 6. A timed property enforceable only if $\Sigma_u = \emptyset$.

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 σ_c in the definition of store$_\varphi$, whereas σ_s is here represented by state q which is the first member of the configuration, such that $q = \text{Reach}(\sigma_s)$.

Proposition 3.5. *The output of the enforcement monitor $\mathcal{E}$ for input σ is* $\text{E}_\varphi(\sigma)$.

In Proposition 3.5, the output of the enforcement monitor is the concatenation of all the outputs of the word labelling the path followed when reading σ. 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 κ_φ in the definition of store$_\varphi$, and consequently the outputs of $\mathcal{E}$ and E_φ are the same.

Remark 3.2. Enforcement monitors as per Definition 3.9 are somewhat similar to the configuration description of EMs in Falcone et al. (2011). The main difference with the EMs considered in Falcone et al. (2011) is that the rule to be applied depends on the memory (the buffer), whereas in Falcone et al. (2011) it only depends on the state and the event received.

4. Enforcement monitoring of timed properties

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

Considering uncontrollable events with timed properties raises several difficulties. First, as in the untimed case, the order of events might be modified. Thus, previous definitions of transparency (Pinisetty et al. 2012), stating that the output of an enforcement function will eventually be a delayed prefix of the input, cannot be used in this situation. Moreover, when delaying some events to have the property satisfied in the future, one must consider the fact that some uncontrollable events could occur at any moment (and cannot be delayed). Finally, some properties become not enforceable because of uncontrollable events, meaning that for these properties it is impossible to obtain sound EMs, as shown in Example 4.1.

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

Example 4.1 (Non-enforceable property). Consider the property defined by the automaton in Figure 6 with alphabet $\{a, b\}$. If all actions are controllable ($\Sigma_\mathrm{u} = \emptyset$), the property is enforceable because an EM just needs to delay events until clock x exceeds 2. Otherwise, the property is not enforceable. For instance, if $\Sigma_\mathrm{u} = \{a\}$, word $(1, a)$ cannot be corrected.

4.1. *Enforcement functions and their properties*

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

Definition 4.1 (Enforcement function). Given an alphabet of actions Σ, an *enforcement function* is a function $E : \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geqslant 0} \rightarrow \mathrm{tw}(\Sigma)$ s.t.:

$$\forall \sigma \in \mathrm{tw}(\Sigma), \forall t \in \mathbb{R}_{\geqslant 0}, \forall t' \geqslant t,$$
$$E(\sigma, t) \preccurlyeq E(\sigma, t') \quad \wedge \quad (\sigma.(t, a) \in \mathrm{tw}(\Sigma) \implies E(\sigma, t) \preccurlyeq E(\sigma.(t, a), t)).$$

Definition 4.1 models physical constraints: an enforcement function cannot 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 4.2 (Soundness). An enforcement function E is *sound* with respect to φ in a time-extension-closed set $S \subseteq \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geqslant 0}$ if $\forall (\sigma, t) \in S, \exists t' \geqslant t, \forall t'' \geqslant t', E(\sigma, t'') \models \varphi$.

An enforcement function is sound in a time-extension-closed set S if for any (σ, t) in S, the value of the enforcement function with input σ from date t satisfies the property in the future. As in the untimed setting, soundness is not defined for all words in $\mathrm{tw}(\Sigma)$, 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 4.1. 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 EMs 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 4.3 (Compliance). Given an enforcement function E defined on an alphabet Σ, we say that E is *compliant* with respect to Σ_u and Σ_c, noted compliant(E, Σ_u, Σ_c), if $\forall \sigma \in \mathrm{tw}(\Sigma), \forall t \in \mathbb{R}_{\geqslant 0}, E(\sigma, t) \preccurlyeq_{\mathrm{d}\Sigma_c} \sigma \wedge E(\sigma, t) =_{\Sigma_u} \mathrm{obs}(\sigma, t) \wedge \forall u \in \Sigma_u, E(\sigma, t).(t, u) \preccurlyeq E(\sigma.(t, u), t)$.

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

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

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

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

Definition 4.4 (Optimality). We say that an enforcement function $E : \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geqslant 0} \to \mathrm{tw}(\Sigma)$ that is compliant with respect to Σ_c and Σ_u and sound in a time-extension-closed set $S \subseteq \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geqslant 0}$ is *optimal* in S if for any enforcement function $E' : \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geqslant 0} \to \mathrm{tw}(\Sigma)$, for all $\sigma \in \mathrm{tw}(\Sigma)$, for all (t, a) s.t. $\sigma.(t, a) \in \mathrm{tw}(\Sigma)$,

$$\mathrm{compliant}(E', \Sigma_u, \Sigma_c) \wedge (\sigma, t) \in S \wedge E'(\sigma, t) = E(\sigma, t) \wedge E(\sigma.(t, a)) \prec_{\mathrm{d}} E'(\sigma.(t, a))$$
$$\implies (\exists \sigma_u \in \mathrm{tw}(\Sigma_u), E'(\sigma.(t, a).\sigma_u) \not\models \varphi).$$

Optimality states that outputting a greater word (with respect to $\preccurlyeq_{\mathrm{d}}$) 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 4.4, E is an optimal enforcement function, and E' is another compliant enforcement function, that we consider having a greater output (with respect to $\preccurlyeq_{\mathrm{d}}$) 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 φ.

An EM delaying events should buffer them until it can output them. Being able to enforce φ 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 σ of controllable actions, two sets of states of the semantics of $\mathcal{A}_\varphi$, $\mathrm{S}(\sigma)$ and $\mathrm{I}(\sigma)$. $\mathrm{S}(\sigma)$ 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. $\mathrm{I}(\sigma)$ is the set of states from which it is possible to emit the first event of σ 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 4.5 (I, S). The sets of states of $[\![\mathcal{A}_\varphi]\!]$, $\mathrm{I}(\sigma)$ and $\mathrm{S}(\sigma)$ are inductively defined over sequences of controllable events as follows: $\mathrm{I}(\epsilon) = \emptyset$ and $\mathrm{S}(\epsilon) = \{q \in F_G \mid q \text{ after } \mathrm{tw}(\Sigma_\mathrm{u}) \subseteq F_G\}$ and, for $\sigma \in \Sigma_\mathrm{c}^*$ and $a \in \Sigma_\mathrm{c}$,

$$\begin{aligned}
\mathrm{I}(a.\sigma) &= \mathrm{Pred}_a(\mathrm{I}(\sigma) \cup \mathrm{S}(\sigma)),\\
\mathrm{S}(\sigma.a) &= \mathrm{S}(\sigma) \cup \max_{\subseteq}(\{X \cup Y \subseteq Q \mid Y \subseteq F_G \wedge Y = \mathrm{up}(Y) \wedge\\
&\qquad (\forall x \in X, \exists i \in \mathrm{I}(\sigma.a), \exists \delta \in \mathbb{R}_{\geqslant 0}, x \text{ after } (\epsilon, \delta) = i \wedge \forall t < \delta, x \text{ after } (\epsilon, t) \in X) \wedge\\
&\qquad (X \cup Y) \cap \mathrm{uPred}(\overline{X \cup Y \cup \mathrm{I}(\sigma.a)}) = \emptyset\})
\end{aligned}$$

Intuitively, in Definition 4.5, $\mathrm{S}(\sigma)$ is the set of states of the semantics of φ that our EM will be allowed to reach with a buffer σ. 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 $\mathrm{S}(\sigma)$, the EM can compute a word of controllable events (taken from its buffer σ) 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 $\mathrm{I}(\sigma)$ is the set of states that the output of the EM will be authorised to 'traverse,' meaning that the EM can emit the first event of its buffer σ immediately from these states, but not wait in them (contrary to the states in $\mathrm{S}(\sigma)$, from which the EM could choose to wait before emitting a new event).

These sets are defined by induction on σ, 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: $\mathrm{I}(\epsilon) = \emptyset$. 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, $\mathrm{S}(\epsilon) = \{q \in F_G \mid q \text{ after } \mathrm{tw}(\Sigma_\mathrm{u}) \subseteq F_G\}$.

If a new controllable event a is received, it is added to the buffer, and then the EM can decide to emit the first event of its buffer to reach a state that is in S or I for its new buffer, this explains the definition of $\mathrm{I}(a.\sigma)$. 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 $\mathrm{S}(\sigma) \subseteq \mathrm{S}(\sigma.a)$. Moreover, $\mathrm{S}(\sigma.a)$ 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 $\mathrm{I}(\sigma.a)$. Y is the set of states that are in F_G and from which the EM can decide to wait for a new uncontrollable event before doing anything. Since $Y \subseteq F_G$, if no uncontrollable event is to be received, the property is satisfied, and otherwise, the EM can decide what to emit to reach F_G. In order to ensure that receiving uncontrollable events does 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 $\mathrm{I}(\sigma.a)$. This is the purpose of the condition $(X \cup Y) \cap \mathrm{uPred}(\overline{X \cup Y \cup \mathrm{I}(\sigma.a)}) = \emptyset$. 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 $x \text{ after } (\epsilon, t) \in X$ and $Y = \mathrm{up}(Y)$. To have the best EM possible, these sets are as large as possible.

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

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

$$\begin{aligned}\mathrm{G}(q, \sigma) = \{w \in \mathrm{tw}(\Sigma) \mid\ & \Pi_\Sigma(w) \preccurlyeq \sigma \wedge \\ & q \text{ after } w \in F_G \wedge \forall t \in \mathbb{R}_{\geqslant 0}, q \text{ after } (w, t) \in \mathrm{S}(\Pi_\Sigma(\mathrm{obs}(w, t))^{-1}.\sigma)\}.\end{aligned}$$

It is now possible to use G to define an enforcement function for φ denoted as E_φ:

Definition 4.6 (Functions store_φ, E_φ). Let store_φ be the function $: \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geqslant 0} \to \mathrm{tw}(\Sigma) \times \mathrm{tw}(\Sigma_c) \times \Sigma_c^*$ defined inductively by
$\forall t \in \mathbb{R}_{\geqslant 0}, \mathrm{store}_\varphi(\epsilon, t) = (\epsilon, \epsilon, \epsilon)$,
and for $\sigma \in \mathrm{tw}(\Sigma)$, (t', a) s.t. $\sigma.(t', a) \in \mathrm{tw}(\Sigma)$, and $t \geqslant t'$, if $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t')$, then

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

with

$$\begin{aligned}
\kappa_\varphi(q, w) &= \min_{\text{lex}}(\max_{\preccurlyeq}(\mathrm{G}(q, w) \cup \{\epsilon\})), \text{for } q \in Q \text{ and } w \in \Sigma_c^*,\\
\mathit{buffer}_c &= \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c,\\
t_1 &= \begin{array}{l}\min(\{t'' \in \mathbb{R}_{\geqslant 0} \mid t'' \geqslant t' \wedge \\ \quad \mathrm{G}(\mathrm{Reach}(\sigma_s.(t', a), t''), \mathit{buffer}_c) \neq \emptyset\} \cup \{+\infty\}),\end{array}\\
\sigma'_b &= \kappa_\varphi(\mathrm{Reach}(\sigma_s.(t', a), \min(\{t, t_1\})), \mathit{buffer}_c) +_t \min(\{t, t_1\}),\\
\sigma'_c &= \Pi_\Sigma(\sigma'_b)^{-1}.\mathit{buffer}_c,\\
t_2 &= \begin{array}{l}\min(\{t'' \in \mathbb{R}_{\geqslant 0} \mid t'' \geqslant t' \wedge \\ \quad \mathrm{G}(\mathrm{Reach}(\sigma_s, t''), \mathit{buffer}_c.a) \neq \emptyset\} \cup \{+\infty\}),\end{array}\\
\sigma''_b &= \kappa_\varphi(\mathrm{Reach}(\sigma_s, \min(\{t, t_2\})), \mathit{buffer}_c.a) +_t \min(\{t, t_2\}),\\
\sigma''_c &= \Pi_\Sigma(\sigma''_b)^{-1}.(\mathit{buffer}_c.a).
\end{aligned}$$

For $\sigma \in \mathrm{tw}(\Sigma)$ and $t \in \mathbb{R}_{\geqslant 0}$, we define $\mathrm{E}_\varphi(\sigma, t) = (\Pi_1(\mathrm{store}_\varphi(\mathrm{obs}(\sigma, t), t)))$.

The function store_φ takes a timed word σ and a date t as input, and outputs three words: σ_s, σ_b and σ_c. σ_s is the output of the enforcement function at time t. σ_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. σ_c is the untimed word composed of the remaining controllable actions of the buffer. When time elapses, after the last event of the input, σ_s is modified to output the events of σ_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 σ_b can change at this moment, changing from ϵ to a word in G. This change of σ_b when letting time elapse can only happen once, since G will not be empty anymore once it has become non-empty. t_1 and t_2 are used for this purpose, they both represent the time at which G becomes non-empty, if $a \in \Sigma_u$ or $a \in \Sigma_c$, respectively. Words are thus calculated from this point whenever G has become non-empty, to ensure that what has already been output is not modified. If G is still empty, then $\min(\{t, t_1\})$ (or $\min(\{t, t_2\})$, depending on whether $a \in \Sigma_c$ or $a \in \Sigma_u$) equals to t, meaning that $\sigma_b = \epsilon$. Most of the time, t_1 or t_2 is equal to t', it is not the case only when G was still empty at time t', but if G was not empty at date t', then t_1 (or t_2) is equal to t'. σ_c contains the controllable actions of the input that have not been output and do not belong to σ_b. It is used to compute the new value of σ_b when possible. When receiving a new event in the input, it is appended to σ_s if it is an uncontrollable event, or the action is appended to the buffer if it is a controllable one. Then, σ_b is computed again, from the new state reached if it was an uncontrollable event, or with the new buffer if it was controllable. Note that t_1 and t_2 may not exist, since they are minima of an interval that can be open, depending on the strictness of the considered guard. In this case, one should consider the infimum instead of the minimum, and add an infinitesimal delay, s.t. the required transition is taken.

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

Definition 4.7 **($\mathrm{Pre}(\varphi)$).** $\mathrm{Pre}(\varphi) = \{(\sigma, t) \mid \sigma \in \mathrm{Pre}(\varphi, t)\}$, where, for $\sigma \in \mathrm{tw}(\Sigma)$ and $t \in \mathbb{R}_{\geqslant 0}$,

$$\begin{aligned}\mathrm{Pre}(\varphi, t) = \{\sigma \in \mathrm{tw}(\Sigma) \mid \exists \sigma' \preccurlyeq \sigma, \exists t' \leqslant t,\\ \mathrm{G}(\mathrm{Reach}(\mathrm{obs}(\sigma', t')_{|\Sigma_u}, t'), \Pi_\Sigma(\mathrm{obs}(\sigma', t')_{|\Sigma_c})) \neq \emptyset\}.\end{aligned}$$

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

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

Proposition 4.1. E_φ *as defined in Definition 4.6 is an enforcement function, as per Definition 4.1.*

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

— $t'' \leqslant t$. Then, in the definition of store_φ, t_1 (or t_2, if a is controllable) has the same value in $\mathrm{store}_\varphi(\sigma, t)$ and $\mathrm{store}_\varphi(\sigma.(t'', a), t')$. Then, comparing t to t_1, either $\mathrm{E}_\varphi(\sigma.(t'', a), t) = \epsilon$ if $t < t_1$, and then $\mathrm{E}_\varphi(\sigma.(t'', a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma.(t'', a), t')$, or $t \geqslant t_1$, and then there exists σ_s and σ_b s.t. $\mathrm{E}_\varphi(\sigma.(t'', a), t) = \sigma_s.\mathrm{obs}(\sigma_b, t)$ and $\mathrm{E}_\varphi(\sigma\,.\,(t'', a), t') = \sigma_s\,.\,\mathrm{obs}(\sigma_b, t')$, meaning that $\mathrm{E}_\varphi(\sigma\,.\,(t'', a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma\,.\,(t'', a), t')$.
— $t'' \geqslant t'$. Then the proposition holds because in the definition of E_φ, only the observation of the input word at the given time is considered, meaning that $\mathrm{E}_\varphi(\sigma.(t'', a), t) = \mathrm{E}_\varphi(\sigma, t)$ and $\mathrm{E}_\varphi(\sigma.(t'', a), t') = \mathrm{E}_\varphi(\sigma, t')$. By induction hypothesis, the proposition thus holds.
— $t < t'' < t'$. Then, $\mathrm{E}_\varphi(\sigma.(t'', a), t) = \mathrm{E}_\varphi(\sigma, t)$ and $\mathrm{E}_\varphi(\sigma.(t'', a), t') = \Pi_1(\mathrm{store}_\varphi(\sigma.(t'', a), t'))$, meaning that, looking at the definition of store_φ, $\mathrm{E}_\varphi(\sigma\,.\,(t'', a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma\,.\,(t'', a), t')$.

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

Proposition 4.2. E_φ *is sound with respect to* φ *in* $\mathrm{Pre}(\varphi)$ *as per Definition 4.2.*

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

— $(\sigma.(t', a), t) \notin \mathrm{Pre}(\varphi)$. Then, the proposition holds.
— $(\sigma.(t', a), t) \in \mathrm{Pre}(\varphi)$, but $(\sigma, t') \notin \mathrm{Pre}(\varphi)$. Then, this is when the input reaches $\mathrm{Pre}(\varphi)$. Considering the definition of $\mathrm{Pre}(\varphi)$, 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.
— $(\sigma, t') \in \mathrm{Pre}(\varphi)$ (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 4.3. E_φ *is compliant as per Definition 4.3.*

Sketch of proof. As in the untimed setting, the proof is made by induction on the input σ, 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 4.4. E_φ *is optimal in* $\mathrm{Pre}(\varphi)$ *as per Definition 4.4.*

Sketch of proof. This proof is made by induction on the input σ. Whenever $\sigma \in \mathrm{Pre}(\varphi)$, since E_φ is sound in $\mathrm{Pre}(\varphi)$, then $\mathrm{E}_\varphi(\sigma)$ is the maximal word (with respect to $\preccurlyeq_{\mathrm{d}}$) that satisfies φ and is safe to output. It is maximal because in the definition of store_φ, κ_φ returns the longest word with lower delays (for lexicographic order), which corresponds to the maximum with respect to $\preccurlyeq_{\mathrm{d}}$. Thus, outputting a greater word (with respect to $\preccurlyeq_{\mathrm{d}}$) would lead to G being empty, meaning that the EM would not be sound. Thus, E_φ is optimal in $\mathrm{Pre}(\varphi)$, since it outputs the maximal word with respect to $\preccurlyeq_{\mathrm{d}}$ 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_φ, as defined in Definition 4.6.

Definition 4.8. An *enforcement monitor* $\mathcal{E}$ for φ is a transition system $\langle C^{\mathcal{E}}, c_0^{\mathcal{E}}, \Gamma^{\mathcal{E}}, \hookrightarrow_{\mathcal{E}}\rangle$ s.t.:
— $C^{\mathcal{E}} = \mathrm{tw}(\Sigma) \times \Sigma_{\mathrm{c}}^* \times Q \times \mathbb{R}_{\geqslant 0} \times \{\top, \bot\}$ is the set of configurations.
— $c_0^{\mathcal{E}} = \langle \epsilon, \epsilon, q_0, 0, \bot\rangle \in C^{\mathcal{E}}$ is the initial configuration.
— $\Gamma^{\mathcal{E}} = ((\mathbb{R}_{\geqslant 0} \times \Sigma) \cup \{\epsilon\}) \times Op \times ((\mathbb{R}_{\geqslant 0} \times \Sigma) \cup \{\epsilon\})$ is the alphabet, composed of an optional input, an operation and an optional output.
The set of operations is $\{\mathrm{compute}(.), \mathrm{dump}(.), \text{pass-uncont}(.), \text{store-cont}(.), \mathrm{delay}(.)\}$.
Whenever $(\sigma, \bowtie, \sigma') \in \Gamma^{\mathcal{E}}$, it will be noted $\sigma / \bowtie / \sigma'$.
— $\hookrightarrow_{\mathcal{E}}$ is the transition relation defined as the smallest relation obtained by applying the following rules given by their priority order:

- **Compute:** $\langle \epsilon, \sigma_c, q, t, \bot\rangle \xhookrightarrow{\epsilon/\,\mathrm{compute}()/\epsilon}_{\mathcal{E}} \langle \sigma_b', \sigma_c', q, t, \top\rangle$, if $\mathrm{G}(q, \sigma_c) \neq \emptyset$, with $\sigma_b' = \kappa_\varphi(q, \sigma_c) +_{\mathrm{t}} t$, and $\sigma_c' = \Pi_\Sigma(\sigma_b')^{-1}.\sigma_c$,
- **Dump:** $\langle (t_b, a).\sigma_b, \sigma_c, q, t_b, \top\rangle \xhookrightarrow{\epsilon/\,\mathrm{dump}(t_b,a)/(t_b,a)}_{\mathcal{E}} \langle \sigma_b, \sigma_c, q', t_b, \top\rangle$, with $q' = q$ after $(0, a)$,
- **Pass-uncont:** $\langle \sigma_b, \sigma_c, q, t, b\rangle \xhookrightarrow{(t,a)/\,\text{pass-uncont}(t,a)/(t,a)}_{\mathcal{E}} \langle \epsilon, \Pi_\Sigma(\sigma_b).\sigma_c, q', t, \bot\rangle$, with $q' = q$ after $(0, a)$,
- **Store-cont:** $\langle \sigma_b, \sigma_c, q, t, b\rangle \xhookrightarrow{(t,c)/\,\text{store-cont}((t,c))/\epsilon}_{\mathcal{E}} \langle \epsilon, \Pi_\Sigma(\sigma_b).\sigma_c.c, q, t, \bot\rangle$,
- **Delay:** $\langle \sigma_b, \sigma_c, (l, v), t, b\rangle \xhookrightarrow{\epsilon/\,\mathrm{delay}(\delta)/\epsilon}_{\mathcal{E}} \langle \sigma_b, \sigma_c, (l, v + \delta), t + \delta, b\rangle$.

