Timeline-based planning over dense temporal domains

Laura Bozzelli [a], Alberto Molinari [b], Angelo Montanari [b], Adriano Peron [a,*], Gerhard Woeginger [c]

[a] *University of Napoli "Federico II", Italy*
[b] *University of Udine, Italy*
[c] *RWTH Aachen University, Germany*

ARTICLE INFO

Keywords:
Planning
Timelines
Metric temporal logic
Timed automata

ABSTRACT

Planning is one of the most studied problems in computer science. In this paper, we focus on the timeline-based approach, where the domain is modeled by a set of independent, but interacting, components, each one represented by a number of state variables, whose behavior over time (timelines) is governed by a set of temporal constraints (transition functions and synchronization rules). Whereas the time domain is usually assumed to be discrete, here we address decidability and complexity issues for timeline-based planning (TP) over dense time.

We first prove that dense TP is undecidable in the general case; then, we show that decidability can be recovered by restricting to synchronization rules with a suitable future semantics. More "tractable" settings can be obtained by additionally constraining the form of intervals used in rules: **EXPSPACE**-completeness is obtained by avoiding singular intervals, and **PSPACE**-completeness by admitting only intervals of the forms $[0, a]$ and $[b, +\infty[$. Finally, **NP**-completeness can be proved for dense TP with purely existential rules only.

1. Introduction

Timeline-based planning [20] (TP for short) represents an alternative to classic action-based planning. The latter aims at determining a sequence of actions that, given the initial state of the world and a goal, transforms, step by step, the state of the world until a state satisfying the goal is reached. TP can be viewed as a more declarative approach, that focuses on what has to happen in order to satisfy the goal rather than on what an agent has to do. In TP, the planning domain is modeled as a set of independent, but interacting, components, each one consisting of a number of *state variables*. The evolution of the values of state variables over time is described by means of a set of *timelines* (sequences of time intervals called *tokens*), and it is governed by a set of transition functions, one for each state variable, and a set of synchronization rules, that constrain the temporal relations among (the values of) state variables.

TP has been successfully exploited in a number of application domains, including space missions, constraint solving, and activity scheduling (see, e.g., [4,8,9,12,16,20]), but a systematic study of its expressiveness and complexity has been undertaken only very recently. The temporal domain is commonly assumed to be *discrete*. In [13], Gigante et al. showed

* Corresponding author.
E-mail addresses: lr.bozzelli@gmail.com (L. Bozzelli), molinari.alberto@gmail.com (A. Molinari), angelo.montanari@uniud.it (A. Montanari),

Table 1
Decidability and complexity of restrictions of the TP problem.

	TP problem	Future TP problem
Unrestricted	Undecidable	(Undecidable?) Non-primitive recursive-hard
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 only	**NP**-complete	//

that TP with bounded temporal relations and token durations, and no temporal horizon, is **EXPSPACE**-complete and expressive enough to capture action-based temporal planning. Later, they proved that **EXPSPACE**-completeness still holds for TP with unbounded interval relations, and that the problem becomes **NEXPTIME**-complete if an upper bound to the temporal horizon is added [14].

In this paper, we study TP over a *dense temporal domain*, without resorting to any form of discretization, which is the commonly adopted "solution". The reason why we assume the temporal domain to be dense is, basically, to increase expressiveness: dense time allows one to abstract away unnecessary details, often artificially added for the necessity of discretizing time, and to suitably represent actions with duration, accomplishments, and temporal aggregates.

The first result we establish is negative: we prove that TP over dense time, in its general formulation, is *undecidable*. So, we study suitable restrictions on the TP problem that make it possible to recover decidability. In fact, we do not only illustrate how decidability can be achieved, but we also show how to guarantee reasonable computational complexities, which are important for the concrete application of TP, by constraining the structure of synchronization rules.

In the general case, a synchronization rule allows a universal quantification over the tokens of a timeline (such a quantification is called *trigger*). When a token is "selected" by a trigger, the rule allows one to compare tokens of the timelines both preceding (past) and following (future) the triggered token (trigger for short). The first restriction we consider limits the comparison to tokens following the triggered one (*future semantics* of trigger rules). The second imposes non-trigger tokens to appear at most once in the constraints set by the rule (*simple* trigger rules). Better complexity results can be obtained by restricting also the type of *intervals* used in rules in order to compare tokens.

Table 1 summarizes the decidability and complexity results proved in the following sections: we will consider suitable mixes of restrictions on TP involving trigger rules with future semantics, simple trigger rules, and intervals in atoms (of trigger rules) which are non-singular (a *singular* interval is an interval of the form $[a, a]$), or unbounded/left-closed with left endpoint 0 (the latter intervals are denoted by $Intv_{(0,\infty)}$).

Organization of the paper In Section 2, we introduce the TP framework. Then, in Section 3, we prove that TP is *undecidable* in the general case, by a reduction from the *halting problem for Minsky 2-counter machines*. In the last part of the section, we discuss the *non-primitive recursive-hardness* of TP under the future semantics of trigger rules (such a result is formally demonstrated in Appendix A). Next, in Section 4, we first show that future TP with simple trigger rules is *decidable* (in non-primitive recursive time), and then we prove membership in **EXPSPACE** (resp., **PSPACE**) under the additional restriction to *non-singular intervals* (resp., intervals in $Intv_{(0,\infty)}$). Matching complexity lower bounds for the last two restrictions are given in C. Finally, in Section 5, we outline an **NP** algorithm for TP with *trigger-less rules only* (which have a purely existential form disallowing universal quantification/trigger) stemming from the results of the previous sections. With a trivial hardness proof, we also show TP with trigger-less rules to be **NP**-complete. This paper is an extended and revised version of [5–7].

2. The TP problem

Let $\mathbb{N}$ be the set of natural numbers (including 0) and $\mathbb{R}_+$ be the set of non-negative real numbers. Let $Intv$ denote the set of intervals of $\mathbb{R}_+$ (both open or closed) whose endpoints are in $\mathbb{N} \cup \{\infty\}$ (notice that $Intv$ includes *singular intervals* $[a, a]$, with $a \in \mathbb{N}$). Moreover, let $Intv_{(0,\infty)}$ be the set of non-singular intervals $I \in Intv$ such that either I is unbounded, or I is left-closed with left endpoint 0. Intervals in $Intv_{(0,\infty)}$ can be represented by expressions of the form $\sim n$, for some $n \in \mathbb{N}$ and $\sim \in \{<, \leq, >, \geq\}$.

We now introduce the basic notions of the TP framework [10,13]. The domain knowledge is encoded by a set of state variables, whose behavior over time is described by transition functions and synchronization rules.

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 state 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 (or duration) function* that maps each $v \in V_x$ to an interval of $Intv$.

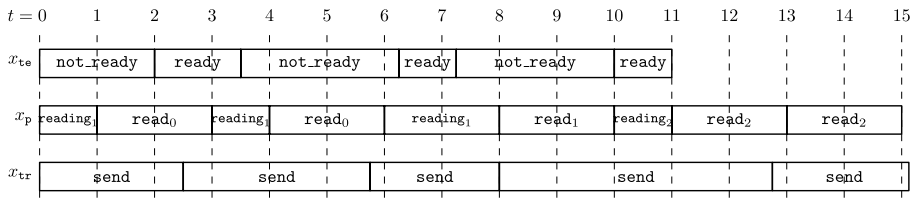

Fig. 1. A multi-timeline for Example 2 where the timeline for variable $x_{\mathtt{te}}$ is the sequence of tokens $(\mathtt{not\_ready}, 2), (\mathtt{ready}, 1.5), (\mathtt{not\_ready}, 2.7), (\mathtt{ready}, 1), (\mathtt{not\_ready}, 2.8), (\mathtt{ready}, 1)$.

A *token* for a state variable x is a pair (v, d) consisting of a value $v \in V_x$ and a duration $d \in \mathbb{R}_+$ such that $d \in D_x(v)$. Intuitively, a state variable represents a component of a system and a token for x represents an interval of time where the component takes the value v. If the interval is singular (i.e., it has duration 0), the token represents an instantaneous activity of the component (the state value is entered and exited instantaneously). In order to identify the variable a token refers to, we shall often denote (v, d) as (x, v, d). The behavior of a state variable x is specified by a *timeline*, which is a non-empty sequence of tokens $\pi = (v_0, d_0) \cdots (v_n, d_n)$ consistent with the value transition function T_x, namely, such that $v_{i+1} \in T_x(v_i)$ for all $0 \leq i < n$. The *start time* $\mathsf{s}(\pi, i)$ and the *end time* $\mathsf{e}(\pi, i)$ of the i-th token of the timeline π are respectively defined as follows:

$$\mathsf{s}(\pi, i) = 0 \text{ if } i = 0, \text{ and } \mathsf{s}(\pi, i) = \sum_{h=0}^{i-1} d_h \text{ otherwise;} \qquad \mathsf{e}(\pi, i) = \sum_{h=0}^{i} d_h.$$

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

Example 2. Let us consider a system consisting of three components (*temperature sensor*, *processing unit*, and *data transmission unit*) respectively modeled by the state variables $x_\ell = (V_{x_\ell}, T_{x_\ell}, D_{x_\ell})$, with $\ell \in \{te, p, tr\}$, where

- $V_{x_{\mathtt{te}}} = \{\mathtt{ready}, \mathtt{not\_ready}\}$, $T_{x_{\mathtt{te}}}(\mathtt{ready}) = \{\mathtt{not\_ready}\}$, $T_{x_{\mathtt{te}}}(\mathtt{not\_ready}) = \{\mathtt{ready}\}$, $D_{x_{\mathtt{te}}}(\mathtt{ready}) = [1, 2]$, $D_{x_{\mathtt{te}}}(\mathtt{not\_ready}) = [2, 3]$,
- $V_{x_{\mathtt{p}}} = \{\mathtt{reading}_1, \mathtt{reading}_2, \mathtt{read}_0, \mathtt{read}_1, \mathtt{read}_2\}$, $T_{x_{\mathtt{p}}}(\mathtt{reading}_1) = \{\mathtt{read}_0, \mathtt{read}_1\}$, $T_{x_{\mathtt{p}}}(\mathtt{read}_0) = \{\mathtt{reading}_1\}$, $T_{x_{\mathtt{p}}}(\mathtt{read}_1) = \{\mathtt{reading}_2\}$, $T_{x_{\mathtt{p}}}(\mathtt{reading}_2) = \{\mathtt{read}_1, \mathtt{read}_2\}$, $T_{x_{\mathtt{p}}}(\mathtt{read}_2) = \{\mathtt{read}_2\}$, $D_{x_{\mathtt{p}}}(\mathtt{reading}_1) = D_{x_{\mathtt{p}}}(\mathtt{reading}_2) = [1, 2]$, $D_{x_{\mathtt{p}}}(\mathtt{read}_0) = D_{x_{\mathtt{p}}}(\mathtt{read}_1) = D_{x_{\mathtt{p}}}(\mathtt{read}_2) = [2, 3]$, and
- $V_{x_{\mathtt{tr}}} = \{\mathtt{send}\}$, $T_{x_{\mathtt{tr}}}(\mathtt{send}) = \{\mathtt{send}\}$, $D_{x_{\mathtt{tr}}}(\mathtt{send}) = [2, 5]$.

The temperature sensor swaps between the state `ready`, where it senses the temperature of the environment, and the state `not_ready`, where it *possibly* sends the temperature value to the processing unit. The processing unit receives *two* temperature samples from the sensor, and sends the average value to the data transmission unit: in state $\mathtt{read}_i$, with $i = 0, 1, 2$, i samples have been already read; in state $\mathtt{reading}_j$, with $j = 1, 2$, it is attempting to read the j-th sample. A multi-timeline for the described system is reported in Fig. 1.

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 (time-point constraints) and on the difference between start/end times of tokens (interval constraints). The synchronization rules exploit an alphabet $\Sigma = \{o, o_0, o_1, o_2, \ldots\}$ of *token names* to refer to the tokens along a multi-timeline, and are based on the notions of *atom* and *existential statement*.

Atom. An *atom* ρ is either a clause of the form $o_1 \leq_I^{e_1,e_2} o_2$ (*interval atom*), or a clause of the forms $o_1 \leq_I^{e_1} n$ or $n \leq_I^{e_1} o_1$ (*time-point atom*), where $o_1, o_2 \in \Sigma$, $I \in Intv$, $n \in \mathbb{N}$, and $e_1, e_2 \in \{\mathsf{s}, \mathsf{e}\}$ (s for start, e for end).

An atom ρ is evaluated with respect to a Σ-*assignment* λ_Π for a given multi-timeline Π, which assigns to each token name $o \in \Sigma$ a pair $\lambda_\Pi(o) = (\pi, i)$, where 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 interval atom $o_1 \leq_I^{e_1,e_2} o_2$ *is satisfied by* λ_Π if $e_2(\lambda_\Pi(o_2)) - e_1(\lambda_\Pi(o_1)) \in I$. A point atom $o \leq_I^{e} n$ (resp., $n \leq_I^{e} o$) *is satisfied by* λ_Π if $n - e(\lambda_\Pi(o)) \in I$ (resp., $e(\lambda_\Pi(o)) - n \in I$).

Existential statement. An *existential statement* $\mathcal{E}$ for a finite set SV of state variables has 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 $1 \leq i \leq n$.

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*.

Given a Σ-assignment λ_Π for a multi-timeline Π of SV, we say that λ_Π *is consistent with the existential statement* $\mathcal{E}$ if, for each quantifier $o_i[x_i = v_i]$, we have $\lambda_\Pi(o_i) = (\pi, h)$, 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 consistent with $\mathcal{E}$ such that each atom in $\mathcal{C}$ is satisfied by λ_Π.

We can now introduce synchronization rules, which constrain tokens, possibly belonging to different timelines.

Definition 3. A *synchronization rule* $\mathcal{R}$ for a finite set SV of state variables is a rule of one of the following forms: (*trigger rules*) $o_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$; (*trigger-less rules*) $\top \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$, 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*. The quantifier $o_0[x_0 = v_0]$ in a trigger-rule is called *trigger*; we impose that only o_0 may occur free in $\mathcal{E}_i$, for all $1 \leq i \leq n$. No token name may occur free in trigger-less rules. A trigger rule $\mathcal{R}$ is *simple* if, for each existential statement $\mathcal{E}$ of $\mathcal{R}$ and each token name o distinct from the trigger, there is at most one *interval atom* of $\mathcal{E}$ where o occurs.

Intuitively, the trigger $o_0[x_0 = v_0]$ acts as a universal quantifier, which states that *for all* the tokens of the timeline for x_0, where x_0 takes the value v_0, at least one of the existential statements $\mathcal{E}_i$ must be satisfied. Trigger-less rules simply assert the satisfaction of some existential statement used to express initial conditions or goals, while trigger rules are much more powerful allowing the specification of invariants and response requirements. *Simple* trigger rules restrict the possibility of comparing multiple times the same token in an existential statement to trigger tokens only.

Example 4. Let us consider the system described in Example 2. The sensor and the processing unit respectively start in state `not_ready` and state $\texttt{reading}_1$. This is enforced by the following two trigger-less rules:
$\top \rightarrow \exists o[x_{\texttt{te}} = \texttt{not\_ready}].o \leq^{\texttt{s}}_{[0,0]} 0$ and $\top \rightarrow \exists o[x_{\texttt{p}} = \texttt{reading}_1].o \leq^{\texttt{s}}_{[0,0]} 0$.
A successful reading is possible only if the sensor and the processing unit are synchronized, namely, when a token of value $\texttt{reading}_j$ *contains* a token `ready`. (For a token o containing a token o', we write $contains(o, o')$ for $o \leq^{\texttt{s,s}}_{[0,+\infty[} o' \wedge o' \leq^{\texttt{e,e}}_{[0,+\infty[} o$.) Analogously, the processing unit can send data to the transmitter only if a token with value `send` contains a token with value $\texttt{read}_2$. If a reading attempt (a token $\texttt{reading}_1$) is unsuccessful, it is followed by a token $\texttt{read}_0$. (For a token o' following a token o, we write $next(o, o')$ for $o \leq^{\texttt{e,s}}_{[0,0]} o'$.) If the reading attempt is successful the token contains a `ready` token and it is followed by a $\texttt{read}_1$ token. To this end, we consider the trigger rule:
$o[x_{\texttt{p}} = \texttt{reading}_1] \rightarrow (\exists o_1[x_{\texttt{p}} = \texttt{read}_0].next(o, o_1) \vee \exists o_2[x_{\texttt{p}} = \texttt{read}_1] \exists o_3[x_{\texttt{te}} = \texttt{ready}].next(o, o_2) \wedge contains(o, o_3))$.
Notice that the trigger rule is not simple since the token name o_3 occurs twice in the definition of $contains(o, o_3)$. A similar rule can be written for the second temperature sampling and for transmission. An example of system goal is the successful transmission of two reads encoded by the following trigger-less rule:
$\top \rightarrow \exists o_1[x_{\texttt{p}} = \texttt{read}_2] \exists o_2[x_{\texttt{tr}} = \texttt{send}].(o_2 \leq^{\texttt{s,s}}_{[0,+\infty[} o_1 \wedge o_1 \leq^{\texttt{e,e}}_{[0,+\infty[} o_2)$.

