Timeline-based planning over dense temporal domains

Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, Gerhard Woeginger

Preprint nr.: 3/2018

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*

Abstract

Planning is one of the most studied problems in computer science. In this paper, we consider the timeline-based approach, where the planning 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 (synchronization rules). Whereas the temporal domain is usually assumed to be discrete, here we address decidability and complexity issues for timeline-based planning (TP) over dense temporal domains.

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

Finally, we show how systems can be described by timelines, and then model checked against property specifications given by formulas of Metric Interval Temporal Logic (MITL), a well-known timed extension of LTL.

Keywords: Planning, Timelines, Metric Temporal Logic, Timed Automata, Computational Complexity
2010 MSC: 03B70, 68Q60

Timeline-based planning (TP for short) represents an alternative to the 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 we reach a state satisfying the goal. Conversely, TP can be considered as a more declarative approach, in that it focuses on what has to happen in order to satisfy the goal instead of what an agent has to do. In TP, the planning domain is modeled as a set of independent (but interacting)

[⋆]This paper is an extended and revised version of [6, 5, 7].

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* (in turn these are sequences of time intervals called *tokens*), and it is governed by a set of transition functions, one for each state variable, and a number of synchronization rules, that constrain the temporal relations among (values of) state variables.

TP has been successfully exploited in a number of application domains, for instance, in space missions, constraint solving, activity scheduling,... (see, e.g., [4, 9, 10, 13, 17, 22]), 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 [14], Gigante et al. showed 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, in [15], 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.

In the following sections we will study TP over a *dense temporal domain*, without having recourse to any form of discretization, which is quite a common trick. The reason why we assume this different version of time domain is, basically, to increase expressiveness: in this way one can abstract from unnecessary (or even "forced") details, often artificially added due to the necessity of discretizing time, and can suitably represent actions with duration, accomplishments and temporally extended goals.

We will study suitable restrictions on the TP problem that allow us to recover its decidability: as a matter of fact, the first result we establish is a negative one, namely, that TP over dense time, in its general formulation, is *undecidable*. Then we will also show how to obtain better computational complexities, which are appropriate to the practical exploitation of TP, by constraining the logical 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 us to compare tokens of the timelines both preceeding (past) and following (future) the trigger token. The first restriction we consider consists in limiting the comparison to tokens in the future with respect to the trigger (*future semantics* of trigger rules). The second imposes that the name of a non-trigger token appears exactly once in the constraints set by the rule (*simple* trigger rules): this syntactical restriction avoids comparisons of multiple token time-events with a non-trigger reference time-event. Better complexity results can be obtained by restricting also the type of *intervals* used in rules in order to compare tokens.

Table 1 summarizes all the decidability and complexity results described in the following: we will consider 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[1], or unbounded/left-closed with left endpoint 0

[1] An interval of the form $[a, a]$, for $a \in \mathbb{N}$, is called *singular*.

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

	TP problem	Future TP problem
Unrestricted	Undecidable	(Decidable?) 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 $\mathit{Intv}_{(0,\infty)}$	?	**PSPACE**-complete
Trigger-less rules only	**NP**-complete	//

(the latter intervals are denoted by $\mathit{Intv}_{(0,\infty)}$).

We now describe the organization of this paper.

Organization of the paper.

- In Section 1 we start by introducing the TP framework, providing some background knowledge on it.
- In Section 2 we prove that TP is *undecidable* in the general case, by a reduction from the *halting problem for Minsky* 2*-counter machines.* The section is concluded commenting on *non-primitive recursive-hardness* of TP under the future semantics of trigger rules (this is formally proved in Appendix B).
- In Section 3, we establish that future TP with simple trigger rules is *decidable* (in non-primitive recursive time), and then we show membership in **EXPSPACE** (respectively, **PSPACE**) under the restriction to *non-singular intervals* (respectively, intervals in $\mathit{Intv}_{(0,\infty)}$).
- Matching complexity lower bounds for the last two restrictions are given in Section 4.
- In Section 5 we outline an **NP** planning algorithm for TP with *trigger-less rules only* (which disallow the universal quantification/trigger and have a purely existential form) 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.
- In Section 6, we tackle a different problem, namely, model checking (MC) for systems described by timelines, where property specifications are given in terms of formulas of *Metric Interval Temporal Logic* (MITL), a timed logic which extends LTL. In this respect TP can be regarded as a sort of necessary condition for MC, the former playing the role of a "feasibility check" of the system description, which is not immediately feasible by definition—as opposed, for example, to Kripke structures—given the presence of synchronization rules that constrain the legal system computations.

1. The TP Problem

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{R}_+$ be the set of non-negative real numbers; moreover, *Intv* denotes the set of intervals of $\mathbb{R}_+$ whose endpoints are in $\mathbb{N} \cup \{\infty\}$, and $\mathit{Intv}_{(0,\infty)}$ is the set of non-singular intervals $I \in \mathit{Intv}$ such that either I is unbounded, or I is left-closed with left endpoint 0. The latter intervals I can be represented by expressions of the form $\sim n$, for some $n \in \mathbb{N}$ and $\sim \in \{<, \leq, >, \geq\}$.

We now introduce notation and basic notions of the TP framework as presented in [11, 14]. In TP, domain knowledge is encoded by a set of state variables, whose behaviour over time is described by transition functions and synchronization rules.

Definition 1 (State variable)**.** 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
- $D_x : V_x \to \mathit{Intv}$ is the *constraint (or duration) function* that maps each $v \in V_x$ to an interval of *Intv*.

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 token for x represents an interval of time where the state variable x takes value v. In order to clarify the variable to which a token refers, we shall often denote (v, d) as (x, v, d).

The behavior of the state variable x is specified by means of 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 defined respectively as follows:

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

and

$$\mathsf{e}(\pi, i) = \sum_{h=0}^{i} d_h.$$

See Figure 1 for an example.

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

Multi-timelines of SV can be constrained by a set of *synchronization rules*, which relate tokens, possibly belonging to different timelines, through temporal

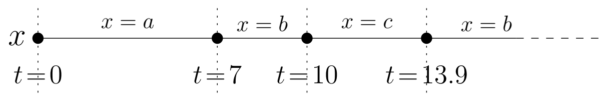

Figure 1: An example of timeline $(a, 7)(b, 3)(c, 3.9)\cdots$ for the state variable $x = (V_x, T_x, D_x)$, where $V_x = \{a, b, c, \ldots\}$, $b \in T_x(a)$, $c \in T_x(b)$, $b \in T_x(c)\ldots$ and $D_x(a) = [5, 8]$, $D_x(b) = [1, 4]$, $D_x(c) = [2, \infty[\ldots$

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 make use of 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*.

Definition 2 (Atom)**.** An *atom* ρ is either a clause of the form $o_1 \leq_I^{e_1,e_2} o_2$ (*interval atom*), or of the forms $o_1 \leq_I^{e_1} n$ or $n \leq_I^{e_1} o_1$ (*time-point atom*), where $o_1, o_2 \in \Sigma$, $I \in \mathit{Intv}$, $n \in \mathbb{N}$, and $e_1, e_2 \in \{\mathsf{s}, \mathsf{e}\}$.

An atom ρ is evaluated with respect to a Σ-*assignment* λ_Π for a given multi-timeline Π, which is a mapping assigning to each token name $o \in \Sigma$ a pair $\lambda_\Pi(o) = (\pi, i)$ such that π is a timeline of Π and $0 \leq i < |\pi|$ is a position along π (intuitively, (π, i) represents the token of Π referenced by the name o).

An 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 $o \leq_I^{e} n$ (respectively, $n \leq_I^{e} o$) *is satisfied by* λ_Π if $n - e(\lambda_\Pi(o)) \in I$ (respectively, $e(\lambda_\Pi(o)) - n \in I$).

Definition 3 (Existential statement)**.** An *existential statement* $\mathcal{E}$ for a finite set SV of state variables is a statement of the form

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

where $\mathcal{C}$ is a conjunction of atoms, $o_i \in \Sigma$, $x_i \in SV$ and $v_i \in V_{x_i}$, for $1 \leq i \leq n$.

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 4 (Synchronization rule)**.** A *synchronization rule* $\mathcal{R}$ for a finite set SV of state variables is a rule of one of the forms

$$o_0[x_0 = v_0] \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k, \qquad \top \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k,$$

where $o_0 \in \Sigma$, $x_0 \in SV$, $v_0 \in V_{x_0}$, and $\mathcal{E}_1, \ldots, \mathcal{E}_k$ are *existential statements.* In rules of the first form (which are called *trigger rules*), the quantifier $o_0[x_0 = v_0]$

is called *trigger*; we require that only o_0 may appear free in $\mathcal{E}_i$, for all $1 \leq i \leq n$. In rules of the second form (*trigger-less rules*), we require that no token name appears free.
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. As an example,

$$o_0[x_0 = v_0] \to \exists o_1[x_1 = v_1].o_0 \leq^{\mathsf{e},\mathsf{s}}_{[2,\infty[} o_1$$

states that after *every* token for x_0 with value v_0 there exists a token for x_1 with value v_1 *starting* at least 2 time instants after the *end* of the former. Trigger-less rules simply assert the satisfaction of some existential statement. The intuitive meaning of *simple* trigger rules is that they disallow simultaneous comparisons of multiple time-events (start/end times of tokens) with a non-trigger reference time-event.

The semantics of synchronization rules is formally defined as follows.

Definition 5 (Semantics of synchronization rules)**.** Let Π be a multi-timeline of a set SV of state variables.

Given a *trigger-less rule* $\mathcal{R}$ of SV, Π *satisfies* $\mathcal{R}$ if Π satisfies some existential statement of $\mathcal{R}$.

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 paper, we shall also focus on a stronger notion of satisfaction of trigger rules, called *satisfaction under the future semantics*: it requires that all non-trigger tokens selected by some quantifier do not start *strictly before* the start time of the trigger token.

Definition 6 (Future semantics of trigger rules)**.** A multi-timeline Π of SV satisfies a trigger rule

$$\mathcal{R} = o_0[x_0 = v_0] \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$$

under the future semantics if Π satisfies the trigger rule obtained from $\mathcal{R}$ by replacing each existential statement

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

by

$$\mathcal{E}'_i = \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n].\Big(\mathcal{C} \wedge \bigwedge_{i=1}^{n} o_0 \leq^{\mathsf{s},\mathsf{s}}_{[0,+\infty[} o_i\Big).$$

Finally, 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. Trigger-less rules can be used to express initial, as well as intermediate conditions and the goals of the problem, while trigger rules are much more powerful and useful, for instance, to specify invariants and response requirements.

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 we require satisfaction under the future semantics of *all* trigger rules.

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 a TP domain $P = (SV, R)$, is there a *future* plan for P?

We refer again to Table 1 for the decidability and complexity results proved in the following about the mentioned problems.

2. TP over dense temporal domains is an undecidable problem

In this section, 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* [21]. The proof somehow resembles the one for the satisfiability problem of Metric Temporal Logic (which will be formally introduced later, in Section 3) 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 (or just *counter machine* for short) is a tuple $M = (\mathsf{Inst}, \ell_{init}, \ell_{halt})$ consisting of a finite set Inst of labeled instructions $\ell : \imath$, where ℓ is a label and $\imath$ is an instruction for either

- *increment* of counter h: $c_h := c_h + 1$; $\texttt{goto}\ \ell_r$, or
- *decrement* of counter h: $\texttt{if}\ c_h > 0\ \texttt{then}\ c_h := c_h - 1$; $\texttt{goto}\ \ell_s\ \texttt{else goto}\ \ell_t$,

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, which is the one to be executed next), 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:

- for an instruction with label ℓ incrementing c_1, we have $(\ell, n_1, n_2) \xrightarrow{M} (\ell_r, n_1 + 1, n_2)$, and

- for an instruction decrementing 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 analogous for instructions changing the value of 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 $n_1, n_2 \in \mathbb{N}$. Given a counter machine M, the *halting problem for M* is to decide whether M halts, and it was proved to be *undecidable* by Minsky [21].

The rest of the section is devoted to showing the following result.

Theorem 7. *The TP problem over dense temporal domains is undecidable (even when a single state variable is involved).*

Proof. We prove the thesis by a reduction from the halting problem for Minsky 2-counter machines. Let us introduce the following notational conventions:

- for increment instructions $\ell : c_h := c_h + 1;$ `goto` ℓ_r, we define $c(\ell) = c_h$ and $succ(\ell) = \ell_r$;
- for decrement instructions $\ell :$ `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 $\mathsf{InstLab}$ 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 without loss of generality that no instruction of M leads to ℓ_{init}, and that ℓ_{init} is the label of an increment instruction. To prove the thesis, 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 timeline for x_M which satisfies all the rules in R_M, that is, a plan for $P = (\{x_M\}, R_M)$.