In a configuration $\langle \sigma_b, \sigma_c, q, t, b\rangle$, σ_b is the word to be output as time elapses; σ_c is the sequence of controllable actions from the input that are not used in σ_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 σ_b and σ_c should be computed (due to the reception of a new event, for example).

The timed word σ_b corresponds to $\mathrm{nobs}(\sigma_b, t)$ from the definition of store_φ, whereas σ_c is the same as in the definition of store_φ. The state q represents σ_s, s.t. $q = \mathrm{Reach}(\sigma_s, t)$.

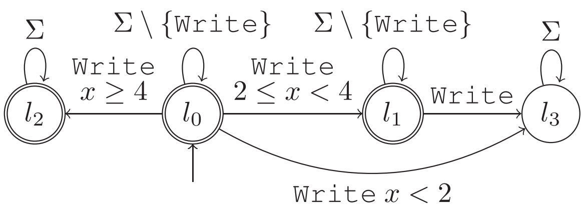

Fig. 7. Example of property without uncontrollable events.

Proposition 4.5. *The output of $\mathcal{E}$ for input σ is* $\mathrm{E}_\varphi(\sigma)$.

As in the untimed setting, in Proposition 4.5, 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 σ. 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 σ. 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 $\perp$, meaning that the word to be emitted can be computed. If the input is in $\mathrm{Pre}(\varphi)$, then rule compute() can be applied, and then the second member of the configuration will have the same value as the second member of store_φ, 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 $\mathrm{obs}(\sigma_b, t)$, where σ_b was computed by rule compute() as second member. Considering the definition of store_φ, it follows that the output of $\mathcal{E}$ with input σ at date t is $\mathrm{E}_\varphi(\sigma, t)$.

4.3. *Example*

Consider Figure 3, representing a property modelling the use of some shared writable device. We can get the status of a lock through the uncontrollable events *LockOn* and *LockOff* indicating that the lock has been locked by someone else, and that it is unlocked, respectively. The uncontrollable event *Auth* is sent by the device to authorise writings. Once the *Auth* event is received, we are able to send the controllable event Write after having waited some time for synchronisation. Each time the lock is taken and released, we must also wait before issuing a new Write order. The sets of events are $\Sigma_\mathrm{c} = \{\mathtt{Write}\}$ and $\Sigma_\mathrm{u} = \{\mathit{Auth}, \mathit{LockOff}, \mathit{LockOn}\}$. Now, let us follow the output of the store_φ function over time with the word $\sigma = (1, \mathit{Auth}).(2, \mathit{LockOn}).(4, \mathtt{Write}).(5, \mathit{LockOff}).(6, \mathit{LockOn}).(7, \mathtt{Write}).(8, \mathit{LockOff})$ as input: let $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\mathrm{obs}(\sigma, t), t)$. Then the values taken by σ_s, σ_b and σ_c over time are given in Table 2. To calculate them, notice that for any valuation $v : \{x\} \to \mathbb{R}_{\geqslant 0}$, $(l_1, v) \in \mathrm{S}(\epsilon)$ and $(l_2, v) \in \mathrm{S}(\epsilon)$, since all uncontrollable words from l_1 and l_2 lead to l_1 or l_2, which are both accepting states.

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

Fig. 8. Execution of an enforcement monitor with input $(1, \mathit{Auth}).(2, \mathit{LockOn}).(4, \mathtt{Write}).(5, \mathit{LockOff}).(6, \mathit{LockOn}).(7, \mathtt{Write}).(8, \mathit{LockOff})$.

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

t	σ_s	σ_b	σ_c
1	$(1, \mathit{Auth})$	ϵ	ϵ
2	$(1, \mathit{Auth}).(2, \mathit{LockOn})$	ϵ	ϵ
4	$(1, \mathit{Auth}).(2, \mathit{LockOn})$	ϵ	$\mathtt{Write}$
5	$(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff})$	$(7, \mathtt{Write})$	ϵ
6	$(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff}).(6, \mathit{LockOn})$	ϵ	$\mathtt{Write}$
7	$(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff}).(6, \mathit{LockOn})$	ϵ	$\mathtt{Write} \cdot \mathtt{Write}$
8	$(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff}).(6, \mathit{LockOn}).(8, \mathit{LockOff})$	$(10, \mathtt{Write}).(10, \mathtt{Write})$	ϵ
10	$(1, \mathit{Auth}).(2, \mathit{LockOn}).(5, \mathit{LockOff}).(6, \mathit{LockOn}).(8, \mathit{LockOff}).(10, \mathtt{Write}).(10, \mathtt{Write})$	ϵ	ϵ

We can also follow the execution of an enforcement monitor enforcing the property in Figure 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. Variable t defines the global time of the execution. Figure 8 shows the execution of the enforcement monitor with input $(1, \mathit{Auth}) \cdot (2, \mathit{LockOn}) \cdot (4, \mathtt{Write}) \cdot (5, \mathit{LockOff}) \cdot (6, \mathit{LockOn}) \cdot (7, \mathtt{Write}) \cdot (8, \mathit{LockOff}))$. In Figure 8, valuations are represented as integers, giving the value of the only clock x of the property, $\mathit{LockOff}$ is abbreviated as $\mathtt{off}$, LockOn as $\mathtt{on}$ and $\mathtt{Write}$ as $\mathtt{w}$. The first column depicts the dates of events, then red text is the current output (σ_s) of the EM, blue text shows the evolution of σ_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: $(1, \mathit{Auth}).(2, \mathtt{on}).(5, \mathtt{off}).(6, \mathtt{on}).(8, \mathtt{off}).(10, \mathtt{w}).(10, \mathtt{w})$.

Remark 4.2. The EM in Definition 4.6 outputs 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 Figure 7 over the set of controllable actions $\Sigma \supseteq \{\mathtt{Write}\}$, and the input timed word $(1, \mathtt{Write}).(1.5, \mathtt{Write})$ input to the EM. The output obtained with our approach at date $t = 4$ is $(4, \mathtt{Write}).(4, \mathtt{Write})$ whereas the output obtained in Pinisetty et al. (2012) would be $(2, \mathtt{Write})$.

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. (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. (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,b) 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 any regular timed property (Pinisetty et al. 2014a,c). In Pinisetty et al. (2014b), 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 EM 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. (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 the 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 EM, 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 received uncontrollable events, 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 the 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.

Appendix A. Proofs

A.1. *Proofs for the untimed setting*

In all this section, we will use the notations from Section 3, meaning that φ is a property whose associated automaton is $\mathcal{A}_\varphi = \langle Q, q_0, \Sigma, \rightarrow, F\rangle$. In some proofs, we also use notations from Definition 3.7.

Proposition 3.1. E_φ *as defined in Definition 3.7 is an enforcement function.*

Proof. Let us consider $\sigma \in \Sigma^*$ and $\sigma' \in \Sigma^*$. If $\sigma' = \epsilon$, then $\mathrm{E}_\varphi(\sigma) = \mathrm{E}_\varphi(\sigma.\sigma') \preccurlyeq \mathrm{E}_\varphi(\sigma.\sigma')$. Otherwise, let us consider $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$, $a = \sigma'(1)$ and $(\sigma_t, \sigma_d) = \mathrm{store}_\varphi(\sigma.a)$. Then, if $a \in \Sigma_\mathrm{u}$, $\sigma_t = \sigma_s.a.\sigma_s'$, where σ_s' is defined in Definition 3.7, meaning that $\sigma_s \preccurlyeq \sigma_t$. If $a \in \Sigma_\mathrm{c}$, then $\sigma_t = \sigma_s.\sigma_s''$, where σ_s'' is defined in Definition 3.7, thus again, $\sigma_s \preccurlyeq \sigma_t$. In both cases, $\mathrm{E}_\varphi(\sigma) = \sigma_s \preccurlyeq \sigma_t = \mathrm{E}_\varphi(\sigma.a)$. Since order $\preccurlyeq$ is transitive, this means that $\mathrm{E}_\varphi(\sigma) \preccurlyeq \mathrm{E}_\varphi(\sigma.a) \preccurlyeq \mathrm{E}_\varphi(\sigma.a.\sigma'(2)) \preccurlyeq \ldots \preccurlyeq \mathrm{E}_\varphi(\sigma.\sigma')$. Thus, E_φ is an enforcement function. □

Lemma A.1. $\forall \sigma \in \Sigma_\mathrm{c}^*, \forall a \in \Sigma_\mathrm{c}, \mathrm{I}(\sigma) \subseteq \mathrm{I}(\sigma.a)$.

Proof. For $\sigma \in \Sigma_\mathrm{c}^*$, let $\mathrm{P}(\sigma)$ be the predicate '$\forall a \in \Sigma_\mathrm{c}, \mathrm{I}(\sigma) \subseteq \mathrm{I}(\sigma.a)$.' Let us show by induction that $\mathrm{P}(\sigma)$ holds for every $\sigma \in \Sigma_\mathrm{c}^*$.

— *Induction basis:* If $a \in \Sigma_\mathrm{c}$, then since $\mathrm{I}(\epsilon) = \emptyset$, $\mathrm{I}(\epsilon) \subseteq \mathrm{I}(a)$. Thus, $\mathrm{P}(\epsilon)$ holds.

— *Induction step:* Let us suppose that for $n \in \mathbb{N}$, for all $\sigma \in \Sigma_\mathrm{c}^*$ s.t. $|\sigma| \leqslant n$, $\mathrm{P}(\sigma)$ holds. Let us then consider $\sigma \in \Sigma_\mathrm{c}^*$ s.t. $|\sigma| = n + 1$, and $a \in \Sigma_\mathrm{c}$. Let $(h, \sigma_0) \in \Sigma_\mathrm{c} \times \Sigma_\mathrm{c}^*$ be s.t. $\sigma = h.\sigma_0$ (they must exist since $|\sigma| > 0$). Then, $|\sigma_0| = n$, and by induction hypothesis, $\mathrm{P}(\sigma_0)$ holds, meaning that $\mathrm{I}(\sigma_0) \subseteq \mathrm{I}(\sigma_0.a)$. Moreover, following the definition of $\mathrm{S}(\sigma_0.a)$, $\mathrm{S}(\sigma_0) \subseteq \mathrm{S}(\sigma_0.a)$. It follows that $\mathrm{S}(\sigma_0) \cup \mathrm{I}(\sigma_0) \subseteq \mathrm{S}(\sigma_0.a) \cup \mathrm{I}(\sigma_0.a)$, and thus $\mathrm{I}(\sigma) = \mathrm{I}(h.\sigma_0) = \mathrm{Pred}_h(\mathrm{S}(\sigma_0) \cup \mathrm{I}(\sigma_0)) \subseteq \mathrm{Pred}_h(\mathrm{S}(\sigma_0.a) \cup \mathrm{I}(\sigma_0.a)) = \mathrm{I}(h.\sigma_0.a) = \mathrm{I}(\sigma.a)$. This means that $\mathrm{P}(\sigma.a)$ holds.

Thus, by induction on the size of $\sigma \in \Sigma_c^*$, for all $\sigma \in \Sigma_c^*$, $\mathrm{P}(\sigma)$ holds. This means that for all $\sigma \in \Sigma_c^*$, for all $a \in \Sigma_c$, $\mathrm{I}(\sigma) \subseteq \mathrm{I}(\sigma.a)$. □

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

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

— *Induction basis:* Let us consider $u \in \Sigma_u$ and $q \in \mathrm{S}(\epsilon)$. Then, since $u \in \Sigma_u$, $u \in \Sigma_u^*$, and following the definition of $\mathrm{S}(\epsilon)$, $q \text{ after } u \in \mathrm{S}(\epsilon)$. Thus, $q \text{ after } u \in \mathrm{S}(\epsilon) \cup \mathrm{I}(\epsilon)$.

— *Induction step:* Let us suppose that for $\sigma \in \Sigma_c^*$, $\mathrm{P}(\sigma)$ holds. Let us then consider $u \in \Sigma_u$, $a \in \Sigma_c$, and $q \in \mathrm{S}(\sigma.a)$. Then, either $q \in \mathrm{S}(\sigma)$ or $q \in \max_{\subseteq}(\{Y \subseteq F_G \mid Y \cap \mathrm{uPred}(\overline{Y \cup \mathrm{I}(\sigma.a)}) = \emptyset\})$. If $q \in \mathrm{S}(\sigma)$, then by induction hypothesis, $\mathrm{P}(\sigma)$ holds, meaning that $q \text{ after } u \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$. Following lemma A.1, $\mathrm{I}(\sigma) \subseteq \mathrm{I}(\sigma.a)$, and since $\mathrm{S}(\sigma) \subseteq \mathrm{S}(\sigma.a)$, it follows that $\mathrm{S}(\sigma) \cup \mathrm{I}(\sigma) \subseteq \mathrm{S}(\sigma.a) \cup \mathrm{I}(\sigma.a)$. Thus, $q \text{ after } u \in \mathrm{S}(\sigma.a) \cup \mathrm{I}(\sigma.a)$. Otherwise, $q \in \max_{\subseteq}(\{Y \subseteq F_G \mid Y \cap \mathrm{uPred}(\overline{Y \cup \mathrm{I}(\sigma.a)}) = \emptyset\})$, and thus $q \text{ after } u \in \mathrm{S}(\sigma.a) \cup \mathrm{I}(\sigma.a)$. Thus, $\mathrm{P}(\sigma.a)$ holds.

By induction on σ, it follows that $\mathrm{P}(\sigma)$ holds for any $\sigma \in \Sigma_c^*$. Thus, for all $\sigma \in \Sigma_c^*$, for all $u \in \Sigma_u$, for all $q \in Q$, $(q \in \mathrm{S}(\sigma)) \implies (q \text{ after } u \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma))$. □

Lemma A.3. $\forall \sigma \in \Sigma_c^*, \forall q \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma), \mathrm{G}(q, \sigma) \neq \emptyset$.

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

— *Induction basis:* Let us consider $q \in \mathrm{S}(\epsilon) \cup \mathrm{I}(\epsilon)$. Then, since $\mathrm{I}(\epsilon) = \emptyset$, $q \in \mathrm{S}(\epsilon)$. Following the definition of $\mathrm{S}(\epsilon)$, this means that ϵ is s.t. $\epsilon \preccurlyeq \epsilon$ and $q \text{ after } \epsilon = q \in \mathrm{S}(\epsilon) = \mathrm{S}(\epsilon^{-1}.\epsilon)$. Thus, $\epsilon \in \mathrm{G}(q, \epsilon)$, meaning that $\mathrm{G}(q, \epsilon) \neq \emptyset$, and thus that $\mathrm{P}(\epsilon)$ holds.

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

  – $q \in \mathrm{S}(\sigma.a)$, then ϵ is s.t. $\epsilon \preccurlyeq \sigma.a$ and $q \text{ after } \epsilon \in \mathrm{S}(\sigma.a) = \mathrm{S}(\epsilon^{-1}.(\sigma.a))$, thus $\epsilon \in \mathrm{G}(q, \sigma.a)$.

  – $q \in \mathrm{I}(\sigma.a)$, then let $(h, \sigma_0) \in \Sigma_c \times \Sigma_c^*$ be s.t. $h.\sigma_0 = \sigma.a$ (they must exist since $|\sigma.a| > 0$). Then, $\mathrm{I}(\sigma.a) = \mathrm{I}(h.\sigma_0) = \mathrm{Pred}_h(\mathrm{S}(\sigma_0) \cup \mathrm{I}(\sigma_0))$, meaning that $q \in \mathrm{Pred}_h(\mathrm{S}(\sigma_0) \cup \mathrm{I}(\sigma_0))$. By induction hypothesis, since $|\sigma_0| = |\sigma| = n$, $\mathrm{P}(\sigma_0)$ holds, meaning that $\mathrm{G}(q \text{ after } h, \sigma_0) \neq \emptyset$. Let us then consider $w \in \mathrm{G}(q \text{ after } h, \sigma_0)$. Then, w is s.t. $w \preccurlyeq \sigma_0$ and $(q \text{ after } h) \text{ after } w \in \mathrm{S}(w^{-1}.\sigma_0)$. Thus, $h.w \preccurlyeq h.\sigma_0$ and $q \text{ after } (h.w) = (q \text{ after } h) \text{ after } w \in \mathrm{S}(w^{-1}.\sigma_0) = \mathrm{S}((h.w)^{-1}.(h.\sigma_0))$. Thus, $h.w \in \mathrm{G}(q, h.\sigma_0) = \mathrm{G}(q, \sigma.a)$.

In both cases, $\mathrm{G}(q, \sigma.a) \neq \emptyset$, meaning that $\mathrm{P}(\sigma.a)$ holds. By induction on the size of $\sigma \in \Sigma_c^*$, it follows that $\mathrm{P}(\sigma)$ holds for any $\sigma \in \Sigma_c^*$, meaning that for all $\sigma \in \Sigma_c^*$, for all $q \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$, $\mathrm{G}(q, \sigma) \neq \emptyset$. □

Lemma A.4. $\forall \sigma \in \Sigma^*, (\sigma \notin \mathrm{Pre}(\varphi) \wedge (\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)) \implies (\sigma_s = \sigma_{|\Sigma_u} \wedge \sigma_c = \sigma_{|\Sigma_c})$.

Proof. For $\sigma \in \Sigma^*$, let $\mathrm{P}(\sigma)$ be the predicate '$(\sigma \notin \mathrm{Pre}(\varphi) \wedge (\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)) \implies (\sigma_s = \sigma_{|\Sigma_u} \wedge \sigma_c = \sigma_{|\Sigma_c})$.' Let us show by induction that $\mathrm{P}(\sigma)$ holds for any $\sigma \in \Sigma^*$.

— *Induction basis:* $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon)$ and since $\epsilon_{|\Sigma_u} = \epsilon_{|\Sigma_c} = \epsilon$, $\mathrm{P}(\epsilon)$ holds.

— *Induction step:* Let us suppose that for $\sigma \in \Sigma^*$, $\mathrm{P}(\sigma)$ holds. Let us then consider $a \in \Sigma$, $(\sigma_s, \sigma_b) = \mathrm{store}_\varphi(\sigma)$, and $(\sigma_t, \sigma_d) = \mathrm{store}_\varphi(\sigma.a)$. Then, if $\sigma.a \in \mathrm{Pre}(\varphi)$, $\mathrm{P}(\sigma.a)$ holds. Let us now consider that $\sigma.a \notin \mathrm{Pre}(\varphi)$. Then, since $\mathrm{Pre}(\varphi)$ is extension closed, $\sigma \notin \mathrm{Pre}(\varphi)$, and thus, by induction hypothesis, $\sigma_s = \sigma_{|\Sigma_u}$ and $\sigma_c = \sigma_{|\Sigma_c}$. We consider two cases:

  – $a \in \Sigma_u$, then $\sigma_t = \sigma_s.a.\sigma_s'$, with $\sigma_s' \in \mathrm{G}(\mathrm{Reach}(\sigma_s.a), \sigma_c) \cup \{\epsilon\}$. Since $\sigma.a \notin \mathrm{Pre}(\varphi)$, $\mathrm{G}(\mathrm{Reach}((\sigma\ .a)_{|\Sigma_u}), (\sigma.a)_{|\Sigma_c}) = \emptyset$. Moreover, since $a \in \Sigma_u$, $(\sigma.a)_{|\Sigma_u} = \sigma_{|\Sigma_u}.a = \sigma_s.a$ and $(\sigma.a)_{|\Sigma_c} = \sigma_{|\Sigma_c} = \sigma_c$, thus $\mathrm{G}(\mathrm{Reach}(\sigma_s.a), \sigma_c) = \emptyset$. It follows that $\sigma_s' \in \{\epsilon\}$, meaning that $\sigma_t = \sigma_s.a = \sigma_{|\Sigma_u}.a = (\sigma.a)_{|\Sigma_u}$, and $\sigma_d = \sigma_s'^{-1}.\sigma_c = \sigma_c = \sigma_{|\Sigma_c} = (\sigma.a)_{|\Sigma_c}$.

