Runtime enforcement of timed properties using games

Matthieu Renard, Antoine Rollet, Yliès Falcone

Under consideration for publication in Formal Aspects of Computing

Runtime Enforcement of Timed Properties using Games

Matthieu Renard[1], Antoine Rollet[1] and Yliès Falcone[2]

[1] Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France. first.last@labri.fr
[2] Univ. Grenoble Alpes, CNRS, Inria, Laboratoire d'Informatique de Grenoble, F-38000 France. ylies.falcone@univ-grenobles-alpes.fr

Abstract. This paper deals with runtime enforcement of timed properties with uncontrollable events. Runtime enforcement consists in defining and using an enforcement mechanism that modifies the executions of a running system to ensure their correctness with respect to the desired property. Uncontrollable events cannot be modified by the enforcement mechanisms and thus have to be released immediately. We present a complete theoretical framework for synthesising such mechanism, modelling the runtime enforcement problem as a Büchi game. It permits to pre-compute the decisions of the enforcement mechanism, thus avoiding to explore the whole execution tree at runtime. The obtained enforcement mechanism is sound, compliant and optimal, meaning that it should output as soon as possible correct executions that are as close as possible to the input execution. This framework takes as input any timed regular property modelled by a timed automaton. We present GREP, a tool implementing this approach. We provide algorithms and implementation details of the different modules of GREP, and evaluate its performance. The results are compared with another state of the art runtime enforcement tool.

1. Introduction

Runtime Verification (RV) ([LS09, FHR13, BFFR18, BF18]) is a powerful technique aiming at checking that the current execution of a running system conform to a given specification. In RV, one uses a *monitor* that acts as a decision procedure giving verdicts from a sequence of events gathered from the execution of the system under scrutiny. The monitor is usually synthesised from a property formalising the specification (e.g. an automaton or a temporal logic formula). This sequence is obtained via instrumentation. RV ressembles passive testing ([CGP03, ACC+04]) and permits to obtain alerts in case of wrong behaviour while the system is running.

This paper deals with *Runtime Enforcement* (RE, [Sch00, LBW09, FMFR11, BJKZ13]) of properties, an extension of Runtime Verification. While RV deals with sequence observation, RE deals with sequence transformation. In RE, one uses a so-called *enforcement mechanism/monitor* (EM) which modifies the sequence of events produced by the system, if needed. According to a set of rules, the EM transforms a possibly incorrect input sequence into another one verifying the required property. Most works on RE abstract away implementation details. Then, it generally consists in defining an *input-output* relation describing how to transform a (possibly incorrect) input sequence of events into an output sequence of events using an EM (see Fig. 1). This transformation is done according to the required property which is used in order to synthesise the EM. One of the advantages of RE is that the whole specification of the system under scrutiny is not

Correspondence and offprint requests to: M. Renard, A. Rollet and Y. Falcone

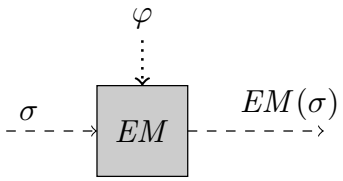


Figure 1. Conceptual view of the runtime enforcement problem.

necessary to generate an EM, only a property that should be satisfied by its output is needed. Depending on the specification formalism of properties, various classes of properties can be enforced on systems, safety properties and transactions for instance; see [Fal10, FFM12, KT12, FMRS18] for overviews and surveys.

Usually, the input-output relation realised by the EM should be *sound* and *transparent*. Soundness and transparency mean that the output sequence should satisfy the required property and that the output should be as close as possible to the input sequence (implying that a correct input sequence should not be modified), respectively. This notion of transparency is rather general since it depends on the definition of "closeness", and contains an implicit notion of *optimality* (which may also be considered separately as in [PFJM14b]).

In case of timed properties, the EM acts as a delaying filter on the input ([PFJ+13, PFJM14b, PFJM15]) ; see [FP19] for a recent overview. It uses a buffer in order to store input events and releases them only at an instant where the satisfaction of the property is ensured (possibly never).

Motivations. This paper deals with RE of timed properties with *uncontrollable* events. In some situations events should be observed by the EM, permitting initiating a reaction if necessary, but without the possibility to modify it. This may happen e.g. in case of a broadcast of an alert, or in case of an interruption message in a system. Such an event is said to be *uncontrollable*. They naturally arise in many concrete scenarios. Uncontrollable events can indeed model some physical event (coming from the environment) that is impossible to prevent, but that the system under scrutiny should observe to react correctly. For instance, in an autonomous car control system, the "crossing line" event could be seen as an uncontrollable event, i.e. only observable by the EM, permitting initiating the necessary trajectory correction. More formally, it is necessary to consider two sets of events: the *controllable* ones that can be modified by the EM (e.g. stored in a buffer as in [PFJ+14]), and the *uncontrollable* ones which cannot.

Challenges. Considering uncontrollable events in the framework implies new challenges. Indeed, the occurrence of uncontrollable events, which should be released instantaneously, may have an impact on the release date of stored events. Thus it is necessary to recompute all of them in such a situation. Moreover, it is necessary to anticipate uncontrollable events, i.e. to compute all the possible reachable (bad) states for any sequence of uncontrollable events. This computing may be costly. Since an EM should react in a reasonable time laps, optimisation strategies should be investigated. Finally, with the classical definitions (e.g. used in [Sch00, LBW09, PFJ+13, PFJM14b, PFJM15]) some properties may never be enforceable. This may arise for instance in case of a sequence of uncontrollable events leading directly in a forbidden state from the initial one. Thus new enforcement approaches are necessary.

Contributions. The first proposition of RE framework for timed properties with uncontrollable events has been proposed in [RFR+17]. Now, we introduce a new RE framework for timed properties using Büchi games (see [GT02] for a detailed description of infinite games, including Büchi games) for generating the EM. This kind of game is adapted since it corresponds to games in which one tries to always be able to reach some nodes called Büchi nodes. An EM tries to always be able to reach an accepting state of the automaton representing the property, even if some uncontrollable events are received, thus it is similar to solving a Büchi game. Using games permits to pre-compute some of the decisions of the EM. It permits to avoid the exploration of the whole execution tree at runtime, thus improving the time overhead of an implementation. A first RE framework using games has been proposed in [RRF17b]. We generalise this approach in case of timed properties. The generated EM is *sound*, *compliant* and *optimal*. Compliance is a weakened variant of

transparency adapted for uncontrollable events. This framework has been implemented in the GREP tool[1]. Performance analysis and comparison with a state-of-the-art tool, TiPEX [PFJM15], show satisfying results.

This paper extends [RRF17a] in which we presented GREP, by providing the complete theoretical framework which has been implemented in GREP. Definitions of soundness, compliance and optimality are given. We also detail the EM synthesis using Büchi games in a timed context. All the proofs of soundness, compliance, optimality and equivalence between the different descriptions of the enforcement mechanism are provided, and we detail implementation aspects of the different modules composing GREP.

Outline. Section 2 introduces preliminary concepts and notations. Section 3 describes the theoretical framework used for the EM generation, based on game theory. Section 4 presents GREP, a tool implementing the approach given in Section 3, providing algorithms, implementation details and performance evaluation. Section 5 discusses some related work. Conclusions and perspectives are drawn in Section 6. Proofs are in appendix A.

2. Preliminaries and Notation

In this section, we describe the notation used in the following parts, and give formal definitions of elements we use, such as words, automata, traces, timed words, and timed automata.

2.1. Untimed Notions

Let Σ be a finite alphabet of actions. We use the standard definitions and notations over finite words and languages. The concatenation of two words w and w' is noted $w \cdot w'$ (or ww' when clear from the context). For two words w and w' of Σ^*, w' is a *prefix* of a word w, noted $w' \preccurlyeq w$, if there exists a word $w'' \in \Sigma^*$ such that $w = w' \cdot w''$. Word w'' is then called the *residual* of w after reading the prefix w'. Formally, we define $0 \notin \Sigma$ such that for any $w \in \Sigma \cup \{0\}$, $w \cdot 0 = 0 \cdot w = 0$, and we write $w'^{-1} \cdot w = w''$ if $w' \cdot w'' = w$ for some unique element w'', or $w'^{-1} \cdot w = 0$ otherwise. In the remaining of the paper, this notation will only be used with prefixes, meaning that the result is never 0, thus it will not be used anymore. Note that $w' \cdot w'' = w' \cdot w'^{-1} \cdot w = w$. These definitions are extended to languages in the natural way. A language $L \subseteq \Sigma^*$ is *extension-closed* if for any words $w \in L$ and $w' \in \Sigma^*$, $w \cdot w' \in L$. Given a word $w = a_1 \cdot a_2 \dots a_{|w|}$ and an integer i such that $1 \leq i \leq |w|$, we note $w(i)$ the i-th element of w, i.e. $w(i) = a_i$. We also note $w_{[..i]}$ the prefix of w of size i: $w_{[..i]} = a_1 \cdot a_2 \dots a_i$.

Given a tuple $e = (e_1, e_2, \dots, e_n)$ of size n, for an integer i such that $1 \leq i \leq n$, we note Π_i the projection on the i-th coordinate, i.e. $\Pi_i(e) = e_i$. The tuple $(e_1, e_2, \dots, e_n)$ is sometimes noted $\langle e_1, e_2, \dots, e_n \rangle$ 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 for any $\sigma \in \Sigma^*$, and any $a \in \Sigma$, $(w \cdot a)_{|\Sigma'} = w_{|\Sigma'} \cdot a$ if $a \in \Sigma'$, or $(w \cdot a)_{|\Sigma'} = w_{|\Sigma'}$ otherwise. We also note $=_{\Sigma'}$ the equality of the restrictions of two words to Σ': for σ and σ' in Σ^*, $\sigma =_{\Sigma'} \sigma'$ if $\sigma_{|\Sigma'} = \sigma'_{|\Sigma'}$. We define in the same way the operator $\preccurlyeq_{\Sigma'}$ by : $\sigma \preccurlyeq_{\Sigma'} \sigma'$ whenever $\sigma_{|\Sigma'} \preccurlyeq \sigma'_{|\Sigma'}$.

2.2. Timed Languages

Let $\mathbb{R}_{\geq 0}$ be the set of non-negative real numbers, and Σ a finite alphabet of actions. An *event* is a pair $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$. We define $\text{delay}((\delta, a)) = \delta$ and $\text{act}((\delta, a)) = a$ the projections of events on delays and actions respectively. A *timed word* over Σ, is a finite sequence of events of $(\mathbb{R}_{\geq 0} \times \Sigma)$. The set of timed words over Σ is denoted $\text{tw}(\Sigma)$. For $\sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n)$, δ_i $(2 \leq i \leq n)$ is the delay between action a_{i-1} and action a_i, and δ_1 is the time elapsed before the first event (i.e. the delay between the beginning of the enforcement process and the occurrence of the first action a_1). We naturally extend the notions of *prefix* and *residual* to timed words.

[1] https://github.com/matthieurenard/GREP

We denote the total time needed to read a timed word σ by $\text{time}(\sigma)$. Formally, $\text{time}(\sigma) = \sum_{i=1}^{|\sigma|} \delta_i$. The *observation* of σ at time t is the timed word noted $\text{obs}(\sigma, t)$ and defined as: $\text{obs}(\sigma, t) = \max_{\preccurlyeq}(\{\sigma' \mid \sigma' \preccurlyeq \sigma \wedge \text{time}(\sigma') \leq t\})$. It corresponds to the word that would be observed at date t when reading σ. We also define the remainder of the observation of σ at date t as $\text{nobs}(\sigma, t) = (\text{obs}(\sigma, t))^{-1} \cdot \sigma$, which corresponds to the events that are to be received after $\text{obs}(\sigma, t)$ when reading σ. Note that the first delay of $\text{nobs}(\sigma, t)$ is relative to the final action of $\text{obs}(\sigma, t)$ and not to t itself.

The *untimed projection* of a timed word σ is noted $\Pi_\Sigma(\sigma)$, and is defined as: $\Pi_\Sigma((\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n)) = a_1 \cdot a_2 \cdots a_n$. It is the sequence of actions of the timed word with dates ignored. For a timed word $\sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n)$, and a delay $t \in \mathbb{R}_{\geq 0}$, σ *delayed by* t is the word noted $\sigma +_\text{t} t$ and such that t is added to the first delay of σ: $\sigma +_\text{t} \delta = (\delta_1 + t, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n)$. Similarly, we define $\sigma -_\text{t} t$, when $\delta_1 \geq t$, as $\sigma -_\text{t} \delta = (\delta_1 - t, a_1) \cdot (\delta_2, a_2) \cdots (\delta_n, a_n)$. We also extend the definition of the restriction of σ to $\Sigma' \subseteq \Sigma$ to timed words, such that only actions belonging to Σ' remain. This operation keeps the *dates* of the events unchanged, not the delays, thus, it is formally defined by induction as follows:

$$\epsilon_{|\Sigma'} = \epsilon,$$

and for $\sigma \in \text{tw}(\Sigma)$ and $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$,

$$(\sigma \cdot (\delta, a))_{|\Sigma'} = \begin{cases} \sigma_{|\Sigma'} \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_{|\Sigma'}), a) & \text{if } a \in \Sigma' \\ \sigma_{|\Sigma'} & \text{otherwise.} \end{cases}$$

Note that to concatenate two restrictions, it is also needed to adjust the delay at the beginning of the second word: for $\sigma \in \text{tw}(\Sigma)$ and $\sigma' \in \text{tw}(\Sigma)$, $(\sigma \cdot \sigma')_{|\Sigma'} = \sigma_{|\Sigma'} \cdot (\sigma'_{|\Sigma'} +_\text{t} (\text{time}(\sigma) - \text{time}(\sigma_{|\Sigma'})))$. The notations $=_{\Sigma'}$ and $\preccurlyeq_{\Sigma'}$ are then naturally extended to timed words.

A *timed language* is any subset of $\text{tw}(\Sigma)$. The notion of *extension-closed* languages is naturally extended to timed languages, i.e. if $L \subseteq \text{tw}(\Sigma)$ is a timed language, L is extension-closed if $L = L \cdot \text{tw}(\Sigma)$. We also extend the notion of extension-closed languages to sets of elements composed of a timed word and a date: a set $S \subseteq \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0}$ is *time-extension-closed* if for any $(\sigma, t) \in S$, for any $w \in \text{tw}(\Sigma)$, and for any $t' \geq t$, $(\sigma \cdot w, t') \in S$. In other words, S is time-extension-closed if for every $\sigma \in \text{tw}(\Sigma)$, there exists a date t from which 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 $\sigma' \preccurlyeq_\text{d} \sigma$, whenever $\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\sigma)$ and for any $i \in [1; |\sigma'| - 1], \text{time}(\sigma_{[..i]}) \leq \text{time}(\sigma'_{[..i]})$. Intuitively, $\sigma' \preccurlyeq_\text{d} \sigma$ if σ' is a prefix of σ whose actions have been delayed, considering dates. 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 $\sigma' \preccurlyeq_{\text{d}\Sigma'} \sigma$ whenever $\sigma'_{|\Sigma'} \preccurlyeq_\text{d} \sigma_{|\Sigma'}$, and $\sigma' \prec_\text{d} \sigma$ whenever $\sigma' \preccurlyeq_\text{d} \sigma$ and $\sigma' \neq \sigma$. We also define a *lexical order* $\leq_\text{lex}$ on timed words with identical untimed projections, such that $\epsilon \leq_\text{lex} \epsilon$, and for two words σ and σ' such that $\Pi_\Sigma(\sigma) = \Pi_\Sigma(\sigma')$, and two events (δ, a) and (δ', a), $(\delta', a) \cdot \sigma' \leq_\text{lex} (\delta, a) \cdot \sigma$ if $\delta' < \delta \vee (\delta = \delta' \wedge \sigma' \leq_\text{lex} \sigma)$.

Consider for example the timed word $\sigma = (1, a) \cdot (1, b) \cdot (1, c) \cdot (2, a)$ over the alphabet $\Sigma = \{a, b, c\}$. Then, $\Pi_\Sigma(\sigma) = a \cdot b \cdot c \cdot a$, $\text{obs}(\sigma, 4) = (1, a) \cdot (1, b) \cdot (1, c)$, $\text{nobs}(\sigma, 4) = (2, a)$, and if $\Sigma' = \{b, c\}$, $\sigma_{|\Sigma'} = (2, b) \cdot (1, c)$, and for instance $(1, a) \cdot (1, b) \cdot (3, c) \preccurlyeq_\text{d} \sigma$, and $\sigma \leq_\text{lex} (1, a) \cdot (0, b) \cdot (3, c) \cdot (3, a)$. Moreover, if $w = (1, a) \cdot (1, b)$, then $w^{-1} \cdot \sigma = (1, c) \cdot (2, a)$.

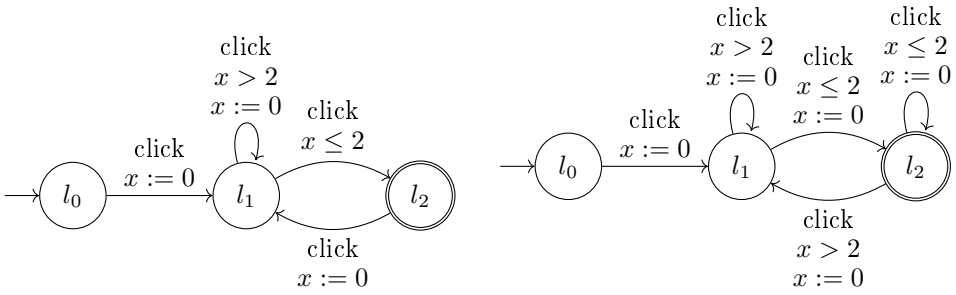
2.3. Timed Automata as Timed Properties

In the following part we recall some notions on timed automata ([AD92]). These notions are classical. A reader not familiar with timed automata may find more details and explanations in [AD92].

Let $X = \{X_1, X_2, \ldots, X_n\}$ be a finite set of *clocks*, i.e. variables that increase regularly with time. A *clock valuation* is a function ν from X to $\mathbb{R}_{\geq 0}$. The set of clock valuations for the set of clocks X is noted $\mathcal{V}(X)$, i.e. $\mathcal{V}(X) = \{\nu \mid \nu : X \to \mathbb{R}_{\geq 0}\}$. We consider the following operations on valuations:

- for any valuation $\nu \in \mathcal{V}(X)$, $\nu + \delta$ is the valuation representing the elapse of δ time units from ν, such that for any $X_i \in X$, $(\nu + \delta)(X_i) = \nu(X_i) + \delta$;
- for any subset $X' \subseteq X$, $\nu[X' \leftarrow 0]$ is the valuation representing ν with clocks in X' reset, such that:

$$(\nu[X' \leftarrow 0]) : X_i \mapsto \begin{cases} 0 \text{ if } X_i \in X' \\ \nu(X_i) \text{ otherwise.} \end{cases}$$



(a) Timed automaton modelling double-clicks (b) Timed automaton modelling double-clicks with overlaps

Figure 2. Timed automata modelling double-clicks, without and with overlaps

$\mathcal{G}(X)$ denotes the set of guards consisting of boolean combinations (i.e. conjunctions or disjunctions) of constraints of the form $X_i \bowtie c$ with $X_i \in X$, $c \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. Given $g \in \mathcal{G}(X)$ and a valuation ν, we write $\nu \models g$ when g holds according to ν.

Timed automaton [AD92] A *timed automaton* (TA) is a tuple $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, G \rangle$, such that L is a set of locations, $l_0 \in L$ is the initial location, X is a set of clocks, 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 timed automaton $\mathcal{A}$ 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, $q_0 = (l_0, \nu_0)$ is the initial state, with $\nu_0 = \nu[X \leftarrow 0]$, $F_G = G \times \mathcal{V}(X)$ is the set of accepting states, $\Gamma = \mathbb{R}_{\geq 0} \cup \Sigma$ is the set of transition labels, that can be either a number representing a delay, or an action. The transition relation $\rightarrow \subseteq Q \times \Gamma \times Q$ is made of transitions of two possible kinds:

- *Delay transitions:* for $\delta \in \mathbb{R}_{\geq 0}$ and $(l, \nu) \in Q$, $\langle (l, \nu), \delta, (l, \nu + \delta) \rangle \in \rightarrow$.
- *Action transitions:* for $a \in \Sigma$, $(l, \nu) \in Q$, $\langle (l, \nu), a, (l', \nu') \rangle \in \rightarrow$ if there exists $(l, g, a, Y, l') \in \Delta$, such that $\nu \models g$ and $\nu' = \nu[Y \leftarrow 0]$.

A timed automaton $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, G \rangle$ is *deterministic* if for any (l, g_1, a, Y_1, l'_1) and (l, g_2, a, Y_2, l'_2) in 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 holds for any clock valuation.

Example 1. Examples of timed automata are given in Fig. 2. In Fig. 2a, $L = \{l_0, l_1, l_2\}$, $X = \{x\}$, $\Sigma = \{\text{click}\}$, $\Delta = \{(l_0, \top, \text{click}, \{x\}, l_1), (l_1, x > 2, \text{click}, \{x\}, l_1), (l_1, x \leq 2, \text{click}, \emptyset, l_2), (l_2, \top, \text{click}, \{x\}, l_1)\}$, and $G = \{l_2\}$, where $\top$ evaluates to true for any clock valuation. This automaton models a double click: considering that the *click* event is a mouse click, the automaton only accepts sequences of clicks that ends with a double-click. The condition for two clicks to be considered as a double-click is that the second one is made less than two time units after the first one. Note that with this modelling, double-clicks can not overlap, i.e. clicking three times in less than two time units will not be considered as ending with a double-click, since only the first two clicks will be considered as a double-click. Allowing overlaps would only require splitting the transition from l_2 to l_1 in two, as described in Fig. 2b.

A *run* ρ from $q \in Q$ is a valid sequence of transitions in $[\![\mathcal{A}]\!]$ starting from q, of the form $\rho = q \xrightarrow{\delta_1} q_1 \xrightarrow{a_1} q_2 \xrightarrow{\delta_2} q_3 \ldots \xrightarrow{a_n} q_{2n} \xrightarrow{\delta_{n+1}} q_{2n+1}$, where $\delta_i \in \mathbb{R}_{\geq 0}$ and $a_i \in \Sigma$, for any i. We can consider runs that alternate between delay and action transitions, since two consecutive delay transitions can be merged into one whose value is the sum of the delays of the original transitions, and two consecutive actions can be separated by a null delay transition (i.e. a delay transition whose delay is 0). The set of runs from q_0 is noted $Run(\mathcal{A})$ and $Run_{F_G}(\mathcal{A})$ 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 $\sigma = (\delta_1, a_1) \cdot (\delta_2, a_2) \ldots (\delta_n, a_n)$. Note that

δ_{n+1} does not appear in the trace, meaning that all runs with different values for δ_{n+1} share the same trace as ρ. We allow ourselves to denote by $q \xrightarrow{(\delta,a)} q'$ if $q \xrightarrow{\delta} q'' \xrightarrow{a} q'$, and thus ρ can be denoted $q \xrightarrow{\sigma} q_{2n} \xrightarrow{\delta_{n+1}} q_{2n+1}$.

A *regular timed property* is a timed language $\varphi \subseteq \text{tw}(\Sigma)$ that is accepted by a timed automaton. 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.

2.4. Traces Manipulation

Given a deterministic timed automaton $\mathcal{A} = \langle L, l_0, X, \Sigma, \Delta, F\rangle$ whose semantics is $[\![A]\!] = \langle Q, q_0, \Gamma, \to, F_G\rangle$, and a timed word $\sigma \in \text{tw}(\Sigma)$, for $q \in Q$, we note q *after* $\sigma = q'$, where q' is such that $q \xrightarrow{\sigma} q'$, i.e. q' is the state reached from q after reading word σ. Since $\mathcal{A}$ is deterministic, there exists only one such q'. We also note $\mathit{Reach}(\sigma) = q_0$ *after* σ. We extend these definitions to languages: if L is a language, q *after* $L = \bigcup_{\sigma\in L}\{q \text{ after } \sigma\}$ and $\mathit{Reach}(L) = q_0$ *after* L. We also allow ourselves to use *after* with actions or delays, with the same definition: for $\delta \in \mathbb{R}_{\geq 0}$, q *after* $\delta = q'$, if $q \xrightarrow{\delta} q'$, and for $a \in \Sigma$, q *after* $a = q'$ if $q \xrightarrow{a} q'$.

We allow the use of the operators *after* and *Reach* with an extra parameter, representing an observation time, such that if $t \in \mathbb{R}_{\geq 0}$, then q *after* $(\sigma, t) = q$ *after* $\text{obs}(\sigma, t)$ *after* $(t - \text{time}(\text{obs}(\sigma, t)))$, and $\mathit{Reach}(\sigma, t) = q_0$ *after* (σ, t). Moreover, for $q = \langle l, \nu\rangle \in Q$, we note $\text{up}(q) = \{\langle l, \nu + t\rangle \in Q \mid t \in \mathbb{R}_{\geq 0}\}$, it is the set of states that will be reached from q as time elapses if no action occurs. This definition is extended to sets of states: for $S \subseteq Q$, $\text{up}(S) = \bigcup_{q\in S} \text{up}(q)$.

Example 2. Consider the property accepting sequences of clicks ending by a double-click, described in Fig. 2a. Let us consider that the set Q of states of the semantics of this TA is $Q = L \times \mathbb{R}_{\geq 0}$, with $L = \{l_0, l_1, l_2\}$, and where the valuations are replaced by the value of the unique clock x. Then, for instance, $\mathit{Reach}((1, \text{click})) = (l_0, 0)$ *after* $(1, \text{click}) = (l_1, 0)$, and $(l_1, 1)$ *after* $((1, \text{click}), 3) = (l_2, 4)$, since $x = 2$ when the action *click* occurs, enabling the transition to l_2, and then 2 time units remain to wait, giving a final value of 4 for x.

2.5. Büchi Games

We recall some general notions on Büchi games in an informal way. More details and formal notations may be found in [GT02]. Considering a directed graph $\langle V, E\rangle$ such that V is a set of vertices, and $E \subseteq V \times V$ a set of edges. V is partitioned into two subsets: V_0 the player P_0 vertices, and V_1 the player P_1 ones. As usual in game graphs, we assume that there always exists an outgoing edge from a vertex. The two players play a game on the graph by forming a (infinite) path from the initial vertex in the graph by moving a token along edges. If the token is in V_0 (resp. V_1), the player P_0 (resp. P_1) moves the token following an edge going out of the vertex. The resulting (infinite) path in the graph is called a *play*. A strategy for a player is a way to extend plays. Formally, a *strategy* for player P_0 is a mapping $\sigma : V^*V_0 \to V$ that, given a finite sequence of vertices (the history of the play) ending in a player P_0 vertex, chooses the next vertex. In order to ensure that strategies always exist, we will consider that the graph is strongly connected. A play $\pi = v_0, v_1, \ldots$ is *consistent* with the strategy σ if for any $v_i \in V_0$, $v_{i+1} = \sigma(v_0 \cdot v_1 \cdot \cdots \cdot v_i)$, meaning that the strategy was followed for any vertex in V_0. The goal of a game can be, for example, to reach a state in a given subset of V (reachability game), or to ensure that a given subset of V is visited an infinite number of times (Büchi games). Thus, given a subset $F_G \subseteq V$ of vertices, the Büchi game for P_0 consists in finding a *winning strategy* σ such that all plays π consistent with σ visit an infinite number of times the set F_G. We refer to the nodes in F_G as *Büchi nodes*.

It is known that it is possible ([CHP08]) to compute the set W_0 of winning vertices for P_0 (i.e. the set of vertices from where there exists a winning strategy for P_0), and the associated winning strategy from all these vertices. From all the other vertices (in $V \setminus W_0$), there exists a winning strategy for P_1, i.e. $W_1 = V \setminus W_0$, thus P_0 can not win the game if P_1 plays perfectly from one of these vertices. Moreover, it is possible to find a strategy that is *memoryless*, meaning that the only the last vertex is needed to compute the next. Formally, a *memoryless strategy* for P_0 is a strategy $\sigma : V_0 \to V$. Such strategies are easier to compute, since they do not require to read the entire history before choosing the transition to follow.

2.6. Functions

In all this paper, we use functions to describe the input/output behaviour of EMs. We then use *input* and *output* to refer to "argument" and "image" of such functions, respectively.

3. Enforcing Timed Properties using a Büchi Game

In this section, we present the theoretical framework for enforcing timed properties using games. This approach has been implemented in the GREP tool. A first approach has been published in [RRF17b], but for untimed properties only.

Given a timed regular property described by a TA, the objective is to synthesise an EM which is sound according to this property, compliant and optimal. To synthesise the EM, we propose to build a two-player game graph representing the possible actions of the EM and the system under scrutiny. Each edge of this graph corresponds to a possible action of a player. The last step consists in computing the set of nodes of the graph from which there exists a winning strategy, i.e. solving a Büchi game. More precisely, the approach is divided into three steps: first, it computes a symbolic graph, which is a finite abstraction of the (infinite) semantics of the TA; then it computes the game graph and the winning strategy for the EM; finally the EM follows the strategy to enforce the property. Using this approach permits to pre-compute some of the decisions of the EM, thus avoiding to explore all paths of the execution tree at runtime.