Encoding of M-computations.. First, we define a suitable encoding of a computation of M as a timeline for x_M. For such an encoding we exploit the finite set of symbols $V = V_{main} \cup V_{check}$ corresponding to the finite domain of the state variable x_M. The sets of *main* values V_{main} and *check* values V_{check} are defined as

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

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

and

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

For each $h \in \{1,2\}$, we denote by V_{c_h} the set of V-values v having the form $v = (\ell, c)$, $v = (\ell, \ell', c)$, or $v = (\ell, op, c)$, where $c \in \{c_h, (c_h, \#)\}$: if $c = c_h$, we say that v is an *unmarked* V_{c_h}-value; otherwise ($c = (c_h, \#)$), v is a *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 a M-configuration (ℓ, n_1, n_2), where the instruction label $\ell \in \mathsf{Inc} \cup \{\ell_{halt}\}$, $n_1 \geq 0$, and $n_2 \geq 0$, has the form:

$$w_{main} = \ell \cdot \underbrace{(\ell, c_1) \cdots (\ell, c_1)}_{n_1 \text{ times}} \cdot \underbrace{(\ell, c_2) \cdots (\ell, c_2)}_{n_2 \text{ times}}.$$

In the 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 zero.

$$(\ell, \mathsf{zero}(\ell)) \cdot \underbrace{(\ell, \mathsf{zero}(\ell), c_2) \cdots (\ell, \mathsf{zero}(\ell), c_2)}_{n_2 \text{ times}},$$

$$(\ell, \mathsf{dec}(\ell)) \cdot (\ell, \mathsf{dec}(\ell), (c_1, \#)) \cdot \underbrace{(\ell, \mathsf{dec}(\ell), c_1) \cdots (\ell, \mathsf{dec}(\ell), c_1)}_{n_1 \text{ times}} \cdot \underbrace{(\ell, \mathsf{dec}(\ell), c_2) \cdots (\ell, \mathsf{dec}(\ell), c_2)}_{n_2 \text{ times}}.$$

In the first case, w'_{main} encodes the configuration $(\ell, 0, n_2)$ and in the second case 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 which will be removed by the decrement. Analogously, the *main*-code for a *decrement* instruction label ℓ with $c(\ell) = c_2$ has two forms symmetric with respect to the previous cases.

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 incrementing the counter c_i (resp., decrementing c_i with c_i greater than 0 in C_p, decrementing c_i with c_i equal to 0 in C_p).

The *check*-code for an instruction label $\ell \in \mathsf{InstLab}$ and an inc_1-operation has the following form

$$(\ell, \mathsf{inc}_1) \cdot (\ell, \mathsf{inc}_1, (c_1, \#)) \cdot \underbrace{(\ell, \mathsf{inc}_1, c_1) \cdots (\ell, \mathsf{inc}_1, c_1)}_{n_1 \text{ times}} \cdot \underbrace{(\ell, \mathsf{inc}_1, c_2) \cdots (\ell, \mathsf{inc}_1, c_2)}_{n_2 \text{ times}},$$

and encodes the configuration $(\ell, n_1 + 1, n_2)$. Note that there is exactly one occurrence of a *marked* V_{c_1}-value which intuitively represents the unit added to the counter by the increment operation.

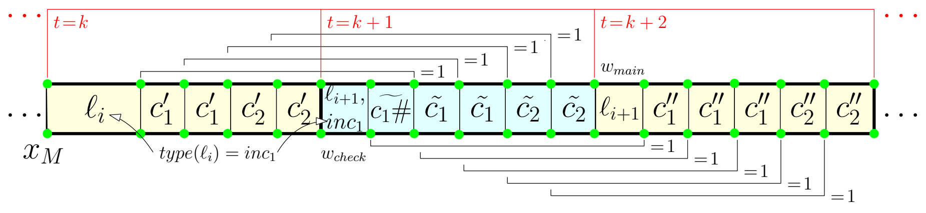
Figure 2: A fragment of a computation code with a configuration code for an instruction ℓ_{i+1}. Main-codes are highlighted in yellow and check-codes in cyan. Each square can also be seen as a token of a timeline for x_M (tokens are decorated with their start time and their temporal constraints). In the figure, for $h \in \{1,2\}$, the symbols c'_h, $\tilde{c_h}$, $\widetilde{c_h\#}$, and c''_h, 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).

The *check*-code for an instruction label $\ell \in \mathsf{InstLab}$ and an operation $op_1 \in \{\mathsf{dec}_1, \mathsf{zero}_1\}$ for the counter c_1 has the form

$$(\ell, op_1) \cdot \underbrace{(\ell, op_1, c_1) \cdots (\ell, op_1, c_1)}_{n_1 \text{ times}} \cdot \underbrace{(\ell, op_1, c_2) \cdots (\ell, op_1, c_2)}_{n_2 \text{ times}},$$

where we require that $n_1 = 0$ if $op_1 = \mathsf{zero}_1$. The *check*-code for a label $\ell \in \mathsf{InstLab}$ and an operation associated with the counter c_2 is defined in a similar way.

A *configuration*-code is a word $w = w_{check} \cdot w_{main}$ such that w_{check} is a *check*-code, w_{main} is a *main*-code, and w_{check} and w_{main} are associated with the same instruction label. The configuration code is *well-formed* if w_{check} and w_{main} encode the same configuration.

Figure 2 depicts the encoding of a configuration-code for the instruction ℓ_{i+1}. The check-code for the instruction ℓ_{i+1} is associated with an increment of the counter c_1 (the type of instruction ℓ_i).

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 (we assume ℓ_i to be the instruction label associated with the configuration code $w^i_{check} \cdot w^i_{main}$):

- $\ell_j \neq \ell_{halt}$;
- if $\ell_j \in \mathsf{Inc}$ with $c(\ell_j) = c_h$, then $\ell_{j+1} = succ(\ell_j)$ and w^{j+1}_{check} is associated with the operation inc_h;
- 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 the 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 $w^i_{check} \cdot w^i_{main}$):

- if $\ell_j \in \mathsf{Inc}$, with $c(\ell_j) = c_h$, then $n^{j+1}_h = n^j_h + 1$ and $n^{j+1}_{3-h} = n^j_{3-h}$;

- if $\ell_j \in \mathsf{Dec}$, with $c(\ell_j) = c_h$, then $n_{3-h}^{j+1} = n_{3-h}^{j}$. Moreover, if w_{check}^{j+1} is associated with dec_h, then $n_h^{j+1} = n_h^j - 1$.

Clearly, a well-formed computation code π encodes a computation of the Minsky 2-counter machine.

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}. The counter machine M halts if and only if there is an initial and halting well-formed computation-code.

Definition of x_M and R_M.. Let us show now how to reduce the problem of checking the existence of an initial and halting well-formed computation-code to a TP problem for the state variable x_M.

The idea is to define a timeline where the sequence of values of its tokens is a well-formed computation-code. The durations of tokens are suitably exploited to guarantee well-formedness of computation-codes. We refer the reader again to Figure 2 for an intuition. Each symbol of the computation-code is associated with a token having a 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. To allow for the encoding of arbitrarily large values of counters in one time unit, the duration of such tokens is not fixed (taking advantage of the dense temporal domain). 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 Figure 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. Note that there is a single *marked* token c_1 in the check-code—that represents the unit added to c_1 by the instruction ℓ_i—which does not correspond to any of the c_1's of the preceding main-code.

We now formally define a state variable x_M and a set R_M of synchronization rules for x_M such that the untimed part of any timeline (i.e., neglecting tokens' durations) for x_M satisfying the rules in R_M is (represents) an initial and halting well-formed computation-code. Thus, M halts if and only if there exists a timeline for x_M satisfying the rules in R_M.

As for x_M, we let $x_M = (V, T, D)$ where, for each $v \in V$, $D(v) =]0, 1]$. This sets the *strict time monotonicity* constraint, namely, the duration of a token along a timeline is always greater than zero and less than or equal to 1.

The value transition function T of x_M ensures the following requirement.

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 $v \in V$.

It is a straightforward task to define T in such a way that the previous requirement is fulfilled (for details, see Appendix A).

Finally, the synchronization rules in R_M ensure the following requirements.

- *Initialization:* every timeline starts with two tokens, the first one having value $(\ell_{init}, \mathsf{zero}_1)$, and the second having value ℓ_{init}. By Claim 8 and the fact that no instruction of M leads to ℓ_{init}, it suffices to require that a timeline has a token with value $(\ell_{init}, \mathsf{zero}_1)$ and a token with value ℓ_{init}. This is ensured by the following 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, it suffices to require that a timeline has a token with value ℓ_{halt}. This is ensured by the following trigger-less rule:

$$\top \rightarrow \exists\, o[x_M = \ell_{halt}].\ \top\,.$$

- *1-Time 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 in $V \setminus (V_{c_1} \cup V_{c_2})$. We require that the difference of the start times of two consecutive tokens along a timeline having a control V-value is exactly 1. Formally, for each pair tk and tk' of tokens 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' having a control V-value, it holds that $\mathsf{s}(tk') - \mathsf{s}(tk) = 1$ (we write this with a little abuse of notation). By Claim 8, strict time monotonicity, and the halting requirement, 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 aim, for each $v \in V_{\mathsf{con}} \setminus \{\ell_{halt}\}$, being V_{con} the set of control V-values, we write the following trigger rule:

$$o[x_M = v] \rightarrow \bigvee_{u \in V_{\mathsf{con}}} \exists\, o'[x_M = u].\ o \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o'.$$

- *Well-formedness of configuration-codes:* we need to guarantee that for each configuration-code $w_{check} \cdot w_{main}$ occurring along a timeline and each counter c_h, the value of c_h along the *main*-code w_{main} and the *check*-code w_{check} coincide. By Claim 8, strict time monotonicity, initialization, and 1-Time distance between consecutive control values, 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 vice versa (ii) each token tk with a V_{c_h}-value in V_{main} is eventually preceded by a token tk' with a V_{c_h}-value such that $\mathsf{s}(tk) - \mathsf{s}(tk') = 1$. As for the former requirement, for each $v \in V_{c_h} \cap V_{check}$, we write the rule:

$$o[x_M = v] \rightarrow \bigvee_{u \in V_{c_h}} \exists\, o'[x_M = u].\ o \leq^{\mathsf{s},\mathsf{s}}_{[1,1]} o'.$$

For the latter, for each $v \in V_{c_h} \cap V_{main}$, we have the rule:

$$o[x_M = v] \rightarrow \bigvee_{u \in V_{c_h}} \exists\, o'[x_M = u].\, o' \leq^{\mathsf{s,s}}_{[1,1]} o.$$

- *Increment and decrement:* we need to guarantee that the increment and decrement instructions are correctly simulated. By Claim 8 and the previously defined synchronization rules, 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 $\ell_{main} \neq \ell_{halt}$, end either $\ell_{main} \in \mathsf{Inc}$ and $\ell_{check} = \mathit{succ}(\ell_{main})$, 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 $n_h^{check} = n_h^{main} - 1$.

  First we observe that, if $\ell_{main} \in \mathsf{Inc}$, our encoding ensures that all $V_{c_{3-h}}$-values in w_{main} and in w_{check} are unmarked, all V_{c_h}-values in w_{main} are unmarked, and there is exactly one marked V_{c_h}-value in w_{check}. If instead $\ell_{main} \in \mathsf{Dec}$, our encoding ensures that all $V_{c_{3-h}}$-values in w_{main} and in w_{check} are unmarked, all V_{c_h}-values in w_{check} are unmarked, and in case $\ell_{check} = \mathsf{dec}(\ell_{main})$, then there is exactly one marked V_{c_h}-value in w_{main}. Thus, by strict time monotonicity and 1-Time 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 have the rule:

$$o[x_M = v] \rightarrow \bigvee_{u \in U_{c_i}} \exists\, o'[x_M = u].\, o \leq^{\mathsf{s,s}}_{[1,1]} o'.$$

  For each $v \in (U_{c_i} \cap V_{check}) \setminus V_{init}$, we have the rule:

$$o[x_M = v] \rightarrow \bigvee_{u \in U_{c_i}} \exists\, o'[x_M = u].\, o' \leq^{\mathsf{s,s}}_{[1,1]} o.$$

This concludes the proof of the theorem. □

It is worth observing that all the above trigger rules are *simple*, hence *undecidability of the TP problem holds also under the restriction to simple trigger rules.*

In order to ensure the well-formedness of configuration-codes and the increment/decrement requirements, a one-to-one correspondence between (suitable) pairs of tokens in main- and check-codes is enforced thanks to the above trigger rules. Whereas most of such rules are (already) satisfied under the future semantics (as the extra conjoined atoms added by Definition 6 would be "subsumed" by already-existing ones), some rules are not (the second ones of the well-formedness and increment/decrement requirements are unsatisfiable under the future semantics). As a result, intuitively, having only rules under the future semantics, we can only force the presence, for every token with value c_h (for $h = 1, 2$), of another token with value c_h starting exactly one time instant later, in the following main-/check-code. However, we cannot prevent extra "spurious" tokens to appear moving from a code to the following one. This is the reason why, with only rules under the future semantics, we lose the ability of encoding computations of (exact) Minsky machines. Only *gainy counter machines* [12]—a variant of Minsky machines whose counters may "erroneously" increase—can be captured, thus proving, as a consequence, non-primitive recursive-hardness of the future TP problem (the halting problem for gainy counter machines is known to be non-primitive recursive [12]).