Definition 5. Let Π be a multi-timeline of a set SV of state variables. (i) Given a *trigger-less rule* $\mathcal{R}$ of SV, Π *satisfies* $\mathcal{R}$ if Π satisfies some existential statement of $\mathcal{R}$. (ii) Given a *trigger rule* $\mathcal{R}$ of SV with trigger $o_0[x_0 = v_0]$, Π *satisfies* $\mathcal{R}$ if, for every position i of the timeline $\pi = \Pi(x_0)$ for x_0 such that $\pi(i) = (v_0, d)$, there exists an existential statement $\mathcal{E}$ of $\mathcal{R}$ and a Σ-assignment λ_Π for Π consistent with $\mathcal{E}$ such that $\lambda_\Pi(o_0) = (\pi, i)$ and λ_Π satisfies all the atoms of $\mathcal{E}$.

In the following, we consider also a stronger notion of satisfaction, called *satisfaction under the future semantics*, which requires that all non-trigger tokens selected by some quantifier do not start *strictly before* the trigger token.

Definition 6. Given a trigger rule $\mathcal{R} = o_0[x_0 = v_0] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$, a multi-timeline Π of SV satisfies $\mathcal{R}$ *under the future semantics* if Π satisfies the trigger rule obtained from $\mathcal{R}$ by replacing, in each existential statement $\mathcal{E}_i = \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n].\mathcal{C}$, the conjunction of atoms $\mathcal{C}$ by $\mathcal{C} \wedge \bigwedge_{i=1}^{n} o_0 \leq^{\texttt{s,s}}_{[0,+\infty[} o_i$.

With reference to Example 4, we observe that the occurrence in the trigger rule of $contains(o, o_3)$ which is by definition $o \leq^{\texttt{s,s}}_{[0,+\infty[} o_3 \wedge o_3 \leq^{\texttt{e,e}}_{[0,+\infty[} o$ can be simplified under the future semantics to $o_3 \leq^{\texttt{e,e}}_{[0,+\infty[} o$ since the first conjunct is imposed by the future semantics itself. This implies that the considered trigger rule which is not *a simple rule* in the general semantics can be converted to *a simple rule* in the future semantics.

A *TP domain* $P = (SV, R)$ is specified by a finite set SV of state variables and a finite set R of synchronization rules for SV modeling their admissible behaviors. A *plan for* $P = (SV, R)$ is a multi-timeline of SV satisfying all the rules in R. A *future plan for* P is defined in a similar way, but it requires the satisfaction of *all* trigger rules under the future semantics.

In the next sections, we will study the following decision problems:

- *TP problem*: given a TP domain $P = (SV, R)$, is there a plan for P?
- *Future TP problem*: given $P = (SV, R)$, is there a *future* plan for P?

3. TP over dense temporal domains is an undecidable problem

We start by settling an important negative result, namely, we show that the TP problem, in its full generality, is *undecidable* over *dense temporal domains*, even when a single state variable is involved. Undecidability is proved via a reduction from the halting problem for *Minsky 2-counter machines* [19]. The proof resembles the one for the satisfiability problem of Metric Temporal Logic (which will be formally introduced in Section 4), with both past and future temporal modalities, interpreted on dense time [3].

As a preliminary step, we give a short account of Minsky 2-counter machines. A Minsky 2-counter machine (*counter machine* for short) is a tuple $M = (\mathsf{Inst}, \ell_{init}, \ell_{halt})$ consisting of a finite set Inst of labeled instructions of the form $\ell : \iota$, where ℓ is a label and ι is an instruction for either (i) *increasing* counter h ($c_h := c_h + 1$; `goto` ℓ_r), or (ii) *decreasing* counter h (`if` $c_h > 0$ `then` $c_h := c_h - 1$; `goto` ℓ_s `else goto` ℓ_t), where $h \in \{1, 2\}$, $\ell_s \neq \ell_t$, and ℓ_r (respectively, ℓ_s, ℓ_t) is either a label of an instruction in Inst or the halting label ℓ_{halt}. Moreover, $\ell_{init} \in \mathsf{Inst}$ is the label of a designated "initial"instruction.

An *M-configuration* is a triple of the form $C = (\ell, n_1, n_2)$, where ℓ is the label of an instruction (intuitively, the next instruction to be executed), and $n_1, n_2 \in \mathbb{N}$ are the current values of the two counters c_1 and c_2, respectively.

M induces a transition relation $\xrightarrow{M}$ over pairs of M-configurations: (i) for an instruction with label ℓ increasing c_1, we have $(\ell, n_1, n_2) \xrightarrow{M} (\ell_r, n_1 + 1, n_2)$, and (ii) for an instruction decreasing c_1, we have $(\ell, n_1, n_2) \xrightarrow{M} (\ell_s, n_1 - 1, n_2)$ if $n_1 > 0$, and $(\ell, 0, n_2) \xrightarrow{M} (\ell_t, 0, n_2)$ otherwise (the same for c_2). An *M-computation* is a *finite* sequence $C_1, \ldots, C_k$ of M-configurations such that $C_i \xrightarrow{M} C_{i+1}$ for all $1 \leq i < k$. *M halts* if there exists an M-computation starting at $(\ell_{init}, 0, 0)$ and leading to (ℓ_{halt}, n_1, n_2), for some $n_1, n_2 \in \mathbb{N}$. The *halting problem for a counter machine M* is to decide whether M halts. The problem was proved to be *undecidable* by Minsky [19].

Theorem 7. *The TP problem over dense temporal domains is undecidable.*

Proof. The proof consists of a reduction from the halting problem for Minsky 2-counter machines. For an instruction ℓ, we use the following notation:

- for increments $\ell : c_h := c_h + 1$; `goto` ℓ_r, we define $c(\ell) = c_h$ and $\mathsf{succ}(\ell) = \ell_r$;
- for decrements ℓ: `if` $c_h > 0$ `then` $c_h := c_h - 1$; `goto` ℓ_r `else goto` ℓ_s, we define $c(\ell) = c_h$, $\mathsf{dec}(\ell) = \ell_r$, and $\mathsf{zero}(\ell) = \ell_s$.

Moreover, let InstLab be the set of instruction labels, including ℓ_{halt}, and let Inc (resp., Dec) be the set of labels for increment (resp., decrement) instructions. We consider a counter machine $M = (\mathsf{Inst}, \ell_{init}, \ell_{halt})$ assuming, w.l.o.g., that no instruction of M leads to ℓ_{init}, and that ℓ_{init} labels an increment instruction. We build in polynomial time a state variable $x_M = (V, T, D)$ and a finite set R_M of synchronization rules over x_M such that M halts if and only if there is a plan for $P = (\{x_M\}, R_M)$, i.e., a timeline for x_M that satisfies all the rules in R_M.

Encoding of M-computations. We start by defining the encoding of a computation of M as a timeline for x_M. The finite domain of the state variable x_M is the set of symbols $V = V_{check} \cup V_{main}$, where

$$V_{check} = \bigcup_{\ell \in \mathsf{InstLab}} \bigcup_{i,h \in \{1,2\}} \bigcup_{op_i \in \{\mathsf{inc}_i, \mathsf{dec}_i, \mathsf{zero}_i\}} \Big(\{(\ell, op_i)\} \cup \{(\ell, op_i, c_h)\} \cup \{(\ell, op_i, (c_h, \#))\} \Big),$$

$$V_{main} = \bigcup_{\ell \in \mathsf{Inc} \cup \{\ell_{halt}\}} \bigcup_{h \in \{1,2\}} \Big(\{\ell\} \cup \{(\ell, c_h)\} \Big) \cup$$

$$\bigcup_{\ell \in \mathsf{Dec}} \bigcup_{\ell' \in \{\mathsf{zero}(\ell), \mathsf{dec}(\ell)\}} \bigcup_{h \in \{1,2\}} \Big(\{(\ell, \ell')\} \cup \{(\ell, \ell', c_h)\} \cup \{(\ell, \ell', (c_h, \#))\} \Big).$$

For each $h \in \{1, 2\}$, we denote by V_{c_h} the set of V-values v of the form $v = (\ell, c)$, $v = (\ell, \ell', c)$, or $v = (\ell, op, c)$, with $c \in \{c_h, (c_h, \#)\}$; if $c = c_h$ (resp., $c = (c_h, \#)$), we say that v is an *unmarked* (resp., *marked*) V_{c_h}-value.

An M-configuration is encoded by a finite word over V consisting of the concatenation of a *check*-code and a *main*-code.

The *main*-code w_{main} for an M-configuration (ℓ, n_1, n_2), with $\ell \in \mathsf{Inc} \cup \{\ell_{halt}\}$, $n_1 \geq 0$, and $n_2 \geq 0$, has the form $w_{main} = \ell \cdot (\ell, c_1)^{n_1} \cdot (\ell, c_2)^{n_2}$.

In case of a *decrement* instruction label $\ell \in \mathsf{Dec}$ such that $c(\ell) = c_1$, the *main*-code w'_{main} has one of the following two forms, depending on whether the value of c_1 in the encoded configuration is equal to or greater than 0:

- either $w'_{main} = (\ell, \mathsf{zero}(\ell)) \cdot (\ell, \mathsf{zero}(\ell), c_2)^{n_2}$
- or $w'_{main} = (\ell, \mathsf{dec}(\ell)) \cdot (\ell, \mathsf{dec}(\ell), (c_1, \#)) \cdot (\ell, \mathsf{dec}(\ell), c_1)^{n_1} \cdot (\ell, \mathsf{dec}(\ell), c_2)^{n_2}$.

In the former case, w'_{main} encodes the configuration $(\ell, 0, n_2)$; in the latter, the configuration $(\ell, n_1 + 1, n_2)$. Note that, in the second case, there is exactly one occurrence of a *marked* V_{c_1}-value which intuitively "marks" the unit of the counter that will be removed by the decrement. The *main*-code for a *decrement* instruction label ℓ with $c(\ell) = c_2$ has two symmetric forms.

The *check*-code is used to trace both an M-configuration C and the type of instruction associated with the configuration C_p preceding C in the considered computation. The type of instruction is given by the symbols inc_i, dec_i, and zero_i, with $i \in \{1, 2\}$: inc_i (resp., dec_i, zero_i) means that C_p is associated with an instruction that increases the counter c_i (resp., decreases c_i with c_i greater than 0 in C_p, decreases c_i with c_i equal to 0 in C_p).

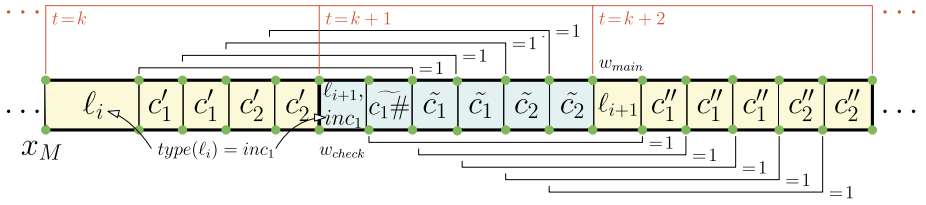

Fig. 2. A computation-code with configuration-code for an instruction ℓ_{i+1}. Main-codes are highlighted in yellow and check-codes in cyan. Each square is a token of a timeline for x_M decorated with their start time and temporal constraints. The symbols c'_h, $\tilde{c}_h$, $\widetilde{c_h\#}$, and c''_h, for $h \in \{1,2\}$, stand respectively for (ℓ_i, c_h), (ℓ_{i+1}, inc_1, c_h), $(\ell_{i+1}, inc_1, (c_h, \#))$, and (ℓ_{i+1}, c_h). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The *check*-code for an instruction label $\ell \in \mathsf{InstLab}$ and an inc_1-operation has the form $(\ell, \mathsf{inc}_1) \cdot (\ell, \mathsf{inc}_1, (c_1, \#)) \cdot (\ell, \mathsf{inc}_1, c_1)^{n_1} (\ell, \mathsf{inc}_1, c_2)^{n_2}$ and encodes the configuration $(\ell, n_1 + 1, n_2)$. The *marked* occurrence of a V_{c_1}-value represents the unit added to the counter by the increment. The *check*-code for an instruction label $\ell \in \mathsf{InstLab}$ and an operation $op_1 \in \{\mathsf{dec}_1, \mathsf{zero}_1\}$ for counter c_1 has the form $(\ell, op_1) \cdot (\ell, op_1, c_1)^{n_1} \cdot (\ell, op_1, c_2)^{n_2}$. where $n_1 = 0$ if $op_1 = \mathsf{zero}_1$. The *check*-code for $\ell \in \mathsf{InstLab}$ and $op_2 \in \{\mathsf{dec}_2, \mathsf{zero}_2\}$ for counter c_2 is defined similarly.

A *configuration*-code is a word $w = w_{check} \cdot w_{main}$ such that w_{check} and w_{main} are a *check*-code and a *main*-code, respectively, associated with the same instruction label. The configuration-code is *well-formed* if w_{check} and w_{main} encode the same configuration. Fig. 2 depicts a configuration-code for the instruction ℓ_{i+1} which increments c_1.

A *computation*-code is a sequence of configuration-codes $\pi = w^1_{check} \cdot w^1_{main} \cdots w^n_{check} \cdot w^n_{main}$ such that, for all $1 \leq j < n$, the following holds (let ℓ_i be the label associated with $w^i_{check} \cdot w^i_{main}$): (i) $\ell_j \neq \ell_{halt}$; (ii) if $\ell_j \in \mathsf{Inc}$, with $c(\ell_j) = c_h$, then $\ell_{j+1} = \mathsf{succ}(\ell_j)$ and w^{j+1}_{check} is associated with the operation inc_h; (iii) if $\ell_j \in \mathsf{Dec}$, with $c(\ell_j) = c_h$, and the first symbol of w^j_{main} is $(\ell_j, \mathsf{zero}(\ell_j))$ (resp., $(\ell_j, \mathsf{dec}(\ell_j))$), then $\ell_{j+1} = \mathsf{zero}(\ell_j)$ (resp., $\ell_{j+1} = \mathsf{dec}(\ell_j)$) and w^{j+1}_{check} is associated with operation zero_h (resp., dec_h).

The computation-code π is *well-formed* if, additionally, each configuration-code in π is *well-formed* and, for all $1 \leq j < n$, the following holds (we assume (ℓ_i, n^i_1, n^i_2) to be the configuration encoded by $w^i_{check} \cdot w^i_{main}$): (i) if $\ell_j \in \mathsf{Inc}$, with $c(\ell_j) = c_h$, then $n^{j+1}_h = n^j_h + 1$ and $n^{j+1}_{3-h} = n^j_{3-h}$; (ii) if $\ell_j \in \mathsf{Dec}$, with $c(\ell_j) = c_h$, then $n^{j+1}_{3-h} = n^j_{3-h}$. Moreover, if w^{j+1}_{check} is associated with dec_h, then $n^{j+1}_h = n^j_h - 1$. A computation-code π is *initial* if it starts with the prefix $(\ell_{init}, \mathsf{zero}_1) \cdot \ell_{init}$, and it is *halting* if it leads to a configuration-code associated with the halting label ℓ_{halt}. Clearly, a well-formed computation-code π encodes a computation of the machine M which halts if and only if there is an initial and halting well-formed computation-code.

Definition of x_M and R_M. In the following, we reduce the problem of checking the existence of an initial and halting well-formed computation-code to a TP problem for a TP domain $(\{x_M\}, R_M)$. The idea is to define a timeline for the state variable x_M where the sequence of values of its tokens is a computation-code whose well-formedness is guaranteed by exploiting the duration of tokens and the synchronization rules in R_M. Now, the untimed part (i.e., neglecting duration of tokens) of any plan for $(\{x_M\}, R_M)$ is an initial and halting well-formed computation-code and M halts if and only such a plan exists. In more detail (see Fig. 2 for an intuition), each symbol of the computation-code is associated with a token with a (non-fixed) positive duration. The overall duration of the sequence of tokens corresponding to a check-code or a main-code amounts exactly to one time unit (dense time allows one to encode arbitrarily large values of counters in one time unit). In two adjacent check/main-codes, the time elapsed between the start times of corresponding elements in the representation of the value of a counter (see elements in Fig. 2 connected by horizontal lines) amounts exactly to one time unit. Such a constraint allows us to compare the values of counters in adjacent codes, either checking for equality, or simulating (by using marked symbols) increment and decrement operations.

Let us now formally define x_M and R_M. As for x_M, we take $x_M = (V, T, D)$ such that $D(v) =]0, 1]$, for each $v \in V$. Note that the duration of any token in a timeline is greater than 0 and less than or equal to 1 (*strict time monotonicity* constraint). It is a tedious but straightforward task to define T of x_M in such a way that the following requirement is fulfilled.

Claim 8. The untimed part of any timeline for x_M, whose first token has value $(\ell_{init}, \mathsf{zero}_1)$, is a prefix of some initial computation-code. Moreover, $(\ell_{init}, \mathsf{zero}_1) \notin T(v)$ for all $v \in V$.

Synchronization rules in R_M ensure the following requirements.

Initialization. Every timeline starts with two tokens, of value $(\ell_{init}, \mathsf{zero}_1)$ and ℓ_{init}, respectively. Since no instruction leads to ℓ_{init}, by Claim 8 we only need two trigger-less rules: $\top \rightarrow \exists o[x_M = (\ell_{init}, \mathsf{zero}_1)]$. $\top$ and $\top \rightarrow \exists o[x_M = \ell_{init}]$. $\top$.

Halting. Every timeline leads to a configuration-code associated with the halting label. By the rules for initialization and Claim 8, the following trigger-less rule suffices: $\top \rightarrow \exists o[x_M = \ell_{halt}]$. $\top$.