GREP is a tool designed to produce EMs, such as the ones described in [RFR+17]. However, the formal definitions from [RFR+17] did not fit well with an effective implementation, thus we propose EMs that are formally equivalent, but that are built in a more implementation-friendly way.

We first introduce enforcement functions and their requirements, namely *soundness*, *compliance* and *optimality*. Enforcement functions are functional descriptions of EMs that represent their input/output behaviour, as depicted in Fig. 1. Enforcement functions serve as abstract non-operational descriptions of EMs which can possibly be reused to obtain alternative operational descriptions of EMs. In addition, enforcement functions facilitate the expression of requirements on EMs. Then, we give details about the EM synthesis framework using games, and show that the obtained EM is sound, compliant and optimal. Then, we provide another operational description of the EM, based on transition systems, and show their equivalence.

3.1. Enforcement Functions and their Properties

In the rest of this section, we consider an alphabet of actions Σ. We give formal definitions of *enforcement functions* and the requirements of such functions, i.e. *soundness*, *compliance*, and *optimality*. The definitions in this section are equivalent to the ones in [RFR+17]. In the following, φ is a timed property defined by a timed automaton $\mathcal{A}_\varphi = \langle L, l_0, X, \Sigma, \Delta, F\rangle$ with semantics $[\![\mathcal{A}_\varphi]\!] = \langle Q, q_0, \Gamma, \to, F_G\rangle$, and the set of actions Σ is partitioned into a set of uncontrollable actions Σ_u, (which can not be modified by the EM) and a set of controllable actions Σ_c (the other actions). As in [PFJ+14], the EM is equipped with a buffer permitting to store (only controllable) actions and release them later. The EM is modelled by a function from $\mathrm{tw}(\Sigma) \times \mathbb{R}_{\geq 0}$ to $\mathrm{tw}(\Sigma)$. Before detailing the definitions, we provide some intuitions about the synthesis of the EM. For this purpose, we illustrate its desired behaviour on an example.

Example 3. Consider property φ_t (Fig. 3) which models the use of a shared writable device with a centralised controller. To be able to `Write` to the device, one must first authenticate (uncontrollable event *Auth*), then wait 2 time units for synchronisation. Uncontrollable events *LockOn* and *LockOff* are used by the controller to notify that someone is writing, meaning that the lock has been taken by someone else, and that the lock has been released, respectively. While the lock is held by someone else, writings are not allowed. And after the lock has been released, one must wait 2 time units before writing. Indeed, in φ_t all transitions arriving in location l_1 reset the clock x. From l_1, emitting event `Write` after 2 time units loops on the same location, whereas emitting it before 2 time units leads to location l_3, which is a non-accepting state. Consider an EM aiming to enforce this property with the input $\sigma = ((1, \mathit{Auth}) \cdot (1, \mathit{LockOn}) \cdot (2, \texttt{Write}) \cdot (1, \mathit{LockOff}) \cdot (1, \mathit{LockOn}) \cdot (1, \texttt{Write}) \cdot (1, \mathit{LockOff}))$. At time $t = 1$, the uncontrollable event *Auth* occurs. It has to be released immediately since it is an uncontrollable event, then φ_t goes to l_1. At $t = 2$, the uncontrollable event *LockOn* is received. It is also emitted instantaneously,

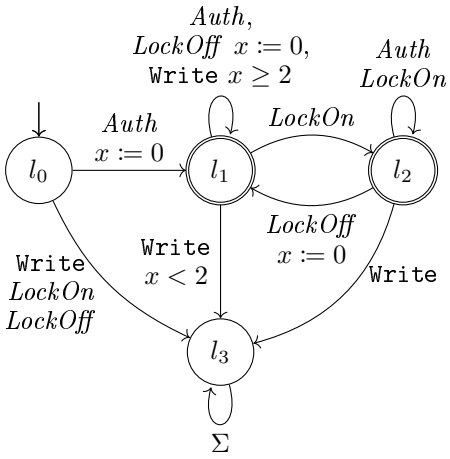


Figure 3. Property φ_t

leading φ_t to l_2. At $t = 4$, the controllable event `Write` occurs. It should not be emitted since it would make the system go to l_3 which is a non-accepting state. It is stored in the buffer. Note that it is not necessary to store the corresponding delay, since while no other uncontrollable event occurs, it is not possible to compute an output time for `Write`. At time $t = 5$, the uncontrollable event *LockOff* occurs. It has to be released immediately, and φ_t goes to l_1. Now it is possible to emit the stored action `Write`, but it is necessary to wait for 2 time units before. This value is associated to `Write` pending output (as described in [PFJ+14]), and should be definitely released if no other uncontrollable event happens meanwhile. At $t = 6$, the uncontrollable event *LockOn* occurs, then φ_t goes to l_2. It is no longer possible to emit `Write`, which is again not associated to an output instant anymore. At time $t = 7$, the second `Write` event occurs, and it is also stored in the buffer without a possibility to compute its output instant. The occurrence at $t = 8$ of the uncontrollable event *LockOff* unblocks the situation, leading φ_t to l_1, and the two stored actions `Write` are again associated with output delays, 2 and 0 respectively, meaning that they should be emitted at $t = 10$ if no other uncontrollable happens in the mean time.

Finally, given the input $\sigma = ((1, \mathit{Auth}) \cdot (1, \mathit{LockOn}) \cdot (2, \texttt{Write}) \cdot (1, \mathit{LockOff}) \cdot (1, \mathit{LockOn}) \cdot (1, \texttt{Write}) \cdot (1, \mathit{LockOff}))$, the output of the EM should be $(1, \mathit{Auth}) \cdot (1, \mathit{LockOn}) \cdot (3, \mathit{LockOff}) \cdot (1, \mathit{LockOn}) \cdot (2, \mathit{LockOff}) \cdot (2, \texttt{Write}) \cdot (0, \texttt{Write})$. Consider now the situation where the uncontrollable event *LockOn* is received at the beginning of the process (i.e. in location l_0). It leads inevitably φ_t to location l_3 which is a non-accepting state without any possibility to go back to an accepting one. This illustrates the fact that φ_t is not always enforceable as we shall discuss later.

An enforcement function is a functional representation of the input/output behaviour of an EM, thus an enforcement function takes a timed word and the current time as input, and outputs a timed word:

Definition 1 (Enforcement Function). An *enforcement function* (over Σ) is a function $E : \mathrm{tw}(\Sigma) \times \mathbb{R}_{\geq 0} \to \mathrm{tw}(\Sigma)$ that satisfies the two following constraints:

1. $\forall \sigma \in \mathrm{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, \forall t' \geq t, E(\sigma, t) \preccurlyeq E(\sigma, t')$,
2. $\forall \sigma \in \mathrm{tw}(\Sigma), \forall \delta \in \mathbb{R}_{\geq 0}, \forall a \in \Sigma, E(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) \preccurlyeq E(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)))$.

We note $\mathcal{F}_{\mathrm{enf}}(\Sigma)$ (or $\mathcal{F}_{\mathrm{enf}}$ when clear from the context) the set of all enforcement functions over Σ.

The requirements in Definition 1 model physical constraints (i.e. the irreversibility of time): an enforcement function can only append events to its output as the input grows. The first constraint ($E(\sigma, t) \preccurlyeq E(\sigma, t')$) corresponds to the elapse of time, whereas the second one ($E(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) \preccurlyeq E(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)))$) corresponds to the read of a new event. They require the new output to be an extension of the previous one.

An enforcement function should fulfil the three following requirements: *soundness*, *compliance* and *optimality*. These notions are usual in the RE literature (see e.g. in [LBW09, FMFR11]), but have to be adapted to a timed context and uncontrollable events. More precisely, the definitions proposed in this section are adapted from [PFJ+14] (RE framework in a timed context, only with controllable events) and [RRF17b]

(RE framework in an untimed context, with uncontrollable events). Soundness states that the output of an enforcement function should satisfy the desired property. This means that the corresponding EMs should modify executions in order to keep the property satisfied. Compliance indicates which operations are allowed on the executions, i.e. how the EM can modify the output. Compliance is a weaker version of the usual notion of transparency (e.g. used in [FMFR11, PFJ+13, PFJM14b, PFJM15]). Indeed, the framework considers controllable and uncontrollable events. Since uncontrollable events have to be released immediately, it may happen that the order of input events is different from the output ones. Moreover, contrary to the usual notion of transparency which contains an implicit notion of optimality, we distinguish explicitly the two notions. Then, optimality requires the EM to emit as many events as possible, while preserving the property from being violated.

We give formal definitions of these requirements as constraints on enforcement functions. In a timed context, *soundness* states that the output of an enforcement function should eventually always satisfy the desired property (here, φ):

Definition 2 (Soundness). An enforcement function $E \in \mathcal{F}_{\text{enf}}$ is *sound* with respect to φ in a time-extension-closed set $S \subseteq \text{tw}(\Sigma) \times \mathbb{R}_{\geq 0}$ if:

$$\forall (\sigma, t) \in S, \exists t' \geq t, \forall t'' \geq t', E(\sigma, t'') \models \varphi.$$

We note $\mathcal{F}_{\text{snd}}(\varphi, S)$ the set of all enforcement functions that are sound with respect to φ in S.

An enforcement function is sound with respect to φ in S if for any $(\sigma, t) \in S$, the output of the enforcement function with input σ from date t eventually always satisfies φ. Notice that in some situations, the occurrence of some words may inevitably lead the system to a non-accepting state. Considering e.g. φ_{t} in Fig. 3, the occurrence of the uncontrollable action *LockOn* in q_0 would inevitably lead the system to the non-accepting state q_3. For this reason, similarly to [RRF17b] in the untimed case, soundness is restricted to an extension-closed set S.

Compliance states that an EM can only delay controllable events, without modifying uncontrollable events (i.e. keeping their dates unchanged). This is a direct adaptation of the choice made in [PFJ+14] where the EM acts as a delaying filter on the input. We recall that Σ_{c} and Σ_{u} are the sets of controllable and uncontrollable actions respectively, and that $\Sigma_u \cap \Sigma_c = \emptyset$:

Definition 3 (Compliance). An enforcement function $E \in \mathcal{F}_{\text{enf}}$ is *compliant* with respect to Σ_{u} and Σ_{c}, noted compliant$(E, \Sigma_{\text{u}}, \Sigma_{\text{c}})$ (or compliant(E) when clear from the context), if it satisfies the following constraints:

1. $\forall \sigma \in \text{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, E(\sigma, t) \preccurlyeq_{\text{d}_{\Sigma_{\text{c}}}} \text{obs}(\sigma, t)$,
2. $\forall \sigma \in \text{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, E(\sigma, t) =_{\Sigma_{\text{u}}} \text{obs}(\sigma, t)$,
3. $\forall \sigma \in \text{tw}(\Sigma), \forall (\delta, u) \in \mathbb{R}_{\geq 0} \times \Sigma_{\text{u}}$,
   $E(\sigma, \text{time}(\sigma \cdot (\delta, u))) \cdot (\text{time}(\sigma \cdot (\delta, u)) - \text{time}(E(\sigma, \text{time}(\sigma \cdot (\delta, u)))), u) \preccurlyeq E(\sigma \cdot (\delta, u), \text{time}(\sigma \cdot (\delta, u)))$.

We note $\mathcal{F}_{\text{cpl}}(\Sigma_{\text{u}}, \Sigma_{\text{c}})$ (or $\mathcal{F}_{\text{cpl}}$ when clear from the context) the set of all enforcement functions that are compliant with respect to Σ_{u} and Σ_{c}.

In Definition 3, the first constraint allows the EM to delay controllable events without changing their order; the second constraint requires that uncontrollable events are not modified; and the third constraint requires that the EM does not react to the reception of an uncontrollable event before outputting it.

For a compliant enforcement function $E \in \mathcal{F}_{\text{enf}}$ and a timed word $\sigma \in \text{tw}(\Sigma)$, we say that $E(\sigma, \infty)$ is the output of E with input σ at infinite time (i.e. when it has stabilised). More formally, $E(\sigma, \infty) = E(\sigma, t)$, with $t \in \mathbb{R}_{\geq 0}$ such that for all $t' \geq t, E(\sigma, t') = E(\sigma, t)$. Since the input σ is finite, and E is a compliant function (guaranteeing that the size of the output is lower or equal than that of the input), the output of E with input word σ is finite, thus such a t exists, and $E(\sigma, \infty)$ is well-defined.

Another objective of an EM is to provide an output as close as possible to the input. Then, a notion of *optimality* has to be defined. It requires that an EM outputs as many events as possible, without breaking compliance and avoiding the possibility of breaking soundness:

Definition 4 (Optimality). A sound and compliant enforcement function $E \in \mathcal{F}_{\text{snd}}(\varphi, S) \cap \mathcal{F}_{\text{cpl}}(\Sigma_{\text{u}}, \Sigma_{\text{c}})$

is *optimal* in S if:

$$\forall E' \in \mathcal{F}_{\text{cpl}}(\Sigma_{\text{u}}, \Sigma_{\text{c}}), \forall \sigma \in \text{tw}(\Sigma), \forall (\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma,$$
$$(\sigma, \text{time}(\sigma \cdot (\delta, a))) \in S \wedge E(\sigma, \text{time}(\sigma \cdot (\delta, a))) = E'(\sigma, \text{time}(\sigma \cdot (\delta, a))) \wedge E(\sigma \cdot (\delta, a), \infty) \prec_{\text{d}} E'(\sigma \cdot (\delta, a), \infty)$$
$$\implies \exists \sigma_u \in \text{tw}(\Sigma_{\text{u}}), E'(\sigma \cdot (\delta, a) \cdot \sigma_u, \infty) \not\models \varphi.$$

Intuitively, a sound and compliant enforcement function is optimal if at any moment, it outputs the longest possible word, with the lowest possible delays, ensuring soundness and compliance. In Definition 4, we suppose that a compliant enforcement function outputs a greater word (with respect to $\preccurlyeq_{\text{d}}$) than an optimal one, and then conclude that it is not sound (since the other function is optimal).

3.2. Synthesising Timed Enforcement Functions

In this part, we define a sound, compliant and optimal inductive enforcement function E_φ. This function is equivalent to the one described in [RFR+17], the difference residing in the use of a Büchi game to compute the safe states. The interest of this approach is twofold: it permits to simplify the definition of the EM, and to obtain globally better performance than in [RFR+17] by precomputing some decisions of the EM. First, we need to compute the set of words that can be emitted by an enforcement function while ensuring compliance and soundness. This set depends on the current state in $[\![\mathcal{A}_\varphi]\!]$, and on the controllable actions of the input that have not been output yet. These actions are the actions that can be output, in the same order as they were received, with arbitrary delays, by the EM. This sequence of actions is called the *buffer* of the EM. At any moment, the EM has to select the "best" sequence that should be emitted when an event occurs. For this purpose, it is necessary to design a set Safe containing the set of words that it is possible to emit at a given instant, while staying sound, then identify the "best" one. Since Safe depends on the sequence that has already been emitted, it needs two parameters: the state q reached by the sequence already released, and the current state of the buffer, i.e. a word in Σ_{c}^*. Then Safe contains all the prefixes of the buffer verifying soundness, i.e. leading to "safe" states. However, this notion of "safe" state is not simple. It nearly corresponds to accepting states of the property, from which any sequence of uncontrollable events, or letting the time elapse, leads to accepting states. Thus we define the *safe emitting* function $\text{Safe} : Q \times \Sigma_{\text{c}}^* \to \text{tw}(\Sigma_{\text{c}})$ such that $\text{Safe}(q, w)$ is the set of timed words whose actions are the actions of w, associated with some delays, such that outputting such a word from q ensures compliance and soundness.

To compute this set of words, we use a Büchi game played on a graph representing the possible actions of the EM, as in [RRF17b]. We represent the EM as player P_0, and the environment as the other player, P_1. The nodes of this graph should be taken in the set $Q \times \Sigma_{\text{c}}^* \times \{0, 1\}$, where node (q, w, p) belongs to player P_p, and represents the state of the EM where q is the corresponding state of φ, and w the buffer. Considering all such nodes, the graph would have an infinite number of nodes, first because the size of the buffer is not bounded, but also because the semantics of a timed automaton has itself an infinite number of states (i.e. Q is also infinite). We then reduce the number of possible buffers to a finite set Σ_{c}^n. Intuitively, since the validity of a sequence only depends on the location that is reached after reading it, Σ_{c}^n is composed of all the controllable actions that can allow the EM to reach a new location. Then, we define Σ_{c}^n as follows:

$$\Sigma_{\text{c}}^n = \{w \in \Sigma_{\text{c}}^* \mid \exists q \in Q, \exists c \in \Sigma_{\text{c}}, \forall \sigma \in \text{tw}(\Sigma), \forall \sigma' \in \text{tw}(\Sigma),$$
$$\Pi_\Sigma(\sigma) = w \cdot c \wedge \Pi_\Sigma(\sigma') \preccurlyeq w \wedge (l', \nu') = q \textit{ after } \sigma \wedge (l'', \nu'') = q \textit{ after } \sigma' \implies l' \neq l''\}$$

Σ_{c}^n is defined as the set of sequences of controllable actions that can be used to form a word that allows to reach a new location, the length of a word in Σ_{c}^n can not be greater than the number of locations in the timed automaton. Thus, Σ_{c}^n is finite since L is finite. The other component Q is also infinite since in our definition of the semantics of a TA time is continuous. We reduce it to a finite set using a finite symbolic abstraction of the semantics of $\mathcal{A}_\varphi$.

3.2.1. A Symbolic Graph

Several abstractions for timed automata exist to reduce their semantics to a finite representation. The simplest, that satisfies all the requirements we need, is the region graph (see [AD92]) of the timed automaton. Unfortunately, this region graph is often very large, thus some more efficient abstractions have been studied. A very common abstraction is the zone graph used to compute reachability in a timed automaton ([BY04]).

A zone is a convex set of clock valuations, usually represented by clock constraints of the form $x \bowtie n$, where x is a clock, $\bowtie \in \{<, \leq, =, \geq, >\}$, and n is a (rational) number, or more generally by $x - y \bowtie n$, where y is another clock. This graph is usually small compared to the region graph. Nevertheless, this graph only preserves information about the existence of a state in the zones (i.e. a transition in the graph represents the existence of a location and a valuation in the source node to a location and a valuation in the destination node). This is not sufficient for our needs, i.e. to play a Büchi game. Indeed, when computing reachability, the used zones ensure that it is possible to reach a state in another zone from a source zone. However, when considering enforcement, the reception of an uncontrollable event can force a transition, disabling the possibility to wait a desired amount of time. Thus, it is needed to refine zones in such a way that all states in a zone behave the same, with respect to the emission of actions, and with respect to the elapse of time. In other words, the zones need to prevent players from "cheating", for example doing "time leaps". A time leap could be, for instance, starting in a zone $x \leq 2$ and going from that zone to the zone $x > 4$. To play a Büchi game, it is needed that zone $x \leq 2$ leads first to zone $2 < x \leq 4$ that then leads to zone $x > 4$. The zone graph we use is thus different from the one used to compute reachability, since zones are divided in such a way that they prevent time leaps and fit well the purpose of playing a Büchi game.

The graph described in [ACH+92] fits our needs and seems to be a good choice. However, we give a list of constraints that are sufficient for a graph to fit our needs. Any symbolic graph satisfying these constraints could be used to generate the game graph, on which we can play a Büchi game. We say that such symbolic graphs are *compatible* with Büchi games according to the following definition (we use the standard notation $\exists!$ for "unique-existence"):

Definition 5 (Compatible symbolic graph). A symbolic graph $\mathcal{G}_\mathrm{s} = \langle V_\mathrm{s}, E_\mathrm{s} \rangle$, with $E_\mathrm{s} \subseteq V_\mathrm{s} \times (\Sigma \cup \{t\}) \times V_\mathrm{s}$ is *compatible* (with Büchi games) if it satisfies the following constraints:

1. $V_\mathrm{s} \subseteq Q$ is a finite set such that $\forall v \in V_\mathrm{s}, \exists l \in L, v \subseteq l \times 2^{\mathcal{V}(X)}$,
2. $\forall q \in Q, \exists! v \in V_\mathrm{s}, q \in v$,
3. $E_\mathrm{s} = E_\mathrm{s}^\mathrm{a} \cup E_\mathrm{s}^\delta$,
4. $\forall v \in V_\mathrm{s}, \forall a \in \Sigma, \exists! v' \in V_\mathrm{s}, (\forall q \in v, \forall q' \in Q, q \xrightarrow{a} q' \implies q' \in v')$,
   and $E_\mathrm{s}^\mathrm{a} = \{(v, a, v') \in V_\mathrm{s} \times \Sigma \times V_\mathrm{s} \mid \exists (q, q') \in v \times v', q \xrightarrow{a} q'\}$,
5. $\forall (v, v') \in V_\mathrm{s}^2, \forall (q, q') \in v \times v', \forall \delta \in \mathbb{R}_{\geq 0}, q \xrightarrow{\delta} q' \implies (\forall q \in v, \exists \delta' \in \mathbb{R}_{\geq 0}, \exists q' \in v', q \xrightarrow{\delta} q')$,
6. $\forall v \in V_\mathrm{s}, \mathrm{up}(v) \neq v \implies \exists! v' \in V_\mathrm{s}, v \neq v' \wedge \forall (q, q') \in v \times v', \exists \delta \in \mathbb{R}_{\geq 0}, q' = q \textit{ after } \delta \wedge \forall \delta' \leq \delta, q \textit{ after } \delta' \in v \cup v'$,
   and $E_\mathrm{s}^\delta = \{(v, t, v') \in V_\mathrm{s} \times \{t\} \times V_\mathrm{s} \mid v \neq v' \wedge \forall (q, q') \in v \times v', \exists \delta \in \mathbb{R}_{\geq 0}, q' = q \textit{ after } \delta \wedge \forall \delta' \leq \delta, q \textit{ after } \delta' \in v \cup v'\}$.

Constraint (1) imposes that all the states of the semantics that are in a node of the symbolic graph share the same location. This allows us to easily define accepting nodes (as nodes whose locations are accepting).

Constraint (2) allows us to match each state of the semantics with a unique node in the symbolic graph.

Constraint (3) splits the set of edges between edges corresponding to actions and edges corresponding to delays. Each of these sets has its own constraints, described in (4) and (6).

Constraint (4) propagates the reachability and determinism of the timed automaton to the symbolic graph for actions, and defines the set E_s^a of edges corresponding to actions. The edges in E_s^a are labelled with the corresponding action from Σ.

Constraint (5) states that if a state of the semantics leads to another with a delay, and they are not in the same node, then all states in the first node can reach a state in the second node with a delay.

Constraint (6) requires that each node of the graph has at most one direct time successor, with which it is linked by an edge in the set E_s^δ of edges corresponding to delays. The edges in E_s^δ are labelled with the special action t, which is supposed not to belong to Σ.

The graph defined in [ACH+92] is a graph that is compatible with Büchi games, as per Definition 5. This graph is the one that has been used as symbolic graph in the implementation of GREP (see Section 4).

Example 4. Consider again property φ_t (Fig. 3). The corresponding symbolic compatible graph of φ_t as per [ACH+92] is given in Fig. 4. In the graph of Fig. 4, the nodes are labelled with a location and a zone, represented as a set of clock constraints. Edges can represent an event transition or the elapse of time. Red edges with filled diamond heads (⟶◆) represent transitions with uncontrollable events, whereas orange

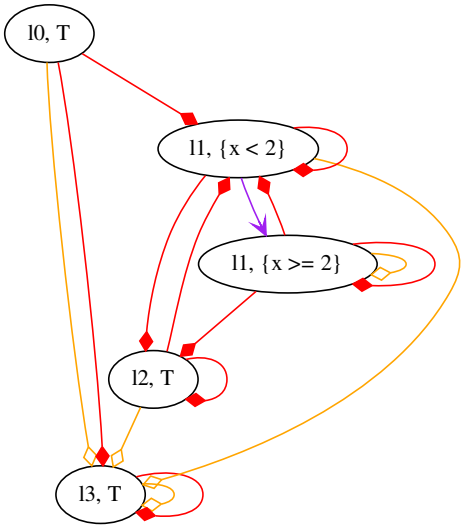


Figure 4. A symbolic compatible graph of φ_t as per [ACH+92].

edges with empty diamond heads (⟶◇) represent transitions with controllable events. Purple edges with "vee" heads (⟶) represent the elapse of time.

Thus, in Fig. 4, orange edges correspond to transitions labelled by `Write`, since it is the only controllable event. Red edges can represent transitions of any other event, *LockOn*, *LockOff*, or *Auth*. Edges are not duplicated, meaning that two events with the same controllability that label the same transition will appear as a unique edge in the graph. For example, from node $(l0, \top)$, events *LockOff* and *LockOn* lead to $(l3, \top)$, but only one red edge is drawn.

From this symbolic graph, we define another graph, as in [RRF17b], on which we play a Büchi game. This graph is called *game graph*.

3.2.2. The Game Graph

This part describes how to construct the graph on which a Büchi game is played. Let us consider $\mathcal{G}_s = (V_s, E_s)$, a symbolic graph compatible with Büchi games. We use Σ_c^n and a compatible symbolic graph $\mathcal{G}_s$ to define a game graph $\mathcal{G}$, the finite graph on which to play the Büchi game:

Definition 6 (Game graph). A graph $\mathcal{G} = \langle V, E \rangle$ is a game graph if:

- $V = V_s \times \Sigma_c^n \times \{0, 1\}$,
- $E = \bigcup_{i=1}^{6} E_i$, with
  - $E_1 = \{(\langle v, w, 0\rangle, \langle v, w, 1\rangle) \in V^2\}$,
  - $E_2 = \{(\langle v, c.w, 0\rangle, \langle v \textit{ after } c, w, 0\rangle) \in V^2 \mid c \in \Sigma_c\}$,
  - $E_3 = \{(\langle v, w, 1\rangle, \langle v \textit{ after } u, w, 0\rangle) \in V^2 \mid u \in \Sigma_u\}$,
  - $E_4 = \{(\langle v, w, 1\rangle, \langle v, w.c, 0\rangle) \in V^2 \mid c \in \Sigma_c \wedge w.c \in \Sigma_c^n\}$,
  - $E_5 = \{(\langle v, w, 1\rangle, \langle v', w, 0\rangle) \in V^2 \mid (v, t, v') \in E_s^\delta\}$,
  - $E_6 = \{(\langle v, w, 1\rangle, \langle v, w, 0\rangle) \in V^2 \mid \mathrm{up}(v) = v\}$,

As per Definition 6, a node in a game graph $\mathcal{G}$ is composed of a node of a symbolic graph $\mathcal{G}_s$, a buffer, and a player it belongs to. The two players are the enforcement mechanism P_0, whose associated number is

0, and the environment P_1, whose associated number is 1. The set of edges is partitioned into six sets, each representing a different type of action. E_1 contains the edges corresponding to P_0 letting P_1 play; edges in E_2 represent P_0 emitting the first event of its buffer; E_3 and E_4 contain edges corresponding to receiving an uncontrollable or controllable event, respectively, both of which being actions of P_1. Edges in E_5 represent time elapse: it changes the node of the symbolic graph to its time successor if it has one. E_6 contains edges that allow us to consider finite inputs. Since plays are infinite, such edges are needed to allow the environment to receive nothing (it can be seen as adding an empty event to the input). Since time elapses when no event is received, these edges exist only from nodes that have no time successor, i.e. nodes that are stable by elapse of time.

On this graph, we play a Büchi game with the set of Büchi nodes being defined as:

$B = \{\langle (l, Z), w, 0\rangle \in V \mid l \in F\}$

We can now consider W_0 the set of winning nodes of this game for player P_0.

Example 5. Consider again property φ_t (Fig. 3) whose symbolic graph was represented in Fig. 4. The game graph associated with φ_t is given in Fig. 5. In this graph, the initial node is the square node, the Büchi nodes are the double-circled nodes, and the winning nodes (the nodes in W_0) are the rounded rectangular ones. The two first members represent a node of the corresponding compatible symbolic graph $\mathcal{G}_s$ (see Fig. 4), and the third member is a prefix of the buffer of the EM, where w stands for the `Write` event, which is the only controllable event. As in the untimed setting, edges are represented differently according to the set they belong to:

- blue edges, with empty triangular heads (⟶▷) belong to E_1 (the EM skips its turn),
- green edges, with filled triangular heads (⟶▶) belong to E_2 (the EM emits the first event of its buffer),
- orange edges, with empty diamond heads (⟶◇) belong to E_4 (a controllable event is received),
- red edges, with filled diamond heads (⟶◆) belong to E_3 (an uncontrollable event is received) or E_6 (no more event is to be received),
- purple edges, with "vee" heads (⟶) belong to E_5 (elapse of time).

For example in Fig. 5, let us consider that the node $(l_1, \{x < 2\}, -, 1)$ has been reached. This node belongs to P_1, meaning that the environment is now playing. To leave this node, there are four cases:

- receiving the uncontrollable event *LockOn* (red edge), which leads to the node $(l_2, T, -, 0)$ corresponding to the location l_2 of the TA.
- receiving the uncontrollable events *LockOff* or *Auth* (red edge), which leads to node $(l_1, \{ x < 2\}, -, 0)$.
- receiving the controllable event `Write` (orange edge), which leads to node $(l_1, \{x < 2\}, \mathtt{w}, 0)$, meaning that the event `Write` has been added to the buffer.
- letting time elapse (purple edge), which leads to node $(l_1, \{x \geq 2\}, -, 0)$, thus updating the symbolic state.