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

Since this result is just an adaptation of the previous one (apart from some technicalities), we report its proof in Appendix B.

In the next section, we will show that future TP with simple trigger rules is indeed decidable in non-primitive recursive time.

3. Decidability of future TP with simple trigger rules

In this section, we show that the decidability of the TP problem can be recovered assuming that the trigger rules are *simple* and *interpreted under the future semantics.* Moreover, under the additional assumption that intervals in trigger rules are non-singular (respectively, are in $Intv_{(0,\infty)}$), the problem is in **EXPSPACE** (respectively, in **PSPACE**). The decidability status of *future TP with arbitrary trigger rules remains an open problem.*

The rest of this section is organized as follows: in Subection 3.1, we recall the framework of Timed Automata (TA) [1] and Metric Temporal logic (MTL) [19]. In Subection 3.2, we reduce the future TP problem with simple trigger rules to the *existential MC problem* for TAs against MTL over *finite timed words.* The latter problem is known to be decidable [23].

[2]We refer to intervals in rules' atoms and in the constraint functions of state variables.

3.1. Timed automata and the logic MTL

We start by recalling the notion of timed automaton (TA) [1] and the logic MTL [19].

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 represents the time at which the "event" a_i occurs) such that $\tau_i \leq \tau_{i+1}$ for all $0 \leq i < n$ (*monotonicity* requirement). The timed word w is also denoted by (σ, τ), where σ is the finite (untimed) word $a_0 \cdots a_n$ and τ is the sequence of timestamps $\tau_0, \ldots, \tau_n$. 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 $val : C \to \mathbb{R}_+$ for C is a function assigning a non-negative real value to each clock in C. Given a value $t \in \mathbb{R}_+$ and a set $Res \subseteq C$ (that we call *reset set*), $(val + t)$ and $val[Res]$ denote the valuations for C defined respectively as follows: for all $c \in C$, $(val + t)(c) = val(c) + t$, and $val[Res](c) = 0$ if $c \in Res$ and $val[Res](c) = val(c)$ otherwise.

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 (Timed automaton TA)**.** A TA over Σ is a tuple $\mathcal{A} = (\Sigma, Q, q_0, C, \Delta, F)$, where Q is a finite set of (control) states, $q_0 \in Q$ 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 $\mathcal{A}$.

Intuitively, in a TA $\mathcal{A}$, while transitions are instantaneous, time can elapse in a control state. The clocks progress at the same speed and can be reset independently of each other when a transition is executed, in such a way that each clock keeps track of the time elapsed since the last reset. Moreover, clock constraints are used as guards of transitions to restrict the behavior of the automaton.

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 at the initial configuration (q_0, val_0), where $val_0(c) = 0$ for all $c \in C$ (*initiation requirement*), and

- for $0 \leq i \leq n$ we have (*consecution requirement*): (*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*) $val_{i+1} = (val_i + \tau_i - \tau_{i-1})[Res]$ (we let $\tau_{-1} = 0$).

The intuitive behavior of the TA $\mathcal{A}$ is the following. Assume that $\mathcal{A}$ is on state $q \in Q$ after reading the symbol (a', τ_i) at time τ_i and, at that time, the clock valuation is val. On 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 time elapsed 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* $\mathcal{L}_T(\mathcal{A})$ of $\mathcal{A}$ 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 TAs $\mathcal{A}_1$, with s_1 states and k_1 clocks, and $\mathcal{A}_2$, with s_2 states and 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 calculated, and has $s_1 + s_2$ states (resp., $s_1 \cdot s_2$ states) and $k_1 + k_2$ clocks (resp., $k_1 + k_2$ clocks).

The logic MTL.. Let us now recall the framework of Metric Temporal Logic (MTL) [19], a well-known timed linear-time temporal logic which extends standard LTL with time constraints on the until modality.

Given a finite set $\mathcal{AP}$ of proposition letters, the set of MTL formulas φ over $\mathcal{AP}$ is defined by the following 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 over timed words over $2^{\mathcal{AP}}$. Given an MTL formula φ, a timed word $w = (\sigma, \tau)$ over $2^{\mathcal{AP}}$, and a position $0 \leq i < |w|$, 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 $(w, j) \models \varphi_2$, $\tau_j - \tau_i \in I$, 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* $\mathcal{L}_T(\varphi)$ of φ is the set of models of φ.

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

In MTL, we use standard shortcuts such as: $\mathsf{F}_I\varphi$ for $\varphi \vee (\top \mathsf{U}_I \varphi)$ (*timed eventually* or *timed future*), and $\mathsf{G}_I\varphi$ for $\neg\mathsf{F}_I\neg\varphi$ (*timed always* or *timed globally*).

We also consider two fragments of MTL, namely, MITL (Metric Interval Temporal Logic) and $\mathsf{MITL}_{(0,\infty)}$ [2]: MITL is obtained by allowing only non-singular intervals of $Intv$ at the subscript of U, while $\mathsf{MITL}_{(0,\infty)}$ is the fragment of MITL obtained 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) the U_I modality in φ.

3.2. *Reduction to existential MC for* TAs *against* MTL

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

In the following, we fix an instance $P = (SV, R)$ 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:

1. 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;
2. 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;
3. 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 exists a future plan for $P = (SV, R)$ if and only if $\mathcal{L}_T(\mathcal{A}_{SV}) \cap \mathcal{L}_T(\mathcal{A}_\exists) \cap \mathcal{L}_T(\varphi_\forall) \neq \emptyset$.

For each $x \in SV$, we let $x = (V_x, T_x, D_x)$. Given an interval $I \in \mathit{Intv}$ and a natural number $n \in \mathbb{N}$, let $n + I$ (respectively, $n - I$) denote the set of non-negative real numbers $\tau \in \mathbb{R}_+$ such that $\tau - n \in I$ (respectively, $n - \tau \in I$). Note that $n + I$ (respectively, $n - I$) 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 $o \leq^{\mathsf{e}}_I n$ (resp., $n \leq^{\mathsf{e}}_I o$), then $I(\rho) = n - I$ (resp., $I(\rho) = n + I$).

We finally define Intv_R as the set of intervals $J \in \mathit{Intv}$ such that $J = I(\rho)$ for some time-point atom ρ occurring in a trigger rule of R.

Encodings of multi-timelines of SV.. We assume that for distinct state variables x and x', the sets V_x and $V_{x'}$ are disjoint. We exploit the following set $\mathcal{AP}$ of proposition letters to encode multi-timelines of SV:

$$\mathcal{AP} = \bigcup_{x \in SV} \mathit{Main}_x \cup \mathit{Deriv},$$

$$\mathit{Main}_x = ((\{\mathit{beg}_x\} \cup V_x) \times V_x) \cup (V_x \times \{\mathit{end}_x\}),$$

$$\mathit{Deriv} = \mathit{Intv}_R \cup \{p_>\} \cup \bigcup_{x \in SV} \bigcup_{v \in V_x} \{\mathit{past}^{\mathsf{s}}_v, \mathit{past}^{\mathsf{e}}_v\}.$$

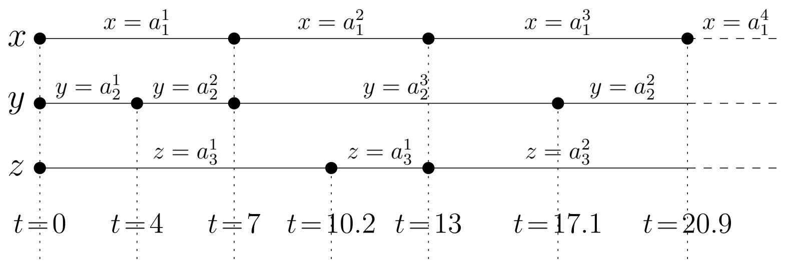

Figure 3: An example of multi-timeline of $SV = \{x, y, z\}$, where in particular $V_x = \{a_1^i \mid 1 \leq i \leq 4\}$, $V_y = \{a_2^i \mid 1 \leq i \leq 3\}$ and $V_z = \{a_3^i \mid 1 \leq i \leq 2\}$. The encoding of the timeline for x depicted in the figure (we show only values in $Main_x$) is $\big(\{(beg_x, a_1^1)\}, 0\big)\big(\{(a_1^1, a_1^2)\}, 7\big)\big(\{(a_1^2, a_1^3)\}, 13\big)\big(\{(a_1^3, a_1^4)\}, 20.9\big)\cdots$ The encoding of the multi-timeline of SV depicted in the figure (we show only values in $Main_x \cup Main_y \cup Main_z$) is $\big(\{(beg_x, a_1^1), (beg_y, a_2^1), (beg_z, a_3^1)\}, 0\big)\big(\{(a_2^1, a_2^2)\}, 4\big)$ $\big(\{(a_1^1, a_1^2), (a_2^2, a_2^3)\}, 7\big)\big(\{(a_3^1, a_3^1)\}, 10.2\big)\big(\{(a_1^2, a_1^3), (a_3^1, a_3^2)\}, 13\big)\big(\{(a_2^3, a_2^2)\}, 17.1\big)\cdots$

Intuitively, we use the propositions in $Main_x$ to encode a token along a timeline for x. The propositions in $Deriv$, as explained below, represent enrichments of the encoding, used for translating simple trigger rules in MTL formulas under the future semantics. 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 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 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. See Figure 3 for an example.

Now we explain the meaning of 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 tag $p_>$ keeps track of whether the current timestamp is strictly greater than the previous one. Finally, the propositions in $\bigcup_{x \in SV} \bigcup_{v \in V_x} \{past_v^{\mathsf{s}}, past_v^{\mathsf{e}}\}$ keep track of past token events occurring at timestamps *coinciding* with the current timestamp.

We start by defining the encoding of timelines for $x \in SV$. 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

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

- $S_i \cap \mathit{Intv}_R$ is the set of intervals $I \in \mathit{Intv}_R$ such that $\tau_i \in I$;
- $p_> \in S_i$ iff either $i = 0$ or $\tau_i > \tau_{i-1}$;
- for all $v \in V_x$, $\mathit{past}^{\mathsf{s}}_v \in S_i$ (resp., $\mathit{past}^{\mathsf{e}}_v \in S_i$) iff 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 timed word w encodes the timeline for x of length $n + 1$ given by $\pi = (v_0, \tau_1)(v_1, \tau_2 - \tau_1) \cdots (v_n, \tau_{n+1} - \tau_n)$. Note that in the encoding, τ_i and τ_{i+1} represent the start time and the end time of the i-th token of the timeline π $(0 \leq i \leq n)$. See the caption of Figure 3 for an example of encoding.

Next, we define the encoding of a multi-timeline of SV. For a set $P \subseteq \mathcal{AP}$ and $x \in SV$, let $P[x] = P \setminus \bigcup_{y \in SV \setminus \{x\}} \mathit{Main}_y$. 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 the following conditions hold:

- 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 $P_i[x] \cap \mathit{Main}_x = \emptyset$ is an encoding of a timeline for x;
- $P_0[x] \cap \mathit{Main}_x \neq \emptyset$ for all $x \in SV$ (initialization).

See again Figure 3 for an example of 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, as shown in the proof of the next proposition.

Proposition 11. *One can construct in exponential time 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*.*

Proof. Let us fix an ordering $SV = \{x_1, \ldots, x_N\}$ of the state variables. Let $\mathcal{H} = \mathit{Deriv} \setminus (\mathit{Intv}_R \cup \{p_>\})$ and $V'_i = V_{x_i} \cup \{\mathit{beg}_{x_i}, \mathit{end}_{x_i}\}$ for all $1 \leq i \leq N$.

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 $v_i \notin \{\mathit{beg}_{x_i}, \mathit{end}_{x_i}\}$. If $v_i = \mathit{beg}_{x_i}$ (resp., $v_i = \mathit{end}_{x_i}$), 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.
- The initial state q_0 is $(\mathit{beg}_{x_1}, \ldots, \mathit{beg}_{x_N}, \emptyset)$.
- The set F of accepting states is the set of all states $(\mathit{end}_{x_1}, \ldots, \mathit{end}_{x_N}, H)$ for any $H \subseteq \mathcal{H}$.