  – $a \in \Sigma_c$, then $\sigma_t = \sigma_s.\sigma_s''$, with $\sigma_s'' \in \mathrm{G}(\sigma_s, \sigma_c.a) \cup \{\epsilon\}$. Since $\sigma.a \notin \mathrm{Pre}(\varphi)$, $\mathrm{G}(\mathrm{Reach}((\sigma\ .a)_{|\Sigma_u}), (\sigma.a)_{|\Sigma_c}) = \emptyset$. Moreover, since $a \in \Sigma_c$, $(\sigma.a)_{|\Sigma_u} = \sigma_{|\Sigma_u} = \sigma_s$ and $(\sigma.a)_{|\Sigma_c} = \sigma_{|\Sigma_c}.a = \sigma_c.a$. Thus, $\mathrm{G}(\mathrm{Reach}(\sigma_s), \sigma_c.a) = \emptyset$, meaning that $\sigma_s'' = \epsilon$. Thus, $\sigma_t = \sigma_s = \sigma_{|\Sigma_u} = (\sigma.a)_{|\Sigma_u}$ and $\sigma_d = \sigma_s''^{-1}.(\sigma_c.a) = \sigma_c.a = \sigma_{|\Sigma_c}.a = (\sigma.a)_{|\Sigma_c}$.

In both cases, $\mathrm{P}(\sigma.a)$ holds. By induction on $\sigma \in \Sigma^*$, for all $\sigma \in \Sigma^*$, if $\sigma \notin \mathrm{Pre}(\varphi)$ and $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$, then $\sigma_s = \sigma_{|\Sigma_u}$ and $\sigma_c = \sigma_{|\Sigma_c}$. □

Proposition 3.2. E_φ *is sound with respect to* φ *in* $\mathrm{Pre}(\varphi)$*, as per Definition 3.2.*

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

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

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

  – $(\sigma.a) \notin \mathrm{Pre}(\varphi)$. Then, $\mathrm{P}(\sigma.a)$ holds.

  – $(\sigma.a) \in \mathrm{Pre}(\varphi) \wedge \sigma \notin \mathrm{Pre}(\varphi)$. Then, since $\mathrm{Pre}(\varphi)$ is extension-closed, it follows that $\sigma\ .\ a \in \{w \in \Sigma^* \mid \mathrm{G}(\mathrm{Reach}(w_{|\Sigma_u}), w_{|\Sigma_c}) \neq \emptyset\}$, meaning that $\mathrm{G}(\mathrm{Reach}((\sigma\ .\ a)_{|\Sigma_u}), (\sigma\ .\ a)_{|\Sigma_c}) \neq \emptyset$. Moreover, since $\sigma \notin \mathrm{Pre}(\varphi)$, following lemma A.4, $\sigma_s = \sigma_{|\Sigma_u}$ and $\sigma_c = \sigma_{|\Sigma_c}$. Now, we consider two cases:

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

    • If $a \in \Sigma_c$, then $(\sigma.a)_{|\Sigma_u} = \sigma_{|\Sigma_u} = \sigma_s$, and $(\sigma.a)_{|\Sigma_c} = \sigma_{|\Sigma_c}.a = \sigma_c.a$. Thus, $\mathrm{G}(\mathrm{Reach}(\sigma_s), \sigma_c.a) \neq \emptyset$, meaning that $\sigma_s'' = \sigma_s^{-1}.\sigma_t \in \mathrm{G}(\mathrm{Reach}(\sigma_s), \sigma_c.a)$. As in the case where $a \in \Sigma_{\mathrm{u}}$, it follows that $\mathrm{Reach}(\sigma_t) \in \mathrm{S}(\sigma_d)$ and thus $\mathrm{E}_\varphi(\sigma.a) \models \varphi$. This means that $\mathrm{P}(\sigma.a)$ holds.

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

- $\sigma \in \mathrm{Pre}(\varphi)$ (and then $(\sigma.a) \in \mathrm{Pre}(\varphi)$ since $\mathrm{Pre}(\varphi)$ is extension closed). Then, by induction hypothesis, $\mathrm{P}(\sigma)$ holds, meaning that $\mathrm{Reach}(\sigma_s) \in \mathrm{S}(\sigma_b)$ and $\mathrm{E}_\varphi(\sigma) \models \varphi$. Again, we consider two cases:
  - If $a \in \Sigma_{\mathrm{u}}$, then, since $\mathrm{Reach}(\sigma_s) \in \mathrm{S}(\sigma_c)$, following lemma A.2, $\mathrm{Reach}(\sigma_s)$ after $a = \mathrm{Reach}(\sigma_s.a) \in \mathrm{S}(\sigma_c) \cup \mathrm{I}(\sigma_c)$. Following lemma A.3, $\mathrm{G}(\mathrm{Reach}(\sigma_s.a), \sigma_b) \neq \emptyset$. Thus, $\sigma'_s = (\sigma_s.a)^{-1}.\sigma_t \in \mathrm{G}(\mathrm{Reach}(\sigma_s.a), \sigma_c)$. It follows that $\mathrm{Reach}(\sigma_s.a.\sigma'_s) = \mathrm{Reach}(\sigma_t) \in \mathrm{S}(\sigma_s'^{-1}.\sigma_c) = \mathrm{S}(\sigma_d)$, and thus, since $\mathrm{S}(\sigma_d) \subseteq F_G$, $\mathrm{E}_\varphi(\sigma.a) = \sigma_t \models \varphi$. Henceforth, $\mathrm{P}(\sigma.a)$ holds.
  - If $a \in \Sigma_{\mathrm{c}}$, then, since $\mathrm{Reach}(\sigma_s) \in \mathrm{S}(\sigma_c)$ and $\mathrm{S}(\sigma_c) \subseteq \mathrm{S}(\sigma_c.a)$, $\mathrm{Reach}(\sigma_s) \in \mathrm{S}(\sigma_c.a)$. Following lemma A.3, $\mathrm{G}(\mathrm{Reach}(\sigma_s), \sigma_c.a) \neq \emptyset$. Thus, $\sigma''_s = \sigma_s^{-1}.\sigma_t \in \mathrm{G}(\mathrm{Reach}(\sigma_s), \sigma_c.a)$. As in the case where $a \in \Sigma_{\mathrm{u}}$, this leads to $\sigma_t \in \mathrm{S}(\sigma_d)$ and $\mathrm{E}_\varphi(\sigma.a) \models \varphi$. Henceforth, $\mathrm{P}(\sigma.a)$ holds.

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

In all cases, $\mathrm{P}(\sigma.a)$ holds. Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.a)$. By induction on σ, $\forall \sigma \in \Sigma^*, (\sigma \in \mathrm{Pre}(\varphi) \wedge (\sigma_s, \sigma_b) = \mathrm{store}_\varphi(\sigma)) \implies (\mathrm{E}_\varphi(\sigma) \models \varphi \wedge \mathrm{Reach}(\sigma_s) \in \mathrm{S}(\sigma_b))$. In particular, for all $\sigma \in \Sigma^*$, $(\sigma \in \mathrm{Pre}(\varphi)) \implies (\mathrm{E}_\varphi(\sigma) \models \varphi)$. This means that E_φ is sound with respect to φ in $\mathrm{Pre}(\varphi)$. □

Proposition 3.3. *E_φ is compliant as per Definition 3.3.*

Proof. For $\sigma \in \Sigma^*$, let $\mathrm{P}(\sigma)$ be the predicate: '$((\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)) \implies (\sigma_{s|\Sigma_{\mathrm{c}}}.\sigma_c = \sigma_{|\Sigma_{\mathrm{c}}} \wedge \sigma_{s|\Sigma_{\mathrm{u}}} = \sigma_{|\Sigma_{\mathrm{u}}})$.' Let us prove that for all $\sigma \in \Sigma^*$, $\mathrm{P}(\sigma)$ holds.

— *Induction basis*: $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon)$ and $\epsilon_{|\Sigma_{\mathrm{c}}} = \epsilon_{|\Sigma_{\mathrm{c}}}.\epsilon$, and $\epsilon_{|\Sigma_{\mathrm{u}}} = \epsilon_{|\Sigma_{\mathrm{u}}}$. Thus, $\mathrm{P}(\epsilon)$ holds.

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

- *Case 1: $a \in \Sigma_{\mathrm{u}}$*. Then, $\sigma_t = \sigma_s.a.\sigma'_s$, where σ'_s is defined in Definition 3.7, and $\sigma_t.\sigma_d = \sigma_s.a.\sigma_c$. Therefore, $\sigma_{t|\Sigma_{\mathrm{c}}}.\sigma_d = (\sigma_t.\sigma_d)_{|\Sigma_{\mathrm{c}}}$, since $\sigma_d \in \Sigma_{\mathrm{c}}^*$. Thus, $\sigma_{t|\Sigma_{\mathrm{c}}}.\sigma_d = \sigma_{s|\Sigma_{\mathrm{c}}}.\sigma_c$. Since $\mathrm{P}(\sigma)$ holds, $\sigma_{t|\Sigma_{\mathrm{c}}}.\sigma_d = \sigma_{|\Sigma_{\mathrm{c}}} = (\sigma.a)_{|\Sigma_{\mathrm{c}}}$.
  Moreover, since $\sigma'_s \in \Sigma_{\mathrm{c}}^*$, $\sigma_{t|\Sigma_{\mathrm{u}}} = \sigma_{s|\Sigma_{\mathrm{u}}}.a$. Since $\mathrm{P}(\sigma)$ holds, this means that $\sigma_{t|\Sigma_{\mathrm{u}}} = \sigma_{|\Sigma_{\mathrm{u}}}.a = (\sigma.a)_{|\Sigma_{\mathrm{u}}}$.
  Thus, $\mathrm{P}(\sigma.a)$ holds.
- *Case 2: $a \in \Sigma_{\mathrm{c}}$*. Then, $\sigma_t = \sigma_s.\sigma''_s$, where σ''_s is defined in Definition 3.7, and $\sigma_t.\sigma_d = \sigma_s.\sigma_c.a$. Therefore, $\sigma_{t|\Sigma_{\mathrm{c}}}.\sigma_d = (\sigma_t.\sigma_d)_{|\Sigma_{\mathrm{c}}} = (\sigma_s.\sigma_c.a)_{|\Sigma_{\mathrm{c}}} = \sigma_{s|\Sigma_{\mathrm{c}}}.\sigma_c.a$. Since $\mathrm{P}(\sigma)$ holds, this means that $\sigma_{t|\Sigma_{\mathrm{c}}}.\sigma_d = \sigma_{\Sigma_{\mathrm{c}}}.a = (\sigma.a)_{|\Sigma_{\mathrm{c}}}$.
  Moreover, since $\sigma''_s \in \Sigma_{\mathrm{c}}^*$, $\sigma_{t|\Sigma_{\mathrm{u}}} = \sigma_{s|\Sigma_{\mathrm{u}}}$. Since $\mathrm{P}(\sigma)$ holds, this means that $\sigma_{t|\Sigma_{\mathrm{u}}} = \sigma_{|\Sigma_{\mathrm{u}}} = (\sigma.a)_{|\Sigma_{\mathrm{u}}}$.
  Thus, $\mathrm{P}(\sigma.a)$ holds.

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

By induction on σ, for all $\sigma \in \Sigma^*$, $((\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)) \implies (\sigma_{s|\Sigma_{\mathrm{c}}}.\sigma_c = \sigma_{|\Sigma_{\mathrm{c}}} \wedge \sigma_{s|\Sigma_{\mathrm{u}}} = \sigma_{|\Sigma_{\mathrm{u}}})$. Moreover, if $\sigma \in \Sigma^*$, $u \in \Sigma_{\mathrm{u}}$, $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$ and $(\sigma_t, \sigma_d) = \mathrm{store}_\varphi(\sigma.u)$, then

$\sigma_t = \sigma_s.u.\sigma'_s$, where σ'_s is defined in Definition 3.7. Thus, $\sigma_s.u \preccurlyeq \sigma_t$, and since $\sigma_s = \mathrm{E}_\varphi(\sigma)$ and $\sigma_t = \mathrm{E}_\varphi(\sigma.u)$, it follows that $\mathrm{E}_\varphi(\sigma).u \preccurlyeq \mathrm{E}_\varphi(\sigma.u)$. Thus, for all $\sigma \in \Sigma^*$, $\mathrm{E}_\varphi(\sigma)_{|\Sigma_c} \preccurlyeq \sigma_{|\Sigma_c} \wedge \mathrm{E}_\varphi(\sigma)_{|\Sigma_u} = \sigma_{|\Sigma_u} \wedge \forall u \in \Sigma_u, \mathrm{E}_\varphi(\sigma).u \preccurlyeq \mathrm{E}_\varphi(\sigma.u)$, meaning that E_φ is compliant. □

Lemma A.5. $\forall \sigma \in \Sigma_c^*, \forall q \in Q, (q \notin \mathrm{S}(\sigma)) \implies (\exists \sigma_u \in \Sigma_u^*, q \text{ after } \sigma_u \notin F \wedge \forall \sigma'_u \preccurlyeq \sigma_u, \sigma'_u \neq \epsilon \implies q \text{ after } \sigma'_u \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma))$.

Proof. For $\sigma \in \Sigma_c^*$ and $q \in Q$, let $\mathrm{P}(\sigma, q)$ be the predicate '$\forall \sigma_u \in \Sigma_u^*, q \,\text{after}\, \sigma_u \in F \vee \exists \sigma'_u \preccurlyeq \sigma_u, \sigma'_u \neq \epsilon \wedge q \text{ after } \sigma'_u \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$.' Let us show the contrapositive of the lemma, that is that for all $\sigma \in \Sigma_c^*$ and $q \in Q$, $\mathrm{P}(\sigma, q) \implies q \in \mathrm{S}(\sigma)$. We consider two cases:

— If $\sigma = \epsilon$, let us consider $q \in Q$ s.t. $\mathrm{P}(\epsilon, q)$ holds. Then, since $\epsilon \in \Sigma_u^*$ and there does not exist a word w satisfying $w \preccurlyeq \epsilon \wedge w \neq \epsilon$, it follows that $q = q \text{ after } \epsilon \in F$. Let us consider $\sigma_u \in \Sigma_u^*$. Then, since $\mathrm{P}(\epsilon, q)$ holds, either $q \text{ after } \sigma_u \in F$, or there exists $\sigma'_u \preccurlyeq \sigma_u$ such that $\sigma'_u \neq \epsilon$ and $q \text{ after } \sigma'_u \in \mathrm{S}(\epsilon) \cup \mathrm{I}(\epsilon)$. In this last case, since $\mathrm{I}(\epsilon) = \emptyset$, $q \text{ after } \sigma'_u \in \mathrm{S}(\epsilon)$. Following the definition of $\mathrm{S}(\epsilon)$, since $\sigma'^{-1}_u.\sigma_u \in \Sigma_u^*$, $(q \text{ after } \sigma'_u) \text{ after } (\sigma'^{-1}_u.\sigma_u) = q \text{ after } \sigma_u \in F$. Thus, in all cases $q \text{ after } \sigma_u \in F$. Thus, for all $\sigma_u \in \Sigma_u^*$, $q \text{ after } \sigma_u \in F$, meaning that $q \in \mathrm{S}(\epsilon)$.

— If $\sigma \neq \epsilon$, there exists $\sigma' \in \Sigma_c^*$ and $a \in \Sigma$ s.t. $\sigma = \sigma'.a$, meaning that $\mathrm{S}(\sigma)$ is s.t. $\mathrm{S}(\sigma) = \mathrm{S}(\sigma') \cup \max_\subseteq(\{Z \subseteq F \mid Z \cap \mathrm{uPred}(\overline{Z \cup \mathrm{I}(\sigma)}) = \emptyset\})$. Let us consider $q \in Q$ s.t. $\mathrm{P}(\sigma, q)$ holds. Then, we define $Y = \{q \text{ after } \sigma_u \mid \sigma_u \in \Sigma_u^* \wedge \forall \sigma'_u \preccurlyeq \sigma_u, \sigma'_u \neq \epsilon \implies q \text{ after } \sigma'_u \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)\}$. Since $\mathrm{P}(\sigma, q)$ holds, $Y \subseteq F$. Moreover, if $y \in Y$ and $u \in \Sigma_u$, then

- either $y \text{ after } u \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$, and then $y \text{ after } u \in (Y \cup \mathrm{S}(\sigma)) \cup \mathrm{I}(\sigma)$,
- or $y \,\text{after}\, u \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$. Then, if $\sigma_u \in \Sigma_u$ is s.t. $y = q \,\text{after}\, \sigma_u$ (σ_u exists since $y \in Y$), then $y \text{ after } u = (q \text{ after } \sigma_u) \text{ after } u = q \text{ after } (\sigma_u.u) \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$. Since $\sigma_u.u \in \Sigma_u^*$, $y \text{ after } u \in Y \subseteq (Y \cup \mathrm{S}(\sigma)) \cup \mathrm{I}(\sigma)$.

Thus, $y \text{ after } u \in (Y \cup \mathrm{S}(\sigma)) \cup \mathrm{I}(\sigma)$, and since following lemma A.2, $\mathrm{S}(\sigma) \cap \mathrm{uPred}(\overline{\mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)}) = \emptyset$, this means that $(Y \cup \mathrm{S}(\sigma)) \cap \mathrm{uPred}(\overline{(Y \cup \mathrm{S}(\sigma)) \cup \mathrm{I}(\sigma)}) = \emptyset$. It follows that $(Y \cup \mathrm{S}(\sigma)) \subseteq \max_\subseteq(\{Z \subseteq F \mid Z \cap \mathrm{uPred}(\overline{Z \cup \mathrm{I}(\sigma)}) = \emptyset\}) \subseteq \mathrm{S}(\sigma)$. Since $q \in Y \subseteq \mathrm{S}(\sigma)$, this means that $q \in \mathrm{S}(\sigma)$.

Thus, for $\sigma \in \Sigma_c^*$ and $q \in Q$, $\mathrm{P}(\sigma, q) \implies q \in \mathrm{S}(\sigma)$. This means that the contrapositive also holds, thus $q \notin \mathrm{S}(\sigma) \implies \neg \mathrm{P}(\sigma, q)$, meaning that $q \notin \mathrm{S}(\sigma) \implies (\exists \sigma_u \in \Sigma_u^*, q \text{ after } \sigma_u \notin F \wedge \forall \sigma'_u \preccurlyeq \sigma_u, q \text{ after } \sigma'_u \neq \epsilon \implies q \text{ after } \sigma'_u \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma))$. □

Proposition 3.4. E_φ *is optimal in* $\mathrm{Pre}(\varphi)$ *as per Definition 3.4.*

Proof. Let E be an enforcement function s.t. $\mathrm{compliant}(E, \Sigma_c, \Sigma_u)$, and let us consider $\sigma \in \mathrm{Pre}(\varphi)$ and $a \in \Sigma$ s.t. $E(\sigma) = \mathrm{E}_\varphi(\sigma)$ and $|E(\sigma.a)| > |\mathrm{E}_\varphi(\sigma.a)|$. Let us also consider $(\sigma_s, \sigma_c) = \mathrm{store}_\varphi(\sigma)$. Let us show that there exists $\sigma_u \in \Sigma_u^*$ s.t. $E(\sigma.a.\sigma_u) \not\models \varphi$. We consider two cases:

— $a \in \Sigma_u$. Then, since E is compliant, and $E(\sigma) = \mathrm{E}_\varphi(\sigma) = \sigma_s$, there exists $\sigma'_{s1} \preccurlyeq \sigma_c$ s.t. $E(\sigma.a) = E(\sigma).a.\sigma_{s1} = \sigma_s.a.\sigma'_{s1}$. Moreover, there exists $\sigma'_s \preccurlyeq \sigma_c$ s.t. $\mathrm{E}_\varphi(\sigma.a) = \mathrm{E}_\varphi(\sigma).a.\sigma'_s = \sigma_s.a.\sigma'_s$. Since $|E(\sigma.a)| > |\mathrm{E}_\varphi(\sigma.a)|$, $|\sigma_{s1}| > |\sigma'_s|$. Considering that $\sigma'_s = \max_\preccurlyeq(\mathrm{G}(\mathrm{Reach}(\sigma_s.a), \sigma_c) \cup \{\epsilon\})$, it follows that $\sigma_{s1} \notin \mathrm{G}(\mathrm{Reach}(\sigma_s.a), \sigma_c)$. Following the definition of G, this means that either $\sigma_{s1} \not\preccurlyeq \sigma_c$, but since E' is compliant, this is not possible, or that Reach$(\sigma_s.a)$ after $\sigma_{s1} \notin$ S$(\sigma_{s1}^{-1}.\sigma_c)$. Let us consider $q = $ Reach$(\sigma_s.a.\sigma_{s1})$ and $\sigma_{c1} = \sigma_{s1}^{-1}.\sigma_c$. Then, $q \notin$ S(σ_{c1}). Following lemma A.5, this means that there exists $\sigma_u \in \Sigma_{\mathrm{u}}^*$ s.t. q after $\sigma_u \notin F$ and for all $\sigma_u' \preccurlyeq \sigma_u$, $\sigma_u' \neq \epsilon \implies q$ after $\sigma_u' \notin$ S$(\sigma_{c1}) \cup$ I(σ_{c1}). Then, we consider two cases:

– If $E(\sigma.a.\sigma_u) = \sigma_s.a.\sigma_{s1}.\sigma_u$, then Reach$(E(\sigma.a.\sigma_u)) \notin F$, meaning that $E(\sigma.a.\sigma_u) \not\models \varphi$.