One time unit distance between consecutive control values. A *control V-value* corresponds to the first symbol of a *main*-code or a *check*-code, i.e., it is an element of $V_{\mathsf{con}} = V \setminus (V_{c_1} \cup V_{c_2})$. For each pair of tokens tk and tk' along a timeline such that tk and tk' have a control V-value, tk precedes tk', and there is no token between tk and tk' with a control V-value, it holds

that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$. By Claim 8, strict time monotonicity, and halting requirements, it suffices to ensure that each token tk having a control V-value distinct from ℓ_{halt} is eventually followed by a token tk' such that tk' has a control V-value and $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$. To this end, for each $v \in V_{\mathsf{con}} \setminus \{\ell_{halt}\}$, we add a trigger rule of the form: $o[x_M = v] \rightarrow \bigvee_{u \in V_{\mathsf{con}}} \exists o'[x_M = u].\, o \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o'$.

Well-formedness of configuration-codes. For each configuration-code $w_{check} \cdot w_{main}$ in a timeline and each counter c_h, the value of c_h in the *main*-code w_{main} and the *check*-code w_{check} must coincide. By Claim 8 and the previous requirements, it suffices to ensure that (i) each token tk with a V_{c_h}-value in V_{check} is eventually followed by a token tk' with a V_{c_h}-value such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$, and (ii) each token tk with a V_{c_h}-value in V_{main} is preceded at some point by a token tk' with a V_{c_h}-value such that $\mathsf{s}(tk) - \mathsf{s}(tk') = 1$. As for (i), for each $v \in V_{c_h} \cap V_{check}$, we add the rule: $o[x_M = v] \rightarrow \bigvee_{u \in V_{c_h}} \exists o'[x_M = u].\, o \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o'$, while for (ii), for each $v \in V_{c_h} \cap V_{main}$, we add the rule: $o[x_M = v] \rightarrow \bigvee_{u \in V_{c_h}} \exists o'[x_M = u].\, o' \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o$.

Increment and decrement. Increments and decrements must be correctly simulated. By Claim 8 and the previous requirements, we can assume that the untimed part π of a timeline is an initial and halting computation-code such that all configuration-codes occurring in π are well-formed.

Let $w_{main} \cdot w_{check}$ be a subword occurring in π such that w_{main} (resp., w_{check}) is a *main*-code (resp., *check*-code). Let ℓ_{main} (resp., ℓ_{check}) be the instruction label associated with w_{main} (resp., w_{check}) and for $i = 1, 2$, let n_i^{main} (resp., n_i^{check}) be the value of counter c_i encoded by w_{main} (resp., w_{check}). Let $c_h = c(\ell_{main})$. By construction, $\ell_{main} \neq \ell_{halt}$, and either $\ell_{main} \in \mathsf{Inc}$ and $\ell_{check} = \mathsf{succ}(\ell_{main})$, or $\ell_{main} \in \mathsf{Dec}$ and $\ell_{check} \in \{\mathsf{zero}(\ell_{main}), \mathsf{dec}(\ell_{main})\}$. Moreover, if $\ell_{main} \in \mathsf{Dec}$ and $\ell_{check} = \mathsf{zero}(\ell_{main})$, then $n_h^{check} = n_h^{main} = 0$. Thus, it remains to ensure the following two requirements:

$(*)$ if $\ell_{main} \in \mathsf{Inc}$, then $n_h^{check} = n_h^{main} + 1$ and $n_{3-h}^{check} = n_{3-h}^{main}$;
$(**)$ if $\ell_{main} \in \mathsf{Dec}$, then $n_{3-h}^{check} = n_{3-h}^{main}$, and whenever $\ell_{check} = \mathsf{dec}(\ell_{main})$, then $n_h^{check} = n_h^{main} - 1$.

By strict time monotonicity and one time unit distance between consecutive control values, it follows that requirements $(*)$ and $(**)$ are captured by the following rules, where U_{c_i} denotes the set of *unmarked* V_{c_i}-values, for $i = 1, 2$, and V_{init} (resp., V_{halt}) is the set of V-values associated with the label ℓ_{init} (resp., ℓ_{halt}). For each $v \in (U_{c_i} \cap V_{main}) \setminus V_{halt}$, we add the rule: $o[x_M = v] \rightarrow \bigvee_{u \in U_{c_i}} \exists o'[x_M = u].\, o \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o'$, and for each $v \in (U_{c_i} \cap V_{check}) \setminus V_{init}$, we add the rule: $o[x_M = v] \rightarrow \bigvee_{u \in U_{c_i}} \exists o'[x_M = u].\, o' \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o$. This ends the proof. □

Note that since all the trigger rules used in the proof are *simple*, *undecidability of the TP problem holds also under the restriction to simple trigger rules.*

It is worth pointing out that trigger rules are necessary to ensure the well-formedness of configuration-codes and the increment/decrement requirements. Consider, for instance, the well-formedness rule. We can force every token with value in V_{c_h} (for $h = 1, 2$) in a w_{check} code to be followed, one time instant later, by a token with value in V_{c_h} in the following w_{main} code. However, under the future semantics, the converse cannot be ensured. As a consequence, we could only enforce that if n is the value encoded for the counter c_h in a check code, then the value encoded for c_h in the following w_{main} code is n' with $n \leq n'$, whereas we had to ensure that $n = n'$. For the same reason, under the future semantics, we can not correctly encode increments and decrements, thus losing the ability of capturing computations of (exact) Minsky machines. In fact, we shall prove that, under the future semantics, we can encode computations of the variant of Minsky machines called *gainy counter machines* [11], whose counters may "erroneously" increase. Since the halting problem for gainy counter machines is known to be non-primitive recursive [11], we can prove the non-primitive recursive-hardness of the future TP problem. The encoding of the gainy counter machines halting problem into the future TP problem, which is an adaptation of the above one for Minsky machines, is reported in Appendix A.

Theorem 9. *The future TP problem is non-primitive recursive-hard also under one of the following two assumptions:* either (i) *the trigger rules are simple,* or (ii) *the intervals (occurring in atoms or constraint functions) are in* $Intv_{(0,\infty)}$.

4. Decidability of future TP with simple trigger rules

In this section, we show that decidability of the TP problem can be recovered by using trigger rules which are both *simple* and *interpreted under the future semantics*. If, in addition, the intervals occurring in trigger rules are non-singular (resp., are in $Intv_{(0,\infty)}$), the problem is in **EXPSPACE** (resp., in **PSPACE**). Decidability of *future TP with arbitrary trigger rules remains an open problem* (we conjecture undecidability). Decidability is proved in Subsection 4.2 by reducing the future TP problem with simple trigger rules to the decidable *existential MC problem* for Timed Automata (TA) [1] against Metric Temporal Logic (MTL) [18] over *finite timed words* (see [21]). We start recalling the basics of TA and MTL.

4.1. Timed automata and the logic MTL

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 $\Sigma \times \mathbb{R}_+$ (τ_i is called a *timestamp* and, intuitively, it represents the time when the "event" a_i occurs) such that $\tau_i \leq \tau_{i+1}$ for all $0 \leq i < n$ (*mono-*

tonicity requirement). We often denote the timed word w by (σ, τ), where σ is the finite (untimed) word $a_0 \cdots a_n$ and τ is the sequence of timestamps $\tau_0, \ldots, \tau_n$. A *timed language* over Σ is a set of timed words over Σ.

Timed automata (TA) Let C be a finite set of clocks. A clock valuation is a function $val : C \to \mathbb{R}_+$ that assigns a non-negative real value to each clock in C. Let $t \in \mathbb{R}_+$ and $Res \subseteq C$ (called *reset set*). For all $c \in C$, let $(val + t)$ be the valuation for C such that $(val + t)(c) = val(c) + t$, and $val[Res]$ be the valuation for C such that $val[Res](c) = 0$, if $c \in Res$, and $val[Res](c) = val(c)$ otherwise.

A *clock constraint* θ over C is a Boolean combination of atomic formulas of the form $c \in I$ or $c - c' \in I$, where $c, c' \in C$ and $I \in Intv$. Given a clock valuation val and a clock constraint θ, val is said to satisfy θ, written $val \models \theta$, if θ evaluates to true after replacing each occurrence of a clock c in θ by $val(c)$, and interpreting Boolean connectives and membership to intervals in the standard way. We denote by $\Phi(C)$ the set of all possible clock constraints over C.

Definition 10. A timed automaton (TA) over Σ is a tuple $\mathcal{A} = (\Sigma, Q, q_0, C, \Delta, F)$, where Q is a finite set of (control) states, $q_0 \in Q$ is the initial state, C is a finite set of clocks, $F \subseteq Q$ is the set of accepting states, and $\Delta \subseteq Q \times \Sigma \times \Phi(C) \times 2^C \times Q$ is the transition relation. The *maximal constant of* $\mathcal{A}$ is the greatest integer occurring as an endpoint of some interval in the clock constraints of the transitions of $\mathcal{A}$.

A configuration of $\mathcal{A}$ is a pair (q, val), where $q \in Q$ and val is a clock valuation for C. 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 from the initial configuration (q_0, val_0), with $val_0(c) = 0$ for all $c \in C$ (*initialization requirement*) and such that, for $0 \le i \le n$, (i) $(q_i, a_i, \theta, Res, q_{i+1}) \in \Delta$ for some $\theta \in \Phi(C)$ and reset set Res, (ii) $(val_i + \tau_i - \tau_{i-1}) \models \theta$, and (iii) $val_{i+1} = (val_i + \tau_i - \tau_{i-1})[Res]$, where $\tau_{-1} = 0$ (*consecution requirement*).

The behavior of a TA $\mathcal{A}$ can be described as follows. Assume that $\mathcal{A}$ is in state $q \in Q$ after reading the symbol (a', τ_i) at time τ_i and, at that time, the clock valuation is val. Upon reading (a, τ_{i+1}), $\mathcal{A}$ chooses a transition of the form $\delta = (q, a, \theta, Res, q') \in \Delta$ such that the constraint θ is fulfilled by $(val + t)$, with $t = \tau_{i+1} - \tau_i$. The control then changes from q to q' and val is updated in such a way as to record the amount of elapsed time t in the clock valuation, and to reset the clocks in Res, namely, val is updated to $(val + t)[Res]$.

A run r is *accepting* if $q_{n+1} \in F$. The *timed language* $\mathcal{L}_T(\mathcal{A})$ is the set of timed words w over Σ such that there is an accepting run of $\mathcal{A}$ on w. As shown in [1], given two TA $\mathcal{A}_1$ and $\mathcal{A}_2$, with s_1 (resp., s_2) states and k_1 (resp., k_2) clocks, the union (resp., intersection) automaton $\mathcal{A}_\vee$ (resp., $\mathcal{A}_\wedge$) such that $\mathcal{L}_T(\mathcal{A}_\vee) = \mathcal{L}_T(\mathcal{A}_1) \cup \mathcal{L}_T(\mathcal{A}_2)$ (resp., $\mathcal{L}_T(\mathcal{A}_\wedge) = \mathcal{L}_T(\mathcal{A}_1) \cap \mathcal{L}_T(\mathcal{A}_2)$) can be effectively computed, and has $s_1 + s_2$ (resp., $s_1 \cdot s_2$) states and $k_1 + k_2$ (resp., $k_1 + k_2$) clocks.

The logic MTL Metric Temporal Logic (MTL) extends LTL with time constraints on the until modality [18]. Let $\mathcal{AP}$ be a finite set of proposition letters. The set of MTL formulas φ over $\mathcal{AP}$ is defined by the grammar: $\varphi ::= \top \mid p \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \mathsf{U}_I \varphi$, where $p \in \mathcal{AP}$, $I \in Intv$, and U_I is the *strict timed until* MTL modality.

MTL formulas over $\mathcal{AP}$ are interpreted on timed words over $2^{\mathcal{AP}}$. Given an MTL formula φ, a timed word $w = (\sigma, \tau)$ over $2^{\mathcal{AP}}$, and a position $0 \le i < |w|$, the satisfaction relation $(w, i) \models \varphi$—meaning that φ holds at position i of w—is defined as follows (we omit the clauses for Boolean connectives):

- $(w, i) \models p \iff p \in \sigma(i)$,
- $(w, i) \models \varphi_1 \mathsf{U}_I \varphi_2 \iff$ there exists $j > i$ such that $\tau_j - \tau_i \in I$, $(w, j) \models \varphi_2$, and $(w, k) \models \varphi_1$ for all $i < k < j$.

A *model of* φ is a timed word w over $2^{\mathcal{AP}}$ such that $(w, 0) \models \varphi$. The *timed language* $\mathcal{L}_T(\varphi)$ is the set of models of φ.

The *existential MC problem for TA against MTL* is the problem of checking, for a given TA $\mathcal{A}$ over $2^{\mathcal{AP}}$ and an MTL formula φ over $\mathcal{AP}$, if $\mathcal{L}_T(\mathcal{A}) \cap \mathcal{L}_T(\varphi) \neq \emptyset$.

In the following, we use standard shortcuts such as $\mathsf{F}_I \varphi$ for $\varphi \vee (\top \mathsf{U}_I \varphi)$ (*timed eventually*) and $\mathsf{G}_I \varphi$ for $\neg \mathsf{F}_I \neg\varphi$ (*timed always*). We also consider two fragments of MTL, namely, MITL (Metric Interval Temporal Logic) and $\mathsf{MITL}_{(0,\infty)}$ [2]: MITL is obtained from MTL by allowing only non-singular intervals of $Intv$ as subscripts of U, while $\mathsf{MITL}_{(0,\infty)}$ is obtained from MITL by allowing only intervals in $Intv_{(0,\infty)}$. The *maximal constant* of an MTL formula φ is the greatest integer occurring as an endpoint of some interval of (the occurrences of) U_I in φ.

4.2. Reduction to existential MC for TA against MTL

We now solve future TP with simple trigger rules by means of an exponential-time reduction to the existential MC problem for TA against MTL.

Let $P = (SV, R)$ be an instance of the problem where the trigger rules in R are simple. The *maximal constant* of P, denoted by K_P, is the greatest integer occurring in the atoms of the rules in R and in the constraint functions of the state variables in SV. The proposed reduction consists of three steps:

- first, we define an encoding of the multi-timelines of SV by means of timed words over $2^{\mathcal{AP}}$ for a suitable finite set $\mathcal{AP}$ of proposition letters, and show how to construct a TA $\mathcal{A}_{SV}$ over $2^{\mathcal{AP}}$ accepting such encodings;

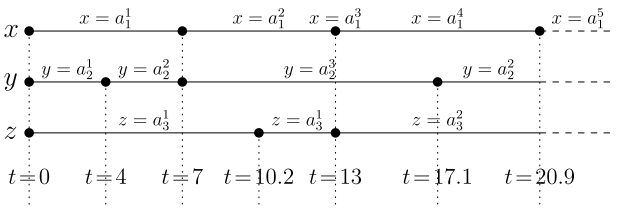

Fig. 3. Example of multi-timeline of $SV = \{x, y, z\}$. The timeline for x is $(a_1^1, 7), (a_1^2, 5), (a_1^3, 0), (a_1^4, 7.9), (a_1^5, \ldots)$. Note that the third token has null duration. The encoding of the timeline for x is $(\{(beg_x, a_1^1), p_>\}, 0)(\{(a_1^1, a_1^2), p_>\}, 7)(\{(a_1^2, a_1^3), p_>\}, 13)(\{(a_1^3, a_1^4), past^{\mathsf{s}}_{a_1^3}, past^{\mathsf{e}}_{a_1^2}\}, 13)$ $(\{(a_1^4, a_1^5), p_>\}, 20.9)\cdots$ The encoding of the multi-timeline is $(\{(beg_x, a_1^1), (beg_y, a_2^1), (beg_z, a_3^1), p_>\}, 0)(\{(a_2^1, a_2^2), p_>\}, 4)(\{(a_1^1, a_1^2), (a_2^2, a_2^3), p_>\}, 7)$ $(\{(a_3^1, a_3^1), p_>\}, 10.2)(\{(a_1^2, a_1^3), (a_3^1, a_3^2), p_>\}, 13)(\{(a_1^3, a_1^4), past^{\mathsf{s}}_{a_1^3}, past^{\mathsf{e}}_{a_1^2}\}, 13)(\{(a_2^3, a_2^2), p_>\}, 17.1)\cdots$.

- next, we build an MTL formula $\varphi_\forall$ over $\mathcal{AP}$ such that for each multi-timeline Π of SV and encoding w_Π of Π, w_Π is a model of $\varphi_\forall$ if and only if Π satisfies all the trigger rules in R under the future semantics;
- finally, we construct a TA $\mathcal{A}_\exists$ over $2^{\mathcal{AP}}$ such that for each multi-timeline Π of SV and encoding w_Π of Π, w_Π is accepted by $\mathcal{A}_\exists$ if and only if Π satisfies all the trigger-less rules in R.

Hence, there is a future plan for $P = (SV, R)$ iff $\mathcal{L}_T(\mathcal{A}_{SV}) \cap \mathcal{L}_T(\mathcal{A}_\exists) \cap \mathcal{L}_T(\varphi_\forall) \neq \emptyset$.

For each $x \in SV$, we let $x = (V_x, T_x, D_x)$. Given an interval $I \in Intv$ and $n \in \mathbb{N}$, let $n + I$ (resp., $n - I$) denote the set of non-negative real numbers $\tau \in \mathbb{R}_+$ such that $\tau - n \in I$ (resp., $n - \tau \in I$). Note that $n + I$ (resp., $n - I$) is a (possibly empty) interval in $Intv$ whose endpoints can be trivially calculated. For an atom ρ in R involving a time constant (time-point atom), let $I(\rho)$ be the interval in $Intv$ defined as follows: if ρ has the form $o \leq^e_I n$ (resp., $n \leq^e_I o$), then $I(\rho) = n - I$ (resp., $I(\rho) = n + I$). Finally, let $Intv_R$ be the set of intervals $J \in Intv$ such that $J = I(\rho)$ for some time-point atom ρ occurring in a trigger rule of R.

