Complexity issues for timeline-based planning over dense time under future and minimal semantics

Laura Bozzelli [a], Angelo Montanari [b], Adriano Peron [a,*]

[a] *University of Napoli "Federico II", Napoli, Italy*
[b] *University of Udine, Udine, Italy*

A B S T R A C T

The problem of timeline-based planning (TP) over dense temporal domains is known to be undecidable in the general case. We first prove that the restriction to the future semantics does not suffice to recover decidability. Then, we introduce two semantic variants of TP, called *strong minimal and weak minimal* semantics, and show that they allow one to express meaningful properties. Both semantics are based on the minimality in the time distances of the existentially-quantified time events from the universally-quantified reference one, but the weak minimal variant distinguishes minimality in the past from minimality in the future. Surprisingly, we show that, despite the (apparently) small differences between the two semantics, the TP problem is still undecidable for the strong minimal one, while it is **PSPACE**-complete for the weak minimal one. Membership in **PSPACE** is determined by exploiting a strictly more expressive extension (ECA^+) of the well-known robust class of Event-Clock Automata (ECA), that allows us to encode the weak minimal TP problem and to reduce it to non-emptiness of Timed Automata (TA). Finally, an extension of ECA^+(ECA^{++}) is considered, proving that its non-emptiness problem is undecidable. We believe that the two extensions of ECA (ECA^+ and ECA^{++}), introduced for technical reasons, are actually valuable per sé in the field of TA.[1]

Keywords:
Timeline-based planning
Dense time
Complexity
Timed automata
Event-clock automata

1. Introduction

Timeline-based planning. Timelines provide an approach to planning alternative to the classic action-based one [3,4]. In the action-based approach of classical planning, the task of the planner is to find a sequence of actions that, applied from an initial state, allow an actor to achieve a given goal. Timeline-based planning (TP), instead, originates from the integration of planning and scheduling concepts in the context of space operations. Unlike action-based planning, timeline-based one does not explicitly distinguish among states, actions, and goals. It models the domain as a set of independent, but interacting, components, whose behavior over time (the timelines) is ruled by a set of temporal constraints, called synchronization rules. In such a framework, a solution plan is a set of timelines expressing a behavior of the system components that satisfies all the rules. Compared to classical action-based temporal planning, TP adopts a more declarative paradigm which focuses on the constraints that sequences of actions have to fulfil to reach a given goal. The declarative flavor allows

* Corresponding author.
E-mail addresses: lr.bozzelli@gmail.com (L. Bozzelli), angelo.montanari@uniud.it (A. Montanari), adrperon@unina.it (A. Peron).
[1] This paper is a revised and extended version of the conference papers [1] and [2], where, among the many improvements, all proofs have been fully detailed.

Table 1
TP problem with standard semantics.

	TP problem	Future TP problem
Unrestricted	Undecidable	Undecidable
Simple trigger rules	Undecidable	Decidable (non-primitive recursive)
Simple trigger rules, non-singular intervals	?	**EXPSPACE**-complete
Simple trigger rules, intervals in $Intv_{(0,\infty)}$	?	**PSPACE**-complete
Trigger-less rules	**NP**-complete	**NP**-complete

knowledge engineers to focus on what has or has not to happen, instead of on what the agent has to do to achieve a goal. Moreover, the modular structure makes it possible to separately model distinct system components. Over the years, TP has been successfully applied in many complex tasks, ranging from long- to short-term mission planning to on-board autonomy [5–10].

In TP, the planning domain is modeled as a set of independent, but interacting, components, each one modeled by a *state variable*. The temporal behavior of a single state variable (component) is described by a sequence of *tokens* (*timeline*), where each token specifies a value of the variable (state) and the period of time during which it takes that value. The overall temporal behavior (set of timelines) is constrained by a set of *synchronization rules* that specify quantitative temporal requirements between the time events (start-time and end-time) of distinct tokens. Synchronization rules have a very simple format: either *trigger rules*, expressing invariants and response properties (for each token in a given state, called *trigger*, there exist some other tokens satisfying some mutual temporal relations), or *trigger-less ones*, expressing goals (there exist some tokens satisfying some mutual temporal relations). Notice that the way in which requirements are specified by synchronization rules corresponds to the "freeze" mechanism in the well-known timed temporal logic TPTL [11], which uses the freeze quantifier to bind a variable to a specific temporal context (a token in the TP setting).

TP has been successfully exploited in a number of application domains, including space missions, constraint solving, and activity scheduling (see, e.g., [12–17]). A systematic study of expressiveness and complexity of TP has been undertaken only very recently in both the discrete-time and the dense-time settings [18–22].

In the discrete-time case, the TP problem turns out to be **EXPSPACE**-complete, and expressive enough to capture action-based temporal planning (see [21,22]).

In this paper we will consider TP over a dense temporal domain, without having recourse to any form of discretization, which is quite a common trick. A reason for assuming this different version of time domain is, basically, to increase expressiveness: in this way one can abstract from unnecessary (or even "forced") details, often artificially added due to the necessity of discretizing time, and can suitably represent actions with duration, accomplishments and temporally extended goals. However, despite the simple format of synchronization rules, the shift to a dense-time domain dramatically increases expressiveness and complexity, depicting a scenario which resembles that of the well-known timed linear temporal logics MTL and TPTL, under a point-wise semantics, which are undecidable in the general setting [11,23]. Known results about the TP problem over dense time are reported in Table 1. The problem in its full generality is undecidable [20], undecidability being caused by the high expressiveness of trigger rules (if only trigger-less rules are used, it is just **NP**-complete [24]). Decidability can be recovered by imposing suitable syntactic/semantic restrictions on the trigger rules. In particular, two significant restrictions have been considered [18,19]: (i) the first one limits the comparison to tokens whose start times follow the start time of the trigger (*future semantics of trigger rules*); (ii) the second one imposes that a non-trigger token can be referenced at most once in the time constraints of a trigger rule (*simple trigger rules*). By imposing the above two restrictions, the TP problem becomes decidable with a non-primitive recursive complexity [19] and can be solved by reducing it to model checking of Timed Automata (TA) [25] against MTL specifications over *finite* timed words, the latter being a known decidable problem [26]. It is worth pointing out that both restrictions effectively contribute to decidability. Indeed, the TP problem is still undecidable when restricted to simple trigger rules [20] and when the future semantics is assumed, but rules are not constrained to be simple. The latter result is illustrated in the next section; a preliminary account of it was given in [1]. As in the case of MTL [27], in the setting of simple trigger rules, better complexity results can be obtained if, in addition, we restrict the type of *intervals* used to compare tokens in simple trigger rules [18,19]. In particular, the problem is **EXPSPACE**-complete when only intervals with a non-null duration are considered (*non-singular intervals*) and **PSPACE**-complete for intervals which start at time 0 or are unbounded (the set of these intervals is denoted by $Intv_{(0,\infty)}$).

Paper contributions. The first contribution of the paper is the already-mentioned proof of undecidability of the TP problem under the future semantics. Its most relevant contributions are the introduction and systematic investigation of alternative semantics for the trigger rules in the dense-time setting, called *minimal semantics*. In the standard semantics of trigger rules, if there are many occurrences of non-trigger tokens carrying the same specified value, say v, nothing forces the choice of a specific occurrence for satisfying the given constraints. As an example, suppose that the trigger token represents a prompt for which a v-valued token is required in response. If many v-valued tokens occur in the timeline, the chosen one is not guaranteed to be the first token occurring after issuing the prompt. In a reactive context, one is usually interested in relating an issued prompt to the first response to it and not to an arbitrarily delayed one. In this paper, we define and study semantics requiring that the rule constraints are satisfied by the suitably-valued tokens occurring close to the trigger one.

A similar idea is exploited by Event-Clock Automata (ECA) [28], a well-known robust subclass of Timed Automata (TA) [25]. In ECA, each symbol a of the alphabet is associated with a *recorder*, or *past clock*, recording (at the current time) the time elapsed since the last occurrence of a, and a *predictor*, or *future clock*, measuring the time required for the next occurrence of a.

In more detail, the minimal semantics of trigger rules is based on the minimality in the time distances of the start times of existentially quantified tokens from the start time of the trigger token in a trigger rule. In fact, the minimality constraint can be used to express two alternative semantics: the *weak minimal semantics*, which distinguishes minimality in the past, with respect to the trigger token, from minimality in the future, and the *strong minimal semantics*, which considers minimality over all the start times (both in the past and in the future). Surprisingly, this apparently small difference in the definitions of weak and strong minimal semantics leads to a dramatic difference in the complexity-theoretic characterization of the TP problem: while the TP problem under the *strong minimal semantics* is still undecidable, the TP problem under the *weak minimal semantics* turns out to be **PSPACE**-complete (which is the complexity of the emptiness problem for TA and ECA [25,28]). **PSPACE** membership of the weak minimal TP problem is shown by a non-trivial exponential-time reduction to non-emptiness of TA. To handle the trigger rules under the weak minimal semantics, we exploit, as an intermediate step in the reduction, a strictly more expressive extension of ECA, called ECA$^+$. This novel extension of ECA is obtained by allowing a larger class of atomic event-clock constraints, namely, *diagonal constraints* between clocks of the *same polarity* (past or future) and *sum constraints* between clocks of *opposite polarity*. In [29], these atomic constraints are used in *event-zones* to obtain symbolic forward and backward analysis semi-algorithms for ECA, which are not guaranteed to terminate. We show that, in analogy to ECA, ECA$^+$ are closed under language Boolean operations and can be translated in exponential time into equivalent TA with an exponential number of control states, but a linear number of clocks. We also investigate an extension of ECA$^+$, called ECA^{++}, where the polarity requirements in the diagonal and sum constraints are relaxed, and we show that the nonemptiness problem for such a class of automata is undecidable.

To summarize, the proposed weak minimal semantics allows one to solve the TP problem in the dense-time setting with a reasonable computational complexity, without imposing any syntactic restriction to the format of synchronization rules. Moreover, it turns out to be still quite expressive and relevant for practical applications. As a by-product, two original extensions of ECA (ECA$^+$ and ECA^{++}) have been introduced to prove the main complexity results, which are interesting per se, as they shed new light on the landscape of event-clock and timed automata.

Outline. The paper is organized as follows. In Section 2, we recall the TP framework, we proof the undecidability of the TP problem under the future semantics and, then, we introduce the strong and weak minimal semantics. In Section 3, we prove that the TP problem under the strong minimal semantics is still undecidable. Next, in Section 4, we introduce ECA$^+$ and ECA^{++}and study their expressiveness and complexity. Finally, in Section 5, by exploiting the results for ECA$^+$, we prove **PSPACE**-completeness of the weak minimal TP problem. Conclusions provide an assessment of the work done and outline future research themes.

2. The timeline-based planning problem

In this section, we first recall the standard TP framework, as described in [9,21,18], and then we introduce the strong and weak minimal semantics.

2.1. The standard TP problem

In TP, the domain knowledge is encoded by a set of state variables, whose behavior over time is described by transition functions and constrained by synchronization rules. We will adopt the following notation. Let $\mathbb{N}$ be the set of natural numbers, $\mathbb{R}_+$ be the set of non-negative real numbers, and *Intv* be the set of intervals in $\mathbb{R}_+$ whose endpoints are in $\mathbb{N} \cup \{\infty\}$. Given a finite word w over some alphabet (or, equivalently, a finite sequence of symbols), $|w|$ denotes the length of w and for all $0 \le i < |w|$, $w(i)$ is the $(i+1)$-th letter of w.

Definition 1. A *state variable* x is a triple $x = (V_x, T_x, D_x)$, where V_x is the *finite domain* of the variable x, $T_x : V_x \to 2^{V_x}$ is the *value transition function*, which maps each $v \in V_x$ to the (possibly empty) set of successor values, and $D_x : V_x \to Intv$ is the *constraint function* that maps each $v \in V_x$ to an interval.

Example 1. As an example of state variable, we consider the modeling of the temporal behavior of an autonomous elevator which operates between two floors. The elevator can stop either at the first floor or the second floor. When the elevator arrives at a certain floor, its door automatically open. It takes at least 2 seconds from its arrival before the door opens but the door must definitely open within 5 seconds. Whenever the elevator's door is open, passengers can enter. The door can close only 4 seconds after the last passenger entered. After the door closes, the elevator travels up or down to the other floor. It takes at least 4 seconds for moving from a floor to the other floor.

The requested elevator behavior can be described by the state variable $x = (V_x, T_x, D_x)$, where the domain V_x consists of the states $open_1$, $open_2$, $close_1$, $close_2$, $enter_1$, $enter_2$, up, and $down$. For each $i = 1, 2$, the values $open_i$ and $close_i$ describes the door operations (opening and closure) at floor i, while $enter_i$ describes the entering of passengers at floor i. Moreover, the value up (resp., $down$) represents the movement of the elevator from the first to the second floor (resp., from the

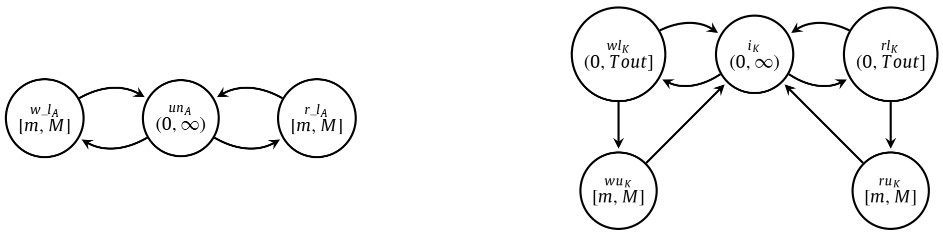
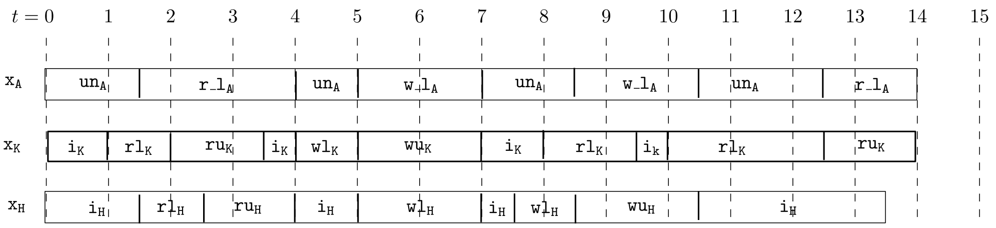


Fig. 1. State variables x_A and x_K.



t = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
x_A: un_A r_l_A un_A w_l_A un_A w_l_A un_A r_l_A
x_K: i_K rl_K ru_K i_K wl_K wu_K i_K rl_K i_K rl_K ru_K
x_H: i_H rl_H ru_H i_H wl_H i_H wl_H wu_H i_H

Fig. 2. A multi-timeline for the state variables x_A, x_K and x_H of Example 2.

second to the first floor). The value transition function T_x is deterministic and is defined as follows: (i) $T_x(open_i) = \{enter_i\}$ and $T_x(enter_i) = \{close_i\}$ for each $i = 1, 2$, (ii) $T_x(close_1) = \{up\}$ and $T_x(up) = \{open_2\}$, and (iii) $T_x(close_2) = \{down\}$ and $T_x(down) = \{open_1\}$.

Finally, the constraint function D_x is defined as: (i) $D_x(open_i) = [2, 5]$, $D_x(enter_i) = (0, \infty)$, and $D_x(close_i) = [4, \infty)$ for each $i = 1, 2$, and (ii) $D_x(up) = D_x(down) = [4, \infty)$.

A *token* for a variable x is a pair (v, d) consisting of a value $v \in V_x$ and a duration $d \in \mathbb{R}_+$ such that $d \in D_x(v)$. For a token $t = (v, d)$, *value*(t) denotes the first component v of t. Intuitively, a token for x represents an interval of time where the state variable x takes value v. The behavior of the state variable x is specified by means of *timelines*, which are non-empty sequences of tokens $\pi = (v_0, d_0) \ldots (v_n, d_n)$ consistent with the value transition function T_x, that is, such that $v_{i+1} \in T_x(v_i)$, for all $0 \leq i < n$. We associate with the i-th token $(0 \leq i \leq n)$ of the timeline π two punctual events: (i) the *start point*, whose timestamp (*start time*), denoted by $\mathsf{s}(\pi, i)$, is 0 if $i = 0$, and $\sum_{h=0}^{i-1} d_h$ otherwise (i.e., the sum of the durations of the tokens preceding the ith one along π), and (ii) the *end point* whose timestamp (*end time*), denoted by $\mathsf{e}(\pi, i)$, is $\mathsf{e}(\pi, i) := \mathsf{s}(\pi, i) + d_i$.

Given a finite set SV of state variables, a *multi-timeline* of SV is a mapping Π assigning to each state variable $x \in SV$ a timeline for x.

Example 2. Let us consider a set of transactions, e.g., database transactions, that access a common shared resource A for read/write operations. The resource A can be unlocked (un_A), read_locked (r_l_A), or write_locked (w_l_A). A state variable $x_A = (V_A, T_A, D_A)$, with $V_A = \{un_A, r\_l_A, w\_l_A\}$, is used to describe the availability/locking of the resource A over time. The value transition function T_A is represented as a graph in Fig. 1 (left). Each node is labeled by a value v and by the constraint $D_A(v)$. The constants m and M are the lower and upper bound, respectively, for the duration of read/write locking.

A state variable $x_K = (V_K, T_K, D_K)$, with K ranging over transaction names, describes the read/write locking requests issued by transaction K for the use of the resource A. A transaction can be idle (i_K), issuing a read or write lock for accessing the resource (rl_K or wl_K, respectively), or reading or writing the resource (ru_K or wu_K, respectively). We have $V_K = \{i_K, rl_K, wl_K, ru_K, wu_K\}$. An issued lock request can be accepted, thus allowing the use of the resource, or reejected. There is a timeout $Tout$ for waiting the availability of the resource. The value transition and constraint functions T_K and D_K are depicted in Fig. 1 (right). A multi-timeline for the state variables x_A (shared resource A), x_K and x_H (for two transactions K and H accessing A) is depicted in Fig. 2. Each rectangle of width d corresponds to a token of duration d. For instance, the timeline for x_A represented in Fig. 2 is the sequence of tokens $(un_A, 1.5), (r\_l_A, 2.5), (un_A, 1), (w\_l_A, 2), (un_A, 1.5), (w\_l_A, 2), (un_A, 2), (r\_l_A, 1.5)$.

Synchronization rules. Let SV be a finite set of state variables. Multi-timelines of SV can be constrained by a set of *synchronization rules*, which relate tokens, possibly belonging to different timelines, through temporal constraints on the start/end-times of tokens (point constraints) and on the difference between start/end-times of tokens (difference constraints). The synchronization rules exploit an alphabet Σ of token names to refer to the tokens along a multi-timeline, and are based on the notions of *atom* and *existential statement*.

An *atom* is either a clause of the form $ev(o) \in I$ (*point atom*), or of the form $ev(o) - ev'(o') \in I$ (*difference atom*), where $o, o' \in \Sigma$, $I \in Intv$, and $ev, ev' \in \{\mathsf{s}, \mathsf{e}\}$. Intuitively, an atom $ev(o) \in I$ asserts that the ev-time (i.e., the start-time if $ev = \mathsf{s}$, and the end-time otherwise) of the token referenced by o is in the interval I, while an atom $ev(o) - ev'(o') \in I$ requires that the difference between the ev-time and the ev'-time of the tokens referenced by o and o', respectively, is in I. Formally, an atom is evaluated with respect to a *Σ-assignment λ_Π for a given multi-timeline Π of SV* which is a mapping assigning to each token name $o \in \Sigma$ a pair $\lambda_\Pi(o) = (\pi, i)$ such that π is a timeline of Π and $0 \leq i < |\pi|$ is a position along π (intuitively, (π, i) represents the token of Π referenced by the name o). An atom $ev(o) \in I$ (resp., $ev(o) - ev'(o') \in I$) *is satisfied by* λ_Π if $ev(\lambda_\Pi(o)) \in I$ (resp., $ev(\lambda_\Pi(o)) - ev'(\lambda_\Pi(o')) \in I$).

An *existential statement* $\mathcal{E}$ (for SV) is a statement of the form

$$\mathcal{E} := \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n].\mathcal{C}$$

where $\mathcal{C}$ is a conjunction of atoms, $o_i \in \Sigma$, $x_i \in SV$, and $v_i \in V_{x_i}$ for each $i = 1, \ldots, n$. The elements $o_i[x_i = v_i]$ are called *quantifiers*. A token name used in $\mathcal{C}$, but not occurring in any quantifier, is said to be *free*. Intuitively, the quantifier $o_i[x_i = v_i]$ binds the name o_i to some token in the timeline for variable x_i having value v_i. A Σ-assignment λ_Π for a multi-timeline Π of SV *satisfies* $\mathcal{E}$ if each atom in $\mathcal{C}$ is satisfied by λ_Π, and for each quantified token name o_i, $\lambda_\Pi(o_i) = (\pi, h)$ where $\pi = \Pi(x_i)$ and the h-th token of π has value v_i. A multi-timeline Π of SV *satisfies* $\mathcal{E}$ if there exists a Σ-assignment λ_Π for Π which satisfies $\mathcal{E}$.

Definition 2. A *synchronization rule* $\mathcal{R}$ for the set SV of state variables has one of the forms

$$\text{(trigger rule) } o_0[x_0 = v_0] \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k, \quad \text{(trigger-less rule) } \top \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k,$$

where $o_0 \in \Sigma$, $x_0 \in SV$, $v_0 \in V_{x_0}$, and $\mathcal{E}_1, \ldots, \mathcal{E}_k$ are *existential statements*. In trigger rules, the quantifier $o_0[x_0 = v_0]$ is called *trigger*, and it is required that only o_0 may appear free in $\mathcal{E}_i$ (for $i = 1, \ldots, k$). For trigger-less rules, it is required that no token name appears free.