– Otherwise, since E is compliant, there exists $\sigma_{s2} \preccurlyeq \sigma_{c1}$ and $\sigma_{u1} \preccurlyeq \sigma_u$ s.t. $\sigma_{s2} \neq \epsilon$, $\sigma_{u1} \neq \epsilon$ and $E(\sigma.a.\sigma_{u1}) = \sigma_s.a.\sigma_{s1}.\sigma_{u1}.\sigma_{s2}$. Let us consider $q' = q$ after $\sigma_{u1}.\sigma_{s2}$ and $\sigma_{c2} = \sigma_{s2}^{-1}.\sigma_{c1}$. Then, since $\sigma_{u1} \preccurlyeq \sigma_u$ and $\sigma_{u1} \neq \epsilon$, q after $\sigma_{u1} \notin$ S$(\sigma_{c1}) \cup$ I(σ_{c1}). Thus, $q' = q$ after $\sigma_{u1}.\sigma_{s2} \notin$ S$(\sigma_{c2}) \cup$ I(σ_{c2}), because otherwise, q after $\sigma_{u1} = $ Pred$_{\sigma_{s2}}(q') \in$ Pred$_{\sigma_{s2}}($S$(\sigma_{c2}) \cup$ I$(\sigma_{c2})) \subseteq$ I(σ_{c1}), which is absurd. Then, we can again use lemma A.5 to find a word $\sigma_{u2} \in \Sigma_{\mathrm{u}}^*$ s.t. q' after $\sigma_{u2} \notin F$ and for any $\sigma_u' \preccurlyeq \sigma_{u2}$, q' after $\sigma_u' \notin$ S$(\sigma_{c2}) \cup$ I(σ_{c2}). Since $\sigma_{s2} \neq \epsilon$, $|\sigma_{c2}| < |\sigma_{c1}|$, 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 $n \in \mathbb{N}$, there exists $(\sigma_{u1}, \sigma_{u2}, \ldots, \sigma_{un})$, and $(\sigma_{s1}, \sigma_{s2}, \ldots, \sigma_{sn})$, s.t. $E(\sigma\,.\,a\,.\,\sigma_{u1}\,.\,\sigma_{u2}\,.\cdots.\,\sigma_{un}) = \sigma_s\,.\,a\,.\,\sigma_{s1}\,.\,\sigma_{u1}\,.\,\sigma_{s2}\,.\,\sigma_{u2}\,.\cdots.\,\sigma_{sn}\,.\,\sigma_{un}$, and Reach$(\sigma_s.a.\sigma_{s1}.\sigma_{u1}.\sigma_{s2}.\sigma_{u2}.\cdots.\sigma_{sn}.\sigma_{un}) \notin F$. This means that, if $\sigma_u = \sigma_{u1}.\sigma_{u2}.\cdots.\sigma_{un}$, then $\sigma_u \in \Sigma_{\mathrm{u}}^*$ and $E(\sigma\,.\,a\,.\,\sigma_u) \not\models \varphi$.

Thus, in all cases, there exists $\sigma_u \in \Sigma_{\mathrm{u}}^*$ s.t. $E(\sigma\,.\,a\,.\,\sigma_u) \not\models \varphi$.

— $a \in \Sigma_{\mathrm{c}}$. The proof is the same as in the case where $a \in \Sigma_{\mathrm{u}}$, by replacing occurrences of '$\sigma_s.a$' by 'σ_s,' and occurrences of 'σ_b' by '$\sigma_b.a$.'

Thus, if E is an enforcement function s.t. there exists $\sigma \in$ Pre(φ), and $a \in \Sigma$ s.t. compliant$(E, \Sigma_{\mathrm{u}}, \Sigma_{\mathrm{c}})$, $E(\sigma) = $ E$_\varphi(\sigma)$ and $|E(\sigma.a)| > |$E$_\varphi(\sigma.a)|$, then there exists $\sigma_u \in \Sigma_{\mathrm{u}}^*$ s.t. $E(\sigma.a.\sigma_u) \not\models \varphi$. This means that E$_\varphi$ is optimal in Pre(φ). □

Proposition 3.5. *The output of the enforcement monitor $\mathcal{E}$ for input σ is* E$_\varphi(\sigma)$.

Proof. Let us introduce some notation for this proof: For a word $w \in \Gamma^{\mathcal{E}*}$, we note input$(w) = \Pi_1(w(1)).\Pi_1(w(2))\ldots\Pi_1(w(|w|))$, the word obtained by concatenating the first members (the inputs) of w. In a similar way, we note output$(w) = \Pi_3(w(1)).\Pi_3(w(2))\ldots\Pi_3(w(|w|))$, the word obtained by concatenating all the third members (outputs) of w. Since all configurations are not reachable from $c_0^{\mathcal{E}}$, for $w \in \Gamma^{\mathcal{E}*}$, we note Reach$(w) = c$ whenever $c_0^{\mathcal{E}} \overset{w}{\hookrightarrow}_{\mathcal{E}} c$, and Reach$(w) = \perp$ if such a c does not exist. We also define the Rules function as follows:

$$\text{Rules} : \begin{cases} \Sigma^* & \rightarrow \Gamma^{\mathcal{E}*} \\ \sigma & \mapsto \max_{\preccurlyeq}(\{w \in \Gamma^{\mathcal{E}*} \mid \text{input}(w) = \sigma \wedge \text{Reach}(w) \neq \perp\}) \end{cases}$$

For a word $\sigma \in \Sigma^*$, Rules(σ) is the trace of the longest valid run in $\mathcal{E}$, i.e., the sequence of all the rules that can be applied with input σ. We then extend the definition of output to words in Σ^*: for $\sigma \in \Sigma^*$, output$(\sigma) = $ output(Rules(σ)). We also note ϵ the empty word of Σ^*, and $\epsilon^{\mathcal{E}}$ the empty word of $\Gamma^{\mathcal{E}*}$. For $\sigma \in \Sigma^*$, let P(σ) be the predicate, 'E$_\varphi(\sigma) = $ output$(\sigma) \wedge (((\sigma_s, \sigma_c) = $ store$_\varphi(\sigma) \wedge$ Reach(Rules$(\sigma)) = \langle q, \sigma_c^{\mathcal{E}}\rangle) \implies (q = $ Reach$(\sigma_s) \wedge \sigma_c = \sigma_c^{\mathcal{E}}))$.'

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

— *Induction basis:* $\mathrm{E}_\varphi(\epsilon) = \epsilon = \mathrm{output}(\epsilon)$. Moreover, $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon)$ and $\mathrm{Reach}(\epsilon^{\mathcal{E}}) = c_0^{\mathcal{E}}$. Therefore, as $c_0^{\mathcal{E}} = \langle q_0, \epsilon \rangle$, $\mathrm{P}(\epsilon)$ holds, because $\mathrm{Reach}(\epsilon) = q_0$.

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

Since $\mathrm{P}(\sigma)$ holds, $\mathrm{Reach}(\mathrm{Rules}(\sigma)) = \langle q, \sigma_c \rangle$ and $\sigma_s = \mathrm{output}(\sigma)$. We consider two cases:

– $a \in \Sigma_\mathrm{u}$. Then, considering $\sigma_s' = (\sigma_s.a)^{-1}.\sigma_t$, $\sigma_t = \sigma_s.a.\sigma_s'$. Since $a \in \Sigma_\mathrm{u}$, rule pass-uncont can be applied: Let us consider $q' = q$ after a. Then, $\langle q, \sigma_c \rangle \xhookrightarrow{a/\,\text{pass-uncont}(a)/a}_{\mathcal{E}} \langle q', \sigma_c \rangle$. Then, if $\sigma_s' = \epsilon$, $\mathrm{G}(q', \sigma_c) = \emptyset$ or $\mathrm{G}(q', \sigma_c) = \{\epsilon\}$, meaning that no other rule can be applied, and thus $\mathrm{P}(\sigma.a)$ would hold. Otherwise, $\sigma_s' \neq \epsilon$ and thus $\sigma_s' \in \mathrm{G}(q', \sigma_c)$, meaning that $\mathrm{G}(q', \sigma_c) \neq \emptyset$ and $\mathrm{G}(q', \sigma_c) \neq \{\epsilon\}$, thus rule $\mathrm{dump}(\sigma_c(1))$ can be applied. Since $\sigma_s' \preccurlyeq \sigma_c$, $\sigma_s'(1) = \sigma_c(1)$, 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 $\sigma_s'(1)^{-1}.\sigma_s' \in \mathrm{G}(q_1, \sigma_c(1)^{-1}.\sigma_c)$, meaning that rule dump can be applied again. Rule dump can actually be applied $|\sigma_s'|$ times, since for all $w \preccurlyeq \sigma_s'$, if $w \neq \sigma_s'$, then $w^{-1}.\sigma_s' \neq \epsilon$ and $w^{-1}.\sigma_s' \in \mathrm{G}(q'$ after $w, w^{-1}.\sigma_c)$. Thus, after rule dump has been applied $|\sigma_s'|$ times, the configuration reached is $\langle q'$ after $\sigma_s', \sigma_s'^{-1}.\sigma_c \rangle$. Moreover, the output produced by all the rules dump is σ_s'. Since no rule can be applied after the $|\sigma_s'|$ applications of the rule dump, $\mathrm{output}(\sigma.a) = \mathrm{output}(\sigma).a.\sigma_s' = \sigma_t$ and $\mathrm{Reach}(\mathrm{Rules}(\sigma.a)) = \langle q'$ after $\sigma_s', \sigma_s'^{-1}.\sigma_c \rangle = \langle q$ after a after $\sigma_s', \sigma_d \rangle = \langle \mathrm{Reach}(\sigma_s)$ after a after $\sigma_s', \sigma_d \rangle = \langle \mathrm{Reach}(\sigma_s.a.\sigma_s'), \sigma_d \rangle = \langle \mathrm{Reach}(\sigma_t), \sigma_d \rangle$.

Thus, if $a \in \Sigma_\mathrm{u}$, $\mathrm{P}(\sigma.a)$ holds.

– $a \in \Sigma_\mathrm{c}$. Then, considering $\sigma_s'' = \sigma_s^{-1}.\sigma_t$, $\sigma_t = \sigma_s.\sigma_s''$. Since $a \in \Sigma_\mathrm{c}$, it is possible to apply the store-cont rule, and $\langle q, \sigma_c \rangle$ after $a/\,\text{store-cont}(a)/\epsilon = \langle q, \sigma_c.a \rangle$. Then, as in the case where $a \in \Sigma_\mathrm{u}$, rule dump can be applied $|\sigma_s''|$ times, meaning that the configuration reached would then be $\langle q$ after $(\sigma_c.a)(1) \,.\, (\sigma_c.a)(2) \,.\, \cdots \,.\, (\sigma_c.a)(|\sigma_s''|), (\sigma_c.a)(|\sigma_s''| + 1) \,.\, (\sigma_c.a)(|\sigma_s''| + 2) \,.\, \cdots \,.\, (\sigma_c.a)(|\sigma_c.a|) \rangle$. Since $\sigma_s'' \preccurlyeq \sigma_c.a$, $(\sigma_c.a)(1) \,.\, (\sigma_c.a)(2) \,.\, \cdots \,.\, (\sigma_c.a)(|\sigma_s''|) = \sigma_s''$, thus $\mathrm{Reach}(\mathrm{Rules}(\sigma.a)) = \langle q$ after $\sigma_s'', \sigma_s''^{-1} \,.\, (\sigma_c.a) \rangle = \langle \mathrm{Reach}(\sigma_t), \sigma_d \rangle$. Moreover, $\mathrm{output}(\sigma.a) = \mathrm{output}(\sigma).\sigma_s'' = \sigma_s.\sigma_s'' = \sigma_t = \mathrm{E}_\varphi(\sigma.a)$.

Thus, if $a \in \Sigma_\mathrm{c}$, $\mathrm{P}(\sigma.a)$ holds. This means that $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.a)$. Thus, by induction on σ, for all $\sigma \in \Sigma^*$, $\mathrm{P}(\sigma)$ holds. In particular, for all $\sigma \in \Sigma^*$, $\mathrm{E}_\varphi(\sigma) = \mathrm{output}(\sigma)$. □

A.2. *Proofs for the timed setting*

Proposition 4.1. E_φ *as defined in Definition 4.6 is an enforcement function as per Definition 4.1.*

Proof. For $\sigma \in \mathrm{tw}(\Sigma)$, let $\mathrm{P}(\sigma)$ be the predicate: '$\forall t \in \mathbb{R}_{\geqslant 0}, \forall t' \geqslant t, \mathrm{E}_\varphi(\sigma, t) \preccurlyeq \mathrm{E}_\varphi(\sigma, t')$.' Let us show by induction that for all $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds.

— *Induction basis:* $\sigma = \epsilon$. Then, let us consider $t \in \mathbb{R}_{\geqslant 0}$, and $t' \geqslant t$. Then, $\mathrm{E}_\varphi(\epsilon, t) = \epsilon \preccurlyeq \epsilon = \mathrm{E}_\varphi(\epsilon, t')$. Thus, $\mathrm{P}(\epsilon)$ holds.

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

– If $t \geqslant t''$, then let us consider $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t'')$, $(\sigma_{t1}, \sigma_{d1}, \sigma_{e1}) = \mathrm{store}_\varphi(\sigma.(t'', a), t)$, and $(\sigma_{t2}, \sigma_{d2}, \sigma_{e2}) = \mathrm{store}_\varphi(\sigma.(t'', a), t')$. Then, $\mathrm{E}_\varphi(\sigma.(t'', a), t) = \sigma_{t1}$ and $\mathrm{E}_\varphi(\sigma.(t'', a), t') = \sigma_{t2}$.

- If $a \in \Sigma_\mathrm{u}$, then considering t_1 as defined in Definition 4.6, $t_1 = \min(\{t_0 \in \mathbb{R}_{\geqslant 0} \mid t_0 \geqslant t'' \wedge \mathrm{G}(\mathrm{Reach}(\sigma_s.(t'', a), t_0), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'')).\sigma_c) \neq \emptyset\})$. Then, $\sigma_{d1} = \min_{\mathrm{lex}}(\max_{\preccurlyeq}(\mathrm{G}(\mathrm{Reach}(\sigma_s.(t'', a), \min(\{t, t_1\})), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'')).\sigma_c) \cup \{\epsilon\})) +_\mathrm{t} \min(\{t, t_1\})$, and $\sigma_{d2} = \min_{\mathrm{lex}}(\max_{\preccurlyeq}(\mathrm{G}(\mathrm{Reach}(\sigma_s.(t'', a), \min(\{t', t_1\})), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'')).\sigma_c) \cup \{\epsilon\})) +_\mathrm{t} \min(\{t', t_1\})$.

  · *Case 1:* $t \geqslant t_1$. Since $t' \geqslant t$, then $t' \geqslant t_1$, thus $\min(\{t', t_1\}) = \min(\{t, t_1\}) = t_1$, thus $\sigma_{d1} = \sigma_{d2}$. It follows that
  $\sigma_{t1} = \sigma_s.(t'', a).\,\mathrm{obs}(\sigma_{d1}, t) \preccurlyeq \sigma_s.(t'', a).\,\mathrm{obs}(\sigma_{d1}, t') = \sigma_s.(t'', a).\,\mathrm{obs}(\sigma_{d2}, t') = \sigma_{t2}$.

  · *Case 2:* $t < t_1$. Then, $\min(\{t, t_1\}) = t$. Since $t < t_1$, by definition of t_1, this means that $\mathrm{G}(\mathrm{Reach}(\sigma_s.(t'', a), t), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'')).\sigma_c) = \emptyset$, and thus $\sigma_{d1} = \epsilon$. Since $\sigma_{d1} = \epsilon$, $\sigma_{t1} = \sigma_s.(t'', a) \preccurlyeq \sigma_s.(t'', a).\,\mathrm{obs}(\sigma_{d2}, t') = \sigma_{t2}$.

  Thus, if $t' \geqslant t \geqslant t''$ and $a \in \Sigma_\mathrm{u}$, $\mathrm{P}(\sigma) \implies \mathrm{E}_\varphi(\sigma.(t'', a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma.(t'', a), t')$.

- Otherwise, $a \in \Sigma_\mathrm{c}$, and then considering t_2 as defined in Definition 4.6, $t_2 = \min(\{t_0 \in \mathbb{R}_{\geqslant 0} \mid t_0 \geqslant t'' \wedge \mathrm{G}(\mathrm{Reach}(\sigma_s, t_0), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'')).\sigma_c.a) \neq \emptyset\})$. Then, $\sigma_{d1} = \min_{\mathrm{lex}}(\max_{\preccurlyeq}(\mathrm{G}(\mathrm{Reach}(\sigma_s, \min(\{t, t_2\})), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'')) . \sigma_c . a) \cup \{\epsilon\})) +_\mathrm{t} \min(\{t, t_2\})$, and
  $\sigma_{d2} = \min_{\mathrm{lex}}(\max_{\preccurlyeq}(\mathrm{G}(\mathrm{Reach}(\sigma_s, \min(\{t', t_2\})), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'')).\sigma_c.a) \cup \{\epsilon\})) +_\mathrm{t} \min(\{t', t_2\})$.

  · *Case 1:* $t \geqslant t_2$. Since $t' \geqslant t$, $t' \geqslant t_2$, meaning that $\min(\{t, t2\}) = \min(\{t', t_2\}) = t_2$, and thus $\sigma_{d1} = \sigma_{d2}$. It follows that $\sigma_{t1} = \sigma_s.\,\mathrm{obs}(\sigma_{d1}, t)) \preccurlyeq \sigma_s.\,\mathrm{obs}(\sigma_{d1}, t') = \sigma_s.\,\mathrm{obs}(\sigma_{d2}, t') = \sigma_{t2}$.

  · *Case 2:* $t < t_2$. Then, $\mathrm{G}(\mathrm{Reach}(\sigma_s, \min(\{t, t_2\})), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'')).\sigma_c.a) = \emptyset$, meaning that $\sigma_{d1} = \epsilon$. Thus, $\sigma_{t1} = \sigma_s \preccurlyeq \sigma_s.\,\mathrm{obs}(\sigma_{d2}, t') = \sigma_{t2}$.

  Thus, if $t' \geqslant t \geqslant t''$ and $a \in \Sigma_\mathrm{c}$, $\mathrm{P}(\sigma) \implies \mathrm{E}_\varphi(\sigma.(t'', a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma.(t'', a), t')$.

Therefore, if $t' \geqslant t \geqslant t''$, for all $a \in \Sigma$, $\mathrm{P}(\sigma) \implies \mathrm{E}_\varphi(\sigma.(t'', a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma.(t'', a), t')$.

– If $t' < t''$, then $t < t''$, and $\mathrm{obs}(\sigma.(t'', a), t) = \mathrm{obs}(\sigma, t)$, and $\mathrm{obs}(\sigma.(t'', a), t') = \mathrm{obs}(\sigma, t')$. Thus, $\mathrm{E}_\varphi(\sigma.(t'', a), t) = \mathrm{store}_\varphi(\mathrm{obs}(\sigma.(t'', a), t), t) = \mathrm{store}_\varphi(\mathrm{obs}(\sigma, t), t) = \mathrm{E}_\varphi(\sigma, t)$, and $\mathrm{E}_\varphi(\sigma.(t'', a), t') = \mathrm{store}_\varphi(\mathrm{obs}(\sigma.(t'', a), t'), t') = \mathrm{store}_\varphi(\mathrm{obs}(\sigma, t'), t') = \mathrm{E}_\varphi(\sigma, t')$. Since $\mathrm{P}(\sigma)$ holds, then $\mathrm{E}_\varphi(\sigma.(t'', a), t) = \mathrm{E}_\varphi(\sigma, t) \preccurlyeq \mathrm{E}_\varphi(\sigma, t') = \mathrm{E}_\varphi(\sigma.(t'', a), t')$.

– If $t < t'' \leqslant t'$, then $\mathrm{obs}(\sigma.(t'', a), t) = \mathrm{obs}(\sigma, t)$. Since $\mathrm{P}(\sigma)$ holds, then $\mathrm{E}_\varphi(\sigma, t) \preccurlyeq \mathrm{E}_\varphi(\sigma, t'')$. Let $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t'')$ and $(\sigma_t, \sigma_d, \sigma_e) = \mathrm{store}_\varphi(\sigma.(t'', a), t')$. Then, $\sigma_t = \sigma_s.(t'', a).\ \mathrm{obs}(\sigma_e, t')$ if $a \in \Sigma_\mathrm{u}$, and $\sigma_t = \sigma_s.\,\mathrm{obs}(\sigma_e, t')$ if $a \in \Sigma_\mathrm{c}$. In both cases, $\sigma_s \preccurlyeq \sigma_t$. This means that $\mathrm{E}_\varphi(\sigma, t'') \preccurlyeq \mathrm{E}_\varphi(\sigma.(t'', a), t')$. Thus, $\mathrm{E}_\varphi(\sigma.(t'', a), t) = \mathrm{E}_\varphi(\sigma, t) \preccurlyeq \mathrm{E}_\varphi(\sigma, t'') \preccurlyeq \mathrm{E}_\varphi(\sigma.(t'', a), t')$.
Thus, if $t < t'' \leqslant t'$, then $\mathrm{P}(\sigma) \implies \mathrm{E}_\varphi(\sigma.(t'', a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma.(t'', a), t')$.