- 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 $p_>$, whereas c_{glob} is a clock that measures the current (global) time and is never reset.

- The relation Δ consists of the transitions

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

  such that:

  - if $(v_1, \ldots, v_N, H) = q_0$, then $P \cap \mathit{Main}_x \neq \emptyset$ for all $x \in \mathit{SV}$ (this ensures initialization);
  - for all $1 \leq i \leq N$, the following holds:
    - *either* $P \cap \mathit{Main}_{x_i} = \emptyset$, $v'_i = v_i$, $\theta_i = \top$, and $c_i \notin \mathit{Res}$ (intuitively, no event associated with x_i occurs in this case),
    - *or* $P \cap \mathit{Main}_{x_i} = (v_i, v'_i)$ (hence, $v_i \neq \mathit{end}_{x_i}$), $v'_i \in T_{x_i}(v_i)$ if both $v_i \in V_{x_i}$ and $v'_i \in V_{x_i}$; $c_i \in \mathit{Res}$ and $\theta_i = c_i \in D_{x_i}(v_i)$ (resp., $\theta_i = c_i \in [0, 0]$) if $v_i \neq \mathit{beg}_{x_i}$ (resp., if $v_i = \mathit{beg}_{x_i}$);
  - $c_{glob} \notin \mathit{Res}$ and

$$\theta_{glob} = \bigwedge_{I \in P \cap \mathit{Intv}_R} c_{glob} \in I \wedge \bigwedge_{I \in \mathit{Intv}_R \setminus P} (c_{glob} \in \overrightarrow{I} \vee c_{glob} \in \overleftarrow{I}),$$

    where, for each $I \in \mathit{Intv}_R \setminus P$, $\overrightarrow{I}$ and $\overleftarrow{I}$ are (possibly empty) maximal intervals in $\mathbb{R}_+$ disjoint from I (e.g., if $I = [3, 5[$, then $\overleftarrow{I} = [0, 3[$ and $\overrightarrow{I} = [5, +\infty[$). Note that $\overrightarrow{I}, \overleftarrow{I} \in \mathit{Intv}$. Recall that, for each $I \in \mathit{Intv}_R$, I must be in P if and only if the current time (given by c_{glob}) is in I;
  - $c_> \in \mathit{Res}$; moreover, if $(v_1, \ldots, v_N, H) = q_0$, then $p_> \in P$ and $\theta_> = \top$, otherwise, *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 $x \in \mathit{SV}$ and $v \in V_x$, $\mathit{past}^{\mathsf{s}}_v \in H'$ iff *either* $P \cap \mathit{Main}_{x_i}$ is of the form (v', v), *or* $p_> \notin P$ and $\mathit{past}^{\mathsf{s}}_v \in H$;
  - for all $x \in \mathit{SV}$ and $v \in V_x$, $\mathit{past}^{\mathsf{e}}_v \in H'$ iff *either* $P \cap \mathit{Main}_{x_i}$ is of the form (v, v'), *or* $p_> \notin P$ and $\mathit{past}^{\mathsf{e}}_v \in H$.

This concludes the proof. □

Encodings of simple trigger rules by MTL *formulas..* We now construct an MTL formula $\varphi_\forall$ over $\mathcal{AP}$ capturing the simple trigger rules in R, under the future semantics.

Proposition 12. *One can construct in linear time 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* $\varphi_\forall$ *iff* Π *satisfies all the simple trigger rules in* R *under the future semantics.*

The formula $\varphi_\forall$ *is an* MITL *formula (resp.,* $\mathsf{MITL}_{(0,\infty)}$ *formula) if the intervals in the trigger rules are non-singular (resp., belong to* $Intv_{(0,\infty)}$*).*

The formula $\varphi_\forall$ *has* $O(|R| \cdot N_A \cdot N_{\mathcal{E}} \cdot \big(|Intv_R| + (\sum_{x\in SV} |V_x|)^2\big))$ *distinct subformulas, with* N_A *the maximum number of atoms in a trigger rule of* R*, and* $N_{\mathcal{E}}$ *the maximum number of existential statements in a trigger rule of* R*.*

Proof. We first introduce some auxiliary propositional (Boolean) formulas over $\mathcal{AP}$. Let $x \in SV$ and $v \in V_x$. We denote by $\psi(\mathsf{s}, v)$ and $\psi(\mathsf{e}, v)$ the two propositional formulas over $Main_x$ defined as follows:

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

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

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 use 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. Additionally, given an MTL formula θ, we define the MTL formula

$$EqTime(\theta) = \theta \vee [\neg p_> \mathsf{U}_{\geq 0} (\neg p_> \wedge \theta)]$$

which is satisfied by an encoding of a multi-timeline of SV 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 $o_t[x_t = v_t] \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$. Then $\varphi_{\mathcal{R}}$ is given by

$$\varphi_{\mathcal{R}} = \mathsf{G}_{\geq 0}\big(\psi(\mathsf{s}, v_t) \to \bigvee_{i=1}^{k} \Phi_{\mathcal{E}_i}\big),$$

where $\Phi_{\mathcal{E}_i}$, with $1 \leq i \leq k$, ensures the fulfillment of the existential statement $\mathcal{E}_i$ of $\mathcal{R}$ under the future semantics.

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 atom in $\mathbf{A}$ where o occurs.

For each token name $o \in \{o_t\} \cup O$, we denote by $\mathit{Intv}_o^{\mathsf{s}}$ (resp., $\mathit{Intv}_o^{\mathsf{e}}$) the set of intervals $J \in \mathit{Intv}$ 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. Note that $\mathit{Intv}_o^{\mathsf{s}}, \mathit{Intv}_o^{\mathsf{e}} \subseteq \mathcal{AP}$, and we exploit the propositional formulas $\xi_o^{\mathsf{s}} = \bigwedge_{I \in \mathit{Intv}_o^{\mathsf{s}}} I$ and $\xi_o^{\mathsf{e}} = \bigwedge_{I \in \mathit{Intv}_o^{\mathsf{e}}} I$ to ensure the fulfillment of the time constraints imposed by the time-point atoms associated with the token o.

The MTL formula $\Phi_{\mathcal{E}}$ is thus given by:

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

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

The construction of χ_ρ depends on the form of ρ. We distinguishes four cases.

1. $\rho = o \leq_I^{e_1,e_2} o_t$ and $o \neq o_t$. We assume $0 \in I$ (the other case being simpler). First, assume that $e_2 = \mathsf{s}$. Under the future semantics, ρ holds iff the start time of the trigger token o_t coincides with the e_1-time of token o. Hence, in this case ($e_2 = \mathsf{s}$), χ_ρ is given by:

   $$\chi_\rho = \xi_o^{e_1} \wedge \big(\mathit{past}_{\mathit{val}(o)}^{e_1} \vee \mathit{EqTime}(\psi(e_1, \mathit{val}(o)))\big).$$

   If instead $e_2 = \mathsf{e}$, then χ_ρ is defined as follows:

   $$\begin{aligned}\chi_\rho = \big[\psi_{\neg x_t} \mathsf{U}_{\geq 0}\{\xi_o^{e_1} \wedge \psi(e_1, \mathit{val}(o)) \wedge \psi_{\neg x_t} \wedge (\psi_{\neg x_t} \mathsf{U}_I \psi(\mathsf{e}, v_t))\}\big] \vee \\ \big[(\psi(e_1, \mathit{val}(o)) \vee \mathit{past}_{\mathit{val}(o)}^{e_1}) \wedge \xi_o^{e_1} \wedge \\ \big(\mathit{EqTime}(\psi(\mathsf{e}, v_t)) \vee (\psi_{\neg x_t} \wedge (\psi_{\neg x_t} \mathsf{U}_I \psi(\mathsf{e}, v_t)))\big)\big] \vee \\ \big[\psi_{\neg x_t} \mathsf{U}_{\geq 0}\{\psi(\mathsf{e}, v_t) \wedge \mathit{EqTime}(\psi(e_1, \mathit{val}(o)) \wedge \xi_o^{e_1})\}\big].\end{aligned}$$

   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 considers the case where the e_1-event of token o precedes the start-event of the trigger token: thus, under the future semantics, it holds that the e_1-time of token o coincides with the start time of the trigger token. Finally, the third disjunct considers the case where the e_1-event of token o follows the end-event of the trigger (hence, the related timestamps must coincide).

2. $\rho = o_t \leq_I^{e_1,e_2} o$ and $o \neq o_t$. We assume $e_1 = \mathsf{e}$ and $0 \in I$ (the other cases being simpler). Then,

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

   where the second disjunct captures the situation where the e_2-time of o coincides with the end time of the trigger token o_t, but the e_2-event of o occurs before the end-event of the trigger token.

3. $\rho = o_t \leq_I^{e_1,e_2} o_t$. This case is straightforward and we omit the details.
4. $\rho = o_1 \leq_I^{e_1,e_2} o_2$, with $o_1 \neq o_t$ and $o_2 \neq o_t$. We assume $o_1 \neq o_2$ and $0 \in I$ (the other cases are simpler). Then,

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

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

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

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 construct a TA for $\mathcal{E}$ which guesses such a chain of events and then checks the temporal relations by means of suitable clock constraints and clock resets. Finally, by the closure of TAs under language union [1], we can build a TA for the whole trigger-less rule. Additionally, exploiting also the closure of TAs under intersection, we construct a TA accepting (encodings of) multi-timelines satisfying all trigger-less rules.

Proposition 13. *One can construct in exponential time 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 $\mathcal{A}_\exists$ iff Π satisfies all the trigger-less rules in R.*

$\mathcal{A}_\exists$ 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.

We recall that, in the encoding of multi-timelines of SV, we assume that, for distinct state variables $x, x' \in SV$, the domains V_x and $V_{x'}$ are disjoint.

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 effective 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

$\mathit{Intv}^{\mathsf{s}}_o$ (resp., $\mathit{Intv}^{\mathsf{e}}_o$) the set of intervals $J \in \mathit{Intv}$ 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^{s}_o and c^{e}_o which, intuitively, are reset when the token chosen for o starts and ends, respectively. The clocks c^{s}_o and c^{e}_o are non-deterministically reset when a start-event for $\mathit{val}(o)$ and the related end-event occur along an encoding of a multi-timeline. The automaton $\mathcal{A}_{\mathcal{E}}$ ensures that the clocks c^{s}_o and c^{e}_o are reset exactly once. $\mathcal{A}_{\mathcal{E}}$ moves to an accepting state only if all the clocks c^{s}_o and c^{e}_o 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^{s}_o (resp., c^{e}_o) is reset, we require that the clock constraint $\bigwedge_{I \in \mathit{Intv}^{\mathsf{s}}_o} c_{glob} \in I$ (resp., $\bigwedge_{I \in \mathit{Intv}^{\mathsf{e}}_o} c_{glob} \in I$) 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 $\{c_{glob}\} \cup \bigcup_{o \in O}\{c^{\mathsf{s}}_o, c^{\mathsf{e}}_o\}$.
- The set of states is $2^{C \setminus \{c_{glob}\}}$. Intuitively, a state keeps track of the clocks in $C \setminus \{c_{glob}\}$ which have been reset so far.
- The initial state q_0 is $\emptyset$.
- 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 $(C_1, P, \theta \wedge \theta_{glob}, \mathit{Res}, C_2)$ such that *either* (*i*) $C_1 = C \setminus \{c_{glob}\}$, $C_2 = C_1$, $\mathit{Res} = \emptyset$, $\theta = \top$, 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 $c^{\mathsf{s}}_o \in C_2 \setminus C_1$, there is a main proposition in P of the form $(v', \mathit{val}(o))$ for some v'.
  - for each $o \in O$, $c^{\mathsf{e}}_o \in C_2 \setminus C_1$ if and only if $c^{\mathsf{s}}_o \in C_1$ and $(\mathit{val}(o), v') \in P$ 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 $\theta = \bigwedge_{\rho \in \mathbf{A}} \mathit{code}(\rho)$, where $\mathbf{A}$ is the set of *interval* atoms of $\mathcal{E}$ and for each interval atom $\rho \in \mathbf{A}$ of the form $o_1 \leq^{e_1, e_2}_{I} o_2$, the clock constraint $\mathit{code}(\rho)$ is defined as follows:
    - ∗ if $c^{e_2}_{o_2} \notin C_1$ and $c^{e_1}_{o_1} \notin C_1$, then $\mathit{code}(\rho) = c^{e_2}_{o_2} - c^{e_1}_{o_1} \in I$ (in this case, both $c^{e_2}_{o_2}$ and $c^{e_1}_{o_1}$ 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 $c^{e_2}_{o_2} - c^{e_1}_{o_1} = 0 \in I$ for the atom to be satisfied);

    * if $c^{e_2}_{o_2} \in C_1$ and $c^{e_1}_{o_1} \in C_1$, then $code(\rho) = c^{e_1}_{o_1} - c^{e_2}_{o_2} \in I$;
    * if $c^{e_2}_{o_2} \in C_1$ and $c^{e_1}_{o_1} \notin C_1$, then $code(\rho) = c^{e_2}_{o_2} \in [0,0] \wedge c^{e_2}_{o_2} \in I$ (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 $c^{e_2}_{o_2} \notin C_1$ and $c^{e_1}_{o_1} \in C_1$, then $code(\rho) = c^{e_1}_{o_1} \in I$.