Intuitively, a trigger $o_0[x_0 = v_0]$ acts as a universal quantifier, which states that *for all* the tokens of the timeline for the state variable x_0 with value v_0, at least one of the existential statements $\mathcal{E}_i$ must be true. Trigger-less rules simply assert the satisfaction of some existential statement. Formally, the *standard semantics* of the synchronization rules is defined as follows. A multi-timeline Π of SV *satisfies* a *trigger-less rule* $\mathcal{R}$ of SV if Π satisfies some existential statement of $\mathcal{R}$. Π *satisfies* a *trigger rule* $\mathcal{R}$ of SV with trigger $o_0[x_0 = v_0]$ if for every position i of the timeline $\Pi(x_0)$ for x_0 such that $\Pi(x_0)(i) = (v_0, d)$, there is an existential statement $\mathcal{E}$ of $\mathcal{R}$ and a Σ-assignment λ_Π for Π such that $\lambda_\Pi(o_0) = (\Pi(x_0), i)$ and λ_Π satisfies $\mathcal{E}$.

Trigger-less are usually exploited to express initial conditions or the goals of the problem, while trigger rules are useful to specify invariants and response requirements.

Example 3. With reference to Example 2, we introduce a set of synchronization rules to guarantee that the shared resource A is accessed in mutual exclusion during writing by transactions K and H. We preliminarily define some shorthand (conjunctions of atoms) to express interval relations between tokens associated with token names o and $\overline{o}$:

- $during(o, \overline{o}) := \mathsf{s}(\overline{o}) - \mathsf{s}(o) \in [0, \infty) \ \wedge \ \mathsf{e}(o) - \mathsf{e}(\overline{o}) \in [0, \infty)$ requires that the token referenced by $\overline{o}$ occurs during the token referenced by o;
- $overlap(o, \overline{o}) := \mathsf{e}(o) - \mathsf{s}(\overline{o}) \in (0, \infty) \ \wedge \ \bigwedge_{ev \in \{\mathsf{s}, \mathsf{e}\}} ev(\overline{o}) - ev(o) \in (0, \infty)$ states that token $\overline{o}$ does not start before token o and crosses the end point of token o.

The pair of trigger-less rules below state the initial conditions: the resource A is initially unlocked and transactions K and H are idle. The first trigger rule ensures that when transaction K reads resource A, the resource is locked for reading (the same can be required for H). The second trigger rule requires that when transaction K writes A (wu_K), there is a write locking token (w_l_A) of A with the same temporal window as token wu_K.

- $\top \to \exists o[x_A = un_A].\, \mathsf{s}(o) \in [0, 0]$ and $\top \to \exists o[x_K = i_K].\, \mathsf{s}(o) \in [0, 0]$;
- $o_0[x_K = ru_K] \to \exists o[x_A = r\_l_A].during(o, o_0)$;
- $o_0[x_K = wu_K] \to \exists o[x_A = w\_l_A].during(o, o_0) \wedge during(o_0, o)$.

The two trigger rules above ensure also the mutual exclusion among reads and writes of the resource A by the same transaction K. The next rule is added to guarantee mutual exclusion when both transactions K and H write A, that is, it ensures that K and H do not feature tokens of value wu_K and wu_H, respectively, with the same temporal window.

$$o_0[x_K = wu_K] \to \bigvee_{s \in \{i_H, wl_H, rl_H\}} \big(\exists o[x_H = s].during(o, o_0) \vee \exists o[x_H = s].during(o_0, o) \vee \\ \exists o[x_H = s].overlap(o_0, o) \ \vee \ \exists o[x_H = s].overlap(o, o_0)\big).$$

Notice that the multi-timeline of Fig. 2 satisfies all the above rules.

Domains and plans. A TP domain $\mathcal{D} = (SV, R)$ is specified by a finite set SV of state variables and a finite set R of synchronization rules modeling their admissible behaviors. A *plan* of $\mathcal{D}$ is a multi-timeline of SV satisfying all the rules in R. The *TP problem* consists of checking, given a domain $\mathcal{D}$, whether there is a plan of $\mathcal{D}$.

We also consider the *discrete-time versions* of the previous problems, where the durations of the tokens in a plan are restricted to be natural numbers.

2.2. The TP problem with future semantics

As already mentioned, the TP problem is undecidable in the general setting. The first negative result presented in the paper is that the problem remain undecidable under a stronger notion of satisfaction of trigger rules, called *satisfaction under the future semantics*. The future semantics requires that all the non-trigger selected tokens do not start *strictly before* the start-time of the trigger token.

Definition 3. A multi-timeline Π of SV *satisfies* a trigger rule $\mathcal{R} = o_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$ *under the future semantics* if Π satisfies the trigger rule obtained from $\mathcal{R}$ by replacing each existential statement $\mathcal{E}_i = \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n].\mathcal{C}$ with $\exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n].\mathcal{C} \wedge \bigwedge_{i=1}^{n} \mathsf{s}(o_i) - \mathsf{s}(o_o) \in [0, +\infty)$. A *future plan* of $\mathcal{D}$ is a multi-timeline of SV satisfying all the rules in R under the future semantics. The *future TP problem* consists of checking, given a domain $\mathcal{D}$, whether there is a future plan of $\mathcal{D}$.

Notice that the multi-timeline of Fig. 2 is a future plan for the domain described in Example 2. The following result negatively answers a question left open in [18].

Theorem 1. *Future TP problem with* one state variable *is undecidable even if the intervals are in* $Intv_{(0,\infty)}$.[2]

Theorem 1 is proved by a polynomial-time reduction from the *halting problem for Minsky* 2*-counter machines* [30]. For the sake of completeness, we recall the definition of halting problem for Minsky 2-counter machines (the notion will be useful in the proof of Theorem 2 as well).

A nondeterministic Minsky 2-counter machine is a tuple $M = (Q, q_{init}, q_{halt}, \Delta)$, where Q is a finite set of (control) locations, $q_{init} \in Q$ is the initial location, $q_{halt} \in Q$ is the halting location, and $\Delta \subseteq Q \times L \times Q$ is a transition relation over the instruction set $L = \{\mathsf{inc}, \mathsf{dec}, \mathsf{zero\_test}\} \times \{1, 2\}$. For a transition $\delta = (q, op, q') \in \Delta$, we define $from(\delta) := q$, $op(\delta) := op$, and $to(\delta) := q'$. Without loss of generality, we assume that:

- for each transition $\delta \in \Delta$, $from(\delta) \neq q_{halt}$ and $to(\delta) \neq q_{init}$, and
- there is exactly one transition $\delta \in \Delta$, denoted δ_{init}, such that $from(\delta) = q_{init}$.

An M-configuration is a pair (q, ν) consisting of a location $q \in Q$ and a counter valuation $\nu : \{1, 2\} \rightarrow \mathbb{N}$. A computation of M is a non-empty *finite* sequence $(q_1, \nu_1), \ldots, (q_k, \nu_k)$ of configurations such that for all $1 \leq i < k$, there is some instruction $op_i = (tag_i, c_i) \in L$ so that $(q_i, op_i, q_{i+1}) \in \Delta$ and: (i) $\nu_{i+1}(c) = \nu_i(c)$ if $c \neq c_i$; (ii) $\nu_{i+1}(c_i) = \nu_i(c_i) + 1$ if $tag_i = \mathsf{inc}$; (iii) $\nu_{i+1}(c_i) = \nu_i(c_i) - 1$ and $\nu_i(c_i) > 0$ if $tag_i = \mathsf{dec}$; and (iv) $\nu_{i+1}(c_i) = \nu_i(c_i) = 0$ if $tag_i = \mathsf{zero\_test}$. The halting problem consist of deciding whether, for a machine M, there is a computation starting at the *initial* configuration (q_{init}, ν_{init}), where $\nu_{init}(1) = \nu_{init}(2) = 0$, and leading to some halting configuration (q_{halt}, ν) (it was proved to be undecidable in [30]).

Proposition 1. *Given a Minsky* 2*-counter M, one can construct (in polynomial time) a TP instance (domain) $P = (\{x_M\}, R_M)$, where the intervals in P are in $Intv_{(0,\infty)}$ such that M halts* if and only if *there exists a future plan for P.*

The proof of above proposition is given in [1], and it is reported in Appendix A for the sake of self containment.

2.3. The TP problem with minimal semantics

In this subsection, we define the variant of the semantics for trigger rules newly proposed and investigated in this paper. In the standard semantics of trigger rules, if there are many occurrences of non-trigger tokens carrying the same specified value, nothing forces the choice of a specific occurrence for satisfying the required constraints. With reference to Example 2, consider the trigger rule $o_0[x_K = rl_K] \rightarrow \exists o[x_K = ru\_K].\mathsf{s}(o) - \mathsf{e}(o_0) \in [0, \infty)$ requiring that each read lock request of transaction K is ultimately satisfied. The timeline π_K for x_K depicted in Fig. 2 fulfils the trigger rule both with the Σ-assignment $\lambda_\Pi(o_0) = (\pi_K, 2)$, $\lambda_\Pi(o) = (\pi_K, 3)$ and the assignment $\lambda'_\Pi(o_0) = (\pi_K, 2)$, $\lambda'_\Pi(o) = (\pi_K, 11)$, namely the witness of the satisfaction of first read lock request is either the first read lock or the last read lock. Actually, the more natural choice would be the following Σ-assignments providing the closest ru_K token to the rl_K trigger occurrence: $\lambda_\Pi(o_0) = (\pi_K, 2)$ and $\lambda_\Pi(o) = (\pi_K, 3)$; $\lambda_\Pi(o_0) = (\pi_K, 8)$ and $\lambda_\Pi(o) = (\pi_K, 11)$; $\lambda_\Pi(o_0) = (\pi_K, 10)$ and $\lambda_\Pi(o) = (\pi_K, 11)$.

Following this idea, the minimal semantics is obtained from the standard semantics by additionally requiring that the given Σ-assignment λ_Π selects for each (existential) quantifier $o[x = v]$ a token for variable x with value v whose start point has a minimal time distance from the start point of the trigger.

[2] $Intv_{(0,\infty)}$ denotes the set of intervals $I \in Intv$ such that either I is unbounded or I is left-closed with left endpoint 0.

Actually, the constraint of minimality can be used to express two alternative semantics: the *weak minimal semantics* which distinguishes minimality in the past (w.r.t. the trigger token) from the minimality in the future, and the *strong minimal semantics* which considers minimality over all the start times (both in the past and in the future) of the tokens for a variable x and x-value v.

Definition 4 *(Weak and strong minimal semantics of trigger rules).* Let $o_0 \in \Sigma$. A Σ-assignment λ_Π for a multi-timeline Π of SV is *weakly minimal* w.r.t. o_0 if for each $o \in \Sigma$ with $\lambda_\Pi(o) = (\pi, i)$, the following holds:

- *minimality in the past:* if $\mathsf{s}(\pi, i) \leq \mathsf{s}(\lambda_\Pi(o_0))$, then there is no position ℓ along the timeline π such that $value(\pi(i)) = value(\pi(\ell))$ and $\mathsf{s}(\pi, i) < \mathsf{s}(\pi, \ell) \leq \mathsf{s}(\lambda_\Pi(o_0))$;
- *minimality in the future:* if $\mathsf{s}(\pi, i) \geq \mathsf{s}(\lambda_\Pi(o_0))$, then there is no position ℓ along the timeline π such that $value(\pi(i)) = value(\pi(\ell))$ and $\mathsf{s}(\pi, i) > \mathsf{s}(\pi, \ell) \geq \mathsf{s}(\lambda_\Pi(o_0))$.

A Σ-assignment λ_Π for Π is *strongly minimal* w.r.t. o_0 if for each $o \in \Sigma$ with $\lambda_\Pi(o) = (\pi, i)$, there is no position ℓ along the timeline π such that $value(\pi(i)) = value(\pi(\ell))$ and $|\mathsf{s}(\pi, \ell) - \mathsf{s}(\lambda_\Pi(o_0))| < |\mathsf{s}(\pi, i) - \mathsf{s}(\lambda_\Pi(o_0))|$.

The weak minimal (resp., strong minimal) semantics of the trigger rules is obtained from the standard one by imposing that the considered Σ-assignment λ_Π is weakly minimal (resp., strongly minimal) w.r.t. the trigger token o_0.

Note that we consider start points of tokens for expressing minimality. Equivalent semantics can be obtained by considering end points of tokens instead. A *weak* (resp., *strong*) *minimal plan* of $\mathcal{D}$ is a multi-timeline of SV satisfying all the rules in R under the weak (resp., strong) minimal semantics of trigger rules. The *weak* (resp. *strong*) *minimal TP problem* is checking given a domain $\mathcal{D}$, whether there is a weak (resp. strong) minimal plan of $\mathcal{D}$. Obviously, any weak minimal or strong minimal plan is also a plan and any strong minimal plan is also a weak minimal plan. Hence, the strong minimal semantics is a refinement of the weak minimal one, which is in turn a refinement of the standard semantics.

Example 4. With reference to Example 3, one can easily check that any plan for the particular domain is a weak minimal plan. For instance, consider the synchronization rule $R = o_0[x_K = ru_K] \to \exists o[x_A = r\_l_A].during(o, o_0)$ and an assignment satisfying R in the standard semantics binding o and o_0 to tokens $\pi_A(i)$ and $\pi_{x_K}(j)$, respectively, for some $1 \leq i \leq |\pi_A|$ and $1 \leq j \leq |\pi_{x_K}|$. Since token $\pi_A(i)$ covers token $\pi_{x_K}(j)$, there cannot be another token $\pi_A(l)$, with $1 \leq l < i$, such that $\mathsf{s}(\pi_A(i)) < \mathsf{s}(\pi_A(l)) \leq \mathsf{s}(\pi_{x_K}(j))$, implying that the same assignment satisfies R also in the weak minimal semantics. Similar arguments can be used for all the other rules. Conversely, a plan is not necessarily a strongly minimal one. As an example, consider the timeline $\pi_{x_A} = (un_A, 1), (r\_l_A, 7), (un_A, 1), (r\_l_A, 7)$ for the state variable x_A and the timeline $\pi_{x_K} = (i_k, 1), (rl_K, 5), (ul_k, 2)$. The given multi-timeline satisfies the synchronization rule R both in the standard and in the weak minimal semantics binding the symbol o to $\pi_{x_A}(2)$. In the strong minimal semantics, the synchronization rule is not satisfied since the symbol o can only be bound to $\pi_{x_A}(4)$ $(|\mathsf{s}(\pi_{x_A}(4)) - \mathsf{s}(\pi_{x_K}(3))| < |\mathsf{s}(\pi_{x_A}(2)) - \mathsf{s}(\pi_{x_K}(3))|)$.

The idea underlying the introduction of the weak and strong minimal semantics of the trigger rules is inspired by research in formal verification and synthesis where in the last two decades many papers have focused on quantitative aspects, in particular boundedness requirements. We observe that the trigger rules represent a first-order formalism for expressing quantitative temporal liveness requirements such as the (future) non-punctual bounded-time request-response condition: "every request p is followed by a response q within k time units". It is well-known that quantitative verification problems which take into account unrestricted quantitative liveness properties are undecidable in the dense-time setting. For example, for the class of Timed Automata (TA) [25], the verification problem (model checking) against the fragment of MTL [27] with past expressing punctual bounded-time response properties is in general undecidable, and with a non-primitive recursive complexity if the past modalities are disallowed [26]. Therefore, in the literature, some subclasses of TA and fragments of timed temporal logics have been introduced to recover decidability and tractability, which are still interesting in practice since they can express non-punctual bounded-time response properties. In particular, the idea behind the minimal semantics of trigger rules is exploited in the class of Event-Clock Automata (ECA) [28], a well-known robust subclass of TA, and in the fragment of MTL represented by Event-Clock Temporal Logic (EC_TL) [31]. These frameworks allow to specify bounds on the time distance between the "trigger" event (the event occurring at the current time) and a "response" event in the future (resp., in the past), where the latter is uniquely determined by requiring that its distance from the trigger is minimal among all events in the future (resp., in the past) associated with a given observable atomic proposition. Note that differently from the format of trigger rules which is based on existential and universal quantification, ECA and EC_TL have explicit mechanisms to refer to the past and to the future. In particular, ECA distinguish between past clock and future clocks, while the logic EC_TL has past and future temporal modalities. In the setting of trigger rules, the most natural notion of minimality in the choice of the existential tokens leads to the strong minimal semantics. On the other hand, the weak minimal semantics seems more interesting from a practical point of view since allows to keep separated the choices made in the past from those made in the future (w.r.t. the starting time of the trigger token). As we will see in the next sections, the weak minimal semantics is also drastically preferable from a complexity-theoretic point of view. As clearly illustrated in the following example, there is a subtlety in the strong minimal semantics which is the

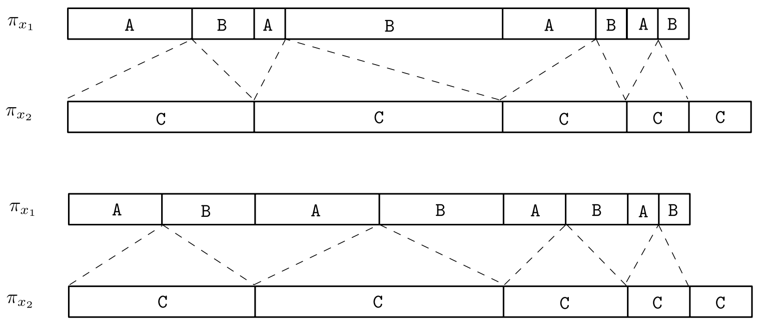


Fig. 3. Multi-timelines for Example 5.

basis of an high expressive power. While in the weak minimal semantics, the "minimal" token is uniquely determined, in the strong semantics there may be two minimal tokens: one in the past and one in the future. In this way, it is possible to enforce that two adjacent temporal intervals have the same duration, which is a key ingredient for encoding counting operations also in the discrete-time setting.

We provide the following example to stress the difference between the weak minimal and strong minimal semantics enlightening the expressive power of the strong minimal one.

Example 5. Let us consider two state variables: $x_1 = (\{A, B\}, T_1, D_1)$, with $T_1(A) = \{B\}$, $T_1(B) = \{A\}$, and $D_1(A) = D_1(B) = (0, \infty)$; $x_2 = (\{C\}, T_2, D_2)$, with $T_2(C) = \{C\}$ and $D_2(C) = (0, \infty)$. Before defining the synchronization rules, we introduce some shorthand (conjunctions of atoms) to express interval relations between tokens associated with token names o and $\overline{o}$:

- $start\_with(o, \overline{o}) := \mathsf{s}(\overline{o}) - \mathsf{s}(o) \in [0, 0] \;\wedge\; \mathsf{e}(\overline{o}) - \mathsf{e}(o) \in (0, \infty)$ requires that the token referenced by o is an initial subinterval of the token referenced by $\overline{o}$;
- symmetrically, $end\_by(o, \overline{o}) := \mathsf{e}(o) - \mathsf{e}(\overline{o}) \in [0, 0] \wedge \mathsf{s}(o) - \mathsf{s}(\overline{o}) \in (0, \infty)$ requires that the token referenced by o is a final subinterval of the token referenced by $\overline{o}$.

We require that: (i) timelines for x_1 start with A-valued tokens; (ii) each A-valued (resp, B-valued) token is an initial subinterval (resp, final subinterval) of a C-valued token; (iii) each B-valued token is preceded and followed by a C-valued token.

- $\top \rightarrow \exists o[x_1 = A].\, \mathsf{s}(o) \in [0, 0]$;
- $o_0[x_1 = A] \rightarrow \exists o[x_2 = C].start\_with(o_0, o)$;
- $o_0[x_1 = B] \rightarrow \exists o[x_2 = C].end\_by(o_0, o)$;
- $o_0[x_1 = B] \rightarrow \exists o[x_2 = C].\mathsf{s}(o_0) - \mathsf{s}(o) \in (0, \infty)$;
- $o_0[x_1 = B] \rightarrow \exists \overline{o}[x_2 = C].\mathsf{s}(\overline{o}) - \mathsf{s}(o_0) \in (0, \infty)$.

Fig. 3 shows two examples of multi-timelines for the considered domain. Both of them are plans in the weak minimal semantics, whereas only the second multi-timeline is a plan for the strong minimal one. As a matter of fact, a plan in the strong minimal semantics satisfies the quite expressive property that every pair of tokens (A, d_A), (B, d_B) of the timeline for x_1 included in the same token (C, d_C) of the timeline for x_2 must fulfill $d_A = d_B$ and then $d_C = 2d_A = 2d_B$. This implies that the strong minimal semantics allows one to enforce very expressive constraints on the duration of tokens (they will be exploited in the following to prove the undecidability of the TP problem under the strong minimal semantics).

Assumption 1 *(*Strict time monotonicity*)*. In the following, for simplifying the technical presentation of some results, without loss of generality, we assume that given a state variable $x = (V_x, T_x, D_x)$, the duration of a token for x is never zero, i.e., for each $v \in V_x$, $0 \notin D_x(v)$.

3. Undecidability of the strong minimal TP problem

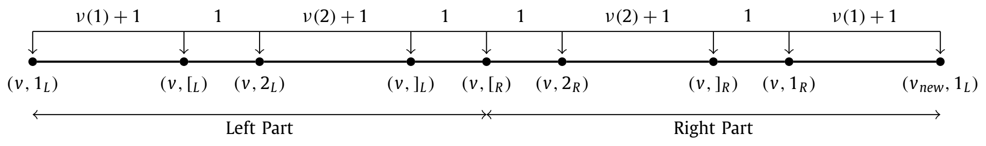
In this section, we prove the undecidability of the strong minimal TP problem by a polynomial-time reduction from the *halting problem for Minsky* 2*-counter machines* [30]. The key feature in the reduction is the ability of expressing for a given value v, a temporal equidistance requirement w.r.t. the start point of the trigger token for the start points of the last token before the trigger with value v and the first token after the trigger with value v (the same feature exemplified in Example 5). It is meaningful to observe that the reduction does not exploit the power of dense-time domain. Thus, the undecidability of the strong minimal TP can be stated also in a discrete time domain. Such a result is surprising since the TP problem in the standard semantics with discrete time domain is decidable.