Encodings of multi-timelines of SV For any pair of distinct state variables x and x', we assume the sets $V_x \cup V_{x'} = \emptyset$. To encode multi-timelines of SV, we make use of the set $\mathcal{AP} = (\bigcup_{x \in SV} Main_x) \cup Deriv$ of proposition letters, where

$$Main_x = (\{beg_x\} \cup V_x) \times V_x) \cup (V_x \times \{end_x\}) \quad \text{and} \quad Deriv = Intv_R \cup \{p_>\} \cup \bigcup_{x \in SV} \bigcup_{v \in V_x} \{past^{\mathsf{s}}_v, past^{\mathsf{e}}_v\}.$$

Intuitively, we use proposition letters in $Main_x$ to encode a token along a timeline for x. Proposition letters in $Deriv$ enrich the encoding in order to translate simple trigger rules in MTL formulas under the future semantics (see below). The tags beg_x and end_x in $Main_x$ are used to mark the start and the end of a timeline for x. A token tk with value v along a timeline for x is encoded by two events: the *start-event* (occurring at the start time of tk) and the *end-event* (occurring at the end time of tk). The start-event of tk is specified by a main proposition letter of the form (v_p, v), where either $v_p = beg_x$ (tk is the first token of the timeline) or v_p is the value of the token for x preceding tk. The end-event of tk is instead specified by a main proposition letter of the form (v, v_s), where either $v_s = end_x$ (tk is the last token of the timeline) or v_s is the value of the token for x following tk. An example of encoding is given in Fig. 3.

Let us consider now the proposition letters in $Deriv$. The elements in $Intv_R$ reflect the semantics of the time-point atoms in the trigger rules of R: for each $I \in Intv_R$, I holds at the current position if the current timestamp τ satisfies $\tau \in I$. The proposition letter $p_>$ is used to mark a timestamp whenever it is strictly greater than the previous one. Finally, a proposition letter $past^{\mathsf{s}}_v$ (resp., $past^{\mathsf{e}}_v$) is used to mark a timestamp τ whenever it is preceded by a token of value v starting (resp., ending) at the same time τ. An *encoding of a timeline for x* is a timed word w over $2^{Main_x \cup Deriv}$ of the form

$$w = (\{(beg_x, v_0)\} \cup S_0, \tau_0)(\{(v_0, v_1)\} \cup S_1, \tau_1) \cdots (\{(v_n, end_x)\} \cup S_{n+1}, \tau_{n+1})$$

where, for all $0 \leq i \leq n + 1$, $S_i \subseteq Deriv$, and (i) $v_{i+1} \in T_x(v_i)$ for $i < n$; (ii) $\tau_0 = 0$ and $\tau_{i+1} - \tau_i \in D_x(v_i)$ for $i \leq n$; (iii) $S_i \cap Intv_R$ is the set of intervals $I \in Intv_R$ such that $\tau_i \in I$; (iv) $p_> \in S_i$ if and only if either $i = 0$ or $\tau_i > \tau_{i-1}$; (v) for all $v \in V_x$, $past^{\mathsf{s}}_v \in S_i$ (resp., $past^{\mathsf{e}}_v \in S_i$) if and only if there is $0 \leq h < i$ such that $\tau_h = \tau_i$ and $v = v_h$ (resp., $\tau_h = \tau_i$, $v = v_{h-1}$, and $h > 0$). Note that the length of w is at least 2. The given timed word w encodes the timeline for x of length $n + 1$ given by $\pi = (v_0, \tau_1)(v_1, \tau_2 - \tau_1) \cdots (v_n, \tau_{n+1} - \tau_n)$. The timestamps τ_i and τ_{i+1} represent the start and the end time of the i-th token of the timeline π ($0 \leq i \leq n$). See again Fig. 3 for an example.

Next, we define the encoding of a multi-timeline of SV. For $P \subseteq \mathcal{AP}$ and $x \in SV$, let $P[x] = P \setminus \bigcup_{y \in SV \setminus \{x\}} Main_y$. An *encoding of a multi-timeline of SV* is a timed word w over $2^{\mathcal{AP}}$ of the form $w = (P_0, \tau_0) \cdots (P_n, \tau_n)$ such that (i) for all $x \in SV$, the timed word obtained from $(P_0[x], \tau_0) \cdots (P_n[x], \tau_n)$ by removing the pairs $(P_i[x], \tau_i)$ such that $P_i[x] \cap Main_x = \emptyset$ is an encoding of a timeline for x, and (ii) $P_0[x] \cap Main_x \neq \emptyset$ for all $x \in SV$ (initialization). See again Fig. 3 for an example of the encoding of a multi-timeline.

We now construct a TA $\mathcal{A}_{SV}$ over $2^{\mathcal{AP}}$ accepting the encodings of the multi-timelines of SV (see the proof of the next proposition).

Proposition 11. *A TA $\mathcal{A}_{SV}$ over $2^{\mathcal{AP}}$, with $2^{O(\sum_{x\in SV}|V_x|)}$ states, $|SV|+2$ clocks, and maximal constant $O(K_P)$, such that $\mathcal{L}_T(\mathcal{A}_{SV})$ is the set of encodings of the multi-timelines of SV, can be built in exponential time.*

Proof. Let us fix an ordering $SV = \{x_1, \ldots, x_N\}$ of the state variables. Let $\mathcal{H} = Deriv \setminus (Intv_R \cup \{p_>\})$ and $V'_i = V_{x_i} \cup \{beg_{x_i}, end_{x_i}\}$, for all $1 \le i \le N$.

The TA $\mathcal{A}_{SV} = (2^{\mathcal{AP}}, Q, q_0, C, \Delta, F)$ is defined as follows.

- The set of states is given by $Q = V'_1 \times \ldots \times V'_N \times 2^{\mathcal{H}}$. Intuitively, for a state $(v_1, \ldots, v_N, H)$, the i-th component v_i keeps track of the value of the last (start-event for a) token for x_i read so far if $v_i \notin \{beg_{x_i}, end_{x_i}\}$. If $v_i = beg_{x_i}$ (resp., $v_i = end_{x_i}$), then no start-event for a token for x_i has been read so far (resp., no start-event for a token for x_i can be read). Moreover, the last component H of the state keeps track of past token events occurring at a timestamp coinciding with the last timestamp.
- $q_0 = (beg_{x_1}, \ldots, beg_{x_N}, \emptyset)$ and $F = \{(end_{x_1}, \ldots, end_{x_N}, H) \mid H \subseteq \mathcal{H}\}$.
- The set of clocks C is given by $C = \{c_1, \ldots, c_N, c_>, c_{glob}\}$. We have a clock c_i for each state variable x_i, which is used to check that the duration of a token for x_i with value v is in $D_{x_i}(v)$. Moreover, $c_>$ is a clock which is always reset and is used to capture the meaning of proposition letter $p_>$, whereas c_{glob} is a clock that measures the current (global) time and is never reset.
- The relation Δ consists of the transitions

  $$((v_1, ..., v_N, H),\ P,\ \theta_1 \wedge \ldots \wedge \theta_N \wedge \theta_> \wedge \theta_{glob},\ Res,\ (v'_1, ..., v'_N, H'))$$ such that

  - if $(v_1, ..., v_N, H) = q_0$, then $P \cap Main_x \neq \emptyset$ for all $x \in SV$ (initialization);
  - for all $1 \le i \le N$, it holds that
    - *either* $P \cap Main_{x_i} = \emptyset$, $v'_i = v_i$, $\theta_i = \top$, and $c_i \notin Res$ (intuitively, no event associated with x_i occurs in this case),
    - *or* $P \cap Main_{x_i} = (v_i, v'_i)$ (and thus $v_i \neq end_{x_i}$), $v'_i \in T_{x_i}(v_i)$, if both $v_i \in V_{x_i}$ and $v'_i \in V_{x_i}$, $c_i \in Res$, and $\theta_i = c_i \in D_{x_i}(v_i)$ (resp., $\theta_i = c_i \in [0,0]$) if $v_i \neq beg_{x_i}$ (resp., if $v_i = beg_{x_i}$);
  - $c_{glob} \notin Res$, $\theta_{glob} = \bigwedge_{I\in P\cap Intv_R} c_{glob} \in I \wedge \bigwedge_{I\in Intv_R\setminus P}(c_{glob} \in \overrightarrow{I} \vee c_{glob} \in \overleftarrow{I})$ where, for each $I \in Intv_R \setminus P$, $\overrightarrow{I}$ and $\overleftarrow{I}$ are (possibly empty) maximal intervals in $\mathbb{R}_+$ disjoint from I (e.g., if $I = [3,5[$, then $\overleftarrow{I} = [0,3[$, $\overrightarrow{I} = [5,+\infty[$). Note that $\overrightarrow{I}$, $\overleftarrow{I} \in Intv$. Recall that for all $I \in Intv_R$, I is in P if and only if the current time (given by c_{glob}) is in I;
  - $c_> \in Res$; moreover, if $(v_1, ..., v_N, H) = q_0$, then $p_> \in P$ and $\theta_> = \top$, else *either* $p_> \in P$ and $\theta_> = c_> \in]0, +\infty[$, *or* $p_> \notin P$ and $\theta_> = c_> \in [0,0]$;
  - $P \cap \mathcal{H} = \emptyset$ if $p_> \in P$; otherwise, $P \cap \mathcal{H} = H$;
  - for all $x \in SV$ and $v \in V_x$, $past^{\mathsf{s}}_v \in H'$ (resp., $past^{\mathsf{e}}_v \in H'$) if and only if *either* $P \cap Main_{x_i}$ has the form $(\overline{v}, v)$ (resp., $(v, \overline{v})$), *or* $p_> \notin P$ and $past^{\mathsf{s}}_v \in H$ (resp., $past^{\mathsf{e}}_v \in H$). □

Encoding of simple trigger rules by MTL formulas We now build an MTL formula $\varphi_\forall$ over $\mathcal{AP}$ for simple trigger rules in R under the future semantics.

Proposition 12. *An MTL formula $\varphi_\forall$, with maximal constant $O(K_P)$, such that for each multi-timeline Π of SV and encoding w_Π of Π, w_Π is a model of $\varphi_\forall$ if and only if Π satisfies all the simple trigger rules in R, under the future semantics, can be built in linear time. The formula $\varphi_\forall$ has $O(|R| \cdot N_A \cdot N_{\mathcal{E}} \cdot (|Intv_R| + (\sum_{x\in SV}|V_x|)^2))$ distinct subformulas, where N_A (resp., $N_{\mathcal{E}}$) is the maximum number of atoms (resp., existential statements) in a trigger rule of R. The formula $\varphi_\forall$ is an MITL (resp., MITL$_{(0,\infty)}$) formula if the intervals in the trigger rules are non-singular (resp., belong to $Intv_{(0,\infty)}$).*

Proof. We first introduce some auxiliary propositional formulas. Let $x \in SV$ and $v \in V_x$; $\psi(\mathsf{s}, v)$ and $\psi(\mathsf{e}, v)$ are two propositional formulas over $Main_x$ defined as:

$$\psi(\mathsf{s}, v) = (beg_x, v) \vee \bigvee_{u\in V_x}(u, v), \qquad \psi(\mathsf{e}, v) = (v, end_x) \vee \bigvee_{u\in V_x}(v, u).$$

Intuitively, $\psi(\mathsf{s}, v)$ (resp., $\psi(\mathsf{e}, v)$) states that a start-event (resp., end-event) for a token for x with value v occurs at the current time. We also introduce the formula $\psi_{\neg x} = \neg \bigvee_{m\in Main_x} m$ asserting that no event for a token for x occurs at the current time. Finally, given an MTL formula θ, we define the MTL formula $EqTime(\theta) = \theta \vee [\neg p_> \mathsf{U}_{\ge 0}(\neg p_> \wedge \theta)]$, which is satisfied by an encoding of a multi-timeline at the current time if θ eventually holds at a position whose timestamp coincides with the current timestamp.

The MTL formula $\varphi_\forall$ has a conjunct $\varphi_{\mathcal{R}}$ for each trigger rule $\mathcal{R} \in R$. Let $\mathcal{R}$ be a trigger rule of the form $o_t[x_t = v_t] \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$.

Then, we have $\varphi_{\mathcal{R}} = \mathsf{G}_{\geq 0}\big(\psi(\mathsf{s}, v_t) \rightarrow \bigvee_{i=1}^{k} \Phi_{\mathcal{E}_i}\big)$, where $\Phi_{\mathcal{E}_i}$ ensures the fulfillment of $\mathcal{E}_i$ under the future semantics ($1 \leq i \leq k$).

Let $\mathcal{E} \in \{\mathcal{E}_1, \ldots, \mathcal{E}_k\}$, O be the set of token names existentially quantified in $\mathcal{E}$, $\mathbf{A}$ be the set of *interval* atoms in $\mathcal{E}$, and, for each $o \in O$, $val(o)$ be the value of the token referenced by o in the associated quantifier. In the construction of $\Phi_{\mathcal{E}}$, we crucially exploit the assumption that $\mathcal{R}$ is simple: for each token name $o \in O$, there is at most one interval atom in $\mathbf{A}$ where o occurs.

For each token name $o \in \{o_t\} \cup O$, we denote by $Intv_o^{\mathsf{s}}$ (resp., $Intv_o^{\mathsf{e}}$) the set of intervals $J \in Intv$ such that $J = I(\rho)$ for some time-point atom ρ occurring in $\mathcal{E}$, which imposes a time constraint on the start (resp., end) time of the token referenced by o. Note that $Intv_o^{\mathsf{s}}, Intv_o^{\mathsf{e}} \subseteq \mathcal{AP}$, and we exploit the propositional formulas $\xi_o^{\mathsf{s}} = \bigwedge_{I \in Intv_o^{\mathsf{s}}} I$ and $\xi_o^{\mathsf{e}} = \bigwedge_{I \in Intv_o^{\mathsf{e}}} I$ to ensure the fulfillment of the time constraints imposed by the time-point atoms associated with the token o.

The MTL formula $\Phi_{\mathcal{E}}$ (recall that $\mathcal{E} \in \{\mathcal{E}_1, \ldots, \mathcal{E}_k\}$) is defined as follows:

$$\Phi_{\mathcal{E}} = \xi_{o_t}^{\mathsf{s}} \wedge [\psi_{\neg x_t} \mathsf{U}_{\geq 0}(\psi(\mathsf{e}, v_t) \wedge \xi_{o_t}^{\mathsf{e}})] \wedge \bigwedge_{\rho \in \mathbf{A}} \chi_\rho,$$

where, for each $\rho \in \mathbf{A}$, the formula χ_ρ captures the future semantics of ρ.

The construction of χ_ρ depends on the form of ρ. We distinguish 4 cases.

- $\rho = o \leq_I^{e_1,e_2} o_t$ and $o \neq o_t$. We assume $0 \in I$ (the other case being simpler). First, let $e_2 = \mathsf{s}$. Under the future semantics, ρ holds if and only if the start time of the trigger token o_t coincides with the e_1-time of o. Hence, we have that $\chi_\rho = \xi_o^{e_1} \wedge \big(past_{val(o)}^{e_1} \vee EqTime(\psi(e_1, val(o)))\big)$.

  If $e_2 = \mathsf{e}$, then $\chi_\rho = \big[\psi_{\neg x_t} \mathsf{U}_{\geq 0}\{\xi_o^{e_1} \wedge \psi(e_1, val(o)) \wedge \psi_{\neg x_t} \wedge (\psi_{\neg x_t} \mathsf{U}_I \psi(\mathsf{e}, v_t))\}\big] \vee \big[(\psi(e_1, val(o)) \vee past_{val(o)}^{e_1}) \wedge \xi_o^{e_1} \wedge \big(EqTime(\psi(\mathsf{e}, v_t)) \vee (\psi_{\neg x_t} \wedge (\psi_{\neg x_t} \mathsf{U}_I \psi(\mathsf{e}, v_t)))\big)\big] \vee \big[\psi_{\neg x_t} \mathsf{U}_{\geq 0}\{\psi(\mathsf{e}, v_t) \wedge EqTime(\psi(e_1, val(o)) \wedge \xi_o^{e_1})\}\big]$.

  The first disjunct (in square brackets) considers the case where the e_1-event of token o occurs strictly between the start-event and the end-event of the trigger token o_t (along the encoding of a multi-timeline of SV). The second disjunct considers the case where the e_1-event of o precedes the start-event of o_t: under the future semantics, it holds that the e_1-time of o coincides with the start time of o_t. Finally, the third disjunct considers the case where the e_1-event of o follows the end-event of o_t (hence, the corresponding timestamps must coincide).
- $\rho = o_t \leq_I^{e_1,e_2} o$ and $o \neq o_t$. We assume $e_1 = \mathsf{e}$ and $0 \in I$ (the other cases being simpler). Then,

  $$\chi_\rho = \big[\psi_{\neg x_t} \mathsf{U}_{\geq 0}(\psi(\mathsf{e}, u_t) \wedge \mathsf{F}_I(\psi(e_2, val(o)) \wedge \xi_o^{e_2}))\big] \vee \big[\psi_{\neg x_t} \mathsf{U}_{\geq 0}(\psi(\mathsf{e}, u_t) \wedge past_{val(o)}^{e_2} \wedge \xi_o^{e_2})\big],$$

  where the second disjunct deals with the case where the e_2-time of o coincides with the end time of o_t, but the e_2-event of o occurs before the end-event of o_t.
- $\rho = o_t \leq_I^{e_1,e_2} o_t$. This case is straightforward and we omit the details.
- $\rho = o_1 \leq_I^{e_1,e_2} o_2$, $o_1 \neq o_t$, and $o_2 \neq o_t$. We assume $o_1 \neq o_2$ and $0 \in I$ (the other cases being simpler). Then,

  $$\begin{aligned}\chi_\rho = \big[past_{val(o_1)}^{e_1} \wedge \xi_o^{e_1} \wedge \mathsf{F}_I(\psi(e_2, val(o_2)) \wedge \xi_o^{e_2})\big] \vee \\ \big[\mathsf{F}_{\geq 0}\{\psi(e_1, val(o_1)) \wedge \xi_o^{e_1} \wedge \mathsf{F}_I(\psi(e_2, val(o_2)) \wedge \xi_o^{e_2})\}\big] \vee \\ \big[past_{val(o_1)}^{e_1} \wedge \xi_o^{e_1} \wedge past_{val(o_2)}^{e_2} \wedge \xi_o^{e_2}\big] \vee \\ \big[past_{val(o_2)}^{e_2} \wedge \xi_o^{e_2} \wedge EqTime(\psi(e_1, val(o_1)) \wedge \xi_o^{e_1})\big] \vee \\ \big[\mathsf{F}_{\geq 0}\{\psi(e_2, val(o_2)) \wedge \xi_o^{e_2} \wedge EqTime(\psi(e_1, val(o_1)) \wedge \xi_o^{e_1})\}\big].\end{aligned}$$

  The first two disjuncts deal with the cases where (under the future semantics) the e_1-event of o_1 precedes the e_2-event of o_2, while the last three disjuncts consider the dual situation. In the latter three cases, the e_1-time of o_1 and the e_2-time of o_2 are equal.