- $\theta_{glob} = \displaystyle\bigwedge_{c^e_o \in C_2 \setminus C_1} \bigwedge_{I \in Intv^e_o} c_{glob} \in I$.
- $Res = C_2 \setminus C_1$.

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

Given a trigger-less rule $\mathcal{R} = \top \to \mathcal{E}_1 \vee \mathcal{E}_2 \vee \ldots \vee \mathcal{E}_k$, 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. □

Conclusion of the construction.. By applying Proposition 11, 12, 13 and well-known results about TAs and MTL over finite timed words [1, 23], we obtain the main result of this section.

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., in **PSPACE**).*

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

By Proposition 11, 12, 13 and the effective closure of TAs under language intersection [1], 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 [23], checking non-emptiness of $\mathcal{L}_T(\mathcal{A}_P) \cap \mathcal{L}_T(\varphi_\forall)$ is decidable. Thus the first part of the theorem holds.

As for the second part, let us assume that the intervals in the trigger rules are non-singular (resp., belong to $Intv_{(0,\infty)}$). By Proposition 12, $\varphi_\forall$ is an MITL (resp., $\mathsf{MITL}_{(0,\infty)}$) formula. By [2], one can build a TA $\mathcal{A}_\forall$ accepting $\mathcal{L}_T(\varphi_\forall)$ having

- $2^{O(K_P \cdot |R| \cdot N_A \cdot N_\mathcal{E} \cdot (|Intv_R| + N_v^2))}$ 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 $\mathcal{A}$,[3] which uses work space *logarithmic* in the number of control 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 constructed on the fly—that is, by looking at their transition relations Δ, one can determine, given a state q, a successor q' and the connecting transition, along with the associated constraints and clocks to reset—and the search in the region automaton of $\mathcal{A}_\wedge$ can be done without explicitly constructing $\mathcal{A}_\wedge$, the result follows. $\square$

In the next section, we consider future TP with simple trigger rules and non-singular intervals in the atoms of trigger rules (resp., intervals in $Intv_{(0,\infty)}$), and prove a matching complexity *lower bound*: **EXPSPACE**-completeness (resp., **PSPACE**-completeness) of the problem follows.

4. 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 exponential length*, which is known to be **EXPSPACE**-complete [16]. Since the reduction is standard, we refer the reader to Appendix C for the details of the construction.

Theorem 15. *The future TP problem, even with* one state variable*, with simple trigger rules and non-singular intervals is **EXPSPACE**-hard (under polynomial-time reductions).*

By putting together Theorem 14, **EXPSPACE**-completeness follows.

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 to it in polynomial time.

[3] The *region automaton* of $\mathcal{A}$ features states of the form (q, r), where q is a state of $\mathcal{A}$ and r a *region*: every region specifies, for each clock c of $\mathcal{A}$, whether its value is integer or not (and, if it is, its value up to K_c, the maximum constant to which c is compared), and the ordering of the fractional parts of the clocks.

The problem *periodic SAT* is defined as follows [24]. 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 [24]; 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 16. *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

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

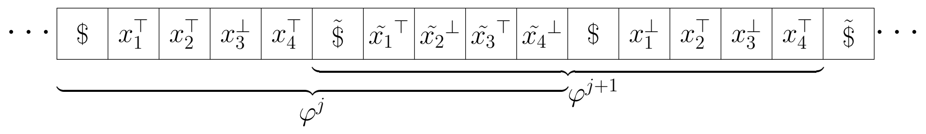

Figure 4: Let the formula φ be defined over two sets of variables, $\Gamma = \{x_1, x_2, x_3, x_4\}$ and $\Gamma^{+1} = \{x_1^{+1}, x_2^{+1}, x_3^{+1}, x_4^{+1}\}$. The j-th copy (we assume j is odd) of φ, i.e., φ^j, is satisfied by the assignment $x_1^j \mapsto \top$, $x_2^j \mapsto \top$, $x_3^j \mapsto \bot$, $x_4^j \mapsto \top$, $x_1^{j+1} \mapsto \top$, $x_2^{j+1} \mapsto \bot$, $x_3^{j+1} \mapsto \top$, $x_4^{j+1} \mapsto \bot$. 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., $x_i^\bot$) 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., $\tilde{x_i}^\bot$) represents a positive (resp., negative) assignment of x_i^j, for some *even* $j \geq 2$. See Figure 4 for an example.

We start with the next simple trigger rules, one for each $v \in V$:

$$o[y = v] \to o \leq^{\mathsf{s,e}}_{[0,2]} o.$$

Paired with the constraint function D, they enforce all tokens' durations to be *exactly* 2: 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, the density of the time domain does not play any role in this proof.

We now add the next rules:

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

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 next rules, that 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:

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

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 $\exists o''[y = \mathit{stop}].o \leq^{\mathsf{e,s}}_{[0,2n]} o''$ 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 16, **PSPACE**-completeness of future TP with simple trigger rules and intervals in $\mathit{Intv}_{(0,\infty)}$ follows.

In the next section we focus on a different restriction of the TP problem, which will allow us to devise a **NP** planning algorithm for it.

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

In this section we describe a TP algorithm, for planning domains where only trigger-less rules are allowed, which requires a polynomial number of (non-deterministic) computation steps. We recall that trigger-less rules are useful, for instance, to express initial, intermediate conditions and reachability goals.

We want to start with the following example, with which we highlight that there is no *polynomial-size* plan for some problem instances/domains. Thus, an explicit enumeration of all tokens of a multi-timeline *does not* represent a suitable polynomial-size certificate.

Example 17. Let us consider the following planning domain. We denote by $p(i)$ the i-th prime number, assuming $p(1) = 1$, $p(2) = 2$, $p(3) = 3$, $p(4) = 5, \ldots$. We define, for $i = 1, \ldots, n$, the state variables $x_i = (\{v_i\}, \{(v_i, v_i)\}, D_{x_i})$ with $D_{x_i}(v_i) = [p(i), p(i)]$. The following rule

$$\top \rightarrow \exists o_1[x_1 = v_1] \cdots \exists o_n[x_n = v_n]. \bigwedge_{i=1}^{n-1} o_i \leq^{\mathsf{e,e}}_{[0,0]} o_{i+1}$$

is asking for 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: no explicit polynomial-time enumeration of such tokens is possible.

As a consequence, there exists no trivial guess-and-check **NP** algorithm. Conversely, one can easily prove the following result.

Theorem 18. *The TP problem with trigger-less rules only is **NP**-hard, even with one state variable (under polynomial-time reductions).*

Proof. There is a trivial reduction from the problem of the existence of a Hamiltonian path in a directed graph.

Given a directed graph $G = (V, E)$, with $|V| = n$, we define the state variable $x = (V, E, D_x)$, where $D_x(v) = [1, 1]$ for each $v \in V$. We add the following trigger-less rules, one for each $v \in V$:

$$\top \to \exists o[x = v].o \geq^{\mathsf{s}}_{[0,n-1]} 0.$$

The rule for $v \in V$ requires that there is a token $(x, v, 1)$ along the timeline for x, which starts no later than $n - 1$. It is easy to check that G contains a Hamiltonian path if and only if there exists a plan for the defined planning domain. $\square$

We now present the aforementioned non-deterministic polynomial-time algorithm, proving that timeline-based planning with trigger-less rules is in **NP**.

We preliminarily have to derive a *finite horizon* (namely, the 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 calculate a *bound to the maximum number of tokens* in a plan. Both can be obtained from the constructions of the TAs described in the proof of Theorem 14: since only trigger-less rules are now allowed, we disregard the construction of the MTL formula $\varphi_\forall$, and restrict our attention to the TA $\mathcal{A}_P$ (i.e., the intersection between $\mathcal{A}_{SV}$ for the state variables in SV from Proposition 11 and $\mathcal{A}_\exists$ for the trigger-less rules in R from Proposition 13), which 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 overall number of quantifiers in the trigger-less rules of R, and accepts all and only the encodings w_Π of multi-timelines Π of SV satisfying all the trigger-less rules in R.

The language emptiness checking algorithm for TAs executed over $\mathcal{A}_P$ 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[4], trying to find a path, from the initial state to a final state, whose length can clearly be bounded by the number of states. We observe that each edge/transition of the region automaton in such a path corresponds,

[4]The factor $2^{2N_q^2}$ is present due to diagonal clock constraints in $\mathcal{A}_P$.

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, which is $\alpha \cdot |SV|$. We can also derive a bound on the horizon of the plan, which is $\alpha \cdot |SV| \cdot (\alpha_K + 1)$, as every transition taken in $\mathcal{A}_P$ may let at most $\alpha_K + 1$ time units pass, as α_K accounts in particular for the maximum constant to which a (any) clock is compared.[5]

Having this pair of bounds, we are now ready to describe the two main phases of the algorithm, corresponding to the following pair of observations. On the one hand, (i) each trigger-less rule requires, as we said, 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 the quantifiers of one of its existential statements). On the other hand, (ii) timelines for different state variables evolve independently of each other. In order to deal with (i), we non-deterministically position such temporal events along timelines; as for (ii), we enforce a correct evolution of each timeline between pairs of "positioned" events, completely independently of the other timelines.

Non-deterministic token positioning. The algorithm starts by non-deterministically selecting, for every trigger-less rule in R, a disjunct—and deleting all the others. Then, for every (left) quantifier $o_i[x_i = v_i]$, it generates the integer part of both the start and the end time of the token for x_i to which o_i is mapped. We call such time instants, respectively, $\mathsf{s}_{int}(o_i)$ and $\mathsf{e}_{int}(o_i)$.[6] We observe that all start/end time $\mathsf{s}_{int}(o_i)$ and $\mathsf{e}_{int}(o_i)$, being less or equal to $\alpha \cdot |SV| \cdot (\alpha_K + 1)$ (the finite horizon bound), have an integer part that can be encoded with polynomially many bits (and thus can be generated in polynomial time).

Let us now consider the fractional parts of the start/end time of the tokens associated with quantifiers. 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 integer ($\mathsf{s}_{frac}(o_i) = 0$, $\mathsf{e}_{frac}(o_i) = 0$) or not ($\mathsf{s}_{frac}(o_i) > 0$, $\mathsf{e}_{frac}(o_i) > 0$). Every such possibility can be generated in polynomial time.

Some trivial tests should now be performed, namely that, for all o_i, $\mathsf{s}_{int}(o_i) \leq \mathsf{e}_{int}(o_i)$, each token is assigned an end time equal or greater than its start time, and no two tokens for the same variable are overlapping.

It is routine to check that, if we change the start/end time of (some of the) tokens associated with quantifiers, but we leave unchanged (i) all the integer parts, (ii) zeroness/non-zeroness of fractional parts, and (iii) the fractional parts' order, then the satisfaction of the (atoms in the) trigger-less rules does

[5]Clearly, and unbounded quantity of time units may pass, but after $\alpha_K + 1$ the last region of the region automaton will certainly have been reached.

[6]We can assume w.l.o.g. that all quantifiers refer to distinct tokens. As a matter of fact, the algorithm can non-deterministically choose to make two (or more) quantifiers $o_i[x_i = v_i]$ and $o_j[x_i = v_i]$ over the same variable and value "collapse" to the same token just by rewriting all occurrences of o_j as o_i in the atoms of the rules.

not change. This is due to all the constants being integers.[7] Therefore we can now check whether all rules are satisfied.