Theorem 2. *The strong minimal TP problem is undecidable even in the discrete-time setting.*

Proof. The proof is by a polynomial-time reduction from the *halting problem for Minsky 2-counter machines* [30]. For a Minsky 2-counter machines $M = (Q, q_{init}, q_{halt}, \Delta)$ we construct a TP instance $\mathcal{D}_M = (SV_M, R_M)$ such that M halts *if and only if* there exists a strong minimal discrete-time plan for $\mathcal{D}_M$.

We exploit a state variable $x_M \in SV_M$ for encoding the evolution of the machine M and additional state variables for checking that the values of counters in the timeline for x_M are correctly updated. The domain V_M of the state variable x_M is $V_M := V_\Delta \times \{1_L, 1_R, 2_L, 2_R, [_L,]_L, [_R,]_R\}$ where V_Δ is the set of pairs $(\delta'_\perp, \delta)$, where $\delta'_\perp \in \Delta \cup \{\perp\}$, $\delta \in \Delta$, and $to(\delta'_\perp) = from(\delta)$ if $\delta'_\perp \neq \perp$, and $\delta = \delta_{init}$ otherwise. Intuitively, in the pair $(\delta'_\perp, \delta)$, δ is the transition currently taken by M from the current non-halting configuration C, while $\delta'_\perp$ is $\perp$ if C is the initial configuration, and δ' is the transition taken by M in the previous computational step, otherwise.

A configuration $C = (q, \nu)$ of M is encoded by the timelines π_C (*configuration codes*) of length 9 for the state variable x_M depicted in the figure above, where $v \in V_\Delta$ (called V_Δ-*value of* π_C) is of the form $(\delta'_\perp, \delta)$ such that $from(\delta) = q$. Note that the configuration code π_C consists of two parts.



In the left part (resp., right part), the encoding of counter 1 (resp., 2) precedes the encoding of counter 2 (resp., 1). The value $\nu(1)$ of counter 1 is encoded by the duration, which is $\nu(1)+1$, of the *counter token* with value marked by 1_L in the left part, and the *counter token* with value marked by 1_R in the right part, and similarly for counter 2. The four tokens with values marked by $[_L$, $]_L$, $[_R$, and $]_R$, respectively, are called *tagged* tokens and their duration is always 1: they are used to check by trigger rules (under the strong minimal semantics) that increment and decrement M-instructions are correctly encoded. Moreover, we require that the configuration code π_C satisfies the following additional requirement (V_Δ-*requirement*), with $v = (\delta'_\perp, \delta)$ and $\delta = (q, op, q')$:

- $v_{new} = v$ if $to(\delta) = q_{halt}$, and v_{new} is of the form (δ, δ'') otherwise (*consecution*);
- if $\delta = \delta_{init}$ then the counter tokens have duration 1;
- if $op = (\mathsf{dec}, c)$ (resp., $op = (\mathsf{zero\_test}, c)$), then the durations of the counter tokens with values (v, c_L) and (v, c_R) are greater than 1 (resp., are equal to 1).

A *pseudo-configuration code* is defined as a configuration code but allowing that the counter tokens (i.e., the tokens with values in $V_\Delta \times \{1_L, 1_R, 2_L, 2_R\}$) have arbitrary duration provided that the restriction that the V_Δ-requirement is fulfilled. In particular, in a pseudo-configuration code, the requirement that for each counter $c \in \{1, 2\}$, the duration of the counter token marked by c_L coincides with the duration of the counter token marked by c_R is relaxed.

A *pseudo-computation code* π_M is a sequence of the form $\pi_M = \pi_0 \cdots \pi_n$ such that

(i) $\pi_i \cdot \pi_{i+1}(0)$ and π_n are *pseudo-configuration codes* for all $0 \leq i < n$;
(ii) if $n > 0$ (resp., $n = 0$), the V_Δ-value of $\pi_0 \cdot \pi_1(0)$ (resp., π_0) is $(\perp, \delta_{init})$ (*initialization*);
(iii) the V_Δ-value of π_n is of the form $(\delta'_\perp, \delta)$ such that $to(\delta) = q_{halt}$ (*halting*).

By construction, we can easily define the transition function T_M and the constraint function of x_M in such a way that (i) the timelines for x_M whose first token has value $v_{init} = ((\perp, \delta_{init}), 1_L)$ correspond to the prefixes of pseudo-computation codes, and (ii) $v_{init} \notin T_M(v)$ for all $v \in V_M$. Hence, for capturing the timelines of x_M representing all and only the pseudo-computation codes it suffices to exploit a *trigger-less* rule $\mathcal{R}_{init,halt}$ requiring that a timeline of x_M visits a token with value $((\perp, \delta_{init}), 1_L)$ (initialization) and a token with value $(\delta'_\perp, \delta)$ such that $to(\delta) = q_{halt}$ (halting).

We now consider the crucial part of the reduction which has to guarantee that along a pseudo-computation code (i.e., a timeline of variable x_M satisfying the trigger-less rule $\mathcal{R}_{init,halt}$) the counters are correctly encoded (i.e., the durations of the left and right tokens for each counter in a pseudo-configuration code coincide) and are updated accordingly to the M-instructions. For each instruction $op \in \{\mathsf{inc}, \mathsf{dec}, \mathsf{zero\_test}\} \times \{1, 2\}$ of the machine M, let V_{op} be the subset of V_Δ consisting of the pairs $(\delta'_\perp, \delta)$ such that $\delta'_\perp \neq \perp$ and $op(\delta'_\perp) = op$. We exploit an additional state variable $x_=$ and, for each instruction op of M, the additional state variable x_{op}. Each of such variables x has domain $V_{check} := \{check_1, check_2, trigger, \perp\}$ and captures the timelines π_x such that the duration of each token is at least 1 and the untimed projection (first component of each token) of π_x is an arbitrary non-empty word over V_{check}. Variable $x_=$ is used in conjunction with trigger rules for enforcing that along a pseudo-computation code π_M the durations of the left and right tokens for counter 1 (resp., 2) coincide (*left-right requirement*), while for each instruction op, variable x_{op} is used in conjunction with trigger rules for ensuring that the durations of the counter tokens in the non-initial pseudo-configuration codes π_C of π_M whose V_Δ values are in V_{op}

are updated consistently with the instruction *op* (*op-requirement*). For this, we first require that the timelines associated to distinct state variables are *synchronized*, i.e., they have the same length and for each position i, the start-times of the ith tokens of the different timelines coincide. Since the duration of a token is not zero, the synchronization requirement can be easily expressed by simple trigger rules (under the strong minimal semantics). Then, for all variables $x \in \{x_=\} \cup \bigcup_{op}\{x_{op}\}$, values $v_{check} \in V_{check}$, and values $v \in V_M$, we have the two trigger rules

$$o[x = v_{check}] \rightarrow \bigvee_{u \in V_M} \exists o'[x_M = u].\, \mathsf{s}(o') - \mathsf{s}(o) \in [0,0]$$

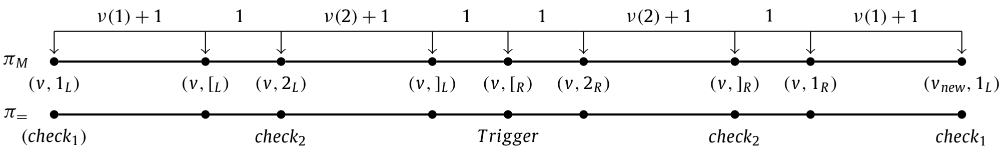
$$o[x_M = v] \rightarrow \bigvee_{u \in V_{check}} \exists o'[x = u].\, \mathsf{s}(o') - \mathsf{s}(o) \in [0,0]$$

Next, we define trigger rules capturing under the strong minimal semantics, the left-right requirement and the *op*-requirement for each M-instruction *op*.

Trigger rules for the left-right requirement. The encoding ensures that the left-right requirement is fulfilled along a pseudo-computation code π_M if and only if for each token $tk_{[_R}$ marked by $[_R$, the following holds: for the last token marked by 1_L (resp., 2_L) preceding token $tk_{[_R}$ and the first token marked by 1_L (resp., $]_R$) following token $tk_{[_R}$, their start points have the same time distance from the start point of $tk_{[_R}$ (see the figure in the following). For this, we first require by trigger rules that for the timeline $\pi_=$ for $x_=$ (synchronized with π_M), a token has value *trigger* (resp., has value $check_1$, resp., has value $check_2$) iff the associated token along π_M has a value in $V_\Delta \times \{[_R\}$ (resp., in $V_\Delta \times \{1_L\}$, resp., in $V_\Delta \times \{2_L,]_R\}$). The trigger rules ensuring the previous requirement are similar to the ones exploited for the synchronization requirement. Finally, we require that for each trigger-token $tk_{trigger}$ along the timeline $\pi_=$ for $x_=$, and for each $\ell = 1, 2$, the start points of the last $check_\ell$-token of $\pi_=$ preceding token $tk_{trigger}$ and the first $check_\ell$-token following token $tk_{trigger}$ have the same time distance from the start point of $tk_{trigger}$ (*check requirement for variable* $x_=$). By the strong minimal semantics, the check requirement for variable $x_=$ can be expressed by the following two trigger rules which ensure that for the $check_\ell$-tokens ($\ell = 1, 2$) of the timeline for $x_=$ whose start points have the smallest time distance from the start point of the trigger, there is one preceding the trigger and one following the trigger (recall that under the strong minimal semantics, each non-trigger selected token needs to have the smallest time distance from the trigger token over the tokens belonging to the same variable and having the same value):

$$o[x_= = trigger] \rightarrow \exists o_1[x_= = check_1] \exists o_2[x_= = check_2].\, \bigwedge_{\ell=1,2} \mathsf{s}(o) - \mathsf{s}(o_\ell) \in [0, \infty)$$

$$o[x_= = trigger] \rightarrow \exists o_1[x_= = check_1] \exists o_2[x_= = check_2].\, \bigwedge_{\ell=1,2} \mathsf{s}(o_\ell) - \mathsf{s}(o) \in [0, \infty)$$

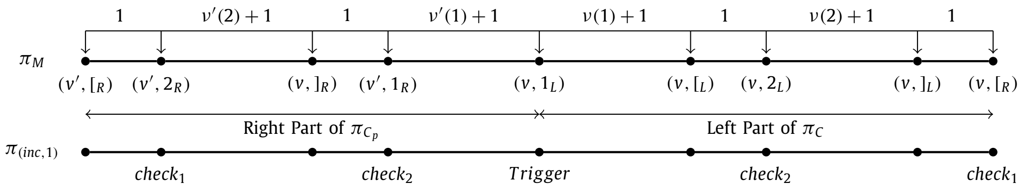


Trigger rules for increment, zero-test, and decrement instructions.

Let $op \in \{\mathsf{inc}, \mathsf{zero\_test}, \mathsf{dec}\} \times \{1, 2\}$ and c be the counter associated with *op*. Given a pseudo-computation code π_M and a non-initial pseudo-configuration code π_C of π_M with V_Δ-value in V_{op}, let tk_1 and tk_2 be the tokens for counter 1 and 2, respectively, in the left part of π_C, and tk'_1 and tk'_2 be the tokens for counter 1 and 2, respectively, in the right part of the pseudo-configuration code π_{C_p} preceding π_C along π_M (see the following figure). We need to ensure that the durations of tokens tk_{3-c} and tk'_{3-c} coincide, and

- case $op = (\mathsf{inc}, c)$: the duration of tk_c is the duration of tk'_c plus one;
- case $op = (\mathsf{dec}, c)$: the duration of tk_c is the duration of tk'_c minus one;
- case $op = (\mathsf{zero\_test}, c)$: the durations of tokens tk_c and tk'_c coincide.

Here, we focus on the case where $op = (\mathsf{dec}, c)$ for some $c \in \{1, 2\}$ (the cases of zero-test and increment instructions being similar). In particular, in the figure we report the timeline for $op = (\mathsf{dec}, 1)$. Our encoding ensures that the previous requirement is fulfilled iff for each token tk_{1_L} of π_M with value in $V_{op} \times \{1_L\}$, the following holds, where $tag_c = 1_R$ if $c = 1$, and $tag_c =]_R$ otherwise: for the last token marked by 2_R (resp., tag_c) preceding token tk_{1_L} and the first token marked by $[_R$ (resp., 2_L) following token tk_{1_L}, their start points have the same time distance from the start point of tk_{1_L}. The previous requirement can be enforced as done for the case of the left-right requirement.



□

4. Some novel extensions of Event-Clock Automata (ECA)

As we shall prove in Section 5, the TP problem under the *weak minimal semantics* is **PSPACE**-complete. Such a complexity result is proved by a non-trivial exponential-time reduction to the non-emptiness of TA. To handle the trigger rules under the weak minimal semantics, we shall exploit, as an intermediate step in the reduction, a strictly more expressive extension of ECA (Event Clock Automata) [28], called ECA$^+$, which is introduced and studied in this paper. This novel extension of ECA is obtained by allowing a larger class of atomic event-clock constraints, namely, diagonal constraints between clocks of the same polarity (past or future) and *sum constraints* between clocks of *opposite polarity*. We show that, similarly to ECA, ECA$^+$ are closed under language Boolean operations and can be translated in exponential time into equivalent TA with an exponential number of control states, but a linear number of clocks. This result will be exploited in Section 5 for handling the trigger rules of the given TP domain under the weak minimal semantics in order to obtain an exponential-time reduction of the weak minimal TP problem to nonemptiness of TA. We believe that such an extension, which is motivated by technical reasons, is interesting per se.

In order to understand in depth this new class of automata, we show that if we relax the requirements in the diagonal and sum constraints, the resulting class of automata, called ECA^{++}, turns out to be very expressive, and its nonemptiness problem becomes undecidable.

The rest of the section is organized as follows. In Subsection 4.1, we briefly recall the class of Timed Automata [25]. Next, in Subsection 4.2, we introduce and address expressiveness issues and closure properties for ECA$^+$ and ECA^{++}. Then, in Subsection 4.3, we define a mapping of ECA$^+$ into equivalent TA. Finally, in Subsection 4.4, we show undecidability of the nonemptiness problem for ECA^{++}.

4.1. Timed Automata

In this section, we recall the class of Timed Automata (TA) [25] over (finite) timed words.

Let Σ be a finite alphabet. A *timed word* w over Σ is a *finite* word $w = (a_0, \tau_0) \cdots (a_n, \tau_n)$ over $\Sigma \times \mathbb{R}_+$ (intuitively, for each i, τ_i is the time at which a_i occurs) such that $\tau_i \leq \tau_{i+1}$ for all $0 \leq i < n$ (monotonicity). The timed word w is also denoted by (σ, τ), where σ is the finite untimed word $a_0 \cdots a_n$ and τ is the sequence of timestamps $\tau_0 \cdots \tau_n$. A *timed language* over Σ is a set of timed words over Σ.

Definition 5 *(Timed Automata).* A TA over Σ is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, C, \Delta, F)$, where Q is a finite set of (control) states, $Q_0 \subseteq Q$ is the set of initial states, C is a finite set of *clocks*, $F \subseteq Q$ is the set of accepting states, and Δ is the finite set of transitions (q, a, θ, Res, q') such that $q, q' \in Q$, $a \in \Sigma$, $Res \subseteq C$ is a clock reset set, and θ is a *clock constraint* over C, that is a conjunction of atomic formulas of the form $c \sim n$ (*simple constraints*) with $c \in C$, $\sim \in \{<, \leq, \geq, >\}$, and $n \in \mathbb{N}$.

We denote by $K_{\mathcal{A}}$ the maximal constant used in the clock constraints of $\mathcal{A}$.

Intuitively, in a TA $\mathcal{A}$, while transitions are instantaneous, time can elapse in a control state. The clocks progress at the same speed and can be reset independently of each other when a transition is executed, in such a way that each clock keeps track of the time elapsed since the last reset. Moreover, clock constraints are used as guards of transitions to restrict the behavior of the automaton. Formally, a configuration of $\mathcal{A}$ is a pair (q, val), where $q \in Q$ and $val : C \to \mathbb{R}_+$ is a clock valuation for C assigning to each clock a non-negative real number. For $t \in \mathbb{R}_+$ and a reset set $Res \subseteq C$, the valuations $(val + t)$ and $val[Res]$ are defined as: for all $c \in C$, $(val + t)(c) = val(c) + t$, and $val[Res](c) = 0$ if $c \in Res$ and $val[Res](c) = val(c)$ otherwise. For a clock constraint θ, val satisfies θ, written $val \models \theta$, if for each conjunct $c \sim n$ of θ, $val(c) \sim n$.

A run r of $\mathcal{A}$ on a timed word $w = (a_0, \tau_0) \cdots (a_n, \tau_n)$ over Σ is a sequence of configurations $r = (q_0, val_0) \cdots (q_{n+1}, val_{n+1})$ starting at an initial configuration (q_0, val_0), with $q_0 \in Q_0$ and $val_0(c) = 0$ for all $c \in C$, and such that for all $0 \leq i \leq n$ (we let $\tau_{-1} = 0$): $(q_i, a_i, \theta, Res, q_{i+1}) \in \Delta$ for some constraint θ and reset set Res, $(val_i + \tau_i - \tau_{i-1}) \models \theta$ and $val_{i+1} = (val_i + \tau_i - \tau_{i-1})[Res]$. The run r is *accepting* if $q_{n+1} \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ of $\mathcal{A}$ is the set of timed words w over Σ such that there is an accepting run of $\mathcal{A}$ over w.

4.2. ECA$^+$ and ECA^{++}

In this section, we introduce an extension, denoted by ECA$^+$, of *Event Clock Automata* (ECA) [28], the latter being a well-known determinizable subclass of TA where the explicit reset of clocks is disallowed. In ECA, clocks have a predefined association with the input alphabet symbols and their values refer to the time distances from previous and next occurrences of input symbols. ECA$^+$ extend ECA by allowing a larger class of atomic event-clock constraints, namely diagonal constraints (alias difference constraints) between clocks of the same polarity (i.e., between past clocks or between future clocks) and *sum constraints* between clocks of *opposite polarity* (i.e. between a past clock and a future clock). Additionally, we consider the extension of ECA$^+$, denoted by ECA^{++}, where the polarity requirements in the diagonal and sum constraints are relaxed.

Here, we adopt a propositional-based approach where the input alphabet is given by $2^{\mathcal{P}}$ for a given set of atomic propositions. The set $C_{\mathcal{P}}$ of event clocks associated with $\mathcal{P}$ is given by $C_{\mathcal{P}} := \bigcup_{p\in\mathcal{P}}\{\overleftarrow{c}_p, \overrightarrow{c}_p\}$. Thus, for each proposition $p \in \mathcal{P}$, there are two event clocks: the *event-recording or past clock* $\overleftarrow{c}_p$ which records the time elapsed since the last occurrence of p in the input word (if any), and the *event-predicting or future clock* $\overrightarrow{c}_p$ which provides the time required to the next occurrence of p (if any). A special value $\perp$ is exploited to denote the absence of a past (resp., future) occurrence of proposition p. Formally, the values of the event clocks at a position i of a timed word w can be deterministically determined as follows.

Definition 6 *(Deterministic clock valuations).* An *event-clock valuation* (over $C_{\mathcal{P}}$) is a mapping $val : C_{\mathcal{P}} \mapsto \mathbb{R}_+ \cup \{\perp\}$, assigning to each event clock a value in $\mathbb{R}_+ \cup \{\perp\}$ ($\perp$ is the *undefined* value). For a timed word $w = (\sigma, \tau)$ over $2^{\mathcal{P}}$ and a position $0 \le i < |w|$, the *event-clock valuation* val_i^w, specifying the values of the event clocks at position i along w, is defined as follows for each $p \in \mathcal{P}$:

$$val_i^w(\overleftarrow{c}_p) = \begin{cases} \tau_i - \tau_\ell & \text{if there exists the unique } 0 \le \ell < i : p \in \sigma(\ell) \text{ and} \\ & \quad \forall k : \ell < k < i \Rightarrow p \notin \sigma(k) \\ \perp & \text{otherwise} \end{cases}$$

$$val_i^w(\overrightarrow{c}_p) = \begin{cases} \tau_\ell - \tau_i & \text{if there exists the unique } i < \ell < |\sigma| : p \in \sigma(\ell) \text{ and} \\ & \quad \forall k : i < k < \ell \Rightarrow p \notin \sigma(k) \\ \perp & \text{otherwise} \end{cases}$$

Definition 7 *(ECA$^+$ and ECA^{++}).* An ECA$^+$ over $2^{\mathcal{P}}$ is a tuple $\mathcal{A} = (2^{\mathcal{P}}, Q, Q_0, C_{\mathcal{P}}, \Delta, F)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states, and Δ is a finite set of transitions (q, a, θ, q'), where $q, q' \in Q$, $a \in 2^{\mathcal{P}}$, and θ is an ECA$^+$ *event-clock constraint* that is a conjunction of atomic formulas of the following forms, where $p, p' \in \mathcal{P}$, $\sim \in \{<, \le, \ge, >\}$, and $n_\perp \in \mathbb{N} \cup \{\perp\}$:

- $\overleftarrow{c}_p \sim n_\perp$ or $\overrightarrow{c}_p \sim n_\perp$ (*simple constraints*);
- $\overleftarrow{c}_p - \overleftarrow{c}_{p'} \sim n_\perp$ or $\overrightarrow{c}_p - \overrightarrow{c}_{p'} \sim n_\perp$ (*diagonal constraints* between event clocks of the same polarity);
- $\overleftarrow{c}_p + \overrightarrow{c}_{p'} \sim n_\perp$ (*sum constraints* between event clocks of opposite polarity).

We denote by $K_{\mathcal{A}}$ the maximal constant used in the event-clock constraints of $\mathcal{A}$.

An ECA [28] is an ECA$^+$ which does *not* use diagonal and sum constraints. We also consider the extension of ECA$^+$, denoted by ECA^{++}, where the transition guards also exploit as conjuncts diagonal (resp., sum) constraints over event clocks of opposite polarity (resp., of the same polarity).