Note that the MTL formula $\varphi_{\mathcal{V}}$ is an MITL formula (resp., $\mathsf{MITL}_{(0,\infty)}$ formula) if the intervals in the trigger rules are non-singular (resp., belong to $Intv_{(0,\infty)}$). □

Encoding of trigger-less rules by a TA We now deal with trigger-less rules. We start by noting that an existential statement $\mathcal{E}$ in a trigger-less rule requires the existence of an *a priori bounded number* of temporal events satisfying mutual temporal relations (namely, in the worst case, the start time and end time of all tokens associated with some quantifier of $\mathcal{E}$). Thus, we can build a TA for $\mathcal{E}$ which guesses such a chain of events by non-deterministically resetting some clocks, and then checks the temporal relations in $\mathcal{E}$ by suitable clock constraints over the mentioned clocks. By the closure of TA under language union [1], we can then build a TA for the whole trigger-less rule. Additionally, by exploiting the closure of TA under intersection, we build a TA accepting (encodings of) multi-timelines satisfying all trigger-less rules. We refer to Appendix B for the proof of the next proposition, where we show how to build such a TA.

Proposition 13. *A TA $\mathcal{A}_\exists$ over $2^{\mathcal{AP}}$ such that, for each multi-timeline Π of SV and encoding w_Π of Π, w_Π is accepted by $\mathcal{A}_\exists$ if and only if Π satisfies all the trigger-less rules in R can be built in exponential time. $\mathcal{A}_\exists$ has $2^{O(N_q)}$ states, $O(N_q)$ clocks, and maximal constant $O(K_P)$, where N_q is the overall number of quantifiers in the trigger-less rules of R.*

Conclusion of the construction The next theorem follows from Propositions 11, 12, 13, and well-known results about TA and MTL over finite timed words [1,21].

Theorem 14. *The future TP problem with simple trigger rules is decidable (with non-primitive recursive complexity). Moreover, if the intervals in the atoms of the trigger rules are non-singular (resp., belong to $Intv_{(0,\infty)}$), then the problem is in **EXPSPACE** (resp., **PSPACE**).*

Proof. Let $P = (SV, R)$ be an instance of the problem with maximal constant K_P. Moreover, let $N_v = \sum_{x \in SV} |V_x|$, N_q be the overall number of quantifiers in the trigger-less rules of R, and N_A (resp., $N_\mathcal{E}$) be the maximum number of atoms (resp., existential statements) in a trigger rule of R.

By Propositions 11, 12, 13, and the closure of TA under language intersection,

- we can build a TA $\mathcal{A}_P$—namely, the intersection of $\mathcal{A}_{SV}$ from Proposition 11 and $\mathcal{A}_\exists$ from Proposition 13—having $2^{O(N_q+N_v)}$ states, $O(N_q + |SV|)$ clocks, and maximal constant $O(K_P)$,
- and an MTL formula $\varphi_\forall$, with $O(|R| \cdot N_A \cdot N_\mathcal{E} \cdot (|Intv_R| + N_v^2))$ distinct subformulas and maximal constant $O(K_P)$,

such that there exists a future plan for P if and only if $\mathcal{L}_T(\mathcal{A}_P) \cap \mathcal{L}_T(\varphi_\forall) \neq \emptyset$. By [21], checking non-emptiness of $\mathcal{L}_T(\mathcal{A}_P) \cap \mathcal{L}_T(\varphi_\forall)$ is decidable. Thus, the first claim of the theorem holds.

As for the second claim, let us assume the intervals in the trigger rules to be non-singular (resp., to belong to $Intv_{(0,\infty)}$). By Proposition 12, $\varphi_\forall$ is an MITL (resp., $\text{MITL}_{(0,\infty)}$) formula. By [2], one can build a TA $\mathcal{A}_\forall$ accepting $\mathcal{L}_T(\varphi_\forall)$ with $2^{O(K_P \cdot |R| \cdot N_A \cdot N_\mathcal{E} \cdot (|Intv_R| + N_v^2))}$ states, $O(K_P \cdot |R| \cdot N_A \cdot N_\mathcal{E} \cdot (|Intv_R| + N_v^2))$ clocks (resp., $2^{O(|R| \cdot N_A \cdot N_\mathcal{E} \cdot (|Intv_R| + N_v^2))}$ states, $O(|R| \cdot N_A \cdot N_\mathcal{E} \cdot (|Intv_R| + N_v^2))$ clocks), and maximal constant $O(K_P)$. Non-emptiness of a TA $\mathcal{A}$ can be solved by an **NPSPACE** = **PSPACE** search algorithm over the *region automaton* of $\mathcal{A}$ (see [1]), which uses work 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 $\mathcal{A}$ [1]. Thus, since $\mathcal{A}_P$, $\mathcal{A}_\forall$, and the intersection $\mathcal{A}_\wedge$ of $\mathcal{A}_P$ and $\mathcal{A}_\forall$ can be built on the fly—i.e., by looking at their Δ transitions one can determine, given a state q, a successor q' and the connecting transition along with the constraints and clocks to reset—and the search in the region automaton of $\mathcal{A}_\wedge$ can be done without explicitly building $\mathcal{A}_\wedge$, the claim follows. □

As a matter of fact, future TP with simple trigger rules and non-singular intervals in the atoms of trigger rules (resp., intervals in $Intv_{(0,\infty)}$) can be proved to be **EXPSPACE**-complete (resp., **PSPACE**-complete). We refer the reader to C for the corresponding proofs of hardness.

5. TP with trigger-less rules only is NP-complete

In this section, we focus on planning domains where *only trigger-less rules* are allowed, and describe a TP planning algorithm that requires a polynomial number of (non-deterministic) computation steps. Trigger-less rules are useful, for instance, to express initial and intermediate conditions as well as reachability goals. We start showing that there exist instances of the problem that do not admit a *polynomial-size* plan. It immediately follows that the problem cannot be solved by a non-deterministic guessing (and check) of a suitable polynomial-size multi-timeline certificate.

Exponential-size plans for TP with trigger-less rules. Consider the following planning domain. Let $p(i)$ be the i-th prime number, say, $p(1) = 1$, $p(2) = 2$, $p(3) = 3$, $p(4) = 5$,.... For $i = 1, \ldots, n$, we define x_i as the variable $(\{v_i\}, \{(v_i, v_i)\}, D_{x_i})$ with $D_{x_i}(v_i) = [p(i), p(i)]$. The rule $\top \rightarrow \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n]. \bigwedge_{i=1}^{n-1} o_i \leq^{\mathsf{e},\mathsf{e}}_{[0,0]} o_{i+1}$ requires the existence of a "synchronization point", where n tokens (one for each variable) have their ends "aligned". Due to the allowed token durations, the first such time point is $\prod_{i=1}^{n} p(i) \geq 2^{n-1}$. Hence, in any plan, the timeline for x_1 features at least 2^{n-1} tokens that clearly cannot be enumerated in polynomial time.

The above considered TP problem shows that there is no trivial guess-and-check **NP** algorithm for TP with trigger-less rules only. In the following, we provide a (nontrivial) non-deterministic polynomial-time algorithm, proving that the problem actually belongs to **NP** (it can be proved to be **NP**-hard by an easy reduction from the *Hamiltonian path problem*).

We preliminarily have to derive a *bounded horizon* (namely, a bounded end time of the last token) for the plans of a (any) instance of TP with trigger-less rules, that is, if an instance $P = (SV, R)$ admits a plan, then P also has a plan whose horizon is no greater than a given bound. Analogously, we have to fix a *bound on the maximum number of tokens* in a plan. Both bounds can be obtained from the TA constructions done in the proof of Theorem 14. Since only trigger-less rules are allowed, we restrict our attention to the TA $\mathcal{A}_P$, i.e., the intersection of the TA $\mathcal{A}_{SV}$ for the variables in SV (Proposition 11) and the TA $\mathcal{A}_\exists$ for the trigger-less rules in R (Proposition 13). $\mathcal{A}_P$ accepts all and only the encodings w_Π of multi-timelines

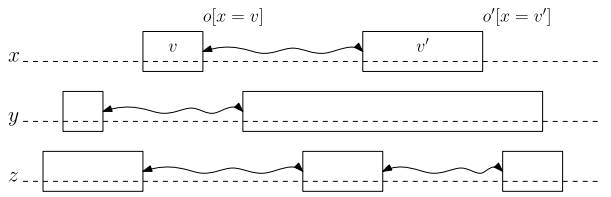

Fig. 4. In a multi-timeline of $SV = \{x, y, z\}$, we non-deterministically place some tokens, which are associated with quantifiers (they are depicted as rectangles).

Π of SV satisfying all the trigger-less rules in R, and it has $\alpha_s = 2^{O(N_q + \sum_{x \in SV} |V_x|)}$ states, $\alpha_c = O(N_q + |SV|)$ clocks, and maximum constant $\alpha_K = O(K_p)$, where N_q is the number of quantifiers in the trigger-less rules of R.

The algorithm to check TA for emptiness applied to $\mathcal{A}_P$ visits the (untimed) region automaton for $\mathcal{A}_P$ [1], which features $\alpha = \alpha_s \cdot O(\alpha_c! \cdot 2^{\alpha_c} \cdot 2^{2N_q^2} \cdot (2\alpha_K + 2)^{\alpha_c})$ states (the factor $2^{2N_q^2}$ accounts for diagonal clock constraints in $\mathcal{A}_P$), looking for a path from the initial state to a final state, whose length is clearly bounded by the number of states. We observe that each edge/transition of the region automaton in such a path corresponds, in the worst case, to the start point of a token for each timeline for the variables in SV (i.e., assuming that all these tokens start simultaneously). This yields a bound on the number of tokens equal to $\alpha \cdot |SV|$. A bound on the horizon of the plan equal to $\alpha \cdot |SV| \cdot (\alpha_K + 1)$ can be derived as well, as each transition taken in $\mathcal{A}_P$ may let pass at most $\alpha_K + 1$ time units, where α_K is the maximum constant to which a (any) clock is compared.

Given these bounds, the proposed algorithm consists of two main steps which rest on the following observations: (i) each trigger-less rule requires the existence of an *a priori bounded number* of temporal events satisfying synchronization relations (in the worst case, the start time and end time of all tokens associated with the quantifiers of one of its existential statements); (ii) apart from these events, timelines for different state variables evolve independently of each other. To deal with (i), we non-deterministically choose and place such temporal events along timelines; as for (ii), in between any pair of "so-positioned" events, we guarantee that each timeline correctly evolves independently of the other ones. See Fig. 4 for a graphical intuition.

Non-deterministic token positioning The algorithm starts by non-deterministically choosing, for each trigger-less rule in R, a disjunct—discharging all the others (if any). Then, for each quantifier $o_i[x_i = v_i]$ of such a disjunct, it *generates* the integer part of both the start and the end time (denoted by $\mathsf{s}_{int}(o_i)$ and $\mathsf{e}_{int}(o_i)$, respectively) of the token associated with o_i. For simplicity, we assume that all quantifiers refer to distinct tokens. It is not difficult to extend the algorithm to possibly associate -by a non-deterministic choice- two or more quantifiers with the same token (it would suffice to rewrite with a common fresh name all the occurrences of quantifiers in the rules associated with the same token). Note that all start/end time $\mathsf{s}_{int}(o_i)$ and $\mathsf{e}_{int}(o_i)$, being less than or equal to $\alpha \cdot |SV| \cdot (\alpha_K + 1)$ (the finite horizon bound), have an integer part that can be encoded with a polynomial number of bits and can be generated in polynomial time. Once the integer part of the start/end time of the tokens has been fixed, to determine the relationship among tokens we have to fix also an ordering of the associated fractional parts. We denote them by $\mathsf{s}_{frac}(o_i)$ and $\mathsf{e}_{frac}(o_i)$. The algorithm non-deterministically generates an *order* of all such fractional parts. In particular, we have to specify, for every token start/end time, whether it is an integer (i.e., $\mathsf{s}_{frac}(o_i) = 0$ / $\mathsf{e}_{frac}(o_i) = 0$) or not (i.e., $\mathsf{s}_{frac}(o_i) > 0$ / $\mathsf{e}_{frac}(o_i) > 0$). Every such combination can be generated in polynomial time.

Some trivial correctness requirements have to be checked, that is, we need to check that each token is assigned an end time greater than or equal to its start time (i.e., for all o_i, $\mathsf{s}_{int}(o_i) \leq \mathsf{e}_{int}(o_i)$), and two different tokens for the same variable do not overlap. Since all the constants appearing in atoms are integers, it is routine to prove that, if we choose two different assignments of the start/end time of tokens associated with quantifiers which are indistinguishable for (i) all the integer parts, (ii) zeroness/non-zeroness of the fractional parts, and (iii) the order of the fractional parts, then the satisfaction of the atoms in the trigger-less rules does not change for the two different assignments.

Enforcing legal token durations and timeline evolutions The second step of the algorithm checks that (1) all the tokens previously associated with a quantifier have a legal duration, and (2) there is a legal timeline evolution between pairs of adjacent tokens associated with quantifiers over the same variable (two tokens are said *adjacent* when there is not another token in the timeline associated with a quantifier in between them). All the requirements are expressed as constraints of a *linear problem*, which can be solved in deterministic polynomial time (e.g., using the ellipsoid algorithm). When needed, we use *strict inequalities*, which are not allowed, in general, in linear programs. We shall describe later how to convert strict inequalities into "equivalent" non-strict ones.

We start by associating non-negative variables $\alpha_{o_i,s}, \alpha_{o_i,e}$ with the fractional parts of the start/end times $\mathsf{s}_{frac}(o_i)$, $\mathsf{e}_{frac}(o_i)$ of each token for a quantifier $o_i[x_i = v_i]$ adding the linear constraints $0 \leq \alpha_{o_i,s} < 1$ and $0 \leq \alpha_{o_i,e} < 1$. In addition, we introduce linear constraints to ensure that the values of $\alpha_{o_i,s}$ and $\alpha_{o_i,e}$ respect the order of the fractional parts guessed in the first step. Next, to enforce requirement (1), for any quantifier $o_i[x_i = v_i]$, we set $a \leq (\mathsf{e}_{int}(o_i) + \alpha_{o_i,e}) - (\mathsf{s}_{int}(o_i) + \alpha_{o_i,s}) \leq b$, where $D_{x_i}(v_i) = [a, b]$. Clearly, strict ($<$) inequalities must be used for a left/right open interval.

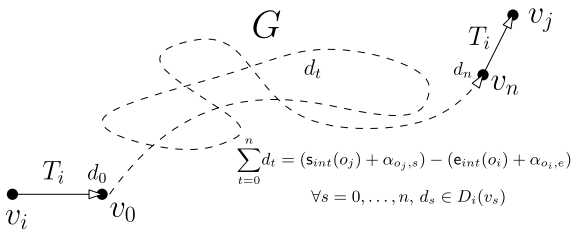

Fig. 5. A legal timeline evolution between a pair of tokens for $o_i[x_i = v_i]$ and $o_j[x_i = v_j]$.

In the following we focus on the existence of a legal timeline evolution between a pair of adjacent tokens associated with quantifiers, say, $o_i[x_i = v_i]$ and $o_j[x_i = v_j]$ (requirement (2)). The case of a legal timeline evolution leading to the first token associated with a quantifier can be treated analogously. Each state variable $x_i = (V_i, T_i, D_i)$ can be seen as a directed graph $G = (V_i, E_i)$, with $E_i = \{(u, v) \mid u \in V_i, v \in T_i(u)\}$ and D_i a function associating with each vertex $v \in V_i$ a duration range. We have to decide if there exist (*i*) a path in G, $v_0 \cdot v_1 \cdots v_{n-1} \cdot v_n$, where vertices can occur repeatedly, and $v_0 \in T_i(v_i)$ and v_n, with $v_j \in T_i(v_n)$, are non-deterministically generated, and (*ii*) a list of non-negative real values $d_0, \ldots, d_n$ such that $\sum_{t=0}^{n} d_t = (\mathsf{s}_{int}(o_j) + \alpha_{o_j,s}) - (\mathsf{e}_{int}(o_i) + \alpha_{o_i,e})$, and for all $s = 0, \ldots, n$, $d_s \in D_i(v_s)$. See Fig. 5 for a graphical hint.