Consequently, in all cases, if $t \leqslant t'$, then $\mathrm{P}(\sigma) \implies \mathrm{E}_\varphi(\sigma.(t'',a),t) \preccurlyeq \mathrm{E}_\varphi(\sigma.(t'',a),t')$. Finally, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.(t'',a))$.

By induction, for all $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Thus, for all $\sigma \in \mathrm{tw}(\Sigma)$, for all $t \in \mathbb{R}_{\geqslant 0}$, for all $t' \geqslant t$, $\mathrm{E}_\varphi(\sigma,t) \preccurlyeq \mathrm{E}_\varphi(\sigma,t')$.

Now, let us consider $\sigma \in \mathrm{tw}(\Sigma)$ and (t,a) s.t. $\sigma.(t,a) \in \mathrm{tw}(\Sigma)$. Then, if $(\sigma_s,\sigma_b,\sigma_c) = \mathrm{store}_\varphi(\sigma,t)$ and $(\sigma_t,\sigma_d,\sigma_e) = \mathrm{store}_\varphi(\sigma.(t,a),t)$, then either $\sigma_t = \sigma_s.(t,a).\sigma'_s$, or $\sigma_t = \sigma_s.\sigma''_s$, whether a is controllable or uncontrollable, respectively, where σ'_s and σ''_s are defined in Definition 4.6. In both cases, $\sigma_s \preccurlyeq \sigma_t$. Thus, $\mathrm{E}_\varphi(\sigma,t) = \Pi_1(\mathrm{store}_\varphi(\mathrm{obs}(\sigma,t),t)) = \sigma_s \preccurlyeq \sigma_t = \Pi_1(\mathrm{store}_\varphi(\mathrm{obs}(\sigma.(t,a),t))) = \mathrm{E}_\varphi(\sigma.(t,a),t)$. This holds because, since $\sigma.(t,a) \in \mathrm{tw}(\Sigma)$, $\mathrm{time}(\sigma) \leqslant t$, thus $\mathrm{obs}(\sigma,t) = \sigma$. Thus, for all $\sigma \in \mathrm{tw}(\Sigma)$, for all $t \in \mathbb{R}_{\geqslant 0}$ and $t' \geqslant t$, $\mathrm{E}_\varphi(\sigma,t) \preccurlyeq \mathrm{E}_\varphi(\sigma,t')$ and $\mathrm{E}_\varphi(\sigma,t) \preccurlyeq \mathrm{E}_\varphi(\sigma.(t,a),t)$. This means that E_φ is an enforcement function. □

Lemma A.6. $\forall t \in \mathbb{R}_{\geqslant 0}, \forall \sigma \in \mathrm{tw}(\Sigma), (\sigma \notin \mathrm{Pre}(\varphi,t) \wedge (\sigma_s,\sigma_b,\sigma_c) = \mathrm{store}_\varphi(\sigma,t)) \implies (\sigma_s = \sigma_{|\Sigma_\mathrm{u}} \wedge \sigma_b = \epsilon \wedge \sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}))$.

Proof. For $\sigma \in \mathrm{tw}(\Sigma)$, let $\mathrm{P}(\sigma)$ be the predicate '$\forall t \geqslant \mathrm{time}(\sigma), (\sigma \notin \mathrm{Pre}(\varphi,t) \wedge (\sigma_s,\sigma_b,\sigma_c) = \mathrm{store}_\varphi(\sigma,t)) \implies (\sigma_s = \sigma_{|\Sigma_\mathrm{u}} \wedge \sigma_b = \epsilon \wedge \sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}))$.' Let us prove by induction that for all $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds.

— *Induction basis:* For $\sigma = \epsilon$, let us consider $t \in \mathbb{R}_{\geqslant 0}$. Then, $\mathrm{store}_\varphi(\epsilon,t) = (\epsilon,\epsilon,\epsilon)$. Considering that $\epsilon \in \mathrm{tw}(\Sigma_\mathrm{u})$, and $\epsilon = \Pi_\Sigma(\epsilon_{|\Sigma_\mathrm{c}})$, $\mathrm{P}(\epsilon)$ trivially holds (whether $\epsilon \in \mathrm{P}(\varphi,t)$ or not).

— *Induction step:* Suppose that for $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Let us consider (t',a) s.t. $\sigma.(t',a) \in \mathrm{tw}(\Sigma)$ and $t \geqslant t'$. Let us also consider $(\sigma_s,\sigma_b,\sigma_c) = \mathrm{store}_\varphi(\sigma,t')$ and $(\sigma_t,\sigma_d,\sigma_e) = \mathrm{store}_\varphi(\sigma.(t',a),t)$. Then, if $\sigma.(t',a) \in \mathrm{Pre}(\varphi,t)$, $\mathrm{P}(\sigma.(t',a))$ trivially holds. Thus, let us suppose that $\sigma.(t',a) \notin \mathrm{Pre}(\varphi,t)$. Since $\sigma \preccurlyeq \sigma.(t',a)$ and $t \geqslant t'$, it follows that $\sigma \notin \mathrm{Pre}(\varphi,t')$. By induction hypothesis, this means that $\sigma_s = \sigma_{|\Sigma_\mathrm{u}}$, $\sigma_b = \epsilon$ and $\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}})$. Then, since $\sigma.(t',a) \notin \mathrm{Pre}(\varphi,t)$ following the definition of $\mathrm{Pre}(\varphi,t)$, this means that for all $t'' \leqslant t$, $\mathrm{G}(\mathrm{Reach}(\mathrm{obs}(\sigma.(t',a),t'')_{|\Sigma_u},t''),\Pi_\Sigma(\mathrm{obs}(\sigma.(t',a),t'')_{|\Sigma_\mathrm{c}})) = \emptyset$. In particular, $\mathrm{G}(\mathrm{Reach}((\sigma.(t',a))_{|\Sigma_\mathrm{u}},t),\Pi_\Sigma((\sigma.(t',a))_{|\Sigma_\mathrm{c}})) = \emptyset$ (since $t \geqslant t'$, $\mathrm{obs}(\sigma.(t',a),t) = \sigma.(t',a)$). Then, there are two cases:

  - *Case 1:* $a \in \Sigma_\mathrm{u}$. Then, since $(\sigma.(t',a))_{|\Sigma_\mathrm{u}} = \sigma_{|\Sigma_\mathrm{u}}.(t',a) = \sigma_s.(t',a)$ and $\Pi_\Sigma((\sigma.(t',a))_{|\Sigma_\mathrm{c}}) = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}) = \Pi_\Sigma(\mathrm{nobs}(\sigma_b,t')).\sigma_c$, we have $\mathrm{G}(\mathrm{Reach}(\sigma_s.(t',a),t),\Pi_\Sigma(\sigma_b,t').\sigma_c) = \emptyset$. This means that $t < t_1$, where t_1 is defined in Definition 4.6, and thus $\sigma_d = \epsilon$. Since $\sigma_t = \sigma_s.(t',a).\,\mathrm{obs}(\sigma_d,t)$, $\sigma_t = \sigma_s.(t',a) = (\sigma.(t',a))_{|\Sigma_\mathrm{u}}$ and $\sigma_e = \sigma_c = \sigma_{|\Sigma_\mathrm{c}} = (\sigma.(t',a))_{|\Sigma_\mathrm{c}}$. Thus, $\mathrm{P}(\sigma.(t',a))$ holds if $a \in \Sigma_\mathrm{u}$.
  - *Case 2:* $a \in \Sigma_\mathrm{c}$. Then, $(\sigma.(t',a))_{|\Sigma_\mathrm{u}} = \sigma_{|\Sigma_\mathrm{u}} = \sigma_s$, and $\Pi_\Sigma((\sigma.(t',a))_{|\Sigma_\mathrm{c}}) = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}).a = \Pi_\Sigma(\mathrm{nobs}(\sigma_b,t')).\sigma_c.a$. Thus, $\mathrm{G}(\mathrm{Reach}(\sigma_s,t),\Pi_\Sigma(\mathrm{nobs}(\sigma_b,t').\sigma_c.a)) = \emptyset$. This means that $t < t_2$, where t_2 is defined in Definition 4.6, and thus $\sigma_d = \epsilon$. Since $\sigma_t = \sigma_s.\,\mathrm{obs}(\sigma_d,t)$, $\sigma_t = \sigma_s = \sigma_{|\Sigma_\mathrm{u}} = (\sigma.(t',a))_{|\Sigma_\mathrm{u}}$, and $\sigma_e = \Pi_\Sigma(\mathrm{nobs}(\sigma_b,t')).\sigma_c.a = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}).a = \Pi_\Sigma((\sigma.(t',a))_{|\Sigma_\mathrm{c}})$. Thus, $\mathrm{P}(\sigma.(t',a))$ holds if $a \in \Sigma_\mathrm{c}$.

  Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.(t',a))$.

By induction, for all $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Thus, for all $\sigma \in \mathrm{tw}(\Sigma)$, for all $t \in \mathbb{R}_{\geqslant 0}$, if $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t)$ and $(\sigma, t) \notin \mathrm{Pre}(\varphi)$, then $\sigma_s = \sigma_{|\Sigma_\mathrm{u}}$, $\sigma_b = \epsilon$, and $\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}})$. $\square$

Lemma A.7. $\forall \sigma \in \Sigma_\mathrm{c}^*, \forall a \in \Sigma_\mathrm{c}, \mathrm{I}(\sigma) \subseteq \mathrm{I}(\sigma.a)$.

Proof. For $\sigma \in \Sigma_\mathrm{c}^*$, let $\mathrm{P}(\sigma)$ be the predicate '$\forall a \in \Sigma_\mathrm{c}, \mathrm{I}(\sigma) \subseteq \mathrm{I}(\sigma.a)$.' Let us show by induction that $\mathrm{P}(\sigma)$ holds for all $\sigma \in \Sigma_\mathrm{c}^*$.

— *Induction basis:* Let us consider $a \in \Sigma_\mathrm{c}$. Then, $\mathrm{I}(\epsilon) = \emptyset \subseteq \mathrm{I}(a)$.

— *Induction step:* Suppose now that for $\sigma \in \Sigma_\mathrm{c}^*$, for all $\sigma' \in \Sigma_\mathrm{c}^*$ s.t. $|\sigma'| \leqslant |\sigma|$, $\mathrm{P}(\sigma')$ holds. Let us then consider $a \in \Sigma_\mathrm{c}$, $a' \in \Sigma_\mathrm{c}$ and $(h, \sigma_0) \in \Sigma_\mathrm{c} \times \Sigma_\mathrm{c}^*$ s.t. $h.\sigma_0 = \sigma.a$ (h and σ_0 exist because $\sigma.a \neq \epsilon$). Then, $\mathrm{I}(\sigma.a.a') = \mathrm{I}(h.\sigma_0.a') = \mathrm{Pred}_h(\mathrm{S}(\sigma_0.a') \cup \mathrm{I}(\sigma_0.a'))$, and $\mathrm{I}(\sigma.a) = \mathrm{I}(h.\sigma_0) = \mathrm{Pred}_h(\mathrm{S}(\sigma_0) \cup \mathrm{I}(\sigma_0))$. Following the definition of S, $\mathrm{S}(\sigma_0) \subseteq \mathrm{S}(\sigma_0.a')$. Moreover, by induction hypothesis, since $|\sigma_0| \leqslant |\sigma|$, $\mathrm{P}(\sigma_0)$ holds, meaning that $\mathrm{I}(\sigma_0) \subseteq \mathrm{I}(\sigma_0.a')$. Thus, $\mathrm{S}(\sigma_0) \cup \mathrm{I}(\sigma_0) \subseteq \mathrm{S}(\sigma_0.a') \cup \mathrm{I}(\sigma_0.a')$. It follows that $\mathrm{I}(\sigma.a) = \mathrm{Pred}_h(\mathrm{S}(\sigma_0) \cup \mathrm{I}(\sigma_0)) \subseteq \mathrm{Pred}_h(\mathrm{S}(\sigma_0.a') \cup \mathrm{I}(\sigma_0.a')) = \mathrm{I}(\sigma.a.a')$. Thus, for all $a' \in \Sigma_\mathrm{c}$, $\mathrm{I}(\sigma.a) \subseteq \mathrm{I}(\sigma.a.a')$, meaning that $\mathrm{P}(\sigma.a)$ holds.

By induction, $\mathrm{P}(\sigma)$ holds for every $\sigma \in \Sigma_\mathrm{c}^*$, meaning that for all $\sigma \in \Sigma_\mathrm{c}^*$, for all $a \in \Sigma_\mathrm{c}$, $\mathrm{I}(\sigma) \subseteq \mathrm{I}(\sigma.a)$. $\square$

Lemma A.8. $\forall q \in Q, \forall \sigma \in \Sigma_\mathrm{c}^*, (q \in \mathrm{S}(\sigma)) \implies (\forall u \in \Sigma_\mathrm{u}, q \text{ after } (0, u) \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma))$.

Proof. For $\sigma \in \Sigma_\mathrm{c}^*$, let $\mathrm{P}(\sigma)$ be the predicate '$\forall q \in Q, (q \in \mathrm{S}(\sigma)) \implies (\forall u \in \Sigma_\mathrm{u}, q \text{ after } (0, u) \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma))$.' Let us show by induction on σ that $\mathrm{P}(\sigma)$ holds for every $\sigma \in \Sigma_\mathrm{c}^*$.

— *Induction basis:* Let us consider $q \in \mathrm{S}(\epsilon)$. Then, for all $u \in \Sigma_\mathrm{u}$, since $(0, u) \in \mathrm{tw}(\Sigma_\mathrm{u})$, considering the definition of $\mathrm{S}(\epsilon)$, $q \text{ after } (0, u) \in \mathrm{S}(\epsilon)$. Thus, $q \in \mathrm{S}(\epsilon) \cup \mathrm{I}(\epsilon)$. Thus, $\mathrm{P}(\epsilon)$ holds.

— *Induction step:* Let us suppose that for $\sigma \in \Sigma_\mathrm{c}^*$, $\mathrm{P}(\sigma)$ holds. Let us consider $a \in \Sigma_\mathrm{c}$ and $q \in \mathrm{S}(\sigma.a)$. Then, considering the definition of $\mathrm{S}(\sigma.a)$, following two cases are possible:

  – Either $q \in \mathrm{S}(\sigma)$, and then, by induction hypothesis, for all $u \in \Sigma_\mathrm{u}$, $q \text{ after } (0, u) \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$. $\mathrm{S}(\sigma) \subseteq \mathrm{S}(\sigma.a)$, and following lemma A.7, $\mathrm{I}(\sigma) \subseteq \mathrm{I}(\sigma.a)$, thus, $q \text{after} (0, u) \in \mathrm{S}(\sigma.a) \cup \mathrm{I}(\sigma.a)$.

  – or $q \in \mathrm{S}(\sigma.a) \setminus \mathrm{S}(\sigma)$, and then, since $(\mathrm{S}(\sigma.a) \setminus \mathrm{S}(\sigma)) \cap \mathrm{uPred}(\overline{(\mathrm{S}(\sigma.a) \setminus \mathrm{S}(\sigma)) \cup \mathrm{I}(\sigma.a)}) = \emptyset$, it follows that if $u \in \Sigma_\mathrm{u}$, $q \text{ after } (0, u) \in (\mathrm{S}(\sigma.a) \setminus \mathrm{S}(\sigma)) \cup \mathrm{I}(\sigma.a) \subseteq \mathrm{S}(\sigma.a) \cup \mathrm{I}(\sigma.a)$.

In both cases, for all $u \in \Sigma_\mathrm{u}$, $q \text{ after } (0, u) \in \mathrm{S}(\sigma.a) \cup \mathrm{I}(\sigma.a)$, meaning that $\mathrm{P}(\sigma.a)$ holds. Thus, by induction, for all $\sigma \in \Sigma_\mathrm{c}^*$, $\mathrm{P}(\sigma)$ holds. Thus, for all $\sigma \in \Sigma_\mathrm{c}^*$, for all $q \in \mathrm{S}(\sigma)$, for all $u \in \Sigma_\mathrm{u}$, $q \text{ after } (0, u) \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$. $\square$

Lemma A.9. *For all $\sigma \in \Sigma_\mathrm{c}^*$, for all $q \in Q$, $(q \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)) \implies (\mathrm{G}(q, \sigma) \neq \emptyset)$.*

Proof. For $\sigma \in \Sigma_\mathrm{c}^*$, let $\mathrm{P}(\sigma)$ be the predicate '$\forall q \in Q, (q \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)) \implies (\mathrm{G}(q, \sigma) \neq \emptyset)$.' Let us then prove by induction on σ that $\mathrm{P}(\sigma)$ holds for every $\sigma \in \Sigma_\mathrm{c}^*$.

— *Induction basis:* Let us consider $q \in \mathrm{S}(\epsilon) \cup \mathrm{I}(\epsilon)$. Since $\mathrm{I}(\epsilon) = \emptyset$, this means that $q \in \mathrm{S}(\epsilon)$. Then, ϵ satisfies $\epsilon \preccurlyeq \Pi_\Sigma(\epsilon)$. Moreover, since $\mathrm{S}(\epsilon) \subseteq F_G$, $q \text{ after } \epsilon = q \in F_G$, and for all $t \in \mathbb{R}_{\geqslant 0}$, $q \text{ after} (\epsilon, t) \in \mathrm{S}(\epsilon)$, because otherwise there would exist $\sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u})$

s.t. q after (ϵ, t) after $\sigma_u \notin F_G$, meaning that q after $(\sigma_u +_t t) \notin F_G$, and thus q would not be in $\mathrm{S}(\epsilon)$. Thus, $\epsilon \in G(q, \epsilon)$. This means that $G(q, \epsilon) \neq \emptyset$, and thus that $\mathrm{P}(\epsilon)$ holds.

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

- If $q \in \mathrm{I}(\sigma.a)$, let us consider $(h, \sigma_0) \in \Sigma_c \times \Sigma_c^*$ s.t. $\sigma.a = h.\sigma_0$. Then, $q \in \mathrm{I}(h.\sigma_0) = \mathrm{Pred}_h(\mathrm{S}(\sigma_0) \cup \mathrm{I}(\sigma_0))$, and since $|\sigma_0| = |\sigma| = n \leqslant n$, by induction hypothesis, $\mathrm{G}(q \text{ after } (0, h), \sigma_0) \neq \emptyset$. Let us consider $w \in \mathrm{G}(q \text{ after } (0, h), \sigma_0)$. Then, $(0, h).w$ satisfies $\Pi_\Sigma((0, h).w) \preccurlyeq h.\sigma_0$, q after $((0, h).w) = q$ after $(0, h)$ after $w \in F_G$, and for all $t \in \mathbb{R}_{\geqslant 0}$, q after $((0, h).w, t) = q$ after $(0, h)$ after $(w, t) \in \mathrm{S}(\Pi_\Sigma(w)^{-1}.\sigma_0) = \mathrm{S}(\Pi_\Sigma((0, h).w)^{-1}.(h.\sigma_0))$. Thus, $(0, h).w \in \mathrm{G}(q, h.\sigma_0) = \mathrm{G}(q, \sigma.a)$. Thus, $\mathrm{G}(q, \sigma.a) \neq \emptyset$.
- If $q \in \mathrm{S}(\sigma.a)$, then there are again two cases:
  - If $q \in \mathrm{S}(\sigma)$, then by induction hypothesis, $\mathrm{G}(q, \sigma) \neq \emptyset$. Since $\mathrm{G}(q, \sigma) \subseteq \mathrm{G}(q, \sigma.a)$, it follows that $\mathrm{G}(q, \sigma.a) \neq \emptyset$.
  - Otherwise, $q \in X \cup Y$, where X and Y are defined in the definition of $\mathrm{S}(\sigma.a)$.
    - If $q \in X$, then there exists $i \in \mathrm{I}(\sigma.a)$ and $\delta \in \mathbb{R}_{\geqslant 0}$ s.t. q after $(\epsilon, \delta) = i$, and for all $t \leqslant \delta$, q after $(\epsilon, t) \in X \subseteq \mathrm{S}(\sigma.a)$. Since $i \in \mathrm{I}(\sigma.a)$, we showed previously that $\mathrm{G}(i, \sigma.a) \neq \epsilon$. Let us consider $w \in \mathrm{G}(i, \sigma.a)$. Then, $w +_t \delta$ satisfies $\Pi_\Sigma(w +_t \delta) \preccurlyeq \sigma.a$, q after $(w +_t \delta) = i$ after $w \in F_G$, and for all $t \in \mathbb{R}_{\geqslant 0}$, if $t < \delta$, then q after $(w +_t \delta, t) = q$ after $(\epsilon, t) \in X \subseteq \mathrm{S}(\sigma.a)$, otherwise, q after $(w +_t \delta, t) = i$ after $(w, t - \delta) \in \mathrm{S}(\sigma.a)$. Thus, $w +_t \delta \in \mathrm{G}(q, \sigma.a)$. Thus, $\mathrm{G}(q, \sigma.a) \neq \emptyset$.
    - Otherwise, $q \in Y$, and then ϵ satisfies $\Pi_\Sigma(\epsilon) \preccurlyeq \sigma.a$, q after $\epsilon \in F_G$, and for all $t \in \mathbb{R}_{\geqslant 0}$, q after $(\epsilon, t) \in \mathrm{up}(q) \subseteq \mathrm{up}(Y) = Y \subseteq \mathrm{S}(\sigma.a)$. Thus, $\epsilon \in \mathrm{G}(q, \sigma.a)$. Thus, $\mathrm{G}(q, \sigma.a) \neq \emptyset$.