In these four situations, the Environment (P_1) gives back the turn to the EM (P_0).

From this game graph, knowing the winning set W_0 for P_0 allows to compute the "safe" states of an EM.

3.2.3. The Enforcement Function

Now, we can finally define *safe emitting* function Safe, and use it to define function E_φ, through function store_φ. Then, we show that E_φ is an enforcement function that is sound, compliant and optimal.

We use W_0, the set of winning nodes of this game for player P_0, to define, for $q \in Q$ and $w \in \Sigma_c^*$, *safe emitting function* $\mathrm{Safe}(q, w)$, computing the set of words that can be output by an EM from state q with buffer w, ensuring compliance and soundness. We also define the notion of *maximal safe word* $\kappa_\varphi(q, w)$ that will be used for defining store_φ.

Safe emitting function (Safe) For $q \in Q$, and $w \in \Sigma_c^*$, the *safe emitting function* Safe is defined by:

$$\mathrm{Safe}(q, w) = \{\sigma \in \mathrm{tw}(\Sigma) \mid \Pi_\Sigma(\sigma) \preccurlyeq w \wedge q \textit{ after } \sigma \in F_G \wedge \forall t \in \mathbb{R}_{\geq 0}, \forall v \in V_s, (q \textit{ after } (\sigma, t) \in v) \implies \langle v, \mathrm{maxbuffer}(\Pi_\Sigma(\mathrm{obs}(\sigma, t))^{-1} \cdot w), 1\rangle \in W_0\},$$

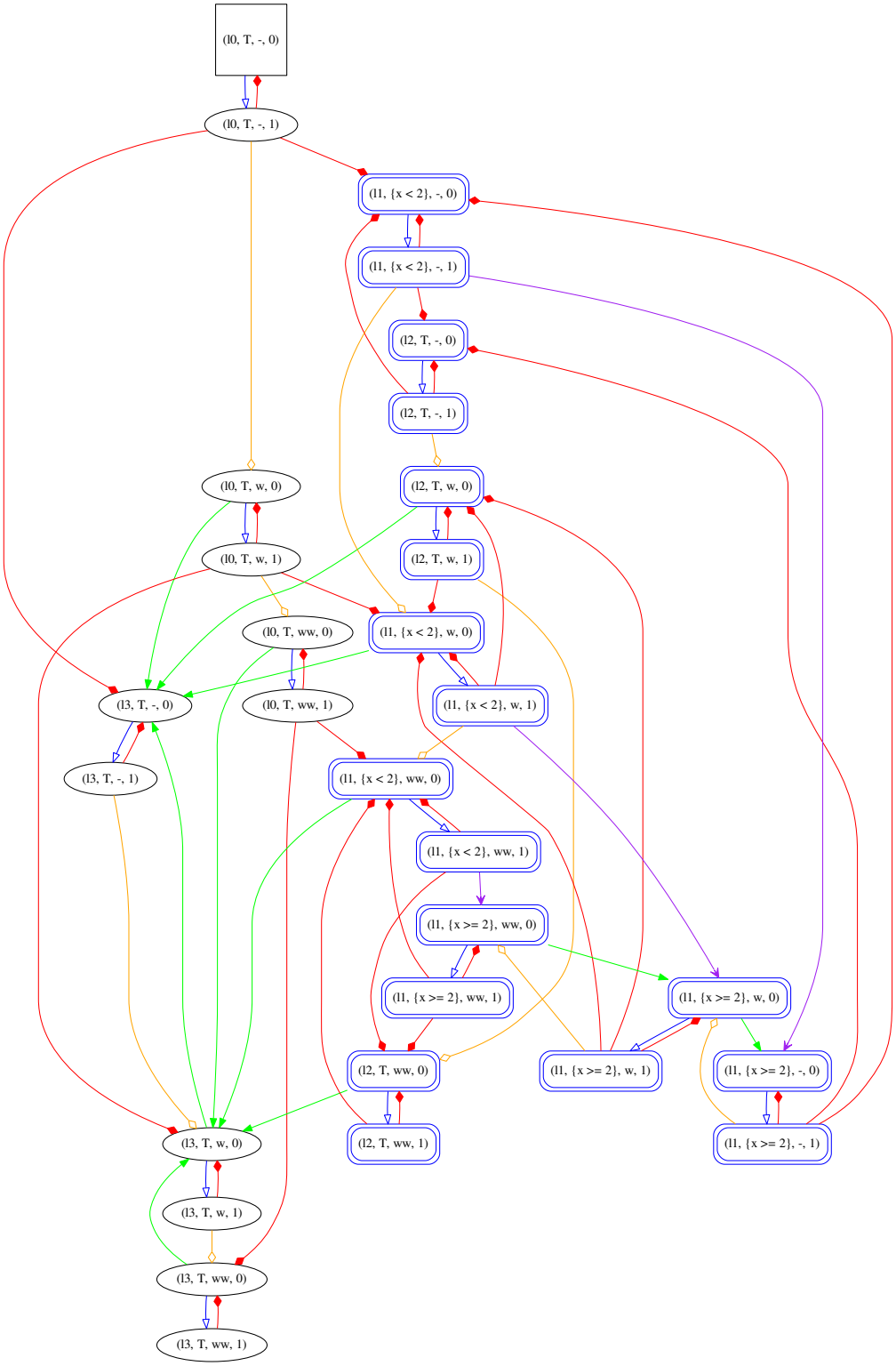


Figure 5. Game graph associated with property φ_t

with:

$\text{maxbuffer}(w) = \max_{\preccurlyeq}(\{w' \preccurlyeq w \mid w' \in \Sigma_c^n\})$.

Maximal safe word (κ_φ) For $q \in Q$, and $w \in \Sigma_c^*$, the *maximal safe word* κ_φ is defined by:

$$\kappa_\varphi(q,w) = \min_{\text{lex}}(\max_{\preccurlyeq}(\{\epsilon\} \cup \bigcup_{t' \in \mathrm{T}(q,w)} \{w' +_t t' \mid w' \in \text{Safe}(q \textit{ after } (\epsilon,t'),w)\}))$$

where:

$\mathrm{T}(q,w) = \{t' \in \mathbb{R}_{\geq 0} \mid \forall t'' < t', \text{Safe}(q \textit{ after } (\epsilon,t''),w) = \emptyset\}$,

Now that the safe emitting function and the maximal safe word have been defined, they can be used to define function E_φ. For this purpose, an intermediate function (store_φ) is necessary:

Definition 7 (E_φ). Let $\text{store}_\varphi : \text{tw}(\Sigma) \to \text{tw}(\Sigma) \times \Sigma_c^*$ be the function defined inductively by:

$\text{store}_\varphi(\epsilon) = \langle \epsilon, \epsilon \rangle$,

and for $\sigma \in \text{tw}(\Sigma)$, $(\delta,a) \in \mathbb{R}_{\geq 0} \times \Sigma$, if $t = \text{time}(\sigma \cdot (\delta,a))$, $(\sigma_\infty, \sigma_c) = \text{store}_\varphi(\sigma)$, and $\sigma_s = \text{obs}(\sigma_\infty, t)$, then

$$\text{store}_\varphi(\sigma \cdot (\delta,a)) = \begin{cases} \langle \sigma_s \cdot (t - \text{time}(\sigma_s), a) \cdot \sigma'_s, \sigma'_c \rangle \text{ if } a \in \Sigma_u \\ \langle \sigma_s \cdot \sigma''_s, \sigma''_c \rangle \text{ if } a \in \Sigma_c \end{cases}$$

where,

$\texttt{buf}_c = \Pi_\Sigma(\text{nobs}(\sigma_\infty, t)) \cdot \sigma_c$,

and

$\sigma'_s = \kappa_\varphi(\mathit{Reach}(\sigma_s \cdot (t - \text{time}(\sigma_s), a)), \texttt{buf}_c)$ $\quad$ $\sigma'_c = \Pi_\Sigma(\sigma'_s)^{-1} \cdot \texttt{buf}_c$,

$\sigma''_s = \kappa_\varphi(\mathit{Reach}(\sigma_s, t), \texttt{buf}_c \cdot a) +_t (t - \text{time}(\sigma_s))$ $\quad$ $\sigma''_c = \Pi_\Sigma(\sigma''_s)^{-1} \cdot (\texttt{buf}_c \cdot a)$.

We then define function E_φ as follows: For $\sigma \in \text{tw}(\Sigma)$ and $t \in \mathbb{R}_{\geq 0}$,

$\mathrm{E}_\varphi(\sigma, t) = \text{obs}(\Pi_1(\text{store}_\varphi(\text{obs}(\sigma, t))), t)$.

Function store_φ takes a timed word σ as input, and outputs two words: σ_∞ and σ_c. σ_∞ is a timed word corresponding to the output of the EM at infinite time, i.e. the events already emitted followed by the controllable events expected to be released according to the fact that no other event occurs meanwhile. σ_c is an untimed word representing the controllable events stored in order not to break the property. Note that contrary to [PFJ+14] when an event is in σ_∞, there is no guarantee that it will be emitted. Indeed, if an uncontrollable event occurs meanwhile, the latter has to be emitted immediately, and the state of the TA is updated. At this moment, it is necessary to check again if the events in σ_∞ may be emitted without violating the property. store_φ is defined in an inductive way, distinguishing the occurrence of an uncontrollable event and a controllable one. In both cases, it is necessary to compute the "best" correct sequence using the safe emitting function Safe. In the definition of $\text{Safe}(q,w)$, the last condition requires that all nodes of the game graph $\mathcal{G}$ that are reached by a word in $\text{Safe}(q,w)$ from q belong to W_0. This is a strong condition, that is required to ensure that it is always possible to compute a word leading to an accepting state. Nevertheless, if the source node is not in W_0, it is possible that letting time elapse leads to a node in W_0, because it disabled some transition in the timed automaton. This explains why we defined the set $\mathrm{T}(q,w)$, that allows us to consider words as potential outputs of the EM if it becomes sound (i.e. can ensure that the property will be satisfied) before the emission of the first event of this word, even if it is not at the time when the last event was received. Intuitively, $\mathrm{T}(q,w)$ contains all the delays t such that an EM must wait at least t time units to be able to be sound. In other words, the EM can not ensure that the property will eventually always be satisfied from state q with buffer w, and it can not ensure it either by waiting less than t time units, for every t in $\mathrm{T}(q,w)$. Then, $\kappa_\varphi(q,w)$ is the word that is to be output by the EM from state q with buffer w provided that the input does not change. It is the maximal word (with respect to $\preccurlyeq_d$) that belongs to $\text{Safe}(q,w)$. If $\text{Safe}(q,w)$ is empty, then $\kappa_\varphi(q,w)$ is the maximal word that belongs to $\text{Safe}(q \textit{ after } (\epsilon,t),w)$, where t is the minimal time for which $\text{Safe}(q \textit{ after } (\epsilon,t),w)$ is not empty. If $\text{Safe}(q \textit{ after } (\epsilon,t),w)$ is empty for every

Table 1. Table showing the evolution of σ_∞ and σ_c in store$_\varphi$ and the output of the EM for the input $\sigma = ((1, Auth) \cdot (1, LockOn) \cdot (2, \texttt{Write}) \cdot (1, LockOff) \cdot (1, LockOn) \cdot (1, \texttt{Write}) \cdot (1, LockOff))$.

t	Output of the EM at time t	σ_∞	σ_c
1	$(1, Auth)$	$(1, Auth)$	ϵ
2	$(1, Auth) \cdot (1, LockOn)$	$(1, Auth) \cdot (1, LockOn)$	ϵ
4	$(1, Auth) \cdot (1, LockOn)$	$(1, Auth) \cdot (1, LockOn)$	Write
5	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff)$	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (2, \texttt{Write})$	ϵ
6	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (1, LockOn)$	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (1, LockOn)$	Write
7	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (1, LockOn)$	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (1, LockOn)$	Write · Write
8	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (1, LockOn) \cdot (2, LockOff)$	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (1, LockOn) \cdot (2, LockOff) \cdot (2, \texttt{Write}) \cdot (0, \texttt{Write})$	ϵ
10	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (1, LockOn) \cdot (2, LockOff) \cdot (2, \texttt{Write}) \cdot (0, \texttt{Write})$	$(1, Auth) \cdot (1, LockOn) \cdot (3, LockOff) \cdot (1, LockOn) \cdot (2, LockOff) \cdot (2, \texttt{Write}) \cdot (0, \texttt{Write})$	ϵ

$t \in \mathbb{R}_{\geq 0}$, then $\kappa_\varphi(q, w) = \epsilon$, meaning that the EM does not output anything. Thus, when the enforcement function is not sound, it outputs nothing but uncontrollable events.

Example 6. We can follow in Table 1 the output of function E$_\varphi$ and the values of σ_∞ and σ_c over time with input word $\sigma = (1, Auth) \cdot (1, LockOn) \cdot (2, \texttt{Write}) \cdot (1, LockOff) \cdot (1, LockOn) \cdot (1, \texttt{Write}) \cdot (1, LockOff)$. As depicted before, at time $t = 4$ the controllable event `Write` occurs but can not be emitted. It is stored in σ_c. At $t = 5$, the occurrence of the uncontrollable event *LockOff* leads the TA into location l_1 and permits to consider `Write` to be released. Then "$(3, LockOff) \cdot (2, \texttt{Write})$" is added at the end of σ_∞, which means that *LockOff* should be emitted immediately, followed by the emission of `Write` expected 2 time units after. The occurrence at $t = 6$ of *LockOn* leads the TA to l_2. Emitting `Write` is not possible anymore, thus the latter is stored again in σ_c, and so on. To understand the behaviour of store$_\varphi$, note that in the associated game graph, shown in Fig. 5, $\langle l_1, Z, w, p\rangle \in W_0$ and $\langle l_2, Z, w, p\rangle \in W_0$, for any $\langle l_1, Z, w, p\rangle \in V$ and $\langle l_2, Z, w, p\rangle \in V$.

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. In Definition 7, function E$_\varphi$ is sound whenever T(q, w) is empty, with q the state reached by the output of E$_\varphi$ at date t and w its buffer at this date. If T(q, w) is empty, then the last value of σ'_s (or σ''_s depending on the controllability of the last input action) is in Safe(q, w), meaning that the node in the game graph $\mathcal{G}$ reached by the EM is in W_0, therefore it is always possible to compute a word that leads to a state in F_G. This leads us to the definition of the *stabilised set of inputs* Pre(φ), which is the time-extension-closed set of inputs in which function E$_\varphi$ is sound (i.e. the S set from the definition of soundness, see Definition 2):

Definition 8 (Stabilised set of inputs (Pre(φ))). Considering a property φ to enforce, the *stabilised set of inputs* Pre(φ) is defined by:

$$\text{Pre}(\varphi) = \{(\sigma, t) \mid \sigma \in \text{Pre}(\varphi, t)\},$$

where, for $\sigma \in \text{tw}(\Sigma)$ and $t \in \mathbb{R}_{\geq 0}$:

$$\text{Pre}(\varphi, t) = \{\sigma \in \text{tw}(\Sigma) \mid \exists t' \leq t, \text{Safe}(Reach(\sigma_{|\Sigma_u}, t'), \Pi_\Sigma(\text{obs}(\sigma, t')_{|\Sigma_c})) \neq \emptyset\} \cdot \text{tw}(\Sigma)$$

Note that Pre(φ) is time-extension-closed, meaning that once E$_\varphi$ is sound, its output will always eventually satisfy φ in the future. Considering that the output of function E$_\varphi$ was only containing uncontrollable events so far, if Safe$(Reach(\sigma_{|\Sigma_u}, t), \Pi_\Sigma(\text{obs}(\sigma, t)_{|\Sigma_c}))$ is not empty, this means that function E$_\varphi$ becomes sound with input σ from time t, since there is a word that is safe to emit. Thus, Pre(φ, t) is the set of inputs for which E$_\varphi$ is sound after date t, and then E$_\varphi$ is sound for any input in Pre(φ) after its associated date.

3.2.4. Properties of function E_φ

We list the different propositions stating that E_φ fulfils all the requirements of enforcement functions: it is an enforcement function, which is sound with respect to φ in $\mathrm{Pre}(\varphi)$, compliant with respect to Σ_u and Σ_c, and optimal in $\mathrm{Pre}(\varphi)$. Proofs can be found in appendix A.

Proposition 1. E_φ as per Definition 7 is an enforcement function, as per Definition 1.

Proposition 2. E_φ is sound with respect to φ in $\mathrm{Pre}(\varphi)$ as per Definition 2.

Proposition 3. E_φ is compliant, as per Definition 3.

Proposition 4. E_φ is optimal in $\mathrm{Pre}(\varphi)$ as per Definition 4.

3.3. Enforcement Monitors

In this part, we give an operational description of an EM whose output is exactly the output of E_φ, using a transition system obeying to a set of rules.

Definition 9. 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:

- $C^\mathcal{E} = \mathrm{tw}(\Sigma) \times \Sigma_\mathrm{c}^* \times Q \times \mathbb{R}_{\geq 0}$ is the set of configurations.
- $c_0^\mathcal{E} = \langle \epsilon, \epsilon, q_0, 0\rangle \in C^\mathcal{E}$ is the initial configuration.
- $\Gamma^\mathcal{E} = ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}) \times Op \times ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\})$ is the alphabet, composed of an optional input, an operation and an optional output.
  The set of operations is $\{\mathrm{dump}(.), \text{pass-uncont}(.), \text{store-cont}(.), \mathrm{delay}(.)\}$.
  Whenever $(\sigma, \bowtie, \sigma') \in \Gamma^\mathcal{E}$, it 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:
  - **Dump:** $\langle (\delta, a) \cdot \sigma_b, \sigma_c, q, \delta\rangle \xhookrightarrow{\epsilon / \mathrm{dump}((\delta,a))/(\delta,a)}_\mathcal{E} \langle \sigma_b, \sigma_c, q', 0\rangle$, with $q' = q$ *after* a,
  - **Pass-uncont:** $\langle \sigma_b, \sigma_c, q, \delta\rangle \xhookrightarrow{u / \text{pass-uncont}(u)/(\delta,u)}_\mathcal{E} \langle \sigma_b', \sigma_c', q', 0\rangle$, with $q' = q$ *after* u, $\sigma_b' = \kappa_\varphi(q', \Pi_\Sigma(\sigma_b) \cdot \sigma_c)$, and $\sigma_c' = \Pi_\Sigma(\sigma_b')^{-1} \cdot (\Pi_\Sigma(\sigma_b) \cdot \sigma_c)$,
  - **Store-cont:** $\langle \sigma_b, \sigma_c, q, \delta\rangle \xhookrightarrow{c / \text{store-cont}(c)/\epsilon}_\mathcal{E} \langle \sigma_b', \sigma_c', q, \delta\rangle$, with $\sigma_b' = \kappa_\varphi(q, \Pi_\Sigma(\sigma_b) \cdot \sigma_c \cdot c) +_\mathrm{t} \delta$ and $\sigma_c' = \Pi_\Sigma(\sigma_b')^{-1} \cdot (\Pi_\Sigma(\sigma_b) \cdot \sigma_c \cdot c)$,
  - **Delay:** $\langle \sigma_b, \sigma_c, (l, v), \delta\rangle \xhookrightarrow{\epsilon / \mathrm{delay}(\delta')/\epsilon}_\mathcal{E} \langle \sigma_b, \sigma_c, (l, v + \delta'), \delta + \delta'\rangle$.

In a configuration $(\sigma_b, \sigma_c, q, \delta)$, σ_b is the word of controllable events waiting to be emitted (at infinite time), which corresponds to σ_∞ of store_φ, but without the events already emitted. σ_c is the word of controllable events stored to avoid to break the property (similar to σ_c in store_φ), q is the current state in the TA of the property, and δ is the time elapsed since the emission of the last event. For each rule, the elements composing a configuration are updated. In the notation $\sigma / \bowtie / \sigma'$, σ and σ' can be seen as an input/output snapshot of the EM. σ corresponds to the occurrence of an event as input of the EM, and refers to the first event of the green input sequence in Fig. 6. Similarly, σ' corresponds to the EM releasing an event, and refers to the last event of the red output sequence in Fig. 6

Example 7. An example of execution of an enforcement monitor as per Definition 9 enforcing property φ_t (see Fig. 3) is given in Fig. 6. We use the following notation: in each line, 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. The input used for the monitor in Fig. 6 is the same as in Table 1: $(1, \mathit{Auth}) \cdot (1, \mathit{LockOn}) \cdot (2, \texttt{Write}) \cdot (1, \mathit{LockOff}) \cdot (1, \mathit{LockOn}) \cdot (1, \texttt{Write}) \cdot (1, \mathit{LockOff})$. In Fig. 6, valuations are represented as integers, giving the value of the only clock x of the property; *LockOff* is abbreviated as *off*, *LockOn* as *on*, and `Write` as `w`. First column depicts the dates of events, red text is the current output (σ_s) of the enforcement monitor, blue text shows the evolution of the first member of the configuration (σ_b) of the

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

Figure 6. Execution of an enforcement monitor with input $(1,\mathit{Auth})\cdot(1,\mathit{LockOn})\cdot(2,\mathtt{Write})\cdot(1,\mathit{LockOff})\cdot(1,\mathit{LockOn})\cdot(1,\mathtt{Write})\cdot(1,\mathit{LockOff})$

monitor and green text depicts the remaining input word at this date. The final output is the same as the one of the enforcement function E_φ as per Definition 7: $(1,\mathit{Auth})\cdot(1,\mathit{on})\cdot(2,\mathit{off})\cdot(1,\mathit{on})\cdot(2,\mathit{off})\cdot(2,\mathtt{w})\cdot(0,\mathtt{w})$.

Proposition 5. The output o of $\mathcal{E}$ as per Definition 9 for input σ at date t is such that $o=\mathrm{E}_\varphi(\sigma,t)$.

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.

Remark 1. For a configuration of the enforcement monitor $c=\langle\sigma_b,\sigma_c,q,\delta\rangle$, we can associate a node of the game graph $(\Pi_1(q),Z,w,0)$, such that $\Pi_2(q)\in Z$, and w is the longest prefix of the buffer of the enforcement monitor, i.e. $\Pi_\Sigma(\sigma_b)\cdot\sigma_c$, such that $(\Pi_1(q),Z,w,0)$ is a node of the game graph. Then, if $(\sigma,t)\in\mathrm{Pre}(\varphi)$, and c is the configuration reached by the enforcement monitor with input σ at date t, then $(\Pi_1(q),Z,w,0)$ is a winning node for player P_0.

4. GREP: Games for Runtime Enforcement of Properties

In this section, we present GREP (Games for Runtime Enforcement of Properties), a tool for RE of timed properties with uncontrollable events. GREP implements the RE framework described in Section 3. It is a tool of about 6,000 lines of code[2] developed using the C language. GREP is open-source, and is available at https://github.com/matthieurenard/GREP. Installation instructions are available in the *INSTALL* file in the repository. However, installing GREP can be tedious, due to some dependencies, thus we also provide a virtual machine (VM) with a pre-compiled version of GREP. This VM also contains all the tools required to run benchmarks and generate plots like the ones given in this section (see Section 4.4). For more information about how to get the VM and run the tool, see https://github.com/matthieurenard/testGREP.

In this section, we first recall general aspects of the approach used in GREP, then we present implementation details of the different modules composing it. Then we detail the possible options available, and finally we evaluate the performance and compare the results with another tool called TiPEX.

4.1. Description of the approach

The strategy of GREP is the one described in Section 3. Given a timed regular property φ, and a partition of its alphabet into a set of controllable events Σ_c and a set of uncontrollable events Σ_u, GREP first builds a symbolic graph that is compatible with Büchi games, as per Definition 5. The graph used is the one described in [ACH+92]. Then, GREP builds a game graph as per Definition 6, using Σ_c as the set of controllable events and Σ_u as the set of uncontrollable events. Once the game graph is constructed, GREP computes the set W_0 of winning nodes for player P_0 (the EM).

GREP can follow a real execution on the game graph, by watching the node that has been reached so far by its output, and the nodes that can be reached by emitting stored controllable actions (i.e. following the corresponding edges in the game graph). Whenever a winning node is reached by P_0, the strategy is to emit as many events as possible, remaining in a winning node all the time. Since the game played is a Büchi game, it is always possible for P_0 to stay in a winning node whenever one is reached. Whenever a winning node is reached, the output of the EM is then guaranteed to satisfy the property.

4.2. General Functioning of GREP

GREP is essentially composed of two modules (cf Fig. 7): the Symbolic Computing Module (SCM) and the Enforcement Monitor Module (EMM). It loads a TA file describing the desired property, and reads the inputs directly from *stdin*. The output of the EM is sent to *stdout*. This approach allows one to use GREP with off-the-shelf applications.

4.2.1. Symbolic Computing Module (SCM)

The Symbolic Computing Module is composed of three main components: a TA loader, the Zone Graph Generator, and the Game Graph Generator.

4.2.2. TA Loader

The TA loader is the component that parses a file containing the description of a timed automaton and loads it into a C structure. The file of the automaton is a textual description following a grammar designed for this purpose.

The file is parsed using a custom grammar, implemented using `lex` and `yacc`. The automaton must also be deterministic and complete (see [AD92]). If the automaton is not deterministic, the behaviour is undefined. Once the timed automaton is loaded, a symbolic graph is computed by the Zone Graph Generator to abstract its infinite semantics into a finite graph that is compatible with Büchi games, as per Definition 5. The graph that is built is actually the one described in [ACH+92].

[2] calculated with cloc (https://github.com/AlDanial/cloc)

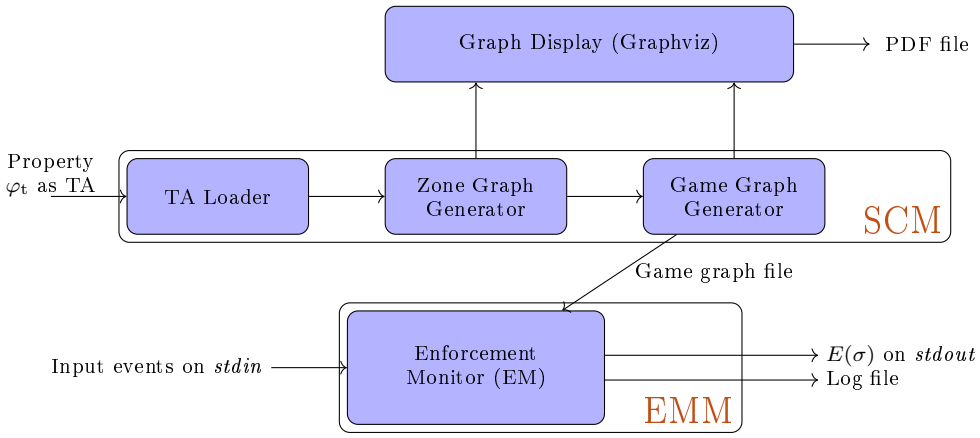


Figure 7. General architecture of GREP

4.2.3. Zone Graph Generator

From the TA, a symbolic graph is constructed using zones. This zone graph must be compatible with Büchi games, as per Definition 5. An algorithm to compute a symbolic graph compatible with Büchi games is given in [ACH+92]. This algorithm has been implemented to compute the symbolic graph in this module.

In GREP, zones are represented by Difference Bound Matrices (DBMs), using the UPPAAL DBM library (UDBM, see [UDB11]), and its C API. The algorithm requires some functionality not provided by this API (some of them exist in some higher-level wrappers), such as complementing zones into a list of zones. This functionality was added to our own wrapper of UDBM. No other third-party library was needed to compute the symbolic graph. This symbolic graph is used to build the final game graph, that will be used by the enforcement monitor.

4.2.4. Game Graph Generator

Using the symbolic graph, the Game Graph Generator builds a graph over which to play a Büchi game whose strategy is the one to be followed by the enforcement monitor. The graph is constructed as described in Definition 6. Once the graph is constructed, the Büchi game is solved for player P_0 (the enforcement monitor), with the set of Büchi nodes being the set of nodes whose location is accepting. The winning nodes are then the nodes from which the enforcement monitor ensures that its output will satisfy the property.

Following a path of winning nodes in the graph gives a strategy to follow such that the final output satisfies the property. This is how the EM uses the graph to actually enforce the property.

4.2.5. Enforcement Monitor Module (EMM)

The EMM uses the SCM to compute the output for a given input. It has five main public functions: init(G), getStrat(), delay(δ), eventRcvd(e), and emit(). Function init(G) initialises the EMM following the strategy from graph G. Function getStrat() gives the strategy to follow, i.e. whether the first action of the buffer should be output or not. Since time is abstracted by the zone graph for the SCM, it needs to be notified that some time has passed, which is done by the use of function delay(δ), where δ is the number of time units that have elapsed since the last call to delay(), or the creation of the EM for the first call. Time units only need to be consistent with the ones used in the property. Function eventRcvd(e) is used to inform the EMM that an event e has been read from the input. In this case, the EMM acts differently depending on the controllability of the event. Function emit() is used to output the first action of the buffer. Uncontrollable events are output by function eventRcvd(), as required by compliance.

Note that these functions allow to use the EMM in both online (on the fly) and offline (with a trace as input) settings. All these functions, except function getStrat(), return the number of time units required to reach the time successor of the current node (∞ if there is no time successor). It is the number of time units

given to function delay() if no event is received before and the strategy is not to emit. Algorithms for these functions are given in algorithms 1 to 5.