Enforcing legal token durations and timeline evolutions. We now continue by checking that: (i) all tokens associated with a quantifier have a legal duration, and that (ii) there exists a legal timeline evolution between pairs of adjacent such tokens over the same variable (here *adjacent* means that there is no other token associated with a quantifier in between). We will enforce all these requirements 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 linear programs. We shall show later how to convert these into 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 every token for a quantifier $o_i[x_i = v_i]$. First, we add the linear constraints

$$0 \leq \alpha_{o_i,s} < 1, \quad 0 \leq \alpha_{o_i,e} < 1.$$

Then, we also need to enforce that the values of $\alpha_{o_i,s}, \alpha_{o_i,e}$ respect the decided order of the fractional parts: for example,

$$0 = \alpha_{o_i,s} = \alpha_{o_j,s} < \alpha_{o_k,s} < \ldots < \alpha_{o_j,e} < \alpha_{o_i,e} = \alpha_{o_k,e} < \ldots$$

To enforce requirement (i), we set, for all $o_i[x_i = v_i]$,

$$a \leq (\mathsf{e}_{int}(o_i) + \alpha_{o_i,e}) - (\mathsf{s}_{int}(o_i) + \alpha_{o_i,s}) \leq b$$

where $D_{x_i}(v_i) = [a, b]$. Clearly, strict ($<$) inequalities must be used for a left/right open interval.

To enforce requirement (ii), namely that there exists a legal timeline evolution between each pair of adjacent tokens for the same state variable, say $o_i[x_i = v_i]$ and $o_j[x_i = v_j]$, we proceed as follows (for a correct evolution between $t = 0$ and the first token, analogous considerations can be made).

Let us consider each state variable $x_i = (V_i, T_i, D_i)$ as a directed graph $G = (V_i, T_i)$ where D_i is a function associating with each vertex $v \in V_i$ a duration range. We have to decide whether or not there exist

- a path in G, possibly with repeated vertices and edges, $v_0 \cdot v_1 \cdots v_{n-1} \cdot v_n$, where $v_0 \in T_i(v_i)$ and v_n with $v_j \in T_i(v_n)$ are non-deterministically generated, and
- 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 $s = 0, \ldots, n$, $d_s \in D_i(v_s)$.

[7]We may observe that, by leaving unchanged all the integer parts and the fractional parts' order, the region of the region graph of the timed automaton does not change.

We guess a set of integers $\{\alpha'_{u,v} \mid (u,v) \in T_i\}$. Intuitively, $\alpha'_{u,v}$ is the number of times the solution path traverses (u,v). 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 $\alpha \cdot |SV|$. Hence the binary encoding of $\alpha'_{u,v}$ can be generated in polynomial time.

We then perform the following deterministic steps.

1. We consider the subset E' of edges of G, $E' = \{(u,v) \in T_i \mid \alpha'_{u,v} > 0\}$. We check whether E' induces a strongly (undirected) connected subgraph of G.
2. We check whether
   - $\sum_{(u,v)\in E'} \alpha'_{u,v} = \sum_{(v,w)\in E'} \alpha'_{v,w}$, for all $v \in V_i \setminus \{v_0, v_n\}$;
   - $\sum_{(u,v_0)\in E'} \alpha'_{u,v_0} = \sum_{(v_0,w)\in E'} \alpha'_{v_0,w} - 1$;
   - $\sum_{(u,v_n)\in E'} \alpha'_{u,v_n} = \sum_{(v_n,w)\in E'} \alpha'_{v_n,w} + 1$.
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 real non-negative 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 following constraints:
$$a \cdot y_v \leq z_v \leq b \cdot y_v,$$
where $D_i(v) = [a,b]$ (an analogous constraint should be written 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.) together check that the values $\alpha'_{u,v}$ for the arcs specify *a directed Eulerian path from v_0 to v_n in a multigraph.* Indeed, the following theorem holds.

Theorem 19. [18] *Let $G' = (V', E')$ be a directed multigraph (E' is a multiset). G' has a (directed) Eulerian path from v_0 to v_n if and only if:*

- *the undirected version of G' is connected, and*
- $|\{(u,v) \in E'\}| = |\{(v,w) \in E'\}|$, *for all* $v \in V' \setminus \{v_0, v_n\}$;
- $|\{(u,v_0) \in E'\}| = |\{(v_0,w) \in E'\}| - 1$;
- $|\{(u,v_n) \in E'\}| = |\{(v_n,w) \in E'\}| + 1$.

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, then every single visit must take at least a and at most b units of time. Hence the overall visitation time is in between $a \cdot y_v$ and $b \cdot y_v$. Vice versa, if the total visitation time is in between $a \cdot y_v$ and $b \cdot y_v$, then it can be slit into y_v intervals, each one falling into $[a,b]$.

The algorithm concludes by solving the linear program given by the variables $\alpha_{o_i,s}$ and $\alpha_{o_i,e}$ for each quantifier $o_i[x_i = v_i]$, and for each pair of adjacent tokens in the same timeline for x_i, for each $v \in V_i$, the variables z_v subject to their constraints.

Finally, in order 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 written so far 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 $\xi s - \eta q - k + \beta_t \le 0$ and $\xi s - \eta q - k - \beta_t \ge 0$, where β_t is an additional fresh non-negative variable, which is *local* to a single constraint. We observe that 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 $r \le \beta_t$, 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 planning algorithm. We can thus conclude the section with the main result.

Theorem 20. *The TP problem with trigger-less rules only is **NP**-complete.*

In the next section, using the results on the variants of TP, we move to a different problem, namely, we shall study MC for MITL specifications over timelines.

6. MC for MITL over timelines

In this section we show how it is possible to model check systems specified in terms of timelines. More precisely, a system is described as a set of state variables along with a set of synchronization rules over them (a TP domain) $P = (SV, R)$. The property specification language we will be assuming is the logic MITL.

We first recall the encoding of multi-timelines already adopted in Section 3.2, over which we interpret MITL, that exploits the set $\mathcal{AP}$ of proposition letters

$$\mathcal{AP} = \bigcup_{x \in SV} Main_x \cup Deriv,$$

where in particular

$$Main_x = ((\{beg_x\} \cup V_x) \times V_x) \cup (V_x \times \{end_x\}).$$

The tags beg_x and end_x mark the beginning and the end of a timeline for x, and a pair $(v, v') \in Main_x$ represents a transition of the value taken by x from

v to v' (a token for x with value v' follows a token with value v). The already introduced formula

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

states that a start-event for a token for x with value v occurs at the current time. Finally, $EqTime(\theta) = \theta \vee [\neg p_> \mathsf{U}_{\geq 0}(\neg p_> \wedge \theta)]$, where $p_> \in Deriv$, 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 one.

Before formalizing the *MC problem for* MITL *formulas over timelines*, we want to start with an easy example of a system whose components are described by timelines, over which we check some properties encoded by MITL formulas.

Example 21. Let us consider the following system, consisting of a *temperature sensor*, a *processing unit*, and a *data transmission unit.* These components are modelled by three state variables, $x_{\texttt{temp}} = (V_{\texttt{temp}}, T_{\texttt{temp}}, D_{\texttt{temp}})$, $x_{\texttt{proc}} = (V_{\texttt{proc}}, T_{\texttt{proc}}, D_{\texttt{proc}})$ and $x_{\texttt{transm}} = (V_{\texttt{transm}}, T_{\texttt{transm}}, D_{\texttt{transm}})$, where

- $V_{\texttt{temp}} = \{\texttt{ready}, \texttt{not\_ready}\}$,
  $T_{\texttt{temp}}(\texttt{ready}) = \{\texttt{not\_ready}\}$, $T_{\texttt{temp}}(\texttt{not\_ready}) = \{\texttt{ready}\}$,
  $D_{\texttt{temp}}(\texttt{ready}) = [1, 2]$, $D_{\texttt{temp}}(\texttt{not\_ready}) = [2, 3]$;

- $V_{\texttt{proc}} = \{\texttt{reading}_1, \texttt{reading}_2, \texttt{read}_0, \texttt{read}_1, \texttt{read}_2\}$,
  $T_{\texttt{proc}}(\texttt{reading}_1) = \{\texttt{read}_0, \texttt{read}_1\}$, $T_{\texttt{proc}}(\texttt{reading}_2) = \{\texttt{read}_1, \texttt{read}_2\}$, $T_{\texttt{proc}}(\texttt{read}_0) = \{\texttt{reading}_1\}$, $T_{\texttt{proc}}(\texttt{read}_1) = \{\texttt{reading}_2\}$, $T_{\texttt{proc}}(\texttt{read}_2) = \{\texttt{read}_2\}$,
  $D_{\texttt{proc}}(\texttt{reading}_1) = D_{\texttt{proc}}(\texttt{reading}_2) = [1, 2]$, $D_{\texttt{proc}}(\texttt{read}_0) = D_{\texttt{proc}}(\texttt{read}_1) = D_{\texttt{proc}}(\texttt{read}_2) = [2, 3]$;

- $V_{\texttt{transm}} = \{\texttt{send}\}$,
  $T_{\texttt{transm}}(\texttt{send}) = \{\texttt{send}\}$,
  $D_{\texttt{transm}}(\texttt{send}) = [2, 5]$.

The temperature sensor alternates between the states `ready` and `not_ready`. In the former, it senses the temperature of the environment and *possibly* sends the temperature value to the processing unit. The purpose of the processing unit is receiving *two* temperature samples from the sensor, and sending the average value to the data transmission unit. While in state $\texttt{read}_i$, for $i = 0, 1, 2$, it has read i samples. While in $\texttt{reading}_j$, for $j = 1, 2$, it is attempting to read the j-th sample. A reading is possible only if the sensor and the processing unit are synchronized: a time interval (token) with value $\texttt{reading}_j$ has to *contain* a token `ready`. Analogously, the processing unit can send data to the transmitter only if a token with value `send` contains one with value $\texttt{read}_2$.

The sensor starts in state `not_ready`. This is specified by the trigger-less rule $\top \to \exists o[x_{\texttt{temp}} = \texttt{not\_ready}].o \leq^{\mathsf{s}}_{[0,0]} 0$. The processing unit starts in state $\texttt{reading}_1$: $\top \to \exists o[x_{\texttt{proc}} = \texttt{reading}_1].o \leq^{\mathsf{s}}_{[0,0]} 0$. (Recall that trigger-less rules may also contain singular intervals at no extra computational cost.) The goal

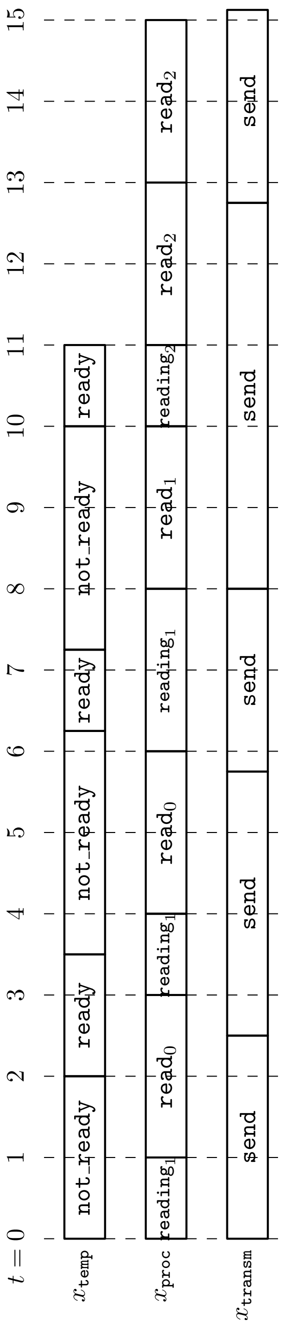

Figure 5: Example of computation for the defined system

of the system is encoded by the rule $\top \to \exists o_1[x_{\texttt{proc}} = \texttt{read}_2]\exists o_2[x_{\texttt{transm}} = \texttt{send}].(o_2 \leq^{\mathsf{s},\mathsf{s}}_{[0,+\infty[} o_1 \wedge o_1 \leq^{\mathsf{e},\mathsf{e}}_{[0,+\infty[} o_2)$.

Let us now encode the fact that the sensor and the processing unit must be synchronized for the latter to receive a temperature sample. We assume the next simple trigger rule to be *interpreted under the future semantics.*

$$o[x_{\texttt{proc}} = \texttt{reading}_1] \to (\exists o_1[x_{\texttt{proc}} = \texttt{read}_0].o \leq^{\mathsf{e},\mathsf{s}}_{[0,1]} o_1)\vee$$
$$(\exists o_2[x_{\texttt{proc}} = \texttt{read}_1]\exists o_3[x_{\texttt{temp}} = \texttt{ready}].o \leq^{\mathsf{e},\mathsf{s}}_{[0,1]} o_2 \wedge o_3 \leq^{\mathsf{e},\mathsf{e}}_{[0,+\infty[} o). \quad (1)$$

Let us observe that, due to the future semantics, the token (referenced by the name) o starts no later than o_3. An analogous rule can be written for the second temperature sample (where $x_{\texttt{proc}} = \texttt{reading}_2$).

In Figure 5 we show an example of plan/computation for the system described by $P = (\{x_{\texttt{temp}}, x_{\texttt{proc}}, x_{\texttt{transm}}\}, R)$.