Thus, for all $q \in \mathrm{S}(\sigma.a) \cup \mathrm{I}(\sigma.a)$, $\mathrm{G}(q, \sigma.a) \neq \emptyset$. Thus, $\mathrm{P}(\sigma.a)$ holds. By induction on σ, $\mathrm{P}(\sigma)$ holds for ever $\sigma \in \Sigma_c^*$, meaning that for all $\sigma \in \Sigma_c^*$, for all $q \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$, $\mathrm{G}(q, \sigma) \neq \emptyset$. □

Proposition 4.2. E_φ *is sound with respect to* φ *in* $\mathrm{Pre}(\varphi)$ *as per Definition 4.2.*

Proof. Notation from 4.6 is to be used in this proof:

$$\kappa_\varphi(q, w) = \min_{\text{lex}}(\max_{\preccurlyeq}(\mathrm{G}(q, w) \cup \{\epsilon\})), \text{for } q \in Q \text{ and } w \in \Sigma_c^*,$$

$$\mathit{buffer}_c = \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c,$$

$$t_1 = \begin{array}{l} \min(\{t'' \in \mathbb{R}_{\geqslant 0} \mid t'' \geqslant t' \wedge \\ \quad \mathrm{G}(\mathrm{Reach}(\sigma_s.(t', a), t''), \mathit{buffer}_c) \neq \emptyset\} \cup \{+\infty\}), \end{array}$$

$$\sigma_b' = \kappa_\varphi(\mathrm{Reach}(\sigma_s.(t', a), \min(\{t, t_1\})), \mathit{buffer}_c) +_t \min(\{t, t_1\}),$$

$$\sigma_c' = \Pi_\Sigma(\sigma_b')^{-1}.\mathit{buffer}_c,$$

$$t_2 = \begin{array}{l} \min(\{t'' \in \mathbb{R}_{\geqslant 0} \mid t'' \geqslant t' \wedge \\ \quad \mathrm{G}(\mathrm{Reach}(\sigma_s, t''), \mathit{buffer}_c.a) \neq \emptyset\} \cup \{+\infty\}), \end{array}$$

$$\sigma_b'' = \kappa_\varphi(\mathrm{Reach}(\sigma_s, \min(\{t, t_2\})), \mathit{buffer}_c.a) +_t \min(\{t, t_2\}),$$

$$\sigma_c'' = \Pi_\Sigma(\sigma_b'')^{-1}.(\mathit{buffer}_c.a).$$

For $\sigma \in \mathrm{tw}(\Sigma)$ and $t \geqslant \mathrm{time}(\sigma)$, let $\mathrm{P}(\sigma, t)$ be the predicate '$(\sigma \in \mathrm{Pre}(\varphi, t) \wedge (\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t)) \implies (\mathrm{E}_\varphi(\sigma) \models \varphi \wedge \mathrm{nobs}(\sigma_b, t) -_{\mathrm{t}} t \in \mathrm{G}(\mathrm{Reach}(\sigma_s, t), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t)).\sigma_c))$.' Let also $\mathrm{P}(\sigma)$ be the predicate: '$\forall t \geqslant \mathrm{time}(\sigma), \mathrm{P}(\sigma, t)$.' Let us show that for all $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds.

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

- *Case 1:* $\epsilon \notin \mathrm{Pre}(\varphi, t)$. Then, $\mathrm{P}(\epsilon)$ trivially holds.
- *Case 2:* $\epsilon \in \mathrm{Pre}(\varphi, t)$. Then, there exists $t' \leqslant t$ s.t. $\mathrm{G}(\mathrm{Reach}(\mathrm{obs}(\epsilon, t')_{|\Sigma_\mathrm{u}}, t'), \epsilon) \neq \emptyset$, meaning that $\mathrm{G}(\mathrm{Reach}(\epsilon, t'), \epsilon) \neq \emptyset$. Thus, following the definition of $\mathrm{G}(\mathrm{Reach}(\epsilon, t'), \epsilon)$, $\epsilon \in \mathrm{G}(\mathrm{Reach}(\epsilon, t'), \epsilon)$ and $\mathrm{Reach}(\epsilon) \in F_G$. Since $\mathrm{E}_\varphi(\epsilon) = \epsilon$ and $\mathrm{Reach}(\epsilon) \in F_G$, $\mathrm{E}_\varphi(\epsilon) \models \varphi$. Thus, because $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon, \epsilon)$, $\mathrm{P}(\epsilon, t)$ holds.

Thus, in both cases, $\mathrm{P}(\epsilon, t)$ holds, meaning that $\mathrm{P}(\epsilon)$ holds.

— *Induction step:* Suppose that for $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Let us consider (t', a) s.t. $\sigma.(t', a) \in \mathrm{tw}(\Sigma)$, and $t \geqslant t' = \mathrm{time}(\sigma.(t', a))$. Let us also consider $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t')$ and $(\sigma_t, \sigma_d, \sigma_e) = \mathrm{store}_\varphi(\sigma.(t', a), t)$.

- *Case 1:* $\sigma.(t', a) \notin \mathrm{Pre}(\varphi, t)$. Then, $\mathrm{P}(\sigma.(t', a), t)$ trivially holds.
- *Case 2:* $\sigma.(t', a) \in \mathrm{Pre}(\varphi, t) \wedge \sigma \notin \mathrm{Pre}(\varphi, t')$. Then, $\sigma \notin \mathrm{Pre}(\varphi, t')$, thus, following lemma A.6, $\sigma_s = \sigma_{|\Sigma_\mathrm{u}}$, $\sigma_b = \epsilon$, and $\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}})$. Since $\sigma.(t', a) \in \mathrm{Pre}(\varphi, t)$, and $\sigma \notin \mathrm{Pre}(\varphi, t')$, there exists $t'' \in \mathbb{R}_{\geqslant 0}$ s.t. $t' \leqslant t'' \leqslant t$, and $\mathrm{G}(\mathrm{Reach}(\mathrm{obs}(\sigma.(t', a), t'')_{|\Sigma_\mathrm{u}}, t''), \Pi_\Sigma (\mathrm{obs}(\sigma.(t', a), t'')_{|\Sigma_\mathrm{c}})) \neq \emptyset$. Since $t'' \geqslant t' = \mathrm{time}(\sigma.(t', a))$, then $\mathrm{obs}(\sigma.(t', a), t'') = \sigma.(t', a)$. This means that $\mathrm{G}(\mathrm{Reach}((\sigma.(t', a))_{|\Sigma_\mathrm{u}}, t''), \Pi_\Sigma((\sigma.(t', a))_{|\Sigma_\mathrm{c}})) \neq \emptyset$.
  - If $a \in \Sigma_\mathrm{u}$, then considering that $(\sigma.(t', a))_{|\Sigma_\mathrm{u}} = \sigma_{|\Sigma_\mathrm{u}}.(t', a) = \sigma_s.(t', a)$, $\sigma_b = \epsilon$, and $\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}})$, this means that $\mathrm{G}(\mathrm{Reach}(\sigma_s.(t', a), t''), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c) \neq \emptyset$. Thus, $t_1 \leqslant t'' \leqslant t$, meaning that $\sigma_d -_{\mathrm{t}} t_1 \in \mathrm{G}(\mathrm{Reach}(\sigma_s.(t', a), t_1), \Pi_\Sigma(\sigma_b).\sigma_c)$. Thus, considering the definition of G, it follows that $\mathrm{nobs}(\sigma_d, t) -_{\mathrm{t}} t \in \mathrm{G}(\mathrm{Reach}(\sigma_s.(t', a).\mathrm{obs}(\sigma_d, t), t), \Pi_\Sigma(\mathrm{obs}(\sigma_d, t))^{-1}.(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c))$. Moreover, $\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c = \sigma_{|\Sigma_\mathrm{c}}$, thus $\Pi_\Sigma(\mathrm{obs}(\sigma_d, t))^{-1}.(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c) = \Pi_\Sigma(\mathrm{nobs}(\sigma_d, t)).\sigma_e$, meaning that $\mathrm{nobs}(\sigma_d, t) -_{\mathrm{t}} t \in \mathrm{G}(\mathrm{Reach}(\sigma_t, t), \Pi_\Sigma(\mathrm{nobs}(\sigma_d, t)).\sigma_e)$. Thus, $\mathrm{P}(\sigma.(t', a), t)$ holds.
  - Otherwise, $a \in \Sigma_\mathrm{c}$. Then, $(\sigma.(t', a))_{|\Sigma_\mathrm{u}} = \sigma_{|\Sigma_\mathrm{u}} = \sigma_s$, $\sigma_b = \epsilon$ and $\sigma_c = \Pi_\Sigma((\sigma\,.\,(t', a))_{|\Sigma_\mathrm{c}}) = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}).a$. This means that $\mathrm{G}(\mathrm{Reach}(\sigma_s, t''), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a) \neq \emptyset$. Thus, $t_2 \leqslant t'' \leqslant t$, therefore $\sigma_d -_{\mathrm{t}} t_2 \in \mathrm{G}(\mathrm{Reach}(\sigma_s, t_2), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'))\,.\,\sigma_c\,.\,a)$. It follows that $\mathrm{nobs}(\sigma_d, t) -_{\mathrm{t}} t \in \mathrm{G}(\mathrm{Reach}(\sigma_s\,.\,\mathrm{obs}(\sigma_d, t), t), \Pi_\Sigma(\mathrm{obs}(\sigma_d, t))^{-1}\,.\,(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'))\,.\,\sigma_c\,.\,a))$. Moreover, $\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a = \Pi_\Sigma((\sigma.(t', a))_{|\Sigma_\mathrm{c}}) = \Pi_\Sigma(\sigma_d).\sigma_e$. Thus, $\Pi_\Sigma(\mathrm{obs}(\sigma_d, t))^{-1}\,.\,(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'))\,.\,\sigma_c\,.\,a) = \Pi_\Sigma(\mathrm{nobs}(\sigma_d, t))\,.\,\sigma_e$. Thus, $\mathrm{nobs}(\sigma_d, t) -_{\mathrm{t}} t \in \mathrm{G}(\mathrm{Reach}(\sigma_t, t), \Pi_\Sigma(\mathrm{nobs}(\sigma_d, t)).\sigma_e)$. This means that $\mathrm{P}(\sigma.(t', a), t)$ holds.

  Thus, if $\sigma.(t', a) \in \mathrm{Pre}(\varphi, t) \wedge \sigma \notin \mathrm{Pre}(\varphi, t')$, $\mathrm{P}(\sigma, t) \implies \mathrm{P}(\sigma.(t', a), t)$.
- *Case 3:* $\sigma.(t', a) \in \mathrm{Pre}(\varphi, t)$ and $\sigma \in \mathrm{Pre}(\varphi, t')$. Then, consider $w_b = \mathrm{nobs}(\sigma_b, t') -_{\mathrm{t}} t'$. By the induction hypothesis, since $\sigma \in \mathrm{Pre}(\varphi, t')$, $\mathrm{E}_\varphi(\sigma) \models \varphi$ and $w_b \in \mathrm{G}(\mathrm{Reach}(\sigma_s, t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c)$.
  - If $a \in \Sigma_\mathrm{u}$, then since $w_b \in \mathrm{G}(\mathrm{Reach}(\sigma_s, t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'))\,.\,\sigma_c)$, $\mathrm{Reach}(\sigma_s, t')$ after $(w_b, 0) = \mathrm{Reach}(\sigma_s, t') \in \mathrm{S}(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c)$. Thus, following lemma A.8, since

$a \in \Sigma_u$, $\mathrm{Reach}(\sigma_s, t')$ after $(0, a) = \mathrm{Reach}(\sigma_s.(t', a)) \in \mathrm{S}(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c) \cup \mathrm{I}(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c)$. Then, following lemma A.9, this means that $\mathrm{G}(\mathrm{Reach}(\sigma_s.(t', a)), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t)).\sigma_c) \neq \emptyset$. It follows that $t_1 = t'$, thus $\min(\{t, t_1\}) = t_1 = t'$, and $\sigma_d -_t t' \in \mathrm{G}(\mathrm{Reach}(\sigma_s\ .\ (t', a), t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t'))\ .\ \sigma_c)$. This implies that $\mathrm{Reach}(\sigma_s\ .\ (t', a)\ .\ \sigma_d) = \mathrm{Reach}(\mathrm{E}_\varphi(\sigma\ .\ (t', a))) \in F_G$, meaning that $\mathrm{E}_\varphi(\sigma.(t', a)) \models \varphi$. Moreover, following the definition of G, $\mathrm{nobs}(\sigma_d, t) -_t t \in \mathrm{G}(\mathrm{Reach}(\sigma_s.(t', a).\mathrm{obs}(\sigma_d, t), g), \Pi_\Sigma(\mathrm{obs}(\sigma_d,\ \ t))^{-1}\ .\ (\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c))$. Thus, since $\sigma_t = \sigma_s.(t', a).\mathrm{obs}(\sigma_d, t)$ and $\Pi_\Sigma(\sigma_d).\sigma_e = \Pi_\Sigma(\mathrm{nobs}(\sigma_b,\ t')).\sigma_c$, it follows that $\mathrm{nobs}(\sigma_d, t) -_t t \in \mathrm{G}(\mathrm{Reach}(\sigma_t, t), \Pi_\Sigma(\mathrm{nobs}(\sigma_d, t)).\sigma_e)$. This means that $\mathrm{P}(\sigma.(t', a), t)$ holds.

- Otherwise, $a \in \Sigma_c$. Since $w_b \in \mathrm{G}(\mathrm{Reach}(\sigma_s, t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c)$, w_b satisfies $\Pi_\Sigma(w_b) \preccurlyeq \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c \preccurlyeq \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a$, $\mathrm{Reach}(\sigma_s, t')$ after $w_b \in F_G$, and for all $t'' \in \mathbb{R}_{\geqslant 0}$, $\mathrm{Reach}(\sigma_s, t')$ after $(w_b, t'') \in \mathrm{S}(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c)$. Since $\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c \preccurlyeq \Pi_\Sigma(\mathrm{nobs}(\sigma_b,\ t')).\sigma_c.a$, $\mathrm{S}(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c) \subseteq \mathrm{S}(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a)$. Thus, for all $t'' \in \mathbb{R}_{\geqslant 0}$, $\mathrm{Reach}(\sigma_s,\ t')$ after $(w_b, t'') \in \mathrm{S}(\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a)$. This means that $w_b \in \mathrm{G}(\mathrm{Reach}(\sigma_s, t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_b,\ t'))\ .\ \sigma_c\ .\ a)$. It follows that $\mathrm{G}(\mathrm{Reach}(\sigma_s, t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a) \neq \emptyset$, and thus, using the same reasoning as in the case where $a \in \Sigma_u$, $t_2 = t'$, and σ_d is s.t. $\mathrm{Reach}(\sigma_s, t')$ after $\sigma_d \in F_G$, meaning that $\mathrm{E}_\varphi(\sigma.(t', a)) \models \varphi$, and $\mathrm{nobs}(\sigma_d, t) -_t t \in \mathrm{G}(\mathrm{Reach}(\sigma_t, t), \Pi_\Sigma(\mathrm{nobs}(\sigma_d, t)).\sigma_e)$. Thus, $\mathrm{P}(\sigma.(t', a), t)$ holds.

Thus, in all cases, for all $t \geqslant t'$, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.(t', a), t)$. This means that $\mathrm{P}(\sigma) \implies \forall t \geqslant t', \mathrm{P}(\sigma.(t', a), t)$. Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.(t', a))$. By induction, for all $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. In particular, for all $(\sigma, t) \in \mathrm{Pre}(\varphi)$, $\mathrm{E}_\varphi(\sigma) \models \varphi$. This means that E_φ is sound in $\mathrm{Pre}(\varphi)$.

□

Proposition 4.3. E_φ *is compliant as per Definition 4.3.*

Proof. For $\sigma \in \mathrm{tw}(\Sigma)$, let $\mathrm{P}(\sigma)$ be the predicate: '$\forall t \geqslant \mathrm{time}(\sigma), (\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t) \implies \sigma_{s|\Sigma_u} = \sigma_{|\Sigma_u} \wedge \Pi_\Sigma(\sigma_{s|\Sigma_c}.\mathrm{nobs}(\sigma_b, t)).\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_c}) \wedge \sigma_{s|\Sigma_c} \preccurlyeq_d \sigma_{|\Sigma_c}$.' Let us prove by induction that for all $\sigma \in \mathrm{tw}(\Sigma), \mathrm{P}(\sigma)$ holds.

— *Induction basis:* For $\sigma = \epsilon$. $\mathrm{store}_\varphi(\epsilon) = (\epsilon, \epsilon, \epsilon)$, and $\epsilon_{|\Sigma_c} = \epsilon_{|\Sigma_u} = \Pi_\Sigma(\epsilon) = \epsilon$. Thus, $\mathrm{P}(\epsilon)$ trivially holds.

— *Induction step:* Suppose now that for some $\sigma \in \mathrm{tw}(\Sigma), \mathrm{P}(\sigma)$ holds. Let us consider (t', a) s.t. $\sigma.(t', a) \in \mathrm{tw}(\Sigma)$, $t \geqslant \mathrm{time}(\sigma)$, $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t')$ and $(\sigma_t, \sigma_d, \sigma_e) = \mathrm{store}_\varphi(\sigma.(t', a), t)$. Then, by induction hypothesis, $\sigma_{s|\Sigma_u} = \sigma_{|\Sigma_u}$, $\Pi_\Sigma(\sigma_{s|\Sigma_c}.\sigma_b).\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_c})$ and $\sigma_{s|\Sigma_c} \preccurlyeq_d \sigma_{|\Sigma_c}$.

  – $a \in \Sigma_u$. By construction, σ_d satisfies $\Pi_\Sigma(\sigma_d) \preccurlyeq \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c$ and $\sigma_d \neq \epsilon \implies \mathrm{date}(\sigma_d(1)) \geqslant t'$.

    - *Projection on* Σ_u: Since $a \in \Sigma_u$, $\sigma_{t|\Sigma_u} = (\sigma_s.(t', a).\mathrm{obs}(\sigma_d, t))_{|\Sigma_u}$. $\sigma_d \in \mathrm{tw}(\Sigma_c)$, thus $\sigma_{t|\Sigma_u} = \sigma_{s|\Sigma_u}.(t', a) = \sigma_{|\Sigma_u}.(t', a) = (\sigma.(t', a))_{|\Sigma_u}$.
    - *Projection on* Σ_c: $\Pi_\Sigma(\sigma_{t|\Sigma_c}\ .\ \mathrm{nobs}(\sigma_d,\ \ t))\ .\ \sigma_e = \Pi_\Sigma((\sigma_s\ .\ (t', a)\ .\ \mathrm{obs}(\sigma_d,\ \ t))_{|\Sigma_c}\ .\ \mathrm{nobs}(\sigma_d,\ \ t))\ .\ \sigma_e = \Pi_\Sigma(\sigma_{s|\Sigma_c}\ .\ \sigma_d)\ .\ \sigma_e = \Pi_\Sigma(\sigma_{s|\Sigma_c}).\Pi_\Sigma(\sigma_d)\ .\ \sigma_e$. By construction, $\Pi_\Sigma(\sigma_d)\ .\ \sigma_e = \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\ \ \sigma_c$. Thus, $\Pi_\Sigma(\sigma_{t|\Sigma_c}.\sigma_d).\sigma_e = \Pi_\Sigma(\sigma_{s|\Sigma_c})$.

$\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c = \Pi_\Sigma(\sigma_{s|\Sigma_c}.\mathrm{nobs}(\sigma_b, t')).\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_c}) = \Pi_\Sigma((\sigma.(t', a))_{|\Sigma_c})$. Moreover, $\sigma_t \in \mathrm{tw}(\Sigma)$, and since $\sigma_t = \sigma_s.(t', a).\mathrm{obs}(\sigma_d, t)$, it follows that for all $i \in [1; |\mathrm{obs}(\sigma_d, t)|]$, $\mathrm{date}(\sigma_d(i)) \geqslant t'$. Since $\sigma_{s|\Sigma_c} \preccurlyeq_d \sigma_{|\Sigma_c}$, for all $i \in [1; |\sigma_{s|\Sigma_c}|]$, $\mathrm{date}(\sigma_{s|\Sigma_c}(i)) \geqslant \mathrm{date}(\sigma_{|\Sigma_c}(i))$. Thus, for all $i \in [1; |\sigma_{t|\Sigma_c}|]$, $\mathrm{date}(\sigma_{t|\Sigma_c}(i)) \geqslant$ date $(\sigma_{|\Sigma_c}(i))$. Since $\Pi_\Sigma(\sigma_{t|\Sigma_c}.\sigma_d).\sigma_e = \Pi_\Sigma(\sigma_{|\Sigma_c})$, $\Pi_\Sigma(\sigma_{t|\Sigma_c}) \preccurlyeq \Pi_\Sigma(\sigma_{|\Sigma_c})$. Thus, $\sigma_{t|\Sigma_c} \preccurlyeq_d \sigma_{|\Sigma_c} = (\sigma.(t', a))_{|\Sigma_c}$.