The EM is represented by a tuple containing the game graph, the current configuration, i.e. location, valuation and buffer, thus it is initialised by init($\mathcal{G}$), given in algorithm 1.

input : A game graph $\mathcal{G}$, that has all information about its related TA
output: An enforcement mechanism, represented by a tuple containing the graph, and the initial state, i.e. initial location, with all clocks evaluating to 0 and an empty buffer

Function `Init`($\mathcal{G}$)
  return $(\mathcal{G}, \langle l_0, \nu_0 \rangle, \epsilon)$;

Algorithm 1: Function `Init`($\mathcal{G}$)

The three functions delay(), eventRcvd(), and emit() are used to modify the EM. They implement the rules of the enforcement monitor described in Definition 9: function delay() corresponds to the rule **Delay**, function eventRcvd() corresponds to rules **Pass** and **Store**, depending on the controllability of the event received, and function emit() corresponds to rule **Dump**. These functions are described in algorithms 2 to 4. In the algorithms, only the actions are emitted, with no delays, because we considered that the enforcement mechanism was used in online mode, thus delays are not output, since they are computed from the real instant at which the event is written by function `writeOutput`(). These algorithms can easily be adapted to be used in offline mode, by adding a delay to the enforcement mechanism, that is increased by function delay(), and used as the delay of the actions that are written by function `writeOutput`() before being reset.

input : The EM $= (\mathcal{G}, q, w)$ as returned by function `Init` or another function described here, and a delay δ
output: The state of the EM after delay δ, and the delay needed to reach the next zone (∞ if there is not any)

Function `delay`$((\mathcal{G} = (V, E), q, w), \delta)$
  delayToNextZone $\leftarrow \min(\{\delta' \in \mathbb{R}_{\geq 0} \mid q \textit{ after } (\delta + \delta') \notin v$, with $v \in V$ such that $q \textit{ after } \delta \in v\} \cup \{\infty\})$;
  return $\langle (\mathcal{G}, q, w)$, delayToNextZone$\rangle$;

Algorithm 2: Function `delay`(EM, δ)

input : The EM $= (\mathcal{G}, q, w)$, and the event e that is received
output: The EM after having received the event e, and the delay to the time successor of the new zone, as in algorithm 2. If the event is uncontrollable, it is output (written) by function `writeOutput`

Function `eventRcvd`$((\mathcal{G} = (V, E), q, w), e)$
  if e *is controllable* **then**
    EM $\leftarrow (\mathcal{G}, q, w \cdot e)$;
    delayToNextZone $\leftarrow \min(\{\delta \in \mathbb{R}_{\geq 0} \mid q \textit{ after } \delta \notin v$, with $v \in V$ such that $q \in v\} \cup \{\infty\})$;
  else
    `/* e is uncontrollable */`
    EM $\leftarrow (\mathcal{G}, q \textit{ after } e, w)$;
    delayToNextZone $\leftarrow \min(\{\delta \in \mathbb{R}_{\geq 0} \mid q \textit{ after } e \textit{ after } \delta \notin v$ with $v \in V$ such that $q \in v\} \cup \{\infty\})$;
    `writeOutput`(e)
  end
  return (EM, delayToNextZone)

Algorithm 3: Function `eventRcvd`(EM, e)

The other primitive of our enforcement mechanism is function getStrat() that indicates whether the first event of the buffer should be emitted or not. This is the function in which the game graph, and the set

input : The EM $= (\mathcal{G}, q, w)$
output: The EM after having emitted the first event e of w, and the delay to the time successor of the new zone. Again, e is written with function `writeOutput`

1 **Function** `emit`$((\mathcal{G} = (V, E), q, w = e \cdot w'))$
2 | delayToNextZone $\leftarrow$ $\min(\{\delta \in \mathbb{R}_{\geq 0} \mid q \textit{ after } w(1) \textit{ after } \delta \notin v$, with $v \in V$ such that $q \textit{ after } w(1) \in v\} \cup \{\infty\})$;
3 | `writeOutput`$((w(1))$;
4 | **return** $\langle(\mathcal{G}, q \textit{ after } w(1), w_{[2..]}),$ delayToNextZone$\rangle$;

Algorithm 4: Function `emit`(EM)

of winning nodes W_0 are used to determine the strategy to follow. Algorithm 5 describes this function. In algorithm 5, a "path in W_0" is a path in $\mathcal{G}$ whose nodes belong to W_0. Moreover, we use $(q, w, 0)$ as a node to simplify, but its existence is not guaranteed in V. Normally, we should consider instead $(q, \text{maxbuffer}(w), 0)$ and search paths by simulating the reception of the remaining events of w that are not in maxbuffer(w) as soon as possible. Note that computing this condition requires to explore all the possible executions in order to compute the maximal number of green edges.

input : The EM $= (\mathcal{G}, q, w)$
output: EMIT if the strategy is to emit the first event of the buffer, DONTEMIT otherwise

1 **Function** `getStrat`$((\mathcal{G}, q, w))$
2 | **if** *there is a path in* W_0 *from* $(q, w, 0)$ *with maximal number of green edges (those in* E_2*) that starts with nodes* $(q, w, 0) \cdot (q \textit{ after } w(1), w_{[2..]}, 0)$ **then**
3 | | **return** EMIT ;
4 | **else**
5 | | **return** DONTEMIT
6 | **end**

Algorithm 5: Function `getStrat`(EM)

Then, the general algorithm to use the EMM in the offline setting is given in algorithm 6. Basically, the EMM builds an EM with function `Init`$(\mathcal{G})$, and then uses the primitive previously defined to produce a compliant output that is sound whenever possible. In algorithm 6, the EM is updated whenever a change of the zone happens (condition `del` $\leq \delta$, line 5), or an event is received (lines 12 and 13). The first event of the buffer of the EM is output if the strategy indicates so (i.e. `getStrat`() returns `EMIT`). Once all the events of the input have been read (line 14), the remaining events of the buffer can be emitted with good delays (line 15 to line 22).

Note that computing an output such that all actions are emitted whenever it is possible to emit them does not require to explore the strategy tree. Depending on the property, the two outputs could be the same. It is the case if the property is such that letting time elapse never enables a transition that eventually allows the EMM to output more events. Then the EMM can work faster by using an optimisation that does not compute any tree, but outputs actions whenever possible, i.e. when the successor node by emitting is winning, if it is specified to do so.

To visualise the difference between the two computations, consider the property described in Fig. 3. For this property, considering for instance that the input word is $(0, \texttt{Write}) \cdot (1, \texttt{Write})$, the output of GREP when exploring the execution tree would be (using delays): $(4, \texttt{Write}) \cdot (4, \texttt{Write})$, whereas using the other algorithm, that emits events as soon as possible, the output would be $(2, \texttt{Write})$. The first one outputs more events, but the second one outputs the `Write` event before.

To produce the second output, function `getStrat`() can be reimplemented as per algorithm 7.

4.3. Running GREP

GREP is shipped with two executables: one to use EM in offline mode, and the other in the online mode. Both of them take their input on the standard input. In the offline mode, the input is composed of events in the

input : The game graph $\mathcal{G}$, the input sequence of events, through function `read` ()
output: The output of the EM, written through functions `emit` and `eventRcvd`

```
1  EM ← Init(G);
2  del ← ∞;
3  while The input sequence has not been read entirely do
4  |  (δ, a) ← read();
5  |  while del ≤ δ do
6  |  |  δ ← δ − del;
7  |  |  (EM, del) ← delay(EM, del);
8  |  |  while getStrat(EM) = EMIT do
9  |  |  |  (EM, del) ← emit(EM);
10 |  |  end
11 |  end
12 |  (EM, del) ← delay(EM, δ);
13 |  (EM, del) ← eventRcvd(EM, a);
14 end
15 while del < ∞ or getStrat(EM) = EMIT do
16 |  while getStrat(EM) = EMIT do
17 |  |  (EM, del) ← emit(EM);
18 |  end
19 |  if del < ∞ then
20 |  |  (EM, del) ← delay(EM, del);
21 |  end
22 end
```

Algorithm 6: Main algorithm to enforce a property in offline mode

input : The EM $= (\mathcal{G}, q, w)$
output: EMIT if the strategy is to emit the first event of the buffer, DONTEMIT otherwise

```
1  Function getStratFast((G, q, w))
2  |  if (q after w(1), maxbuffer(w[2..]), 0) ∈ W0 then
3  |  |  return EMIT ;
4  |  else
5  |  |  return DONTEMIT
6  |  end
```

Algorithm 7: Function `getStratFast`(EM)

form (t, a), where t is a date and a is an action, controllable or uncontrollable. In the online mode, only the action is given, the date is computed from the real time through a call to `gettimeofday()`. Note that these executables may be built only on UNIX-like systems because of some system calls such as `gettimeofday()` and `clock_gettime()`. Excepting this, the tool is not system-dependent. The output (events with their dates) is printed on the standard output.

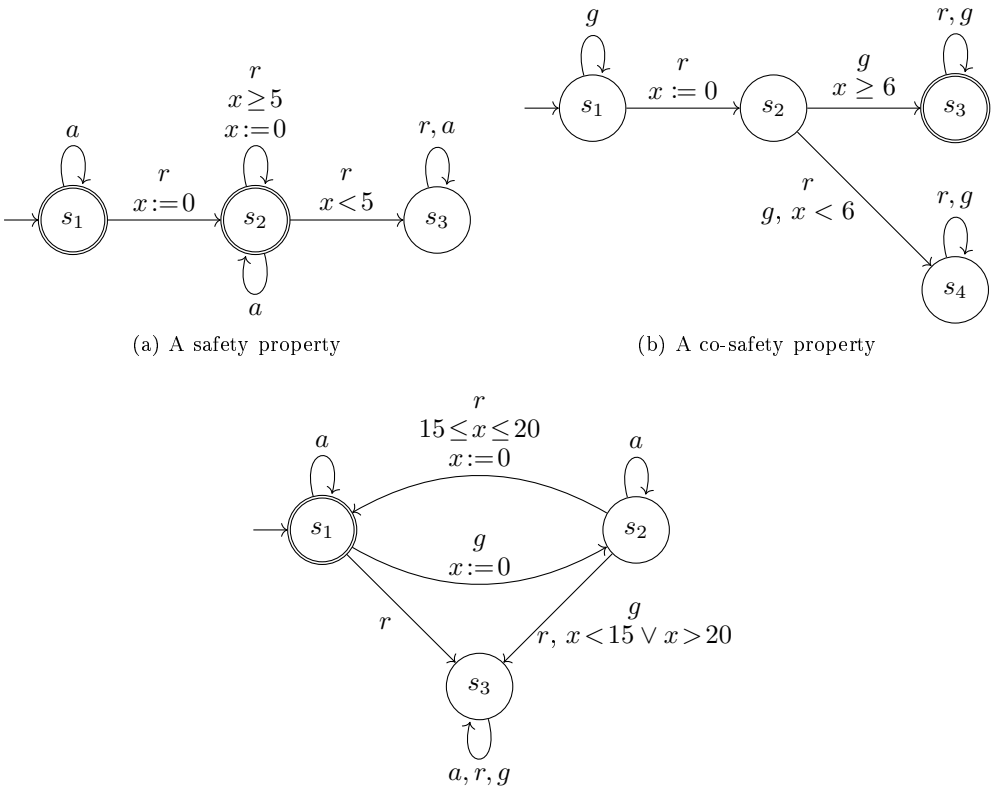
GREP provides a "fast" mode (`-f`), where actions are output whenever they can be instead of outputting the longest word possible with minimal dates. This option corresponds to using function `getStratFast`() presented in algorithm 7 instead of function `getStrat`() described in algorithm 5, which is used by default. Using option `-f` is usually faster, but the outputs might differ depending on the property.

Let us show an example of command for GREP:

```
game_enf_offline -a phit.tmtn -l log -d gameGraph.pdf < input
```

will enforce the property described in the file `phit.tmtn`, logging in the file `log`, reading its events from the file `input`. It will also draw the game graph in the file `gameGraph.pdf`.

The EM logs the mode in which it runs (default or fast) at the beginning, and when it stops, it logs the input, its output, the controllable actions that have not been output (what remains in its buffer), and a verdict



(a) A safety property

(b) A co-safety property

(c) A response property

Figure 8. Properties used to benchmark GREP

that is WIN if its output satisfies the property, or LOSS otherwise (some properties are not enforceable, as discussed in [RFR+17]).

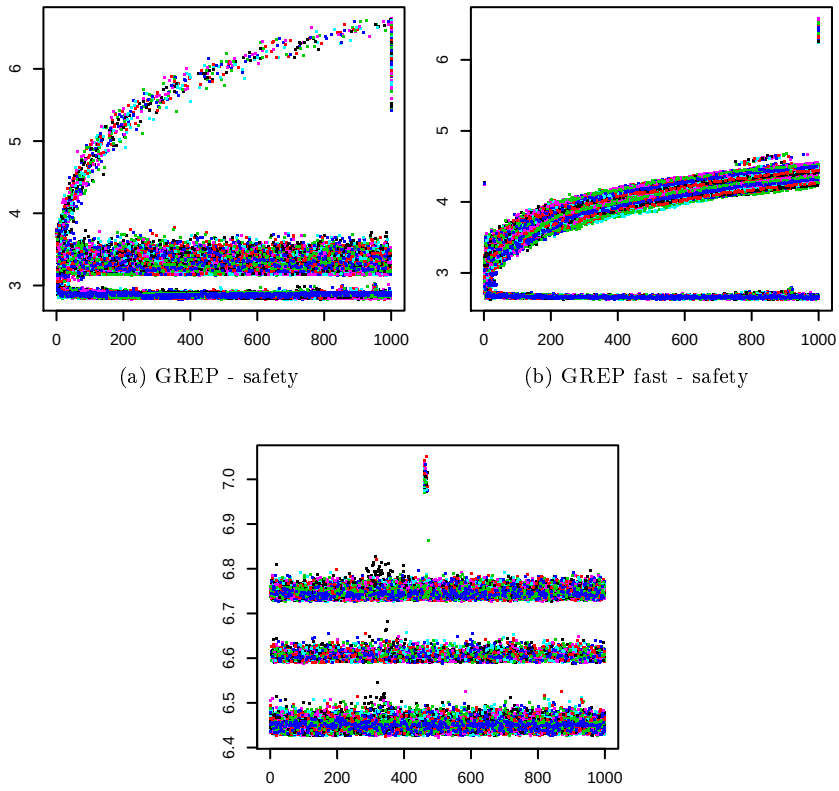
For example, considering property φ_t, the previous command with the input file containing the sequence $(0, \texttt{Write}) \cdot (1, \mathit{Auth}) \cdot (2, \texttt{Write}) \cdot (3, \mathit{LockOn}) \cdot (4, \texttt{Write}) \cdot (5, \mathit{LockOff})$ $(6, \mathit{LockOn}) \cdot (7, \mathit{LockOff})$, produces the output $(1, \mathit{Auth}) \cdot (2, \texttt{Write}) \cdot (2, \texttt{Write}) \cdot (3, \mathit{LockOn}) \cdot (5, \mathit{LockOff}) \cdot (6, \mathit{LockOn})$ $(7, \mathit{LockOff}) \cdot (9, \texttt{Write})$.

4.4. Performance Evaluation

4.4.1. Comparison with TiPEX

The performance of GREP has been evaluated on three properties that come along with TiPEX, the tool to which we compare. TiPEX (see [PFJM15]) is, to our knowledge, the only other tool that acts as an EM for timed regular properties. The three properties are described in Fig. 8. The safety property states that after the first r action, there should be at least 5 time units between any two subsequent r actions. Moreover, a actions are not constrained. The co-safety property states that the first r action should be followed by a g action, with a delay of at least 6 time units, and no intervening r action. After this prescribed sequence, g and r actions can happen freely. The response property states that every g action should be followed by a r action within 15 to 20 time units, without any g action occurring between them, and no r action should occur before its corresponding g action.

For each of these properties, GREP has been run 100 times on every input among 100 inputs of 1000 events randomly generated. The time between the reception of two events has been saved for all of these



(a) GREP - safety

(b) GREP fast - safety

(c) TiPEX - safety

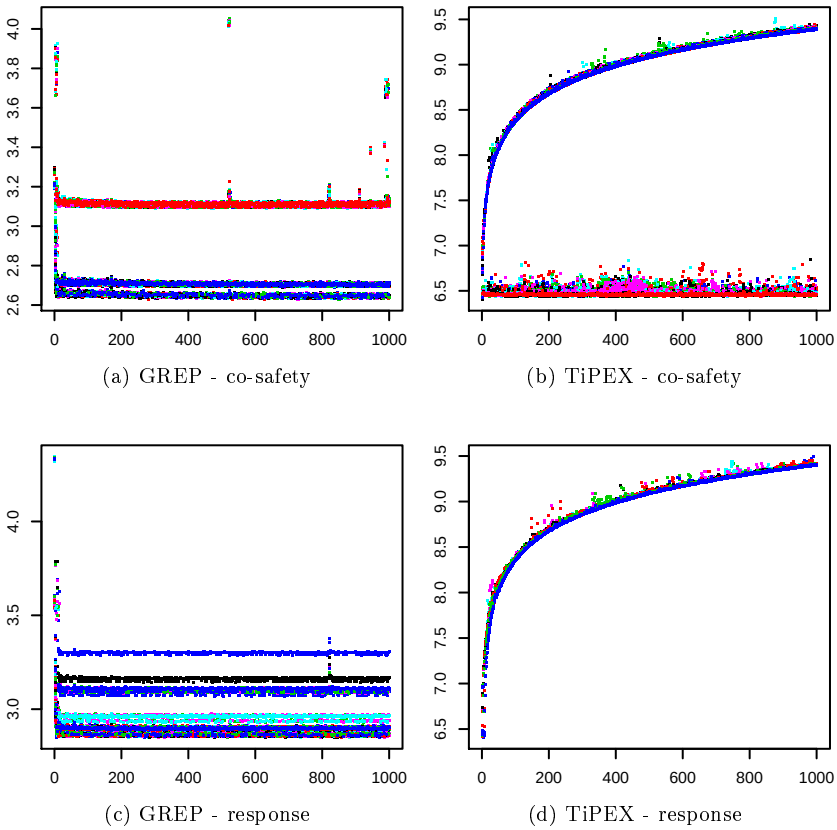
Figure 9. Comparison of timings of GREP and TiPEX on the safety property. "GREP fast" means that option `-f` is used. The x axis corresponds to the events of the input (from 1 to 1000), and the y axis corresponds to the logarithm of the timings (in nanoseconds) between the reads of the events.

executions. The same times have been computed for TiPEX[3], reducing the number of inputs and iterations to have the benchmarks run in a reasonable amount of time. Figures 9 and 10 give a graphical visualisation of the performance of GREP and TiPEX.

Figures 9 and 10 are obtained as follows: each input is iterated several times (100 for GREP, less for TiPEX[4]), and the computation times (in nanoseconds) of the tool between the reads of two consecutive events of the input are stored. Then, the median time is computed for each of these times between all the iterations. We then plot the logarithm (in base 10) of these times against the reads of the events. We use a logarithmic scale because many values are low, and they would be merged in a line when using a linear scale. The results for GREP with option `-f` are given only for the safety property because they are similar to the results without the option for the two other properties. We can see that GREP is faster than TiPEX by several orders of magnitude. GREP outputs many events in less than $10\,\mu s$ (4 on the scale of the graphs), whereas TiPEX takes at least 1 ms (6 on the scale of the graph) to output them. For the safety property, we can see that for some inputs, GREP takes an increasing amount of time to compute the strategy. This is due to the exploration of the strategy tree, which grows with the number of stored controllable actions. Using the optimised setting (`-f`) allows GREP to compute its output faster, as shown in Fig. 9b. The last vertical line has also many high values, because it represents the time to emit all the remaining actions after the last event from the input was read. For the co-safety and response properties, the time GREP takes between

[3] We patched TiPEX to retrieve the times as we do in our tool, only modifying it to get times properly, and did not change the behaviour inside the part that is being measured.

[4] For some properties, running TiPEX was too long to run it as many times as GREP.



(a) GREP - co-safety (b) TiPEX - co-safety

(c) GREP - response (d) TiPEX - response

Figure 10. Timings of GREP and TiPEX on the response and co-safety properties. The x axis corresponds to the events of the input (from 1 to 1000), and the y axis corresponds to the logarithm of the timings (in nanoseconds) between the reads of the events.

two events is less variable than for the safety property, mainly because the strategy of GREP is simpler: it consists in either emitting everything for the co-safety property (once state s_3 is reached) or emitting nothing for the response property, if the first stored controllable is an r while in state s_1. TiPEX, on the other hand, takes a linearly-increasing amount of time to emit some events.

Thus GREP performs globally better than TiPEX on these properties. Besides, another advantage of GREP over TiPEX is its ability to handle uncontrollable events. To our knowledge, no other RE tool has this functionality. However, using uncontrollable events can lower the performance of GREP, as discussed in Section 4.4.2.

4.4.2. Performance Evaluation with Uncontrollable Events

In this section, we show the limits of GREP when using uncontrollable events, with a property that is voluntarily designed to be hard to be enforced by GREP, at least in its default mode.

Consider property φ_{u} described in Fig. 11, with u an uncontrollable event and c a controllable one. This property has two locations, s_1 and s_2 that are symmetrical: both of them require that a certain delay (15 time units for s_1 and 10 time units for s_2) has elapsed since the last event to emit a c event. As in Section 4.4.1, GREP has been tested for this property, using 100 random inputs of 1000 events. The results are presented in Fig. 12. As in Section 4.4.1, the x-axis of the plots represents the events of the inputs, from 1 to 1000, and the y-axis is the logarithm of the timings, in nanoseconds, between the reads of two consecutive events. The timings have been plotted with (Fig. 12b) and without (Fig. 12a) option `-f`.

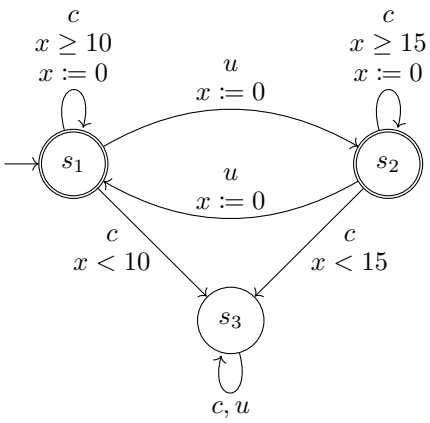
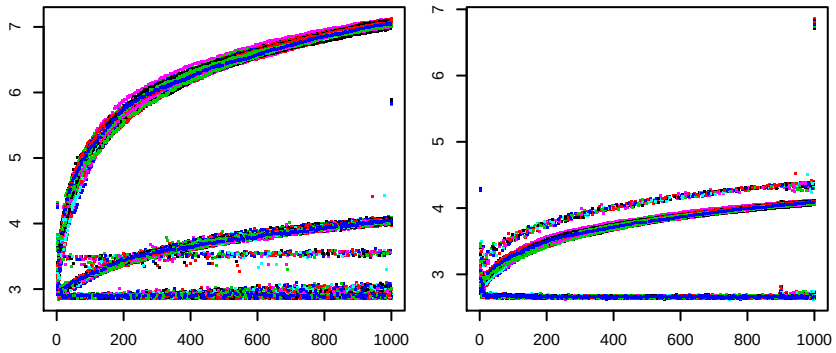


Figure 11. Property φ_{u}.



(a) Timings of GREP for φ_{u} without option `-f`.

(b) Timings of GREP for φ_{u} with option `-f`.

Figure 12. Timings of GREP for property φ_{u} with and without option `-f`.

Considering Fig. 12a, one can notice four apparent different behaviours: for some inputs, the timings between events is constant, and can be low, i.e. of about one microsecond, or a little bit higher, i.e. of about 10 microseconds; for some other inputs the timings are increasing, up to about 10 microseconds, or up to about 10 milliseconds for the last events. This difference between runs can be explained by the randomness of the events of the inputs. This benchmark has been made to show the limitation of GREP, thus the delays between events have been taken randomly between 0 and 3, meaning that events are received faster than it is possible to output controllable events (remember that c events must have a delay greater than 10 time units). Thus, depending on the proportion of uncontrollable events, that are emitted immediately, the buffer of stored controllable events grows as events are read. Property φ_{u} has been specifically designed to increase the number of stored controllable events.

Thus, in the worst case, the computation time of GREP increases with the size of its buffer. For some properties such as φ_{u}, receiving events with small delays (compared to guards) increases the size of the buffer, meaning that the computational overhead introduced by GREP could become too high for a use in online mode.

However, considering Fig. 12b, we can see that GREP performs better with option `-f`. Note that for φ_{u}, the outputs are the same with or without option `-f`. In the worst case, where GREP used 10 milliseconds without option `-f`, it only requires about 100 microseconds with option `-f`. In both cases, the timings increase with the size of the buffer, but option `-f` reduces the growth of the timings, and may allow using GREP in online mode where it is not possible without option `-f`. This difference between the use of option `-f` and not

using it can be explained by the fact that with option `-f`, GREP does not explore all the possible executions to output the longest word possible, but only decides if it is possible to emit a limited number of events.

4.4.3. Discussion

Limitations of GREP are theoretical and practical. Because of uncontrollable events, some properties may not be enforceable. The framework considers infinite buffers, but in practice, such a situation would inevitably lead to the saturation of the buffer. Adding bounds to the buffer in the theoretical framework is currently under investigation. Even in case of enforceable property, we have identified situations where the size of the buffer would inevitably grow. It is the case when the EM receives events with small delays compared to the required staying time in a state of the property (as shown in Fig. 12). In this case, the overhead may be too important to use the EM online. It depends on the inputs which are usually unpredictable. Another obstacle is hardware : such things are more likely to be used in embedded environments, which can have small computational power/memory.

GREP is a proof of concept tool. Despite these limitations, it fits pretty well in practice on usual properties.

5. Related Work

A first RE approach has been proposed by Schneider et al. in [Sch00]. They propose an EM synthesising *security automata* permitting to enforce safety properties described with Büchi automata. The EM watches the executions step by step and terminates whenever the property is going to be violated. Later, Bloem et al. present in [BKKW15] a framework to synthesise an enforcement mechanism (named *shield*) from safety automata by solving a 2-player game. A shield acts instantaneously and cannot buffer actions. It is *k-stabilising*: whenever a property violation is unavoidable, the shield allows to deviate from the property for k consecutive steps (as in [CEFJ15]). Whenever a second violation occurs within k steps, then the shield enters into a fail-safe mode, where it ensures only correctness. A similar approach has been proposed by Wu et al. in [WZW16], adding the ability to handle burst errors.

In [LBW09], Ligatti et al. synthesise *edit automata* from finite-state automata. They add in their framework the ability to insert or suppress events from the execution of the system, and use a memory to store the suffix of an invalid execution until it becomes valid again. Thus it permits to enforce a bigger set of properties than safety ones, and more precisely the set of *renewal properties*. Falcone et al. present in [FMFR11] a framework to enforce *response properties* from the *Safety-Progress* classification ([MP90, CMP92]). These properties are similar to the *infinite renewal properties* of [LBW09]. In [DLR15], the authors model enforcement mechanisms as Mandatory Results Automata (MRA). MRAs extend edit automata by refining the input/output relationship of an EM and thus allowing a more precise description of the enforcement abilities.

All these previously mentioned approaches deal only with untimed properties. The first RE framework considering timed properties has been proposed by Pinisetty et al. in [PFJ+13]. In this approach, the EM acts as a delaying filter on the input sequence of timed events in order to provide a correct output sequence according to the property modelled as a timed automaton ([AD92]). The framework of [PFJ+13] is able to enforce safety and co-safety properties, and has been generalised in [PFJ+14] for any regular timed property described by a timed automaton. Variants of this work have been proposed ([FJMP16] for suppressing events, [PFJM14a] for parametric properties), and a tool has been developed [PFJM15].

To our knowledge, very little work has been done on RE with uncontrollable events. Basin et al. extend in [BKZ11] the approach of Schneider et al. ([Sch00]), permitting to enforce safety properties where some of the events in the specification are uncontrollable. This work has been generalised in [BJKZ13], presenting enforcement of security policies with controllable and uncontrollable events. As in [Sch00], the system stops the execution in case of violation of the property. Basin et al. propose to adapt their framework for timed properties. In this case, time constraints are represented with special uncontrollable *tick* events. In our approach, we consider dense time using the expressiveness power of timed automata, any regular property, and our EM are more flexible since they block the system only when delaying events cannot prevent from violating the property, thus offering the possibility to correct more violations.

In previous work, we introduced runtime enforcement for timed properties described by timed automata, with uncontrollable events ([RFR+17]). We have adapted the usual notions of soundness, transparency

(changed to compliance) and optimality in this new context, and provided an approach to synthesise a sound, compliant and optimal EM. A first attempt of using games in the generation process has been proposed in [RRF17b]. This framework considers only untimed properties. A tool permitting to enforce timed properties with uncontrollable events and using game theory has been implemented and presented in [RRF17a].