Let us fix an event-clock valuation *val*. We extend in the natural way the valuation *val* to differences (resp., sums) of event clocks: for all $c, c' \in C_{\mathcal{P}}$, $val(c - c') = val(c) - val(c')$ and $val(c + c') = val(c) + val(c')$ where each sum or difference involving $\perp$ evaluates to $\perp$. Given an event-clock constraint θ, *val* satisfies θ, written $val \models \theta$, if for each conjunct $t \sim n_\perp$ of θ, either (i) $val(t) \neq \perp$, $n_\perp \neq \perp$, and $val(t) \sim n_\perp$, or (ii) $val(t) = \perp$, $n_\perp = \perp$, and $\sim \in \{\le, \ge\}$.

A run π of an ECA$^+$ (resp., ECA^{++}) $\mathcal{A}$ over a timed word $w = (\sigma, \tau)$ is a sequence of states $\pi = q_0, \ldots, q_{|w|}$ such that $q_0 \in Q_0$ and for all $0 \le i < |w|$, $(q_i, \sigma(i), \theta, q_{i+1}) \in \Delta$ for some constraint θ such that $val_i^w \models \theta$. The run π is *accepting* if $q_{|w|} \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ of $\mathcal{A}$ is the set of timed words w over $2^{\mathcal{P}}$ such that there is an accepting run of $\mathcal{A}$ on w.

We observe that the standard class of ECA allows to express atomic constraints on the time distance between two events where one of them occurs at the current time. In this way, it is possible to specify standard bounded-time response requirements such as "every request p is followed by a response q within k time units". The novel class of ECA$^+$ extends ECA by the additional ability to temporally compare two events where none of them occurs at the current time. This allows, for instance, to express a *context-based* version of bounded-time response properties. In order to illustrate this, we consider a simple example. Assume that a client can send two kinds of requests to the server: *default* and *critical* ones. While there is no bound on the time for processing a default request, we require that every critical request is processed within a given amount of time, say 1 time unit. This scenario is modeled by the ECA$^+$ in Fig. 4, where proposition *reqD* (*reqC*) models the sending of a default (resp., critical) request to the server, *exec* represents the starting of request processing by the server, and proposition *send* represents the termination of server processing and the sending of a response to the client.

We now formally show that ECA$^+$ are more expressive than ECA, i.e. the class of ECA$^+$ timed languages includes strictly the class of ECA timed languages. Let us consider the ECA$^+$ $\mathcal{A}_p$, depicted below, whose set $\mathcal{P}$ of atomic propositions consists of a unique proposition p.

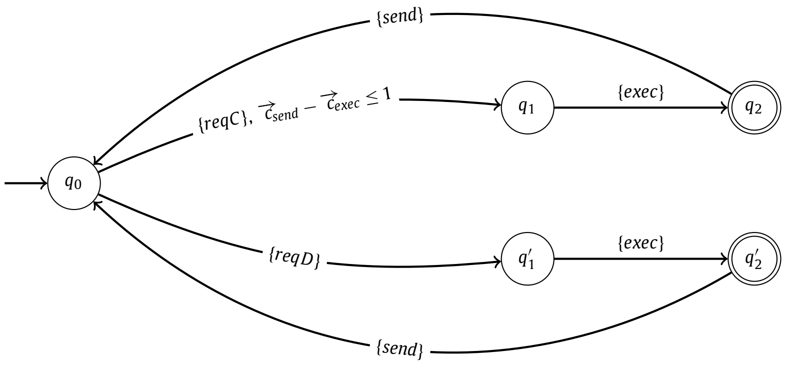
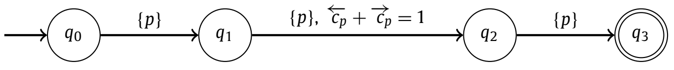

{send}
{reqC}, $\overrightarrow{c}_{send} - \overrightarrow{c}_{exec} \le 1$
q_1
{exec}
q_2
q_0
{reqD}
q'_1
{exec}
q'_2
{send}

Fig. 4. Example of an ECA$^+$ specifying a context-based bounded-time response requirement.



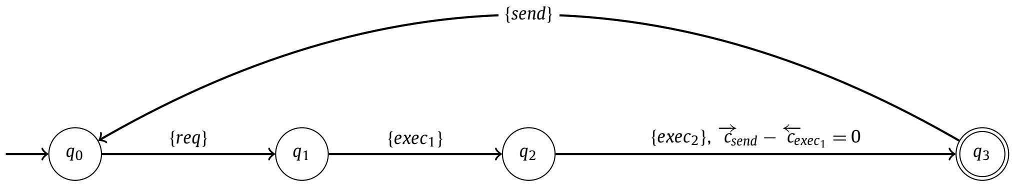

{send}
q_0
{req}
q_1
{exec$_1$}
q_2
{exec$_2$}, $\overrightarrow{c}_{send} - \overleftarrow{c}_{exec_1} = 0$
q_3

Fig. 5. Example of an ECA^{++}.

The automata $\mathcal{A}_p$ accepts timed words w of length 3 of the form $(\{p\}, \tau_0), (\{p\}, \tau_1), (\{p\}, \tau_2)$ such that the time difference between the first and last symbol is 1, i.e. $\tau_2 - \tau_0 = 1$. We show that there is no ECA accepting $\mathcal{L}_T(\mathcal{A}_p)$. Indeed, let w_1 and w_2 be the timed words over $2^{\mathcal{P}}$ (with $\mathcal{P} = \{p\}$) of length 3 defined as follows:

- $w_1 = (\{p\}, 0), (\{p\}, 0.1), (\{p\}, 0.9)$;
- $w_2 = (\{p\}, 0), (\{p\}, 0.1), (\{p\}, 1.0)$.

For each $0 \le i \le 2$, let us denote by val_i^1 and val_i^2 the event-clock valuations over $C_{\mathcal{P}}$ associated with w_1 and w_2, respectively, at position i. By construction, the following easily follows for all positions $0 \le i \le 2$ and event-clocks $z \in C_{\mathcal{P}}$: either (i) $val_i^1(z) = val_i^2(z)$, or (ii) $0 < val_i^1(z) < 1$ and $0 < val_i^2(z) < 1$. Hence, simple atomic event-clock constraints cannot distinguish the valuations val_i^1 and val_i^2. It follows that for each ECA $\mathcal{A}$ over $2^{\mathcal{P}}$, $w_1 \in \mathcal{L}_T(\mathcal{A})$ iff $w_2 \in \mathcal{L}_T(\mathcal{A})$. On the other hand, by definition of the language $\mathcal{L}_T(\mathcal{A}_p)$, $w_2 \in \mathcal{L}_T(\mathcal{A}_p)$ and $w_1 \notin \mathcal{L}_T(\mathcal{A}_p)$. Hence, $\mathcal{L}_T(\mathcal{A}_p)$ is not definable by ECA and we obtain the following result.

Theorem 3. *For a proposition p, there is a timed language over $2^{\{p\}}$ which is definable by ECA$^+$ but is* not *definable by ECA. Hence, ECA$^+$ are strictly more expressive than ECA.*

For completeness, we also investigate the class of ECA^{++}, the extension of ECA$^+$ where the polarity requirements in the diagonal and sum constraints are relaxed. ECA^{++} extend ECA$^+$ by adding a very powerful layer of modeling facilities. In particular, arbitrary sum constraints allow to compare the durations of two temporal intervals where one of them ends at the current time and the other one starts at the current time. Moreover, arbitrary diagonal constraints can enforce a bound on the sum of durations of two temporal intervals both starting (resp., ending) at the current time. As a simple example, depicted in Fig. 5, we consider a variant of the ECA$^+$ illustrated in Fig. 4, where the server processing of a client request is subdivided in two phases of equal duration. Note that in Fig. 5 the starting of the first phase (resp., second phase) is modeled by proposition $exec_1$ (resp., $exec_2$).

Closure properties. Similarly to the case of ECA [28], the class of timed languages accepted by ECA$^+$ (resp., ECA^{++}) is closed under Boolean operations.

Theorem 4 *(Closure properties). Given two ECA$^+$ (resp., ECA^{++}) $\mathcal{A}$ and $\mathcal{A}'$ over $2^{\mathcal{P}}$ with n and n' states, respectively, one can construct ECA$^+$ (resp., ECA^{++}) $\mathcal{A}_\cup$, $\mathcal{A}_\cap$, and $\mathcal{A}_c$ such that:*

- *$\mathcal{A}_\cup$ (resp., $\mathcal{A}_\cap$) accepts $\mathcal{L}_T(\mathcal{A}) \cup \mathcal{L}_T(\mathcal{A}')$ (resp., $\mathcal{L}_T(\mathcal{A}) \cap \mathcal{L}_T(\mathcal{A}')$) and has $n + n'$ (resp., $n \cdot n'$) states and greatest constant* $\max(K_\mathcal{A}, K_{\mathcal{A}'})$*;*
- *$\mathcal{A}_c$ accepts the complement of $\mathcal{L}_T(\mathcal{A})$ and has $2^{O(n)}$ states and greatest constant $K_\mathcal{A}$.*

In the remaining part of the subsection we give the proof of Theorem 4. Actually, the results for union and intersection are straightforward and omitted, and we focus on the closure under complementation which is crucially based on the fact that event-clock values are purely determined by the input timed word. We prove the complementation result for the class of ECA^{++} (the result for ECA^{+} being similar). We define an homomorphism from ECA^{++} to standard Nondeterministic Finite Automata (NFA) over finite words and vice versa. Note that an NFA is defined as an ECA^{++} but we omit the set of event clocks, and the set of clock constraints occurring in the transition function.

Let us fix an ECA^{++} $\mathcal{A} = (2^\mathcal{P}, Q, Q_0, C_\mathcal{P}, \Delta, F)$ over $2^\mathcal{P}$ and let $Const = \{n_0, \dots, n_k\}$ be the set of constants used in the event-clock constraints of $\mathcal{A}$ ordered for increasing values, i.e. such that $0 \le n_0 < n_1 \dots < n_k$. The finite set $Intv_\mathcal{A}$ of intervals over $\mathbb{R} \cup \{\perp\}$ is defined as follows:

$$Intv_\mathcal{A} := \{[\perp,\perp], (-\infty, 0), [0,0], (0, n_0)\} \cup \bigcup_{i=0}^{i=k-1} \{[n_i, n_i], (n_i, n_{i+1})\} \cup \{[n_k, n_k], (n_k, \infty)\}$$

Note that we also consider the interval $(-\infty, 0)$ consisting of the negative real numbers. The set *Terms* of ECA^{++} terms over $\mathcal{P}$ is the set of expressions of the form c, or $c - c'$, or $c + c'$, where $c, c' \in C_\mathcal{P}$ (i.e., the set of left-hand side expressions in the atomic event-clock constraints of ECA^{++} over $2^\mathcal{P}$). For a term t and an interval $I \in Intv_\mathcal{A}$, we denote by $t \in I$, the event-clock constraint $\theta(t, I)$ defined as follows:

- if $I = [\perp, \perp]$, then $\theta(t, I) := t \le \perp \wedge t \ge \perp$;
- if $I = (-\infty, 0)$, then $\theta(t, I) := t < 0$;
- if $I = (n, n')$ (resp., $I = [n, n]$, resp., $I = (n, \infty)$) where $n, n' \in \mathbb{N}$, then $\theta(t, I) := t > n \wedge t < n'$ (resp., $\theta(t, I) := t \ge n \wedge t \le n$, resp., $\theta(t, I) := t > n$).

A *region g of $\mathcal{A}$* is a mapping $g : Terms \mapsto Intv_\mathcal{A}$ assigning to each term in *Terms* an interval in $Intv_\mathcal{A}$ such that for each $t \in Terms$, if $g(t) = (-\infty, 0)$, then $t = c - c'$ for some clocks c and c' and $g(c' - c) \neq (-\infty, 0)$. The mapping g induces the ECA^{++} event-clock constraint $\bigwedge_{t \in Terms} t \in g(t)$. We denote by $[g]$ the (possibly empty) set of event-clock valuations over $C_\mathcal{P}$ satisfying the event-clock constraint associated with g, and by *Reg* the set of regions of $\mathcal{A}$. For an ECA^{++} event-clock constraint θ over $C_\mathcal{P}$, let $[\theta]$ be the set of event-clock valuations over $C_\mathcal{P}$ satisfying θ.

Remark 1. By construction, the following properties hold.

- The set *Reg* of regions represents a partition of the set of event-clock valuations over $C_\mathcal{P}$:
  (i) for all event-clock valuations *val* over $C_\mathcal{P}$, there is a region $g \in Reg$ such that $val \in [g]$,
  (ii) for all regions $g, g' \in Reg$, $g \neq g' \Rightarrow [g] \cap [g'] = \emptyset$.
- for each event-clock constraint θ of $\mathcal{A}$ and region $g \in Reg$, either $[g] \subseteq [\theta]$ or $[g] \cap [\theta] = \emptyset$.

We associate with the alphabet $2^\mathcal{P}$ and the set of regions *Reg* the alphabet $\Lambda = 2^\mathcal{P} \times Reg$, called *interval alphabet*. Elements of Λ are pairs of the form (a, g), where $a \in 2^\mathcal{P}$ and g is a region of $\mathcal{A}$ which is meant to represent the associated event-clock constraint $\bigwedge_{t \in Terms} t \in g(t)$. A word $\lambda = (a_0, g_0) \dots (a_{n-1}, g_{n-1})$ over Λ induces in a natural way a set of timed words over $2^\mathcal{P}$, denoted $tw(\lambda)$, defined as follows: $w = (\sigma, \tau) \in tw(\lambda)$ iff $\sigma = a_0 \dots a_{n-1}$ and for all $0 \le i \le n-1$, $val_i^w \in [g_i]$. We extend the mapping tw to languages $\mathcal{L}$ over Λ in the obvious way: $tw(\mathcal{L}) := \bigcup_{\lambda \in \mathcal{L}} tw(\lambda)$. By means of the mapping tw, words over Λ define a partition of the set of timed words over $2^\mathcal{P}$.

Lemma 1. *The following two statements hold:*

1. *for each timed word w over $2^\mathcal{P}$, there is a word λ over Λ such that $w \in tw(\lambda)$;*
2. *for all words λ and λ' over Λ, if $\lambda \neq \lambda'$, then $tw(\lambda) \cap tw(\lambda') = \emptyset$.*

Proof. As for Statement 1, let $w = (\sigma_0, \tau_0) \dots (\sigma_{n-1}, \tau_{n-1})$ be a timed word over $2^\mathcal{P}$. By Remark 1, for all $0 \le i \le n-1$, there is a region $g_i \in Reg$ such that $val_i^w \in [g_i]$. Let $\lambda = (\sigma_0, g_0) \dots (\sigma_{n-1}, g_{n-1})$. We have that $w \in tw(\lambda)$, and the result follows.

As for Statement 2, let λ and λ' be two distinct words over Λ. Let us assume that $tw(\lambda) \cap tw(\lambda') \neq \emptyset$ and derive a contradiction. Hence, by construction, $|\lambda| = |\lambda'| = n$ for some n, $\lambda = (a_0, g_0) \dots (a_{n-1}, g_{n-1})$, $\lambda' = (a_0, g'_0) \dots (a_{n-1}, g'_{n-1})$, and there is a timed word w over Σ of the form $w = (a_0, \tau_0) \dots (a_{n-1}, \tau_{n-1})$ such that $val_i^w \in [g_i] \cap [g'_i]$ for all $i \ge 0$. Since $\lambda \neq \lambda'$, there exists $0 \le i \le n-1$ such that $g_i \neq g'_i$. By Remark 1, $[g_i] \cap [g'_i] = \emptyset$ which is a contradiction since $val_i^w \in [g_i] \cap [g'_i]$, and the result follows. □

The following two propositions establish an untimed homomorphism from ECA^{++} to NFA, and a timed homomorphism from NFA to ECA^{++}, respectively.

Proposition 2 *(Untimed homomorphism). Let $\mathcal{A} = (2^{\mathcal{P}}, Q, Q_0, C_{\mathcal{P}}, \Delta, F)$ be an ECA^{++}, and Λ be the interval alphabet induced by $\mathcal{A}$. Then, one can construct an NFA Untimed$(\mathcal{A})$ over Λ of the form $(\Lambda, Q, Q_0, \Delta', F)$ such that $tw(\mathcal{L}(Untimed(\mathcal{A}))) = \mathcal{L}_T(\mathcal{A})$.*

Proof. The transition relation Δ' of *Untimed*$(\mathcal{A})$ consists of the transitions $(q, (a, g), q')$ such that there is some event-clock constraint θ of $\mathcal{A}$ so that $[g] \subseteq [\theta]$ and $(q, a, \theta, q') \in \Delta$. By Remark 1 and Lemma 1(1), we easily derive the correctness of the construction. □

Proposition 3 *(Timed homomorphism). Let $\mathcal{A} = (\Lambda, Q, Q_0, \Delta, F)$ be an NFA over an interval alphabet associated with $2^{\mathcal{P}}$. Then, one can construct an ECA^{++} Timed$(\mathcal{A})$ over $2^{\mathcal{P}}$ of the form $\mathcal{A} = (2^{\mathcal{P}}, Q, Q_0, C_{\mathcal{P}}, \Delta', F)$ such that $\mathcal{L}_T(Timed(\mathcal{A})) = tw(\mathcal{L}(\mathcal{A}))$.*

Proof. The transition relation Δ' of *Timed*$(\mathcal{A})$ consists of the transitions (q, a, θ, q') such that there is $(q, (a, g), q') \in \Delta$ so that $\theta = \bigwedge_{t \in Terms} t \in g(t)$. By Remark 1 and Lemma 1(1), we easily derive the correctness of the construction. □

By Lemma 1, Proposition 2, Proposition 3, and the known closure properties of NFA, we have the following result.

Theorem 5 *(Closure under complementation of ECA^{++}). Given an ECA^{++} $\mathcal{A}$ over $2^{\mathcal{P}}$ with n states, one can construct in singly exponential time an ECA^{++} $\overline{\mathcal{A}}$ over $2^{\mathcal{P}}$ accepting the complement of $\mathcal{L}_T(\mathcal{A})$ having $2^{O(n)}$ states and greatest constant $K_{\mathcal{A}}$.*

Proof. Let $\mathcal{A}$ be an ECA^{++} over $2^{\mathcal{P}}$ with n states, and Λ be the interval alphabet induced by $\mathcal{A}$. By Proposition 2, we can construct an NFA *Untimed*$(\mathcal{A})$ over Λ with n states such that $tw(\mathcal{L}(Untimed(\mathcal{A}))) = \mathcal{L}_T(\mathcal{A})$. By classical results, starting from the NFA *Untimed*$(\mathcal{A})$, one can construct in singly exponential time an NFA $\overline{Untimed(\mathcal{A})}$ over Λ accepting $\Lambda^* \setminus \mathcal{L}(Untimed(\mathcal{A}))$ with $2^{O(n)}$ states. Applying Proposition 3 to the NFA $\overline{Untimed(\mathcal{A})}$, one can construct in linear time an ECA^{++} $\overline{\mathcal{A}}$ over $2^{\mathcal{P}}$ with $2^{O(n)}$ states such that $\mathcal{L}_T(\overline{\mathcal{A}}) = tw(\Lambda^* \setminus \mathcal{L}(Untimed(\mathcal{A})))$. Since $\mathcal{L}_T(\mathcal{A}) = tw(\mathcal{L}(Untimed(\mathcal{A})))$, by Lemma 1, $\overline{\mathcal{A}}$ accepts all and only the timed words over $2^{\mathcal{P}}$ which are not in $\mathcal{L}_T(\mathcal{A})$, and the result follows. □

4.3. From ECA^{+} to Timed Automata

It is known that ECA can be translated in singly exponential time into equivalent TA [28]. In this section, we generalize this result to the class of ECA^{+}.

Theorem 6 *(From ECA^{+} to TA). Given an ECA^{+} $\mathcal{A}$ over $2^{\mathcal{P}}$, one can construct in exponential time a TA $\mathcal{A}'$ over $2^{\mathcal{P}}$ such that $\mathcal{L}_T(\mathcal{A}') = \mathcal{L}_T(\mathcal{A})$ and $K_{\mathcal{A}'} = K_{\mathcal{A}}$. Moreover, $\mathcal{A}'$ has $n \cdot 2^{O(p)}$ states and $O(p)$ clocks, where n is the number of $\mathcal{A}$-states and p is the number of event-clock atomic constraints used by $\mathcal{A}$.*

Proof. We sketch the main ideas underlying the translation providing full details in Appendix B. Let $\mathcal{A} = (2^{\mathcal{P}}, Q, Q_0, C_{\mathcal{P}}, \Delta, F)$ be an ECA^{+} over $2^{\mathcal{P}}$. The TA $\mathcal{A}'$ accepting $\mathcal{L}_T(\mathcal{A})$ is essentially obtained from $\mathcal{A}$ by replacing each atomic event-clock constraint of $\mathcal{A}$ with a set of standard clocks together with associated reset operations and clock constraints. To remove simple event-clock constraints of $\mathcal{A}$, we can proceed as in [28]. Therefore, we focus on the removal of diagonal constraints (over clocks of the same polarity) and sum constraints (over clocks of opposite polarity).