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

– $a \in \Sigma_c$. By construction, σ_d satisfies $\Pi_\Sigma(\sigma_d) \preccurlyeq \Pi_\Sigma(\sigma_b).\sigma_c.a$, and $\sigma_d \neq \epsilon \implies \mathrm{date}(\sigma_d(1)) \geqslant t'$.

- *Projection on* Σ_u: $\sigma_{t|\Sigma_u} = (\sigma_s\,.\,\mathrm{obs}(\sigma_d, t))_{|\Sigma_u}$. Since $\sigma_d \in \mathrm{tw}(\Sigma_c)$, $\sigma_{t|\Sigma_u} = \sigma_{s|\Sigma_u} = \sigma_{|\Sigma_u} = (\sigma.(t', a))_{|\Sigma_u}$.
- *Projection on* Σ_c: $\Pi_\Sigma(\sigma_{t|\Sigma_c}\,.\,\mathrm{nobs}(\sigma_d,\ t))\,.\,\sigma_e = \Pi_\Sigma((\sigma_s\,.\,\mathrm{obs}(\sigma_d,\ t))_{|\Sigma_c}\,.\,\mathrm{nobs}(\sigma_d,\ t))\,.\,\sigma_e = \Pi_\Sigma(\sigma_{s|\Sigma_c}\,.\,\sigma_d)\,.\,\sigma_e = \Pi_\Sigma(\sigma_{s|\Sigma_c})\,.\,\Pi_\Sigma(\sigma_d)\,.\,\sigma_e$. By construction, it is ensured that $\Pi_\Sigma(\sigma_d).\sigma_e = \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a$. It follows that $\Pi_\Sigma(\sigma_{t|\Sigma_c}.\sigma_d).\sigma_e = \Pi_\Sigma(\sigma_{s|\Sigma_c}).\Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a = \Pi_\Sigma(\sigma_{s|\Sigma_c}.\mathrm{nobs}(\sigma_b, t')).\sigma_c.a = \Pi_\Sigma(\sigma_{|\Sigma_c}).a = \Pi_\Sigma((\sigma.(t', a))_{|\Sigma_c})$. Moreover, considering t_2 as defined in Definition 4.6, $t_2 \geqslant t'$, and $t \geqslant t'$, thus $\min(\{t, t_2\}) \geqslant t'$, which means that since there exists $w_d \in \mathrm{tw}(\Sigma)$ s.t. $\sigma_d = w_d +_t \min(\{t, t_2\})$, if $\sigma_d \neq \epsilon$, then $\mathrm{date}(\sigma_d(1)) \geqslant t'$. Thus, for all $i \in [1; |\sigma_d|]$, $\mathrm{date}(\sigma_d(i)) \geqslant t' = \mathrm{time}(\sigma.(t', a))$. This still holds if $\sigma_d = \epsilon$, because then $[1; |\sigma_d|] = \varnothing$. Since $\sigma_{s|\Sigma_c} \preccurlyeq_d \sigma_{|\Sigma_c}$, for all $i \in [1; |\sigma_{s|\Sigma_c}|]$, $\mathrm{date}(\sigma_{s|\Sigma_c}(i)) \geqslant \mathrm{date}(\sigma_{|\Sigma_c}(i))$. Thus, for all $i \in [1; |\sigma_{t|\Sigma_c}|]$, $\mathrm{date}(\sigma_{t|\Sigma_c}(i)) \geqslant \mathrm{date}((\sigma\,.\,(t', a))_{|\Sigma_c}(i))$. Since $\Pi_\Sigma(\sigma_{t|\Sigma_c}\,.\,\mathrm{nobs}(\sigma_d,\ t))\,.\,\sigma_e = \Pi_\Sigma((\sigma.(t', a))_{|\Sigma_c})$, $\Pi_\Sigma(\sigma_{t|\Sigma_c}) \preccurlyeq \Pi_\Sigma((\sigma.(t', a))_{|\Sigma_c})$. Thus $\sigma_{t|\Sigma_c} \preccurlyeq_d (\sigma.(t', a))_{|\Sigma_c}$.

Thus, if $a \in \Sigma_c$, $\mathrm{P}(\sigma.(t, a))$ holds.

Thus, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma.(t, a))$. By induction, for all $\sigma \in \mathrm{tw}(\Sigma)$, for all $t \geqslant \mathrm{time}(\sigma)$, $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t) \implies \sigma_{s|\Sigma_u} = \sigma_{|\Sigma_u} \wedge \Pi_\Sigma(\sigma_{s|\Sigma_c}.\mathrm{nobs}(\sigma_b, t)).\sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_c}) \wedge \sigma_{s|\Sigma_c} \preccurlyeq_d \sigma_{s|\Sigma_c}$. Thus, E_φ is compliant. □

Lemma A.10. $\forall \sigma \in \Sigma_c^*, \forall q \in Q, (q \notin \mathrm{S}(\sigma)) \implies (\exists \sigma_u \in \mathrm{tw}(\Sigma_u), (q \text{ after } \sigma_u \notin F_g) \wedge (\forall t > 0, q \text{ after } (\sigma_u, t) \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)) \wedge (\forall \sigma'_u \preccurlyeq \sigma_u, \sigma'_u \neq \epsilon \implies q \text{ after } \sigma'_u \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)))$.

Proof. For $\sigma \in \Sigma_c^*$ and $q \in Q$, let $\mathrm{P}(\sigma, q)$ be the predicate '$\forall \sigma_u \in \mathrm{tw}(\Sigma_u), (q \text{ after } \sigma_u \in F_G) \vee (\exists t > 0, q \text{ after } (\sigma_u, t) \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)) \vee (\exists \sigma'_u \preccurlyeq \sigma_u, \sigma'_u \neq \epsilon \wedge q \text{ after } \sigma'_u \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma))$.' Let us show the contrapositive of the proposition, that is that for all $\sigma \in \Sigma_c^*$, for all $q \in Q$, $(\mathrm{P}(\sigma, q)) \implies (q \in \mathrm{S}(\sigma))$.

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

exists $\sigma'_u \preccurlyeq \sigma_u$ s.t. q after $\sigma'_u \in \mathrm{S}(\epsilon) \cup \mathrm{I}(\epsilon)$. If the last condition holds, since $\mathrm{I}(\epsilon) = \emptyset$, then q after $\sigma'_u \in \mathrm{S}(\epsilon)$. Then, following the definition of $\mathrm{S}(\epsilon)$, since ${\sigma'_u}^{-1}.\sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u})$, it follows that q after σ'_u after ${\sigma'_u}^{-1}.\sigma_u = q$ after $\sigma_u \in F_G$. Thus, for all $\sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u})$, q after $\sigma_u \in F_G$, meaning that $q \in \mathrm{S}(\epsilon)$.

— If $\sigma \neq \epsilon$, there exists $(\sigma', a) \in \Sigma_\mathrm{c}^* \times \Sigma_\mathrm{c}$ s.t. $\sigma = \sigma'.a$. Let us consider $q \in Q$ s.t. $\mathrm{P}(\sigma, q)$ holds. Then, for all $\sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u})$, q after $\sigma_u \in F_G$, or there exists $t > 0$ s.t. q after $(\sigma_u, t) \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$, or there exists $\sigma'_u \preccurlyeq \sigma_u$ s.t. $\sigma'_u \neq \epsilon$ and q after $\sigma'_u \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$. Let X_s and Y_s be s.t. $\mathrm{S}(\sigma) = \mathrm{S}(\sigma'.a) = \mathrm{S}(\sigma') \cup X_s \cup Y_s$, with

- $\forall x \in X_s, \exists i \in \mathrm{I}(\sigma'.a), \exists \delta \in \mathbb{R}_{\geqslant 0}, x \text{ after } (\epsilon, \delta) = i \wedge \forall t \leqslant \delta, x \text{ after } (\epsilon, t) \in X_s$,
- $Y_s \subseteq F_G \wedge \mathrm{up}(Y_s) = Y_s$ and
- $(X_s \cup Y_s) \cap \mathrm{uPred}(\overline{X_s \cup Y_s \cup \mathrm{I}(\sigma'.a)}) = \emptyset$.

X_s and Y_s correspond to the sets X and Y in the definition of $\mathrm{S}(\sigma'.a)$, respectively. Let us consider $X_0 = \{q \text{ after } (\sigma_u, t) \mid \sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u}) \wedge t \in \mathbb{R}_{\geqslant 0} \wedge \forall t' \in]0; t], q \text{ after } (\sigma_u, t') \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma) \wedge \forall \sigma'_u \preccurlyeq \sigma_u, \sigma'_u \neq \epsilon \implies q \text{ after } \sigma'_u \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)\}$ and $Y_0 = \{y \in X_0 \mid \mathrm{up}(y) \subseteq X_0 \cup Y_s\}$. Then, $Y_0 \subseteq X_0$, and $\mathrm{up}(Y_0) = Y_0$. Moreover, if $y \in Y_0$, then $\mathrm{up}(y) \subseteq X_0 \cup Y_s$, and more precisely, $\mathrm{up}(y) \subseteq Y_0 \cup Y_s$, since all the states in $\mathrm{up}(y)$ are also in Y_0 if $y \in Y_0$. Since $\mathrm{up}(Y_s) = Y_s$, either $\mathrm{up}(y) \subseteq Y_0$ or there exists $t \in \mathbb{R}_{\geqslant 0}$ s.t. for all $t' < t$, y after $(\epsilon, t') \in Y_0$ and $\mathrm{up}(y \text{ after } (\epsilon, t)) \subseteq Y_s$. Since $\mathrm{P}(\sigma, q)$ holds, and $Y_s \subseteq F_G$, in both cases, $y \in F_G$, meaning that $Y_0 \subseteq F_G$. Let us now consider $Y = Y_s \cup Y_0$, $X = X_s \cup (X_0 \setminus Y_0)$ and $x \in X$. Let us suppose that $x \notin X_s$, meaning that $x \in X_0 \setminus Y_0$. Following the definition of X_0 and Y_0, this means that there exists $\delta > 0$ and $i \in \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma)$ such that x after $(\epsilon, \delta) = i$, and they can be chosen such that for all $t < \delta$, x after $(\epsilon, t) \in X_0$. Suppose now that $i \in \mathrm{S}(\sigma)$, and more precisely that $i \in Y_s$. Then, $\mathrm{up}(i) \subseteq Y_s$ and $\mathrm{up}(i) \cap \mathrm{uPred}(\overline{X_s \cup Y_s \cup \mathrm{I}(\sigma)}) = \emptyset$, and since for all $t < \delta$, x after $(\epsilon, t) \in X_0$, it follows that $\mathrm{up}(x) \subseteq X_0 \cup Y_s$, meaning that $x \in Y_0$, which is absurd. Thus, $i \notin Y_s$. This means that either $i \in \mathrm{I}(\sigma)$ or $i \in X_s$. Thus, there exists $\delta' \in \mathbb{R}_{\geqslant 0}$ s.t. i after $(\epsilon, \delta') \in \mathrm{I}(\sigma)$ and for all $t < \delta'$, i after $(\epsilon, t) \in X_s \subseteq X$ (if $i \in \mathrm{I}(\sigma)$, then $\delta' = 0$). Then, x after $(\epsilon, \delta + \delta') = i$, and for all $t < \delta + \delta'$, x after $(\epsilon, t) \in X$. Moreover, $(X \cup Y) \cap \mathrm{uPred}(\overline{X \cup Y \cup \mathrm{I}(\sigma)}) = \emptyset$ since $Y = Y_s \cup Y_0 \subseteq \mathrm{S}(\sigma) \cup X_0$, $X \subseteq X_s \cup X_0$, and $X \cup Y = X_0 \cup \mathrm{S}(\sigma)$. This means that $X \cup Y \subseteq \mathrm{S}(\sigma)$ and since $X_0 \subseteq X \cup Y$, $X_0 \subseteq \mathrm{S}(\sigma)$. Since $q = q$ after $(\epsilon, 0)$ with $\epsilon \in \mathrm{tw}(\Sigma_\mathrm{u})$ and $t \in \mathbb{R}_{\geqslant 0}$, $q \in X_0$ and thus $q \in \mathrm{S}(\sigma)$. Thus, if $\sigma \neq \epsilon$ and $q \in Q$, $\mathrm{P}(\sigma, q) \implies q \in \mathrm{S}(\sigma)$.

Thus, for all $\sigma \in \Sigma_\mathrm{c}^*$, for all $q \in Q$, $\mathrm{P}(\sigma, q) \implies q \in \mathrm{S}(\sigma)$. Thus, the contrapositive also holds, meaning that for all $\sigma \in \Sigma_\mathrm{c}^*$, for all $q \in Q$, $q \notin \mathrm{S}(\sigma) \implies \neg\mathrm{P}(\sigma, q)$, that is $q \notin \mathrm{S}(\sigma) \implies (\exists \sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u}), q \text{ after } \sigma_u \notin F_G \wedge \forall t > 0, q \text{ after } (\sigma_u, t) \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma) \wedge \forall \sigma'_u \preccurlyeq \sigma_u, \sigma'_u \neq \epsilon \implies q \text{ after } \sigma'_u \notin \mathrm{S}(\sigma) \cup \mathrm{I}(\sigma))$. □

Proposition 4.4. E_φ *is optimal in* $\mathrm{Pre}(\varphi)$ *as per Definition 4.4.*

Proof. Let us consider $E' : \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geqslant 0} \to \mathrm{tw}(\Sigma)$, that is compliant with respect to Σ_c and Σ_u. Let us also consider $\sigma \in \mathrm{tw}(\Sigma)$ and (t', a) s.t. $\sigma.(t', a) \in \mathrm{tw}(\Sigma)$. Suppose now that $(\sigma, t') \in \mathrm{Pre}(\varphi)$, $E'(\sigma, t') = \mathrm{E}_\varphi(\sigma, t')$ and that $\mathrm{E}_\varphi(\sigma.(t', a)) \prec_\mathrm{d} E'(\sigma.(t', a))$. Consider $(\sigma_s, \sigma_b, \sigma_c) = \mathrm{store}_\varphi(\sigma, t')$, and $(\sigma_t, \sigma_d, \sigma_e) = \mathrm{store}_\varphi(\sigma.(t', a), t)$, where t is s.t. $\sigma_t =$

$\mathrm{E}_\varphi(\sigma.(t',a))$. Then, considering proof of soundness, since $(\sigma, t') \in \mathrm{Pre}(\varphi)$, $\mathrm{nobs}(\sigma_b, t') -_{\mathrm{t}} t' \in \mathrm{G}(\mathrm{Reach}(\sigma_s, t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c)$.

— If $a \in \Sigma_{\mathrm{u}}$, this means that $\sigma_d -_{\mathrm{t}} t' \in \mathrm{G}(\mathrm{Reach}(\sigma_s.(t',a)), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c)$. Let us consider $q = \mathrm{Reach}(\sigma_s.(t',a))$ and $\mathit{buff}_{\mathrm{c}} = \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c$. Then, $\sigma_d -_{\mathrm{t}} t' = \min_{\mathrm{lex}}(\max_{\preccurlyeq}(\mathrm{G}(q, \mathit{buff}_{\mathrm{c}})))$. E' is compliant with respect to Σ_{c} and Σ_{u}, thus, since $\mathrm{E}_\varphi(\sigma, t') = E'(\sigma, t')$, there exists $\sigma_{d2} \in \mathrm{tw}(\Sigma)$ s.t. $E'(\sigma.(t',a)) = \sigma_s.(t',a).\sigma_{d2}$. Since $\mathrm{E}_\varphi(\sigma.(t',a)) \prec_{\mathrm{d}} E'(\sigma.(t',a))$, then $\sigma_d \prec_{\mathrm{d}} \sigma_{d2}$, thus $\sigma_d -_{\mathrm{t}} t' \prec_{\mathrm{d}} \sigma_{d2} -_{\mathrm{t}} t' = w_{d2}$, meaning that $w_{d2} \notin \mathrm{G}(q, \mathit{buff}_{\mathrm{c}})$. Then, following the definitions of G and S, there are several cases:

- $\Pi_\Sigma(w_{d2}) \not\preccurlyeq \mathit{buff}_{\mathrm{c}}$. But, since E' is compliant, and $E'(\sigma) = \mathrm{E}_\varphi(\sigma)$, this is not possible.
- q after $w_{d2} \notin F_G$, meaning that $E'(\sigma.(t',a)) \not\models \varphi$.
- There exists $t'' \in \mathbb{R}_{\geqslant 0}$ s.t. q after $(w_{d2}, t'') \notin \mathrm{S}(\Pi_\Sigma(\mathrm{obs}(w_{d2}, t''))^{-1}.\mathit{buff}_{\mathrm{c}})$. Let us then note $\mathit{buff}_{\mathrm{c}2} = \Pi_\Sigma(\mathrm{obs}(w_{d2}), t'')^{-1}.\mathit{buff}_{\mathrm{c}}$ and $q_2 = q$ after (w_{d2}, t''). Then, following lemma A.10, there exists $\sigma_u \in \mathrm{tw}(\Sigma_{\mathrm{u}})$ s.t. q_2 after $\sigma_u \notin F_G$, for all $t > 0$, q after $(\sigma_u, t) \notin \mathrm{S}(\mathit{buff}_{\mathrm{c}2}) \cup \mathrm{I}(\mathit{buff}_{\mathrm{c}2})$, and for all $\sigma'_u \preccurlyeq \sigma_u$, $\sigma'_u \neq \epsilon \implies q_2$ after $\sigma'_u \notin \mathrm{S}(\mathit{buff}_{\mathrm{c}2}) \cup \mathrm{I}(\mathit{buff}_{\mathrm{c}2})$. Then, considering that E' is compliant, either $E'(\sigma.(t',a).(\sigma_u +_{\mathrm{t}} t'')) = \sigma_s.(t',a).\,\mathrm{obs}(w_{d2} +_{\mathrm{t}} t', t'').(\sigma_u +_{\mathrm{t}} t'')$, meaning that $E'(\sigma.(t',a).\sigma_u) \not\models \varphi$, or there exists $\sigma'_u \preccurlyeq \sigma_u$, $w_{d3} \neq \epsilon$ such that $\Pi_\Sigma(w_{d3}) \preccurlyeq \Pi_\Sigma(\mathit{buff}_{\mathrm{c}2})$ and $\mathrm{Reach}(E'(\sigma.(t',a).(\sigma'_u +_{\mathrm{t}} (t' + t'')))) = q_2$ after σ'_u after w_{d3}. Since $\sigma'_u \preccurlyeq \sigma_u$, q_2 after $(\sigma'_u, \mathrm{date}(w_{d3}(1))) \notin \mathrm{S}(\mathit{buff}_{\mathrm{c}2}) \cup \mathrm{I}(\mathit{buff}_{\mathrm{c}2})$. Considering the definition of I, q_2 after σ'_u after $w_{d3}(1) \notin \mathrm{S}(\Pi_\Sigma(w_{d3}(1))^{-1}.\mathit{buff}_{\mathrm{c}2}) \cup \mathrm{I}(\Pi_\Sigma(w_{d3}(1))^{-1}.\mathit{buff}_{\mathrm{c}2})$, because otherwise q_2 after $\sigma'_u \in \mathrm{Pred}_{w_{d3}(1)}(\mathrm{S}(\Pi_\Sigma(w_{d3}(1))^{-1}.\mathit{buff}_{\mathrm{c}2}) \cup \mathrm{I}(\Pi_\Sigma(w_{d3}(1))^{-1}.\mathit{buff}_{\mathrm{c}2})) = \mathrm{I}(\mathit{buff}_{\mathrm{c}2})$, which is wrong. In the same way, q_2 after σ'_u after $(w_{d3}, \mathrm{date}(w_{d3}(1))) \notin \mathrm{S}(\Pi_\Sigma(\mathrm{obs}(w_{d3}, \mathrm{date}(w_{d3}(1))))^{-1}.\,\mathit{buff}_{\mathrm{c}2}) \cup \mathrm{I}(\Pi_\Sigma(\mathrm{obs}(w_{d3}, \mathrm{date}(w_{d3}(1))))^{-1}.\,\mathit{buff}_{\mathrm{c}2})$. Thus, since it is not in S, we can find again a word in $\mathrm{tw}(\Sigma_{\mathrm{u}})$ 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 output will never reach S nor I, and since E' can only output a limited number of controllable events (no more than $|\mathit{buff}_{\mathrm{c}}|$), 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 A.10, there would be $\sigma_{\mathrm{ug}} \in \mathrm{tw}(\Sigma_{\mathrm{u}})$ s.t. $E'(\sigma.(t',a).\sigma_{\mathrm{ug}}) \not\models \varphi$.