6. Conclusion and Future Work

Conclusion This paper presents a complete RE framework for timed properties with uncontrollable events. It describes the theoretical framework based on a Büchi game approach, and presents GREP, the corresponding implemented tool. GREP builds a two-player game graph representing the possible actions of the EM and the system under scrutiny (that acts as the environment). Each vertex of the graph belongs to one of these two players, and each edge represents a possible action of the player owning the source vertex. GREP then solves a Büchi game by computing a set of nodes of the graph from which there exists a winning strategy. Using this approach allows to avoid the exploration of the whole execution tree at runtime. This paper provides benchmarks and compares the results with a state of the art tool TiPEX. They show that GREP provides better computing times in case of properties without uncontrollable events, since TiPEX does not handle uncontrollable events. To our knowledge, there exists no tool for enforcing timed properties with uncontrollable events.

Future work We propose several ways for future works. The proposed approach considers RE in an abstracted way. Even if GREP is build in a way permitting to plug it directly on a system using *stdin* and *stdout* flows, we did not consider practical aspects. In some real situations, instrumentation considerations may be investigated in order to plug GREP. Moreover, our EM is allowed only to delay events. Adding richer primitives, such as suppressing events for instance, could increase the enforcing power of the mechanism, and may be adapted to different situations. Besides, this framework considers only timed regular properties described by timed automata. One could build EM for properties with different formalism, such as TLTL properties for instance. Finally, in this work we focus on EM for single systems. Extending this for distributed systems is a challenging task. For instance, it may be necessary to use several EMs, implying communication (problems) between them. The enforcing ability of the EM according to distributed properties also appears to be a challenging open question.

A. Proofs

Proposition 1. E_φ as per Definition 7 is an enforcement function, as per Definition 1.

Proof. We have to prove the two following properties:

1. $\forall \sigma \in \mathrm{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, \forall t' \geq t,$
   $\mathrm{E}_\varphi(\sigma, t) \preccurlyeq \mathrm{E}_\varphi(\sigma, t')$
2. $\forall \sigma \in \mathrm{tw}(\Sigma), \forall \delta \in \mathbb{R}_{\geq 0}, \forall a \in \Sigma,$
   $\mathrm{E}_\varphi(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)))$

For $\sigma \in \mathrm{tw}(\Sigma)$, let $\mathrm{P}(\sigma)$ be the predicate "$\forall t \in \mathbb{R}_{\geq 0}, \forall t' \geq t, \forall (\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma, \mathrm{E}_\varphi(\sigma, t) \preccurlyeq \mathrm{E}_\varphi(\sigma, t') \wedge \mathrm{E}_\varphi(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)))$
$(\langle (\sigma, t), o_1 \rangle \in \mathrm{E}_\varphi \wedge \langle (\sigma, t'), o_2 \rangle \in \mathrm{E}_\varphi \wedge \langle (\sigma, \mathrm{time}(\sigma \cdot (\delta, a))), o_3 \rangle \in \mathrm{E}_\varphi \wedge \langle (\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))), o_4 \rangle \in \mathrm{E}_\varphi) \implies (o_1 \preccurlyeq o_2 \wedge o_3 \preccurlyeq o_4)$".

Let us show by induction that $\mathrm{P}(\sigma)$ holds for any $\sigma \in \mathrm{tw}(\Sigma)$:

Induction basis: for $\sigma = \epsilon$, let us consider $t \in \mathbb{R}_{\geq 0}$ and $t' \geq t$. Then, $\mathrm{E}_\varphi(\epsilon, t) = \epsilon = \mathrm{E}_\varphi(\epsilon, t') \preccurlyeq \mathrm{E}_\varphi(\epsilon, t')$. Moreover, for $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $\mathrm{E}_\varphi(\sigma, \delta) = \epsilon \preccurlyeq \mathrm{E}_\varphi((\delta, a), \delta)$ Thus, $\mathrm{P}(\epsilon)$ holds.

Induction step: suppose that $\mathrm{P}(\sigma)$ holds for some $\sigma \in \mathrm{tw}(\Sigma)$. Then, let us consider $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $t \in \mathbb{R}_{\geq 0}$, and $t' \geq t$. We first prove that the first condition holds. Let us consider $\langle (\sigma, t), o_1 \rangle \in \mathrm{E}_\varphi$, $\langle (\sigma, t'), o_2 \rangle \in \mathrm{E}_\varphi$, $\langle (\sigma \cdot (\delta, a), t), o_1' \rangle \in \mathrm{E}_\varphi$, and $\langle (\sigma \cdot (\delta, a), t'), o_2' \rangle \in \mathrm{E}_\varphi$. We have to prove that $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t')$.

Three cases are possible:

1. $t \leq t' < \mathrm{time}(\sigma \cdot (\delta, a))$. Then $\mathrm{obs}(\sigma \cdot (\delta, a), t') = \mathrm{obs}(\sigma, t')$, and $\mathrm{obs}(\sigma \cdot (\delta, a), t) = \mathrm{obs}(\sigma, t)$. Let us consider $\langle \sigma_{s1}, \sigma_c \rangle = \mathrm{store}_\varphi(\mathrm{obs}(\sigma, t))$ and $\langle \sigma_{s2}, \sigma_c' \rangle = \mathrm{store}_\varphi(\mathrm{obs}(\sigma, t'))$. Then, considering the definition of E_φ (Definition 7), $\mathrm{E}_\varphi(\sigma, t) = \mathrm{obs}(\sigma_{s1}, t) = \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t)$, and $\mathrm{E}_\varphi(\sigma, t') = \mathrm{obs}(\sigma_{s2}, t') = \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t')$ (since $\mathrm{obs}(\sigma, t) = \mathrm{obs}(\sigma \cdot (\delta, a), t)$ and $\mathrm{obs}(\sigma, t') = \mathrm{obs}(\sigma \cdot (\delta, a), t')$). Following the induction hypothesis, $\mathrm{P}(\sigma)$ holds, meaning that $\mathrm{E}_\varphi(\sigma, t) \preccurlyeq \mathrm{E}_\varphi(\sigma, t')$. This means that $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t')$.
2. $t < \mathrm{time}(\sigma \cdot (\delta, a)) \leq t'$. Then, $\mathrm{obs}(\sigma \cdot (\delta, a), t) = \mathrm{obs}(\sigma, t)$, meaning that (see previous case) $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t) = \mathrm{E}_\varphi(\sigma, t)$. Following the induction hypothesis, since $\mathrm{P}(\sigma)$ holds, $\mathrm{E}_\varphi(\sigma, t) \preccurlyeq \mathrm{E}_\varphi(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a)))$. Thus, we have to show that $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t')$. Since $\mathrm{time}(\sigma \cdot (\delta, a)) \leq t'$, $\mathrm{obs}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))) = \sigma \cdot (\delta, a) = \mathrm{obs}(\sigma \cdot (\delta, a), t')$, thus if $\langle \sigma_{s2}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma \cdot (\delta, a))$, then $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))) = \mathrm{obs}(\sigma_{s2}, \mathrm{time}(\sigma \cdot (\delta, a)))$, and $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t') = \mathrm{obs}(\sigma_{s2}, t')$. Since $\mathrm{time}(\sigma \cdot (\delta, a)) \leq t'$, this means that $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a))) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t')$.
   Thus $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t')$.
3. $\mathrm{time}(\sigma \cdot (\delta, a)) \leq t \leq t'$. Then, $\mathrm{obs}(\sigma \cdot (\delta, a), t) = \mathrm{obs}(\sigma \cdot (\delta, a), t') = \sigma \cdot (\delta, a)$. Thus, if $\langle \sigma_{s0}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma \cdot (\delta, a))$, then $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t) = \mathrm{obs}(\sigma_{s0}, t)$ and $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t') = \mathrm{obs}(\sigma_{s0}, t')$. Since $t \leq t'$, this means that $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t')$.

Thus, in all cases, the first required condition holds (i.e. $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), t) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a), t')$).

Let us now consider $(\delta', a') \in \mathbb{R}_{\geq 0} \times \Sigma$. We have to show that $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a'))) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a) \cdot (\delta', a'), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a')))$. Since $\mathrm{obs}(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a'))) = \sigma \cdot (\delta, a)$ and $\mathrm{obs}(\sigma \cdot (\delta, a) \cdot (\delta', a'), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a'))) = \sigma \cdot (\delta, a) \cdot (\delta', a')$, if $\langle \sigma_{s3}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma \cdot (\delta, a))$ and $\langle \sigma_{s4}, \sigma_c' \rangle = \mathrm{store}_\varphi(\sigma \cdot (\delta, a) \cdot (\delta', a'))$, then $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a'))) = \mathrm{obs}(\sigma_{s3}, \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a')))$ and $\mathrm{E}_\varphi(\sigma \cdot (\delta, a) \cdot (\delta', a'), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a'))) = \mathrm{obs}(\sigma_{s4}, \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a')))$. Following the definition of store_φ (Definition 7), it is clear that $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a'))) \preccurlyeq \sigma_{s4}$. Thus, since $\mathrm{time}(\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a')))) \leq \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a'))$, $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a'))) \preccurlyeq \mathrm{E}_\varphi(\sigma \cdot (\delta, a) \cdot (\delta', a'), \mathrm{time}(\sigma \cdot (\delta, a) \cdot (\delta', a')))$.

This means that $\mathrm{P}(\sigma \cdot (\delta, a))$ holds.

Thus, for any $\sigma \in \mathrm{tw}(\Sigma)$ and $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma \cdot (\delta, a))$.

Thus, by induction on σ, $\mathrm{P}(\sigma)$ holds for any $\sigma \in \mathrm{tw}(\Sigma)$. Thus E_φ is an enforcement function. □

Lemma 1. $\forall \sigma \in \mathrm{tw}(\Sigma), \forall t \geq \mathrm{time}(\sigma), (\sigma \notin \mathrm{Pre}(\varphi, t) \wedge \langle \sigma_{s0}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma)) \implies (\mathrm{obs}(\sigma_{s0}, t) = \sigma_{|\Sigma_\mathrm{u}} \wedge \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, t)) \cdot \sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}))$.

Proof. For $\sigma \in \mathrm{tw}(\Sigma)$ and $t \geq \mathrm{time}(\sigma)$, let $\mathrm{P}(\sigma, t)$ be the predicate "$(\sigma \notin \mathrm{Pre}(\varphi, t) \wedge \langle \sigma_{s0}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma)) \implies (\mathrm{obs}(\sigma_{s0}, t) = \sigma_{|\Sigma_\mathrm{u}} \wedge \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, t)) \cdot \sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}))$", and $\mathrm{P}(\sigma)$ be the predicate "$\forall t \geq \mathrm{time}(\sigma), \mathrm{P}(\sigma, t)$". Let us then prove by induction on σ that $\mathrm{P}(\sigma)$ holds for any $\sigma \in \mathrm{tw}(\Sigma)$.

Induction basis: for $\sigma = \epsilon$, let us consider $t \in \mathbb{R}_{\geq 0}$. Then, $\mathrm{store}_\varphi(\epsilon) = \langle \epsilon, \epsilon \rangle$. Since $\mathrm{obs}(\epsilon, t) = \epsilon_{|\Sigma_\mathrm{u}}$, and $\Pi_\Sigma(\mathrm{nobs}(\epsilon_{|\Sigma_\mathrm{c}}, t)) \cdot \epsilon = \Pi_\Sigma(\epsilon_{|\Sigma_\mathrm{c}})$, it follows that $\mathrm{P}(\epsilon, t)$ holds, and thus $\mathrm{P}(\epsilon)$ holds.

Induction step: let us suppose that for $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Let us consider $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $\langle \sigma_{s0}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma)$, $\langle \sigma_{t0}, \sigma_d \rangle = \mathrm{store}_\varphi(\sigma \cdot (\delta, a))$, and $\sigma_s = \mathrm{obs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))$. Let us also consider $t \geq \mathrm{time}(\sigma \cdot (\delta, a))$.
If $\sigma \cdot (\delta, a) \in \mathrm{Pre}(\varphi, t)$, then $\mathrm{P}(\sigma \cdot (\delta, a), t)$ trivially holds.
Let us consider that $\sigma \cdot (\delta, a) \notin \mathrm{Pre}(\varphi, t)$.

- If $a \in \Sigma_\mathrm{u}$, then $\sigma_{t0} = \sigma_s \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a) \cdot \sigma'_s$ for some $\sigma'_s \in \mathrm{tw}(\Sigma)$. Since $\sigma \cdot (\delta, a) \notin \mathrm{Pre}(\varphi, t)$, this means that for any $t' \leq t$, $\mathrm{Safe}(\mathit{Reach}((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{u}}, t'), \Pi_\Sigma(\mathrm{obs}(\sigma \cdot (\delta, a), t'))_{|\Sigma_\mathrm{c}}) = \emptyset$. This means that for any $t' \leq t - \mathrm{time}(\sigma \cdot (\delta, a))$, $t' \notin \mathrm{T}(\mathit{Reach}((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{u}}), \Pi_\Sigma(\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}})$. Now, by induction hypothesis, since $\sigma \notin \mathrm{Pre}(\varphi, \mathrm{time}(\sigma \cdot (\delta, a)))$ (otherwise $\sigma \cdot (\delta, a)$ would be in $\mathrm{Pre}(\varphi, t)$), $\sigma_{|\Sigma_\mathrm{u}} = \sigma_s$, and $\Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))))_{|\Sigma_\mathrm{c}} \cdot \sigma_c = \Pi_\Sigma(\sigma)_{|\Sigma_\mathrm{c}}$. Thus, for any $t' \leq t - \mathrm{time}(\sigma \cdot (\delta, a))$, $t' \notin \mathrm{T}(\mathit{Reach}(\sigma_s \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a)), \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c)$. Thus, $\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma \cdot (\delta, a))) = \epsilon$. It follows that $\mathrm{obs}(\sigma_{t0}, t) = \sigma_s \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a) \cdot \mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma \cdot (\delta, a))) = \sigma_s \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a) = (\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{u}}$ and $\Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t)) \cdot \sigma_d = \sigma'_s \cdot \sigma_d = \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c = \Pi_\Sigma(\sigma)_{|\Sigma_\mathrm{c}} = \Pi_\Sigma((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}})$. Thus $\mathrm{P}(\sigma \cdot (\delta, a), t)$ holds.
- Otherwise, $a \in \Sigma_c$, and there exists σ''_s such that $\sigma_{t0} = \sigma_s \cdot \sigma''_s$. Since $\sigma \cdot (\delta, a) \notin \mathrm{Pre}(\varphi, t)$, for any $t' \leq t$, $\mathrm{Safe}(\mathit{Reach}((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{u}}, t'), \Pi_\Sigma(\mathrm{obs}(\sigma \cdot (\delta, a), t'))_{|\Sigma_\mathrm{c}}) = \emptyset$. Thus, for any $t' \leq t - \mathrm{time}((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{u}})$, $t' \notin \mathrm{T}(\mathit{Reach}((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{u}}), \Pi_\Sigma((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}}))$. Now, by induction hypothesis, considering that $(\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{u}} = \sigma_{|\Sigma_\mathrm{u}}$ and $(\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}} = \sigma_{|\Sigma_\mathrm{c}} \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_{|\Sigma_\mathrm{c}}), a)$, and since $\sigma \notin \mathrm{Pre}(\varphi, \mathrm{time}(\sigma \cdot (\delta, a)))$, for any $t' \leq t - \mathrm{time}(\sigma \cdot (\delta, a))$, $t' \notin \mathrm{T}(\mathit{Reach}(\sigma_s, \mathrm{time}(\sigma \cdot (\delta, a))), \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c \cdot a)$. Thus, $\mathrm{obs}(\sigma''_s -_\mathrm{t} (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s)), t - \mathrm{time}(\sigma \cdot (\delta, a))) = \epsilon$. Thus, $\mathrm{obs}(\sigma''_s, t - \mathrm{time}(\sigma \cdot (\delta, a)) + \mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s)) -_\mathrm{t} (t - \mathrm{time}(\sigma \cdot (\delta, a))) = \epsilon$, meaning that $\mathrm{obs}(\sigma''_s, t - \mathrm{time}(\sigma_s)) = \epsilon$.
Thus, $\mathrm{obs}(\sigma_{t0}, t) = \sigma_s \cdot \mathrm{obs}(\sigma''_s, t - \mathrm{time}(\sigma_s)) = \sigma_s = \sigma_{|\Sigma_\mathrm{u}}$, and $\Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t)) \cdot \sigma_d = \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c \cdot a = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}}) \cdot a = \Pi_\Sigma((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}})$.
Thus $\mathrm{P}(\sigma \cdot (\delta, a), t)$ holds.

In both cases, $\mathrm{P}(\sigma \cdot (\delta, a), t)$ holds. Thus, it holds for any $t \geq \mathrm{time}(\sigma \cdot (\delta, a))$, meaning that $\mathrm{P}(\sigma \cdot (\delta, a))$ holds.
This means that for any $\sigma \in \mathrm{tw}(\Sigma)$ and $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma \cdot (\delta, a))$.

Thus, we have shown by induction on σ that $\mathrm{P}(\sigma)$ holds for any $\sigma \in \mathrm{tw}(\Sigma)$. □

Lemma 2. $\forall q \in Q, \forall w \in \Sigma_\mathrm{c}^*, \forall \sigma \in \mathrm{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, \sigma \in \mathrm{Safe}(q, w) \implies \mathrm{nobs}(\sigma, t) -_\mathrm{t} (t - \mathrm{time}(\mathrm{obs}(\sigma, t))) \in \mathrm{Safe}(q \mathit{\ after\ } (\sigma, t), \Pi_\Sigma(\mathrm{obs}(\sigma, t))^{-1} \cdot w)$.

Proof. Let us consider q, w and σ such that $\sigma \in \mathrm{Safe}(q, w)$, and $t \in \mathbb{R}_{\geq 0}$. Following the definition of Safe, this means that the three following properties hold:

1. $\Pi_\Sigma(\sigma) \preccurlyeq w$,
2. $q \mathit{\ after\ } \sigma \in F_G$,
3. $\forall t \in \mathbb{R}_{\geq 0}, \forall v \in V_\mathrm{s}$,
   $q \mathit{\ after\ } (\sigma, t) \in v \implies \langle v, \mathrm{maxbuffer}(\Pi_\Sigma(\mathrm{obs}(\sigma, t))^{-1} \cdot w), 1 \rangle \in W_0$.

Now, considering $\sigma' = \mathrm{nobs}(\sigma, t) -_\mathrm{t} (t - \mathrm{time}(\mathrm{obs}(\sigma, t)))$, σ' satisfies the following properties:

1. $\Pi_\Sigma(\sigma') = \Pi_\Sigma(\mathrm{nobs}(\sigma, t) -_\mathrm{t} (t - \mathrm{time}(\mathrm{obs}(\sigma, t))))$
   $= \Pi_\Sigma(\mathrm{nobs}(\sigma, t))$,
   thus,
   $\Pi_\Sigma(\mathrm{obs}(\sigma, t)) \cdot \Pi_\Sigma(\sigma') = \Pi_\Sigma(\mathrm{obs}(\sigma, t)) \cdot \Pi_\Sigma(\mathrm{nobs}(\sigma, t))$
   $= \Pi_\Sigma(\sigma)$.

Since $\Pi_\Sigma(\sigma) \preccurlyeq w$, this means that:
$\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\text{obs}(\sigma,t))^{-1} \cdot w$.

2. $(q \textit{ after } (\sigma,t)) \textit{ after } \sigma' = (q \textit{ after } (\sigma,t)) \textit{ after } (\text{nobs}(\sigma,t) -_\text{t} (t - \text{time}(\text{obs}(\sigma,t))))$
$= q \textit{ after } \sigma.$
Thus, $(q \textit{ after } (\sigma,t)) \textit{ after } \sigma' \in F_G$.

3. For $t' \in \mathbb{R}_{\geq 0}$,
$(q \textit{ after } (\sigma,t)) \textit{ after } (\sigma',t') = (q \textit{ after } (\sigma,t)) \textit{ after } (\text{nobs}(\sigma,t) -_\text{t} (t - \text{time}(\text{obs}(\sigma,t))), t')$
$= q \textit{ after } (\sigma, t+t').$
Since $t+t' \in \mathbb{R}_{\geq 0}$, then if $v \in V_\text{s}$ is such that $(q \textit{ after } (\sigma,t)) \textit{ after } (\sigma',t') \in v$, then $\langle v, \text{maxbuffer}(\Pi_\Sigma(\text{obs}(\sigma, t+t'))^{-1} \cdot w), 1\rangle \in W_0$. Moreover, since $t \geq \text{time}(\text{obs}(\sigma,t))$, $\Pi_\Sigma(\text{obs}(\sigma,t+t'))^{-1} \cdot w = \Pi_\Sigma(\text{obs}(\sigma',t'))^{-1} \cdot (\Pi_\Sigma(\text{obs}(\sigma,t))^{-1} \cdot w)$. Thus, $\langle v, \text{maxbuffer}(\Pi_\Sigma(\text{obs}(\sigma',t'))^{-1} \cdot (\Pi_\Sigma(\text{obs}(\sigma,\, t))^{-1} \cdot w)), 1\rangle \in W_0$.

This means that $\sigma' = \text{nobs}(\sigma,t) -_\text{t} (t - \text{time}(\text{obs}(\sigma,t))) \in \text{Safe}(q \textit{ after } (\sigma,t), \Pi_\Sigma(\text{obs}(\sigma,t))^{-1} \cdot w)$. □

Proposition 2. E_φ is sound with respect to φ in $\text{Pre}(\varphi)$ as per Definition 2.

Proof. Notation from Definition 7 is to be used in this proof: for $\sigma \in \text{tw}(\Sigma)$, if $\langle \sigma_{s0}, \sigma_c\rangle = \text{store}_\varphi(\sigma)$, $(\delta,a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $t = \text{time}(\sigma \cdot (\delta,a))$, and $\sigma_s = \text{obs}(\sigma_{s0}, t)$, then,

$$\texttt{buf}_\text{c} = \Pi_\Sigma(\text{nobs}(\sigma_\infty, t)) \cdot \sigma_c,$$

and

$$\sigma'_s = \kappa_\varphi(\mathit{Reach}(\sigma_s \cdot (t - \text{time}(\sigma_s), a)), \texttt{buf}_\text{c}) \qquad \sigma'_c = \Pi_\Sigma(\sigma'_s)^{-1} \cdot \texttt{buf}_\text{c},$$
$$\sigma''_s = \kappa_\varphi(\mathit{Reach}(\sigma_s, t), \texttt{buf}_\text{c} \cdot a) +_\text{t} (t - \text{time}(\sigma_s)) \qquad \sigma''_c = \Pi_\Sigma(\sigma''_s)^{-1} \cdot (\texttt{buf}_\text{c} \cdot a).$$

We have to prove that for any $(\sigma,t) \in \text{Pre}(\varphi)$, there exists $t' \geq t$ such that for any $t'' \geq t'$, $\text{E}_\varphi(\sigma,t'') \models \varphi$.
For $\sigma \in \text{tw}(\Sigma)$, and $t \geq \text{time}(\sigma)$, let $\text{P}(\sigma,t)$ be the predicate "$((\sigma,t) \in \text{Pre}(\varphi) \wedge \langle \sigma_s, \sigma_c\rangle = \text{store}_\varphi(\sigma)) \implies (\sigma_s \models \varphi \wedge \text{nobs}(\sigma_s,t) -_\text{t} (t - \text{time}(\text{obs}(\sigma_s,\, t))) \in \text{Safe}(\mathit{Reach}(\sigma_s,t), \Pi_\Sigma(\text{nobs}(\sigma_s,t)) \cdot \sigma_c))$". Let also $\text{P}(\sigma)$ be the predicate: "$\forall t \geq \text{time}(\sigma), \text{P}(\sigma,t)$". Let us show by induction that for any $\sigma \in \text{tw}(\Sigma)$, $\text{P}(\sigma)$ holds.

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

- If $\epsilon \notin \text{Pre}(\varphi,t)$, then, $\text{P}(\epsilon)$ trivially holds.
- Otherwise, $\epsilon \in \text{Pre}(\varphi,t)$. Then, following the definition of $\text{Pre}(\epsilon,t)$ (Definition 8), there exists $t' \leq t$ such that $\text{Safe}(\mathit{Reach}(\epsilon_{|\Sigma_\text{u}}, t'), \epsilon) \neq \emptyset$, meaning that $\text{Safe}(\mathit{Reach}(\epsilon,t'),\epsilon) \neq \emptyset$. Thus, following the definition of $\text{Safe}(\mathit{Reach}(\epsilon,t'),\epsilon)$, $\epsilon \in \text{Safe}(\mathit{Reach}(\epsilon,t'),\epsilon)$, and $\mathit{Reach}(\epsilon) \in F_G$, thus $\epsilon \models \varphi$. Since $\text{store}_\varphi(\epsilon) = \langle \epsilon, \epsilon\rangle$ and $\epsilon \models \varphi$, $\text{P}(\epsilon,t)$ holds.

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

Induction step: suppose that for $\sigma \in \text{tw}(\Sigma)$, $\text{P}(\sigma)$ holds. Let us consider $(\delta,a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $t \geq \text{time}(\sigma \cdot (\delta,a))$, $\langle \sigma_{s0}, \sigma_c\rangle = \text{store}_\varphi(\sigma)$, $\sigma_s = \text{obs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta,a)))$, $\langle \sigma_{t0}, \sigma_d\rangle = \text{store}_\varphi(\sigma \cdot (\delta,a))$, and $\sigma_t = \text{obs}(\sigma_{t0}, t)$. We have to prove that $(\sigma \cdot (\delta,a), t) \in \text{Pre}(\varphi) \implies \sigma_t \models \varphi \wedge \text{nobs}(\sigma_t,t) -_\text{t} (t - \text{time}(\text{obs}(\sigma_t,t))) \in \text{Safe}(\mathit{Reach}(\sigma_t,t), \Pi_\Sigma(\text{nobs}(\sigma_t,t)) \cdot \sigma_d)$.