To find the solution path, we guess a set of integers $P_{\alpha'} = \{\alpha'_{u,v} \mid (u, v) \in E_i\}$. Intuitively, $\alpha'_{u,v} \geq 0$ is the number of times the solution path traverses the edge (u, v) in G. Since every time an edge is traversed a new token starts, each $\alpha'_{u,v}$ is bounded by the number of tokens, i.e., by $\alpha \cdot |SV|$ and then its binary encoding of can be generated in polynomial time. Now, we have to check whether there is a path in G from v_0 to v_n where the set of edges in the path is $P_{\alpha',>} = \{(u, v) \mid \alpha'_{u,v} > 0\}$ and the number of occurrences of an edge (u, v) in the path is precisely $\alpha'_{u,v}$ (we call it a $P_{\alpha'}$-path). The solution of the problem is obtained by considering an Eulerian path problem in a multigraph $G_{P_{\alpha'}} = (V_i, E'_i)$, where E'_i is a multiset of edges such that the underling set of edges is $P_{\alpha',>}$ and the multiplicity of each edge $(u, v) \in P_{\alpha',>}$ is $\alpha'_{u,v}$. Clearly, there is a $P_{\alpha'}$-path from v_0 to v_n in G if and only if there exists an Eulerian path from v_0 to v_n in $G_{P_{\alpha'}}$. The existence of the Eulerian path can expressed using linear constraints by exploiting the following characterization.

Theorem 15. *[17] Let $G' = (V', E')$ be a directed multigraph (E' is a multiset). G' has a (directed) Eulerian path from v_0 to v_n iff: (i) the undirected version of G' is connected and (ii) $|\{(u, v) \in E'\}| = |\{(v, w) \in E'\}|$ for all $v \in V' \setminus \{v_0, v_n\}$; (iii) $|\{(u, v_0) \in E'\}| = |\{(v_0, w) \in E'\}| - 1$; (iv) $|\{(u, v_n) \in E'\}| = |\{(v_n, w) \in E'\}| + 1$.*

Then, we perform the following 4 deterministic steps.

1. We consider the subset $E' = \{(u, v) \in T_i \mid \alpha'_{u,v} > 0\}$ of edges of G, and we check if E' induces a strongly (undirected) connected subgraph of G.
2. We check if (i) $\sum_{(u,v_0)\in E'} \alpha'_{u,v_0} = \sum_{(v_0,w)\in E'} \alpha'_{v_0,w} - 1$, (ii) $\sum_{(u,v_n)\in E'} \alpha'_{u,v_n} = \sum_{(v_n,w)\in E'} \alpha'_{v_n,w} + 1$, (iii) $\sum_{(u,v)\in E'} \alpha'_{u,v} = \sum_{(v,w)\in E'} \alpha'_{v,w}$ for $v \in V_i \setminus \{v_0, v_n\}$.
3. For all $v \in V_i \setminus \{v_0\}$, we define $y_v = \sum_{(u,v)\in E'} \alpha'_{u,v}$ (y_v is the number of times the solution path gets into v). Moreover, $y_{v_0} = \sum_{(v_0,u)\in E'} \alpha'_{v_0,u}$.
4. We define the non-negative real variables z_v, for every $v \in V_i$ (z_v is the total waiting time of the path on the node v), subject to the constraints $a \cdot y_v \leq z_v \leq b \cdot y_v$, where $D_i(v) = [a, b]$ (an analogous constraint is added for open intervals). Finally, we set $\sum_{v\in V_i} z_v = (\mathsf{s}_{int}(o_j) + \alpha_{o_j,s}) - (\mathsf{e}_{int}(o_i) + \alpha_{o_i,e})$.

Steps 1 and 2 check that the chosen values $\alpha'_{u,v}$ allow *a directed Eulerian path from v_0 to v_n in a multigraph*, as guaranteed by Theorem 15. Steps 3 and 4 evaluate the waiting times of the path in some vertex v with duration interval $[a, b]$. If the solution path visits the vertex y_v times, every visit must take at least a and at most b time units. Hence, the overall visiting time is in between $a \cdot y_v$ and $b \cdot y_v$. Vice versa, if the total visiting time is in between $a \cdot y_v$ and $b \cdot y_v$, it can be split into y_v intervals, each one falling into $[a, b]$.

The algorithm ends by solving the linear program over the variables $\alpha_{o_i,s}$, $\alpha_{o_i,e}$, for each quantifier $o_i[x_i = v_i]$, and the variables z_v, for each pair of adjacent tokens in the same timeline for x_i and $v \in V_i$, subject to the respective constraints. To conform to linear programming, we have to replace all strict inequalities with non-strict ones. It is straightforward to observe that all constraints involving strict inequalities we have are of (or can easily be converted into) the following forms: $\xi s < \eta q + k$ or $\xi s > \eta q + k$, where s and q are variables, and ξ, η, k are constants. We replace them, respectively, by $\xi s - \eta q - k + \beta_t \leq 0$ and $\xi s - \eta q - k - \beta_t \geq 0$, where β_t is an additional fresh non-negative variable, which is *local* to a single constraint. The original inequality and the new one are equivalent if and only if β_t is a small enough *positive* number. Moreover, we add another non-negative variable, say r, which is subject to a constraint $r \leq \beta_t$, for each of the introduced variables β_t (i.e., r is less than or equal to the minimum of all β_t's). Finally, we maximize the value of r when solving the linear program. We have that $\max r > 0$ if and only if there is an admissible solution where the values of all β_t's are

positive (and thus the original strict inequalities hold true). This ends the description of the algorithm which proves that the TP problem with trigger-less rules is in **NP**.

Hardness can be easily proved by a reduction from the problem of the existence of a hamiltonian path in a directed graph.

Proposition 16. *The TP problem with trigger-less rules only is **NP**-hard (under polynomial-time reductions).*

Proof. Let $G = (V, E)$, with $|V| = n$, be a directed graph. We let $x = (V_x, T_x, D_x)$, where $V_x = V$, T_x encodes E, and $D_x(v) = [1, 1]$ for all $v \in V_x$, and we add the following trigger-less rules, one for each $v \in V_x$: $\top \rightarrow \exists o[x = v].o \geq^{\mathsf{s}}_{[0,n-1]} 0$. The rule for $v \in V$ requires that there is a token $(x, v, 1)$ in the timeline for x, which starts no later than $n - 1$. It is easy to check that G contains a hamiltonian path iff there is a plan for the defined planning domain. □

The following theorem immediately follows.

Theorem 17. *The TP problem with trigger-less rules only is **NP**-complete.*

6. Conclusions and future work

In this paper, we addressed the TP problem over dense temporal domains. Timelines over discrete time have been extensively and fruitfully used in temporal planning. Moving from discrete to dense time is important for expressiveness: dense time allows one not to deal with unnecessary (or even artificially "forced") details and to properly express properties such as accomplishments, actions with duration, and temporal aggregates. Since TP over dense time turns out to be undecidable in its general form, we have identified and studied "intermediate" decidable cases of the problem, which suitably constrain the structure of synchronization rules. By restricting also the type of intervals used in trigger rules, better complexity results (**EXPSPACE** and **PSPACE**) can be obtained. Finally, if only trigger-less rules are allowed, TP over dense time becomes **NP**-complete. As for future work, we shall investigate open issues pointed out in Table 1.

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.

Acknowledgements

The research has been supported by projects *Formal methods for verification and synthesis of discrete and hybrid systems* (GNCS), and *ENCASE—Efforts in the uNderstanding of Complex interActing SystEms* (PRID).

Appendix A. Non-primitive recursive-hardness of future TP

Theorem *(9). The future TP problem is non-primitive recursive-hard also under one of the following two assumptions:* either (1) *the trigger rules are simple,* or (2) *the intervals (occurring in atoms or constraint functions) are in* $Intv_{(0,\infty)}$.

The proof is by a polynomial-time reduction from the halting problem for *gainy counter machines* [11], a variant of Minsky machines, whose counters may erroneously increase. The machine is a tuple $M = (Q, q_{init}, q_{halt}, n, \Delta)$, where:

- Q is a finite set of (control) locations/states, $q_{init} \in Q$ is the initial location, and $q_{halt} \in Q$ is the halting location,
- $n \in \mathbb{N} \setminus \{0\}$ is the number of counters of M, and
- $\Delta \subseteq Q \times \mathsf{L} \times Q$ is a transition relation over the instruction set $\mathsf{L} = \{\mathsf{inc}, \mathsf{dec}, \mathsf{zero}\} \times \{1, \ldots, n\}$.

We adopt the following notational conventions. For an instruction $op \in \mathsf{L}$, let $c(op) \in \{1, \ldots, n\}$ be the counter associated with op. For a transition $\delta \in \Delta$ of the form $\delta = (q, op, q')$, we define $from(\delta) = q$, $op(\delta) = op$, $c(\delta) = c(op)$, and $to(\delta) = q'$. We denote by op_{init} the instruction $(\mathsf{zero}, 1)$. W.l.o.g., we make these assumptions: (i) for each transition $\delta \in \Delta$, $from(\delta) \neq q_{halt}$ and $to(\delta) \neq q_{init}$; (ii) there is exactly one transition in Δ, denoted δ_{init}, having as source the initial location q_{init}. An *M-configuration* is a pair (q, ν) consisting of a location $q \in Q$ and a counter valuation $\nu : \{1, \ldots, n\} \rightarrow \mathbb{N}$. Given two valuations ν and ν', we write $\nu \geq \nu'$ if and only if $\nu(c) \geq \nu'(c)$ for all $c \in \{1, \ldots, n\}$.

Under the *exact semantics* (with no errors), M induces a transition relation, denoted by $\longrightarrow$, over pairs of M-configurations and instructions, defined as follows: for configurations (q, ν) and (q', ν'), and instructions $op \in \mathsf{L}$, we have $(q, \nu) \xrightarrow{op} (q', \nu')$ if the following holds, where $c \in \{1, \ldots, n\}$ is the counter associated with the instruction op:

- $(q, op, q') \in \Delta$ and $\nu'(c') = \nu(c')$ for all $c' \in \{1, \ldots, n\} \setminus \{c\}$;

- $\nu'(c) = \nu(c) + 1$ if $op = (\mathsf{inc}, c)$;
- $\nu'(c) = \nu(c) - 1$ if $op = (\mathsf{dec}, c)$ (in particular, it has to be $\nu(c) > 0$);
- $\nu'(c) = \nu(c) = 0$ if $op = (\mathsf{zero}, c)$.

The *gainy semantics* is obtained from the exact one by allowing *increment* errors. Formally, M induces a transition relation, denoted by $\longrightarrow_{gainy}$, defined as follows: for configurations (q, ν) and (q', ν'), and instructions $op \in \mathsf{L}$, we have $(q, \nu) \xrightarrow{op}_{gainy} (q', \nu')$ if the following holds, where $c = c(op)$ is the counter associated with the instruction op: $(q, \nu) \xrightarrow{op}_{gainy} (q', \nu')$ iff there are valuations ν_+ and ν'_+ such that $\nu_+ \geq \nu$, $(q, \nu_+) \xrightarrow{op} (q', \nu'_+)$, and $\nu' \geq \nu'_+$. Equivalently, $(q, \nu) \xrightarrow{op}_{gainy} (q', \nu')$ iff the following conditions hold:

- $(q, op, q') \in \Delta$ and $\nu'(c') \geq \nu(c')$ for all $c' \in \{1, \ldots, n\} \setminus \{c\}$;
- $\nu'(c) \geq \nu(c) + 1$ if $op = (\mathsf{inc}, c)$;
- $\nu'(c) \geq \nu(c) - 1$ if $op = (\mathsf{dec}, c)$;
- $\nu(c) = 0$ if $op = (\mathsf{zero}, c)$.

A (gainy) *M-computation* is a finite sequence of the form:

$$(q_0, \nu_0) \xrightarrow{op_0}_{gainy} (q_1, \nu_1) \xrightarrow{op_1}_{gainy} \cdots \xrightarrow{op_{k-1}}_{gainy} (q_k, \nu_k).$$

M halts if there exists an M-computation starting at the *initial* configuration (q_{init}, ν_{init}), where $\nu_{init}(c) = 0$ for all $c \in \{1, \ldots, n\}$, and leading to some halting configuration (q_{halt}, ν). Given a gainy counter machine M, *the halting problem for M* is to decide whether M halts, and it was shown to be decidable and non-primitive recursive [11].

We now prove the following result, from which Theorem 9 directly follows.

Proposition 18. *One can construct in polynomial time a TP domain $P = (\{x_M\}, R_M)$ where the trigger rules in R_M are simple (resp., the intervals in P are in $Intv_{(0,\infty)}$) such that M halts iff there is a future plan for P.*

Proof. We focus on the reduction where the intervals in P are in $Intv_{(0,\infty)}$. At the end of the proof, we show how to adapt the construction for the case of simple trigger rules with arbitrary intervals.

Encoding of M-computations. First, we define a suitable encoding of a computation of M as a timeline for x_M. For this, we exploit the finite set of symbols $V = V_{main} \cup V_{sec} \cup V_{dummy}$ corresponding to the finite domain of the state variable x_M. The set of *main* values V_{main} is given by

$$V_{main} = \{(\delta, op) \in \Delta \times \mathsf{L} \mid op \neq (\mathsf{inc}, c) \text{ if } op(\delta) = (\mathsf{zero}, c)\}.$$

Intuitively, in the encoding, a main value (δ, op) keeps track of the transition δ used in the current step of the computation, while op represents the instruction exploited in the previous computation step (if any). The set of *secondary* values V_{sec} is defined as $V_{sec} = V_{main} \times \{1, \ldots, n\} \times 2^{\{\#_{\mathsf{inc}}, \#_{\mathsf{dec}}\}}$, where $\#_{\mathsf{inc}}$ and $\#_{\mathsf{dec}}$ are two special symbols used as markers. V_{sec} is used for encoding counter values, as shown later. Finally, the set of *dummy values* is $V_{dummy} = (V_{main} \cup V_{sec}) \times \{dummy\}$; their use will be clear when we introduce synchronization rules: they are used to specify punctual time constraints by means of non-simple trigger rules over intervals in $Intv_{(0,\infty)}$.

Given a word $w \in V^*$, we denote by $||w||$ the length of the word obtained from w by removing dummy symbols. For $c \in \{1, \ldots, n\}$ and $v_{main} = (\delta, op) \in V_{main}$, the *set* $Tag(c, v_{main})$ *of markers of counter c for the main value* v_{main} is the subset of $\{\#_{\mathsf{inc}}, \#_{\mathsf{dec}}\}$ defined as follows: (*i*) $\#_{\mathsf{inc}} \in Tag(c, v_{main})$ iff $op = (\mathsf{inc}, c)$; (*ii*) $\#_{\mathsf{dec}} \in Tag(c, v_{main})$ iff $op(\delta) = (\mathsf{dec}, c)$;

A *c-code for the main value* $v_{main} = (\delta, op)$ is a finite word w_c over V such that *either* (*i*) w_c is empty and $\#_{\mathsf{inc}} \notin Tag(c, v_{main})$, *or* (*ii*) $op(\delta) \neq (\mathsf{zero}, c)$ and $w_c = (v_{main}, c, Tag(c, v_{main}))(v_{main}, c, \emptyset, dummy)^{h_0} \cdot (v_{main}, c, \emptyset) \cdot (v_{main}, c, \emptyset, dummy)^{h_1} \cdots (v_{main}, c, \emptyset) \cdot (v_{main}, c, \emptyset, dummy)^{h_n}$ for some $n \geq 0$ and $h_0, h_1, \ldots, h_n \geq 0$. The c-code w_c encodes the value for the counter c given by $||w_c||$. Intuitively, w_c can be seen as an interleaving of secondary values with dummy ones (they have technical use without encoding any counter value).

A *configuration-code* w *for a main value* $v_{main} = (\delta, op) \in V_{main}$ is a finite word over V of the form $w = v_{main} \cdot (v_{main}, dummy)^h \cdot w_1 \cdots w_n$, where $h \geq 0$ and for each counter $c \in \{1, \ldots, n\}$, w_c is a c-code for the main value v_{main}. The configuration-code w encodes the M-configuration $(from(\delta), \nu)$, where $\nu(c) = ||w_c||$ for all $c \in \{1, \ldots, n\}$. Note that if $op(\delta) = (\mathsf{zero}, c)$, then $\nu(c) = 0$ and $op \neq (\mathsf{inc}, c)$. The marker $\#_{\mathsf{inc}}$ occurs in w iff op is an increment instruction, and in such a case $\#_{\mathsf{inc}}$ marks the *first* symbol of the encoding $w_{c(op)}$ of counter $c(op)$. Intuitively, if the operation performed in the previous step increments counter c, then the tag $\#_{\mathsf{inc}}$ "marks" the unit of the counter c in the current configuration which has been added by the increment.

The marker $\#_{\mathsf{dec}}$ occurs in w iff δ is a decrement instruction and the value of counter $c(\delta)$ in w is non-zero; in such a case, $\#_{\mathsf{dec}}$ marks the *first* symbol of the encoding $w_{c(\delta)}$ of counter $c(\delta)$. Intuitively, if the operation to be performed in the current step decrements counter c and the current value of c is non-zero, then the tag $\#_{\mathsf{dec}}$ marks the unit of the counter c in the current configuration which has to be removed by the decrement.