Thus, if $a \in \Sigma_{\mathrm{u}}$, there exists $\sigma_u \in \mathrm{tw}(\Sigma_{\mathrm{u}})$ such that $E'(\sigma.(t',a).\sigma_u) \not\models \varphi$.

— If $a \in \Sigma_{\mathrm{c}}$, then since $(\sigma, t') \in \mathrm{Pre}(\varphi)$, $\sigma_d -_{\mathrm{t}} t' \in \mathrm{G}(\mathrm{Reach}(\sigma_s, t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a)$. Considering $q = \mathrm{Reach}(\sigma_s)$ and $\mathit{buff}_{\mathrm{c}} = \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a$, the previous proof (when $a \in \Sigma_{\mathrm{u}}$) still holds. Thus, if $a \in \Sigma_{\mathrm{c}}$, there also exists $\sigma_u \in \mathrm{tw}(\Sigma_{\mathrm{u}})$ s.t. $E'(\sigma.(t',a).\sigma_u) \not\models \varphi$.

This means that whenever $E'(\sigma) = \mathrm{E}_\varphi(\sigma) \wedge \mathrm{E}_\varphi(\sigma.(t',a)) \prec_{\mathrm{d}} E'(\sigma.(t',a))$, then there exists $\sigma_u \in \Sigma_{\mathrm{u}}$ s.t. $E'(\sigma.(t',a).\sigma_u) \not\models \varphi$. Thus, E_φ is optimal. □

Proposition 4.5. *The output of $\mathcal{E}$ for input σ is* $\mathrm{E}_\varphi(\sigma)$.

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

— $C^{\mathcal{E}} = \mathrm{tw}(\Sigma) \times \Sigma_{\mathrm{c}}^* \times Q \times \mathbb{R}_{\geqslant 0} \times \{\top, \bot\}$ is the set of configurations.

— $c_0^{\mathcal{E}} = \langle \epsilon, \epsilon, q_0, 0, \bot \rangle \in C^{\mathcal{E}}$ is the initial configuration.

— $\Gamma^{\mathcal{E}} = ((\mathbb{R}_{\geqslant 0} \times \Sigma) \cup \{\epsilon\}) \times Op \times ((\mathbb{R}_{\geqslant 0} \times \Sigma) \cup \{\epsilon\})$ 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 $\{\text{compute}, \text{dump}, \text{pass-uncont}, \text{store-cont}, \text{delay}\}$.

Let us also introduce some specific notation. For a sequence of rules $w \in \Gamma^{\mathcal{E}*}$, we note $\text{input}(w) = \Pi_1(w(1)).\Pi_1(w(2))\ldots\Pi_1(w(|w|))$ the concatenation of all inputs from w. In the same way, we define $\text{output}(w) = \Pi_3(w(1)).\Pi_3(w(2))\ldots\Pi_3(w(|w|))$ the concatenation of all outputs from w. Since all configurations are not reachable from $c_0^{\mathcal{E}}$, for a word $w \in \Gamma^{\mathcal{E}*}$, we will say that $\text{Reach}(w) = c$ if $c_0^{\mathcal{E}} \overset{w}{\hookrightarrow}_{\mathcal{E}} c$, or $\text{Reach}(w) = \bot$ 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:

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

Since time is not discrete, the rule delay can be applied an infinite number of times by slicing time. Thus, we consider that the rule delay is always applied a minimum number of times, i.e., when two rules delay are consecutive, they are merged into one rule delay, whose parameter is the sum of the parameters of the two rules. The runs obtained are equivalent, but it allows to consider the maximum (for prefix order) of the set used in the definition of Rules. We then extend output to timed words with a date: for $\sigma \in \text{tw}(\Sigma)$, and a date t, $\text{output}(\sigma, t) = \text{output}(\text{Rules}(\sigma, t))$. For $\sigma \in \text{tw}(\Sigma)$ and $t \in \mathbb{R}_{\geqslant 0}$, let $\text{P}(\sigma, t)$ be the predicate, '$\text{E}_\varphi(\sigma, t) = \text{output}(\sigma, t) \wedge (((\sigma_s, \sigma_b, \sigma_c) = \text{store}_\varphi(\text{obs}(\sigma, t), t) \wedge \langle \sigma_b^{\mathcal{E}}, \sigma_c^{\mathcal{E}}, q^{\mathcal{E}}, t, b\rangle = \text{Reach}(\text{Rules}(\sigma, t))) \implies \sigma_b^{\mathcal{E}} = \text{nobs}(\sigma_b, t) \wedge \sigma_c^{\mathcal{E}} = \sigma_c \wedge q^{\mathcal{E}} = \text{Reach}(\sigma_s, t) \wedge (b = \top \implies \text{G}(q^{\mathcal{E}}, \sigma_c^{\mathcal{E}}) \neq \emptyset))$.' Let $\text{P}(\sigma)$ be the predicate '$\forall t \in \mathbb{R}_{\geqslant 0}, \text{P}(\sigma, t)$ holds.' Let us then prove that for all $\sigma \in \text{tw}(\Sigma), \text{P}(\sigma)$ holds.

— *Induction basis*: For $\sigma = \epsilon$, let us consider $t \in \mathbb{R}_{\geqslant 0}$. Then, $\text{store}_\varphi(\epsilon, t) = (\epsilon, \epsilon, \epsilon)$ and $\text{Reach}(\epsilon, t) = \langle l_0, v_0 + t\rangle$. On the other hand, the only rules that can be applied are delay, and possibly compute, since there is not any input, nor any element to dump. Thus, $\text{Rules}(\epsilon, t) = \epsilon/\text{delay}(t)/\epsilon$, or there exists $t' \geqslant t$ s.t. $\text{Rules}(\epsilon, t) = \epsilon/\text{delay}(t')/\epsilon\,.\,\epsilon/\text{compute}()/\epsilon\,.\,\epsilon/\text{delay}(t - t')/\epsilon$. Let us consider $c = \text{Reach}(\text{Rules}(\epsilon, t))$. Then, $c = \langle \epsilon, \epsilon, \langle l_0, v_0 + t\rangle, t, b\rangle$. If rule compute appears in $\text{Rules}(\epsilon, t)$, then $b = \top$, meaning that $\text{G}(q_0 \text{ after } (\epsilon, t'), \epsilon) \neq \emptyset$, and thus that $\text{G}(q_0 \text{ after } (\epsilon, t), \epsilon) \neq \emptyset$ since $t \geqslant t'$. Otherwise $b = \bot$. All the other values remain unchanged between the two cases. In both cases, $\text{output}(\text{Rules}(\epsilon, t)) = \epsilon = \text{E}_\varphi(\epsilon, t)$. Thus, $\text{P}(\epsilon)$ holds.

— *Induction step*: Let us suppose now that for some $\sigma \in \text{tw}(\Sigma), \text{P}(\sigma)$ holds. Let us consider $(t', a) \in \mathbb{R}_{\geqslant 0} \times \Sigma$ s.t. $\sigma.(t', a) \in \text{tw}(\Sigma)$. Let us then prove that $\text{P}(\sigma.(t', a))$ holds. Let us consider $t \in \mathbb{R}_{\geqslant 0}$, $c = \langle \sigma_b^{\mathcal{E}}, \sigma_c^{\mathcal{E}}, q^{\mathcal{E}}, t', b\rangle = \text{Reach}(\text{Rules}(\sigma, t'))$, $(\sigma_s, \sigma_b, \sigma_c) = \text{store}_\varphi(\sigma, t')$ and $(\sigma_t, \sigma_d, \sigma_e) = \text{store}_\varphi(\text{obs}(\sigma.(t', a), t), t)$. If $t < t'$, then $\text{obs}(\sigma.(t', a), t) = \text{obs}(\sigma, t)$ and $\text{P}(\sigma.(t', a), t)$ trivially holds, since $\text{P}(\sigma)$ holds. Thus, in the following, we consider that $t \geqslant t'$, so that $\text{store}_\varphi(\text{obs}(\sigma.(t', a), t), t) = \text{store}_\varphi(\sigma.(t', a), t)$:

   – If $a \in \Sigma_{\text{u}}$, rule pass-uncont can be applied. Let us consider $c' = c \text{ after } ((t', a)/\text{pass-uncont}((t', a))/(t', a))$. Then, $c' = \langle \epsilon, \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}, q', t', \bot\rangle$ with $q' = q^{\mathcal{E}} \text{after} (0, a)$. Then, if $t \geqslant t_1^{\mathcal{E}}$, where $t_1^{\mathcal{E}} = \min(\{t'' \mid t'' \geqslant t' \wedge \text{G}(q' \text{ after } (\epsilon, t'' - t'), \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}) \neq$

$\emptyset\}$), then rule delay$(t_1^{\mathcal{E}} - t')$ can be applied, followed by rule compute. Since $q^{\mathcal{E}} = \mathrm{Reach}(\sigma_s, t')$, $\sigma_b^{\mathcal{E}} = \mathrm{nobs}(\sigma_b, t')$, and $\sigma_c^{\mathcal{E}} = \sigma_c$ (by induction hypothesis), then $\mathrm{G}(q'$ after $(\epsilon, t'' - t'), \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}) = \mathrm{G}(\mathrm{Reach}(\sigma_s.(t', a), t''), \Pi_\Sigma(\mathrm{nobs}(\sigma_b,\ t')).\sigma_c)$, thus $t_1^{\mathcal{E}} = t_1$, where t_1 is defined in Definition 4.6. Thus, c' after $((\epsilon/\ \mathrm{delay}(t_1^{\mathcal{E}} - t')/\epsilon)\ .\ (\epsilon/\ \mathrm{compute}\ /\epsilon)) = \langle \sigma_d^{\mathcal{E}}, \sigma_e^{\mathcal{E}}, q'$ after $(\epsilon, t_1 - t'), t_1, \top\rangle$, with $\sigma_d^{\mathcal{E}} = \kappa_\varphi(q'$ after $(\epsilon, t_1 - t'), \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}) +_{\mathrm{t}} t_1 = \kappa_\varphi(\mathrm{Reach}(\sigma_s.(t', a), t_1), \Pi_\Sigma(\sigma_b).\sigma_c) +_{\mathrm{t}} t_1 = \sigma_d$, and thus $\sigma_e^{\mathcal{E}} = \sigma_e$. Then, rules delay and dump can be applied until date t is reached. In the end, $\mathrm{Reach}(\mathrm{Rules}(\sigma.(t', a), t)) = c'$ after w, where w is composed of an alternation of rules delay and dump, thus $\mathrm{Reach}(\mathrm{Rules}(\sigma.(t', a), t)) = \langle \mathrm{nobs}(\sigma_d^{\mathcal{E}}, t), \sigma_e^{\mathcal{E}}, q'\,\mathrm{after}(\mathrm{obs}(\sigma_d^{\mathcal{E}}, t) -_{\mathrm{t}} t', t - t'), t, \top\rangle = \langle \mathrm{nobs}(\sigma_d, t), \sigma_e, \mathrm{Reach}(\sigma_t, t), t, \top\rangle$. Then, $\mathrm{output}(\mathrm{Rules}(\sigma\ .\ (t', a), t)) = \mathrm{output}(\mathrm{Rules}(\sigma, t'))\ .\ (t', a)\ .\ \mathrm{obs}(\sigma_d^{\mathcal{E}}, t) = \sigma_s\ .\ (t', a)\ .\ \mathrm{obs}(\sigma_d, t) = \sigma_t$. Thus, if $t \geqslant t_1$, $\mathrm{P}(\sigma.(t', a), t)$ holds. Otherwise, $t < t_1$, and then rule dump cannot be applied, since $\Pi_5(c') = \bot$, and rule compute also cannot be applied. Thus, the only rule that can be applied is delay, so that $\mathrm{Reach}(\mathrm{Rules}(\sigma.(t', a), t)) = \langle \epsilon, \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}, q'$ after $(\epsilon, t - t'), t', \bot\rangle$. Since $t < t_1$, this means that $\sigma_d = \epsilon$ and $\sigma_e = \Pi_\Sigma(\sigma_b)\ .\ \sigma_c$. Thus, $\mathrm{output}(\mathrm{Rules}(\sigma\ .\ (t', a), t)) = \mathrm{output}(\mathrm{Rules}(\sigma, t')).(t', a) = \sigma_s.(t', a) = \sigma_t$, and $\sigma_d^{\mathcal{E}} = \sigma_d$, and $\sigma_e^{\mathcal{E}} = \sigma_e$. This means that $\mathrm{P}(\sigma.(t', a), t)$ holds when $t < t_1$. Thus, if $a \in \Sigma_{\mathrm{u}}$, then $\mathrm{P}(\sigma.(t', a), t)$ holds for all $t \geqslant t'$.

– Otherwise, $a \in \Sigma_{\mathrm{c}}$. Then, rule store-cont can be applied. Let us consider $c' = c$ after $((t', a)/\ \mathrm{store\text{-}cont}(a)/\epsilon)$. Then, $c' = \langle \epsilon, \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}.a, q^{\mathcal{E}}, t', \bot\rangle$. Let us consider $t_2^{\mathcal{E}} = \min(\{t'' \in \mathbb{R}_{\geqslant 0} \mid t'' \geqslant t' \wedge \mathrm{G}(q^{\mathcal{E}}$ after $(\epsilon, t'' - t'), \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}.a) \neq \emptyset\})$. Since $\mathrm{G}(q^{\mathcal{E}}$ after $(\epsilon, t'' - t'), \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}.a) = \mathrm{G}(\mathrm{Reach}(\sigma_s, t''), \Pi_\Sigma(\mathrm{nobs}(\sigma_b, t')).\sigma_c.a)$, it follows that $t_2^{\mathcal{E}} = t_2$ as defined in Definition 4.6. If $t \geqslant t_2^{\mathcal{E}} = t_2$, then rule delay$(t_2 - t')$ can be applied, followed by rule compute. Then, c' after $((\epsilon/\ \mathrm{delay}(t_2 - t')/\epsilon).(\epsilon/\ \mathrm{compute}()/\epsilon)) = \langle \sigma_d^{\mathcal{E}}, \sigma_e^{\mathcal{E}}, q$ after $(\epsilon, t_2 - t'), t_2, \top\rangle$, where $\sigma_d^{\mathcal{E}} = \kappa_\varphi(q$ after $(\epsilon, t_2 - t'), \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}.a) +_{\mathrm{t}} t_2 = \kappa_\varphi(\mathrm{Reach}(\sigma_s, t_2), \Pi_\Sigma(\sigma_b).\sigma_c.a) +_{\mathrm{t}} t_2 = \sigma_d$. Then, $\sigma_e^{\mathcal{E}} = \sigma_e$. Then, an alternation of rules delay and dump can be applied until date t is reached. This leads to $\mathrm{Reach}(\mathrm{Rules}(\sigma\ .\ (t',\ a),\ t)) = \langle \mathrm{nobs}(\sigma_d^{\mathcal{E}},\ t),\ \sigma_e^{\mathcal{E}},\ q$ after $(\mathrm{obs}(\sigma_d^{\mathcal{E}},\ t),\ t),\ t,\ \top\rangle = \langle \mathrm{nobs}(\sigma_d,\ t),\ \sigma_e,\ \mathrm{Reach}(\sigma_t,\ t),\ t,\ \top\rangle$. Moreover, $\mathrm{output}(\mathrm{Rules}(\sigma\,.(t',\ a),\ t)) = \mathrm{output}(\sigma, t').\mathrm{obs}(\sigma_d, t) = \sigma_s.\mathrm{obs}(\sigma_d, t) = \mathrm{E}_\varphi(\sigma.(t', a), t)$. Thus, if $t \geqslant t_2$, $\mathrm{P}(\sigma.(t', a), t)$ holds. Otherwise, $t < t_2$, meaning that $\sigma_d^{\mathcal{E}} = \epsilon = \sigma_d$, and $\sigma_e^{\mathcal{E}} = \Pi_\Sigma(\sigma_b^{\mathcal{E}}).\sigma_c^{\mathcal{E}}.a = \Pi_\Sigma(\mathrm{nobs}(\sigma_b,\ t'))\,.\,\sigma_c\,.\,a = \sigma_e$, and $\mathrm{output}(\sigma\,.(t',\ a),\ t) = \mathrm{output}(\sigma,\ t') = \sigma_s = \mathrm{E}_\varphi(\sigma.(t', a), t)$. Thus, $\mathrm{P}(\sigma.(t', a), t)$ holds.

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

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

References

Alur, R. and Dill, D. (1992). The theory of timed automata. In: de Bakker, J., Huizing, C., de Roever, W. and Rozenberg, G. (eds.) *Real-Time: Theory in Practice*, Lecture Notes in Computer Science, vol. 600, Berlin Heidelberg: Springer, 45–73.

Basin, D., Jugé, V., Klaedtke, F. and Zălinescu, E. (2013). Enforceable security policies revisited. *ACM Transactions on Information and System Security*. **16** (1) 3:1–3:26.

Basin, D., Klaedtke, F. and Zalinescu, E. (2011). Algorithms for monitoring real-time properties. In: Khurshid, S. and Sen, K. (eds.) *Proceedings of the 2nd International Conference on Runtime Verification (RV 2011)*, Lecture Notes in Computer Science, vol. 7186, Springer-Verlag, 260–275.

Bloem, R., Könighofer, B., Könighofer, R. and Wang, C. (2015). Shield synthesis: - Runtime enforcement for reactive systems. In: *Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015*, London, UK, 533–548.

Charafeddine, H., El-Harake, K., Falcone, Y. and Jaber, M. (2015). Runtime enforcement for component-based systems. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015*, 1789–1796.

Colombo, C., Pace, G. J. and Schneider, G. (2009a). LARVA — safer monitoring of real-time Java programs (tool paper). In: Hung, D.V. and Krishnan, P. (eds.) *Proceedings of the 7th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2009)*, IEEE Computer Society, 33–37.

Colombo, C., Pace, G. J. and Schneider, G. (2009b). Safe runtime verification of real-time properties. In: Ouaknine, J. and Vaandrager, F.W. (eds.) *Proceedings of the 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2009)*, Lecture Notes in Computer Science, vol. 5813, Springer, 103–117.

Dolzhenko, E., Ligatti, J. and Reddy, S. (2015). Modeling runtime enforcement with mandatory results automata. *International Journal of Information Security* **14** (1) 47–60.

Falcone, Y., Havelund, K. and Reger, G. (2013). A tutorial on runtime verification. In: Broy, M., Peled, D.A. and Kalus, G. (eds.) *Engineering Dependable Software Systems, NATO Science for Peace and Security Series, D: Information and Communication Security*, vol. 34, IOS Press, 141–175.

Falcone, Y., Mounier, L., Fernandez, J. and Richier, J. (2011). Runtime enforcement monitors: Composition, synthesis, and enforcement abilities. *Formal Methods in System Design* **38** (3) 223–262.

Leucker, M. and Schallhart, C. (2009). A brief account of runtime verification. *Journal of Logic Programming* **78** (5) 293–303.

Ligatti, J., Bauer, L. and Walker, D. (2009). Run-time enforcement of nonsafety policies. *ACM Transactions on Information and System Security* **12** (3) 19:1–19:41.

Pinisetty, S., Falcone, Y., Jéron, T. and Marchand, H. (2014a). Runtime enforcement of regular timed properties. In: Cho, Y., Shin, S.Y., Kim, S., Hung, C. and Hong, J. (eds.) *Symposium on Applied Computing, SAC*, Gyeongju, Republic of Korea: ACM, 1279–1286.

Pinisetty, S., Falcone, Y., Jéron, T. and Marchand, H. (2014b). Runtime enforcement of parametric timed properties with practical applications. In: Lesage, J., Faure, J., Cury, J.E.R. and Lennartson, B. (eds.) *Proceedings of the 12th International Workshop on Discrete Event Systems, WODES*, Cachan, France: International Federation of Automatic Control, 420–427.

Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A. and Nguena-Timo, O. (2014c). Runtime enforcement of timed properties revisited. *Formal Methods in System Design* **45** (3) 381–422.

Pinisetty, S., Falcone, Y., Jéron, T., Marchand, H., Rollet, A. and Nguena-Timo, O.L. (2012). Runtime enforcement of timed properties. In: Qadeer, S. and Tasiran, S. (eds.) *Runtime Verification, 3rd International Conference, RV 2012, Revised Selected Papers*, Lecture Notes in Computer Science, vol. 7687, Istanbul, Turkey: Springer, 229–244.

Ramadge, P.J. and Wonham, W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization* **25** (1) 206–230.

Ramadge, P.J. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE* **77** (1) 81–98.

Renard, M., Falcone, Y., Rollet, A., Pinisetty, S., Jéron, T. and Marchand, H. (2015). Enforcement of (timed) properties with uncontrollable events. In: Leucker, M., Rueda, C. and Valencia, F.D. (eds.) *Theoretical Aspects of Computing - ICTAC 2015*, Lecture Notes in Computer Science, vol. 9399, Springer International Publishing, 542–560.

Sammapun, U., Lee, I. and Sokolsky, O. (2005). RT-MaC: Runtime monitoring and checking of quantitative and probabilistic properties. In: *Proceedings of the IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, 0:147–153.

Schneider, F.B. (2000). Enforceable security policies. *ACM Transactions on Information and System Security* **3** (1) 30–50.