- If $\sigma \cdot (\delta,a) \notin \text{Pre}(\varphi,t)$, then $\text{P}(\sigma \cdot (\delta,a), t)$ trivially holds.
- If $\sigma \cdot (\delta,a) \in \text{Pre}(\varphi,t) \wedge \sigma \notin \text{Pre}(\varphi, \text{time}(\sigma \cdot (\delta,a)))$, then, since $\sigma \notin \text{Pre}(\varphi, \text{time}(\sigma \cdot (\delta,a)))$, following lemma 1, since $\text{obs}(\sigma, \text{time}(\sigma \cdot (\delta,a))) = \sigma$, $\text{obs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta,a))) = \sigma_s = \text{obs}(\sigma_{|\Sigma_\text{u}}, \text{time}(\sigma \cdot (\delta,a))) = \sigma_{|\Sigma_\text{u}}$ and $\Pi_\Sigma(\text{nobs}(\sigma_{s0},\, \text{time}(\sigma \cdot (\delta,a)))) \cdot \sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_\text{c}})$. Since $\sigma \cdot (\delta,a) \in \text{Pre}(\varphi,t)$, and $\sigma \notin \text{Pre}(\varphi, \text{time}(\sigma \cdot (\delta,a)))$, following the definition of $\text{Pre}(\varphi,t)$ and $\text{Pre}(\varphi, \text{time}(\sigma \cdot (\delta,a)))$ (Definition 8), there exists $t' \in \mathbb{R}_{\geq 0}$ such that $\text{time}(\sigma \cdot (\delta,a)) \leq t' \leq t$, and $\text{Safe}(\mathit{Reach}((\sigma \cdot (\delta,a))_{|\Sigma_\text{u}}, t'), \Pi_\Sigma(\text{obs}(\sigma \cdot (\delta,a), t')_{|\Sigma_\text{c}})) \neq \emptyset$. Let us consider the minimum such t'. Since $t' \geq \text{time}(\sigma \cdot (\delta,a))$, then $\text{obs}(\sigma \cdot (\delta,a), t') = \sigma \cdot (\delta,a)$. This means that:

$$\text{Safe}(\mathit{Reach}((\sigma \cdot (\delta,a))_{|\Sigma_\text{u}}, t'), \Pi_\Sigma((\sigma \cdot (\delta,a))_{|\Sigma_\text{c}})) \neq \emptyset. \tag{1}$$

– If $a \in \Sigma_u$, then considering that $(\sigma \cdot (\delta, a))_{|\Sigma_u} = \sigma_{|\Sigma_u} \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_{|\Sigma_u}), a) = \sigma_s \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s), a)$, and $\Pi_\Sigma(\text{nobs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c = \Pi_\Sigma(\sigma_{|\Sigma_c}) = \Pi_\Sigma((\sigma \cdot (\delta, a))_{|\Sigma_c})$, (1) becomes:

$$\text{Safe}(\mathit{Reach}(\sigma_s \cdot (\,\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s), a), t'), \\ \Pi_\Sigma(\text{nobs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c) \neq \emptyset.$$

Let us consider $\delta' = \text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s)$, such that $\text{time}(\sigma \cdot (\delta, a)) = \text{time}(\sigma_s \cdot (\delta', a))$, and $t'' = t' - \text{time}(\sigma \cdot (\delta, a))$.
Then, t' is the minimum number such that $\text{Safe}(\mathit{Reach}(\sigma_s \cdot (\delta', a),\ t'), \texttt{buf}_c) \neq \emptyset$. Therefore, $t'' \in \text{T}(\mathit{Reach}(\sigma_s \cdot (\delta', a)), \texttt{buf}_c)$.
Thus, there exists $w' \in \text{Safe}(\mathit{Reach}(\sigma_s \cdot (\delta', a))\ \mathit{after}\ (\epsilon, t''), \texttt{buf}_c)$ such that $\sigma'_s = w' +_t t''$. Thus, $\sigma'_s -_t t'' \in \text{Safe}(\mathit{Reach}(\sigma_s \cdot (\delta', a), t'), \texttt{buf}_c)$.
Now, note that:

$$\begin{aligned}\text{nobs}(\sigma_{t0}, t) &= \text{obs}(\sigma_{t0}, t)^{-1} \cdot \sigma_{t0} \\ &= \text{obs}(\sigma_s \cdot (\delta', a) \cdot \sigma'_s, t)^{-1} \cdot \sigma_{t0}\end{aligned}$$

Since $t \geq \text{time}(\sigma \cdot (\delta, a)) = \text{time}(\sigma_s \cdot (\delta', a))$,

$$\begin{aligned}\text{nobs}(\sigma_{t0}, t) &= (\sigma_s \cdot (\delta', a) \cdot \text{obs}(\sigma'_s, t - \text{time}(\sigma_s \cdot (\delta', a))))^{-1} \cdot \sigma_{t0} \\ &= \text{obs}(\sigma'_s, t - \text{time}(\sigma \cdot (\delta, a)))^{-1} \cdot \\ &\qquad ((\sigma_s \cdot (\delta', a))^{-1} \cdot (\sigma_s \cdot (\delta', a) \cdot \sigma'_s)) \\ &= \text{obs}(\sigma'_s, t - \text{time}(\sigma \cdot (\delta, a)))^{-1} \cdot \sigma'_s \\ &= \text{nobs}(\sigma'_s, t - \text{time}(\sigma \cdot (\delta, a)))\end{aligned}$$

We know that $\sigma'_s -_t t'' \in \text{Safe}(\mathit{Reach}(\sigma_s \cdot (\delta', a), t'), \texttt{buf}_c)$, thus following lemma 2, since $t \geq t', t - t' \geq 0$, and

$$\begin{aligned}&\text{nobs}(\sigma'_s -_t t'', t - t') -_t (t - t' - \text{time}(\text{obs}(\sigma'_s -_t t'', t - t'))) \in \\ &\quad \text{Safe}(\mathit{Reach}(\sigma_s \cdot (\delta', a), t')\ \mathit{after}\ (\sigma'_s -_t t'', t - t'), \\ &\quad\quad \Pi_\Sigma(\text{obs}(\sigma'_s -_t t'', t - t'))^{-1} \cdot \texttt{buf}_c)\end{aligned} \tag{2}$$

Now, note that for any $\sigma \in \text{tw}(\Sigma)$, $t \in \mathbb{R}_{\geq 0}$ and $t' \in \mathbb{R}_{\geq 0}$,

$$\text{nobs}(\sigma -_t t, t') = \begin{cases} \text{nobs}(\sigma, t + t') -_t t' \text{ if } \texttt{delay}(\sigma(1)) > t + t' \\ \text{nobs}(\sigma, t + t') \text{ otherwise} \end{cases}$$

The reason is that the operator $-_t$ affects only the first delay of the word, thus if this delay is in $\text{obs}(\sigma -_t t, t')$, i.e. $\texttt{delay}(\sigma(1)) \geq t + t'$, the remaining events are not changed by the $-_t$ operator.
Thus, if $\texttt{delay}(\sigma'_s(1)) > t - t' + t'' = t - t' + t' - \text{time}(\sigma \cdot (\delta, a)) = t - \text{time}(\sigma \cdot (\delta, a))$, then

$$\begin{aligned}\text{nobs}(\sigma'_s -_t t'', t - t') &= \text{nobs}(\sigma'_s, t - t' + t'') -_t t'' \\ &= \text{nobs}(\sigma'_s, t - \text{time}(\sigma \cdot (\delta, a))) -_t t''\end{aligned}$$

Moreover, since $\texttt{delay}(\sigma'_s(1)) > t - \text{time}(\sigma \cdot (\delta, a))$, $\text{obs}(\sigma'_s -_t t'', t - t') = \epsilon$, and $\text{obs}(\sigma_{t0}, t) = \sigma_s \cdot (\delta, a)$, thus:

$$\begin{aligned}&\text{nobs}(\sigma'_s -_t t'', t - t') -_t (t - t' - \text{time}(\text{obs}(\sigma'_s -_t t'', t - t'))) \\ &\quad = (\text{nobs}(\sigma'_s, t - \text{time}(\sigma \cdot (\delta, a))) -_t t'') -_t (t - t') \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_t (t - t' + t'') \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_t (t - \text{time}(\sigma \cdot (\delta, a))) \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_t (t - \text{time}(\text{obs}(\sigma_{t0}, t)))\end{aligned}$$

On the other hand, if $\texttt{delay}(\sigma'_s(1)) \leq t - \text{time}(\sigma \cdot (\delta, a))$, then
$\text{nobs}(\sigma'_s -_t t'', t - t') = \text{nobs}(\sigma'_s, t - \text{time}(\sigma \cdot (\delta, a)))$
Moreover, since $\texttt{delay}(\sigma'_s(1)) \leq t - \text{time}(\sigma \cdot (\delta, a))$, $\text{obs}(\sigma'_s -_t t'', t - t') = \text{obs}(\sigma'_s, t - \text{time}(\sigma \cdot (\delta, a))) -_t t''$, thus

$$
\begin{aligned}
&\mathrm{nobs}(\sigma'_s -_\mathrm{t} t'', t-t') -_\mathrm{t} (t - t' - \mathrm{time}(\mathrm{obs}(\sigma'_s -_\mathrm{t} t'', t-t'))) \\
&\quad = \mathrm{nobs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))) \\
&\qquad -_\mathrm{t} (t - t' - \mathrm{time}(\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))) -_\mathrm{t} t'')) \\
&\quad = \mathrm{nobs}(\sigma_{t0}, t) \\
&\qquad -_\mathrm{t} (t - t' + t'' - \mathrm{time}(\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))))) \\
&\quad = \mathrm{nobs}(\sigma_{t0}, t) -_\mathrm{t} (t - \mathrm{time}(\sigma\cdot(\delta,a)) - \\
&\qquad \mathrm{time}(\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))))) \\
&\quad = \mathrm{nobs}(\sigma_{t0}, t) -_\mathrm{t} (t - \mathrm{time}(\sigma\cdot(\delta,a) - \\
&\qquad (\mathrm{time}(\mathrm{obs}(\sigma_{t0}, t)) - \mathrm{time}(\sigma\cdot(\delta,a))))) \\
&\quad = \mathrm{nobs}(\sigma_{t0}, t) -_\mathrm{t} (t - \mathrm{time}(\mathrm{obs}(\sigma_{t0}, t)))
\end{aligned}
$$

Thus, in both cases, (2) becomes:

$$
\begin{aligned}
&\mathrm{nobs}(\sigma_{t0}, t) -_\mathrm{t} (t - \mathrm{time}(\mathrm{obs}(\sigma_{t0}, t))) \in \\
&\quad \mathrm{Safe}(\mathit{Reach}(\sigma_s\cdot(\delta',a), t') \mathit{\ after\ } (\sigma'_s -_\mathrm{t} t'', t-t'), \\
&\qquad \Pi_\Sigma(\mathrm{obs}(\sigma'_s -_\mathrm{t} t'', t-t'))^{-1} \cdot \mathtt{buf}_\mathrm{c})
\end{aligned}
$$

Now, since $t' \geq \mathrm{time}(\sigma\cdot(\delta,a))$,

$$
\begin{aligned}
&\mathit{Reach}(\sigma_s\cdot(\delta',a), t') \mathit{\ after\ } (\sigma'_s -_\mathrm{t} t'', t-t') \\
&\quad = \mathit{Reach}(\sigma_s\cdot(\delta',a)) \mathit{\ after} \\
&\qquad (\epsilon, t' - \mathrm{time}(\sigma_s\cdot(\delta',a))) \mathit{\ after\ } (\sigma'_s -_\mathrm{t} t'', t-t') \\
&\quad = \mathit{Reach}(\sigma_s\cdot(\delta',a)) \mathit{\ after} \\
&\qquad ((\sigma'_s -_\mathrm{t} t'') +_\mathrm{t} t'', t - t' + t'') \\
&\quad = \mathit{Reach}(\sigma_s\cdot(\delta',a)) \mathit{\ after\ } (\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))) \\
&\quad = \mathit{Reach}(\sigma_s\cdot(\delta',a)\cdot\sigma'_s, t) \\
&\quad = \mathit{Reach}(\sigma_{t0}, t)
\end{aligned}
$$

and

$$
\begin{aligned}
&\Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t))\cdot\sigma_d \\
&\quad = \Pi_\Sigma(\mathrm{nobs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))))\cdot(\Pi_\Sigma(\sigma'_s)^{-1}\cdot\mathtt{buf}_\mathrm{c}) \\
&\quad = \Pi_\Sigma(\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a)))^{-1}\cdot\sigma'_s)\cdot \\
&\qquad (\Pi_\Sigma(\sigma'_s)^{-1}\cdot\mathtt{buf}_\mathrm{c}) \\
&\quad = \Pi_\Sigma(\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))))^{-1}\cdot\Pi_\Sigma(\sigma'_s)\cdot \\
&\qquad (\Pi_\Sigma(\sigma'_s)^{-1}\cdot\mathtt{buf}_\mathrm{c}) \\
&\quad = \Pi_\Sigma(\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))))^{-1}\cdot\mathtt{buf}_\mathrm{c}
\end{aligned}
$$

On the other hand,

$$
\begin{aligned}
&\Pi_\Sigma(\mathrm{obs}(\sigma'_s -_\mathrm{t} t'', t-t'))^{-1}\cdot\mathtt{buf}_\mathrm{c} \\
&\quad = \Pi_\Sigma(\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))) -_\mathrm{t} t'')^{-1}\cdot\mathtt{buf}_\mathrm{c} \\
&\quad = \Pi_\Sigma(\mathrm{obs}(\sigma'_s, t - \mathrm{time}(\sigma\cdot(\delta,a))))^{-1}\cdot\mathtt{buf}_\mathrm{c}
\end{aligned}
$$

Thus, $\Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t))\cdot\sigma_d = \Pi_\Sigma(\mathrm{obs}(\sigma'_s -_\mathrm{t} t'', t-t'))^{-1}\cdot\mathtt{buf}_\mathrm{c}$.
Considering all this, (2) becomes:

$$
\begin{aligned}
&\mathrm{nobs}(\sigma_{t0}, t) -_\mathrm{t} (t - \mathrm{time}(\mathrm{obs}(\sigma_{t0}, t))) \in \\
&\quad \mathrm{Safe}(\mathit{Reach}(\sigma_{t0}, t), \Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t))\cdot\sigma_d)
\end{aligned}
$$

– Otherwise, $a \in \Sigma_\mathrm{c}$. Then, $(\sigma\cdot(\delta,a))_{|\Sigma_\mathrm{u}} = \sigma_{|\Sigma_\mathrm{u}} = \sigma_s$, and $\Pi_\Sigma((\sigma\cdot(\delta,a))_{|\Sigma_\mathrm{c}}) = \Pi_\Sigma(\sigma_{|\Sigma_\mathrm{c}})\cdot a = \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma\cdot(\delta,a))))\cdot\sigma_c\cdot a$. Thus, (1) becomes:

$$
\mathrm{Safe}(\mathit{Reach}(\sigma_s, t'), \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma\cdot(\delta,a))))\cdot\sigma_c\cdot a) \neq \emptyset
$$

Since t' is the minimum date that satisfies this equation, $t' - \text{time}(\sigma \cdot (\delta, a)) \in \text{T}(\mathit{Reach}(\sigma_s, \text{time}(\sigma \cdot (\delta, a))), \texttt{buf}_\text{c} \cdot a)$.
Thus, there exists $w' \in \text{Safe}(\mathit{Reach}(\sigma_s, \text{time}(\sigma \cdot (\delta, a)))\ \mathit{after}\ (\epsilon, t' - \text{time}(\sigma \cdot (\delta, a))), \texttt{buf}_\text{c} \cdot a)$ such that $\sigma''_s = (w' +_\text{t} t' - \text{time}(\sigma \cdot (\delta, a))) +_\text{t} (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s))$. Thus, $\sigma''_s -_\text{t} (t' - \text{time}(\sigma \cdot (\delta, a)) + \text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s)) = \sigma''_s -_\text{t} (t' - \text{time}(\sigma_s)) \in \text{Safe}(\mathit{Reach}(\sigma_s, t'), \texttt{buf}_\text{c} \cdot a)$. Let us consider $t'' = t' - \text{time}(\sigma_s)$, such that $\sigma''_s -_\text{t} t'' \in \text{Safe}(\mathit{Reach}(\sigma_s, t'), \texttt{buf}_\text{c} \cdot a)$.
Now,

$$\begin{aligned}\text{nobs}(\sigma_{t0}, t) &= \text{obs}(\sigma_{t0}, t)^{-1} \cdot \sigma_{t0} \\ &= \text{obs}(\sigma_s \cdot \sigma''_s, t)^{-1} \cdot \sigma_{t0}\end{aligned}$$

Since $\text{time}(\sigma_s) \leq t$, it follows that

$$\begin{aligned}\text{nobs}(\sigma_{t0}, t) &= (\sigma_s \cdot \text{obs}(\sigma''_s, t - \text{time}(\sigma_s)))^{-1} \cdot (\sigma_s \cdot \sigma''_s) \\ &= \text{obs}(\sigma''_s, t - \text{time}(\sigma_s))^{-1} \cdot (\sigma_s^{-1} \cdot (\sigma_s \cdot \sigma''_s)) \\ &= \text{obs}(\sigma''_s, t - \text{time}(\sigma_s))^{-1} \cdot \sigma''_s \\ &= \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s))\end{aligned}$$

We know that $\sigma''_s -_\text{t} t'' \in \text{Safe}(\mathit{Reach}(\sigma_s, t'), \texttt{buf}_\text{c} \cdot a)$ and that $t \geq t'$ meaning that $t - t' \geq 0$. Thus, following lemma 2:

$$\begin{aligned}&\text{nobs}(\sigma''_s -_\text{t} t'', t - t') -_\text{t} (t - t' - \text{time}(\text{obs}(\sigma''_s -_\text{t} t'', t - t'))) \in \\ &\quad \text{Safe}(\mathit{Reach}(\sigma_s, t')\ \mathit{after}\ (\sigma''_s -_\text{t} t'', t - t'), \\ &\qquad \Pi_\Sigma(\text{obs}(\sigma''_s -_\text{t} t'', t - t'))^{-1} \cdot (\texttt{buf}_\text{c} \cdot a))\end{aligned}$$

If $\texttt{delay}(\sigma''_s(1)) > t - \text{time}(\sigma_s)$ (i.e. $\texttt{delay}((\sigma''_s -_\text{t} t'')(1)) > t - t'$), then:

$$\begin{aligned}\text{nobs}(\sigma''_s -_\text{t} t'', t - t') &= \text{nobs}(\sigma''_s, t - t' + t'') -_\text{t} t'' \\ &= \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s)) -_\text{t} t''\end{aligned}$$

and $\text{obs}(\sigma''_s -_\text{t} t'', t - t') = \epsilon$ and $\text{obs}(\sigma_{t0}, t) = \sigma_s$. Thus,

$$\begin{aligned}&\text{nobs}(\sigma''_s -_\text{t} t'', t - t') -_\text{t} (t - t' - \text{time}(\text{obs}(\sigma''_s -_\text{t} t'', t - t'))) \\ &\quad = (\text{nobs}(\sigma''_s, t - \text{time}(\sigma_s)) -_\text{t} t'') -_\text{t} (t - t') \\ &\quad = \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s)) -_\text{t} (t - t' + t'') \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_\text{t} (t - \text{time}(\sigma_s)) \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_\text{t} (t - \text{time}(\text{obs}(\sigma_{t0}, t)))\end{aligned}$$

Otherwise, $\texttt{delay}(\sigma''_s(1)) \leq t - \text{time}(\sigma_s)$, and then $\text{nobs}(\sigma''_s -_\text{t} t'', t - t') = \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s))$, thus:

$$\begin{aligned}&\text{nobs}(\sigma''_s -_\text{t} t'', t - t') -_\text{t} (t - t' - \text{time}(\text{obs}(\sigma''_s -_\text{t} t'', t - t'))) \\ &\quad = \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s)) -_\text{t} \\ &\qquad (t - t' - \text{time}(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s)) -_\text{t} t'')) \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_\text{t} \\ &\qquad (t - t' - (\text{time}(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s))) - t'')) \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_\text{t} \\ &\qquad (t - t' + t'' - (\text{time}(\text{obs}(\sigma_{t0}, t)) - \text{time}(\sigma_s))) \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_\text{t} \\ &\qquad (t - \text{time}(\sigma_s) - \text{time}(\text{obs}(\sigma_{t0}, t)) + \text{time}(\sigma_s)) \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_\text{t} (t - \text{time}(\text{obs}(\sigma_{t0}, t)))\end{aligned}$$

Thus, in both cases, this means that

$$\begin{aligned}&\text{nobs}(\sigma_{t0}, t) -_\text{t} (t - \text{time}(\text{obs}(\sigma_{t0}, t))) \in \\ &\quad \text{Safe}(\mathit{Reach}(\sigma_s, t')\ \mathit{after}\ (\sigma''_s -_\text{t} t'', t - t'), \\ &\qquad \Pi_\Sigma(\text{obs}(\sigma''_s -_\text{t} t'', t - t'))^{-1} \cdot (\texttt{buf}_\text{c} \cdot a))\end{aligned}$$

Now, since $t' \geq \text{time}(\sigma_s)$,

$$
\begin{aligned}
&\mathit{Reach}(\sigma_s, t') \mathit{\ after\ } (\sigma_s'' -_{\mathrm{t}} t'', t - t') \\
&\qquad = \mathit{Reach}(\sigma_s) \mathit{\ after\ } (\epsilon, t' - \mathrm{time}(\sigma_s)) \mathit{\ after\ } \\
&\qquad\quad (\sigma_s'' -_{\mathrm{t}} t'', t - t') \\
&\qquad = \mathit{Reach}(\sigma_s) \mathit{\ after\ } (\epsilon, t'') \mathit{\ after\ } (\sigma_s'' -_{\mathrm{t}} t'', t - t') \\
&\qquad = \mathit{Reach}(\sigma_s) \mathit{\ after\ } ((\sigma_s'' -_{\mathrm{t}} t'') +_{\mathrm{t}} t'', t - t' + t'') \\
&\qquad = \mathit{Reach}(\sigma_s) \mathit{\ after\ } (\sigma_s'', t - \mathrm{time}(\sigma_s)) \\
&\qquad = \mathit{Reach}(\sigma_s \cdot \sigma_s'', t)
\end{aligned}
$$

and

$$
\begin{aligned}
&\Pi_\Sigma(\mathrm{obs}(\sigma_s'' -_{\mathrm{t}} t'', t - t'))^{-1} \cdot (\mathtt{buf}_{\mathrm{c}} \cdot a) \\
&\qquad = \Pi_\Sigma(\mathrm{obs}(\sigma_s'', t - \mathrm{time}(\sigma_s)) -_{\mathrm{t}} t'')^{-1} \cdot (\mathtt{buf}_{\mathrm{c}} \cdot a) \\
&\qquad = \Pi_\Sigma(\mathrm{obs}(\sigma_s'', t - \mathrm{time}(\sigma_s)))^{-1} \cdot (\mathtt{buf}_{\mathrm{c}} \cdot a)
\end{aligned}
$$

Moreover, since

$$
\begin{aligned}
&\Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t)) \cdot \sigma_d \\
&\qquad = \Pi_\Sigma(\mathrm{obs}(\sigma_{t0}, t)^{-1} \cdot \sigma_{t0}) \cdot \sigma_d \\
&\qquad = (\Pi_\Sigma(\mathrm{obs}(\sigma_{t0}, t))^{-1} \cdot \Pi_\Sigma(\sigma_{t0})) \cdot \sigma_d \\
&\qquad = \Pi_\Sigma(\mathrm{obs}(\sigma_s \cdot \sigma_s'', t))^{-1} \cdot (\Pi_\Sigma(\sigma_{t0}) \cdot \sigma_d) \\
&\qquad = \Pi_\Sigma(\sigma_s \cdot \mathrm{obs}(\sigma_s'', t - \mathrm{time}(\sigma_s)))^{-1} \cdot (\Pi_\Sigma(\sigma_{t0}) \cdot \sigma_d) \\
&\qquad = (\Pi_\Sigma(\sigma_s) \cdot \Pi_\Sigma(\mathrm{obs}(\sigma_s'', t - \mathrm{time}(\sigma_s))))^{-1} \cdot \\
&\qquad\quad (\Pi_\Sigma(\sigma_s \cdot \sigma_s'') \cdot \sigma_d) \\
&\qquad = \Pi_\Sigma(\mathrm{obs}(\sigma_s'', t - \mathrm{time}(\sigma_s)))^{-1} \cdot \\
&\qquad\quad (\Pi_\Sigma(\sigma_s)^{-1} \cdot (\Pi_\Sigma(\sigma_s) \cdot \Pi_\Sigma(\sigma_s'') \cdot \sigma_d)) \\
&\qquad = \Pi_\Sigma(\mathrm{obs}(\sigma_s'', t - \mathrm{time}(\sigma_s)))^{-1} \cdot (\Pi_\Sigma(\sigma_s'') \cdot \sigma_d) \\
&\qquad = \Pi_\Sigma(\mathrm{obs}(\sigma_s'', t - \mathrm{time}(\sigma_s)))^{-1} \cdot \\
&\qquad\quad (\Pi_\Sigma(\sigma_s'') \cdot (\Pi_\Sigma(\sigma_s'')^{-1} \cdot (\mathtt{buf}_{\mathrm{c}} \cdot a))) \\
&\qquad = \Pi_\Sigma(\mathrm{obs}(\sigma_s'', t - \mathrm{time}(\sigma_s)))^{-1} \cdot (\mathtt{buf}_{\mathrm{c}} \cdot a)
\end{aligned}
$$

it follows that $\Pi_\Sigma(\mathrm{obs}(\sigma_s'' -_{\mathrm{t}} t'', t - t'))^{-1} \cdot (\mathtt{buf}_{\mathrm{c}} \cdot a) = \Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t)) \cdot \sigma_d$, and thus (2) becomes:

$$
\begin{aligned}
&\mathrm{nobs}(\sigma_{t0}, t) -_{\mathrm{t}} (t - \mathrm{time}(\mathrm{obs}(\sigma_{t0}, t))) \in \\
&\quad \mathrm{Safe}(\mathit{Reach}(\sigma_{t0}, t), \Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t)) \cdot \sigma_d)
\end{aligned}
$$

Thus, in both cases,

$$
\begin{aligned}
&\mathrm{nobs}(\sigma_{t0}, t) -_{\mathrm{t}} (t - \mathrm{time}(\mathrm{obs}(\sigma_{t0}, t))) \in \\
&\quad \mathrm{Safe}(\mathit{Reach}(\sigma_{t0}, t), \Pi_\Sigma(\mathrm{nobs}(\sigma_{t0}, t)) \cdot \sigma_d).
\end{aligned}
$$

Since this holds for any $t \in \mathbb{R}_{\geq 0}$, in particular, if $t = \mathrm{time}(\sigma_{t0})$, this means that $\epsilon \in \mathrm{Safe}(\mathit{Reach}(\sigma_{t0}), \sigma_d)$, meaning that $\mathit{Reach}(\sigma_{t0})$ *after* $\epsilon = \mathit{Reach}(\sigma_{t0}) \in F_G$. This means that $\sigma_{t0} \models \varphi$.
Thus, if $\sigma \cdot (\delta, a) \in \mathrm{Pre}(\varphi, t) \wedge \sigma \notin \mathrm{Pre}(\varphi, \mathrm{time}(\sigma \cdot (\delta, a)))$, $\mathrm{P}(\sigma) \implies \mathrm{P}(\sigma \cdot (\delta, a), t)$.

- Otherwise, $\sigma \cdot (\delta, a) \in \mathrm{Pre}(\varphi, t)$ and $\sigma \in \mathrm{Pre}(\varphi, \mathrm{time}(\sigma \cdot (\delta, a)))$. Then, by induction hypothesis:

$$
\begin{aligned}
&\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))) -_{\mathrm{t}} \\
&\quad (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\mathrm{obs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))))) \in \\
&\mathrm{Safe}(\mathit{Reach}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))), \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c)
\end{aligned}
$$

  - If $a \in \Sigma_{\mathrm{u}}$, since $\mathrm{Safe}(\mathit{Reach}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))), \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c) \neq \emptyset$, following the definition of Safe, it means that there exists $\sigma' \in \mathrm{tw}(\Sigma_{\mathrm{c}})$ such that the three following properties hold:
    1. $\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c$,
    2. $\mathit{Reach}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))$ *after* $\sigma' \in F \times \mathbb{R}_{\geq 0}$,

3. for any $t' \in \mathbb{R}_{\geq 0}$, if $v \in V_s$ is such that $\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))) \in v$, then $\langle v, \text{maxbuffer}(\Pi_\Sigma(\text{obs}(\sigma', t'))^{-1} \cdot \mathtt{buf}_c), 1\rangle \in W_0$.

In particular, for item 3, with $t' = 0$, we get $\langle v, \text{maxbuffer}(\mathtt{buf}_c), 1\rangle \in W_0$, with $\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))) \in v$. Thus, following the edge $(\langle v, \text{maxbuffer}(\mathtt{buf}_c), 1\rangle, \langle v \ \mathit{after}\ a, \text{maxbuffer}(\mathtt{buf}_c), 0\rangle) \in E_3$, since W_0 is the winning region for player 0, it follows that $\langle v \ \mathit{after}\ a, \text{maxbuffer}(\mathtt{buf}_c), 0\rangle \in W_0$.
Thus, there exists a winning strategy for player 0 from node $\langle v, \text{maxbuffer}(\mathtt{buf}_c), 0\rangle$, meaning that there exists a play π such that the set of nodes visited infinitely often by π, noted $\inf(\pi)$, is such that $\inf(\pi) \cap F_G \times \Sigma_c^n \times \{0, 1\} \neq \emptyset$, and $\pi(1) = \langle v, \text{maxbuffer}(\mathtt{buf}_c), 0\rangle$. Moreover, we can choose π such that no edge from E_3 or E_4 (corresponding to receiving uncontrollable or controllable events, respectively) is taken when playing π. This is possible since W_0 is the winning region for player 0, thus it is winning for all the strategies of player 1, and the edges of E_3 and E_4 leave a node belonging to player 1. Now, since the only cycles in the graph without the edges of E_3 and E_4 are cycles of the form $\langle v, w, 0\rangle\langle v, w, 1\rangle\langle v, w, 0\rangle$, with $(\langle v, w, 0\rangle, \langle v, w, 1\rangle) \in E1$ and $(\langle v, w, 1\rangle, \langle v, w, 0\rangle) \in E6$, it follows that π ends with such a cycle repeated indefinitely, i.e. $\pi = \pi_0 \cdot (\langle v_e, w_e, 0\rangle \cdot \langle v_e, w_e, 1\rangle)^\omega$ for some finite π_0. Thus, $\inf(\pi) = \{\langle v_e, w_e, 0\rangle, \langle v_e, w_e, 1\rangle\}$, meaning that $v_e \subseteq F_G$.
This allows us to associate a word σ' to π. To build it, we first build a sequence in $Q \times \mathbb{R}_{\geq 0} \times \text{tw}(\Sigma_c)$ by induction as follows:

$$(q_0, \delta_0, w_0) = (\mathit{Reach}(\sigma_s, \text{time}(\sigma \cdot (\delta, a)))\ \mathit{after}\ (0, a), 0, \epsilon)$$

and, for $i \in \mathbb{N}$,

$$(q_{i+1}, \delta_{i+1}, w_{i+1}) =$$
$$\begin{cases} (q_i, \delta_i, w_i) & \text{if } (\pi(i), \pi(i+1)) \in E_1 \cup E_6 \\ (q_i \ \mathit{after}\ (\delta_i, c), 0, w_i \cdot (\delta_i, c)) & \text{if } (\pi(i), \pi(i+1)) \in E_2, \text{ with } \pi(i) = \langle v, c \cdot w, 0\rangle \text{ for some } (c, w) \in \Sigma_c \times \Sigma_c^* \\ (q_i, \delta_i + \delta, w_i) & \text{if } (\pi(i), \pi(i+1)) \in E_5, \text{ with } \delta = \min(\{\delta' \in \mathbb{R}_{\geq 0} \mid q_i \ \mathit{after}\ (\epsilon, \delta_i + \delta') \in \Pi_1(\pi(i+1))\}) \end{cases}$$

Now since $\pi = \pi_0 \cdot (\langle v_e, w_e, 0\rangle \cdot \langle v_e, w_e, 1\rangle)^\omega$, there exists $n \in \mathbb{N}$ such that for any $n' \geq n$, $(\pi(n'), \pi(n'+1)) \in E_1 \cup E_6$, meaning that $(q_{n'}, \delta_{n'}, w_{n'}) = (q_n, \delta_n, w_n)$. Thus, the sequence stabilises. Let us consider $\sigma' = w_n$, where w_n is the third component of the previous sequence when it is stabilised. Then, σ' satisfies:

1. $\Pi_\Sigma(\sigma') \preccurlyeq \text{maxbuffer}(\mathtt{buf}_c)$, because there is no edge $(\pi(i), \pi(i+1))$ belonging to E_3 or E_4, and $\Pi_2(\pi(1)) = \text{maxbuffer}(\mathtt{buf}_c)$.
2. $\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))\ \mathit{after}\ (0, a)\ \mathit{after}\ \sigma' \in F_G$, because it belongs to $v_e \subseteq F_G$ (v_e is such that $\pi = \pi_0 \cdot (\langle v_e, w_e, 0\rangle \cdot v_e, w_e, 1\rangle)^\omega$).
3. For any $t' \in \mathbb{R}_{\geq 0}$, if $v \in V_s$ is such that $\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))\ \mathit{after}\ (0, a)\ \mathit{after}\ (\sigma', t') \in v$, then $\langle v, \Pi_\Sigma(\text{nobs}(\sigma', t'))^{-1} \cdot \mathtt{buf}_c, 1\rangle \in W_0$, because π is winning for player 0. By construction of σ', and because of the different constraints required on $\mathcal{G}_s$, this implies that all states $v \in V_s$ such that $\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))\ \mathit{after}\ (0, a)\ \mathit{after}\ (\sigma', t') \in v$ are in W_0, for any $t' \in \mathbb{R}_{\geq 0}$. We know by construction of σ' that this holds for some t', when an edge belonging to E_5 can be followed. The constraint item (6) required on V_s (see Definition 5) ensures that this is thus true for all t'.