A *computation*-code is a sequence of configuration-codes $\pi = w_{(\delta_0,op_0)} \cdots w_{(\delta_k,op_k)}$, where, for all $0 \leq i \leq k$, $w_{(\delta_i,op_i)}$ is a configuration-code with main value (δ_i, op_i), and whenever $i < k$, it holds that $to(\delta_i) = from(\delta_{i+1})$ and $op(\delta_i) = op_{i+1}$. Note that by our assumptions $to(\delta_i) \neq q_{halt}$ for all $0 \leq i < k$, and $\delta_j \neq \delta_{init}$ for all $0 < j \leq k$. The computation-code π is *initial* if the first configuration-code $w_{(\delta_0,op_0)}$ is $(\delta_{init}, op_{init})$ (which encodes the initial configuration), and it is *halting* if for the last configuration-code $w_{(\delta_k,op_k)}$ in π, it holds that $to(\delta_k) = q_{halt}$. For all $0 \leq i \leq k$, let (q_i, ν_i) be the M-configuration encoded by the configuration-code $w_{(\delta_i,op_i)}$ and $c_i = c(\delta_i)$. The computation-code π is *well-formed* if, additionally, for all $0 \leq j \leq k-1$, the following conditions hold:

- $\nu_{j+1}(c) \geq \nu_j(c)$ for all $c \in \{1, \ldots, n\} \setminus \{c_j\}$ (*gainy monotonicity*);
- $\nu_{j+1}(c_j) \geq \nu_j(c_j) + 1$ if $op(\delta_j) = (\mathsf{inc}, c_j)$ (*increment requirement*);
- $\nu_{j+1}(c_j) \geq \nu_j(c_j) - 1$ if $op(\delta_j) = (\mathsf{dec}, c_j)$ (*decrement requirement*).

Clearly, M halts *iff* there is 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 any *future plan* for $P = (\{x_M\}, R_M)$ is an initial and halting well-formed computation-code. Thus, M halts if and only if there is a future plan of P.

Formally, the state variable x_M is given by $x_M = (V, T, D)$ where, for each $v \in V$, $D(v) =]0, \infty[$ if $v \notin V_{dummy}$, and $D(v) = [0, \infty[$ otherwise: we require that the duration of a non-dummy token is always greater than zero (*strict time monotonicity*). The value transition function T of x_M ensures the following.

Claim 19. The untimed part of any timeline for x_M whose first token has value $(\delta_{init}, op_{init})$ corresponds to a prefix of some initial computation-code. Moreover, $(\delta_{init}, op_{init}) \notin T(v)$ for all $v \in V$.

The definition of T is easy and omitted. Let $V_{halt} = \{(\delta, op) \in V_{main} \mid to(\delta) = q_{halt}\}$. By Claim 19 and the assumption that $from(\delta) \neq q_{halt}$ for each transition $\delta \in \Delta$, to ensure the initialization and halting requirements, it suffices to enforce the timeline to feature a token with value $(\delta_{init}, op_{init})$ and a token with value in V_{halt}. This is captured by the trigger-less rules $\top \to \exists o[x_M = (\delta_{init}, op_{init})].\top$ and $\top \to \bigvee_{v \in V_{halt}} \exists o[x_M = v].\top$.

The well-formedness requirement is captured by the trigger rules in R_M which express the following punctual time constraints.

- *2-Time distance between consecutive main values:* the overall duration of the sequence of tokens corresponding to a configuration-code amounts exactly to 2 time units. By Claim 19, 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 $\mathsf{s}(tk') - \mathsf{s}(tk) = 2$. To this aim, for each $v \in V_{main} \setminus V_{halt}$, we have the following non-simple trigger rule with intervals in $Intv_{(0,\infty)}$ which uses a dummy token for capturing the punctual time constraint:

$$o[x_M = v] \to \bigvee_{u \in V_{main}} \bigvee_{u_d \in V_{dummy}} \exists o'[x_M = u] \exists o_d[x_M = u_d].o \leq^{\mathsf{s},\mathsf{s}}_{[1,+\infty[} o_d \wedge o_d \leq^{\mathsf{s},\mathsf{s}}_{[1,+\infty[} o' \wedge o \leq^{\mathsf{s},\mathsf{s}}_{[0,2]} o'.$$

- For a counter $c \in \{1, \ldots, n\}$, let us denote as $V_c \subseteq V_{sec}$ the set of secondary values given by $V_{main} \times \{c\} \times 2^{\{\#_{\mathsf{inc}}, \#_{\mathsf{dec}}\}}$. We require that each token tk with a V_c-value of the form $((\delta, op), c, Tag)$ such that $c \neq c(\delta)$ and $to(\delta) \neq q_{halt}$ is eventually followed by a token tk' with a V_c-value such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 2$. Note that our encoding, Claim 19, strict time monotonicity, and 2-Time distance between consecutive main values guarantee that the previous requirement captures *gainy monotonicity*. Thus, for each counter c and $v \in V_c$ such that v is of the form $((\delta, op), c, Tag)$, where $c \neq c(\delta)$ and $to(\delta) \neq q_{halt}$, we have the following non-simple trigger rule over $Intv_{(0,\infty)}$:

$$o[x_M = v] \to \bigvee_{u \in V_c} \bigvee_{u_d \in V_{dummy}} \exists o'[x_M = u] \exists o_d[x_M = u_d].o \leq^{\mathsf{s},\mathsf{s}}_{[1,+\infty[} o_d \wedge o_d \leq^{\mathsf{s},\mathsf{s}}_{[1,+\infty[} o' \wedge o \leq^{\mathsf{s},\mathsf{s}}_{[0,2]} o'.$$

- For capturing the increment and decrement requirements, by construction, we enforce that: (i) each token tk with a V_c-value of the form $((\delta, op), c, Tag)$ such that $to(\delta) \neq q_{halt}$ and $\delta = (\mathsf{inc}, c)$ is eventually followed by a token tk' with a V_c-value which is *not* marked by $\#_{\mathsf{inc}}$ such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 2$; (ii) each token tk with a V_c-value of the form $((\delta, op), c, Tag)$ such that $to(\delta) \neq q_{halt}$, $\delta = (\mathsf{dec}, c)$, and $\#_{\mathsf{dec}} \notin Tag$ is eventually followed by a token tk' with a V_c-value such that $\mathsf{s}(tk') - \mathsf{s}(tk) = 2$. These requirements can be expressed by non-simple trigger rules with intervals in $Intv_{(0,\infty)}$ similar to the previous ones.

Finally, to prove Proposition 18 for the case of simple trigger rules with arbitrary intervals, it suffices to remove the dummy values and replace the conjunction $o \leq^{\mathsf{s},\mathsf{s}}_{[1,+\infty[} o_d \wedge o_d \leq^{\mathsf{s},\mathsf{s}}_{[1,+\infty[} o' \wedge o \leq^{\mathsf{s},\mathsf{s}}_{[0,2]} o'$ in the previous trigger rules with the "punctual" atom $o \leq^{\mathsf{s},\mathsf{s}}_{[2,2]} o'$, whose interval at the subscript is singular.

This concludes the proof of Proposition 18. □

Appendix B. Proof of Proposition 13

Proposition 13. *A TA $\mathcal{A}_\exists$ over $2^{\mathcal{AP}}$ such that, for each multi-timeline Π of SV and encoding w_Π of Π, w_Π is accepted by $\mathcal{A}_\exists$ if and only if Π satisfies all the trigger-less rules in R can be built in exponential time. $\mathcal{A}_\exists$ has $2^{O(N_q)}$ states, $O(N_q)$ clocks, and maximal constant $O(K_P)$, where N_q is the overall number of quantifiers in the trigger-less rules of R.*

Proof. Let $\mathcal{E}$ be an existential statement for SV such that no token name appears free in $\mathcal{E}$. We first show how to construct a TA $\mathcal{A}_\mathcal{E}$ over $2^{\mathcal{AP}}$ such that for each multi-timeline Π of SV and encoding w_Π of Π, w_Π is accepted by $\mathcal{A}_\mathcal{E}$ iff Π satisfies $\mathcal{E}$. Then, we exploit the well-known closure of TA under language union and language intersection to prove the proposition.

Let O be the set of token names existentially quantified in the existential statement $\mathcal{E}$ and, for each $o \in O$, let $val(o)$ be the value of the token referenced by o in the associated quantifier. For each token name $o \in O$, we denote by $Intv_o^\mathsf{s}$ (resp., $Intv_o^\mathsf{e}$) the set of intervals $J \in Intv$ such that $J = I(\rho)$ for some time-point atom ρ occurring in $\mathcal{E}$ which imposes a time constraint on the start time (resp., end time) of the token referenced by o.

We first outline the construction of $\mathcal{A}_\mathcal{E}$. We associate two clocks with each token name $o \in O$, namely c_o^s and c_o^e which, intuitively, are reset when the token chosen for o starts and ends, respectively. The clocks c_o^s and c_o^e are non-deterministically reset when a start-event for $val(o)$ and the related end-event occur along an encoding of a multi-timeline. The automaton $\mathcal{A}_\mathcal{E}$ ensures that the clocks c_o^s and c_o^e are reset exactly once. $\mathcal{A}_\mathcal{E}$ moves to an accepting state only if all the clocks c_o^s and c_o^e for each $o \in O$ have been reset and the time constraints that encode the interval atoms in $\mathcal{E}$ are fulfilled. To deal with time-point atoms, we also exploit, like in the previous proofs, a global clock c_{glob} which measures the current time and is never reset: whenever the clock c_o^s (resp., c_o^e) is reset, we require that the clock constraint $\bigwedge_{I \in Intv_o^\mathsf{s}} c_{glob} \in I$ (resp., $\bigwedge_{I \in Intv_o^\mathsf{e}} c_{glob} \in I$) is fulfilled. The TA $\mathcal{A}_\mathcal{E} = (2^{\mathcal{AP}}, Q, q_0, C, \Delta, F)$ is formally defined as follows.

- The set C of clocks is $\{c_{glob}\} \cup \bigcup_{o \in O} \{c_o^\mathsf{s}, c_o^\mathsf{e}\}$.
- The set of states is $2^{C \setminus \{c_{glob}\}}$ and q_0 is $\emptyset$. Intuitively, a state keeps track of the clocks in $C \setminus \{c_{glob}\}$ which have been reset so far.
- The set of final states F is given by the singleton $\{C \setminus \{c_{glob}\}\}$. In such a state all clocks different from c_{glob} have been reset.
- The transition relation Δ consists of the transitions $(C_1, P, \theta \wedge \theta_{glob}, Res, C_2)$ such that *either* (i) $C_1 = C \setminus \{c_{glob}\}$, $C_2 = C_1$, $Res = \emptyset$, $\theta = \top$, and $\theta_{glob} = \top$ (intuitively $\mathcal{A}_\mathcal{E}$ loops unconditionally in its final state), *or* (ii) $C_1 \subset C \setminus \{c_{glob}\}$, $C_2 \supseteq C_1$ ($\mathcal{A}_\mathcal{E}$ has not reached its final state yet), and the following conditions hold:
  - for each $c_o^\mathsf{s} \in C_2 \setminus C_1$, there is a main proposition in P of the form $(v', val(o))$ for some v'.
  - for each $o \in O$, $c_o^\mathsf{e} \in C_2 \setminus C_1$ if and only if $c_o^\mathsf{s} \in C_1$ and $(val(o), v') \in P$ for some v'.
  - if $C_2 \subset C \setminus \{c_{glob}\}$ (in this case $\mathcal{A}_\mathcal{E}$ is not transitioning to its final state), then $\theta = \top$.
    Conversely, if $C_2 = C \setminus \{c_{glob}\}$ (here $\mathcal{A}_\mathcal{E}$ moves to the final state), then $\theta = \bigwedge_{\rho \in \mathbf{A}} code(\rho)$, where $\mathbf{A}$ is the set of *interval* atoms of $\mathcal{E}$ and for each interval atom $\rho \in \mathbf{A}$ of the form $o_1 \leq_I^{e_1, e_2} o_2$, the clock constraint $code(\rho)$ is defined as follows:
    - if $c_{o_2}^{e_2} \notin C_1$ and $c_{o_1}^{e_1} \notin C_1$, then $code(\rho) = c_{o_2}^{e_2} - c_{o_1}^{e_1} \in I$ (in this case, both $c_{o_2}^{e_2}$ and $c_{o_1}^{e_1}$ are reset simultaneously by the transition to the final state C_2, meaning that o_2's e_2-event and o_1's e_1-event have the same timestamp; hence it must be that $c_{o_2}^{e_2} - c_{o_1}^{e_1} = 0 \in I$ for the atom to be satisfied);
    - if $c_{o_2}^{e_2} \in C_1$ and $c_{o_1}^{e_1} \in C_1$, then $code(\rho) = c_{o_1}^{e_1} - c_{o_2}^{e_2} \in I$;
    - if $c_{o_2}^{e_2} \in C_1$ and $c_{o_1}^{e_1} \notin C_1$, then $code(\rho) = c_{o_2}^{e_2} \in [0, 0] \wedge c_{o_2}^{e_2} \in I$ (o_2's e_2-event and o_1's e_1-event must have the same timestamp; as before, it must be that $0 \in I$);
    - if $c_{o_2}^{e_2} \notin C_1$ and $c_{o_1}^{e_1} \in C_1$, then $code(\rho) = c_{o_1}^{e_1} \in I$.
  - $\theta_{glob} = \bigwedge_{c_o^e \in C_2 \setminus C_1} \bigwedge_{I \in Intv_o^e} c_{glob} \in I$ and $Res = C_2 \setminus C_1$.

Note that $\mathcal{A}_\mathcal{E}$ has $2^{O(m)}$ states, $O(m)$ clocks and maximal constant $O(K)$, with m the number of quantifiers in $\mathcal{E}$ and K the maximal constant in $\mathcal{E}$.

Given a trigger-less rule $\mathcal{R} = \top \rightarrow \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$, we construct the TA $\mathcal{A}_\mathcal{R}$ resulting from the union of the automata $\mathcal{A}_{\mathcal{E}_1}, \ldots, \mathcal{A}_{\mathcal{E}_k}$. Then the TA $\mathcal{A}_\exists$ is obtained as intersection of the automata $\mathcal{A}_\mathcal{R}$, for all $\mathcal{R} \in R$ being trigger-less rules. By [1], $\mathcal{A}_\exists$ has $2^{O(N_q)}$ states, $O(N_q)$ clocks, and maximal constant $O(K_P)$, where N_q is the overall number of quantifiers in the trigger-less rules of R. □

Appendix C. Future TP with simple trigger rules and non-singular intervals: hardness

In this section, we first consider the future TP problem with simple trigger rules and non-singular intervals, and prove that it is **EXPSPACE**-hard by a polynomial-time reduction from the *domino-tiling problem for grids with rows of single exponen-*

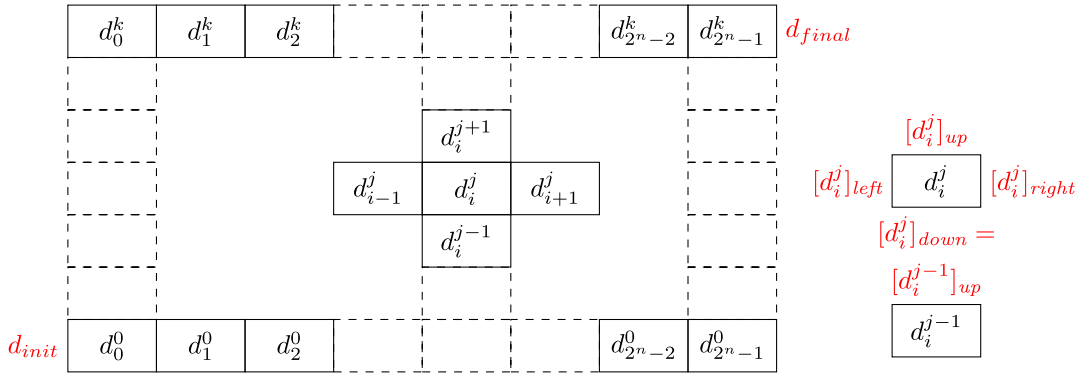

Fig. C.6. A (generic) instance of the domino-tiling problem, where d^i_j denotes $f(i, j)$.

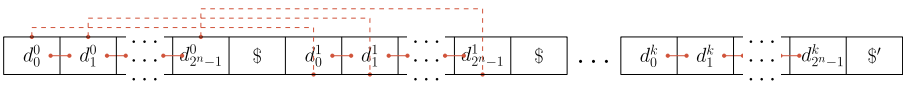

Fig. C.7. A timeline encoding the ordered concatenation of the rows of a tiling. Red lines represent the horizontal and vertical constraints among domino-types.

tial length, which is known to be **EXPSPACE**-complete [15]. Hardness holds also when only a single state variable is involved. By putting together Theorem 14, **EXPSPACE**-completeness of the problem follows.

We start by introducing the domino-tiling problem for grids with rows of single exponential length. An instance $\mathcal{I}$ of such a problem is a tuple $\mathcal{I} = (C, \Delta, n, d_{init}, d_{final})$, 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 $d_{init}, d_{final} \in \Delta$ are two distinguished domino-types (respectively, the initial and final domino-types). The *size* of $\mathcal{I}$ is defined as $|C| + |\Delta| + n$. Intuitively, a tiling of a grid is a color labeling of the edges of each cell (see Fig. C.6). Formally, a *tiling of* $\mathcal{I}$ is a mapping $f : [0, k] \times [0, 2^n - 1] \to \Delta$, for some $k \geq 0$, that satisfies the following constraints:

- two adjacent cells in a row have the same color on the shared edge, namely, for all $(i, j) \in [0, k] \times [0, 2^n - 2]$, $[f(i, j)]_{right} = [f(i, j + 1)]_{left}$ (*horizontal requirement*);
- two adjacent cells in a column have the same color on the shared edge, namely, for all $(i, j) \in [0, k - 1] \times [0, 2^n - 1]$, $[f(i, j)]_{up} = [f(i + 1, j)]_{down}$ (*vertical requirement*);
- $f(0, 0) = d_{init}$ (*initialization requirement*) and $f(k, 2^n - 1) = d_{final}$ (*acceptance requirement*).