Removal of diagonal predicting constraints. Let us consider a diagonal predicting clock constraint $\eta : \overrightarrow{c}_p - \overrightarrow{c}_{p'} \sim n_\perp$ of $\mathcal{A}$ with $n_\perp \in \mathbb{N} \cup \{\perp\}$. We consider the case $n_\perp \neq \perp$ (the case $n_\perp = \perp$ being simpler). For handling the constraint η, the TA $\mathcal{A}'$ exploits the fresh standard clock c_η and in case $n_\perp = 0$ and $\sim \in \{\geq, <\}$, the additional fresh standard clock $\hat{c}_\eta$. The first (resp., second) clock is reset *only if* proposition p' (resp., p) occurs in the current input symbol. Assume that the prediction η is done by $\mathcal{A}$ at position i of the input word for the first time. Then, the simulating TA $\mathcal{A}'$ carries the obligation η in its control state in order to check that there are next positions where p and p' occur and $\tau_p - \tau_{p'} \sim n_\perp$ holds, where τ_p (resp., $\tau_{p'}$) is the timestamp associated with the first next position $i_p > i$ (resp., $i_{p'} > i$) where p (resp., p') occurs. Note that all the predictions η done by $\mathcal{A}$ before positions i_p and $i_{p'}$ correspond to the same obligation. First, assume that the first next position $i_{p'} > i$ where p' occurs strictly precedes position i_p. In this case, on reading position $i_{p'}$, $\mathcal{A}'$ resets the clock c_η and replaces the old obligation η with the updated obligation (η, p') in order to check that the constraint $c_\eta \sim n_\perp$ holds when the next p occurs (i.e., at position i_p). If a new prediction η is done at a position $j_{new} \geq i_{p'}$ strictly preceding i_p, the fresh obligation η is carried in the control state together with the obligation (η, p'). We distinguish two cases:

- p' occurs in some position strictly following j_{new} and strictly preceding i_p. Let j' be the smallest of such positions. On reading position j', $\mathcal{A}'$ replaces the old obligations η and (η, p') with (η, p') and resets the clock c_η *iff* η is a lower bound constraint, i.e., $\sim \in \{>, \geq\}$. This is safe since if η is a lower bound, then the fulfillment of prediction η at j_{new} guarantees the fulfillment of prediction η at position i. Vice versa, if η is an upper bound, then the fulfillment

of prediction η at i guarantees the fulfillment of prediction η' at position j_{new}. Thus, when η is a lower bound, new obligations (η, p') rewrite the old ones, while when η is an upper bound, new obligations (η, p') are ignored.
- there is no position strictly following j_{new} and strictly preceding i_p, where p' occurs. In this case, when i_p is read, the old obligation η is replaced by the obligation (η, p) unless p' occurs at position i_p (in the latter case, $\mathcal{A}'$ simply checks that $0 \sim n_\perp$).

In both cases on reading position i_p, the constraint $c_\eta \sim n_\perp$ is checked and the obligation (η, p') is discarded. The case where $i_{p'} = i_p$ is trivial (on reading position i, $\mathcal{A}'$ checks that $0 \sim n_\perp$ holds). Finally, assume that i_p strictly precedes $i_{p'}$. The cases where either $c \neq 0$ or $\sim \in \{\leq, >\}$, since in these cases if η is a lower bound (resp., upper bound), then the prediction η done at position i is not satisfied (resp., is satisfied). Thus, we focus on the case where $c = 0$ and $\sim \in \{\geq, <\}$. If $\sim$ is $\geq$ (resp., $\sim$ is $<$), then on reading position i_p, the clock $\hat{c}_\eta$ is reset and the old obligation η is replaced by the updated obligation (η, p) in order to check that the constraint $\hat{c}_\eta = 0$ (resp., $\hat{c}_\eta > 0$) holds when the next p' occurs (i.e., at position $i_{p'}$). Moreover, if $\sim$ is $\geq$, then new obligations (η, p) occurring before position $i_{p'}$ are ignored, i.e., the clock $\hat{c}_\eta$ is not reset at such positions. On the other hand, if $\sim$ is $<$, then the new obligations (η, p) occurring before position $i_{p'}$ rewrite the old ones. i.e. clock $\hat{c}_\eta$ is reset at such positions.

Finally, in order to ensure that raised obligations about η are eventually checked, the accepting states of $\mathcal{A}'$ do not contain such obligations.

Removal of diagonal recording constraints. Let us consider a diagonal recording clock constraint $\eta : \overleftarrow{c}_p - \overleftarrow{c}_{p'} \sim n_\perp$ of $\mathcal{A}$ where $n_\perp \in \mathbb{N} \cup \{\perp\}$. For each proposition p'' whose associated event-recording clock $\overleftarrow{c}_{p''}$ occurs in some constraint of $\mathcal{A}$, the TA $\mathcal{A}'$ exploits a standard clock $c_{p''}$ which is reset whenever p'' occurs in the current input symbol. Moreover, $\mathcal{A}'$ keeps tracks in the control state of the set of propositions occurred in the prefix w of the input read so far together with the past information concerning the indication whether the following requirement holds or not (such an indication is represented by the presence or not in the control state of the constraint η): the prefix w contains occurrences of both propositions p and p' and in case $n_\perp \neq \perp$, $\tau_{p'} - \tau_p \sim n_\perp$ holds, where τ_p and $\tau_{p'}$ are the timestamps associated with the last occurrences of p and p' in the prefix w, respectively. In order to check these conditions when $n_\perp \neq \perp$, $\mathcal{A}'$ exploits the clock constraint $c_p \sim n_\perp$ whenever p' occurs in the current input symbol a, $p \notin a$, and p previously occurred, and the clock constraint $c_{p'} = 0$ (resp., $c_{p'} > 0$) whenever p occurs in the current input symbol a, $p' \notin a$, p' previously occurred, $c = 0$, and $\sim$ is $\geq$ (resp., $\sim$ is $<$). Thus, when the constraint η is exploited by $\mathcal{A}$ in the current transition, the simulating TA $\mathcal{A}'$ simply checks that the past indication η is present (resp., is not present) in the current control state if $n_\perp \neq \perp$ (resp., $n_\perp = \perp$). If the check is negative, the input is rejected.

Removal of sum constraints. Now, let us consider a sum constraint $\eta : \overleftarrow{c}_p + \overrightarrow{c}_{p'} \sim n_\perp$ of $\mathcal{A}$ where $n_\perp \in \mathbb{N} \cup \{\perp\}$. We consider the case $n_\perp \neq \perp$ (the other case being simpler). In this case, the TA $\mathcal{A}'$ exploits two fresh standard clocks, namely c_η and $\hat{c}_\eta$, which are reset *only if* the proposition p (associated with the event-recording clock of η) occurs in the current input symbol. In particular, $\mathcal{A}$ operates in two modes which alternate each other: the η-mode and the $\hat{\eta}$-mode. Initially $\mathcal{A}'$ is in the η-mode. When there is no obligation about η, the clocks c_η and $\hat{c}_\eta$ are reset whenever proposition p occurs in the current input position. When instead there is an obligation about η, in the η-mode (resp., $\hat{\eta}$-mode), the clock c_η (resp., $\hat{c}_\eta$) is reset whenever proposition p occurs, while the clock $\hat{c}_\eta$ (resp., c_η) is never reset. Moreover, we have two types of obligations mutually exclusive: η and $\hat{\eta}$. The obligation η (resp., $\hat{\eta}$) is raised when the ECA $\mathcal{A}$ exploits the constraint η and the TA $\mathcal{A}'$ is in the η-mode (resp., $\hat{\eta}$-mode) in order to check that the constraint $c_\eta \sim n_\perp$ (resp., $\hat{c}_\eta \sim n_\perp$) holds at the first next position where the proposition p' occurs. More precisely, we first consider the case where η is an upper bound constraint, i.e. $\sim \in \{<, \leq\}$. Assume that at the current input position i, $\mathcal{A}'$ is in the η-mode, there are no obligations about the constraint η in the current control state of $\mathcal{A}'$, and $\mathcal{A}$ exploits in the current transition the event-clock constraint η. If p did not occur in a previous input position, the TA $\mathcal{A}'$ rejects the input. Otherwise, $\mathcal{A}'$ carries the obligation η in its control state in order to check that the constraint $c_\eta \sim n_\perp$ holds at the first position $j_{check} > i$ where the proposition p' occurs (if any). If a proposition p occurs at a position $j \geq i$ of the input strictly preceding j_{check}, the TA proceeds as follows:

- $\mathcal{A}$ switches to the $\hat{\eta}$-phase (if j is the first position following i and preceding j_{check} where p occurs) and the clock $\hat{c}_\eta$ is reset at position j in order to handle the obligations raised at positions h of the input following or coinciding with j_{check}. When position j_{check} is read, the constraint $c_\eta \sim n_\perp$ is checked and the obligation η is discarded unless the constraint η is used in the current transition of $\mathcal{A}$. In the latter case, since the $\hat{\eta}$-mode is active, $\mathcal{A}$ replaces the obligation η with the new obligation $\hat{\eta}$.
- The clock c_η is not reset at position j. Being η an upper bound constraint, if η is used by $\mathcal{A}$ at a position $j' > j$ strictly preceding j_{check}, then the choice of not resetting c_η is safe: the fulfillment of the upper bound constraint η at the previous position i guarantees the fulfillment of the constraint at a position j' such that $j < j' < j_{check}$.

The case where η is a lower bound constraint (i.e. $\sim \in \{>, \geq\}$) is similar, but this time new obligations rewrite the old ones. In particular, whenever η is used by $\mathcal{A}$ and $\mathcal{A}'$ is in the η-phase (resp., $\hat{\eta}$-phase) and p' does not hold at the current input position, the old obligation (if any) is replaced by the new obligation η (resp., $\hat{\eta}$) in order to check that the constraint $c_\eta \sim n_\perp$ (resp., $\hat{c}_\eta \sim n_\perp$) holds at the first next position where the proposition p' occurs.

For ensuring that raised obligations about η are eventually checked, the accepting states of $\mathcal{A}'$ do not contain such obligations. The above described construction is formally reported in Appendix B. $\square$

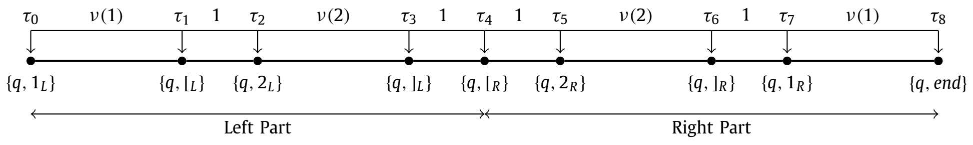
4.4. Undecidability of nonemptiness of ECA^{++}

Since the nonemptiness problem for TA is decidable, as a corollary of Theorem 6 we have that the nonemptiness problem for ECA^{+} is decidable as well. If we consider ECA^{++} we have a completely different picture since the counterpart of Theorem 6 cannot be stated for this class of automata. In fact, we show that the nonemptiness problem for ECA^{++} is undecidable using a reduction from the halting problem for Minsky 2-counter machines [30] similar to the one provided for the strong minimal TP problem.

Theorem 7. *The nonemptiness problem of ECA^{++} is undecidable even for the subclass of ECA^{++} which use only simple atomic event-clock constraints and diagonal constraints over event clocks of opposite polarity of the form $\overleftarrow{c}_p - \overrightarrow{c}_{p'} = 0$.*

Proof. The proof is by a polynomial-time reduction from the halting problem for Minsky 2-counter machines [30]. We fix such a machine $M = (Q, q_{init}, q_{halt}, \Delta)$ and we adopt for the machine M the same notational conventions as in the proof of Theorem 2. Let $\mathcal{P}$ be the set of atomic propositions given by $Q \cup \{1_L, 1_R, 2_L, 2_R, [_L,]_L, [_R,]_R, end\}$. We construct an ECA^{++} $\mathcal{A}_M$ over $\mathcal{P}$ such that $\mathcal{L}_T(\mathcal{A}_M) \neq \emptyset$ iff M halts.

First, we define a suitable encoding of the computations of M which is similar to the one exploited in the proof of Theorem 2. A configuration $C = (q, \nu)$ of M is encoded by the timed word w_C over $2^{\mathcal{P}}$ of length 9 (uniquely determined modulo the initial timestamp) illustrated in the following figure.



Note that the configuration code w_C is subdivided in two parts. In the left part (resp., right part), the encoding of counter 1 (resp., 2) precedes the encoding of counter 2 (resp., 1). The value $\nu(1)$ of counter 1 is encoded by the time distance to the next point, which is $\nu(1)$, of the counter point marked by 1_L in the left part, and the counter point marked by 1_R in the right part, and similarly for counter 2. The four points marked by $[_L$, $]_L$, $[_R$, and $]_R$, respectively, are called *tagged* points and their time distance to the next point is always 1 (*tag requirement*). Intuitively, they are used to check by event clock constraints that increment and decrement M-instructions are correctly encoded.

A computation $\pi = C_1, \ldots, C_k$ of M is then encoded by a timed word w_π of the form $w_\pi = w_1 \cdot \ldots \cdot w_k$ such that (i) w_i is a code of configuration C_i for all $1 \le i \le k$, and (ii) the last timespamp of w_j coincides with the first timestamp of w_{j+1} for all $1 \le j < k$. Note that w_π is uniquely determined modulo the initial timestamp.

We now illustrate the construction of the ECA^{++} $\mathcal{A}_M$ over $2^{\mathcal{P}}$ which accepts all and only the timed words over $2^{\mathcal{P}}$ which encode the computations of M starting at the initial configuration and leading to a halting configuration (hence, $\mathcal{L}_T(\mathcal{A}_M) \neq \emptyset$ iff M halts). The control structure and the acceptance condition of $\mathcal{A}_M$ capture the timed words w of the form $w = w_1 \ldots w_k$ such that for each $1 \le i \le k$, (i) the untimed part of w_i coincides with the untimed part of some configuration code, i.e., it is of the form

$$\{q, 1_L\}, \{q, [_L\}, \{q, 2_L\}, \{q,]_L\}, \{q, [_R\}, \{q, 2_R\}, \{q,]_R\}, \{q, 1_R\}, \{q, end\}$$

for some $q \in Q$, (ii) the state associated with w_1 is q_{init}, and (iii) the state associated with w_k is q_{halt}. We say that each subword w_i is a *pseudo-code*. Moreover, in order to ensure that each pseudo-code is indeed a code of some M-configuration and $w_1 \cdot \ldots \cdot w_k$ is faithful to the evolution of M, $\mathcal{A}_M$ exploits event-clock constraints as follows.

$\mathcal{A}_M$ ensures timestamp consistency of the tagged points of a pseudo-code w_i (tag requirement): i.e., the time distance from a point marked by a proposition in $\{[_L,]_L, [_R,]_R\}$ to the next point is 1. To that purpose, when the $[_L$-point (resp., $]_L$-point) of the left part of w_i is read, $\mathcal{A}_M$ exploits in the current transition the predicting event-clock constraint $\overrightarrow{c}_{2_L} = 1$ (resp., $\overrightarrow{c}_{[_R} = 1$). Moreover, when the $[_R$-point (resp., $]_R$-point) of the right part of w_i is read, $\mathcal{A}_M$ exploits in the current transition the predicting event-clock constraint $\overrightarrow{c}_{2_R} = 1$ (resp., $\overrightarrow{c}_{1_R} = 1$).

$\mathcal{A}_M$ ensures timestamp consistency of the left part and right part of a pseudo-code w_i: i.e., the time distance of the counter 1_L-point (resp., 2_L-point) to the next point coincides with the time distance of the counter 1_R-point (resp., 2_R-point) to the next point. For capturing these requirements, when the $[_R$-point of w_i is read, $\mathcal{A}_M$ exploits in the current transition, together with the predicting event-clock constraint ensuring the tag requirement for $[_R$, the event-clock constraint $\overleftarrow{c}_{2_L} - \overrightarrow{c}_{]_R} = 0 \wedge \overleftarrow{c}_{1_L} - \overrightarrow{c}_{end} = 0$.

$\mathcal{A}_M$ ensures that the timestamp of the last point of w_i coincides with the timestamp of the first point of w_{i+1} if $i < k$. When an *end*-point which is not associated with the halt state of M is read, $\mathcal{A}_M$ exploits in the current transition the predicting constraint $\overrightarrow{c}_{1_L} = 0$.

$\mathcal{A}_M$ keeps track whether the values of counters encoded by the current pseudo-code w_i are 0 or not. When the first point (which is a 1_L-point) of a pseudo-code w_i is read, $\mathcal{A}_M$ guesses if the value of the first counter (resp., second counter) encoded by w_i is zero or not, and keeps tracks of the associated information in the control state until the first point of the next pseudo code (if any) is read. In order to check that the guess is correct, when the counter 2_R-point (resp., 1_R-point) of the right part of w_i is read, $\mathcal{A}_M$ exploits in the current transition the event-clock constraint $\overrightarrow{c}_{]_R} = 0$ or $\overrightarrow{c}_{]_R} > 0$ (resp., $\overrightarrow{c}_{end} = 0$ or $\overrightarrow{c}_{end} > 0$), depending on whether the value of the second counter (resp. first counter) is guessed to have a zero (resp., non-zero) value. In particular, $\mathcal{A}_M$ ensures that *the first pseudo-code w_1 encodes the initial configuration of M.*

$\mathcal{A}_M$ ensures that the pseudo-code w_{i+1} is a code of some configuration of M which is a successor of the configuration encoded by w_i. When the first point of w_{i+1} is read, $\mathcal{A}_M$ guesses an instruction $op \in L$ of M (recall that $L = \{\text{inc}, \text{dec}, \text{zero\_test}\} \times \{1, 2\}$) such that for each $\ell = 1, 2$, if the value of counter ℓ in w_i is 0 (resp., is not 0), then $op \neq (\text{dec}, \ell)$ (resp., $op \neq (\text{zero\_test}, \ell)$). Moreover, $\mathcal{A}_M$ checks that (q_i, op, q_{i+1}) is a transition of M, where q_i (resp., q_{i+1}) is the state associated with w_i (resp., w_{i+1}). If the check is negative, the input is rejected. Otherwise, $\mathcal{A}_M$ proceeds as follows. Here, we crucially exploit the requirements on the tagged points, and the fact that the left part of w_{i+1} is preceded by the right part of w_i (in particular, the encoding of counter 1 in the left part of w_{i+1} is preceded by the encoding of counter 1 in the right part of w_i). Note that we can assume that w_i is the code of some M-configuration, and we denote by n_ℓ the value of counter $\ell = 1, 2$ in w_i.

- $op = (\text{zero\_test}, \ell)$ for some $\ell = 1, 2$: $\mathcal{A}_M$ needs to ensure that the values of counter 1 (resp., 2) in w_i and w_{i+1} coincide. For this, $\mathcal{A}_M$ exploits in the transition from the first point of w_{i+1} (i.e., the counter 1_L-point of w_{i+1}) the event-clock constraint $\overleftarrow{c}_{1_R} - \overrightarrow{c}_{[_L} = 0 \wedge \overleftarrow{c}_{[_R} - \overrightarrow{c}_{[_R} = 0$.
- $op = (\text{inc}, \ell)$ for some $\ell = 1, 2$: $\mathcal{A}_M$ needs to ensure that the values of counter $3 - \ell$ in w_i and w_{i+1} coincide, and the value of counter ℓ in w_{i+1} is $n_\ell + 1$. If $\ell = 1$, then $\mathcal{A}_M$ exploits in the transition from the first point of w_{i+1} (the counter 1_L-point of w_{i+1}) the event-clock constraint $\overleftarrow{c}_{]_R} - \overrightarrow{c}_{[_L} = 0 \wedge \overleftarrow{c}_{[_R} - \overrightarrow{c}_{]_L} = 0$. The previous constraint (together with the tag requirements) ensures that the value of counter 1 in w_{i+1} is $n_1 + 1$, and the sum of the counter values in w_{i+1} is $n_1 + n_2 + 1$. Similarly, if $\ell = 2$, then, $\mathcal{A}_M$ exploits in the transition from the first point of w_{i+1} the event-clock constraint $\overleftarrow{c}_{1_R} - \overrightarrow{c}_{[_L} = 0 \wedge \overleftarrow{c}_{[_R} - \overrightarrow{c}_{]_L} = 0$.
- $op = (\text{dec}, \ell)$ for some $\ell = 1, 2$: $\mathcal{A}_M$ needs to ensure that the values of counter $3 - \ell$ in w_i and w_{i+1} coincide, and the value of counter ℓ in w_{i+1} is $n_\ell - 1$. If $\ell = 1$, then $\mathcal{A}_M$ exploits in the transition from the first point of w_{i+1} (the counter 1_L-point of w_{i+1}) the event-clock constraint $\overleftarrow{c}_{1_R} - \overrightarrow{c}_{2_L} = 0 \wedge \overleftarrow{c}_{2_R} - \overrightarrow{c}_{[_R} = 0$. The previous constraint (together with the tag requirements) ensures that the value of counter 1 in w_{i+1} is $n_1 - 1$, and the sum of the counter values in w_{i+1} is $n_1 + n_2 - 1$. Similarly, if $\ell = 2$, then, $\mathcal{A}_M$ exploits in the transition from the first point of w_{i+1} the event-clock constraint $\overleftarrow{c}_{1_R} - \overrightarrow{c}_{[_L} = 0 \wedge \overleftarrow{c}_{2_R} - \overrightarrow{c}_{[_R} = 0$.

Note that the ECA^{++} $\mathcal{A}_M$ uses only simple atomic event-clock constraints and diagonal constraints over event clocks of opposite polarity of the form $\overleftarrow{c}_p - \overrightarrow{c}_{p'} = 0$. $\square$

5. Decidability of the weak minimal TP problem

In this section, by exploiting the results of Section 4, we show that the weak minimal TP problem is decidable and **PSPACE**-complete. The upper bound is obtained by an exponential-time reduction to nonemptiness of Timed Automata (TA) [25]. In order to handle the trigger rules under the weak minimal semantics, we exploit as an intermediate step the class of ECA$^+$, introduced and investigated in Section 4.