Thus, $\sigma' \in \text{Safe}(\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))\ \mathit{after}\ (0, a), \mathtt{buf}_c)$, so $\text{Safe}(\mathit{Reach}(\sigma_s \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s), a)), \mathtt{buf}_c) \neq \emptyset$. Thus, $0 \in \text{T}(\mathit{Reach}(\sigma_s \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s), a)), \mathtt{buf}_c)$, meaning that $\sigma'_s \in \text{Safe}(\mathit{Reach}(\sigma_s \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s), a)), \mathtt{buf}_c)$. Let us consider $t' = \text{time}(\sigma \cdot (\delta, a))$.
Now, following lemma 2, since $t \geq t'$, $t - t' \geq 0$, then:
$\text{nobs}(\sigma'_s, t - t') -_t (t - t' - \text{time}(\text{obs}(\sigma'_s, t - t'))) \in$

$$\text{Safe}(\mathit{Reach}(\sigma_s \cdot (t' - \text{time}(\sigma_s), a))\ \mathit{after}\ (\sigma'_s, t - t'), \Pi_\Sigma(\text{obs}(\sigma'_s, t - t'))^{-1} \cdot \mathtt{buf}_c)$$

Since $t \geq t' = \text{time}(\sigma \cdot (\delta, a))$,
$$\begin{aligned}\text{nobs}(\sigma_{t0}, t) &= \text{nobs}(\sigma_s \cdot (t' - \text{time}(\sigma_s), a) \cdot \sigma'_s, t) \\ &= \text{nobs}(\sigma'_s, t - \text{time}(\sigma_s \cdot (t' - \text{time}(\sigma_s), a))) \\ &= \text{nobs}(\sigma'_s, t - \text{time}(\sigma \cdot (\delta, a))) \\ &= \text{nobs}(\sigma'_s, t - t')\end{aligned}$$
and $\text{obs}(\sigma_{t0}, t) = \sigma_s \cdot (t' - \text{time}(\sigma_s), a) \cdot (\text{obs}(\sigma'_s, t - t'))$. Thus, $\text{time}(\text{obs}(\sigma_{t0}, t)) = \text{time}(\sigma_s \cdot (t' - \text{time}(\sigma_s), a)) + \text{time}(\text{obs}(\sigma'_s, t - t')) = t' + \text{time}(\text{obs}(\sigma'_s, t - t'))$.
This means that:
$$\begin{aligned}&\text{nobs}(\sigma'_s, t - t') -_{\text{t}} (t - t' - \text{time}(\text{obs}(\sigma'_s, t - t'))) \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_{\text{t}} (t - t' - (\text{time}(\text{obs}(\sigma_{t0}, t) - t'))) \\ &\quad = \text{nobs}(\sigma_{t0}, t) -_{\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t)))\end{aligned}$$
Thus,
$$\begin{aligned}&\text{nobs}(\sigma_{t0}, t) -_{\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t))) \in \\ &\quad \text{Safe}(\mathit{Reach}(\sigma_s \cdot (t' - \text{time}(\sigma_s), a)) \mathit{\ after\ } (\sigma'_s, t - t'), \\ &\qquad \Pi_\Sigma(\text{obs}(\sigma'_s, t - t'))^{-1} \cdot \texttt{buf}_\text{c})\end{aligned}$$
Since
$$\begin{aligned}&\mathit{Reach}(\sigma_s \cdot (t' - \text{time}(\sigma_s), a)) \mathit{\ after\ } (\sigma'_s, t - t') \\ &\quad = \mathit{Reach}(\sigma_s \cdot (t' - \text{time}(\sigma_s, a)) \cdot \sigma'_s, t - t' + \\ &\qquad \text{time}(\sigma_s \cdot (t' - \text{time}(\sigma_s), a))) \\ &\quad = \mathit{Reach}(\sigma_{t0}, t - t' + \text{time}(\sigma \cdot (\delta, a))) \\ &\quad = \mathit{Reach}(\sigma_{t0}, t)\end{aligned}$$
and
$$\begin{aligned}&\Pi_\Sigma(\text{nobs}(\sigma_{t0}, t)) \cdot \sigma_d \\ &\quad = \Pi_\Sigma(\text{obs}(\sigma_{t0}, t)^{-1} \cdot \sigma_{t0}) \cdot \sigma_d \\ &\quad = (\Pi_\Sigma(\text{obs}(\sigma_{t0}, t))^{-1} \cdot \Pi_\Sigma(\sigma_{t0})) \cdot \sigma_d \\ &\quad = \Pi_\Sigma(\text{obs}(\sigma_s \cdot (t' - \text{time}(\sigma_s), a) \cdot \sigma'_s, t))^{-1} \cdot \\ &\qquad (\Pi_\Sigma(\sigma_s \cdot (t' - \text{time}(\sigma_s), a) \cdot \sigma'_s) \cdot \sigma_d) \\ &\quad = (\Pi_\Sigma(\sigma_s \cdot (t' - \text{time}(\sigma_s), a)) \cdot \Pi_\Sigma(\text{obs}(\sigma'_s, t - t')))^{-1} \cdot \\ &\qquad (\Pi_\Sigma(\sigma_s \cdot (t' - \text{time}(\sigma_s), a)) \cdot \Pi_\Sigma(\sigma'_s) \cdot \sigma_d) \\ &\quad = \Pi_\Sigma(\text{obs}(\sigma'_s, t - t'))^{-1} \cdot (\Pi_\Sigma(\sigma_s \cdot (t' - \text{time}(\sigma_s), a))^{-1} \cdot \\ &\qquad (\Pi_\Sigma(\sigma_s \cdot (t' - \text{time}(\sigma_s), a)) \cdot \Pi_\Sigma(\sigma'_s) \cdot \sigma_d)) \\ &\quad = \Pi_\Sigma(\text{obs}(\sigma'_s, t - t'))^{-1} \cdot (\Pi_\Sigma(\sigma'_s) \cdot \sigma_d) \\ &\quad = \Pi_\Sigma(\text{obs}(\sigma'_s, t - t'))^{-1} \cdot (\Pi_\Sigma(\sigma'_s) \cdot (\Pi_\Sigma(\sigma'_s)^{-1} \cdot \texttt{buf}_\text{c})) \\ &\quad = \Pi_\Sigma(\text{obs}(\sigma'_s, t - t'))^{-1} \cdot \texttt{buf}_\text{c}\end{aligned}$$
it follows that:
$$\begin{aligned}&\text{nobs}(\sigma_{t0}, t) -_{\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t))) \in \\ &\quad \text{Safe}(\mathit{Reach}(\sigma_{t0}, t), \Pi_\Sigma(\text{nobs}(\sigma_{t0}, t)) \cdot \sigma_d)\end{aligned}$$

– Otherwise, $a \in \Sigma_\text{c}$, and then, since $\text{Safe}(\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))), \Pi_\Sigma(\text{nobs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c) \neq \emptyset$, there exists $\sigma' \in \text{tw}(\Sigma)$ that satisfies the three following constraints:

1. $\Pi_\Sigma(\sigma') \preccurlyeq \Pi_\Sigma(\text{nobs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c$,
2. $\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))) \mathit{\ after\ } \sigma' \in F \times \mathbb{R}_{\geq 0}$,
3. for any $t' \in \mathbb{R}_{\geq 0}$, if $v \in V_\text{s}$ is such that $\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))) \in v$, then $\langle v, \text{maxbuffer}(\Pi_\Sigma(\text{obs}(\sigma', t'))^{-1} \cdot \texttt{buf}_\text{c}), 1\rangle \in W_0$.

Thus, item 1 can be written as $\Pi_\Sigma(\sigma') \preccurlyeq \texttt{buf}_\text{c} \cdot a$, and from item 3 we can deduce that for any $t' \in \mathbb{R}_{\geq 0}$,

if $v \in V_s$ is such that $\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))) \in v$, then $\langle v, \text{maxbuffer}(\Pi_\Sigma(\text{obs}(\sigma', t'))^{-1} \cdot (\mathtt{buf}_c \cdot a)), 1\rangle \in W_0$. This last property holds because adding a controllable event to the buffer only gives more possibilities to the EM (in the game graph, if $\langle v, w, p\rangle$ is winning, then $\langle v, w \cdot c, p\rangle$ is also winning). This means that $\sigma' \in \text{Safe}(\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))), \mathtt{buf}_c \cdot a)$, and thus, $\text{Safe}(\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))), \mathtt{buf}_c \cdot a) \neq \emptyset$.
Thus, $0 \in \text{T}(\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))), \mathtt{buf}_c \cdot a)$, meaning that $\sigma''_s -_t (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s)) \in \text{Safe}(\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))), \mathtt{buf}_c \cdot a)$. Let us consider $t' = \text{time}(\sigma \cdot (\delta, a))$, and $t'' = t' - \text{time}(\sigma_s)$. Then, following lemma 2,
$\text{nobs}(\sigma''_s -_t t'', t - t') -_t (t - t' - \text{time}(\text{obs}(\sigma''_s -_t t'', t - t'))) \in$

$\quad \text{Safe}(\mathit{Reach}(\sigma_{s0}, t')\ \mathit{after}\ (\sigma''_s -_t t'', t - t'),$

$\quad\quad \Pi_\Sigma(\text{nobs}(\sigma''_s -_t t'', t - t'))^{-1} \cdot (\mathtt{buf}_c \cdot a))$

Now, if $\mathtt{delay}(\sigma''_s(1)) > t - \text{time}(\sigma_s)$ (i.e. $\mathtt{delay}((\sigma''_s -_t t'')(1)) > t - t'$), then
$\text{nobs}(\sigma''_s -_t t'', t - t') = \text{nobs}(\sigma''_s, t - t' + t'') -_t t''$
$= \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s)) -_t t''$
Since $\text{nobs}(\sigma_{t0}, t) = \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s))$, it follows that $\text{nobs}(\sigma''_s -_t t'', t - t') = \text{nobs}(\sigma_{t0}, t) -_t t''$.
Moreover, $\text{obs}(\sigma''_s -_t t'', t - t') = \epsilon$ since $\mathtt{delay}(\sigma''_s(1)) > t - \text{time}(\sigma_s)$, thus, considering that $\text{obs}(\sigma_{t0}, t) = \sigma_s$,
$\text{nobs}(\sigma''_s -_t t'', t - t') -_t (t - t' - \text{time}(\text{obs}(\sigma''_s, t - t')))$
$= \text{nobs}(\sigma_{t0}, t) -_t (t - t' + t'')$
$= \text{nobs}(\sigma_{t0}, t) -_t (t - \text{time}(\sigma_s))$
$= \text{nobs}(\sigma_{t0}, t) -_t (t - \text{time}(\text{obs}(\sigma_{t0}, t)))$

On the other hand, if $\mathtt{delay}(\sigma''_s(1)) \leq t - \text{time}(\sigma_s)$, then $\text{nobs}(\sigma''_s -_t t'', t - t') = \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s)) = \text{nobs}(\sigma_{t0}, t)$, and since
$\text{time}(\text{obs}(\sigma_{t0}, t)) = \text{time}(\text{obs}(\sigma_s \cdot \sigma''_s, t))$
$= \text{time}(\sigma_s \cdot (\text{obs}(\sigma''_s, t - \text{time}(\sigma_s)), t))$
$= \text{time}(\sigma_s) + \text{time}(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s)))$
it follows that
$\text{nobs}(\sigma''_s -_t t'', t - t') -_t (t - t' - \text{time}(\text{obs}(\sigma''_s -_t t'', t - t')))$
$= \text{nobs}(\sigma''_s, t - \text{time}(\sigma_s)) -_t$
$\quad (t - t' - \text{time}(\text{obs}(\sigma''_s, t - t' + t'') -_t t''))$
$= \text{nobs}(\sigma_{t0}, t) -_t (t - t' - (\text{time}(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s))) - t''))$
$= \text{nobs}(\sigma_{t0}, t) -_t (t - t' + t'' - (\text{time}(\text{obs}(\sigma_{t0}, t)) - \text{time}(\sigma_s)))$
$= \text{nobs}(\sigma_{t0}, t) -_t (t - \text{time}(\sigma_s) + \text{time}(\sigma_s) - \text{time}(\text{obs}(\sigma_{t0}, t)))$
$= \text{nobs}(\sigma_{t0}, t) -_t (t - \text{time}(\text{obs}(\sigma_{t0}, t)))$

Thus, in both cases,
$\text{nobs}(\sigma_{t0}, t) -_t (t - \text{time}(\text{obs}(\sigma_{t0}, t))) \in$

$\quad \text{Safe}(\mathit{Reach}(\sigma_{s0}, t')\ \mathit{after}\ (\sigma''_s -_t t'', t - t'),$

$\quad\quad \Pi_\Sigma(\text{obs}(\sigma''_s -_t t'', t - t'))^{-1} \cdot (\mathtt{buf}_c \cdot a))$

Since
$\mathit{Reach}(\sigma_{s0}, t')\ \mathit{after}\ (\sigma''_s -_t t'', t - t')$
$= \mathit{Reach}(\sigma_s)\ \mathit{after}\ (\epsilon, t' - \text{time}(\sigma_s))\ \mathit{after}$
$\quad (\sigma''_s -_t t'', t - t')$
$= \mathit{Reach}(\sigma_s)\ \mathit{after}\ ((\sigma''_s -_t t'') +_t t'', t - t' + t'')$
$= \mathit{Reach}(\sigma_s)\ \mathit{after}\ (\sigma''_s, t - \text{time}(\sigma_s))$
$= \mathit{Reach}(\sigma_s \cdot \sigma''_s, t)$
$= \mathit{Reach}(\sigma_{t0}, t)$

and
$\Pi_\Sigma(\text{nobs}(\sigma_{t0}, t)) \cdot \sigma_d$
$= \Pi_\Sigma(\text{obs}(\sigma_{t0}, t)^{-1} \cdot \sigma_{t0}) \cdot \sigma_d$
$= (\Pi_\Sigma(\sigma_s) \cdot \Pi_\Sigma(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s))))^{-1} \cdot$
$(\Pi_\Sigma(\sigma_s) \cdot \Pi_\Sigma(\sigma''_s) \cdot \sigma_d)$
$= \Pi_\Sigma(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s)))^{-1} \cdot$
$(\Pi_\Sigma(\sigma_s)^{-1} \cdot (\Pi_\Sigma(\sigma_s) \cdot \Pi_\Sigma(\sigma''_s) \cdot \sigma_d))$
$= \Pi_\Sigma(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s)))^{-1} \cdot$
$(\Pi_\Sigma(\sigma''_s) \cdot \Pi_\Sigma(\sigma''_s)^{-1} \cdot (\texttt{buf}_\text{c} \cdot a))$
$= \Pi_\Sigma(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s)))^{-1} \cdot (\texttt{buf}_\text{c} \cdot a)$
considering that
$\Pi_\Sigma(\text{obs}(\sigma''_s \mathbin{-_\text{t}} t'', t - t'))^{-1} \cdot (\texttt{buf}_\text{c} \cdot a)$
$= \Pi_\Sigma(\text{obs}(\sigma''_s, t - t' + t'') \mathbin{-_\text{t}} t'')^{-1} \cdot (\texttt{buf}_\text{c} \cdot a)$
$= \Pi_\Sigma(\text{obs}(\sigma''_s, t - \text{time}(\sigma_s)))^{-1} \cdot (\texttt{buf}_\text{c} \cdot a)$
we finally obtain
$\text{nobs}(\sigma_{t0}, t) \mathbin{-_\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t))) \in$
$\text{Safe}(\mathit{Reach}(\sigma_{t0}, t), \Pi_\Sigma(\text{nobs}(\sigma_{t0}, t)) \cdot \sigma_d)$

Thus, in both cases,

$\text{nobs}(\sigma_{t0}, t) \mathbin{-_\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t))) \in$
$\text{Safe}(\mathit{Reach}(\sigma_{t0}, t), \Pi_\Sigma(\text{nobs}(\sigma_{t0}, t)) \cdot \sigma_d).$

In particular, this means that $\mathit{Reach}(\sigma_{t0}, t)$ *after* $\text{nobs}(\sigma_{t0}, t) \mathbin{-_\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t))) \in F_G$. Since
$\mathit{Reach}(\sigma_{t0}, t)$ *after* $\text{nobs}(\sigma_{t0}, t) \mathbin{-_\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t)))$
$= \mathit{Reach}(\text{obs}(\sigma_{t0}, t))$ *after* $(\epsilon, t - \text{time}(\text{obs}(\sigma_{t0}, t)))$ *after*
$\text{nobs}(\sigma_{t0}, t) \mathbin{-_\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t)))$
$= \mathit{Reach}(\text{obs}(\sigma_{t0}, t))$ *after* $(\text{nobs}(\sigma_{t0}, t) \mathbin{-_\text{t}}$
$(t - \text{time}(\text{obs}(\sigma_{t0}, t)))) \mathbin{+_\text{t}} (t - \text{time}(\text{obs}(\sigma_{t0}, t)))$
$= \mathit{Reach}(\text{obs}(\sigma_{t0}, t))$ *after* $\text{nobs}(\sigma_{t0}, t)$
$= \mathit{Reach}(\text{obs}(\sigma_{t0}, t) \cdot \text{nobs}(\sigma_{t0}, t))$
$= \mathit{Reach}(\sigma_{t0})$
this means that $\mathit{Reach}(\sigma_{t0}) \in F_G$, meaning that $\sigma_{t0} \models \varphi$.
Thus, if $\sigma \in \text{Pre}(\varphi, \text{time}(\sigma \cdot (\delta, a)))$, $\text{P}(\sigma) \implies \text{P}(\sigma \cdot (\delta, a), t)$.

Thus, in all cases, for any $t \in \mathbb{R}_{\geq 0}$, $\text{P}(\sigma) \implies \text{P}(\sigma \cdot (\delta, a), t)$. This means that $\text{P}(\sigma) \implies \text{P}(\sigma \cdot (\delta, a))$.

We then have shown by induction that $\text{P}(\sigma)$ holds for any $\sigma \in \text{tw}(\Sigma)$. In particular, we have shown that for any $(\sigma, t) \in \text{Pre}(\varphi)$, $\langle \sigma_s, \sigma_c \rangle = \text{store}_\varphi(\sigma) \implies \sigma_s \models \varphi$. Thus there exists t' that we can consider such that $t' \geq t$, that is such that for any $t'' \geq t'$, $\sigma_s = \text{E}_\varphi(\sigma, t'')$.
Thus, E_φ is sound in $\text{Pre}(\varphi)$. □

Proposition 3. E_φ is compliant, as per Definition 3.

Proof. We have to prove that the three following properties hold:

1. $\forall \sigma \in \text{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, \text{E}_\varphi(\sigma, t) \preccurlyeq_{\text{d}_{\Sigma_\text{c}}} \text{obs}(\sigma, t)$
2. $\forall \sigma \in \text{tw}(\Sigma), \forall t \in \mathbb{R}_{\geq 0}, \text{E}_\varphi(\sigma, t) =_{\Sigma_\text{u}} \text{obs}(\sigma, t)$
3. $\forall \sigma \in \text{tw}(\Sigma), \forall (\delta, u) \in \mathbb{R}_{\geq 0} \times \Sigma_\text{u},$
$\text{E}_\varphi(\sigma, \text{time}(\sigma \cdot (\delta, u))) \cdot (\text{time}(\sigma \cdot (\delta, u)) - \text{time}(\text{E}_\varphi(\sigma, \text{time}(\sigma \cdot (\delta, u)))), u) \preccurlyeq \text{E}_\varphi(\sigma \cdot (\delta, u), \text{time}(\sigma \cdot (\delta, u))).$

We start by proving items 1 and 2.

For $\sigma \in \mathrm{tw}(\Sigma)$, let $\mathrm{P}(\sigma)$ be the predicate "$\langle \sigma_{s0}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma) \implies (\sigma_{s0} \preccurlyeq_{\mathrm{d}\Sigma_\mathrm{c}} \sigma \wedge \sigma_{s0} =_{\Sigma_\mathrm{u}} \sigma \wedge \Pi_\Sigma(\sigma_{s0})_{|\Sigma_\mathrm{c}} \cdot \sigma_c = \Pi_\Sigma(\sigma)_{|\Sigma_\mathrm{c}})$". Let us prove by induction that $\mathrm{P}(\sigma)$ holds for any $\sigma \in \mathrm{tw}(\Sigma)$.

Induction basis: for $\sigma = \epsilon$, $\mathrm{store}_\varphi(\epsilon) = \langle \epsilon, \epsilon \rangle$, and since $\epsilon \preccurlyeq_{\mathrm{d}\Sigma_\mathrm{c}} \epsilon$, $\epsilon =_{\Sigma_\mathrm{u}} \epsilon$, and $\Pi_\Sigma(\epsilon)_{|\Sigma_\mathrm{c}} \cdot \epsilon = \Pi_\Sigma(\epsilon)_{|\Sigma_\mathrm{c}}$, it follows that $\mathrm{P}(\epsilon)$ holds.

Induction step: Suppose that for $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Let us consider $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $\langle \sigma_{s0}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma)$, $\langle \sigma_{t0}, \sigma_d \rangle = \mathrm{store}_\varphi(\sigma \cdot (\delta, a))$, and $\sigma_s = \mathrm{obs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))$.

- If $a \in \Sigma_\mathrm{u}$, then there exists σ'_s such that $\sigma_{t0} = \sigma_s \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a) \cdot \sigma'_s$, and $\Pi_\Sigma(\sigma'_s) \cdot \sigma_d = \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c$. Thus, since $a \in \Sigma_\mathrm{u}$
  $\Pi_\Sigma(\sigma_{t0})_{|\Sigma_\mathrm{c}} \cdot \sigma_d$
  $= \Pi_\Sigma(\sigma_s \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a) \cdot \sigma'_s)_{|\Sigma_\mathrm{c}} \cdot \sigma_d$
  $= \Pi_\Sigma(\sigma_s)_{|\Sigma_\mathrm{c}} \cdot \Pi_\Sigma(\sigma'_s)_{|\Sigma_\mathrm{c}} \cdot \sigma_d$
  Now, following the induction hypothesis, $\sigma_{s0} =_{\Sigma_\mathrm{u}} \sigma$, and since $\mathrm{nobs}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) = \epsilon$, it follows that $\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))) \in \mathrm{tw}(\Sigma_\mathrm{c})$, and thus $\sigma'_s \in \mathrm{tw}(\Sigma_\mathrm{c})$ too. Also following the induction hypothesis, we know that $\Pi_\Sigma(\sigma_{s0})_{|\Sigma_\mathrm{c}} \cdot \sigma_c = \Pi_\Sigma(\sigma)_{|\Sigma_\mathrm{c}}$. It follows that
  $\Pi_\Sigma(\sigma_{t0})_{|\Sigma_\mathrm{c}} \cdot \sigma_d = \Pi_\Sigma(\sigma_s)_{|\Sigma_\mathrm{c}} \cdot \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c$
  $= \Pi_\Sigma(\mathrm{obs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))) \cdot$
  $\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))))_{|\Sigma_\mathrm{c}} \cdot \sigma_c$
  $= \Pi_\Sigma(\sigma_{s0})_{|\Sigma_\mathrm{c}} \cdot \sigma_c$
  $= \Pi_\Sigma(\sigma)_{|\Sigma_\mathrm{c}}$
  $= \Pi_\Sigma(\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}}$
  Moreover, following the induction hypothesis, $\sigma_{s0} \preccurlyeq_{\mathrm{d}\Sigma_\mathrm{c}} \sigma$, thus in particular, $\sigma_s \preccurlyeq_{\mathrm{d}\Sigma_\mathrm{c}} \sigma$, meaning that if $i \in [1; |\sigma_s|]$, then $\mathrm{time}(\sigma_{s|\Sigma_\mathrm{c}[..i]}) \leq \mathrm{time}(\sigma_{|\Sigma_\mathrm{c}[..i]})$, and since $i \leq |\sigma_{|\Sigma_\mathrm{c}}|$, that means that $\mathrm{time}(\sigma_{s|\Sigma_\mathrm{c}[..i]}) \leq \mathrm{time}((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}[..i]})$. Since $\mathrm{time}(\sigma_s \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a)) = \mathrm{time}(\sigma \cdot (\delta, a))$, it follows that for any $i \in [|\sigma_s| + 1; |\sigma_{t0|\Sigma_\mathrm{c}}|]$, $\mathrm{time}(\sigma_{t0|\Sigma_\mathrm{c}[..i]}) \geq \mathrm{time}(\sigma \cdot (\delta, a))$ (remember that the restriction to an alphabet conserves dates, not delays). Thus, for any $i \in [1; |\sigma_{t0|\Sigma_\mathrm{c}}|]$, $\mathrm{time}(\sigma_{t0|\Sigma_\mathrm{c}[..i]}) \geq \mathrm{time}((\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}[..i]})$. Since we have already shown that $\Pi_\Sigma(\sigma_{t0})_{|\Sigma_\mathrm{c}} \cdot \sigma_d = \Pi_\Sigma(\sigma)_{|\Sigma_\mathrm{c}}$, we know that $\Pi_\Sigma(\sigma_{t0})_{|\Sigma_\mathrm{c}} \preccurlyeq \Pi_\Sigma(\sigma)_{|\Sigma_\mathrm{c}}$. This means that $\sigma_{t0} \preccurlyeq_{\mathrm{d}\Sigma_\mathrm{c}} \sigma \cdot (\delta, a)$.
  Finally, by induction hypothesis, $\sigma_{s0} =_{\Sigma_\mathrm{u}} \sigma$, thus, since
  $\sigma_{t0|\Sigma_\mathrm{u}} = (\sigma_s \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a))_{|\Sigma_\mathrm{u}}$
  $= \sigma_{s|\Sigma_\mathrm{u}} \cdot ((\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s), a) +_\mathrm{t}$
  $(\mathrm{time}(\sigma_s) - \mathrm{time}(\sigma_{s|\Sigma_\mathrm{u}})))$
  $= \sigma_{s|\Sigma_\mathrm{u}} \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s) +$
  $\mathrm{time}(\sigma_s) - \mathrm{time}(\sigma_{s|\Sigma_\mathrm{u}}), a)$
  $= \sigma_{s|\Sigma_\mathrm{u}} \cdot (\mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_{s|\Sigma_\mathrm{u}}), a)$
  $= \sigma_{|\Sigma_\mathrm{u}} \cdot ((\delta, a) +_\mathrm{t} (\mathrm{time}(\sigma) - \mathrm{time}(\sigma_{|\Sigma_\mathrm{u}})))$
  $= (\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{u}}$
  Thus, $\mathrm{P}(\sigma \cdot (\delta, a))$ holds.
- Otherwise, $a \in \Sigma_\mathrm{c}$, and then, there exists $\sigma''_s \in \mathrm{tw}(\Sigma)$ such that $\sigma_{t0} = \sigma_s \cdot \sigma''_s$ and $\Pi_\Sigma(\sigma''_s) \cdot \sigma_d = \Pi_\Sigma(\mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c \cdot a$.
  Thus,

$$\begin{aligned}\Pi_\Sigma(\sigma_{t0})_{|\Sigma_c} \cdot \sigma_d &= \Pi_\Sigma(\sigma_s)_{|\Sigma_c} \cdot \Pi_\Sigma(\sigma''_s) \cdot \sigma_d \\ &= \Pi_\Sigma(\sigma_s)_{|\Sigma_c} \cdot \Pi_\Sigma(\text{nobs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c \cdot a \\ &= \Pi_\Sigma(\sigma_{s0})_{|\Sigma_c} \cdot \sigma_c \cdot a \\ &= \Pi_\Sigma(\sigma)_{|\Sigma_c} \cdot a \\ &= \Pi_\Sigma(\sigma \cdot (\delta, a))_{|\Sigma_c}\end{aligned}$$

As in the case where $a \in \Sigma_u$, for any $i \in [1; |\sigma_{s|\Sigma_c}|]$, $\text{time}(\sigma_{t0|\Sigma_c[..i]}) \leq \text{time}(\sigma_{|\Sigma_c[..i]})$. Moreover, by construction, $\texttt{delay}(\sigma''_s(1)) \geq \text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s)$, thus for $i \in [|\sigma_{s|\Sigma_c}| + 1; \text{time}(\sigma_{t0|\Sigma_c})]$, if $i' = i - |\sigma_{s|\Sigma_c}|$, then $\text{time}(\sigma_{t0|\Sigma_c[..i]}) = \text{time}(\sigma_s) + \text{time}(\sigma''_{s[..i']}) \geq \text{time}(\sigma_s) + \text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s) = \text{time}(\sigma \cdot (\delta, a))$. Since $\text{time}(\sigma \cdot (\delta, a)) \leq \text{time}((\sigma \cdot (\delta, a))_{|\Sigma_c[..i]})$, and $\Pi_\Sigma(\sigma_{t0})_{|\Sigma_c} \preccurlyeq \Pi_\Sigma(\sigma \cdot (\delta, a))_{|\Sigma_c}$, this means that $\sigma_{t0} \preccurlyeq_{d\Sigma_c} \sigma \cdot (\delta, a)$.
Finally, $\sigma_{t0|\Sigma_u} = \sigma_{s|\Sigma_u} = \sigma_{|\Sigma_u} = (\sigma \cdot (\delta, a))_{|\Sigma_u}$.
Thus $\text{P}(\sigma \cdot (\delta, a))$ holds.