Let us now specify some properties in MITL (more precisely, $\mathsf{MITL}_{(0,\infty)}$) to check on the system model. The idea is that such properties must hold true over *all possible computations (plans)* of the described system, in order for the MC problem to be satisfied.

- $\mathsf{G}_{<2}\, \neg\psi(\mathsf{s}, \texttt{ready})$. This property holds true in any system computation, as the sensor does not ever get ready by 2 seconds;

- $\mathsf{F}_{\leq 8}\, \psi(\mathsf{s}, \texttt{read}_1)$. This property is not true in all computations (but it is, e.g., in the one of Figure 5), because the sensor and the processing unit may synchronize for the first time after 8 seconds;

- $\mathsf{F}_{\geq 0}\big(\psi(\mathsf{s}, \texttt{ready}) \wedge (\top\, \mathsf{U}_{>0}\, \psi(\mathsf{s}, \texttt{ready}))\big)$. This property holds true in any system computation, since the system guarantees, after some time, to eventually send the data via the transmitter. In order for this to happen, the sensor must become ready (at least) twice.

- $\mathsf{G}_{\geq 0}\big(\psi(\mathsf{s}, \texttt{read}_1) \to \mathsf{F}_{\leq 3}\, \psi(\mathsf{s}, \texttt{read}_2)\big)$. This property is not true in all computations as the processing unit, after reading the first sample, may not be able to read the second one by 3 time units (e.g., when the transmitter and the processing unit do not synchronize as soon as possible).

- $\mathsf{G}_{\geq 0}\big(\psi(\mathsf{s}, \texttt{reading}_1) \wedge (\mathit{EqTime}(\psi(\mathsf{s}, \texttt{ready})) \vee \mathit{past}^{\mathsf{s}}_{\texttt{ready}}) \to \mathsf{F}_{\leq 2}\, \psi(\mathsf{s}, \texttt{read}_1)\big)$. We recall that the proposition letter $\mathit{past}^{\mathsf{s}}_{\texttt{ready}}$ is true at the time it is interpreted if there is a past token for $x_{\texttt{temp}}$ with value $\texttt{ready}$ starting at the same timestamp. The formula considers a situation where a token with value $\texttt{reading}_1$ starts together with a token $\texttt{ready}$. The property expressed by the formula is not true in general, as either (i) the token $\texttt{reading}_1$ may not contain the token $\texttt{ready}$, hence $x_{\texttt{proc}}$ will not move to the state $\texttt{read}_1$ by 2 time units, or (ii) the token $\texttt{reading}_1$ is followed by a token $\texttt{read}_0$. As for the latter case, the system description rule (1) states that if there is a transition from state $\texttt{reading}_1$ to $\texttt{read}_1$, then the processing unit and the sensors must have synchronized. However, the converse implication need not hold: the two component may fail to communicate anyway (the processing unit remaining in $\texttt{read}_0$).

Let us now formally define the MC problem for MITL formulas over timelines. As shown in the proof of Theorem 14, given a system model $P_{\text{sys}} = (SV, R)$, it is possible to build a TA $\mathcal{A}_{\text{sys}}$ that accepts all and only the encodings w_Π of multi-timelines Π of SV satisfying all the rules in R.

Definition 22 (Model checking)**.** Given a system model $P_{\text{sys}} = (SV, R)$ and a MITL formula φ over $\mathcal{AP}$, the MC problem for MITL formulas over timelines is to decide whether or not $\mathcal{L}_T(\mathcal{A}_{\text{sys}}) \subseteq \mathcal{L}_T(\varphi)$.

We recall that [2] given a MITL (resp., $\mathsf{MITL}_{(0,\infty)}$) formula ψ, where N is the number of distinct subformulas of ψ, and K the largest integer constant appearing in ψ, we can build a TA $\mathcal{A}_\psi$ accepting the models of ψ, with $O(2^{N \cdot K})$ (resp., $O(2^N)$) states, $O(N \cdot K)$ (resp., $O(N)$) clocks, and maximum constant $O(K)$. Deciding its emptiness requires space *logarithmic* in the number of states of $\mathcal{A}_\psi$ and *polynomial* in the number of clocks and in the length of the encoding of K, hence exponential (resp., polynomial) space.

In order to decide if $\mathcal{L}_T(\mathcal{A}_{\text{sys}}) \subseteq \mathcal{L}_T(\varphi)$, we check whether $\mathcal{L}_T(\mathcal{A}_{\text{sys}}) \cap \mathcal{L}_T(\mathcal{A}_{\neg\varphi}) = \emptyset$ by making the intersection $\mathcal{A}_\wedge$ of $\mathcal{A}_{\text{sys}}$ and $\mathcal{A}_{\neg\varphi}$, and checking for emptiness of its timed language. The size of $\mathcal{A}_\wedge$ is polynomial in those of $\mathcal{A}_{\text{sys}}$ and $\mathcal{A}_{\neg\varphi}$. Moreover $\mathcal{A}_{\text{sys}}$, $\mathcal{A}_{\neg\varphi}$ and $\mathcal{A}_\wedge$ can be built on the fly, and the emptiness test can be done without explicitly constructing them as well. The next result follows by these observations and by Theorem 14.

Theorem 23. *The MC problem for MITL formulas over timelines, with simple future trigger rules and non-singular intervals, is in* ***EXPSPACE****.*

The MC problem for $\mathsf{MITL}_{(0,\infty)}$ formulas over timelines, with simple future trigger rules and intervals in $Intv_{(0,\infty)}$, is in ***PSPACE****.*

Clearly, **EXPSPACE**- and **PSPACE**-completeness of the above MC problems follow by the underlying future TP problems. This concludes the section.

7. Conclusions and future work

In this paper we have considered the timeline-based planning problem (TP) over dense temporal domains. Timelines have been fruitfully used in temporal planning for quite a long time to describe planning domains. Having recourse to dense time is important for expressiveness: in this way one can avoid unnecessary (or even "forced") details and properly express properties such as accomplishments, actions with duration and temporally extended goals. Since TP turns out to be undecidable in its general form, we have identified and studied "intermediate" decidable cases of the problem, which enforce forms of synchronization rules having lower expressive power than that of general ones. By restricting also the type of intervals used in trigger rules, better complexity results (**EXPSPACE** or **PSPACE**) can be obtained. Finally, if we only allow trigger-less rules, TP becomes **NP**-complete.

At the end of the paper, we have shown how timelines can be employed as system descriptions/models, which are checked against properties specified in the logic MITL. TP is a sort of "necessary condition" for timeline-based MC: TP boils down to a feasibility check of the system description; moreover MC can easily be solved once TP has, as both timelines and the property specification language MITL can be translated into timed automata [1], which have been studied for a long time and are at the basis of well-known model checkers (e.g., Uppaal [20]).

As for future work, future TP with arbitrary trigger rules shall be investigated, whose decidability remains an open problem. Moreover, we would like to study timeline-based MC where property specifications are given by formulas of *Halpern and Shoham's modal logic of time intervals* (HS). A *timed* extension of HS *over dense domains* would be required by timelines; however, in the literature, only *metric* extensions of HS have been proposed over the natural numbers [8]. Defining such an extension, and linking it with known results regarding other timed logics and/or timed automata, is an interesting research theme.

Acknowledgements. We would like to acknowledge: the fundamental contribution by G. Woeginger to the **NP**-completeness result; the support from the GNCS project *Formal methods for verification and synthesis of discrete and hybrid systems*; the support from the project *(PRID) ENCASE—Efforts in the uNderstanding of Complex interActing SystEms*.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

[2] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996. doi:10.1145/227595.227602.

[3] R. Alur and T. A. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Information and Computation*, 104(1):35–77, 1993. doi:10.1006/inco.1993.1025.

[4] J. Barreiro, M. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, J. Ong, E. Remolina, T. Smith, and D. Smith. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proceedings of ICKEPS*, 2012.

[5] L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. Complexity of timeline-based planning over dense temporal domains: exploring the middle ground. In *Proceedings of GandALF*, pages 191–205. EPTCS, 2018. doi:10.4204/EPTCS.277.14.

[6] L. Bozzelli, A. Molinari, A. Montanari, and A. Peron. Decidability and Complexity of Timeline-based Planning over Dense Temporal Domains. In *Proceedings of KR*. AAAI Press, 2018. Full version at `https://www.uniud.it/it/ateneo-uniud/ateneo-uniud-organizzazione/dipartimenti/dmif/assets/preprints/1-2018-molinari`.

[7] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and G. Woeginger. Timeline-based planning over dense temporal domains with trigger-less rules is NP-complete. In *Proceedings of ICTCS*. CEUR Workshop Proceedings, 2018.

[8] D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Metric propositional neighborhood logics on natural numbers. *Software and System Modeling*, 12(2):245–264, 2013. `doi:10.1007/s10270-011-0195-y`.

[9] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and N. Policella. An Innovative Product for Space Mission Planning: An A Posteriori Evaluation. In *Proceedings of ICAPS*, pages 57–64. AAAI Press, 2007.

[10] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye. Timeline-based space operations scheduling with external constraints. In *Proceedings of ICAPS*, pages 34–41. AAAI Press, 2010.

[11] M. Cialdea Mayer, A. Orlandini, and A. Umbrico. Planning and Execution with Flexible Timelines: a Formal Account. *Acta Informatica*, 53(6–8):649–680, 2016. `doi:10.1007/s00236-015-0252-z`.

[12] S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *Transactions on Computational Logic*, 10(3):16:1–16:30, 2009. `doi:10.1145/1507244.1507246`.

[13] J. Frank and A. Jónsson. Constraint-based Attribute and Interval Planning. *Constraints*, 8(4):339–364, 2003. `doi:10.1023/A:1025842019552`.

[14] N. Gigante, A. Montanari, M. Cialdea Mayer, and A. Orlandini. Timelines are Expressive Enough to Capture Action-based Temporal Planning. In *Proceedings of TIME*, pages 100–109. IEEE Computer Society, 2016. `doi:10.1109/TIME.2016.18`.

[15] N. Gigante, A. Montanari, M. Cialdea Mayer, and A. Orlandini. Complexity of timeline-based planning. In *Proceedings of ICAPS*, pages 116–124. AAAI Press, 2017.

[16] D. Harel. *Algorithmics: The spirit of computing*. Springer, 3rd edition, 2012.

[17] A. K. Jónsson, P. H. Morris, N. Muscettola, K. Rajan, and B. D. Smith. Planning in Interplanetary Space: Theory and Practice. In *Proceedings of ICAPS*, pages 177–186. AAAI Press, 2000.

[18] D. Jungnickel. *Graphs, Networks and Algorithms.* Algorithms and Computation in Mathematics. Springer, 2013. doi:10.1007/978-3-642-32278-5.

[19] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. doi:10.1007/BF01995674.

[20] G. K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1:134–152, 1997. doi:10.1007/s100090050010.

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

[22] N. Muscettola. HSTS: Integrating Planning and Scheduling. In *Intelligent Scheduling*, pages 169–212. Morgan Kaufmann, 1994.

[23] J. Ouaknine and J. Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007. doi:10.2168/LMCS-3(1:8)2007.

[24] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

Appendix A. Definition of the value transition function T in the proof of Theorem 7

The value transition function T of x_M is defined as follows.