Checking the existence (respectively, non-existence) of a tiling of $\mathcal{I}$ is an **EXPSPACE**-complete problem [15]. We can now prove the following.

Theorem 20. *The future TP problem, even with* one state variable*, with simple trigger rules and non-singular intervals is* ***EXPSPACE****-hard (under polynomial-time reductions).*

Proof. For the reduction, we define the state variable $y = (V, T, D)$ where:

- $V = \{\$, \$'\} \cup \Delta$ (with $\$, \$' \notin \Delta$),
- $T(\$) = \Delta$ and $T(\$') = \{\$'\}$,
- for $d \in \Delta \setminus \{d_{final}\}$, $T(d) = \{\$\} \cup \{d' \in \Delta \mid [d]_{right} = [d']_{left}\}$,
- $T(d_{final}) = \{\$, \$'\} \cup \{d' \in \Delta \mid [d_{final}]_{right} = [d']_{left}\}$,
- for all $v \in V$, $D(v) = [2, +\infty[$.

Basically, the domain of the state variable y contains all domino-types, as well as two auxiliary symbols \$ and \$′. The idea is encoding a tiling by the concatenation of its rows, separated by an occurrence of \$. The last row is terminated by \$′.

More precisely, each cell of the grid is encoded by (the value of) a token having duration 2. A row of the grid is then represented by the sequence of tokens of its cells, ordered by increasing column index. Finally, a full tiling is just given by the timeline for y obtained by concatenating the sequences of tokens of all rows, ordered by increasing row index. See Fig. C.7 for an example.

We observe that T guarantees the horizontal constraint among domino-types, and that it allows only occurrences of \$′ after the first \$′.

We start with the following simple trigger rules, one for each $v \in V$: $o[y = v] \rightarrow o \leq^{s,e}_{[0,2]} o$. These, paired with the constraint function D, enforce the duration of all tokens to be exactly 2. This is done for technical convenience: intuitively, since we exclude singular intervals, requiring, for instance, that a token o' starts t instants of time after the end of o, with $t \in [\ell, \ell+1]$ and even $\ell \in \mathbb{N}$, boils down to o' starting *exactly* ℓ instants after the end of o. We also observe that, given the constant token duration, in this proof the density of the time domain does not play any role.

We now define the following synchronization rules (of which all trigger ones are simple and future). The next ones state (together) that the *first occurrence* of (a token having value) \$ starts exactly at $2 \cdot 2^n$: $\top \rightarrow \exists o[y = \$].o \geq^{s}_{[0,1]} 2 \cdot 2^n$, and $o[y = \$] \rightarrow o \geq^{s}_{[0,+\infty[} 2 \cdot 2^n$. Thus, all tokens before such a first occurrence of \$ have a value in Δ. Every occurrence of \$ must be followed, after exactly $2 \cdot 2^n$ instants of time (namely, after 2^n tokens), by another occurrence of \$ or of \$': $o[y = \$] \rightarrow (\exists o'[y = \$].o \leq^{e,s}_{[2 \cdot 2^n, 2 \cdot 2^n+1]} o') \vee (\exists o''[y = \$'].o \leq^{e,s}_{[2 \cdot 2^n, 2 \cdot 2^n+1]} o'')$.

Now we force every token with value $d \in \Delta$ either (i) to be followed, after $2 \cdot 2^n$ instants, by another token with value $d' \in \Delta$, in particular, satisfying the vertical requirement, i.e., $[d]_{up} = [d']_{down}$, or (ii) to be in the last row (which is terminated by \$'). For each $d \in \Delta$,

$$o[y = d] \rightarrow \Big(\bigvee_{d' \in \Delta,\ [d]_{up} = [d']_{down}} \exists o'[y = d'].o \leq^{e,s}_{[2 \cdot 2^n, 2 \cdot 2^n+1]} o' \Big) \vee (\exists o''[y = \$'].o \leq^{e,s}_{[0, 2 \cdot 2^n - 2]} o'').$$

It is straightforward to check that the above defined rules along with the horizontal constraint guaranteed by the function T, enforce the following property.

Proposition 21. *There exists $k' \in \mathbb{N}^+$ such that all tokens with value \$ end at all and only times $k \cdot 2(2^n+1)$, for $1 \leq k < k'$. Moreover the first token with value \$' ends at time $k' \cdot 2(2^n+1)$. All other tokens having end time less than $k' \cdot 2(2^n+1)$ have value in Δ and satisfy the horizontal and vertical constraints.*

The initialization and acceptance requirements are set by means of the following pair of trigger-less rules: $\top \rightarrow \exists o[y = d_{init}].o \geq^{s}_{[0,1]} 0$ and $\top \rightarrow \exists o[y = d_{final}] \exists o'[y = \$'].o \leq^{e,s}_{[0,1]} o'$. The former rule states that a token with value d_{init} must start at $t = 0$, the latter that a token with value d_{final} must occur just before the terminator of the last row \$'. To conclude the proof, we observe that the state variable $y = (V, T, D)$ and all synchronization rules can be generated in polynomial time in the size of the instance $\mathcal{I}$ of the domino-tiling problem (in particular, all interval bounds and time constants of time-point atoms have a value, encoded in binary, which is in $O(2^n)$). □

We now focus on the case with intervals in $Intv_{(0,\infty)}$, proving that the problem is **PSPACE**-hard (and thus **PSPACE**-complete by Theorem 14) by reducing periodic SAT [Papadimitriou, 1994] to it in polynomial time.

We recall the *periodic SAT* problem. We are given a Boolean formula φ in *conjunctive normal form*, defined over two sets of variables, $\Gamma = \{x_1, \ldots, x_n\}$ and $\Gamma^{+1} = \{x_1^{+1}, \ldots, x_n^{+1}\}$, namely, $\varphi = \bigwedge_{t=1}^{m} \Big(\bigvee_{x \in (\Gamma \cup \Gamma^{+1}) \cap L_t^+} x \vee \bigvee_{x \in (\Gamma \cup \Gamma^{+1}) \cap L_t^-} \neg x \Big)$, where m is the number of conjuncts of φ and, for $1 \leq t \leq m$, L_t^+ (resp., L_t^-) is the set of variables occurring non-negated (resp., negated) in the t-th conjunct of φ. Moreover, the formula φ^j, for $j \in \mathbb{N} \setminus \{0\}$, is defined as φ in which we replace each variable $x_i \in \Gamma$ by a fresh one x_i^j, and $x_i^{+1} \in \Gamma^{+1}$ by x_i^{j+1}.

Periodic SAT is then the problem of deciding the satisfiability of the (infinite-length) formula $\Phi = \bigwedge_{j \in \mathbb{N} \setminus \{0\}} \varphi^j$, that is, deciding the existence of a truth assignment of (infinitely many) variables x_i^j, for $i = 1, \ldots, n$, $j \in \mathbb{N} \setminus \{0\}$, satisfying Φ. Periodic SAT is **PSPACE**-complete; in particular membership to such a class is proved by showing that one can equivalently check the satisfiability of the (finite-length) formula $\Phi_f = \bigwedge_{j=1}^{2^{2n}+1} \varphi^j$. Intuitively, 2^{2n} is the number of possible truth assignments to variables of $\Gamma \cup \Gamma^{+1}$, thus, after $2^{2n}+1$ copies of φ, we can find a repeated assignment: from that point, we can just loop through the previous assignments. We now reduce periodic SAT to our problem. Hardness also holds when only a single state variable is involved, and also restricting to intervals of the form $[0, a]$.

Theorem 22. *The future TP problem, even with* one state variable*, with simple trigger rules and intervals $[0, a]$, $a \in \mathbb{N} \setminus \{0\}$, is **PSPACE**-hard (under polynomial-time reductions).*

Proof. Let us define the state variable $y = (V, T, D)$, where

- $V = \{\$, \tilde{\$}, stop\} \cup \{x_i^{\top}, x_i^{\bot}, \tilde{x_i}^{\top}, \tilde{x_i}^{\bot} \mid i = 1, \ldots, n\}$,
- $T(\$) = \{x_1^{\top}, x_1^{\bot}\}$, $T(\tilde{\$}) = \{\tilde{x_1}^{\top}, \tilde{x_1}^{\bot}\}$ and $T(stop) = \{stop\}$,
- for $i = 1, \ldots, n-1$, $T(x_i^{\top}) = T(x_i^{\bot}) = \{x_{i+1}^{\top}, x_{i+1}^{\bot}\}$,
- for $i = 1, \ldots, n-1$, $T(\tilde{x_i}^{\top}) = T(\tilde{x_i}^{\bot}) = \{\tilde{x_{i+1}}^{\top}, \tilde{x_{i+1}}^{\bot}\}$,
- $T(x_n^{\top}) = T(x_n^{\bot}) = \{\tilde{\$}, stop\}$,
- $T(\tilde{x_n}^{\top}) = T(\tilde{x_n}^{\bot}) = \{\$, stop\}$, and
- for all $v \in V$, $D(v) = [2, +\infty[$.

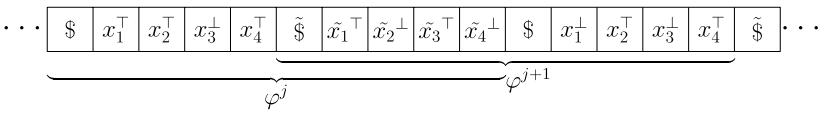

Fig. C.8. Let the formula φ be defined over two sets of variables, $\Gamma = \{x_1, x_2, x_3, x_4\}$ and $\Gamma^{+1} = \{x_1^{+1}, x_2^{+1}, x_3^{+1}, x_4^{+1}\}$. The j-th copy (we assume j is odd) of φ, i.e., φ^j, is satisfied by the assignment $x_1^j \mapsto \top$, $x_2^j \mapsto \top$, $x_3^j \mapsto \perp$, $x_4^j \mapsto \top$, $x_1^{j+1} \mapsto \top$, $x_2^{j+1} \mapsto \perp$, $x_3^{j+1} \mapsto \top$, $x_4^{j+1} \mapsto \perp$. The analogous for φ^{j+1}.

Intuitively, we represent an assignment of variables x_i^j by means of a timeline for y: after every occurrence of the symbol \$, n tokens are present, one for each x_i, and the value $x_i^\top$ (resp., $x_i^\perp$) represents a positive (resp., negative) assignment of x_i^j, for some *odd* $j \geq 1$. Then, there is an occurrence of $\tilde{\$}$, after which n more tokens occur, again one for each x_i, and the value $\tilde{x_i}^\top$ (resp., $\tilde{x_i}^\perp$) represents a positive (resp., negative) assignment of x_i^j, for some *even* $j \geq 2$. See Fig. C.8 for an example. We start with the next simple trigger rules, one for each $v \in V$: $o[y = v] \rightarrow o \leq^{\mathsf{s},\mathsf{e}}_{[0,2]} o$. Paired with the constraint function D, they enforce all tokens' durations to be *exactly* 2 (as in the previous proof). Then We add the following rules:

- $\top \rightarrow \exists o[y = \$].o \geq^{\mathsf{s}}_{[0,1]} 0$;
- $\top \rightarrow \exists o[y = \tilde{\$}].o \geq^{\mathsf{s}}_{[0,1]} (2^{2n} + 1) \cdot 2(n + 1)$;
- $\top \rightarrow \exists o[y = stop].o \geq^{\mathsf{s}}_{[0,1]} (2^{2n} + 2) \cdot 2(n + 1)$.

They respectively impose that (i) a token with value \$ starts exactly at $t = 0$ (recall that the duration of every token is 2); (ii) there exists a token with value $\tilde{\$}$ starting at $t = (2^{2n} + 1) \cdot 2(n + 1)$; (iii) a token with value *stop* starts at $t = (2^{2n}+2)\cdot 2(n+1)$. We are forcing the timeline to encode truth assignments for variables $x_1^1, \ldots, x_n^1, \ldots, x_1^{2^{2n}+2}, \ldots, x_n^{2^{2n}+2}$: as a matter of fact, we will decide satisfiability of the finite formula $\Phi_f = \bigwedge_{j=1}^{2^{2n}+1} \varphi^j$, which is equivalent to Φ.

We now consider the following rules, which enforce the satisfaction of each φ^j or, equivalently, of φ over the assignments of $(x_1^j, \ldots, x_n^j, x_1^{j+1}, \ldots, x_n^{j+1})$.

For the t-th conjunct of φ, we define the future simple rule:

$$
\begin{aligned}
o[y = \tilde{\$}] \rightarrow\ & \Big(\bigvee_{x_i \in \Gamma \cap L_t^+} \exists o'[y = \tilde{x_i}^\top].o \leq^{\mathsf{e},\mathsf{s}}_{[0,4n]} o' \Big) \vee \Big(\bigvee_{x_i^{+1} \in \Gamma^{+1} \cap L_t^+} \exists o'[y = x_i^\top].o \leq^{\mathsf{e},\mathsf{s}}_{[0,4n]} o' \Big) \vee \\
& \Big(\bigvee_{x_i \in \Gamma \cap L_t^-} \exists o'[y = \tilde{x_i}^\perp].o \leq^{\mathsf{e},\mathsf{s}}_{[0,4n]} o' \Big) \vee \Big(\bigvee_{x_i^{+1} \in \Gamma^{+1} \cap L_t^-} \exists o'[y = x_i^\perp].o \leq^{\mathsf{e},\mathsf{s}}_{[0,4n]} o' \Big) \vee \\
& \exists o''[y = stop].o \leq^{\mathsf{e},\mathsf{s}}_{[0,2n]} o''.
\end{aligned}
$$

Basically, this rule (the rule where the trigger has value \$ being analogous) states that, after every occurrence of $\tilde{\$}$, a token o', making true at least a (positive or negative) literal in the conjunct, must occur by $4n$ time instants (i.e., before the following occurrence of $\tilde{\$}$). The disjunct $\exists o''[y = stop].o \leq^{\mathsf{e},\mathsf{s}}_{[0,2n]} o''$ is present just to avoid evaluating φ on the n tokens before (the first occurrence of) *stop*.

The variable y and all synchronization rules can be generated in time polynomial in $|\varphi|$ (in particular, all interval bounds and time constants of time-point atoms have a value, encoded in binary, in $O(2^{2n})$). □

By Theorem 14 and Theorem 22, **PSPACE**-completeness of future TP with simple trigger rules and intervals in $Intv_{(0,\infty)}$ follows.

References

[1] R. Alur, D.L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (2) (1994) 183–235.
[2] R. Alur, T. Feder, T.A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1) (1996) 116–146.
[3] R. Alur, T.A. Henzinger, Real-time logics: complexity and expressiveness, Inf. Comput. 104 (1) (1993) 35–77.
[4] 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 ICKEPS, 2012.
[5] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, Complexity of timeline-based planning over dense temporal domains: exploring the middle ground, in: Proc. of GandALF, EPTCS, 2018, pp. 191–205.
[6] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, Decidability and complexity of timeline-based planning over dense temporal domains, in: Proc. of KR 2018, AAAI Press, 2018, pp. 627–628.
[7] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, G. Woeginger, Timeline-based planning over dense temporal domains with trigger-less rules is NP-complete, in: Proc. of ICTCS. CEUR WP, 2018, pp. 116–127.
[8] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, N. Policella, An innovative product for space mission planning: an a posteriori evaluation, in: Proc. of ICAPS, AAAI, 2007, pp. 57–64.

[9] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, S. Frye, Timeline-based space operations scheduling with external constraints, in: Proc. of ICAPS, AAAI, 2010, pp. 34–41.
[10] M. Cialdea Mayer, A. Orlandini, A. Umbrico, Planning and execution with flexible timelines: a formal account, Acta Inform. 53 (6–8) (2016) 649–680.
[11] S. Demri, R. Lazic, LTL with the freeze quantifier and register automata, ACM Trans. Comput. Log. 10 (3) (2009) 16:1–16:30.
[12] J. Frank, A. Jónsson, Constraint-based attribute and interval planning, Constraints 8 (4) (2003) 339–364.
[13] N. Gigante, A. Montanari, M. Cialdea Mayer, A. Orlandini, Timelines are expressive enough to capture action-based temporal planning, in: Proc. of TIME, IEEE Computer Society, 2016, pp. 100–109.
[14] N. Gigante, A. Montanari, M. Cialdea Mayer, A. Orlandini, Complexity of timeline-based planning, in: Proc. of ICAPS, AAAI, 2017, pp. 116–124.
[15] D. Harel, Algorithmics: The Spirit of Computing, Springer, 2012.
[16] A.K. Jónsson, P.H. Morris, N. Muscettola, K. Rajan, B.D. Smith, Planning in interplanetary space: theory and practice, in: Proc. of ICAPS, AAAI, 2000, pp. 177–186.
[17] D. Jungnickel, Graphs, Networks and Algorithms, Algorithms and Computation in Mathematics, Springer, 2013.
[18] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Syst. 2 (4) (1990) 255–299.
[19] M.L. Minsky, Computation: Finite and Infinite Machines. Automatic Computation, Prentice-Hall, Inc., 1967.
[20] N. Muscettola, HSTS: integrating planning and scheduling, in: Intelligent Scheduling, Morgan Kaufmann, 1994, pp. 169–212.
[21] J. Ouaknine, J. Worrell, On the decidability and complexity of metric temporal logic over finite words, Log. Methods Comput. Sci. 3 (1) (2007).