In the following, we fix a TP domain $\mathcal{D} = (SV, R)$. We construct a TA accepting suitable encodings of the multi-timelines of SV which satisfy the rules in R under the weak minimal semantics. For each $x \in SV$, let $x = (V_x, T_x, D_x)$. We first define an encoding of the multi-timelines of SV by means of timed words over $2^{\mathcal{P}}$ for the set $\mathcal{P}$ of propositions given by $\{\text{init}\} \cup \bigcup_{x \in SV} \mathcal{P}_x$ where for each $x \in SV$, $\mathcal{P}_x = \{x\} \times V_x \times \{\text{s}, \text{e}\} \times \{0, 1\}$. We use the propositions in $\mathcal{P}_x$ to encode the tokens tk along a timeline for x: the start point and end point of tk are specified by propositions (x, v, s, b) and (x, v, e, b), respectively, where $b \in \{0, 1\}$ and v is the value of tk. The meaning of the bit $b \in \{0, 1\}$ is explained below. The additional proposition $\text{init} \in \mathcal{P}$ is used to mark the first point of a multi-timeline code in order to check point atoms of trigger rules by ECA$^+$ event-clock constraints. A *code for a timeline for x* is a timed word w over $2^{\mathcal{P}_x}$ of the form

$$\begin{aligned} w = \; & (\{(x, v_0, \text{s}, b_0)\}, \tau_0), \; (\{(x, v_0, \text{e}, b_0)\}, \tau_1), \; (x, v_1, \text{s}, b_1)\}, \tau_1), \; (\{(x, v_1, \text{e}, b_1)\}, \tau_2), \cdots \\ & \cdots (\{(x, v_n, \text{s}, b_n)\}, \tau_n), \; (\{(x, v_n, \text{e}, b_n)\}, \tau_{n+1}) \end{aligned}$$

such that for all $0 \le i \le n$:

- $v_{i+1} \in T_x(v_i)$ if $i < n$;
- $\tau_0 = 0$ and $\tau_{i+1} - \tau_i \in D_x(v_i)$;

- let ℓ_i be the greatest index $0 \leq j < i$ such that $v_j = v_i$ if such an index exists, and let $\ell_i := \bot$ otherwise. Then, $b_i = (b_{\ell_i} + 1) \mod 2$ if $\ell_i \neq \bot$, and $b_i = 0$ otherwise.

Intuitively, for each value $v \in V_x$ occurring along w, the associated bit acts as a modulo 2 counter which is incremented at each visit of v along w. In the handling of the trigger rules under the weak minimal semantics, such a bit is used by ECA$^+$ event-clock constraints to reference the end-event of a token whose start-event (x, v, s) is the first occurrence of (x, v, s, b) for some $b \in \{0, 1\}$ after the current input position. The timed word w encodes the timeline for x of length $n+1$ given by $\pi = (v_0, \tau_1 - \tau_0) \dots (v_n, \tau_{n+1} - \tau_n)$. Note that since the duration of a token is not zero, we have that $\tau_{i+1} > \tau_i$ for all $0 \leq i \leq n$.

A *code for a multi-timeline for SV* is obtained by shuffling different timelines (one for each variable $x \in SV$), i.e., it is a non-empty timed word $w = (P_0, \tau_0) \cdots (P_n, \tau_n)$ over $2^{\mathcal{P}}$ such that:

- for all $x \in SV$, the timed word obtained from $(P_0 \cap \mathcal{P}_x, \tau_0) \cdots (P_n \cap \mathcal{P}_x, \tau_n)$ by removing the pairs $(\emptyset, \tau_i)$ is a code of a timeline for x;
- $\mathsf{init} \in P_0$, $\mathsf{init} \notin P_i$ for all $1 \leq i \leq n$, and $P_0 \cap \mathcal{P}_x \neq \emptyset$ for all $x \in SV$ (initialization).

Now, we show that the trigger rules in R under the weak minimal semantics can be handled by ECA$^+$ over $2^{\mathcal{P}}$. The start and end points of the chosen non-trigger tokens are mapped to last and next occurrences of propositions in $\mathcal{P}$ w.r.t. the current input position (trigger) of a multi-timeline encoding, while the atoms in the rules are mapped to ECA$^+$ event-clock constraints. Note that ECA$^+$ cannot express trigger-less rules since the semantics of these rules does not constraint the chosen punctual events to be closest as possible to a reference event. In the following, the *maximal constant* $K_{\mathcal{D}}$ of $\mathcal{D}$ is the greatest integer occurring in the atoms of R and in the constraint functions of the variables in SV.

Lemma 2. *One can construct in exponential time an ECA$^+$ $\mathcal{A}_\forall$ over $2^{\mathcal{P}}$ such that for each multi-timeline Π of SV and encoding w_Π of Π, w_Π is accepted by $\mathcal{L}_T(\mathcal{A}_\forall)$ iff Π satisfies the trigger rules in R under the weak minimal semantics. Moreover, $\mathcal{A}_\forall$ has a unique state, $O(N_a)$ atomic event-clock constraints, and maximal constant $O(K_{\mathcal{D}})$, where N_a is the overall number of atoms in the trigger rules in R.*

Proof. Let $\mathcal{R}$ be a trigger rule for SV with trigger $o_0[x_0 = v_0]$. We show how to build an ECA$^+$ $\mathcal{A}_{\mathcal{R}}$ over $2^{\mathcal{P}}$ satisfying Lemma 2 with $\mathcal{A}_\forall$ and R replaced with $\mathcal{A}_{\mathcal{R}}$ and $\{\mathcal{R}\}$, respectively. Hence, by applying the closure of ECA$^+$ under language intersection (Theorem 4), the general result where R contains an arbitrary number of trigger rules follows.

Essentially, given the encoding w_Π of a multi-timeline Π of SV, at each position i of w_Π where a trigger start-event $(x_0, v_0, \mathsf{s}, b)$ of $\mathcal{R}$ occurs for some $b \in \{0, 1\}$, the ECA$^+$ $\mathcal{A}_{\mathcal{R}}$ guesses an existential statement $\mathcal{E}$ of $\mathcal{R}$ and a weak minimal assignment λ_Π of Π w.r.t. o_0 such that $\lambda_\Pi(o_0) = (\Pi(x_0), i)$ and λ_Π is consistent with the bindings in the quantifiers $o[x = v]$ of $\mathcal{E}$ (*$\mathcal{E}$-binding consistency*), and checks by event-clock constraints that λ_Π satisfies the atoms in $\mathcal{E}$.

Fix an existential statement $\mathcal{E}$ of $\mathcal{R}$, and let $O(\mathcal{E})$ be the set of token names (existentially) quantified by $\mathcal{E}$. Let $\mathit{ref}(o_0) := (x_0, v_0)$ and for each $o \in O(\mathcal{E})$, let $\mathit{ref}(o)$ be the pair state variable/value referenced by o in the associated quantifier of $O(\mathcal{E})$. A weak minimal assignment w.r.t. o_0 restricted to the set of token names in $O(\mathcal{E}) \cup \{o_0\}$ can be specified by a tuple (P, C, F, g) (called *symbolic assignment of $\mathcal{E}$*), where P, C, and F are sets in $(O(\mathcal{E}) \cup \{o_0\}) \times \{\mathsf{s}, \mathsf{e}\}$, $g : O(\mathcal{E}) \cup \{o_0\} \mapsto \{0, 1\}$ assigns to each token name o in $O(\mathcal{E}) \cup \{o_0\}$ a bit in $\{0, 1\}$, and the following holds:

- The sets P, C, and F represent a partition of the set $(O(\mathcal{E}) \cup \{o_0\}) \times \{\mathsf{s}, \mathsf{e}\}$, i.e. $P \cup C \cup F = (O(\mathcal{E}) \cup \{o_0\}) \times \{\mathsf{s}, \mathsf{e}\}$ and P, C, and F are pairwise disjunct;
- $(o_0, \mathsf{s}) \in C$;
- for each $o \in O(\mathcal{E}) \cup \{o_0\}$, if $(o, \mathsf{e}) \in P$ (resp., $(o, \mathsf{e}) \in C$), then $(o, \mathsf{s}) \in P$.

Given an input symbol $a \in 2^{\mathcal{P}}$, the symbolic assignment (P, C, F, g) of $\mathcal{E}$ is *consistent with a* if the following holds:

- for all $(o, ev) \in C$, $(\mathit{ref}(o), ev, g(o)) \in a$;
- for all $o \in O(\mathcal{E})$, if $(\mathit{ref}(o), ev, b) \in a$ for some $b \in \{0, 1\}$ and $ev \in \{\mathsf{s}, \mathsf{e}\}$, then $(o, ev) \in C$ and $b = g(o)$.

Intuitively, given the encoding w_Π of a multi-timeline Π of SV and a $\mathcal{R}$-trigger position i of w_π (i.e., a position where the trigger start-event $(x_0, v_0, \mathsf{s}, b)$ of $\mathcal{R}$ occurs for some $b \in \{0, 1\}$), a symbolic assignment (P, C, F, g) of $\mathcal{E}$ consistent with the current input symbol $w_\Pi(i)$ encodes a weak minimal assignment λ_Π of Π w.r.t. o_0 such that $\lambda_\Pi(o_0) = (\Pi(x_0), i)$ and λ_Π is $\mathcal{E}$-binding consistent. Since the duration of a token is never zero (strict time monotonicity), accordingly to the weak minimal semantics, each start pair (o, s) in P (resp., in C, resp., in F) references the last previous occurrence (resp., current occurrence, resp., first next occurrence) of a start event of the form $(\mathit{ref}(o), \mathsf{s}, b)$ in w_Π w.r.t. position i, $b = g(o)$ (*start weak minimal requirement*), and the associated end pair (o, e) references the matching end event $(\mathit{ref}(o), \mathsf{e}, g(o))$. Note that we need to ensure that the start weak minimal requirement is fulfilled, the associated events exist and the start-event $(\mathit{ref}(o), \mathsf{s}, g(o))$ and the end-event $(\mathit{ref}(o), \mathsf{e}, g(o))$ for a name o in $O(\mathcal{E}) \cup \{o_0\}$ are associated to the same token. Additionally,

Table 2
Event-clock constraints for the difference atoms of $\mathcal{E}$ and the assignment (P, C, F, g) of $\mathcal{E}$.

P	C	F	$ev(o_2) - ev(o_1) \in I$
$(o_i, ev_i)_{i=1,2}$			$\overleftarrow{c}_{(ref(o_1),ev_1,g(o_1))} - \overleftarrow{c}_{(ref(o_2),ev_2,g(o_2))} \in I$
		$(o_i, ev_i)_{i=1,2}$	$\overrightarrow{c}_{(ref(o_2),ev_2,g(o_2))} - \overrightarrow{c}_{(ref(o_1),ev_1,g(o_1))} \in I$
(o_1, ev_1)		(o_2, ev_2)	$\overleftarrow{c}_{(ref(o_1),ev_1,g(o_1))} + \overrightarrow{c}_{(ref(o_2),ev_2,g(o_2))} \in I$
(o_2, ev_2)		(o_1, ev_1)	$\overleftarrow{c}_{(ref(o_2),ev_2,g(o_2))} + \overrightarrow{c}_{(ref(o_1),ev_1,g(o_1))} \in I \cap [0, 0]$
(o_1, ev_1)	(o_2, ev_2)		$\overleftarrow{c}_{(ref(o_1),ev_1,g(o_1))} \in I$
	(o_2, ev_2)	(o_1, ev_1)	$\overrightarrow{c}_{(ref(o_1),ev_1,g(o_1))} \in I \cap [0, 0]$
(o_2, ev_2)	(o_1, ev_1)		$\overleftarrow{c}_{(ref(o_2),ev_2,g(o_2))} \in I \cap [0, 0]$
	(o_1, ev_1)	(o_2, ev_2)	$\overrightarrow{c}_{(ref(o_2),ev_2,g(o_2))} \in I$
	$(o_i, ev_i)_{i=1,2}$		`true` if $0 \in I$, `false` otherwise

Table 3
Event-clock constraints for the point atoms of $\mathcal{E}$ and the assignment (P, C, F, g) of $\mathcal{E}$.

P	C	F	$ev(o) \in I$
(o, ev)			$\overleftarrow{c}_{\mathsf{init}} - \overleftarrow{c}_{(ref(o),ev,g(o))} \in I$
	$\mathsf{init} \notin a$	(o, ev)	$\overleftarrow{c}_{\mathsf{init}} + \overrightarrow{c}_{(ref(o),ev,g(o))} \in I$
	$\mathsf{init} \in a$	(o, ev)	$\overrightarrow{c}_{(ref(o),ev,g(o))} \in I$
	(o, ev), $\mathsf{init} \notin a$		$\overleftarrow{c}_{\mathsf{init}} \in I$
	(o, ev), $\mathsf{init} \in a$		`true` if $0 \in I$, `false` otherwise

for encoding the weak minimal semantics, we also need to ensure that for each start pair (o, s) in P (resp., in F), if the last previous occurrence (resp., first next occurrence) of the start event $(ref(o), \mathsf{s}, g(o))$ in w_Π w.r.t. position i has a time distance from the current position i greater than zero, then for each $b \in \{0, 1\}$, the first next occurrence (resp., last previous occurrence) of a start event (if any) of the form $(ref(o), \mathsf{s}, b)$ in w_Π w.r.t. position i also has a time distance from i greater than zero. For checking the previous requirements, we exploit the following ECA$^+$ event-clock constraint over $C_{\mathcal{P}}$ denoted by $\theta_c(P, C, F, g)$.

$$\bigwedge_{(o,\mathsf{s})\in P} \big(\overleftarrow{c}_{(ref(o),\mathsf{s},g(o)+1 \mod 2)} = \bot \vee \overleftarrow{c}_{(ref(o),\mathsf{s},g(o)+1 \mod 2)} - \overleftarrow{c}_{(ref(o),\mathsf{s},g(o))} > 0\big) \wedge$$

$$\bigwedge_{(o,\mathsf{s})\in F} \big(\overrightarrow{c}_{(ref(o),\mathsf{s},g(o)+1 \mod 2)} = \bot \vee \overrightarrow{c}_{(ref(o),\mathsf{s},g(o)+1 \mod 2)} - \overrightarrow{c}_{(ref(o),\mathsf{s},g(o))} > 0\big) \wedge$$

$$\bigwedge_{\{(o,\mathsf{s})\in P \mid (o,\mathsf{e})\notin P\}} \big(\overleftarrow{c}_{(ref(o),\mathsf{e},g(o))} = \bot \vee \overleftarrow{c}_{(ref(o),\mathsf{e},g(o))} - \overleftarrow{c}_{(ref(o),\mathsf{s},g(o))} > 0\big) \wedge$$

$$\bigwedge_{\{(o,\mathsf{e})\in F \mid (o,\mathsf{s})\notin F\}} \big(\overrightarrow{c}_{(ref(o),\mathsf{s},g(o))} = \bot \vee \overrightarrow{c}_{(ref(o),\mathsf{s},g(o))} - \overrightarrow{c}_{(ref(o),\mathsf{e},g(o))} > 0\big) \wedge$$

$$\bigwedge_{(o,\mathsf{s})\in P} \big(\overleftarrow{c}_{(ref(o),\mathsf{s},g(o))} = 0 \vee [\overleftarrow{c}_{(ref(o),\mathsf{s},g(o))} > 0 \wedge \bigwedge_{b\in\{0,1\}} (\overrightarrow{c}_{(ref(o),\mathsf{s},b)} = \bot \vee \overrightarrow{c}_{(ref(o),\mathsf{s},b)} > 0)]\big) \wedge$$

$$\bigwedge_{(o,\mathsf{s})\in F} \big(\overrightarrow{c}_{(ref(o),\mathsf{s},g(o))} = 0 \vee [\overrightarrow{c}_{(ref(o),\mathsf{s},g(o))} > 0 \wedge \bigwedge_{b\in\{0,1\}} (\overleftarrow{c}_{(ref(o),\mathsf{s},b)} = \bot \vee \overleftarrow{c}_{(ref(o),\mathsf{s},b)} > 0)]\big)$$

Note that in the above constraint, we exploit Boolean disjunction. The latter can be removed by using nondeterminism in the transition relation. We crucially observe that the module 2 counter used in the encoding of tokens and the above event-clock constraint also ensure that each end pair (o, e) in P (resp., in F) references the last previous occurrence (resp., first next occurrence) of the end event $(ref(o), e, g(o))$ in w_Π w.r.t. position i.

We now associate to the symbolic assignment (P, C, F, g) an event-clock constraint over $C_{\mathcal{P}}$, denoted by $\theta(\mathcal{E}, P, C, F, g)$, obtained from the body of the existential statement $\mathcal{E}$ by replacing each difference atom (resp., point atom) with the *generalized* atomic event-clock constraint indicated in Table 2 (resp., Table 3). Note that in the definition of such event-clock constraints, we use the Boolean constants `true` and `false`. Moreover, note that in Tables 2–3, we exploit *generalized* ECA$^+$ atomic event-clock constraints of the form $t \in I$ for an interval $I \in Intv$ which correspond to a conjunction $t \succ \ell \wedge t \prec \mathsf{u}$ of ECA$^+$ atomic constraints, where $\succ \in \{>, \geq\}$ and $\prec \in \{<, \leq\}$, $\ell \in \mathbb{N}$ and $\mathsf{u} \in \mathbb{N} \cup \{\infty\}$ (if $\mathsf{u} = \infty$, i.e. I is unbounded, then the upper-bound conjunct is not exploited).

The ECA$^+$ $\mathcal{A}_{\mathcal{R}}$ is then defined as follows. $\mathcal{A}_{\mathcal{R}}$ has a unique state q and for each input symbol $a \in 2^{\mathcal{P}}$, $q \xrightarrow{a,\varphi} q$ is a transition of $\mathcal{A}_{\mathcal{R}}$ iff either (i) $(ref(o_0), \mathsf{s}, b) \notin a$ for all $b \in \{0, 1\}$ and $\varphi = \texttt{true}$, or (ii) $(ref(o_0), \mathsf{s}, b) \in a$ for some $b \in \{0, 1\}$ and there exists an existential statement $\mathcal{E}$ of $\mathcal{R}$ and a symbolic assignment (P, C, F, g) of $\mathcal{E}$ consistent with the input symbol a such that $\varphi = \theta_c(P, C, F, g) \wedge \theta(\mathcal{E}, P, C, F, g)$. This concludes the proof of Lemma 2. □

For the trigger-less rules in R, the following result (Lemma 3) has been established in [19] for a slightly different encoding of the multi-timelines. The result can be easily adapted to the encoding proposed here.

Lemma 3. *One can construct in exponential time a TA $\mathcal{A}_\exists$ over $2^{\mathcal{P}}$ accepting the codes of the multi-timelines of SV which satisfy the* trigger-less *rules in R. Moreover, $\mathcal{A}_\exists$ has $2^{O(N_q+\sum_{x\in SV}|V_x|)}$ states, $O(|SV|+N_q)$ clocks, and maximal constant $O(K_{\mathcal{D}})$, where N_q is the overall number of quantifiers in the trigger-less rules of R.*

We can now establish the main result of this section.

Theorem 8. *Given a TP domain $\mathcal{D}=(SV,R)$, one can build in exponential time a TA $\mathcal{A}_{\mathcal{D}}$ with $2^{O(N+\sum_{x\in SV}|V_x|)}$ states, $O(N+|SV|)$ clocks, and maximal constant $O(K_{\mathcal{D}})$, where N is the overall number of quantifiers and atoms in the rules of R, such that $\mathcal{L}_T(\mathcal{A}_{\mathcal{D}})\neq\emptyset$* iff *there is a* weak minimal plan *of $\mathcal{D}$. Moreover, the weak minimal TP problem is* **PSPACE**-*complete.*

Proof. By Theorem 6 and Lemmata 2–3, the first part of Theorem 8 concerning the construction of the TA $\mathcal{A}_{\mathcal{D}}$ for the TP domain $\mathcal{D}$, directly follows (recall that TA are effectively and polynomial-time closed under language intersection). For the second part of Theorem 8, we recall that non-emptiness of a TA $\mathcal{A}$ can be solved by an **NPSPACE** search algorithm in the *region graph* of $\mathcal{A}$ which uses space logarithmic in the number of states of $\mathcal{A}$ and polynomial in the number of clocks and in the length of the encoding of the maximal constant of $\mathcal{A}$ [25]. Thus, since $\mathcal{A}_{\mathcal{D}}$ can be built on the fly, and the search in the region graph of $\mathcal{A}_{\mathcal{D}}$ can be done without explicitly constructing $\mathcal{A}_{\mathcal{D}}$, membership in **PSPACE** of the weak minimal TP problem follows.

Finally, we show **PSPACE**-hardness of the weak minimal TP problem by a polynomial time reduction from a domino-tiling problem for grids with rows of linear length [32]. Fix an instance $\mathcal{I}$ of such a problem which is a tuple $\mathcal{I}=(C,\Delta,n,\delta_{init},\delta_{final})$, where C is a finite set of colors, $\Delta\subseteq C^4$ is a set of tuples $(c_{down},c_{left},c_{up},c_{right})$ of four colors, called *domino-types*, $n>0$ is a natural number encoded in *unary*, and $\delta_{init},\delta_{final}\in\Delta$ are two distinguished domino-types (respectively, the initial and final domino-types). A *grid* of $\mathcal{I}$ is a mapping $f:[1,\ell]\times[1,n]\mapsto\Delta$ for some $\ell\in\mathbb{N}\setminus\{0\}$. Note that each row of a grid consists of n cells and each cell contains a domino type. A tiling *of* $\mathcal{I}$ is a grid $f:[1,\ell]\times[1,n]\mapsto\Delta$ satisfying the following additional requirements.

- two adjacent cells in a row have the same color on the shared edge, namely, for all $(i,j)\in[1,\ell]\times[1,n-1]$, $[f(i,j)]_{right}=[f(i,j+1)]_{left}$ (*row constraint*);
- two adjacent cells in a column have the same color on the shared edge, namely, for all $(i,j)\in[1,\ell-1]\times[1,n]$, $[f(i,j)]_{up}=[f(i+1,j)]_{down}$ (*column constraint*);
- $f(1,1)=\delta_{init}$ (*initialization*) and $f(\ell,n)=\delta_{final}$ (*acceptance*).

Without loss of generality, we can assume that if there is a tiling of $\mathcal{I}$, then there is a tiling $f:[1,\ell]\times[1,n]\mapsto\Delta$ of $\mathcal{I}$ satisfying the following additional requirements: the first cell of f is the unique containing δ_{init}, and the last cell of f is the unique containing δ_{final}.

It is well-known that checking the existence of a tiling of $\mathcal{I}$ is a **PSPACE**-complete problem [32]. We construct in polynomial time a TP instance $\mathcal{D}=(\{x_{\mathcal{I}}\},R_{\mathcal{I}})$ such that there exists a weak minimal plan of $\mathcal{D}$ iff there exists a tiling of $\mathcal{I}$. Hence, the result follows.