- For each instruction label $\ell \in \mathsf{Inc} \cup \{\ell_{halt}\}$, let $P_\ell = \emptyset$ if $\ell = \ell_{halt}$, and $P_\ell = \{(succ(\ell), \mathsf{inc}_h)\}$ otherwise, where $c_h = c(\ell)$. Then, $T(\ell)$, $T((\ell, c_i))$, and $T((\ell, (c_i, \#))$, for $i = 1, 2$, are defined as follows:

$$T(\ell) = \{(\ell, c_1), (\ell, c_2)\} \cup P_\ell$$

$$T((\ell, c_1)) = \{(\ell, c_1), (\ell, c_2)\} \cup P_\ell$$

$$T((\ell, c_2)) = \{(\ell, c_2)\} \cup P_\ell$$

- For each instruction label $\ell \in \mathsf{Dec}$ and for each $\ell' \in \{\mathsf{zero}(\ell), \mathsf{dec}(\ell)\}$, $T((\ell, \ell'))$, $T((\ell, \ell', c_i))$, and $T((\ell, \ell', (c_i, \#))$, for $i = 1, 2$, are defined as:

$$T((\ell, \ell')) = \begin{cases} \{(\ell, \ell', c_2), (\ell', \mathsf{zero}_1)\} & \text{if } c(\ell) = c_1, \ell' = \mathsf{zero}(\ell) \\ \{(\ell, \ell', c_1), (\ell', \mathsf{zero}_2)\} & \text{if } c(\ell) = c_2, \ell' = \mathsf{zero}(\ell) \\ \{(\ell, \ell', (c_1, \#))\} & \text{if } c(\ell) = c_1, \ell' = \mathsf{dec}(\ell) \\ \{(\ell, \ell', c_1), (\ell, \ell', (c_2, \#))\} & \text{otherwise} \end{cases}$$

$$T((\ell, \ell', c_1)) = \begin{cases} \emptyset & \text{if } c(\ell) = c_1, \ell' = \mathsf{zero}(\ell) \\ \{(\ell, \ell', c_1), (\ell', \mathsf{zero}_2)\} & \text{if } c(\ell) = c_2, \ell' = \mathsf{zero}(\ell) \\ \{(\ell, \ell', c_1), (\ell, \ell', c_2), (\ell', \mathsf{dec}_1)\} & \text{if } c(\ell) = c_1, \ell' = \mathsf{dec}(\ell) \\ \{(\ell, \ell', c_1), (\ell, \ell', (c_2, \#))\} & \text{otherwise} \end{cases}$$

$$T((\ell, \ell', c_2)) = \begin{cases} \{(\ell, \ell', c_2), (\ell', \mathsf{zero}_1)\} & \text{if } c(\ell) = c_1, \ell' = \mathsf{zero}(\ell) \\ \emptyset & \text{if } c(\ell) = c_2, \ell' = \mathsf{zero}(\ell) \\ \{(\ell, \ell', c_2), (\ell', \mathsf{dec}_1)\} & \text{if } c(\ell) = c_1, \ell' = \mathsf{dec}(\ell) \\ \{(\ell, \ell', c_2), (\ell', \mathsf{dec}_2)\} & \text{otherwise} \end{cases}$$

$$T((\ell, \ell', (c_1, \#))) = \begin{cases} \{(\ell, \ell', c_1), (\ell, \ell', c_2), (\ell', \mathsf{dec}_1)\} & \text{if } c(\ell) = c_1, \ell' = \mathsf{dec}(\ell) \\ \emptyset & \text{otherwise} \end{cases}$$

$$T((\ell, \ell', (c_2, \#))) = \begin{cases} \{(\ell, \ell', c_2), (\ell', \mathsf{dec}_2)\} & \text{if } c(\ell) = c_2, \ell' = \mathsf{dec}(\ell) \\ \emptyset & \text{otherwise} \end{cases}$$

- For each label $\ell \in \mathsf{InstLab}$ and operation $op \in \{\mathsf{inc}_1, \mathsf{inc}_2, \mathsf{zero}_1, \mathsf{zero}_2, \mathsf{dec}_1, \mathsf{dec}_2\}$, $T((\ell, op))$, $T((\ell, op, c_i))$, and $T((\ell, op, (c_i, \#))$, for $i = 1, 2$, are defined as follows, where $S_\ell = \{(\ell, \mathsf{zero}(\ell)), (\ell, \mathsf{dec}(\ell))\}$ if $\ell \in \mathsf{Dec}$, and $S_\ell = \{\ell\}$ otherwise:

$$T((\ell, op)) = \begin{cases} \{(\ell, op, c_2)\} \cup S_\ell & \text{if } op = \mathsf{zero}_1, \ell \neq \ell_{init} \\ \{(\ell, op, c_1)\} \cup S_\ell & \text{if } op = \mathsf{zero}_2, \ell \neq \ell_{init} \\ \{(\ell, op, c_1), (\ell, op, c_2)\} \cup S_\ell & \text{if } op \in \{\mathsf{dec}_1, \mathsf{dec}_2\}, \ell \neq \ell_{init} \\ \{(\ell, op, (c_1, \#))\} & \text{if } op = \mathsf{inc}_1, \ell \neq \ell_{init} \\ \{(\ell, op, c_1), (\ell, op, (c_2, \#))\} & \text{if } op = \mathsf{inc}_2, \ell \neq \ell_{init} \\ \{\ell_{init}\} & \text{if } op = \mathsf{zero}_1, \ell = \ell_{init} \\ \emptyset & \text{otherwise} \end{cases}$$

$$T((\ell, op, c_1)) = \begin{cases} \emptyset & \text{if } op = \mathsf{zero}_1 \text{ or } \ell = \ell_{init} \\ \{(\ell, op, c_1)\} \cup S_\ell & \text{if } op = \mathsf{zero}_2, \ell \neq \ell_{init} \\ \{(\ell, op, c_1), (\ell, op, c_2)\} \cup S_\ell & \text{if } op \in \{\mathsf{dec}_1, \mathsf{dec}_2, \mathsf{inc}_1\}, \\ & \quad \ell \neq \ell_{init} \\ \{(\ell, op, c_1), (\ell, op, (c_2, \#))\} & \text{if } op = \mathsf{inc}_2, \ell \neq \ell_{init} \end{cases}$$

$$T((\ell, op, c_2)) = \begin{cases} \emptyset & \text{if } op = \mathsf{zero}_2 \text{ or } \ell = \ell_{init} \\ \{(\ell, op, c_2)\} \cup S_\ell & \text{otherwise} \end{cases}$$

$$T((\ell, op, (c_1, \#))) = \begin{cases} \emptyset & \text{if } op \neq \mathsf{inc}_1 \text{ or } \ell = \ell_{init} \\ \{(\ell, op, c_1), (\ell, op, c_2)\} \cup S_\ell & \text{otherwise} \end{cases}$$

$$T((\ell, op, (c_2, \#))) = \begin{cases} \emptyset & \text{if } op \neq \mathsf{inc}_2 \text{ or } \ell = \ell_{init} \\ \{(\ell, op, c_2)\} \cup S_\ell & \text{otherwise} \end{cases}$$

This concludes the definition of T of x_M.

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

In this section, we establish the following result.

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

Theorem 9 is proved by a polynomial-time reduction from the halting problem for *gainy counter machines* [12], a variant of standard Minsky machines, whose counters may erroneously increase. Such a 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:

- for each transition $\delta \in \Delta$, $from(\delta) \neq q_{halt}$ and $to(\delta) \neq q_{init}$, and
- 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 $\nu : \{1, \ldots, n\} \to \mathbb{N}$. 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 $v(c) > 0$);
- $\nu'(c) = \nu(c) = 0$ 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 [12].

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

Proposition 24. *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 of an M-computation, 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

$$V_{sec} = V_{main} \times \{1, \dots, n\} \times 2^{\{\#_{\mathsf{inc}}, \#_{\mathsf{dec}}\}},$$

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 $c \in \{1, \dots, n\}$ 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:

- $\#_{\mathsf{inc}} \in Tag(c, v_{main})$ iff $op = (\mathsf{inc}, c)$;
- $\#_{\mathsf{dec}} \in Tag(c, v_{main})$ 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 $h_0, h_1, \dots, h_n \geq 0$. 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, the latter being present only for technical aspects, but not 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 $c \in \{1, \dots, n\}$, 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 $c \in \{1, \dots, n\}$. 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 of the computation 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 $\mathit{to}(\delta_i) = \mathit{from}(\delta_{i+1})$ and $op(\delta_i) = op_{i+1}$. Note that by our assumptions $\mathit{to}(\delta_i) \neq q_{halt}$ 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 $\mathit{to}(\delta_k) = q_{halt}$. 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 $\mathit{Intv}_{(0,\infty)}$ 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 requirement.

Claim 25. 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 $v \in V$.

T can be built by adapting the construction of Appendix A.

Let $V_{halt} = \{(\delta, op) \in V_{main} \mid \mathit{to}(\delta) = q_{halt}\}$. By Claim 25 and the assumption that $\mathit{from}(\delta) \neq q_{halt}$ 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

$$\top \rightarrow \bigvee_{v \in V_{halt}} \exists o[x_M = v]. \top .$$

The crucial well-formedness requirement is captured by the trigger rules in R_M which express the following punctual time constraints. Note that we take advantage of the dense temporal domain to allow for the encoding of arbitrarily large values of counters in two time units.

- *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 25, 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:

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

- 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 25, 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 $Intv_{(0,\infty)}$:

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

- For capturing the increment and decrement requirements, by construction, it suffices to enforce that:

  1. 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 $\mathsf{s}(tk') - \mathsf{s}(tk) = 2$;
  2. 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

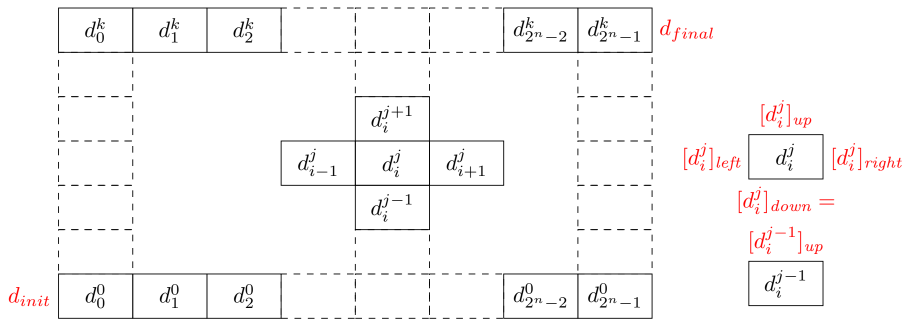

Figure C.6: A (generic) instance of the domino-tiling problem, where d^i_j denotes $f(i, j)$.

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 24 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 24. □

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

We prove that the future TP problem with simple trigger rules and non-singular intervals is **EXPSPACE**-hard. Hardness holds also when only a single state variable is involved. The claim is proved by a polynomial-time reduction from the *domino-tiling problem for grids with rows of single exponential length* [16]. We start by introducing such problem.

An instance $\mathcal{I}$ of a domino-tiling problem for grids with rows of single exponential length 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 labelling of the edges of each cell (see Figure 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*);

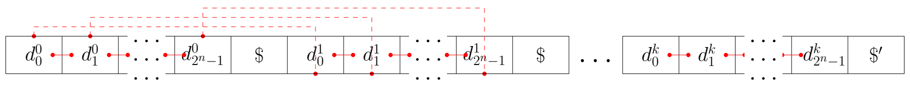

Figure C.7: A timeline encoding the ordered concatenation of the rows of a tiling. Red lines represent the horizontal and vertical constraints among domino-types.

- 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 [16].

We can now prove the following.

Theorem ([15]). *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 sake of 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 Figure 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 next simple trigger rules, one for each $v \in V$:

$$o[y = v] \to o \leq^{\mathsf{s,e}}_{[0,2]} o.$$

These, paired with the constraint function D, enforce all tokens' durations 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 $2 \cdot 2^n$:

$$\top \to \exists o[y = \$].o \geq^{\mathsf{s}}_{[0,1]} 2 \cdot 2^n, \tag{C.1}$$

and

$$o[y = \$] \to o \geq^{\mathsf{s}}_{[0,+\infty[} 2 \cdot 2^n. \tag{C.2}$$

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 \$$'$.

$$\begin{aligned} & o[y = \$] \to \\ & \qquad (\exists o'[y = \$].o \leq^{\mathsf{e,s}}_{[2\cdot 2^n, 2\cdot 2^n+1]} o') \vee (\exists o''[y = \$'].o \leq^{\mathsf{e,s}}_{[2\cdot 2^n, 2\cdot 2^n+1]} o''). \end{aligned} \tag{C.3}$$

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 $d \in \Delta$,

$$\begin{aligned} & o[y = d] \to \\ & \Big(\bigvee_{d' \in \Delta,\, [d]_{up} = [d']_{down}} \exists o'[y = d'].o \leq^{\mathsf{e,s}}_{[2\cdot 2^n, 2\cdot 2^n+1]} o' \Big) \vee (\exists o''[y = \$'].o \leq^{\mathsf{e,s}}_{[0, 2\cdot 2^n-2]} o''). \end{aligned} \tag{C.4}$$

It is straightforward to check that rules (C.1), (C.2), (C.3), and (C.4), along with the horizontal constraint guaranteed by the function T, enforce the following property.

Proposition 26. *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 $k' \cdot 2(2^n+1)$. Finally, 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.*

Finally, we settle the initialization and acceptance requirements by means of the following pair of trigger-less rules:

$$\top \to \exists o[y = d_{init}].o \geq^{\mathsf{s}}_{[0,1]} 0,$$

$$\top \to \exists o[y = d_{final}] \exists o'[y = \$'].o \leq^{\mathsf{e,s}}_{[0,1]} o'.$$

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 $y = (V, T, D)$ as well as all synchronization rules can be generated in polynomial time in the size of the instance $\mathcal{I}$ of the domino-tiling problem (in particular, note that all interval bounds and time constants of time-point atoms have a value, encoded in binary, which is in $O(2^n)$). $\square$