In both cases, $\text{P}(\sigma) \implies \text{P}(\sigma \cdot (\delta, a))$.

Thus, we have shown by induction that for all $\sigma \in \text{tw}(\Sigma)$, $\text{P}(\sigma)$ holds. Consequently, for any $\sigma \in \text{tw}(\Sigma)$, if $\langle \sigma_{s0}, \sigma_c \rangle = \text{store}_\varphi(\sigma)$, then $\sigma_{s0} \preccurlyeq_{d\Sigma_c} \sigma$ and $\sigma_{s0} =_{\Sigma_u} \sigma$. Thus, for any $t \in \mathbb{R}_{\geq 0}$, $\text{E}_\varphi(\sigma, t) = \text{obs}(\sigma_{s0}, t) \preccurlyeq_{d\Sigma_c} \text{obs}(\sigma, t)$, and $\text{E}_\varphi(\sigma, t) = \text{obs}(\sigma_{s0}, t) =_{\Sigma_u} \text{obs}(\sigma, t)$.
Thus, items 1 and 2 hold.

Now, let us prove item 3. Let us consider $\sigma \in \text{tw}(\Sigma)$, $\langle \sigma_{s0}, \sigma_c \rangle = \text{store}_\varphi(\sigma)$, $(\delta, u) \in \mathbb{R}_{\geq 0} \times \Sigma_u$, $\langle \sigma_{t0}, \sigma_d \rangle = \text{store}_\varphi(\sigma \cdot (\delta, u))$, and $\sigma_s = \text{obs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, u)))$. Then, $\sigma_s = \text{E}_\varphi(\sigma, \text{time}(\sigma \cdot (\delta, u)))$, and following the definition of store_φ (Definition 7), $\sigma_s \cdot (\text{time}(\sigma \cdot (\delta, u)) - \text{time}(\sigma_s), u) \preccurlyeq \sigma_{t0}$. Thus, $\text{E}_\varphi(\sigma \cdot (\delta, u), \text{time}(\sigma \cdot (\delta, u))) = \text{obs}(\sigma_{t0}, \text{time}(\sigma \cdot (\delta, u)))$. Since $\text{time}(\sigma_s \cdot (\text{time}(\sigma \cdot (\delta, u)) - \text{time}(\sigma_s), u)) = \text{time}(\sigma \cdot (\delta, u))$, it follows that $\sigma_s \cdot (\text{time}(\sigma \cdot (\delta, u)) - \text{time}(\sigma_s), u) \preccurlyeq \text{E}_\varphi(\sigma \cdot (\delta, u), \text{time}(\sigma \cdot (\delta, u)))$.

We have then shown that E_φ is compliant with respect to Σ_u and Σ_c. □

Proposition 4. E_φ is optimal in $\text{Pre}(\varphi)$ as per Definition 4.

Proof. Let us consider $\sigma \in \text{tw}(\Sigma)$, $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$ such that $(\sigma, \text{time}(\sigma \cdot (\delta, a))) \in \text{Pre}(\varphi)$, E an enforcement function that is compliant with respect to Σ_u and Σ_c, and such that $E(\sigma, \text{time}(\sigma \cdot (\delta, a))) = \text{E}_\varphi(\sigma, \text{time}(\sigma \cdot (\delta, a)))$. Let us suppose that $\text{E}_\varphi(\sigma \cdot (\delta, a), \infty) \prec_d E(\sigma \cdot (\delta, a), \infty)$. We then have to prove that there exists $\sigma_u \in \text{tw}(\Sigma_u)$ such that $E(\sigma \cdot (\delta, a) \cdot \sigma_u, \infty) \not\models \varphi$.

Let us consider $\sigma_s = \text{obs}(\text{E}_\varphi(\sigma, \text{time}(\sigma \cdot (\delta, a))), \text{time}(\sigma \cdot (\delta, a)))$. Then, since E_φ and E are compliant, and $\text{E}_\varphi(\sigma, \text{time}(\sigma \cdot (\delta, a))) = E(\sigma, \text{time}(\sigma \cdot (\delta, a)))$, there exists $\sigma'_s \in \text{tw}(\Sigma)$ such that $\text{E}_\varphi(\sigma \cdot (\delta, a), \infty) = \sigma_s \cdot \sigma'_s$ and $\sigma^E_s \in \text{tw}(\Sigma)$ such that $E(\sigma \cdot (\delta, a), \infty) = \sigma_s \cdot \sigma^E_s$. Now, since $\text{E}_\varphi(\sigma \cdot (\delta, a), \infty) \prec_d E(\sigma \cdot (\delta, a), \infty)$, this means that $\sigma'_s \prec_d \sigma^E_s$.

- If $a \in \Sigma_u$, since $(\sigma, \text{time}(\sigma \cdot (\delta, a))) \in \text{Pre}(\varphi)$, we know that (see proof of Proposition 2) $\sigma'_s \in \text{Safe}(\mathit{Reach}(\sigma_s), \Pi_\Sigma(\sigma_s)^{-1}_{|\Sigma_c} \cdot \Pi_\Sigma(\sigma)_{|\Sigma_c})$. Now, since $\sigma'_s \prec_d \sigma^E_s$, and since σ'_s is the maximal word for $\preccurlyeq_d$ that is in Safe, this means that $\sigma^E_s \notin \text{Safe}(\mathit{Reach}(\sigma_s), \Pi_\Sigma(\sigma_s)^{-1}_{|\Sigma_c} \cdot \Pi_\Sigma(\sigma)_{|\Sigma_c})$. This means that one of the following does not hold :
  1. $\Pi_\Sigma(\sigma^E_s) \preccurlyeq \Pi_\Sigma(\sigma_s)^{-1}_{|\Sigma_c} \cdot \Pi_\Sigma(\sigma)_{|\Sigma_c}$, but if this did not hold, then E would not be compliant.
  2. $\mathit{Reach}(\sigma_s)$ *after* $\sigma^E_s \notin F_G$. If this does not hold, then $\mathit{Reach}(\sigma_s \cdot \sigma^E_s) \notin F_G$, meaning that $E(\sigma \cdot (\delta, a), \infty) \not\models \varphi$.
  3. $\forall t \in \mathbb{R}_{\geq 0}, \forall v \in V_s, \mathit{Reach}(\sigma_s \cdot \sigma^E_s, t) \in v \implies \langle v, \text{maxbuffer}(\Pi_\Sigma(\sigma_s \cdot \text{obs}(\sigma^E_s, t))^{-1} \cdot \Pi_\Sigma(\sigma)_{|\Sigma_c}), 1 \rangle \in W_0$. If this does not hold, then there exists $t \in \mathbb{R}_{\geq 0}$ and $v \in V_s$ such that $\mathit{Reach}(\sigma_s \cdot \sigma^E_s, t) \in v$ and $\langle v, \text{maxbuffer}(\Pi_\Sigma(\sigma \cdot \text{obs}(\sigma^E_s, t))^{-1} \cdot \Pi_\Sigma(\sigma)_{|\Sigma_c}), 1 \rangle \notin W_0$. Then, there exists a winning strategy for player 1 from this node. This means that we can construct a word by following the winning strategy of player 1, like it is done in the proof of Proposition 2: depending on the edge followed in the game graph, player 1 can add an uncontrollable event to the input word (the delays are given by the edges corresponding to letting time elapse) that allows to stay in a node not belonging to W_0. This can be

done until the strategy of player 0 goes back to the previous node, making a loop if it has no time successor. This must ultimately happen since adding controllable events to the input only gives player 0 more possibilities, thus player 1 can choose only edges corresponding to adding uncontrollable events or letting time elapse. By privileging the elapse of time, it can ensure that the word will be finite. Thus, player 1 can build a word $\sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u})$ such that $E(\sigma \cdot (\delta, a) \cdot \sigma_u, \infty) \not\models \varphi$.

In any possible case, there exists $\sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u})$ such that $E(\sigma \cdot (\delta, a) \cdot \sigma_u, \infty) \not\models \varphi$ (in the second case, $\sigma_u = \epsilon$).

- Otherwise, $a \in \Sigma_\mathrm{c}$, and we can prove as in the previous case that there exists $\sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u})$ such that $E(\sigma \cdot (\delta, a) \cdot \sigma_u, \infty) \not\models \varphi$. All that is needed is to adapt the parameters of Safe: $\sigma'_s \in \mathrm{Safe}(\mathit{Reach}(\sigma_s, \mathrm{time}(\sigma \cdot (\delta, a))), \Pi_\Sigma(\sigma_s)^{-1}_{|\Sigma_\mathrm{c}} \cdot \Pi_\Sigma(\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}})$, and thus $\sigma_s^E \notin \mathrm{Safe}(\mathit{Reach}(\sigma_s, \mathrm{time}(\sigma \cdot (\delta, a))), \Pi_\Sigma(\sigma_s)^{-1}_{|\Sigma_\mathrm{c}} \cdot \Pi_\Sigma(\sigma \cdot (\delta, a))_{|\Sigma_\mathrm{c}})$, but the arguments are the same.

Thus, if E is compliant, and $\sigma \in \mathrm{tw}(\Sigma)$ and $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$ are such that $(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) \in \mathrm{Pre}(\varphi)$, $E(\sigma, \mathrm{time}(\sigma \cdot (\delta, a))) = \mathrm{E}_\varphi(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)))$, and $\mathrm{E}_\varphi(\sigma \cdot (\delta, a), \infty) \prec_\mathrm{d} E(\sigma \cdot (\delta, a), \infty)$, then there exists $\sigma_u \in \mathrm{tw}(\Sigma_\mathrm{u})$ such that $E(\sigma \cdot (\delta, a) \cdot \sigma_u, \infty) \not\models \varphi$.

This means that E_φ is optimal in $\mathrm{Pre}(\varphi)$. □

Proposition 5. The output o of $\mathcal{E}$ as per Definition 9 for input σ at date t is such that $o = \mathrm{E}_\varphi(\sigma, t)$.

Proof. In this proof, we use some notation from Section 3.3:

- $C^\mathcal{E} = \mathrm{tw}(\Sigma) \times \Sigma_\mathrm{c}^* \times Q \times \mathbb{R}_{\geq 0}$ is the set of configurations.
- $c_0^\mathcal{E} = \langle \epsilon, \epsilon, q_0, 0 \rangle \in C^\mathcal{E}$ is the initial configuration.
- $\Gamma^\mathcal{E} = ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\}) \times \mathit{Op} \times ((\mathbb{R}_{\geq 0} \times \Sigma) \cup \{\epsilon\})$ is the alphabet, composed of an optional input, an operation and an optional output.
  The set of operations is {dump(.), pass-uncont(.), store-cont(.), `delay`(.)}.

For a sequence of rules $w \in (\Gamma^\mathcal{E})^*$, we note the concatenation of all the inputs of w, $\mathrm{input}(w) = \Pi_1(w(1)) \cdot \Pi_1(w(2)) \dots \Pi_1(w(|w|))$, and $\mathrm{output}(w) = \Pi_3(w(1)) \cdot \Pi_3(w(2)) \dots \Pi_3(w(|w|))$ the concatenation of all the outputs of w. Since all configurations are not reachable from $c_0^\mathcal{E}$, for $w \in (\Gamma^\mathcal{E})^*$, we note $\mathrm{Reach}^\mathcal{E}(w) = c$ if $c_0^\mathcal{E} \xhookrightarrow{w}_\mathcal{E} c$ for some configuration $c \in C^\mathcal{E}$, or $\mathrm{Reach}^\mathcal{E}(w) = \bot$ if such a configuration does not exist. For a word $\sigma \in \mathrm{tw}(\Sigma)$, and a date $t \in \mathbb{R}_{\geq 0}$, we note $\mathrm{Rules}(\sigma, t) = \max_{\preccurlyeq}(\{w \in (\Gamma^\mathcal{E})^* \mid \mathrm{input}(w) = \mathrm{obs}(\sigma, t) \wedge \mathrm{Reach}^\mathcal{E}(w) \neq \bot \wedge \Pi_4(\mathrm{Reach}^\mathcal{E}(w)) = t - \mathrm{time}(\mathrm{output}(w))\})$. We also note $\mathrm{Reach}^\mathcal{E}(\sigma, t) = \mathrm{Reach}^\mathcal{E}(\mathrm{Rules}(\sigma, t))$. $\mathrm{Rules}(\sigma, t)$ represent the sequence that the EM applies with input word σ until date t. Since rule `delay`() can be applied an infinite number of times by slicing time, we only consider words in $(\Gamma^\mathcal{E})^*$ that are minimal in the number of rules `delay`(), i.e. the word obtained by merging two consecutive rules `delay`() into one with the sum of delays of the two rules, until stabilisation. This allows to define $\mathrm{Rules}(\sigma, t)$ correctly, without "cheating" by slicing time to increase the length of the word. Note that the words obtained by merging or adding `delay`() rules this way reach exactly the same configurations in the end. We will also allow ourselves to extend the use of output to timed words, such that $\mathrm{output}(\sigma, t) = \mathrm{output}(\mathrm{Rules}(\sigma, t))$.

We have to show that for any $\sigma \in \mathrm{tw}(\Sigma)$, and $t \in \mathbb{R}_{\geq 0}$, $\mathrm{output}(\sigma, t) = \mathrm{E}_\varphi(\sigma, t)$.

Now, for $\sigma \in \mathrm{tw}(\Sigma)$ and $t \in \mathbb{R}_{\geq 0}$, let $\mathrm{P}(\sigma, t)$ be the predicate "$\langle \sigma_{s0}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma) \implies (\mathrm{output}(\sigma, t) = \mathrm{obs}(\sigma_{s0}, t) \wedge \mathrm{Reach}^\mathcal{E}(\sigma, t) = \langle \mathrm{nobs}(\sigma_{s0}, t), \sigma_c, \mathit{Reach}(\sigma_{s0}, t), t - \mathrm{time}(\mathrm{obs}(\sigma_{s0}, t)) \rangle)$", and $\mathrm{P}(\sigma)$ be the predicate "$\forall t \in \mathbb{R}_{\geq 0}$, $\mathrm{P}(\sigma, t)$". Let us then show by induction that $\mathrm{P}(\sigma)$ holds for any $\sigma \in \mathrm{tw}(\Sigma)$.

Induction basis: if $\sigma = \epsilon$, then let us consider $t \in \mathbb{R}_{\geq 0}$. Then, $\mathrm{store}_\varphi(\epsilon) = \langle \epsilon, \epsilon \rangle$. On the other hand, the only rule that can be applied is `delay`(t), thus $\mathrm{Reach}^\mathcal{E}(\epsilon, t) = \langle \epsilon, \epsilon, q_0 \ \mathit{after}\ (\epsilon, t), t \rangle$.

Thus, $\mathrm{output}(\epsilon, t) = \mathrm{obs}(\epsilon, t)$, and $\mathrm{Reach}^\mathcal{E}(\epsilon, t) = \langle \mathrm{nobs}(\epsilon, t), \epsilon, \mathit{Reach}(\epsilon, t), t - \mathrm{time}(\epsilon, t) \rangle$. Thus, $\mathrm{P}(\epsilon, t)$ holds. Thus, for any $t \in \mathbb{R}_{\geq 0}$, $\mathrm{P}(\epsilon, t)$ holds, meaning that $\mathrm{P}(\epsilon)$ holds.

Induction step: let us suppose that for $\sigma \in \mathrm{tw}(\Sigma)$, $\mathrm{P}(\sigma)$ holds. Let us consider $(\delta, a) \in \mathbb{R}_{\geq 0} \times \Sigma$, $t \in \mathbb{R}_{\geq 0}$, $\langle \sigma_{s0}, \sigma_c \rangle = \mathrm{store}_\varphi(\sigma)$, $\langle \sigma_{t0}, \sigma_d \rangle = \mathrm{store}_\varphi(\sigma \cdot (\delta, a))$, $\sigma_s = \mathrm{obs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a)))$, and $c = \mathrm{Reach}^\mathcal{E}(\sigma, \mathrm{time}(\sigma \cdot (\delta, a)))$. Then, by induction hypothesis, $c = \langle \mathrm{nobs}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))), \sigma_c, \mathit{Reach}(\sigma_{s0}, \mathrm{time}(\sigma \cdot (\delta, a))), \mathrm{time}(\sigma \cdot (\delta, a)) - \mathrm{time}(\sigma_s) \rangle$.

If $t < \text{time}(\sigma \cdot (\delta, a))$, then $\text{Reach}^{\mathcal{E}}(\sigma \cdot (\delta, a), t) = \text{Reach}^{\mathcal{E}}(\sigma, t)$, and $\text{obs}(\sigma_{t0}, t) = \text{obs}(\sigma_{s0}, t)$, meaning that $\text{P}(\sigma \cdot (\delta, a), t)$ holds.

Then, let us consider that $t \geq \text{time}(\sigma \cdot (\delta, a))$.

- If $a \in \Sigma_{\text{u}}$, then rule pass-uncont(a) can be applied, meaning that c *after* $(a/\,\text{pass-uncont}(a)/a) = \langle \sigma'_b, \sigma'_c, q, 0\rangle$, with $q = \mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))$ *after* $(0, a) = \mathit{Reach}(\sigma_s \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s), a))$, $\sigma'_b = \kappa_\varphi(q, \Pi_\Sigma(\text{nobs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta,\, a)))) \cdot \sigma_c)$, and $\sigma'_c = \Pi_\Sigma(\sigma'_b)^{-1} \cdot (\Pi_\Sigma(\sigma_b) \cdot \sigma_c)$. Thus, σ'_b is such that $\sigma_{t0} = \sigma_s \cdot \sigma'_b$, thus c *after* $(a/\,\text{pass-uncont}(a)/a) = \langle \sigma_s^{-1} \cdot \sigma_{t0}, \sigma_d, \mathit{Reach}(\sigma_s \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s, a))), 0\rangle$. Then, rules `delay`() and dump() can be applied, until date t is reached, leading to the configuration $\langle \text{nobs}(\sigma_{t0}, t), \sigma_d, \mathit{Reach}(\sigma_{t0}, t), t - \text{time}(\text{obs}(\sigma_{t0}, t))\rangle$.
  Moreover, considering the transitions taken,
  $$\begin{aligned}\text{output}(\sigma \cdot (\delta, a), t) &= \text{output}(\sigma, \text{time}(\sigma \cdot (\delta, a))) \cdot (\text{time}(\sigma \cdot (\delta, a)) - \\ &\qquad \text{time}(\sigma_s), a) \cdot \text{obs}(\sigma'_b, t - \text{time}(\sigma \cdot (\delta, a))) \\ &= \sigma_s \cdot (\text{time}(\sigma \cdot (\delta, a)) - \text{time}(\sigma_s), a) \cdot \\ &\qquad \text{obs}(\sigma'_b, t - \text{time}(\sigma \cdot (\delta, a))) \\ &= \text{obs}(\sigma_{t0}, t)\end{aligned}$$
  Thus, $\text{P}(\sigma \cdot (\delta, a), t)$ holds.
- Otherwise, $a \in \Sigma_c$, and then rule store-cont(a) can be applied from configuration c, leading to c *after* $(a/\,\text{store-cont}(a)/\epsilon) = \langle \sigma'_b, \sigma'_c, \mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))), t - \text{time}(\sigma_s)\rangle$, with $\sigma'_b = \kappa_\varphi(\mathit{Reach}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a))), \Pi_\Sigma(\text{nobs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c \cdot a) +_{\text{t}} (t - \text{time}(\sigma_s))$ and $\sigma'_c = \Pi_\Sigma(\sigma'_b)^{-1} \cdot (\Pi_\Sigma(\text{nobs}(\sigma_{s0}, \text{time}(\sigma \cdot (\delta, a)))) \cdot \sigma_c \cdot a$. Thus, σ'_b is such that $\sigma_{t0} = \sigma_s \cdot \sigma'_b$, and $\sigma'_c = \sigma_d$. Then, rules `delay`() and dump() can be applied until date t is reached, leading to $\text{Reach}^{\mathcal{E}}(\sigma \cdot (\delta, a), t) = \langle \text{nobs}(\sigma_{t0}, t),\ \sigma_d,\ \mathit{Reach}(\sigma_{t0}, t), t - \text{time}(\text{obs}(\sigma_{t0}, t))\rangle$.
  Moreover, considering the transitions taken,
  $$\begin{aligned}\text{output}(\sigma \cdot (\delta, a), t) &= \text{output}(\sigma, \text{time}(\sigma \cdot (\delta, a))) \cdot \text{obs}(\sigma'_b, t - \text{time}(\sigma_s)) \\ &= \sigma_s \cdot \text{obs}(\sigma'_b, t - \text{time}(\sigma_s)) \\ &= \text{obs}(\sigma_s \cdot \sigma'_b, t) \\ &= \text{obs}(\sigma_{t0}, t)\end{aligned}$$
  Thus, $\text{P}(\sigma \cdot (\delta, a), t)$ holds.

Thus, in both cases, $\text{P}(\sigma \cdot (\delta, a), t)$ holds.

This means that for any $t \in \mathbb{R}_{\geq 0}$, $\text{P}(\sigma \cdot (\delta, a), t)$ holds.

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

We have then shown by induction that $\text{P}(\sigma)$ holds for any $\sigma \in \text{tw}(\Sigma)$. In particular, this means that for any $\sigma \in \text{tw}(\Sigma)$, if $\langle \sigma_{s0}, \sigma_c\rangle = \text{store}_\varphi(\sigma)$, then for any $t \in \mathbb{R}_{\geq 0}$, $\text{obs}(\sigma_{s0}, t) = \text{output}(\sigma, t)$. Thus, $\text{E}_\varphi(\sigma, t) = \text{obs}(\sigma_{s0}, t) = \text{output}(\sigma, t)$. □

References

[ACC+04] Baptiste Alcalde, Ana Cavalli, Dongluo Chen, Davy Khuu, and David Lee. Network protocol system passive testing for fault management: A backward checking approach. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 150–166. Springer, 2004.

[ACH+92] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, David Dill, and Howard Wong-Toi. Minimization of timed transition systems. In *CONCUR'92*, pages 340–354. Springer, 1992.

[AD92] Rajeev Alur and David Dill. The theory of timed automata. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer Berlin Heidelberg, 1992.

[BF18] Ezio Bartocci and Yliès Falcone, editors. *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*. Springer, 2018.

[BFFR18] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In Bartocci and Falcone [BF18], pages 1–33.

[BJKZ13] David Basin, Vincent Jugé, Felix Klaedtke, and Eugen Zălinescu. Enforceable security policies revisited. *ACM Trans. Inf. Syst. Secur.*, 16(1):3:1–3:26, June 2013.

[BKKW15] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: Runtime enforcement for reactive systems. *CoRR*, abs/1501.02573, 2015.

[BKZ11] David Basin, Felix Klaedtke, and Eugen Zalinescu. Algorithms for monitoring real-time properties. In Sarfraz Khurshid and Koushik Sen, editors, *Proceedings of the 2nd International Conference on Runtime Verification (RV 2011)*, volume 7186 of *Lecture Notes in Computer Science*, pages 260–275. Springer-Verlag, 2011.

[BY04] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. *Lecture Notes in Computer Science*, 3098:87–124, 2004.

[CEFJ15] Hadil Charafeddine, Khalil El-Harake, Yliès Falcone, and Mohamad Jaber. Runtime enforcement for component-based systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015*, pages 1789–1796, 2015.

[CGP03] Ana Cavalli, Caroline Gervy, and Svetlana Prokopenko. New approaches for passive testing using an extended finite state machine specification. *Information and Software Technology*, 45(12):837–852, 2003.

[CHP08] Krishnendu Chatterjee, Thomas A Henzinger, and Nir Piterman. Algorithms for büchi games. *arXiv preprint arXiv:0805.2620*, 2008.

[CMP92] Edward Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. *Automata, languages and programming*, pages 474–486, 1992.

[DLR15] Egor Dolzhenko, Jay Ligatti, and Srikar Reddy. Modeling runtime enforcement with mandatory results automata. *International Journal of Information Security*, 14(1):47–60, 2015.

[Fal10] Yliès Falcone. You should better enforce than verify. In Howard Barringer, Yliès Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon J. Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann, editors, *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volume 6418 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2010.

[FFM12] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349–382, 2012.

[FHR13] Yliès Falcone, Klaus Havelund, and Giles Reger. A tutorial on runtime verification. In Manfred Broy, Doron A. Peled, and Georg Kalus, editors, *Engineering Dependable Software Systems*, volume 34 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 141–175. IOS Press, 2013.

[FJMP16] Yliès Falcone, Thierry Jéron, Hervé Marchand, and Srinivas Pinisetty. Runtime enforcement of regular timed properties by suppressing and delaying events. *Systems & Control Letters*, 123:2–41, 2016.

[FMFR11] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[FMRS18] Yliès Falcone, Leonardo Mariani, Antoine Rollet, and Saikat Saha. Runtime failure prevention and reaction. In Bartocci and Falcone [BF18], pages 103–134.

[FP19] Yliès Falcone and Srinivas Pinisetty. On the runtime enforcement of timed properties. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification - 19th International Conference, RV 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11757 of *Lecture Notes in Computer Science*, pages 48–69. Springer, 2019.

[GT02] Erich Gradel and Wolfgang Thomas. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

[KT12] Raphaël Khoury and Nadia Tawbi. Which security policies are enforceable by runtime monitors? A survey. *Computer Science Review*, 6(1):27–45, 2012.

[LBW09] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12(3):19:1–19:41, January 2009.

[LS09] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[MP90] Zohar Manna and Amir Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 377–410. ACM, 1990.

[PFJ+13] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and OmerLandry Nguena Timo. Runtime enforcement of timed properties. In Shaz Qadeer and Serdar Tasiran, editors, *Runtime Verification*, volume 7687 of *Lecture Notes in Computer Science*, pages 229–244. Springer Berlin Heidelberg, 2013.

[PFJ+14] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Landry Nguena-Timo. Runtime enforcement of timed properties revisited. *Formal Methods in System Design*, 45(3):381–422, 2014.

[PFJM14a] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand. Runtime enforcement of parametric timed properties with practical applications. In *12th International Workshop on Discrete Event Systems, WODES 2014, Cachan, France, May 14-16, 2014.*, pages 420–427, 2014.

[PFJM14b] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand. Runtime enforcement of regular timed properties. In Yookun Cho, Sung Y. Shin, Sang-Wook Kim, Chih-Cheng Hung, and Jiman Hong, editors, *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 1279–1286. ACM, 2014.

[PFJM15] Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand. TiPEX: A Tool Chain for Timed Property Enforcement During eXecution. In Ezio Bartocci and Rupak Majumdar, editors, *RV'2015, 6th International Conference on Runtime Verification*, volume 9333 of *Lecture Notes in Computer Science*, page 12, Vienne, Austria, September 2015. Springer.

[RFR+17] Matthieu Renard, Yliès Falcone, Antoine Rollet, Thierry Jéron, and Hervé Marchand. Optimal enforcement of (timed) properties with uncontrollable events. *Mathematical Structures in Computer Science*, page 1–46, 2017.

[RRF17a] Matthieu Renard, Antoine Rollet, and Yliès Falcone. Grep: Games for the runtime enforcement of properties. In Nina Yevtushenko, Ana Rosa Cavalli, and Hüsnü Yenigün, editors, *Testing Software and Systems - ICTSS 2017*, pages 259–275. Springer International Publishing, 2017.

[RRF17b] Matthieu Renard, Antoine Rollet, and Yliès Falcone. Runtime enforcement using Büchi games. In *Proceedings of Model Checking Software - 24th International Symposium, SPIN 2017, Co-located with ISSTA 2017, Santa Barbara, USA*, pages 70–79. ACM Press, July 2017.

[Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, February 2000.

[UDB11] UDBM. Uppaal DBM Library. http://people.cs.aau.dk/~adavid/UDBM/, 2011. Accessed: 2017-04-27.

[WZW16] Meng Wu, Haibo Zeng, and Chao Wang. Synthesizing runtime enforcer of safety properties under burst error. In *8th NASA Formal Methods Symposium NFM16*, Minneapolis, USA, June 2016.