First, we define a suitable encoding of the tilings of $\mathcal{I}$ by timelines of the state variable $x_{\mathcal{I}}$. The finite domain V of the state variable $x_{\mathcal{I}}$ is given by $\Delta\times[1,n]\times\{0,1\}$.

We encode the row of a tiling by concatenating the codes of the row's cells starting from the first cell, and by marking the encoding with a tag which is a bit in $\{0,1\}$. Each cell is in turn encoded by a token of overall duration 1 which keeps track of the associated content and position along the row. Formally, a *row-code* is a timeline π of $x_{\mathcal{I}}$ of length n having the form $((\delta_1,1,b),d_1)\dots((\delta_n,n,b),d_n)$ such that the following holds:

- for all $i\in[1,n-1]$, $[\delta_i]_{right}=[\delta_{i+1}]_{left}$ (*row constraint*);
- for all $i\in[1,n]$, $d_i=1$ (*cell duration requirement*).

We say that $b\in\{0,1\}$ is the tag of the row-code π. A sequence ν of row-codes is *well-formed* if for each non-last row-code in ν with tag b, the next row-code in ν has tag $1-b$ for all $b\in\{0,1\}$. Tilings f are then encoded by timelines corresponding to well-formed concatenations of the codes of the rows of f starting from the first row. The following claim is straightforward.

Claim 1. *One can construct a state variable $x_{\mathcal{I}}=(V,T,D)$ such that the timelines of $x_{\mathcal{I}}$ whose first token has value $(\delta_{init},1,b)$ for some $b\in\{0,1\}$ correspond to the prefixes of well-formed concatenations of row-codes. Moreover, $(\delta_{init},i,b)\notin T(v)$ for all $v\in V$, $i\in[1,n]$, and $b\in\{0,1\}$, and $T((\delta_{final},n,b))=\emptyset$ for all $b\in\{0,1\}$.*

We now define the set $R_{\mathcal{I}}$ of synchronization rules which under the weak minimal semantics of trigger rules captures the timelines of $x_{\mathcal{I}}$ encoding tilings of $\mathcal{I}$. By Claim 1 and the assumption that the first cell (resp., last cell) of a tiling is the unique containing the domino type δ_{init} (resp., δ_{final}), in order to capture the well-formed concatenations of row-codes satisfying the initialization and halting requirements, it suffices to ensure that a timeline of $x_{\mathcal{I}}$ has a token with value $(\delta_{init}, 1, b)$ and a token with value (δ_{final}, n, b') for some $b, b' \in \{0, 1\}$. This can be expressed by the trigger-less rules $\top \rightarrow \bigvee_{b\in\{0,1\}} \exists o[x_{\mathcal{I}} = (\delta_{init}, 1, b)].\top$ and $\top \rightarrow \bigvee_{b\in\{0,1\}} \exists o[x_{\mathcal{I}} = (\delta_{final}, n, b)].\top$.

Finally, in order to capture the column constraint, we exploit trigger rules under the weak minimal semantics. Note that our encoding ensures that the difference $\mathsf{s}(tk') - \mathsf{s}(tk)$ of the start times of two tokens tk and tk' encodings cells of the same position and belonging to two consecutive row codes is always n. Thus, by Claim 1, the column constraint can be enforced by requiring that for each token tk with value (δ, i, b) (encoding the ith cell of a row), *either* (i) tk is followed by a token tk' with value (δ_{final}, n, b) such that $\mathsf{s}(tk') - \mathsf{s}(tk) = n - i$ (i.e., tk belongs to the last row-code), *or* (ii) tk is followed by a token tk' with value $(\delta', i, 1 - b)$ such that $\mathsf{s}(tk') - \mathsf{s}(tk) = n$ and $\delta_{up} = \delta'_{down}$ (column constraint). Thus, for each $(\delta, i, b) \in V \setminus \{(\delta_{final}, n, 0), (\delta_{final}, n, 1)\}$, we have the following trigger rule, where $V_\delta = \{\delta' \in \Delta \mid \delta_{up} = \delta'_{down}\}$:

$$\begin{aligned} o[x_{\mathcal{I}} = (\delta, i, b)] \rightarrow\ & \exists o'[x_{\mathcal{I}} = (\delta_{final}, n, b)].\ \mathsf{s}(o') - \mathsf{s}(o) \in [n - i, n - i] \vee \\ & \bigvee_{\delta'\in V_\delta} \exists o'[x_{\mathcal{I}} = (\delta', i, 1 - b)].\ \mathsf{s}(o') - \mathsf{s}(o) \in [n, n] \end{aligned}$$

Note that for the previous trigger rules, our encoding ensures that the weak minimal semantics coincides with the standard one. This concludes the proof of Theorem 8. □

6. Conclusions

In this paper, we addressed the TP problem in the dense-time setting. First, we negatively answered the question of decidability of the TP problem with future semantics, which was left open in [18]. Then, we introduced and investigated two novel semantics in the dense-time domain (the weak and strong minimal semantics) aimed at overcoming the structural restrictions on rule formats introduced in [18] to recover decidability. Surprisingly, we showed that, despite the apparently small difference between the two semantics, the strong minimal one leads to an undecidable TP problem, while the weak minimal one leads to a **PSPACE**-complete TP problem. In order to solve the weak minimal TP problem, we investigated two novel and strictly more expressive extensions of ECA which are interesting per se in the field of timed automata. As for future work, the most relevant issue we want to investigate is the expressiveness comparison between the TP framework with standard semantics and the TP framework with (weak or strong) minimal semantics. We expect the two frameworks to be incomparable from the expressiveness viewpoint. In fact, since the syntax of synchronization rules does not feature negation, it is not clear how to force minimality in the standard semantics. Conversely, it is not clear how to mimics the free token name assignments of the standard semantics in the framework enforcing the minimal semantics. As for more technical issues, we will study the strong minimal TP problem when just one or two state variables are used, whose decidability remains an open issue. Moreover, we aim at investigating the TP problem in the controllability setting, where the values of some variables are not under the system control, but depend on the environment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Undecidability of future TP problem

In this section we report the complete proof of Theorem 1. The result is stated by the following Proposition.

Proposition 1. *Given a Minsky 2-counter machine $M = (Q, q_{init}, q_{halt}, \Delta)$, one can construct (in polynomial time) a TP instance (domain) $P = (\{x_M\}, R_M)$ where the intervals in P are in $Intv_{(0,\infty)}$ such that M halts* iff *there exists a future plan for P.*

Proof. First, we define a suitable encoding of a computation of M as the untimed part of a timeline (i.e., neglecting tokens' durations and accounting only for their values) for x_M. For this, we exploit the finite set of symbols $V := V_{main} \cup V_{sec}$ corresponding to the finite domain of the state variable x_M. The set of *main* values V_{main} is the set of M-transitions, i.e. $V_{main} = \Delta$. The set of *secondary* values V_{sec} is defined as $V_{sec} := \Delta \times \{1, 2\} \times \{\#, beg, end\}$, where $\#$, beg, and end are three special symbols used as markers. Intuitively, in the encoding of an M-computation a main value keeps track of the transition used in the current step of the computation, while the set V_{sec} is used for encoding counter values.

For $c \in \{1, 2\}$, a *c-code for the main value $\delta \in \Delta$* is a finite word w_c over V_{sec} of the form $(\delta, c, beg) \cdot (\delta, c, \#)^h \cdot (\delta, c, end)$ for some $h \geq 0$ such that $h = 0$ if $op(\delta) = (\text{zero\_test}, c)$. The c-code w_c encodes the value for counter c given by h (or equivalently $|w_c| - 2$). Note that only the occurrences of the symbols $(\delta, c, \#)$ encode units in the value of counter c, while the symbol (δ, c, beg) (resp., (δ, c, end)) is only used as left (resp., right) marker in the encoding.

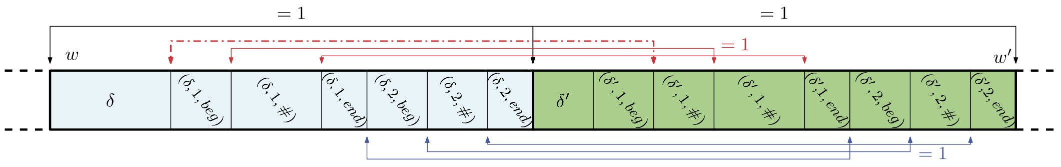


Fig. A.6. The figure shows two adjacent configuration-codes, w (highlighted in cyan) and w' (in green), the former for $\delta = (q, (\mathsf{inc}, 1), q') \in \Delta$ and the latter for $\delta' = (q', \dots) \in \Delta$; w encodes the M-configuration (q, ν) where $\nu(1) = \nu(2) = 1$, and w' the M-configuration (q', ν') where $\nu'(1) = 2$ and $\nu'(1) = 1$. The "1-Time distance between consecutive main values requirement" (represented by black lines with arrows) forces a token with a main value to be followed, after exactly one time instant, by another token with a main value. Since $op(\delta) = (\mathsf{inc}, 1)$, the value of counter 2 does not change in this computation step, and thus the values for counter 2 encoded by w and w' must be equal. To this aim the "equality requirement" (represented by blue lines with arrows) sets a one-to-one correspondence between pairs of tokens associated with counter 2 in w and w' (more precisely, a token tk with value $(\delta, 2, \_)$ in w is followed by a token tk' with value $(\delta', 2, \_)$ in w' such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ and $\mathsf{e}(tk') - \mathsf{e}(tk) = 1$). Finally, the "increment requirement" (red lines) performs the increment of counter 1 by doing something analogous to the previous case, but with a difference: the token tk' with value $(\delta', 1, \#)$ is in w' in the place where the token tk with value $(\delta, 1, beg)$ was in w (i.e., $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ and $\mathsf{e}(tk') - \mathsf{e}(tk) = 1$). The token tk'' with value $(\delta', 1, beg)$ is "anticipated", in such a way that $\mathsf{e}(tk'') - \mathsf{s}(tk) = 1$ (this is denoted by the dashed red line): the token with main value δ' in w' has a shorter duration than that with value δ in w, leaving space for tk'', so as to represent the unit added by δ to counter 1. Clearly density of the time domain plays a fundamental role here. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

A *configuration-code w for a main value $\delta \in \Delta$* is a finite word over V of the form $w = \delta \cdot w_1 \cdot w_2$ such that for each counter $c \in \{1, 2\}$, w_c is a c-code for the main value δ. The configuration-code w encodes the M-configuration $(from(\delta), \nu)$, where $\nu(c) = |w_c| - 2$ for all $c \in \{1, 2\}$. Note that if $op(\delta) = (\mathsf{zero\_test}, c)$, then $\nu(c) = 0$.

A *computation*-code is a non-empty sequence of configuration-codes $\pi = w_{\delta_1} \cdots w_{\delta_k}$, where for all $1 \le i \le k$, w_{δ_i} is a configuration-code with main value δ_i, and whenever $i < k$, it holds that $to(\delta_i) = from(\delta_{i+1})$. Note that by our assumptions $to(\delta_i) \neq q_{halt}$ for all $1 \le i < k$, and $\delta_j \neq \delta_{init}$ for all $1 < j \le k$. The computation-code π is *initial* if the first configuration-code w_{δ_1} has the main value δ_{init} and encodes the initial configuration, and it is *halting* if for the last configuration-code w_{δ_k} in π, it holds that $to(\delta_k) = q_{halt}$. For all $1 \le i \le k$, let (q_i, ν_i) be the M-configuration encoded by the configuration-code w_{δ_i} and $c_i = c(\delta_i)$. The computation-code π is *well-formed* if, additionally, for all $1 \le j < k$, the following holds:

- $\nu_{j+1}(c) = \nu_j(c)$ if either $c \neq c_j$ or $op(\delta_j) = (\mathsf{zero\_test}, c_j)$ (*equality requirement*);
- $\nu_{j+1}(c_j) = \nu_j(c_j) + 1$ if $op(\delta_j) = (\mathsf{inc}, c_j)$ (*increment requirement*);
- $\nu_{j+1}(c_j) = \nu_j(c_j) - 1$ if $op(\delta_j) = (\mathsf{dec}, c_j)$ (*decrement requirement*).

Clearly, M halts *iff* there exists an initial and halting well-formed computation-code.

Definition of x_M and R_M. We now define a state variable x_M and a set R_M of synchronization rules for x_M with intervals in $Intv_{(0,\infty)}$ such that the untimed part of every *future plan* of $P = (\{x_M\}, R_M)$ is an initial and halting well-formed computation-code. Thus, M halts if and only if there is a future plan of P.

Formally, variable x_M is given by $x_M = (V = V_{main} \cup V_{sec}, T, D)$, where for each $v \in V$, $D(v) = (0, \infty)$. Thus, we require that the duration of a token is always greater than zero (*strict time monotonicity*). The value transition function T of x_M ensures the following property.

Claim. *The untimed parts of the timelines for x_M whose first token has value δ_{init} correspond to the prefixes of initial computation-codes. Moreover, $\delta_{init} \notin T(v)$ for all $v \in V$.*

By construction, it is a trivial task to define T so that the previous requirement is fulfilled.

Let $V_{halt} = \{\delta \in \Delta \mid to(\delta) = q_{halt}\}$. By Claim above and the assumption that $from(\delta) \neq q_{halt}$ for each transition $\delta \in \Delta$, in order to enforce the initialization and halting requirements, it suffices to ensure that a timeline has a token with value δ_{init} and a token with value in V_{halt}. This is captured by the trigger-less rules $\top \rightarrow \exists o[x_M = \delta_{init}].\top$ and $\top \rightarrow \bigvee_{v \in V_{halt}} \exists o[x_M = v].\top$.

Finally, the crucial well-formedness requirement is captured by the trigger rules in R_M which express punctual time constraints.[3] We refer the reader to Fig. A.6, that gives an intuition on the properties enforced by the rules we are about to define. In particular, we essentially take advantage of the dense temporal domain to allow for the encoding of arbitrarily large values of counters in two time units.

Trigger rules for 1-Time distance between consecutive main values. We define non-simple trigger rules requiring that the overall duration of the sequence of tokens corresponding to a configuration-code amounts exactly to two time units. By Claim above, strict time monotonicity, and the halting requirement, it suffices to ensure that each token tk having a main value in $V_{main} \setminus V_{halt}$ is eventually followed by a token tk' such that tk' has a main value and $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ (this denotes—with a

[3] Such punctual constrains are expressed by pairs of conjoined atoms whose intervals are in $Intv_{(0,\infty)}$.

little abuse of notation—that the difference of start times is exactly 1). To this aim, for each $v \in V_{main} \setminus V_{halt}$, we write the non-simple trigger rule with intervals in $Intv_{(0,\infty)}$:

$$o[x_M = v] \rightarrow \bigvee_{u \in V_{main}} \exists o'[x_M = u].\, \mathsf{s}(o') - \mathsf{s}(o) \in [1, +\infty) \,\wedge\, \mathsf{s}(o') - \mathsf{s}(o) \in [0, 1].$$

Trigger rules for the equality requirement. In order to ensure the equality requirement, we exploit the fact that the end time of a token along a timeline corresponds to the start time of the next token (if any). Let $V_{sec}^{=}$ be the set of secondary states $(\delta, c, t) \in V_{sec}$ such that $to(\delta) \neq q_{halt}$, and either $c \neq c(\delta)$ or $op(\delta) = (\text{zero\_test}, c)$. Moreover, for a counter $c \in \{1, 2\}$ and a tag $t \in \{beg, \#, end\}$, let $V_c^t \subseteq V_{sec}$ be the set of secondary states given by $\Delta \times \{c\} \times \{t\}$. We require the following:

(*) each token tk with a $(V_c^t \cap V_{sec}^{=})$-value is eventually followed by a token tk' with a V_c^t-value such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ (i.e., the difference of start times is exactly 1). Moreover, if $t \neq end$, then $\mathsf{e}(tk') - \mathsf{e}(tk) = 1$ (i.e., the difference of end times is exactly 1).

Condition (*) is captured by the following non-simple trigger rules with intervals in $Intv_{(0,\infty)}$:

- for each $v \in V_c^t \cap V_{sec}^{=}$ and $t \neq end$,

$$\begin{aligned} o[x_M = v] \rightarrow \textstyle\bigvee_{u \in V_c^t} \exists o'[x_M = u].\ & \mathsf{s}(o') - \mathsf{s}(o) \in [1, +\infty) \,\wedge\, \mathsf{s}(o') - \mathsf{s}(o) \in [0, 1] \,\wedge \\ & \mathsf{e}(o') - \mathsf{e}(o) \in [1, +\infty) \,\wedge\, \mathsf{e}(o') - \mathsf{e}(o) \in [0, 1]; \end{aligned}$$

- for each $v \in V_c^{end} \cap V_{sec}^{=}$,

$$o[x_M = v] \rightarrow \bigvee_{u \in V_c^{end}} \exists o'[x_M = u].\, \mathsf{s}(o') - \mathsf{s}(o) \in [1, +\infty) \,\wedge\, \mathsf{s}(o') - \mathsf{s}(o) \in [0, 1].$$

We now show that Condition (*) together with strict time monotonicity and 1-Time distance between consecutive main values ensure the equality requirement. Let π be a timeline of x_M satisfying all the rules defined so far, w_δ and $w_{\delta'}$ two *adjacent* configuration-codes along π with w_δ preceding $w_{\delta'}$ (note that $to(\delta) \neq q_{halt}$), and $c \in \{1, 2\}$ a counter such that either $c \neq c(\delta)$ or $op(\delta) = (\text{zero\_test}, c)$. Let $tk_0 \cdots tk_{\ell+1}$ (resp., $tk'_0 \cdots tk'_{\ell'+1}$) be the sequence of tokens associated with the c-code of w_δ (resp., $w_{\delta'}$). We need to show that $\ell = \ell'$. By construction tk_0 and tk'_0 have value in V_c^{beg}, $tk_{\ell+1}$ and $tk'_{\ell'+1}$ have value in V_c^{end}, and for all $1 \leq i \leq \ell$ (resp., $1 \leq i' \leq \ell'$), tk_i has value in $V_c^{\#}$ (resp., $tk'_{i'}$ has value in $V_c^{\#}$). Then strict time monotonicity, 1-Time distance between consecutive main values, and Condition (*) guarantee the existence of an *injective* mapping $g : \{tk_0, \ldots, tk_{\ell+1}\} \rightarrow \{tk'_0, \ldots, tk'_{\ell'+1}\}$ such that $g(tk_0) = tk'_0$, $g(tk_{\ell+1}) = tk'_{\ell'+1}$, and for all $0 \leq i \leq \ell$, if $g(tk_i) = tk'_j$ (note that $j < \ell' + 1$), then $g(tk_{i+1}) = tk'_{j+1}$ (we recall that the end time of a token is equal to the start time of the next token along a timeline, if any). These properties ensure that g is *surjective* as well. Hence, g is a bijection and $\ell' = \ell$.

Trigger rules for the increment requirement. Let V_{sec}^{inc} be the set of secondary states $(\delta, c, t) \in V_{sec}$ such that $to(\delta) \neq q_{halt}$ and $op(\delta) = (\mathrm{inc}, c)$. By reasoning like in the case of the rules ensuring the equality requirement, in order to express the increment requirement, it suffices to enforce the following conditions for each counter $c \in \{1, 2\}$:

(i) each token tk with a $(V_c^{beg} \cap V_{sec}^{\mathrm{inc}})$-value is eventually followed by a token tk' with a V_c^{beg}-value such that $\mathsf{e}(tk') - \mathsf{s}(tk) = 1$ (i.e., the difference between the end time of token tk' and the start time of token tk is exactly 1);
(ii) for each $t \in \{beg, \#\}$, each token tk with a $(V_c^t \cap V_{sec}^{\mathrm{inc}})$-value is eventually followed by a token tk' with a $V_c^{\#}$-value such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ and $\mathsf{e}(tk') - \mathsf{e}(tk) = 1$ (i.e., the difference of start times and end times is exactly 1). Observe that the token with a $(V_c^{beg} \cap V_{sec}^{\mathrm{inc}})$-value is associated with a token with $V_c^{\#}$-value anyway;
(iii) each token tk with a $(V_c^{end} \cap V_{sec}^{\mathrm{inc}})$-value is eventually followed by a token tk' with a V_c^{end}-value such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ (i.e., the difference of start times is exactly 1);

Intuitively, if w and w' are two *adjacent* configuration-codes along a timeline of x_M, with w preceding w', (i) and (ii) force a token tk' with a $V_c^{\#}$-value in w' to "take the place" of the token tk with $(V_c^{beg} \cap V_{sec}^{\mathrm{inc}})$-value in w (i.e., they have the same start and end times). Moreover a token with V_c^{beg}-value must immediately precede tk' in w'.

These requirements can be expressed by non-simple trigger rules with intervals in $Intv_{(0,\infty)}$ similar to the ones defined for the equality requirement.

Trigger rules for the decrement requirement. For capturing the decrement requirement, it suffices to enforce the following conditions for each counter $c \in \{1, 2\}$, where V_{sec}^{dec} denotes the set of secondary states $(\delta, c, t) \in V_{sec}$ such that $to(\delta) \neq q_{halt}$ and $op(\delta) = (\mathrm{dec}, c)$:

(i) each token tk with a $(V_c^{beg} \cap V_{sec}^{\mathrm{dec}})$-value is eventually followed by a token tk' with a V_c^{beg}-value such that $\mathsf{s}(tk') - \mathsf{e}(tk) = 1$ (i.e., the difference between the start time of token tk' and the end time of token tk is exactly 1);

(ii) each token tk with a $(V_c^{\#} \cap V_{sec}^{\mathsf{dec}})$-value is eventually followed by a token tk' with a V_c^t-value where $t \in \{beg, \#\}$ such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ and $\mathsf{e}(tk') - \mathsf{e}(tk) = 1$ (i.e., the difference of start times and end times is exactly 1);

(iii) each token tk with a $(V_c^{end} \cap V_{sec}^{\mathsf{dec}})$-value is eventually followed by a token tk' with a V_c^{end}-value such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ (i.e., the difference of start times is exactly 1).

Analogously, (i) and (ii) produce an effect which is symmetric w.r.t. the case of increment.

Again, these requirements can be easily expressed by non-simple trigger rules with intervals in $Intv_{(0,\infty)}$ as done before for expressing the equality requirement.

By construction, the untimed part of a future plan of $P = (\{x_M\}, R_M)$ is an initial and halting well-formed computation-code. Vice versa, by exploiting denseness of the temporal domain, the existence of an initial and halting well-formed computation-code implies the existence of a future plan of P. This concludes the proof of Proposition 1. □

Appendix B. From ECA$^+$ to Timed Automata

Full details of the construction for Theorem 6

Theorem 6 *(From ECA$^+$ to TA). Given an ECA$^+$ $\mathcal{A}$ over $2^{\mathcal{P}}$, one can construct in exponential time a TA $\mathcal{A}'$ over $2^{\mathcal{P}}$ such that $\mathcal{L}_T(\mathcal{A}') = \mathcal{L}_T(\mathcal{A})$ and $K_{\mathcal{A}'} = K_{\mathcal{A}}$. Moreover, $\mathcal{A}'$ has $n \cdot 2^{O(p)}$ states and $O(p)$ clocks, where n is the number of $\mathcal{A}$-states and p is the number of event-clock atomic constraints used by $\mathcal{A}$.*

Proof. Let $\mathcal{A} = (2^{\mathcal{P}}, Q, Q_0, C_{\mathcal{P}}, \Delta, F)$ be an ECA$^+$ over $2^{\mathcal{P}}$. We now provide the formal definition of the TA $\mathcal{A}'$ accepting $\mathcal{L}_T(\mathcal{A})$ starting with some additional notation.

A *past event* of $\mathcal{A}$ is

- *either* a diagonal constraint of $\mathcal{A}$ over event-recording clocks,
- *or* a proposition $p \in \mathcal{P}$ s.t. the recording clock $\overleftarrow{c}_p$ occurs in some clock constraint of $\mathcal{A}$,
- or the element m_η or $m_{\hat{\eta}}$ for some sum constraint η of $\mathcal{A}$ which does not involve the special value $\perp$.

Intuitively, a past event $p \in P$ indicates that proposition p occurred in some previous input position, while a past event $\eta : \overleftarrow{c}_p - \overleftarrow{c}_{p'} \sim n_\perp$ indicates that the prefix w of the input read so far contains occurrences of both propositions p and p' and in case $n_\perp \neq \perp$, $\tau_{p'} - \tau_p \sim n_\perp$ holds, where τ_p and $\tau_{p'}$ are the timestamps associated with the last occurrences of p and p' in the prefix w, respectively. Finally, a past event m_η (resp., $m_{\hat{\eta}}$) for a sum constraint η denotes the η-mode (resp., $\hat{\eta}$-mode) in the handling of constraint η. A *past set* of $\mathcal{A}$ is a set P of past events such that for each sum constraint η of $\mathcal{A}$ which does not involve the special value $\perp$, $m_\eta \in P$ iff $m_{\hat{\eta}} \notin P$. An *obligation* of $\mathcal{A}$ is

- *either* a simple predicting constraint $\overrightarrow{c}_p \sim n_\perp$ of $\mathcal{A}$,
- *or* a sum clock constraint of $\mathcal{A}$,
- *or* an element of the form $\hat{\eta}$ for some sum constraint η of $\mathcal{A}$ which does not involve the special value $\perp$,
- *or* an element of the form η (resp., (η, p), resp., (η, p')), where η is a diagonal predicting constraint $\eta : \overrightarrow{c}_p - \overrightarrow{c}_{p'} \sim n_\perp$ of $\mathcal{A}$ involving the predictor clocks $\overrightarrow{c}_p$ and $\overrightarrow{c}_{p'}$.

An *obligation set of* $\mathcal{A}$ is a set of obligations O such that: (i) for each diagonal predicting constraint $\eta : \overrightarrow{c}_p - \overrightarrow{c}_{p'} \sim n_\perp$ of $\mathcal{A}$, it is not the case that both (η, p) and (η, p') are in O, and (ii) for each sum constraint η of $\mathcal{A}$ which does not involve the special value $\perp$, it is not the case that both η and $\hat{\eta}$ are in O.
Let C_{st} be the finite set consisting of the following standard clocks:

- the clock c_p for each recording clock $\overleftarrow{c}_p$ which occurs in some atomic event-clock constraint of $\mathcal{A}$ that does not involve the special value $\perp$.
- the clock c_η for each atomic constraint η of $\mathcal{A}$ which does not involve the spacial value $\perp$ such that either η is a simple predicting constraint, or a diagonal constraint over event-predicting clocks, or a sum constraint.
- the clock $\hat{c}_\eta$ for each diagonal predicting constraint $\eta : \overrightarrow{c}_p - \overrightarrow{c}_{p'} \geq 0$ of $\mathcal{A}$.
- the clock $\hat{c}_\eta$ for each sum constraint η of $\mathcal{A}$ which does not involve the special value $\perp$.

Finally, let Ψ be the finite set consisting of the following simple atomic constraints over C_{st}:

- the constraint $c_p \sim n_\perp$ for each simple recording constraint $\overleftarrow{c}_p \sim n_\perp$ (resp., diagonal recording constraint $\overleftarrow{c}_p - \overleftarrow{c}_{p'} \sim n_\perp$) of $\mathcal{A}$ with $n_\perp \neq \perp$;
- the constraint $c_{p'} = 0$ for each diagonal recording constraint $\overleftarrow{c}_p - \overleftarrow{c}_{p'} \geq 0$ of $\mathcal{A}$;
- the constraint $c_\eta \sim n_\perp$ for each sum constraint $\eta : \overleftarrow{c}_p + \overrightarrow{c}_{p'} \sim n_\perp$ (resp., simple predicting constraint $\eta : \overrightarrow{c}_p \sim n_\perp$, resp., diagonal predicting constraint $\eta : \overrightarrow{c}_p - \overrightarrow{c}_{p'} \sim n_\perp$) of $\mathcal{A}$ with $n_\perp \neq \perp$;

- the constraint $\hat{c}_\eta = 0$ for each diagonal predicting constraint $\eta : \overrightarrow{c}_p - \overrightarrow{c}_{p'} \geq 0$ of $\mathcal{A}$;
- the constraint $\hat{c}_\eta \sim n_\bot$ for each sum constraint $\eta : \overleftarrow{c}_p + \overrightarrow{c}_{p'} \sim n_\bot$ of $\mathcal{A}$ with $n_\bot \neq \bot$.

The TA $\mathcal{A}'$ is formally given by $\mathcal{A}' = (2^{\mathcal{P}}, Q', Q'_0, C_{st}, \Delta', F')$. The set Q' of states consists of the triples of the form (q, P, O) such that q is a state of $\mathcal{A}$, P is a past set of $\mathcal{A}$, and O is an obligation set of $\mathcal{A}$. The set Q'_0 of initial states consists of the states of the form $(q_0, P_0, \emptyset)$ such that $q_0 \in Q_0$ and P_0 is the set of elements of the form m_η (η-mode) where η is a sum constraint of $\mathcal{A}$ which does not involve the value $\bot$ (initially there are neither obligations nor past events associated with propositions in P and diagonal recording constraints). The set F' of accepting states consists of the states of the form (q, P, O) such that $q \in F$ and O contains only obligations associated with atomic constraints involving the $\bot$ value.

Finally, the transition relation Δ' of the TA $\mathcal{A}'$ is defined as follows. For each transition $q \xrightarrow{a,\theta} q'$ of the ECA$^+$ $\mathcal{A}$, we have in the TA $\mathcal{A}'$ the transitions of the form $(q, P, O) \xrightarrow{a,\theta',Res} (q', P', O')$, where the sets of past events P and P', the sets of obligations O and O', the reset set *Res*, and the clock constraint θ' satisfy the following requirements:

- The clock constraint θ' has as conjuncts only simple atomic constraints in Ψ.
- For each past event p in $\mathcal{P}$, (i) $p \in P'$ iff either $p \in a$ or $p \in P$, and (ii) $c_p \in Res$ iff $p \in a$.
- For each simple recording clock constraint $\eta : \overleftarrow{c}_p \sim n_\bot$ of $\mathcal{A}$:
  - if η is a conjunct of θ, then $p \in P$ iff $n_\bot \neq \bot$;
  - if $n_\bot \neq \bot$ and η is a conjunct of θ, then $c_p \sim n_\bot$ is a conjunct of θ'.
- For each diagonal recording clock constraint $\eta : \overleftarrow{c}_p - \overleftarrow{c}_{p'} \sim n_\bot$ of $\mathcal{A}$:
  - if η is a conjunct of θ, then $\eta \in P$ iff $n_\bot \neq \bot$;
  - if $n_\bot = \bot$, then $\eta \in P'$ iff either $\eta \in P$ or $p, p' \in a \cup P$;
  - if $n_\bot \neq \bot$, then $\eta \in P'$ iff
    - either $\eta \in P$ and $p, p' \notin a$,
    - or $p \in P \setminus a$, $p' \in a$, and $c_p \sim n_\bot$ is a conjunct of θ',
    - or $p, p' \in a$ and $0 \sim n_\bot$ holds,
    - or $p' \in P \setminus a$, $p \in a$, and *either* (i) $\sim$ is $\leq$, *or* (ii) $\sim$ is $<$ and $n_\bot \neq 0$, *or* (iii) $\sim$ is $<$, $n_\bot = 0$, and $c_{p'} > 0$ is a conjunct of θ', *or* (iv) $\sim$ is $\geq$, $n_\bot = 0$, and $c_{p'} = 0$ is a conjunct of θ'.
- For each simple predicting clock constraint $\eta : \overrightarrow{c}_p \sim n_\bot$ of $\mathcal{A}$:
  - $\eta \in O'$ iff *either* η is a conjunct of θ, *or* $\eta \in O$ and $p \notin a$;
  - if $n_\bot \neq \bot$, then $c_\eta \in Res$ iff η is a conjunct of θ and *either* $\eta \notin O$ *or* $\sim \in \{>, \geq\}$;
  - if $n_\bot \neq \bot$, then $c_\eta \sim n_\bot$ is a conjunct of θ' iff $\eta \in O$ and $p \in a$;
  - if $n_\bot = \bot$, then either $\eta \notin O$ or $p \notin a$.
- For each sum clock constraint $\eta : \overleftarrow{c}_p + \overrightarrow{c}_{p'} \sim \bot$:
  - $\eta \in O'$ iff $p \in P$ and *either* η is a conjunct of θ, *or* $\eta \in O$ and $p' \notin a$;
  - either $\eta \notin O$ or $p' \notin a$.
- For each sum clock constraint $\eta : \overleftarrow{c}_p + \overrightarrow{c}_{p'} \sim n$ of $\mathcal{A}$ such that $n \neq \bot$:
  - if η occurs in θ, then $p \in P$;
  - $P \cap \{m_\eta, m_{\hat{\eta}}\} \neq P' \cap \{m_\eta, m_{\hat{\eta}}\}$ iff $p \in a$, $p' \notin a$, and *either* (i) $m_\eta \in P$ and $\eta \in O$, *or* (ii) $m_{\hat{\eta}} \in P$ and $\hat{\eta} \in O$;
  - $c_\eta \in Res$ (resp., $c_{\hat{\eta}} \in$ *Res*) iff $p \in a$ and *either* $\{\eta, \hat{\eta}\} \cap O = \emptyset$ *or* $m_\eta \in P'$ (resp., $m_{\hat{\eta}} \in P'$);
  - $c_\eta \sim n_\bot$ (resp., $c_{\hat{\eta}} \sim n_\bot$) is a conjunct of θ' iff $\eta \in O$ (resp., $\hat{\eta} \in O$) and $p' \in a$;
  - if $\sim \in \{<, \leq\}$ (upper bound), then $\eta \in O'$ iff $p \in P$ and *either* (i) $\eta \in O$ and $p' \notin a$, *or* (ii) η is a conjunct of θ, $m_\eta \in P$ (η-mode), and ($p' \notin a$ implies that $\{\eta, \hat{\eta}\} \cap O = \emptyset$);
  - if $\sim \in \{<, \leq\}$ (upper bound), $\hat{\eta} \in O'$ iff $p \in P$ and *either* (i) $\hat{\eta} \in O$ and $p' \notin a$, *or* (ii) η is a conjunct of θ, $m_{\hat{\eta}} \in P$ ($\hat{\eta}$-mode), and ($p' \notin a$ implies that $\{\eta, \hat{\eta}\} \cap O = \emptyset$);
  - if $\sim \in \{>, \geq\}$ (lower bound), then $\eta \in O'$ iff $p \in P$ and *either* (i) $\eta \in O$, $p' \notin a$, and θ is *not* a conjunct of θ, *or* (ii) η is a conjunct of θ and $m_\eta \in P$;
  - if $\sim \in \{>, \geq\}$ (lower bound), then $\hat{\eta} \in O'$ iff $p \in P$ and *either* (i) $\hat{\eta} \in O$, $p' \notin a$, and θ is *not* a conjunct of θ, *or* (ii) η is a conjunct of θ and $m_{\hat{\eta}} \in P$.
- For each diagonal predicting clock constraint $\eta : \overrightarrow{c}_p - \overrightarrow{c}_{p'} \sim n_\bot$ of $\mathcal{A}$:
  - $\eta \in O'$ iff *either* η is a conjunct of θ, *or* $\eta \in O$ and $p, p' \notin a$;
  - $(\eta, p) \in O'$ iff *either* (i) $(\eta, p) \in O$ and $p' \notin a$, or (ii) $\eta \in O$, $p \in a$, and $p' \notin a$;
  - $(\eta, p') \in O'$ iff *either* (i) $(\eta, p') \in O$ and $p \notin a$, or (ii) $\eta \in O$, $p' \in a$, and $p \notin a$;
  - if $n_\bot \neq \bot$, then $c_\eta \in Res$ iff $p' \in a$, $p \notin a$, $\eta \in O$, and *either* $(\eta, p') \notin O$ *or* $\sim \in \{>, \geq\}$;
  - if $n_\bot = 0$ and $\sim$ is $\geq$, then $\hat{c}_\eta \in Res$ iff $p \in a$, $p' \notin a$, $\eta \in O$, and $(\eta, p) \notin O$;
  - if $n_\bot = 0$ and $\sim$ is $<$, then $\hat{c}_\eta \in Res$ iff $p \in a$, $p' \notin a$, and $\{\eta, (\eta, p)\} \cap O \neq \emptyset$;
  - if $n_\bot \neq \bot$, $\eta \in O$, and $\{p, p'\} \subseteq a$ then $0 \sim n_\bot$ holds;
  - if $n_\bot \neq \bot$, $(\eta, p') \in O$ and $p \in a$, then $c_\eta \sim n_\bot$ is a conjunct of θ';
  - if $n_\bot \neq \bot$, $(\eta, p) \in O$ and $p' \in a$, then *either* (i) $\sim$ is $\leq$, *or* (ii) $\sim$ is $<$ and $n_\bot \neq 0$, *or* (iii) $\sim$ is $<$, $n_\bot = 0$, and $\hat{c}_\eta > 0$ is a conjunct of θ', *or* (iv) $\sim$ is $\geq$, $n_\bot = 0$, and $\hat{c}_\eta = 0$ is a conjunct of θ';

– if $n_\perp = \perp$ and $\eta \in O$ (resp., $(\eta, p) \in O$, resp., $(\eta, p') \in O$), then $\{p, p'\} \nsubseteq a$ (resp., $p' \notin a$, resp., $p \notin a$).

This concludes the proof of Theorem 6. □

References

[1] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, Undecidability of future timeline-based planning over dense temporal domains?, in: G. Cordasco, L. Gargano, A.A. Rescigno (Eds.), Proceedings of the 21st Italian Conference on Theoretical Computer Science, Ischia, Italy, September 14-16, 2020, in: CEUR Workshop Proceedings, CEUR-WS.org, vol. 2756, 2020, pp. 155–166, http://ceur-ws.org/Vol-2756/paper_15.pdf.

[2] L. Bozzelli, A. Montanari, A. Peron, Taming the complexity of timeline-based planning over dense temporal domains, in: A. Chattopadhyay, P. Gastin (Eds.), 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS, 2019, December 11-13, 2019, Bombay, India, in: LIPIcs, vol. 150, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 34:1–34:14.

[3] M. Fox, D. Long, PDDL2.1: an extension to PDDL for expressing temporal planning domains, J. Artif. Intell. Res. 20 (2003) 61–124, https://doi.org/10.1613/jair.1129.

[4] J. Rintanen, Complexity of concurrent temporal planning, in: Proc. of the 17th ICAPS, AAAI, 2007, pp. 280–287, http://www.aaai.org/Library/ICAPS/2007/icaps07-036.php.

[5] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, E. Tronci, Flexible timeline-based plan verification, in: Proc. of the 32nd KI, in: LNCS, vol. 5803, Springer, 2009, pp. 49–56.

[6] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, E. Tronci, Analyzing flexible timeline-based plans, in: Proc. of the 19th ECAI, in: Frontiers in Artificial Intelligence and Applications, vol. 215, IOS Press, 2010, pp. 471–476.

[7] M. Cialdea Mayer, A. Orlandini, An executable semantics of flexible plans in terms of timed game automata, in: Proc. of the 22nd TIME, IEEE Computer Society, 2015, pp. 160–169.

[8] M. Cialdea Mayer, A. Orlandini, A. Ubrico, A formal account of planning with flexible timelines, in: Proc. of the 21st TIME, IEEE Computer Society, 2014, pp. 37–46.

[9] M. Cialdea Mayer, A. Orlandini, A. Umbrico, Planning and execution with flexible timelines: a formal account, Acta Inform. 53 (6–8) (2016) 649–680, https://doi.org/10.1007/s00236-015-0252-z.

[10] A. Cimatti, A. Micheli, M. Roveri, Timelines with temporal uncertainty, in: Proc. of the 27th AAAI, 2013, http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6319.

[11] R. Alur, T.A. Henzinger, A really temporal logic, J. ACM 41 (1) (1994) 181–204, https://doi.org/10.1145/174644.174651.

[12] J. Barreiro, M. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, J. Ong, E. Remolina, T. Smith, D. Smith, EUROPA: a platform for AI planning, scheduling, constraint programming, and optimization, in: Proc. of the 4th ICKEPS, 2012.

[13] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, N. Policella, An innovative product for space mission planning: an a posteriori evaluation, in: Proc. of the 17th ICAPS, 2007, pp. 57–64, http://www.aaai.org/Library/ICAPS/2007/icaps07-008.php.

[14] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, S. Frye, Timeline-based space operations scheduling with external constraints, in: Proc. of the 20th ICAPS, AAAI, 2010, pp. 34–41, http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1432.

[15] J. Frank, A. Jónsson, Constraint-based attribute and interval planning, Constraints 8 (4) (2003) 339–364, https://doi.org/10.1023/A:1025842019552.

[16] A.K. Jónsson, P.H. Morris, N. Muscettola, K. Rajan, B.D. Smith, Planning in interplanetary space: theory and practice, in: Proc. of the 5th AIPS, AAAI, 2000, pp. 177–186, http://www.aaai.org/Library/AIPS/2000/aips00-019.php.

[17] N. Muscettola, HSTS: integrating planning and scheduling, in: Intelligent Scheduling, Morgan Kaufmann, 1994, pp. 169–212.

[18] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, G.J. Woeginger, Timeline-based planning over dense temporal domains, Theor. Comput. Sci. 813 (2020) 305–326, https://doi.org/10.1016/j.tcs.2019.12.030.

[19] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, Complexity of timeline-based planning over dense temporal domains: exploring the middle ground, in: Proc. of the 9th GandALF 2018, in: EPTCS, vol. 277, 2018, pp. 191–205.

[20] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, Decidability and complexity of timeline-based planning over dense temporal domains, in: Proc. of the 16th KR, AAAI Press, 2018, pp. 627–628, https://aaai.org/ocs/index.php/KR/KR18/paper/view/17995.

[21] N. Gigante, A. Montanari, M. Cialdea Mayer, A. Orlandini, Timelines are expressive enough to capture action-based temporal planning, in: Proc. of the 23rd TIME, IEEE Computer Society, 2016, pp. 100–109.

[22] N. Gigante, A. Montanari, M. Cialdea Mayer, A. Orlandini, Complexity of timeline-based planning, in: Proc. of the 27th ICAPS, AAAI Press, 2017, pp. 116–124, https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15758.

[23] J. Ouaknine, J. Worrell, On metric temporal logic and faulty Turing machines, in: Proc. of the 9th FOSSACS, in: LNCS, vol. 3921, Springer, 2006, pp. 217–230.

[24] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, G.J. Woeginger, Timeline-based planning over dense temporal domains with trigger-less rules is NP-complete, in: Proc. of the 19th ICTCS, in: CEUR Workshop Proceedings, vol. 2243, 2018, pp. 116–127, http://ceur-ws.org/Vol-2243/paper11.pdf.

[25] R. Alur, D.L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (2) (1994) 183–235, https://doi.org/10.1016/0304-3975(94)90010-8.

[26] J. Ouaknine, J. Worrell, On the decidability and complexity of metric temporal logic over finite words, Log. Methods Comput. Sci. 3 (1) (2007), https://doi.org/10.2168/LMCS-3(1:8)2007.

[27] R. Alur, T. Feder, T.A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1) (1996) 116–146, https://doi.org/10.1145/227595.227602.

[28] R. Alur, L. Fix, T.A. Henzinger, Event-clock automata: a determinizable class of timed automata, Theor. Comput. Sci. 211 (1–2) (1999) 253–273, https://doi.org/10.1016/S0304-3975(97)00173-4.

[29] G. Geeraerts, J. Raskin, N. Sznajder, Event clock automata: from theory to practice, in: Proc. of the 9th FORMATS, in: LNCS, vol. 6919, Springer, 2011, pp. 209–224.

[30] M.L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, Inc., 1967.

[31] J. Raskin, P. Schobbens, The logic of event clocks - decidability, complexity and expressiveness, J. Autom. Lang. Comb. 4 (3) (1999) 247–286.

[32] D. Harel, Algorithmics: The Spirit of Computing, 2nd edition, Wesley, 1992.